# DOWNSTREAM RESOURCE ALLOCATION IN DOCSIS 3.0 CHANNEL BONDED NETWORKS

---

A Dissertation
Presented to
the Graduate School of
Clemson University

---

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Computer Science

---

by
Scott A. Moser
August 2011

---

Accepted by:
James J. Martin, Committee Chair
James M. Westall
Harold C. Grossman
Brian C. Dean

# ABSTRACT

Modern broadband internet access cable systems follow the Data Over Cable System Interface Specification (DOCSIS) for data transfer between the individual cable modem (CM) and the Internet. The newest version of DOCSIS, version 3.0, provides an abstraction referred to as bonding groups to help manage bandwidth and to increase bandwidth to each user beyond that available within a single 6MHz. television channel. Channel bonding allows more than one channel to be used by a CM to provide a virtual channel of much greater bandwidth. This combining of channels into bonding groups, especially when channels overlap between more than one bonding group, complicates the resource allocation problem within these networks.

The goal of resource allocation in this research is twofold, to provide for fairness among users while at the same time making maximum possible utilization of the available system bandwidth. The problem of resource allocation in computer networks has been widely studied by the academic community. Past work has studied resource allocation in many network types, however application in a DOCSIS channel bonded network has not been explored.

This research begins by first developing a definition of fairness in a channel bonded system. After providing a theoretical definition of fairness we implement simulations of different scheduling disciplines and evaluate their performance against this theoretical ideal. The complexity caused by overlapped channels requires even the simplest scheduling algorithms to be modified to work correctly.

We then develop an algorithm to maximize the use of the available system bandwidth. The approach involves using competitive analysis techniques and an online algorithm to dynamically reassign flows among the available channels. Bandwidth usage and demand requests are monitored for bandwidth that is underutilized, and demand that is unsatisfied, and real time changes are made to the flow-to-channel mappings to improve the utilization of the total available bandwidth.

The contribution of this research is to provide a working definition of fairness in a channel bonded environment, the implementation of several scheduling disciplines and evaluation of their adherence to that definition, and development of an algorithm to improve overall bandwidth utilization of the system.

# ACKNOWLEDGMENTS

Foremost I want to thank my advisor Jim Martin. I was at Clemson less than two months before two different people, both of whom are on the committee, told me "you need to talk to Jim Martin". Due to my part time status we have had an eight year association during which time I have learned a great deal from him about the entire research process.

Additionally I want to thank each of the other members of the committee. Mike Westall, from whom I learned to always use the simplest possible examples to prove a concept. Harold Grossman, who taught the first course I took here. I believe that compiler course to be the best possible course to evaluate someone's overall understanding of the field of computer science, (it was also the most time consuming one in my life). Lastly Brian Dean, the latest member of the committee, who exposed me to the world of algorithms and the joy of not having to operate in the "real world". These four committee members were all great mentors during this process and have all become friends as well.

I would also like to thank Cisco Systems, Inc. for the Research Award *DOCSIS 3.0 Scheduling Algorithms*, which provided the topic for this dissertation and much valuable input on current DOCSIS technology.

Lastly I would like to thank the School of Computing, Computer Science Division (formerly the Computer Science Department), for the opportunity to undertake this program on a part time basis.

TABLE OF CONTENTS

Page

Table of Contents (Continued)

Page

LIST OF TABLES

List of Tables (Continued)

Table Page

LIST OF FIGURES

List of Figures (Continued)

Figure Page

# CHAPTER ONE

# INTRODUCTION

One of the dominant broadband access methods to the home and small business user is the cable network. The Data Over Cable System Interface Specification (DOCSIS) standard [1] defines the operation of these networks. This system uses the standard television channel to transfer data within the cable network. A single television channel is allocated a 6 MHz bandwidth within which to operate. In early DOCSIS systems one channel was used for downstream transmission and a second channel was used for upstream transmission. In the early DOCSIS networks increases in data rates were primarily achieved through improvements in modulation techniques within that 6 MHz bandwidth restriction. With the release of DOCSIS 3.0 data rates can now be increased by using channel bonding, which allows for the use of more than one channel for both upstream and downstream traffic.

The term *bandwidth* is used in different contexts. Comer [9] provides two different definitions for bandwidth. The first is *analog bandwidth*, "the difference between the highest and lowest frequencies of the constituent parts (i.e. the highest and lowest frequencies obtained by Fourier analysis)." This is the context in which we have used the term in the previous paragraph to describe the 6 MHz bandwidth of the television channel. The second of Comer's definitions is *network bandwidth*, which is the more commonly used form in computer networking, used to represent the data rate through a channel, frequently described as channel capacity or throughput. With the

exception of the previous paragraph, in this work when we use the term *bandwidth* we are referring to network bandwidth.

The primary goals of resource allocation are: 1.) to maximize utilization of the available network bandwidth, 2.) to provide fairness of bandwidth allocation between the users of the network and 3.) to provide predictable levels of service that meet negotiated service qualities. Resource allocation within networks is a widely studied area, but with the addition of channel bonding into the DOCSIS standard new challenges are introduced. Because not all users have access to the same resources in a channel bonded system these goals can sometimes be in conflict.

### 1.1 The DOCSIS Resource Allocation Problem

Two broad dimensions to the problem of network resource management are:

- Service models: At the highest level, a network can provide any combination of guaranteed services, differentiated services, or a simple best effort service. A guaranteed service provides services that meet specific performance criteria. A differentiated service typically allows traffic to be divided into classes and have the network treat each class differently when subject to various congestion situations. Best effort treats all data the same. Each model requires different resource allocation strategies.
- Scope or location of the management problem: Resource allocation mechanisms might operate at a local router level or at a global, network-wide level. The control mechanisms generally differ depending on the scope

however they must operate in unison to achieve overall allocation goals and objectives.

In addition to the dimensions described above, a further way to describe the range of mechanisms that are available for managing bandwidth is with respect to the time scale of control. This range of mechanisms includes the following:

- Microseconds: Packet scheduling disciplines determine which packets get serviced when a link becomes congested and also how the queue is managed.
- Milliseconds: End-to-end congestion control algorithms such as the control algorithms supported by TCP stacks manage how a flow reacts to signs of network congestion.
- Minutes or hours: Traffic management methods such as Comcast's Fairshare management that modifies the allocation of resources based on control procedures that are based on relatively large time scales.
- Days or weeks: Admission control and capacity planning methods are used to ensure that the network is adequately provisioned to meet throughput and delay requirements.

The research described in this dissertation deals exclusively with downstream, best effort traffic. All packets are considered equal and no flow is given preference over another. Further we assume all flows have full access to all available bandwidth (i.e., rate control is not considered). Our work considers resource allocation at the central

resource controller, in DOCSIS this is the Cable Modem Termination System (CMTS). Our work deals with two different time scales. The first is the microsecond level and the packet scheduling disciplines that are applied. The second is the minutes-hours time scale and deals with the remapping of traffic across channels to improve bandwidth utilization as demands change.

The DOCSIS standards do not specify resource allocation policies, but they do define mechanisms that permit cable system operators to define and implement policies as they see fit. Study of resource allocation in versions of DOCSIS prior to version 3.0 was primarily limited to upstream data scheduling. The downstream problem was considered trivial since it was a centrally controlled activity and only one channel needed to be scheduled. Significant work was done on upstream scheduling to achieve fairness and/or to provide different levels of service quality. These efforts are detailed in the Background section in Chapter 2 of this document.

### 1.2 DOCSIS 3.0 Channel Bonding

The early versions of DOCSIS allow a cable modem (CM) to receive on a single channel and transmit on another single channel. This imposes a limit to the bandwidth available to an individual CM user due to the 6MHz. bandwidth of a television channel. The newest version of this standard, DOCSIS 3.0, adds channel bonding capability. DOCSIS defines a “service flow” as an upstream or downstream flow of packets that is identified by a service flow identifier (SFID). The DOCSIS 3.0 standard [1] provides the following definition; “A set of two or more channels over which the CMTS schedules the

information of a service flow is called a ‘bonding group’ ”. A single CM can now access multiple channels for transmission and reception of data. Some channels are treated individually while other channels are assigned to bonding groups and treated as a single logical channel. Both upstream and downstream channels available to a CM can be organized as: 1) a single channel; 2) divided into bonding groups; or 3) a mix of bonding groups and individual channels. DOCSIS 3.0 allows a CM to receive on a single, downstream channel or bonding group and transmit on another single upstream channel or bonding group. Each individual service flow is assigned to a bonding group.

The CMTS can use any of the channels in a bonding group but is not required to use all channels in the bonding group. Channels can be assigned to multiple, overlapping bonding groups and these bonding groups can be dynamically reassigned. This allows the CMTS the flexibility needed to optimize the use of the available bandwidth and to balance the loading on the channels when making scheduling assignments. With the ability to have multiple paths for transmission to the same destination comes the requirement for sequence numbers since now multiple packets can be sent simultaneously. In addition, due to differences in delay parameters on different channels packets can arrive out of order. The sequence numbers allow for reassembly of the packets in proper order at the CM.

During initialization the CM notifies the CMTS of its channel capabilities. The specification details both Receive Channels and Receive Modules. A Receive Channel refers to a single downstream channel on a single center frequency. Each CM implements a fixed number of Receive Channels. A Receive Module refers to a physical

layer implementation shared by multiple Receive Channels. The Receive Modules represent a group of channels that must be maintained together. For example this could be a shared tuner with a certain range, or a shared signal processing module. Reconfiguring any of the channels in a module can cause a disruption in all channels in the module. To simplify the job of the CMTS there are standard module profiles. For example there is a module that defines four physical channels that can be set up in any set of ten adjacent channel frequencies.

The CMTS decides how to allocate grants on the upstream channels in the bonding group. Requests can be made by the CM on any channel in the bonding group. The CMTS then allocates bandwidth on any subset of channels in the bonding group with the available space. This allows the CMTS to perform real time load balancing as the grants are made. The CMTS can also consider the physical layer parameters of each channel in providing optimal allocations across the channels. As with previous DOCSIS versions the scheduling algorithms are not defined, leaving those decisions to the vendors.

While most past work with DOCSIS scheduling dealt with the upstream data flow, channel bonding now makes the downstream direction less than trivial since we now deal with a situation where not all flows have access to the same set of resources. As video streaming is now the dominant Internet broadband application, managing downstream bandwidth in a multi-channel environment has emerged as a crucial problem in the cable industry.

### 1.3 Problem Formulation and Research Objectives

The above discussion indicates that the overall DOCSIS 3.0 resource allocation problem encompasses a large range of efforts. This research addresses a small subset of this overall problem area. We investigate the problem of managing downstream traffic in a channel bonded network based upon the two goals of resource allocation, fairness and utilization.

The first effort involved the definition of fairness in a channel bonding system and the evaluation of several different scheduling disciplines for their adherence to that definition of fairness.

The second effort of the project develops a method to more efficiently utilize the available system bandwidth. Dynamic analysis of the bandwidth utilization is monitored and algorithms are developed to allow real time analysis of the resource allocation in the network and adjust the channel mapping to improve performance. This is achieved by moving flows between channels to allow more efficient use of bandwidth that is either underutilized or inaccessible given the current bonding group arrangements and the current demand requests of the individual flows.

In summary, the research objectives of our work are to:

- Develop a theoretical fair queuing model that defines fairness in a channel bonded environment and develop an offline algorithm that achieves optimal fairness as defined by the theoretical model.

- Implement computationally feasible algorithms to approximate fair queuing, at the packet level, in a DOCSIS 3.0 channel bonded network.
- Develop an online algorithm to dynamically modify the flow-to-channel assignments such that the total system bandwidth can be more efficiently utilized in attempting to satisfy changing demands.

### 1.4 Chapter Outline

The objective of this work is to develop resource allocation strategies for use in a DOCSIS 3.0 channel bonded network. These strategies are then implemented and tested in two ways. Individual components are coded and tested in a standalone manner for basic operational capability. After obtaining a working packet scheduling approach it is added to the *ns* simulator. The *ns* simulator gives us the ability to test the solutions in a complete network environment and allows us to do more extensive analysis.

This dissertation is organized as follows:

- In Chapter 2, we review background material on the scheduling disciplines of interest, the general operation of DOCSIS, and the relevant background literature dealing with scheduling in DOCSIS.
- In Chapter 3, we develop a definition of fairness for the channel bonded environment and implement a simulation to calculate the fair bandwidth allocations.

- In Chapter 4, we discuss the modification of standard scheduling disciplines to operate in the channel bonded environment. We implement these packet scheduling disciplines in the *ns* tool and evaluate their performance against the fairness standard developed in Chapter 3.
- In Chapter 5, we move to the second goal of resource allocation, bandwidth utilization. We use the ideas of competitive analysis to develop first, an offline optimal algorithm and secondly, an online remapping algorithm. We then analyze the results to evaluate the improvement over the system without remapping and the competitiveness of the online algorithm developed.
- In Chapter 6, we summarize results and discuss open issues for further research.

# CHAPTER TWO

# BACKGROUND

The specific areas of background for this effort fall into several categories. The first area details the development of the scheduling disciplines used in this project. This is followed by an overview of DOCSIS operation. We then follow with a survey of the literature covering previous work on DOCSIS scheduling. This work dealt exclusively with upstream scheduling because the downstream problem was trivial. With the introduction of channel bonding the downstream scheduling problem takes on new interest. Additional areas of background are included in the relevant chapters that follow.

## 2.1 Scheduling Disciplines

We now review the development of scheduling disciplines beginning with first come first served queuing and progressing through several improvements to the disciplines used for this project. These scheduling disciplines fall into two broad categories. The first is round robin scheduling where flows are served in sequence. The second is time stamp based scheduling where each packet is given a time stamp and the packets are served in the order of the timestamps.

### FIFO/FCFS Queuing

First in first out queuing, also known as first come first served, is generally considered the standard basic queue behavior and is the most widely used queuing discipline at this time. The arrival order is the same as the service order. This is the

standard method for store and forward traffic handling. Due to this fact most routers that use FIFO queuing have been highly optimized for performance, with designs that make this process as fast as possible. This makes these routers highly efficient.

The advantage of the FIFO queuing discipline is that in environments where there is significant bandwidth and the router is primarily absorbing short term overloads, this is a very efficient approach. However there are significant disadvantages, especially in heavily loaded networks.

The FIFO discipline offers the same level of service to all arriving packets. For this reason, it tends to favor non-rate-adaptive applications such as UDP, over rate-adaptive applications, such as TCP, which decreases transmission rates when it encounters congestion. Applications that make no effort to reduce their transmission rate when congestion occurs will get more bandwidth by default.

Therefore FIFO is inherently unsuited for ensuring that competing flows receive their apportioned shares of network bandwidth. The main purpose of queuing strategies for different service levels is to counter this and intentionally give preferential treatment to the classes of service with higher priority.

FIFO is a very efficient algorithm, scales well and provides very predictable outcomes. The maximum jitter introduced is proportionate to the size of the queue. The biggest negative is that an uncontrolled or bursting source may totally consume the entire queue.

**Priority Queuing**

In priority queuing individual output queues are established for each priority level, or service class. Each of these queues will then be individually processed as FIFO queues. Which of these FIFO queues is used for each packet is based on the packet classification function. The system relies on the processor to identify the service class for each arriving packet and to then place the packet in the queue for that service class. The priority scheduling algorithm then selects a packet from the head of queue *n*, as long as all other higher priority queues are empty.

An advantage to priority queuing is that the problem of non-rate-adaptive traffic benefiting, in the case of FIFO queuing, can be fixed by giving the rate-adaptive applications a higher priority. With a higher priority level for the rate-adaptive traffic the non-rate-adaptive flows will no longer receive a higher level of service by default.

The biggest disadvantage for the priority queuing method is the possibility of buffer starvation for the lower level queues. If there is a heavy load of high priority traffic the lower priority queues will not get any service time. This problem gets exacerbated by the fact that the delays caused by the lower priority queue starvation causes retransmission timers to fire, thereby causing retransmissions and even more low priority traffic to enter the queues. Therefore the low priority throughput efficiency plummets. Even after the high priority traffic clears, it takes a while before the low priority traffic can get back to a normal situation.

A basic approach taken to prevent high priority traffic from using all resources is to combine this queue management approach with admission controls. The admission

strategy is used to limit the high priority traffic entering the network to a level that will not consume all available resources.

With priority queuing the highest classes of service receive a low jitter, low loss service as long as the high priority traffic is less than the available network capacity. Of course the other side of the situation is that all other traffic may be completely stalled.

### Generalized Processor Sharing (GPS)

To achieve fairness at a time interval on the same order as the maximum size packet requires that something other than strict round robin scheduling be used. It requires that higher priority queues be serviced more frequently than lower priority queues.

Generalized processor sharing [36] represents an ideal work-conserving weighted fair share model. It is a weighted fair sharing resource allocation that uses an infinitely small service quanta. This is a theoretical, "ideal" (i.e. non-implementable) system due to the fact that it ignores the reality that data is actually quantized into packets. The value of GPS is that it defines a metric used to measure how close a real implementation comes to this ideal behavior.

During any time interval when there are exactly $N$ non-empty queues, GPS will service the head packets of each of the $N$ queues simultaneously. Each queue will be serviced at a rate of $1/N$ of the available link speed.

**Weighted Fair Queuing (WFQ)**

In a realistic packet system only one queue can be serviced at a time and an entire packet must be serviced before another packet can receive service. Weighted fair queuing is an approximation of GPS behavior at the packet level. WFQ is sometimes referred to as packet-by-packet GPS or simulated bit-by-bit weighted round robin. The relative share of the link granted to each queue is in accordance with the weight for each queue. Each class receives service in proportion to its relative weight. When a given class requires less than its weighted allocation the excess is shared among all other service classes in proportion to the remaining classes relative weights.

WFQ emulates the behavior of GPS by transmitting packets in the same order that the trailing bit of each packet would have been transmitted by a GPS system. When the server is ready to transmit the next packet it selects from all queued packets, the first packet that would complete service in the equivalent GPS system if no additional packets were to arrive after that instant in time.

If a bound on the number of service classes is known, then the minimum amount of resource allocated to each service class is predictable. WFQ offers a mechanism to guarantee the minimum level of resource allocated to each service class.

Parekh [36] establishes important relationships between GPS and its corresponding WFQ packet implementation:

1.) In terms of delay, a packet will finish service in a WFQ system later than in the corresponding GPS system by no more than the transmission time of one maximum size packet.

2.) In terms of total number of bits served for each session, a WFQ system does not fall behind a corresponding GPS system by more than one maximum size packet.

The largest disadvantage of WFQ is its complexity. WFQ is implemented using a GPS simulator. Whenever a packet arrives, and whenever a packet is removed from the queue, iterative scans must be made of all per-flow states to re-compute the GPS simulated behavior.

### Worst-case Fair Weighted Fair Queuing ($WF^2Q$)

Bennett and Zhang [5] show that while WFQ cannot fall behind GPS by more than one maximum size packet, it can be far ahead of GPS in terms of number of bits served for a session. They propose a new algorithm they call Worst-case Fair Weighted Fair queuing, or $WF^2Q$. Their algorithm, when selecting the next packet for transmission, rather than selecting from all the packets at the server, as in WFQ, the server only considers the set of packets that have started (and possibly finished) receiving service in the corresponding GPS system.

They show that service provided by $WF^2Q$ can neither be too far behind, nor too far ahead, when compared to that of GPS. They therefore conclude that $WF^2Q$ provides almost identical service to that of the ideal GPS system.

The GPS derived algorithms have no requirement for a small set of preconfigured service classes because the algorithm will adapt to a dynamically changing set. WFQ based algorithms attempt to fairly share the available resource across all traffic flows, while obeying any relative weighting that may be applied to any individual traffic class.

This prevents an uncontrolled application from bursting traffic into the network to the detriment of all other traffic. Because WFQ based algorithms do provide a fair outcome across all active traffic flows it does offer an effective implementation for providing guaranteed services. However, the computational complexity of these approaches is the major disadvantage and makes them unusable for high speed networks

**Self Clocked Fair Queuing (SCFQ)**

Variations of WFQ have been proposed where tradeoffs are made between complexity and the time it takes to perform the iterative queue scans. These alternatives achieve algorithm improvements but at the expense of accuracy of scheduling. One of these simplifications is self clocked fair queuing. While the WFQ based techniques use a simulated virtual time to calculate the finish times of packets in the queue, SCFQ [17] uses an internal version of virtual time extracted from the packet at the head of the queue.

The finish time of a packet is the transmission time of that packet divided by the flow weight and added to the finish time of the previous packet in the flow. The transmission time of the packet is the length of the packet divided by the transmission rate. Therefore, the service tag of a packet is equal to the total normalized service provided to that flow up to that time. The only problem occurs when a flow becomes idle. While the flow is idle its virtual time remains fixed while all flows remaining active will continue to advance. When the flow restarts it would receive an unfair advantage, getting total use of the output until its virtual time catches up to the other flows. The SCFQ algorithm corrects for this by adding the packet's transmission time to the

maximum of the finish time of the last packet in the flow, or the finish time of the current packet being transmitted. This causes the virtual time of a restarting flow to be advanced to the virtual time of the current packet in the transmission output, thereby removing the gap in time between the previously idle flow and all other flows. The finish time, or service tag, formula is:

$$F_k^i = ( L_k^i / r_k ) + \max ( F_k^{i-1} , v ( a_k^i ) )$$

Where:

$F_k^i$ - is the finish time of the i[th] packet of the k[th] flow

$L_k^i$ - is the length of the packet

$r_k$ - is the normalized transmission rate of the flow

$F_k^{i-1}$ - is the finish time of the previous packet in the k[th] flow

$v ( a_k^i )$ - is the virtual time of the packet currently being transmitted

**Deficit Round Robin (DRR)**

Deficit round robin [39] provides a means to deal with the unfairness in strict round robin scheduling where a queue with larger packets can receive more than its intended weighting would indicate. In DRR a deficit counter is associated with each queue. The deficit counter is initialized to zero. During each round a quantum of bits is added to the deficit counter. The packet at the head of the queue is transmitted if its size is not greater than the deficit count. If there are insufficient bits in the deficit counter to transmit the packet, the queue is skipped and waits for the next round when another

quantum will be added to the deficit counter. If the packet is transmitted, the size of the packet is subtracted from the deficit counter.

During a round if an empty queue is encountered its deficit counter is cleared. This will keep the queue from building up a large credit, which will eventually lead to unfairness when packets do arrive to that queue. To assure that every queue with packets waiting will always transmit at least one packet on each round, the quantum size can be set to be equal to the maximum packet size the network can handle. The primary benefit of DRR is its ease of implementation. While it does approximate GPS, DRR can not ensure fairness for timescales less than one round time.

### 2.2 DOCSIS Operation

The DOCSIS standard [1] provides a MAC layer protocol for use on Hybrid Fiber-Coax (HFC) networks. At the lowest levels the network employs a hierarchical structure with the head-end connecting to a group of Cable Modem Termination System (CMTS) units each of which interfaces with many Cable Modem (CM) units. The system provides an asymmetric data path to the CM users with lower bandwidth in the upstream return path to the CMTS, while more of the cable bandwidth is allocated for downstream transmissions from the CMTS to the CMs.

The CMTS controls the upstream flow of data between itself and the CMs attached to it by sending MAP messages to the CMs to indicate transmission timeslots for each CM waiting to send data. The DOCSIS standard provides for Quality of Service (QoS) mechanisms so that different levels of service can be accommodated. The CMTS

bases its timeslot allocation decisions on the QoS level of each data flow and real time requests from the CMs for transmission bandwidth.

A contention scheme is used for the CMs to request transmission bandwidth. The MAC protocol calls for each MAP allocation to contain timeslots for user data, maintenance data, and contention slots to be used for transmission requests. A CM requests a transmission timeslot by using one of the available contention slots. Since the upstream and downstream channels are on different frequencies the CM can not detect a collision as it transmits and must rely on notification from the CMTS during the next MAP time of the time slot allocations. The CM determines that contention has occurred if the next MAP message does not either assign a transmission slot, or acknowledge that the assignment is pending.

**Basic operation**

Once powered on, the CM establishes a connection to the network and maintains this connection until the power to it is turned off. Registration of the CM onto the network involves acquiring upstream and downstream channels and encryption keys from the CMTS and an IP address from the ISP. The CM also determines propagation time from the CMTS in order to synchronize itself with the CMTS (and in effect the network) and finally logs in and provides its unique identifier over the secure channel. Due to the shared nature of these cable networks, transmissions are encrypted in both the upstream and downstream directions.

DOCSIS specifies an asymmetric data path with downstream and upstream data flows on two separate frequencies. The upstream and downstream carriers provide two shared channels on all pre-version 3.0 CMs. On the downstream link the CMTS is a single data source and all CMs receive every transmission. On the upstream link all CMs may transmit and the CMTS is the single sink.

Packets sent over the downstream channel are broken into 188 byte MPEG frames each with 4 bytes of header and a 184 byte payload. Although capable of receiving all frames, a CM is typically configured to receive only frames addressed to its MAC address or frames addressed to the broadcast address. In addition to downstream user data, the CMTS will periodically send management frames. These frames include operations such as ranging, channel assignment, operational parameter download, CM registration, etc. Additionally, the CMTS periodically sends MAP messages over the downstream channel that identify future upstream TDMA slot assignments. The CMTS makes these upstream CM bandwidth allocations based on CM requests and Quality of Service (QoS) policy requirements.

The upstream channel is divided into a stream of time division multiplexed 'mini-slots' which, depending on system configuration, normally contain from 8 to 32 bytes of data. The CMTS must generate the time reference to identify these mini-slots. Due to variations in propagation delays from the CMTS to the individual CMs, each CM must learn its distance from the CMTS and compensate accordingly such that all CMs will have a system wide time reference to allow them to accurately identify the proper

location of the mini-slots. This is called ranging and is part of the CM initialization process.

Ranging involves a process of multiple handshakes between the CMTS and each CM. The CMTS periodically sends sync messages containing a timestamp. The CMTS also sends periodic bandwidth allocation MAPs. From the bandwidth allocation MAP the CM learns the ranging area from the starting mini-slot number and the ranging area length given in the message. The CM will then send a ranging request to the CMTS. The CMTS, after evaluating timing offsets and other parameters in the ranging request, will return to the CM a ranging response containing adjustment parameters. This process allows each CM to identify accurately the timing locations of each individual mini-slot.

In addition to generating a timing reference so that the CMs can accurately identify the mini-slot locations, the CMTS must also control access to the mini-slots by the CMs to avoid collisions during data packet transmissions. For best effort traffic, CMs must request bandwidth for upstream transmissions. There are several mechanisms available: contention BW requests, piggybacked BW requests and concatenated BW requests.

### Contention Bandwidth Requests

The CMTS must periodically provide transmission opportunities for CMs to send a request for bandwidth to the CMTS. As in slotted Aloha networks [2], random access bandwidth request mechanisms are inefficient as collisions will occur if two (or more) CMs attempt to transmit a request during the same contention mini-slot. Most

implementations will have a minimum number of contention mini-slots to be allocated per MAP time, and in addition, any unallocated mini-slot will be designated as a contention mini-slot.

When a packet arrives at the CM that requires upstream transmission, the CM prepares a contention-based BW request by computing the number of mini-slots that are required to send the packet including all framing overhead. The contention algorithm requires the CM to randomly select a number of contention mini-slots to skip before sending (an initial back-off). This number is drawn from a range between 0 and a value that is provided by the CMTS in each MAP. The values sent are assumed to be a power of 2, so that a 5 would indicate a range of 0 – 31. After transmission, if the CM does not receive an indication that the request was received, the CM must randomly select another number of contention mini-slots to skip before retrying the request. The CM is required to exponentially back-off the range with each collision with the maximum back-off specified by a maximum back-off range parameter contained in each MAP. The CM will drop the packet after it has attempted to send the request 16 times.

As an example of the operation of the truncated exponential back-off algorithm, assume that the CMTS has sent an initial back-off value of 4, indicating a range of 0 – 15, and a maximum back-off value of 10, indicating a range of 0 – 1023. The CM, having data to send and looking for a contention mini-slot to use to request bandwidth, will generate a random number within the initial back-off range. Assume that an 11 is randomly selected. The CM will wait until eleven available contention mini-slots have passed. If the next MAP contains 6 contention mini-slots, the CM will wait. If the

following MAP contains 2 contention mini-slots, a total of 8, the CM will still continue to wait. If the next MAP contains 8 contention mini-slots the CM will wait until 3 contention mini-slots have passed, 11 total, and transmit it's request in the fourth contention mini-slot in that MAP.

The CM then looks for either a Data Grant from the CMTS or a Data Acknowledge. If neither is received, the CM assumes a collision has occurred. The current back-off range is then doubled, i.e. the current value is increased from 4 to 5 making the new back-off range 0 – 31, and the process is repeated. The CM selects a random value within this new range, waits the required number of contention mini-slots, and resends its request. The back-off value continues to be incremented, doubling the range, until it reaches the maximum back-off value, in this example 10, or a range of 0 – 1023. The current back-off range will then remain at this value for any subsequent iterations of the loop. The process is repeated until either the CM receives a Data Grant or Data Acknowledge from the CMTS, or the maximum number of 16 attempts is reached.

**Piggybacked BW requests**

To minimize the frequency of contention-based bandwidth requests, a CM can piggyback a request for bandwidth on an upstream data frame. For certain traffic dynamics, this can completely eliminate the need for contention-based bandwidth requests.

The MAC header has the capability of defining an Extended Header field. Extended Headers can be used to request bandwidth for additional upstream transmissions, during the current data transmission. This allows the request for bandwidth to be made outside of the contention process and thereby reduces the occurrence of collisions and consequently the access delay. This process will allow the transmission of data, without the possibility of collisions, when there are large packet flows to be passed upstream.

### Concatenating Packets

DOCSIS provides both Fragmentation MAC Headers, for splitting large packets into several smaller packets, and Concatenation MAC Headers, to allow multiple smaller packets to be combined and sent in a single MAC burst. Concatenation can also be used to reduce the occurrence of collisions by reducing the number of individual transmission opportunities needed. Concatenation is the only method for transmitting more than one packet in a single transmission opportunity. The CMTS, receiving the Concatenation MAC Header, must then 'unpack' the user data correctly. The Concatenation MAC Header precludes the use of the Extended Header field and therefore piggybacking of future requests can not be done in a concatenated frame.

### QoS

DOCSIS manages bandwidth in terms of Service Flows that are specified with Service Flow IDs (referred to as a SID). Traffic arriving at either the CMTS or the CM

for transmission over the DOCSIS network is mapped to an existing SID and treated based on the profile. A CM will have at least 2 SIDs allocated, one for downstream Best Effort Service (BE) traffic and a second for upstream BE traffic. The upstream SIDs at the CM are implemented as FIFO queues. Other types of traffic, such as VoIP, might be assigned to a different SID that supports a different scheduling service; e.g., Unsolicited Grant Service (UGS) for toll quality telephony. The DOCSIS specification purposely does not specify the upstream bandwidth allocation algorithms so that vendors are able to develop their own solutions. DOSCIS requires CMs to support the following set of scheduling services: Unsolicited Grant Service (UGS), Real-Time Polling Service (rtPS), Unsolicited Grant Service with Activity Detection (UGS-AD), Non-Real-Time Polling Service (nrtPS) and Best Effort Service (BE).

**Unsolicited Grant Service (UGS)** is designed to support real-time data flows generating fixed size packets on a periodic basis. For this service the CMTS provides fixed-size grants of bandwidth on a periodic basis. The CM is prohibited from using any contention requests. Piggybacking is prohibited. All CM upstream transmissions must use only the unsolicited data grants.

**Real-Time Polling Service (rtPS)** is designed to support real-time data flows generating variable size packets on a periodic basis. For this service the CMTS provides periodic unicast request opportunities regardless of network congestion. The CM is prohibited from using any contention requests. Piggybacking is prohibited. The CM is allowed to specify the size of the desired grant. These service flows effectively release

their transmission opportunities to other service flows when inactive, demonstrating more efficient bandwidth utilization than UGS flows at the expense of delay.

**Unsolicited Grant Service with Activity Detection (UGS-AD)** is designed to support UGS flows that may become inactive for periods of time. This service combines UGS and rtPS with only one being active at a time. UGS-AD provides Unsolicited Grants when the flow is active and reverts to rtPS when the flow is inactive.

**Non-Real-Time Polling Service (nrtPS)** is designed to support non real-time data flows generating variable size packets on a regular basis. For this service the CMTS provides timely unicast request opportunities regardless of network congestion. The CM is allowed to use contention request opportunities.

**Best Effort Service (BE)** is designed to provide efficient service to best effort traffic. The CM is allowed to use contention or piggyback requests for bandwidth.

### 2.3 Review of DOCSIS Scheduling Literature

DOCSIS 1.0 provided only a best effort service to the user. In this version resource allocation dealt foremost with fairness, but some efforts were studied to allow different service levels to be provided. Resource allocation in these early systems also dealt with the allocation of data slots versus contention slots for transmission requests.

Previous DOCSIS scheduling research can be divided into two general areas. The first is the distribution of upstream mini-slots. What percentage of the upstream bandwidth should be devoted to contention requests in order to optimize throughput and

delay? The second area deals only with data slots and how to allocate them to achieve fairness and different levels of service.

### Scheduling Contention Slots

The first area of research dealing with DOCSIS scheduling involves the allocation of data slots vs. contention slots and the percentage of the bandwidth that should be dedicated to contention. The basic tradeoff to be made is if you provide more contention slots, access delay is reduced since there will be less collisions. However, more contention slots mean less data slots, more overhead, lower channel utilization, and greater latency in transferring data packets.

The link between contention slot scheduling and the collision resolution settings was studied [30] using throughput and the request access delay (RAD) to measure the efficiency of the collision resolution algorithms. The RAD is a measure of how much time a station takes to transmit a request. This is formally defined as the time from the reception of the data until the CM receives an acknowledgement of the request. Two general types of collision resolution were studied. First, Random-select, sets the backoff-start and backoff-end to the same value. Here the allocation of a fixed number of request mini-slots was studied using 3, 6, 8 or 12 mini-slots per MAP. The second method, Ethernet-like, has the backoff-start, S, and backoff-end, E, set to different values. With this approach several allocation strategies were used. In S, every MAP has S contention slots. In E, every MAP has E contention slots. In MeanSE, every MAP has (S+E)/2 contention slots. In Dbl, S slots are allocated unless there were C collisions in the last

MAP, in which case max(2*C, E) slots are allocated. In Exp, S slots are allocated unless there were C collisions in the last MAP, in which case max(2C, E) slots are allocated. In SE, S slots are allocated unless there were collisions in the last MAP, in which case E slots are allocated.

The results show that for the Random-select method allocating 8 mini-slots per MAP provided the best RAD. For throughput, the results were comparable until the load reached 75% at which point most requests are piggybacked and throughput did not increase. For the Ethernet-like method the SE strategy performed the best. The E and MeanSE strategies, which over-allocate contention slots, caused a reduction in throughput. Because there is no way to know how many collisions occurred per slot, over-allocating after collisions have occurred, quickly resolves the collisions. The results showed that the window size should be enlarged when the load is medium and shrunk when the load is light, when there is less demand, or heavy, when piggybacking causes fewer contention requests. A window that is too small causes too many initial collisions while one that is too large defers so long that the RAD increases. The results show that a size range of 4 to 32 works best.

The allocation of contention slots was optimized [43] using probabilities based on monitored performance. The contention process was divided into the two phases of the initial resolution and the collision resolution. The idea was that allocating r mini-slots to resolve r requests maximizes the mini-slot throughput. The problem then becomes one of determining the proper value for r. The technique was to use estimates to determine the probable number of requests. One approach used during the initial resolution phase was

to estimate the number of slots allocated for contention based on requests from previous frames. During the collision resolution phase a table lookup was then used based on the number of allocated slots, the number of successful slots and the number of collided slots. The actual number of requests colliding in a collided slot is unknown. The approach used was to allocate 3 mini-slots for each collided slot. It was shown that most collisions involve only 2 requests, but that the probability of a second collision increases if only two slots are allocated per collision, therefore three slots are allocated to reduce the request access delay. Simulation and analysis were used to show the performance of the various proposed methods of estimation.

A simulation study [8] was used to determine the optimal number of contention slots in a MAP. The first parameter studied was the size of a MAP. After simulating MAP sizes of 1 – 16 ms. it was found that a 2 ms. MAP optimizes throughput, collision avoidance, access delay and buffer sizes. As the offered load increased the performance gap between 2 ms. and other sizes became more pronounced. After determining the optimal MAP size to be 2 ms. the optimal number of contention slots within a 2 ms. MAP was studied. The simulations showed that 6 contention slots is the best choice with a 2 ms. MAP. It was also determined that with these parameters the goodput is about 58.5%.

A queuing model was used [29] to evaluate the optimal fraction, c*, of the channel that should be used for contention slots to minimize response time. This model was composed of N+2 queues, where N is the number of cable modems in the network. The remaining two queues were for the contention and reservation channels. The contention queue holds those packets contending for a reservation. After leaving the

contention queue the packet moves to the reservation queue. The first factor studied was the impact of the size of the contention window. As would be expected, the optimum c* and the mean response time both increase as the window size increases, since increasing the window size decreases collision probability but lengthens the time to transmit and therefore the response time. The next factor studied was the offered load. As the load increases the probability that a new packet finds an empty transmission buffer decreases therefore more opportunity to use piggybacking and less need for the contention channel. The final factor studied was the number of CMs in the network. More CMs causes the traffic to be less regular creating more need for a larger reservation channel. However, more CMs also increased the probability that a new packet arrives to an empty transmission buffer, causing the need for a larger contention channel as well. The study determined that 10 – 15% of the slots in the MAP should be allocated to contention. With lower data rates, closer to 15%, and with higher data rates, closer to 10%, due to the increased use of piggybacking at higher data rates.

A priority based system was developed [18] by allocating different numbers of contention slots for each priority level. Changes to the DOCSIS specification were proposed to use one of the class types for the registration request message (REG REQ) to request a priority. Then a reserved field in the allocation MAP message would be used to assign a priority level to each contention slot. Contention slots are dynamically allocated for each priority level, giving more slots to the higher priority traffic. For the highest priority a different back-off value is set equal to the number of contention slots reserved for the highest priority traffic. A simulation was run using three priority levels. The

highest priority traffic had low access delay even at high load. The lowest priority traffic was treated as best effort and received any remaining contention slots.

Even after priority was established in DOCSIS 1.1 the priority levels did not apply to the contention process. Collisions are not separated and resolved according to those priority levels. A multi-priority access scheme was implemented [28] by setting back-off values that were inversely proportional to the priority. The higher the priority of the flow, the lower the back-off value for that flow was set. In addition, a weighted average of the number of contending CMs is computed and used to dynamically adjust the back-off values to achieve lower access delay for higher priority traffic.

A contention slot allocation algorithm was developed [20] based on the following description. Only allocate enough contention slots so that the average throughput of the contention slots closely matches the number of new packets that can be transmitted in a maximum frame period. Since the random binary exponential backoff algorithm has a throughput efficiency of 33%, the number of requests that can be sent should be three times the number of nrtPS and BE packets that can be transmitted in a maximum length frame. If there are more outstanding requests than can be serviced in the next two frames then no contention slots should be allocated and piggybacking should be disabled to prevent any new requests. In DOCSIS 1.1 priority levels were added. After determining the proper number of contention slots per frame, the next step was to divide those slots between the different priority levels. The approach used was to develop a weighting for each priority, probably based on a pricing model, and allocate that percentage of the total slots to each priority. To attempt to try to avoid assigning slots that will not be used, a

moving average was maintained of the number of slots used by each priority level. This was used as a prediction of the number of slots needed by each level. If the weightings gave more slots to a given priority than the prediction indicated was needed, then the slots were redistributed to other priorities, preferably higher ones.

### Scheduling with bandwidth and delay guarantees

The second area of DOCSIS scheduling research deals only with data slots and how to allocate them to achieve different levels of service. This second area can be broken into the two versions; DOCSIS 1.0 without the availability of different service levels, and DOCSIS 1.1 after the different QoS levels were added to the standard.

### Without QoS services (DOCSIS 1.0)

The DOCSIS 1.0 standard did not provide different levels of service, therefore early efforts to provide QoS capabilities had to deal with how to prioritize the traffic. As indicated in the contention slot section, some early attempts to provide priority levels utilized the allocation of different numbers of contention slots so that some flows could get more opportunity to transfer data quicker. The other method was to set up different back-off values to establish priority by giving some flows earlier opportunities to send data. In this section we look at some of the research that attempted to delineate levels of service in DOCSIS 1.0. Most of these efforts dealt with bandwidth fairness.

A simulation study was used [38] to provide a baseline characterization of DOCSIS 1.0 performance using a prioritized first come first serve algorithm. Both

isochronous traffic and on-off traffic were studied. No fragmentation or concatenation was used in this study. It was shown that with on-off traffic and 1500 byte packets, the maximum throughput was 1965 kbps. With small packets of 100 bytes the throughput is only 1550 kbps. The use of concatenation could improve the small packet performance. It was found that isochronous traffic such as VoIP results in a mean access delay in the range of 10 – 20 ms. The conclusion was that these services could only be supplied using a proprietary committed information rate service.

In [41] a simulation study was used to examine the capacity and delay characteristics using DOCSIS 1.0 for delivery of isochronous streams from 8 – 64 kbps. It was found that the number of streams supported was lower than theoretical maximum due to protocol overhead, collisions, and the lagging effects of the request/grant process. Packet size effects throughput with larger packets providing better throughput. Smaller MAPs provide more frequent opportunities to schedule high priority streams and therefore provide better response time. However, it will result in more collisions and therefore more wasted bandwidth. The final conclusion was that delay sensitive applications can not be handled by DOCSIS 1.0 unless placed on a dedicated channel.

As previously discussed the RAD was used as a means of measuring the efficiency of the collision resolution algorithms. In the same study [30] the Data Transfer Delay (DTD) was used to measure the efficiency of the transmission scheduling algorithms. The DTD is defined as the time beginning when the CMTS receives the request and puts it into the scheduling queue and ends when the CM completes the transmission of the packet. Two simple algorithms were studied, Shortest Job First (SJF)

and Longest Job First (LJF). As expected, SJF had the lowest DTD but the worst RAD. The LJF was the opposite. This is explained by the fact that with a small DTD the queue at the CM has a greater probability of being empty and therefore less use of piggybacking occurs. A modified SJF (MSJF) algorithm was introduced which, rather than allocating all data slots consecutively within the MAP, allocates the data slots in blocks distributed throughout the MAP, thereby allowing more time for additional packets to arrive at the queue. Since the end of the transmission is delayed this increased the DTD but decreased the RAD by increasing the use of piggybacking. The MSJF algorithm provided a good balance between RAD and DTD.

A simulation was used [37] to test a scheduling algorithm to guarantee both minimum bit rates with fairness, for best effort traffic, and delay bounds, for CBR traffic such as VoIP. For the bit rate guarantees the algorithm for fair bandwidth assignment was based on accumulated bandwidth usage. A table was maintained indicating the number of bits granted to each stream. Every few MAPs, 20 in the simulation, the table is sorted by this usage amount. Higher priority is given to the streams that have received the lowest grant amount. After every 1000 MAPs the amounts in the table were cleared to prevent an idle flow from being granted excessive bandwidth when traffic resumed. The study showed all flows received more than their minimum bit rate guarantee and all excess bandwidth was fairly distributed between all flows. A second simulation was run with both best effort traffic and constant bit rate traffic with a delay bound. The algorithm for the delay sensitive portion of the traffic assigned a fixed maximum percentage of the bandwidth for delay sensitive flows and did not accept new delay

sensitive flows if that percentage would be exceeded. If a flow was rejected it was given the option to enter the system as a best effort flow. As delay sensitive flows were admitted to the system the bandwidth needed by that flow is removed from the bandwidth allocated for best effort traffic. The new delay sensitive flow was granted a slot to transmit upon being granted admission. It was assumed that the CM would then piggyback all ongoing requests after that unless the flow became idle. When the flow resumed, a process referred to as dynamic polling used one of two mechanisms to restart the grant process. If the CMTS sensed no collisions it waited for a request from the CM in one of the contention slots. If collisions were occurring, the CMTS polled the delay sensitive flows. This provided more efficient use of the bandwidth than just using polling on all delay sensitive flows during every MAP.

In an effort to improve TCP performance over DOCSIS networks a new scheduling algorithm was proposed [24] called Long Packet Deferment (LPD) whose goal was to reduce the sending rates of long packets and increase those of short packets in an attempt to achieve true fairness in sharing. Since each CM can have only one grant per MAP, regardless of the packet size, large and small packets are treated the same. This causes round trip times to be the same on both upstream and downstream channels. Due to the asymmetry of the channels this causes the downstream bandwidth to be poorly utilized for TCP traffic, which depends on the ACK return for clocking. LPD assigned a deferment value to each packet based on its length and the packet was placed into a priority queue network with the lower deferment values having higher priority. The deferment value indicated the number of MAPS that must pass before the packet is

scheduled, unless all shorter packets have already been scheduled. As each MAP was sent, the deferment values were adjusted and the packets were moved to the proper queue. Simulation results showed that LPD provided performance improvements over first come first serve scheduling. Both downstream bandwidth utilization and access delay was also improved.

**With QoS services (DOCSIS 1.1)**

The DOCSIS 1.1 standard added the different levels of service that the previous research showed were needed to achieve delay guarantees. Research now turned to how to use those available services to provide not only guaranteed bandwidth but also delay guarantees to flows that require them.

Namman and Rom did a series of work on DOCSIS scheduling based on the bin packing problem. In [32] the problem of scheduling best effort traffic into the fixed size gaps left between the previously scheduled UGS packets, using fragmentation, was investigated. In this instance the items to be packed are the best effort packets and the bins are the mini-slots available after UGS is scheduled. Their study started with an analysis of the Next Fit (NF) algorithm and then adds fragmentation ability to the algorithm (NFf). The algorithm continues to place packets into the bin until the next one will not fit. At that point the current packet is fragmented to fit and the bin is closed, the next bin is opened and the remaining fragment is placed into the new bin. In [33] the same problem was studied with variable size bins, i.e. the restriction was removed that all gaps between the previously scheduled UGS packets be the same size. The variable bin

size version is related to the multiple knapsack problem, known as subset sum, where the profit of each item is equal to its size. Example results for channel utilization for the average case were; for NF, 79%, for NFf ,98%. For worst case they were; for NF, 50.5%, and NFf ,98%. Their conclusions were that scheduling efficiency increases with bin size, the ability to fragment improves efficiency considerably, and while the algorithm is inefficient, it is simple and runs in linear time.

In [34] Namman and Rom move their investigation of bin packing work to the problem of scheduling CBR (UGS) flows, the step that came previous to the scheduling of the best effort packets. They studied two distinct cases; the situation where all flows have the same grant interval, and the case where there are two different grant intervals but one is a multiple of the other. They start by showing that for the case where all flows have the same grant interval the scheduling problem is easy. For the case with two different grant intervals, with one a multiple of the other, they deal with the Feasible-Set problem and the Optimal-Schedule problem. The Feasible-Set problem is a decision problem dealing with the question of whether there exists a legal schedule, while the Optimal-Schedule problem deals with finding a legal schedule that is optimal for a subset of the flows. In this case the two optimizations we are interested in is the maximum number of flows, or maximizing the channel utilization. They show the Feasible-Set problem to be NP-complete and the Optimal-Schedule problem to be NP-hard. An approximation algorithm, Next Fit with Jitter (NFJ), was developed. This was a modification to the standard Next Fit approach which also accounts for possible variations in bin size based on the jitter constraint. One bin is still open at a time and

packets are scheduled into the bin until the bin is full then the bin is closed and a new one is opened. The algorithm allows for the possibility of using the jitter to change the bin size if needed, adjusting the size of the next bin as appropriate. It is then shown that when grant size is smaller than tolerated jitter, a common occurrence for flows such as VoIP, NFJ is optimal.

A system was developed [12] to implement QoS scheduling to provide both bandwidth and delay guarantees. To provide bandwidth guarantees the system uses a SCFQ scheduler. All requests from CMs are time stamped as they enter the queue for the SCFQ scheduler. Several issues concerning the updating of the virtual clock in an HFC system are discussed. The problems occur due to the fact that transmissions are not continuously flowing but are blocked together by the MAP generation process. It was shown that each time a MAP is sent the virtual time should be updated to the finish time of the last grant in the preceding MAP. Since there is no bound on the access delay time, due to the contention resolution process, the HFC scheduling discipline alone is unable to provide delay guarantees. To provide delay guarantees in this system the DOCSIS UGS service is employed. A separate scheduler was provided for delay guarantees using a shaped virtual clock scheduler. Each time a request is serviced the next request is generated and stamped with a start and finish time. The system now contains two queues, one for the UGS grants and one for the requested grants. Grants are serviced in order of the earliest finish time, with the UGS queue having priority over the best effort queue.

In [20] a scheduler is developed to provide a full implementation of all DOCSIS 1.1 service classes. Grants to be scheduled are kept in three types of queues. The Type 1 queue is fed by a grant generator which generates all UGS grants and rtPS unicast request opportunities. The remaining queues are fed by the requests received from either contention, piggybacked, or unicast opportunities. Type 2 is used for flows requiring a minimum bandwidth reservation and contains a series of N priority queues. The priority being based on the amount of bandwidth reserved. The single Type 3 queue holds requests for all flows having no bandwidth reservation and functions as the lowest of the priorities in the priority queue chain used for Type 2. The Type 1 queue is FIFO. The individual Type 2 queues are FIFO. The Type 3 queue is a single priority queue. Within the Type 2 and Type 3 priority queue chain a WFQ discipline is used with the weightings based on the minimum bandwidth reserved. If two or more packets have equal virtual finish times then the one with the highest priority is selected for service first.

In [21] a simulation is studied to extend the operation of the UGS and rtPS services to cases where the packets are not generated at fixed intervals. Source traffic models were used, along with monitoring of the inter-packet arrival times to predict when the next grant should be issued. If another packet is waiting in the queue the next request is piggybacked. If no packet is waiting, the arrival time of the packet being sent is noted in an extra field in the header. The mean and standard deviation of the inter-packet time distribution are used to predict the next grant time by adding the standard deviation to the last packet reception time. Two traffic types were studied; video game traffic that is delay sensitive and standard best effort data traffic that is delay tolerant. UGS was used

for the delay sensitive traffic and rtPS was used for the data tolerant traffic, using the source traffic model and the inter-packet distribution to determine the grant times for data or polling requests. The results showed delays that comply with the application requirements while still achieving high bandwidth utilization.

Dealing with VBR traffic over DOCSIS networks has been difficult. The transfer of video across DOCSIS networks was evaluated [7][27][28]. In [7] it was shown that using UGS for video transfer underutilizes the network due to the bursty nature of the traffic. It was then shown that using rtPS introduces too much delay due to the increased time required to wait for the request slot and then receive a grant in the following MAP. A new service is then proposed called Unsolicited Grant Piggyback Request Service. The idea is to provide a periodic grant that is somewhat less than the average bit rate being used and then have the CM use a piggyback request for the remaining slots required at any given time. This new service was required due to the fact that the standard does not allow piggybacking to be used with UGS. This new service achieves the regularly recurring grants so there is no wait for contention, but without the potential waste of a full UGS grant that would not normally be fully utilized. At the same time, it is not necessary to wait for a request grant, then to send a request for the exact amount needed, then wait for the grant to be given. The value of the unsolicited allocation portion determines the tradeoff between the channel utilization and the latency. It was found through simulation that a typical range for the unsolicited allocation portion is [0.3 – 1.1] * average bit rate. A dynamic adjustment scheme was presented to adjust the unsolicited allocation based on two measured variables; the previous unsolicited

allocation and the additional grant amount in response to the piggybacked requests. These two variables are used, along with previously determined increase and decrease constants, to dynamically adjust the unsolicited allocation amount. It was shown that this new service improves both latency and bandwidth utilization over UGS and rtPS, but shows similar results for jitter. The improvement in latency was greater with more bursty data.

A similar proposal was made in [27] and [28]. In this case the size used for the grant amount was calculated as unsolicited_allocation(n+1) = unsolicited_allocation(n) – unused_bytes(n) + piggyback-request(n), where n is the average of the values for the previous N MAPs. Due to the self-similarity of video traffic this estimate for the average worked well.

A two-phase approach was used [44] to assign packets to mini-slots in implementing QoS for DOCSIS. This approach began by defining the satisfying region of each flow. The satisfying region, based on grant interval, size and jitter, defines what space a flow can be transmitted in. A QoS violation occurs when two or more flows have overlapping satisfying regions where a particular mini-slot must be used by more than one flow. The satisfying regions were then used to define the local cost and the global cost of mini-slots. The local cost was defined as the probability of a given mini-slot being occupied by a given flow. The global cost is the maximum local cost among all the flows requiring a given mini-slot.

The first phase of the two-phase scheduling algorithm was used to determine the scheduling sequence. The second phase is the mini-slot assignment phase. In the

scheduling sequence phase a sequence estimator is used to determine the order of the packets within each priority level. The sequence estimator is the sum of the global costs of all mini-slots in a given flow's satisfying region and gives a measurement of the probability of QoS violations. The packets are sequenced by increasing sequence estimator values. Once the sequence of packets is determined the mini-slot allocation phase is used to assign the flows to specific mini-slots within the MAP. In the mini-slot allocation phase an assignment estimator is calculated to measure the probability that a given interval of flows will produce a QoS violation. The assignment estimator is calculated for each contiguous set of mini-slots that falls in a flow's satisfying region. The set with the highest estimator is chosen, thereby leaving open the slots that provide more possible opportunities for the other flows to be scheduled. The findings of the study indicated that the largest improvement came from the mini-slot assignment phase since the satisfying region is so much larger than the grant size.

The only previous research dealing with the switching of channels appears to be from Namman and Rom [35] where the switching of telephony calls between channels in a DOCSIS 1.1 system was considered using a simple case with only UGS flows. Each CM has access to multiple channels but can only use one at a time. This investigation dealt with the case where all calls have the same values of grant interval, grant size and jitter. Even with this simplifying restriction the problem was shown to be NP-hard. Each upstream channel was divided into frames equal in length to the grant interval of the calls. Each frame contained U call slots therefore, since all calls on a CM must be on the same channel, a single CM could have at most U calls. A CM can be switched to another

channel, but this requires all calls on that CM to be switched to the new channel, without violating the jitter constraint of the calls. The scheduling problem thus changes from selecting a time slot to selecting a time slot and on which channel with the restriction that all calls on a CM must remain on the same channel and be switched together. Each frame was divided into W jitter windows where the length of W is less than the tolerated jitter of the calls. This allows a call to be moved between channels if it remains in the same jitter window. This approach simplifies the scheduling problem but imposes scheduling restrictions that degrade performance.

# CHAPTER THREE

# DEFINING FAIRNESS IN DOCSIS 3.0

As discussed in Chapter 2, GPS provides the ideal definition of what constitutes fair queuing at a conceptual level and provides a baseline that can be used to determine how closely a packet based scheduling discipline comes to this theoretical, non-packet based ideal. Our first task is to develop a model and an implementation to calculate those fair allocations in a downstream, multiple channel environment.

## 3.1 Max-min Fairness

Strict fair sharing is concerned with how to evenly divide a resource among several unequal requests, especially when some requests require fewer resources than others. This does not pose a problem when the sum of the requests is less than the available resources. The real question to answer becomes, when the sum of the requests is greater than the available resource, how should that resource be divided to achieve fairness?

The first approach is to discount all requests by the proportion of total demand above the available resources. This approach under services all requests by the same proportion and penalizes small requests and large requests equally. An alternative is to allocate the same proportion of the available resource to every request. This approach penalizes the larger requests at the expense of the smaller requests.

The early literature focused on the 'flow control' problem in packet switched networks [4] [11] [15] [16] [23]. Fairness was considered an outcome of a given flow control method. Gerla and Kleinrock [15] indicate that fairness is the fair allocation of resources among competing users and that unfairness is a natural byproduct of uncontrolled competition.

With a single channel of capacity C and *n* users the fair share allocation is simply C/*n*. Fair, in this case, means equal. However, there are flows that will require less than C/*n*. In this case the excess not required by one flow should be shared equally with all other flows. Keshav [26] defines an algorithm for this case of equal fairness:

- Resources are allocated in order of increasing demand.
- No source gets a resource share larger than its demand
- Sources with unsatisfied demands get an equal share of the resource.

After sorting the requests into increasing order, the total capacity of the channel is divided by the number of requests producing the equal fair share amount. The first, smallest, request is granted the lesser of this fair amount, or its request. The amount assigned is deducted from the total capacity amount. The process is then repeated until all requests have been assigned.

As an example assume that the resource capacity is 20 units. The requests are for 4, 5, 8, and 10 units. Dividing the resource into four equal parts gives us 5 units. Request 1 is satisfied with 1 unit remaining to be shared by the other 3 requests. This leaves 15 + 1, or 16 units to be shared among the three remaining requests, allocating

each 5.33 units. Request 2 is therefore satisfied with .33 units remaining, leaving 11.33 units to be shared by requests 3 and 4. Therefore both requests 3 and 4 receive 5.66 units.

For a system that wishes to provide different levels of service to different users a logical extension of this algorithm is to weight requests to reflect different priorities. If there are two requests with weightings 1 and 2, respectively, two units of service will be provided to the second request for each unit granted to the first. This is again expressed by Keshav as follows:

- Resources are allocated in order of increasing demand, normalized by weight.
- No source gets a resource share larger than its demand
- Sources with unsatisfied demands get resource shares in proportion to their weights.

Jaffe [23] extends fairness to include different users operating over links of different capacities by claiming that a given allocation of bandwidth is fair if 1) each user's throughput is at least as large as all other users that share its bottleneck link, and 2) the only factor that prevents a user from obtaining higher throughput is the bottleneck link. This definition falls under the umbrella of the widely accepted 'max-min' approach to managing resources which require that flows get the same share of a bottleneck. This amounts to applying Keshav's algorithm to the flows using each bottleneck link, in turn, rather than applying it system wide. Max-min allocation gives preference to low

bandwidth consuming flows by giving the maximum possible bandwidth to the source receiving the least among competing flows at a bottleneck.

Max-min allocation is a commonly used criterion for identifying the correct share of bandwidth allocated to flows in a network. Within the networking community this idea was originally (and independently) proposed by Jaffe [23] and Hayden [19]. The max-min criterion dictates that the smallest session must be as large as possible, and subsequently, the second smallest session must be as large as possible, continuing until further allocations are not possible.

Max-min fairness was defined earlier in the algorithms community by Megiddo [31]. Although the term max-min fairness was not specifically used the algorithm was the same. It was shown that if all possible allocation vectors were sorted in increasing order the lexicographically greatest vector represents the max-min fair allocation. It was also shown that a max-min fair allocation will be a maximum flow for the network proving that achieving fairness does not require sacrificing throughput. In general, this property holds as long as we are dealing with a flow problem – such as the one under consideration here – that can be formulated with a single source or sink, rather than with multiple source-sink pairs.

GPS is an ideal scheduling discipline that has desirable properties. In the context of processor scheduling, it has been shown that GPS based algorithms provide strong fairness on single processor systems however they do not generalize easily in multi-core or multiprocessor environments. In the context of networks, packet-based approximations of GPS have been shown to provide a max-min weighted fair allocation

on a single channel. As we will show in the next section, achieving consistent fairness assuming a GPS-based fair queuing model in a multichannel environment is more complex than in the single channel case. Our work assumes that the desired fairness strategy is max-min fair. Our next step is to build a model of the channel bonded network and implement an algorithm to calculate max-min fair allocations.

### 3.2 The Channel Bonded Network as a Network Flow Problem

The channel bonding system input can be modeled as a set of three vectors describing demands, channel capacities, and the mapping between flows and channels. A demand vector $D_i$, i = 1 … *n*, holds the individual bandwidth requests of *n* flows. A channel vector $C_j$, j = 1 … *m*, holds the bandwidth of each of *m* channels. A two-dimensional binary channel map $M_{ij}$ indicates the channels available to each flow; if $M_{ij}$ = 1, then flow i is connected to channel j.

The output of the process is described by two vectors describing allocation of bandwidth to each flow and allocation of bandwidth to each channel. The vector $A_i$, i = 1 … *n*, describes the assignments, where $A_i$ is the allocation to flow i. Each entry $CA_{ij}$ in a two-dimensional channel allocation map indicates the amount of flow i assigned to channel j.

The DOCSIS 3.0 standard [1] allows a bit in the "Provisioned Attribute Mask" to indicate if a channel is treated as a single channel or is part of a bonding group. Without loss of functionality, in this study we treat individual channels as single channel bonding

groups and treat the system as allowing a bonding group to consist of one or more channels.

We conjectured that max-min fairness is an appropriate model to use for the DOCSIS 3.0 scenario considered here, since it provides both a fair allocation of bandwidth as well as maximum utilization of the available bandwidth. A flow network model of the channel bonding system was built to implement Megiddo's max-min fair algorithm and calculate the fair flow allocations. This was implemented in two phases. The first was to find an allocation for which min $\{A_i : i = 1 \ldots n\}$ is as large as possible. That is, we wish to find an allocation maximizing the amount of bandwidth received by all flows. The second phase then finds a max-min fair allocation (in which the allocation vector A, sorted in increasing order, is lexicographically maximal), by repeated application of the preceding algorithm. This max-min fair allocation will also maximize the total bandwidth allocated to all flows.

A flow network [3] [10] G = (V, E) is a directed graph in which each edge $(u, v) \in E$ has a nonnegative capacity $c_{uv}$. There is a single source vertex *s* and a single sink vertex *t*. The maximum flow problem involves determining the maximum flow through the network from *s* to *t*, subject to the capacity constraints of all edges. The Ford-Fulkerson algorithm [13] can be used to find the maximum flow through the network. The Ford-Fulkerson algorithm also provides the flow allocation across each edge (or in our case channel) of the graph in the maximum flow.

A flow network graph can be constructed to model the channel bonded system by starting with a bipartite graph with a left vertex for each flow and a right vertex for each

channel. Edges are drawn from each individual flow vertex to the channels that the flow has access to. The single source vertex *s* has an edge to each flow vertex, and each channel vertex has an edge to the single sink vertex *t*. Figure 3.1 shows an example network where flow 1 can access channels 1 and 2, flow 2 can access channel 2, and flow *n* can access channels 2 and *m*.

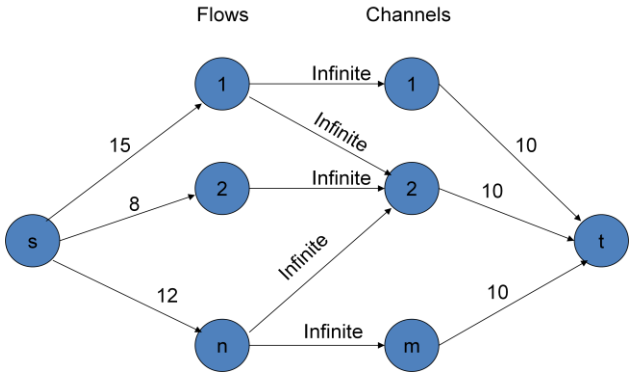


Figure 3.1: Flow Network for Channel Bonded System.

The capacities for each edge are shown on the graph. Each source to flow edge is given a capacity equal to the bandwidth request amount of the corresponding flow, the values in the demand vector D. In this example D = (15, 8, 12). Each channel to sink edge is given a capacity equal to the bandwidth of that particular channel, the values in the channel vector C. Again, for this example C = (10, 10, 10). Each flow to channel edge is given infinite capacity, so that the flow is dependent only on the request amounts and the channel bandwidth. Running the Ford-Fulkerson algorithm on the resulting network will yield the maximum flow through this network.

In this example, using a breadth first search to find the augmenting paths in the Ford-Fulkerson algorithm, the allocation vector is A = (15, 5, 10) and the maximum flow value is 30. This can be verified by the max-flow min-cut theorem, which states that the maximum flow through a network is equal to the aggregate capacity of a minimum cut separating the source from the sink. In our example, a minimum cut of capacity 30 (matching the value of our flow) separates all vertices but the sink from the sink.

### 3.3 Finding Fair Allocations

As was the case above, finding a maximum flow through this network will not usually provide a fair share assignment. The maximum flow problem isn't concerned with fairness, only aggregate throughput. However Megiddo's approach shows how to determine a fair allocation by solving a succession of maximum flow problems.

The first step in Megiddo's max-min fair allocation algorithm is to find an allocation A whose minimum component is as large as possible. That is, we wish to maximize the value of x such that we can find an allocation vector with $A_i \geq x$ for i = 1…*n*. Perhaps the simplest approach for this problem is to binary search on x. In each step of the binary search, we set x to the capacity of each outgoing edge from the source, and we then check if a maximum flow fully saturates all of these edges. If not, our guess for x was too high, since there is no way to allocate at least x units to every flow, so we revise our guess for x downwards. Otherwise, our guess was correct or too low. In theory, this approach might loop forever if the final maximum value of x is a number like 1/3 that has no exact binary representation. However, in practice, we can terminate the

binary search once it has determined x to within some desired tolerance. One can also apply more sophisticated algorithms to determine x more efficiently; for example, Gallo, Grigoriadis and Tarjan[14] describe a more complicated algorithm for solving this problem (which they call a parametric maximum flow problem) in the same worst-case running time as a single standard maximum flow problem. In either approach, we limit the initial range for x to $[0, \min D_i]$, since a higher value would unnecessarily allocate more bandwidth to some flow i than its demand $D_i$.

Once we have determined the maximum value of x (possibly equal to the minimum demand) such that all flows can be allocated at least x units of bandwidth, we will have reached a bottleneck point where further increases in x need to "leave some flows behind", as there is no way to allocate more than x units of bandwidth across the board to every flow. Specifically, there will be some subset $S \subseteq \{1, \ldots, n\}$ of flows for which $x = D_i$ for all $i \in S$, or if this is not the case, then there must exist some subset S of flows such that it is impossible to allocate strictly more than x units of bandwidth to all flows $i \in S$ simultaneously. In the second case, we can locate S using the maximum flow minimum cut theorem, by including flow i in S if and only if all of the channels available for use by flow i are completely saturated by our maximum flow solution (since we might have terminated our binary search on x early to avoid an infinite loop, we must treat nearly saturated channels as saturated for this purpose). Note that the flows in S are those that cannot be unilaterally increased, while the remaining flows may still be able to accept higher bandwidth allocations. We therefore "freeze" the flows in S, never again raising their associated capacities in our flow network. For the remaining flows, we

repeat the entire process again, increasing their allocations by binary searching for a new value of x such that all flows (excluding those frozen in S) can be assigned at least x units of bandwidth. We then identify a second set of "frozen" flows, and repeat the process iteratively until all flows are finally assigned. The final result will give an approximate max-min fair allocation (approximate due to the fact that we may terminate our binary searches early) that is also a maximum flow.

Since all flows do not have access to the same set of resources (channels), it is possible to find the max-min fair amount that can be provided to every flow and still have bandwidth remaining on some channels. This can occur when demand from all flows connected to a given channel is less than the capacity of that channel. For example, if the demand vector in Figure 3.1 were changed to D = (15, 8, 5), since channel *m* would only be connected to flow *n* with a demand only equal to half the channel capacity, the remainder of channel *m* would be unusable. Without changing the bonding group assignments this bandwidth cannot be utilized.

The flow network graph was coded to implement the max-min fair algorithm. It was then tested to provide allocations for each flow to channels to achieve max-min throughput. Figure 3.2 is an example result showing the output of the algorithm. In this example there are ten flows all with demand 1000, and four channels all with capacity 1000. The map M provides a channel assignment designed to prove the lexicographically maximum assignment of the allocations. This example has three single channel bonding groups (1, 2 and 3) which all intersect multi-channel bonding groups. In addition there

are three multi-channel bonding groups (channels 1 and 2, channels 2 and 3, and channels 3 and 4). The bottleneck link (channel) with the largest number of flows is channel 4.

D = (1000,1000,1000,1000,1000,1000,1000,1000,1000,1000)

C = (1000,1000,1000,1000)

| | **Max-min fair allocations** | | | |
|---|---|---|---|---|
| | C0 | C1 | C2 | C3 |
| M = ( (1,1,0,0) | 1000 | | | |
| (0,1,0,0) | | 500 | | |
| (0,1,1,0) | | 500 | | |
| (0,0,1,0) | | | 333 | |
| (0,0,1,0) | | | 333 | |
| (0,0,1,1) | | | 333 | |
| (0,0,0,1) | | | | 250 |
| (0,0,0,1) | | | | 250 |
| (0,0,0,1) | | | | 250 |
| (0,0,0,1) ) | | | | 250 |

Figure 3.2: Allocation Example.

It can be seen that the initial fair allocation was 250 to all ten flows. At that point the bottleneck at channel 3 is settled. The algorithm continues with the flows that have access to channels 0, 1, and 2. Next the channel 2 bottleneck is divided with 333 to each flow. The process then continues with the flows having access to channels 0 and 1. Those two flows get 500 each, leaving capacity only on channel 0. The increase continues until channel 0 is full at 1000.

As shown, this model satisfies the objective to develop a theoretical fair queuing model that defines what constitutes fairness within a channel bonded environment and provide an implementation that will calculate fair allocations.

# CHAPTER FOUR

# IMPLEMENTING FAIR SCHEDULING IN DOCSIS 3.0

In the previous section we demonstrated an offline approach for finding the max-min fair allocation over a network involving bonded channels. In this section, we explore online algorithms for achieving max-min fair allocation in DOCSIS 3.0. Our objective is to design and evaluate specific packet scheduling techniques that could be implemented by a CMTS.

Two widely studied classes of scheduling disciplines are those based on round robin scheduling and those based on time stamp scheduling. As presented in the background section, both categories of scheduling have been thoroughly studied in the literature. The majority of prior work has focused on Internet or high speed networks. The portion of this prior research that addresses cable networks has typically focused on the upstream allocation problem. Bonded channels complicate bandwidth management in either the upstream or downstream direction. It seemed appropriate to focus on the simpler case of downstream and to defer the upstream study to future work. In addition to focusing just on downstream, we further limited the scope of the study by selecting two scheduling disciplines, one from each category: DRR and SCFQ. We leave for future work the study of a broader set of algorithms and a more thorough analysis that considers important issues such as worst case fairness and packet delay bounds and computational complexity. Our study focused on the ability of the two selected schedulers to approximate max-min bandwidth allocation in a downstream multichannel environment.

Figure 4.1 illustrates a multichannel fair queuing network model. If a packet from flow *i* (with demand $d_i$ and weight $w_i$ ) arrives to a busy channel it is inserted into $f_i's$ queue. Once the channel becomes available, the packet scheduler selects which flow to service. The scheduler must factor in weight information that has been configured. Channel bonding complicates this process because not all flows are going to the same channel and even more so because any packet in a given flow could be forwarded to one of several channels. For the single channel case, the scheduler inherently preserves the sequence of packets that get transmitted. This is not true in the multichannel case as channels might be overlapped (i.e., multiple flows are allowed to use the channel). In this case, the scheduler needs more information to ensure the correct service order.

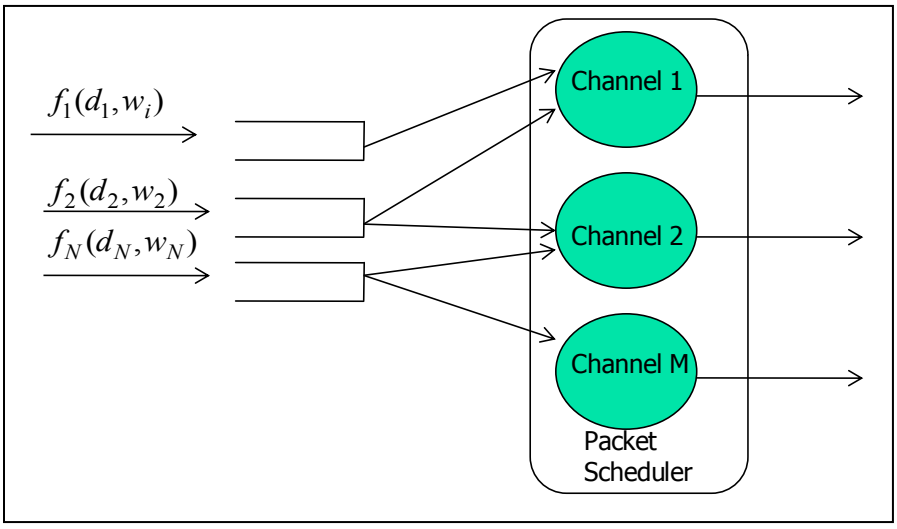


Figure 4.1. Multiple Channel Network Model

Crucial to the design of multichannel scheduling algorithms is the location or 'perspective' of the additional information that is required to ensure proper service ordering. We define the two 'perspectives' that were studied in our research:

- *Global scheduling* maintains all state associated with the flow selection algorithm using information that is maintained at the global per-flow level.
- *Channel scheduling* maintains algorithm state at the individual channel levels.

We explore implementations of DRR and SCFQ packet scheduling algorithms that are based on global or channel scheduling perspectives.

### 4.1 Simulation System Model

The DOCSIS 2.0 *ns*2 simulation model that was developed in prior work was extended to support the downstream DOCSIS 3.0 capabilities. Figure 4.2 is a functional diagram of the scheduling system model that was implemented in *ns*2. This would be a component of the CMTS simulation model. Any number of channels can be assigned to cable modems as well as to flows. The assignments are statically made at the start of the simulation. The DOCSIS 3.0 specification considers a bonding group to be an abstraction that helps manage and organize the use of multiple channels across sets of diverse users. A DOSCIS implementation could potentially devise a hierarchical management scheme allowing flows to be managed at the bonding group level and then to be further managed at the channel level. Our system model is a single level scheme

that operates on all flows that have been assigned to a set of channels. We use the concept of bonding groups only to help configure channels to flows.

As illustrated in Figure 4.2, a token bucket filter regulates the arrival process and passes the shaped stream to per flow scheduling queues. For the results presented in this dissertation, the regulator was disabled so that flows were not subject to service rates.

The public entry points to the scheduler function identified in Figure 4.2 are as follows:

- init(): The scheduler's data structures and state is initialized
- packetArrival() : This routine receives packets from the regulator, finds the possible set of channels the flow has been assigned, selects the first available channel, and forwards the packet by invoking the SendFrame routine. If all channels are busy, the packet is queued.
- selectPacket(): This is invoked when a channel becomes available (i.e., a previously assigned frame transmission completes). On entry the routine creates the ActiveList (the list of flows with data queued). The state associated with the algorithm (e.g., the deficitCount or the serviceTags for DRR and SCFQ respectively) is maintained either globally (a single array indexed by all active flows) or on a channel perspective (a double array indexed by the active flow index and the channel index). The routine selects which flow to service next, dequeus the packet, and forwards it over the channel by invoking the SendFrame routine.

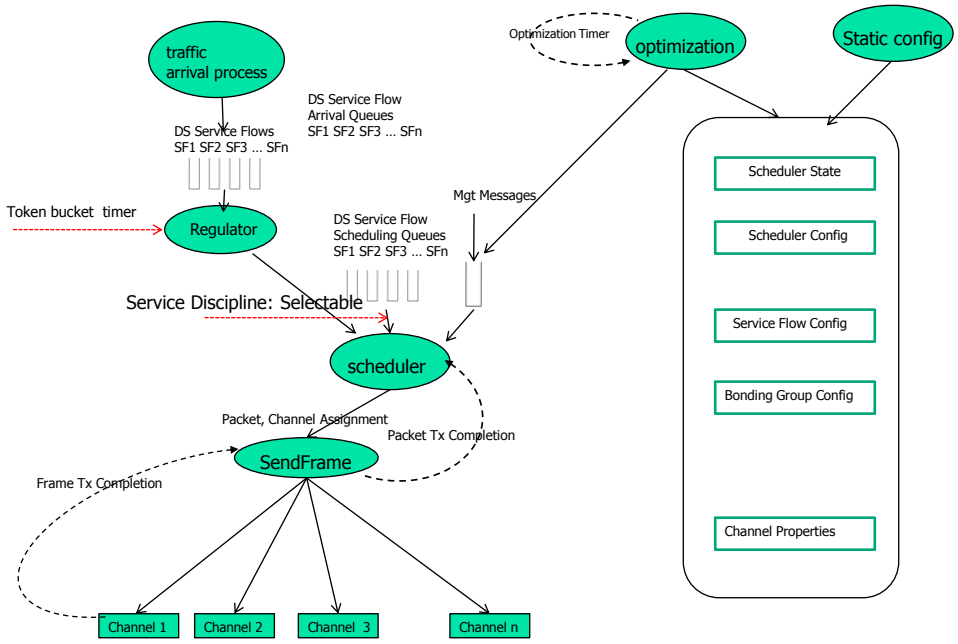


Figure 4.2: DOCSIS 3.0 *ns* System Model Scheduling.

The optimization component identified in Figure 4.2 represents flow-to-channel remapping that would periodically be performed. This function is discussed in more detail in the next section.

We have studied both global and channel perspective approaches for DRR and SCFQ packet scheduling. In the next sections we describe the implementations and in Section 4.4 we present our analysis.

### 4.2 DRR in a Channel Bonded Network

An implementation of DRR in a channel bonded system is complicated by each flow having access to different numbers of channels. This makes the tracking of a round problematic since between one round and the next a given flow may be up for transmission when a channel is unavailable, or a flow may be included in multiple rounds for different channels.

In a global scheduler implementation the rounds can require skipping of flows based on whether the proper channel becomes available when that flow is next up in the round. A given flow would have to be skipped if their turn in the round arrived and there was no available channel to transmit a packet from that flow. Therefore there is no way to cycle through a normal round of service since we can not guarantee the flow the opportunity to transmit at any given time. Therefore precise implementation of DRR would then require more than just a sequential list of flows to implement the round robin cycle. Correct round robin operation with a global scheduler would likely require significant list handling overhead. We chose not to present results involving global DRR.

In a channel scheduler implementation there would be a separate round for the flows in each channel. In this case the rounds would always provide flows that can transmit their packets. However, now we have the situation where the same flow will appear in more than one round robin list, giving them multiple transmission opportunities. A channel scheduler implementation, as with the global scheduler, also makes it impossible to guarantee fairness.

We implemented DRR as a channel scheduler. Figure 4.3 shows pseudo code for the init method and Figure 4.4 shows pseudo code for the selectPacket method. The round array in this implementation contains all flows that can send on that channel even if they have no packets queued. The algorithm skips the empty flows in the list.

```
init()
{
    for (i=0; i<MAX_FLOWS; i++)
        deficitCount[i] = 0;

    for (i=0; i<MAX_CHANNELS; i++)
        indexArray[i] = 0;
}
```

Figure 4.3: Pseudo Code – DRR init.

```
selectPacket()
{
    size = flowArray->size;
    for (j=0; j<size; j++)
    {       // for each flow with access to this channel
            i = indexArray[channelNumber];
            flow[i] = flowArray[i];
            if (packetsQueued > 0)
            {
                    if (pkt->getSize() <= deficitCount[i])
                    {
                            pkt = removePacket();  // send the pkt
                            if (packetsQueued() == 0)
                            {  //  no more pkts in this flow
                                    deficitCount[i] = 0;
                                    break;
                            }
                            else
                                    advanceToNextFlow();
                    }

                    advanceToNextFlow();
            }  // end if queue is not empty
    }  // end for each flow
}


advanceToNextFlow()
{
    indexArray[channelNumber]++;
    if (indexArray[channelNumber] > size)
    {
            indexArray[channelNumber] = 0;
            Add quantum to deficitCount for each flow in the list;
    }
}
```

Figure 4.4: Pseudo Code – DRR selectPacket.

### 4.3 SCFQ in a Channel Bonded Network

Since the input queue process is separate from the scheduler the system allows access to only the head of queue packet. Service tags can't be applied when each individual packet arrives as in standard SCFQ implementations. Instead, an array within the scheduler is used to hold the current service tag of the first packet in each queue. The service tag is computed when the previous packet is transmitted and the service tag of the now departed packet in the internal array is replaced by the service tag for the next packet.

For the global scheduler a single virtual time is maintained for the system and each flow receives a service tag based on that single virtual time reference. This implementation is identical to that of a single channel system when considering the assignment of service tags. The difference is in the selection phase where each flow with access to the newly available channel is searched to find the one with the lowest service tag value. Figure 4.5 is pseudo code for the global scheduler SCFQ init function. Figure 4.6 is pseudo code for the global scheduler SCFQ selectPacket function.

```
init()
{
    for (i=0; i<MAX_FLOWS; i++)
            serviceTags[i] = IDLE;

    virTime = 0;
}
```

Figure 4.5: Pseudo Code – Global Scheduler SCFQ init.

```
selectPacket()
{
    if (more flows on this channel)
    {
        for (i=0; i<FlowCount; i++)
        {       // for each flow with access to this channel
            if (packetsQueued > 0)
            {
                if (serviceTags[flowID] == IDLE)
                {       // this flow has been idle and is restarting
                    pktTxTime = getTxTime();
                    bw_factor = 1 / number of flows on channel;
                    serviceTags[flowID] = virTime +
                                            pktTxTime / bw_factor;
                }

                if(serviceTags[flowID] <= bestTag)
                {       // check to see if this is the earliest tag
                    bestTag = serviceTags[flowID];
                }
            }  // end if queue is not empty
        }  // end for each flow
    }  // end if flows > 0
```

Figure 4.6: Pseudo Code – Global Scheduler SCFQ selectPacket.

```
    if(bestTag != IDLE)
    {       // we found an active flow on this channel
            pkt = removePacket();  // send the pkt
            virTime = bestTag;

            // check to see if there is another pkt in the queue
            if (packetsQueued() > 0)
            {       // set serviceTag for next packet
                    pktTxTime = getTxTime();
                    bw_factor = 1 / number of flows on channel;
                    serviceTags[flowID] += pktTxTime / bw_factor;
            }
            else
            {       // queue is empty, mark as idle
                    serviceTags[flowID] = IDLE;
            }
    }

    // check to see if ALL flows are idle
    all_idle = 1;
    for (i=0; i<MAX_FLOWS; i++)
    {
            if (serviceTags[i] != IDLE)
            {
                    all_idle = 0;
                    break;
            }
    }
    if (all_idle == 1)
            virTime = 0;

}
```

Figure 4.6 (continued): Pseudo Code – Global Scheduler SCFQ selectPacket.

For the channel scheduler a separate virtual time is maintained for each channel accessible to that flow. When a service tag is calculated for the head of queue packet a separate service tag is calculated for each channel available to the flow. A two dimensional array (flows X channels) provides each flow the capability to store a unique service tag for each channel to which it has access. At packet selection time each flow with access to the newly available channel is examined, and the head packet of the flow having the lowest service tag is scheduled. Figure 4.7 is pseudo code for the channel scheduler SCFQ init function. Figure 4.8 is pseudo code for the channel scheduler SCFQ selectPacket function.

```
init()
{
    for (i=0; i<MAX_CHANNELS; i++)
    {
            for (j=0; j<MAX_FLOWS; j++)
                    serviceTags[i][j] = IDLE;

            virTime[i] = 0;
    }
}
```

Figure 4.7: Pseudo Code – Channel Scheduler SCFQ init.

```
selectPacket()
{
    if (more flows on this channel)
    {
            for (i=0; i<FlowCount; i++)
            {       // for each flow with access to this channel
                    if (packetsQueued > 0)
                    {
                            if (serviceTags[channelNumber][flowID] == IDLE)
                            {       // this flow has been idle and is restarting
                                    pktTxTime = getTxTime();
                                    bw_factor = 1 / number of flows on channel;
                                    for (j=0; j<MAX_CHANNELS; j++)
                                            serviceTags[j][flowID] = virTime[j] +
                                            pktTxTime / bw_factor;
                            }

                            if(serviceTags[channelNumber][flowID] <= bestTag)
                            {       // check to see if this is the earliest tag
                                    bestTag = serviceTags[channelNumber][flowID];
                            }
                    }  // end if queue is not empty
            }  // end for each flow
    }  // end if flows > 0
```

Figure 4.8: Pseudo Code – Channel Scheduler SCFQ selectPacket.

```
    if(bestTag != IDLE)
    {       // we found an active flow on this channel
            pkt = removePacket();  // send the pkt
            virTime[channelNumber] = bestTag;

            // check to see if there is another pkt in the queue
            if (packetsQueued() > 0)
            {       // set serviceTag for next packet
                    pktTxTime = getTxTime();
                    bw_factor = 1 / number of flows on channel;
                    for (j=0; j<MAX_CHANNELS; j++)
                            serviceTags[j][flowID] += pktTxTime / bw_factor;
            }
            else
            {       // queue is empty, mark as idle
                    for (j=0; j<MAX_CHANNELS; j++)
                            serviceTags[j][flowID] = IDLE;
            }
    }

    // check to see if ALL flows on this channel are idle
    all_idle = 1;
    for (i=0; i<MAX_FLOWS; i++)
    {
            if (serviceTags[channelNumber][i] != IDLE)
            {
                    all_idle = 0;
                    break;
            }
    }
    if (all_idle == 1)
            virTime[channelNumber] = 0;

}
```

Figure 4.8 (continued): Pseudo Code – Channel Scheduler SCFQ selectPacket.

### 4.4 Results

An implicit result is that ALL packet scheduling algorithms worked perfectly (i.e., they achieve max-min allocation) in any scenario that does not involve channel overloading. Our analysis includes two issues: first, we explain the issue with global scheduling; second we demonstrate the difficulties round robin algorithms must overcome in a multichannel environment.

#### Unfairness in Global Scheduler SCFQ

We begin with a simple example that shows why global SCFQ can not always maintain correct (as defined by max-min fairness) service order. Consider a network with 6 flows and 2 channels. Flows 0 – 3 can access channel 0 and flows 4 and 5 can access channel 1. Assume all flows have equal weight and both channels remain backlogged. Because the time cost is the transmit time (packet length / channel rate) divided by the packet's fractional share of the channel, the head packet of flows 4 and 5 will have timestamps of 2T while the head packet of flows 0 – 3 will have timestamps of 4T.

This means that the global virtual time will alternate between 2T and 4T as packets are scheduled alternately on channel 0 and then channel 1. Assume that at some time flow 4 goes idle temporarily. If flow 4 restarts, just after a packet is scheduled on channel 0, flow 4 will receive a 4T timestamp causing it to be stalled until flow 5 advances to exceed that timestamp. For this reason all subsequent testing was limited to channel schedulers.

**Unfairness in DRR**

We limit the analysis to a simple scenario involving two flows competing for bandwidth in a two channel network. The scenario can be seen in Figure 4.1 if we assume there are two flows and two channels. Flow 1 is assigned to use channel 1, flow 2 can use both channels. The experimental parameters are the bandwidth demand of each flow and the packet scheduling discipline. The flows are configured with a constant bit rate (CBR) traffic generator that sends packets of a configured size periodically to meet the configured sending rate. The default packet size is 1000 bytes. The model assumes ideal channels that operate at 256 QAM providing a raw channel capacity of 42.88 Mbps. We model physical layer and framing overhead by reducing the raw capacity by a constant factor of 10.3% leading to an effective data rate (i.e., the rate that is available to applications) of 38.425 Mbps.

We conducted three experiments:

- Experiment 1: Flow 1 demand is held at 10 Mbps, flow 2 demand is varied from 30 Mbps to 80 Mbps. The flow's packet size was fixed at 1000 bytes.
- Experiment 2: Flow 2 demand is held at 100 Mbps, flow 1 demand varied from 10 Mbps to 60 Mbps. The flow's packet size was fixed at 1000 bytes.
- Experiment 3: Identical to Experiment 2 except the packet size was varied uniformly in the range of [600 bytes, 1400 bytes]. The mean packet size was 1000 bytes so the CBR traffic generator settings were not changed.

Figure 4.9 illustrates the results of Experiment 1. As seen in the figure both DRR and SCFQ performed similarly. Since flow 1 can only transmit on channel 1 and its demand is only 10 Mbps, all flow 1 packets are routed to channel 1. The flow 2 packets fill the remaining bandwidth on channels 1 and 2. When the flow 2 demand reaches 70 Mbps the total demand exceeds the capacity of the two channels and does not increase further.

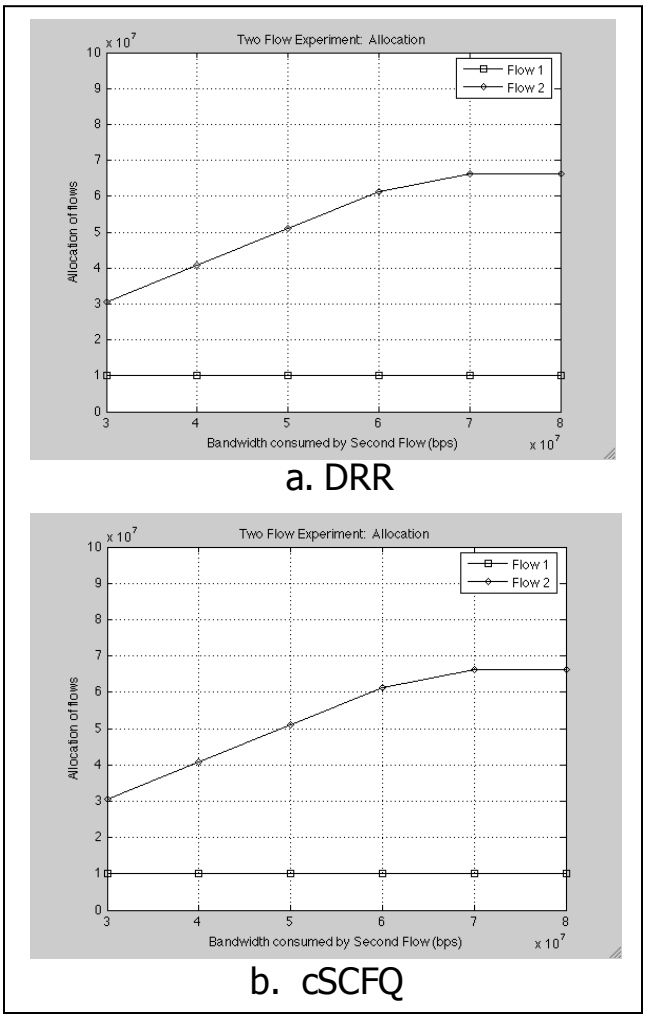


Figure 4.9: Experiment 1: Two Flow Allocation Results

Figure 4.10 illustrates the results from Experiment 2 where flow 2 has a fixed demand of 100 Mbps and the demand for flow 1 is varied between 10 and 60 Mbps. The flow 2 demand alone overloads both channels forcing the scheduler to share channel 1 between the two flows.

Based on the lexicographically maximum assignments of max-min fairness the lowest demand should be maximized and no flow should get less than any other flow unless its demand is less. As is shown in the results in Figure 4.10 DRR reduces the bandwidth for flow 2 and provides increased bandwidth to flow 1.

Logically in this case, to provide max-min fairness, all flow 1 traffic should be routed to channel 1(the only choice) and all flow 2 traffic should be routed to channel 2, providing equal bandwidth to both flows. With round robin scheduling, when a packet needs to be scheduled on channel 2 it will always be from flow 2, the only flow with access to channel 2. However, when a packet needs to be scheduled on channel 1 it will be, round robin, one from flow 1 and one from flow 2. Therefore, DRR will not provide max-min fair scheduling.

As shown in Figure 4.10 the SCFQ scheduler provides equal bandwidth to both flows, based on the timestamps of the packets, and provides max-min fair allocations.

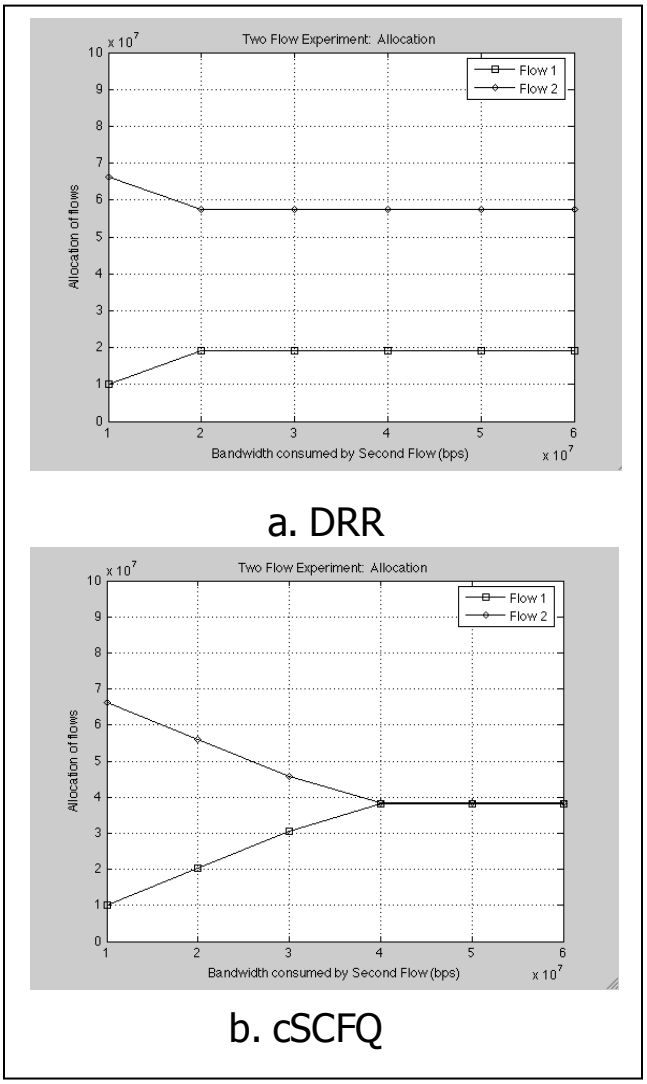


Figure 4.10: Experiment 2: Two Flow Allocation Results

Experiment 3 extends Experiment 2 to ensure the results are not dependent on the packet size. The CBR traffic generator was modified to randomly select the packet size of each transmission based on a uniform distribution in the range of 600 bytes to 1400 bytes. Because the mean packet size is equal to the packet size that was used in Experiment 2, we expect identical results since both DRR and SCFQ were originally designed to operate correctly when subject to variable packet sizes. Figure 4.11 confirms this conjecture.

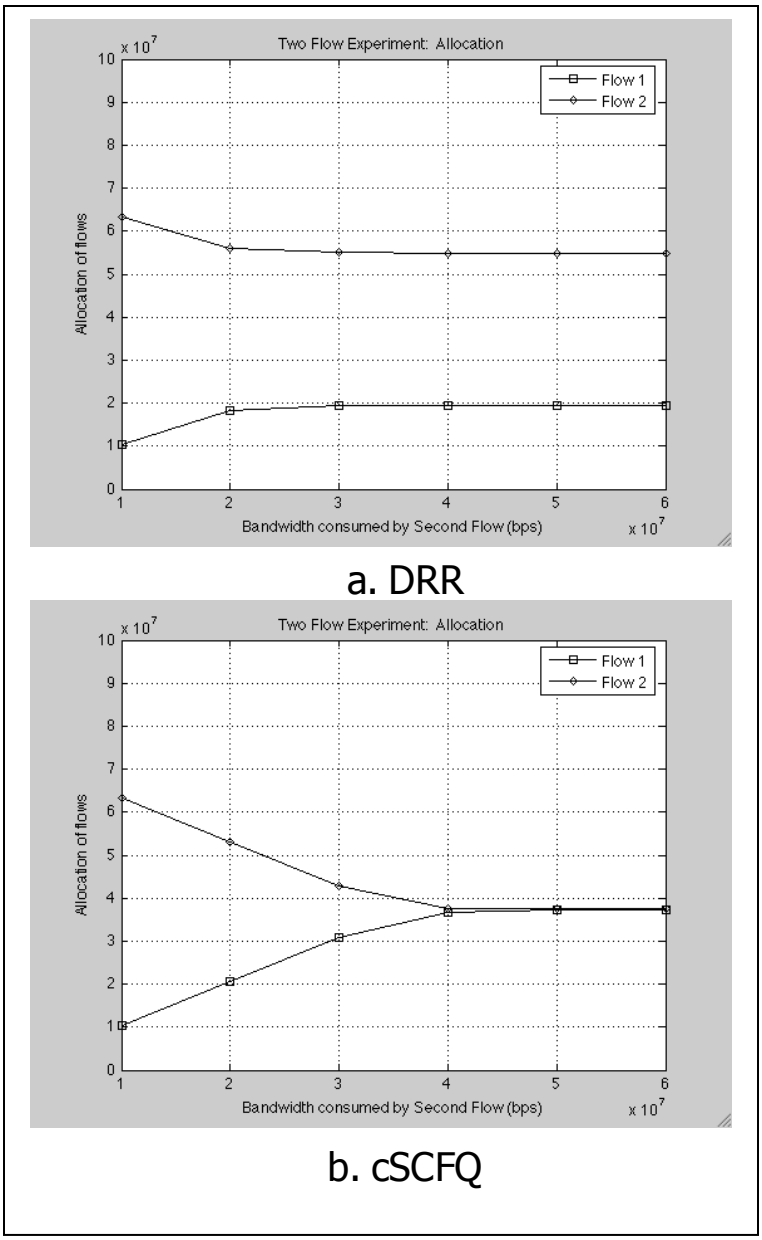


Figure 4.11: Experiment 3: Two Flow Allocation Results with Variable Packet Size

We summarize our results as follows:

- We have shown that a round robin based packet scheduler, such as DRR, will encounter situations where it can not preserve the correct service ordering (as defined by max-min fairness).
- While it is possible to develop more complex round robin based schedulers that potentially offer an appropriate compromise between algorithm complexity and consistent fairness, we have shown that time stamped based algorithms, such as SCFQ, naturally avoid any complexity due to bonded channels.
- Finally, we have shown that, at least in the SCFQ case, the algorithm must maintain state not only on a per flow basis, but also on a per channel basis. In all experiments we performed, we never found a scenario where channel scheduler SCFQ failed to converge to the max-min allocation.

# CHAPTER FIVE

# MAXIMIZING BANDWIDTH UTILIZATION

The scheduling of packets operates at the microseconds time scale. Its purpose is to provide fairness. A max-min fair scheduler solves the problem of fair allocations given the current bonding group assignments. The scheduler can't work outside those restrictions. It is the purpose of the remapping algorithm, operating at the minutes-hours timescale, to fix that problem. The purpose of remapping is to look at the amount of bandwidth that is not used due to the current bonding group assignments and change them if there is still unsatisfied demand. The input to the remapping process is the amount of unsatisfied demand of the flows and the amount of unutilized bandwidth of the channels. When there is a persistent underutilization on any channel, or group of channels, and there are flows with unsatisfied demand, it indicates the need for changing the channel map.

Situations will exist where the current arrangement of bonding groups, and the existing flow demands, will leave bandwidth on some channels unused because no flow with demand remaining has access to those channels. Consider the max-min example results shown in Figure 5.1 below where bonding group 1 contains channels 1 and 2, bonding group 2 contains channels 2 and 3, and bonding group 3 contains channels 3 and 4. Here flow 1 is on bonding group 3, flows 2 – 9 are on bonding group 1, and flows 10 and 11 are on bonding group 2.

D = (6000000,12000000,12000000,12000000,12000000,12000000,12000000,
12000000,12000000,6000000,6000000)

C = (38425000,38425000,38425000,38425000)

**Max-min fair allocations**

| M = | C0 | C1 | C2 | C3 |
|---|---|---|---|---|
| ( (0,0,1,1) | | | 6000000 | |
| (1,1,0,0) | 9606250 | | | |
| (1,1,0,0) | 9606250 | | | |
| (1,1,0,0) | 9606250 | | | |
| (1,1,0,0) | 9606250 | | | |
| (1,1,0,0) | | 9606250 | | |
| (1,1,0,0) | | 9606250 | | |
| (1,1,0,0) | | 9606250 | | |
| (1,1,0,0) | | 9606250 | | |
| (0,1,1,0) | | | 6000000 | |
| (0,1,1,0) ) | | | 6000000 | |

Figure 5.1: Unbalanced Channels.

In this example channels 0 and 1 are fully loaded and flows 1 – 8 have unsatisfied demand. Channel 2 is less than 50% loaded and channel 3 is unused. In this case shifting some of the flows 1 – 8 to bonding channels 2 and 3 can provide all flows with their full demands. A network operator will initially assign channels to CMs, and flows to channels, based on negotiated service plans and anticipated workloads. Remapping is required to periodically rebalance the system as traffic patterns diverge from the expected patterns over time.

Borodin and El-Yaniv [6] describe several related optimization problems with relation to scheduling jobs on multiple machines. They also show that the machine scheduling problem is naturally related to edge congestion minimization in virtual circuit routing. The goal in *load balancing* is "to minimize the maximum load on any machine" or in our case any channel. The purpose of load balancing is, therefore, to even out the load on the channels.

They also describe what they term the *call admission/throughput* problem, where the goal is to "maximize the number of (or profit accrued from) jobs that are scheduled or calls that are routed". In our situation, this involves maximizing the number of demand packets sent. They describe this as follows: "call admission is a packing problem in which, informally, one tries to maximize the profit (e.g., throughput) obtained from packing requests into a constrained environment".

Our objective is a solution to the call admission/throughput problem since the problem requirement is to satisfy as much demand as possible or, stated another way, sending as many packets as possible given the available bandwidth and the current demand. We are not specifically concerned with whether or not we are putting approximately equal amounts of data on each channel, which is the goal of load balancing, but only that we are utilizing as much bandwidth as possible in satisfying the given demand levels.

### 5.1 Online Algorithms and Competitive Analysis

Traditional algorithm design assumes the algorithm has complete knowledge of all inputs. With online algorithms the input is supplied incrementally and the algorithm often must provide incremental outputs. An online algorithm must therefore provide outputs without currently having knowledge of all future inputs.

Online algorithms are often described as a request-answer game where an adversary generates requests and the algorithm must serve them one at a time. Formally, an algorithm A is presented with a sequence $s = s(1), s(2), \ldots, s(m)$. The requests, $s(t)$, $1 \leq t \leq m$, must be served in the order of occurrence. When serving request $s(t)$, algorithm A has no knowledge of request $s(t')$, where $t' > t$.

Sleator and Tarjan [40] suggested comparing the performance of an online algorithm to the performance of an optimal offline algorithm. An offline algorithm is an algorithm that has complete knowledge of all inputs prior to producing its output. Karlin, Manasse, Rudolf and Sleator [25] used the term *competitive analysis* to describe the process of comparing the online result to an optimal offline algorithm. The closer an online algorithm approximates the optimal offline solution the more competitive it is.

If an online algorithm A is compared to an offline algorithm OPT, using the input I, A is said to be c-competitive if

$$A(I) \leq \text{c} * OPT(I)$$

The factor c is called the *competitive ratio* and is the maximum, over all possible inputs I, of $\frac{A(I)}{OPT(I)}$.

The classic "ski problem" from competitive analysis provides an example of this approach. Consider the decision of a skier wanting to determine if it is advantageous to purchase skis or to just rent skis for each trip. If the purchase price is $500 and the daily rental fee is $50, it seems straight forward that if the skier intends to ski more than ten days during the season then the skis should be purchased. However there are many variables involved in answering that question and the input is not available at the beginning of the process.

First of all is the consideration of the weather that year. How long will the season last? (How long will the flow remain backlogged?). How many warm spells will cause the conditions to be unacceptable for skiing? (How often will the flow go idle?). How long will the bad conditions last? (How long will the flow remain idle?).

The skier also must consider how frequently their schedule will permit them to go skiing. How many trips will I be able to make? (How frequently will flows have data to send?). How many days can I stay on each trip? (How long will the flow burst data?). So while the question seems simple (will I ski more than ten days?) there are numerous variables involved that make an answer to that simple question very difficult.

In the request-answer game format, where the adversary provides the input sequence, the adversary will maximize the algorithm cost by making the day that you purchase skis the last day that you ski. The competitive analysis approach is to put an upper (worst case) bound on the result. As an example, if I rent skis until I reach $500 and then purchase skis, I know that I will never spend more than $1000, which is never more than twice the optimal cost. If I don't reach $500 in daily rentals I save money. If I

do it costs me more, but there is a known upper bound on the cost. If the skier skis *n* times and $n \leq 10$, the cost is *50n* which is exactly equal to the optimal cost. If $n > 10$, the cost is 2 x 500, twice the optimal cost which is the cost of purchasing the skis on day one. The algorithm is therefore 2-competitve.

In our problem domain, if a channel is underutilized and there are flows on other channels with more demand than can currently be satisfied, how long should I wait before moving more flows to that underutilized channel, not knowing if the conditions will change? A similar approach to the ski problem can be taken in this case.

### 5.2 Methodology

The remapping problem is an ideal application for an online algorithm. The input to the algorithm is an incremental series of flow demands as time progresses. The output requires the incremental movement of flows to different channels depending upon those changing demands over time in an attempt to utilize additional available bandwidth if all demands are not being met. Due to the unknown future of demand requests, this provided a reasonable application of the competitive analysis approach. Since there is a cost, in lost throughput, when switching a CM from one channel to another it provides a situation similar to the ski problem. In this case the approach is to incur costs, the unsatisfied flow demands, in the short term until the sum of those costs exceeds the future cost of the bandwidth lost to switching channels. Our approach was to use the spirit of competitive analysis to quantitatively analyze the improvement of remapping.

Our methodology involved developing an Integer Linear Program to use as an offline algorithm to find the optimal solution against which to compare the online results. An online remapping algorithm was then developed based on a competitive analysis approach. The online algorithm results were compared to the result without remapping to show the improvement from remapping. The online algorithm results were compared to the optimal offline results to analyze the degree of that improvement.

### Optimal Offline Algorithm

To find an optimal offline result to use as a baseline for the online algorithm an integer linear program was developed. Since DOCSIS 3.0 allows for bonding groups to be redefined during operation, to simplify the remapping, the assumption was made that any flow could be moved to any channel. The restriction being that each flow could only be assigned to the maximum number of channels available on that CM. The following variables were defined.

| | |
|---|---|
| Time | $t = 0, \ldots, T$ (with decisions made at $t = 1, \ldots, T$) |
| CMs | 1 … n |
| Channels | 1 … m |

The following inputs were required by the program.

| | |
|---|---|
| Incoming traffic (demand) | $a_{it}$ for $CM_i$ at time $t$ |
| Max number of channels per CM | $M_i$ for $CM_i$ |
| Capacity per unit time | $C_j$ for channel $j$ |

$a_{it}$ is defined as the traffic arriving in the interval [$t - 1, t$ ].

Following are the decision variables for the problem.

| | |
|---|---|
| $X_{ijt}$ | The amount of traffic, as a fraction of $j$ 's BW, from $CM_i$ to channel $j$ at time $t$ |
| $Z_{ijt}$ | = 1 if $CM_i$ is mapped to channel $j$ at time $t$ ; = 0 otherwise |
| $e_{it}$ | The amount of unsatisfied demand at $CM_i$, time $t$ |

If the variable $Z_{ijt}$ is 1, it indicates that the channel is being moved and data can't be sent at time $t$ , but that it can be sent at time $t + 1$.

The overriding goal is to fully utilize the available bandwidth to the extent to which it is demanded by all flows. It is therefore desired to minimize the amount of unsatisfied demand, for all CMs, at the end time T, as indicated in Figure 5.2 below.

OPT = Min $\sum_{i=1}^{n} e_{iT}$

Subject to:

$\forall_{i,t}$ : $\sum_{j=1}^{m} Z_{ijt} \leq M_i$ (1)

$\forall_{i,j,t}$ : $X_{ijt} \leq Z_{ijt}$ (2)

$\forall_{i,j,t}$ : $X_{ijt} \leq Z_{ijt-1}$ (3)

$\forall_{j,t}$ : $\sum_{i=1}^{n} X_{ijt} \leq 1$ (4)

$\forall_{i,t}$ : $e_{it} = e_{it-1} + a_{it} - \sum_{j=1}^{m} X_{ijt-1} \quad C_j$ (5)

$\forall_{i,j,t}$ : $0 \leq X_{ijt} \leq 1$ (6)

$\forall_{i,j,t}$ : $Z_{ijt} \in \{ 0, 1 \}$ (7)

$\forall_{i,t}$ : $e_{it} \geq 0$ (8)

Figure 5.2: Optimal Offline Algorithm.

Condition (1) states that for every CM the number of channels mapped must be less than or equal to the maximum number of channels the CM can tune. Condition (2) states that the percentage of traffic allocated (X) must be less than or equal to Z, which is zero if not mapped, or 1 if the CM can send on this channel. Condition (3) indicates that we can only send on this channel if it was mapped to this CM in the previous timeslot.

Condition (4) ensures that the sum of the allocations for all the CMs on this channel can't exceed 100% of the channel. Condition (5) states that the current unsatisfied demand is equal to the previous unsatisfied demand plus the new demand minus the amount sent (the fraction of the channel used times the capacity of the channel). Condition (6) keeps the fraction of the channel used between 0 and 100%. Condition (7) indicates that the CM is either mapped to this channel (1) or it is not (0). Condition (8) ensures that the amount of excess demand can't be negative.

Effectively the only demand the offline algorithm can't plan for is the demand from t0 to t1 since channels can't be switched until time t1. The possibility does exist for demand that is unsatisfied between t0 and t1 to be satisfied during later timeslots if switched to channels with excess bandwidth at that time, or other demands on those channels are later reduced. Therefore unsatisfied demand for the offline algorithm will be near zero unless the overall demand exceeds the total capacity of all channels.

### Online Remapping Algorithm

There were two questions to address in developing the online algorithm for this problem. The first question is when to remap the channels. The second question is how to remap the channels.

To answer the first question it was decided to monitor the amount of bandwidth that was being wasted. Wasted bandwidth is defined as the difference between the maximum possible data transferred and the actual data transferred. Maximum possible data transfer is that which could be achieved if there were no restrictions on which flows were assigned to which channels, within the constraints of the maximum number of channels per CM. The actual amount of data transferred is the max-min fair allocation given the current bonding group restrictions.

To calculate the maximum possible bandwidth allocation, $B_{max}$, a simple greedy bin packing algorithm, shown in Figure 5.3, was used to assign the current flow demands to the channels within the constraints of the channel capacity and the number of allowed channels per CM. The previously developed max-min program was used to calculate the actual bandwidth assignments, $B_{actual}$. The wasted bandwidth is $B_{max} - B_{actual}$. The wasted bandwidth is accumulated until it reaches a given threshold.

The threshold was related to the cost of switching the channels, in this case the maximum amount of throughput that would be lost during the time required to switch the channels. We used the assumption that the time to switch channels in the CM would be 500 ms and, to simplify the process, that all channels would be unavailable during this switching interval. We tested three different thresholds to evaluate the effect of the

threshold waiting time. The first threshold was equal to the amount of maximum data throughput lost on all channels during the 500 ms switching time. The second threshold was two times the throughput lost and the third threshold was three times the throughput lost. Figure 5.4 shows pseudo code for the main program loop that monitors the wasted bandwidth and determines when the threshold is reached and remapping is required.

```
/* fill the channels */
channels_full = 0;
ch_index = 0;
for (i=0; i<num_flows; i++)
{
     while (demand[i] > 0 &&
            !channels_full &&
            channels_per_cm[i] < cm_channels[i])
     {
          if (demand[i] <= remaining_capacity[ch_index])
          { /* remaining demand will fit in current channel */
               remaining_capacity[ch_index] -= demand[i];
               demand_assigned += demand[i];
               demand[i] = 0;
          }

          else
          { /* remaining demand must be split */
               demand_assigned += remaining_capacity[ch_index];
               demand[i] -= remaining_capacity[ch_index];
               remaining_capacity[ch_index] = 0;
               channels_per_cm[i]++;
          }

          if (remaining_capacity[ch_index] <= 0)
          {
               ch_index++;
               if (ch_index >= num_channels)
                    channels_full = 1;
          }
     } /* end while */
}  /* end for */
```

Figure 5.3: Pseudo Code – find_max_allocation.

```
for ( each timestamp )
{
     /* get next list of demands */
     for (i=0; i<num_flows; i++)
          get flow_demands[i];

     /* update running weighted demand averages */
     for (i=0; i<num_flows; i++)
          if (average_demand[i] < 0)
               average_demand[i] = flow_demands[i];
          else
               average_demand[i] = 0.9 * average_demand[i] +
                                   0.1 * flow_demands[i];

     for (i=0; i<num_flows; i++)
          flow_demands[i] += unsat_demand[i];

     find_allocation();  /* find max-min fair allocation */

     bw_allocated = 0;
     for (i=0; i<num_flows; i++)
          unsat_demand[i] = 0.0;

     totalUnsatDemand = 0.0;
     for (i=0; i<num_flows; i++)
     {
          bw_allocated += flow[i];  /* actual amount allocated */
          unsat_demand[i] += flow_demands[i] - flow[i];
          totalUnsatDemand += unsat_demand[i];
     }

     max_allocated = find_max_allocation();
     wastedBW = max_allocated - bw_allocated;
     accumulatedWaste += wastedBW;

     if (accumulatedWaste > threshold)
     {
          remap();
          remap_init(); /* reset variables */
     }
}
```

Figure 5.4: Pseudo Code – When To Remap.

The second question deals with how to remap the channels, after the threshold is exceeded, to best improve the bandwidth utilization. Three remap strategies were tested. The first, remap1, was a simple approach that moved only one flow at a time. The approach determines the channel with the most unused bandwidth. The flow with the most unsatisfied demand is then determined. That flow is then moved to the channel with the most unused capacity. Figure 5.5 shows pseudo code for remap1.

A second remap function, remap2, was then developed to determine an optimal map by using an integer linear program that uses the average demand of each flow to calculate the maximum possible throughput given the channel capacities and the maximum number of channels per CM. The remap2 function was used only to provide an optimal approach for comparison purposes, since implementation in real time would be too slow. The integer linear program for the remap2 function is shown in Figure 5.6

```
/* find channel with most unused BW */
for (i=0; i<num_channels; i++)
      if (accumulatedLostCH[i] > max_lost)
      {
            max_lost = accumulatedLostCH[i];
            channel = i;
      }

if (max_lost == 0)
      return;

/* find flow with most unsatisfied demand */
for (i=0; i<num_flows; i++)
      if (unsat_demand[i] > max_demand)
      {
            max_demand = unsat_demand[i];
            flow = i;
      }

if (max_demand == 0)
      return;

/* find first channel that flow has access to */
old_channel = num_channels + 1;
for (i=0; i<num_channels; i++)
      if (channel_map[flow][i] == 1)
      {
            old_channel = i;
            channel_map[flow][i] = 0;
            break;
      }

if (old_channel == num_channels + 1)
      printf("ERROR: Channel access problem\n");
else /* switch to new channel */
      channel_map[flow][channel] = 1;

      make_flow_network();
```

Figure 5.5: Pseudo Code – remap1.

The following variables were defined for remap2:

| | |
|---|---|
| CMs | 1 … n |
| Channels | 1 … m |
| $X_{ij}$ | Percent of channel $j$ used by $CM_i$ |
| $C_j$ | Capacity of channel $j$ |
| $d_i$ | Average demand of $CM_i$ |
| $Z_{ij}$ | Mapping of $CM_i$ to channel $j$ ; 0 if not mapped, 1 if mapped |
| $M_i$ | Maximum number of channels on $CM_i$ |

Max $\sum_{i=1,j=1}^{n,m} X_{ij}\ C_j$

Subject to:

$\forall_i$: $\sum_{j=1}^{m} X_{ij}\, C_j \leq d_i$ (1)

$\forall_j$: $\sum_{i=1}^{n} X_{ij} \leq 1$ (2)

$\forall_{i,j}$: $X_{ij} \leq Z_{ij}$ (3)

$\forall_i$: $\sum_{j=1}^{m} Z_{ij} = M_i$ (4)

$\forall_{i,j}$: $0 \leq X_{ij} \leq 1$ (5)

$\forall_{i,j}$: $Z_{ij} \in \{ 0, 1 \}$ (6)

Figure 5.6: Integer Linear Program – remap2.

The goal of remapping the channels is to maximize the total channel capacity used. Condition (1) indicates that each CM will use at most the current average demand. Condition (2) ensures that all CMs on a given channel will not exceed 100% of the capacity of the channel. Condition (3) states that the percentage of traffic allocated (X) must be less than or equal to Z, which is zero if not mapped, or 1 if the CM can send on this channel. Condition (4) limits the number of channels for each CM to the maximum allowed. Condition (5) ensures each channel is 0 – 100% utilized. Condition (6) is the channel map variable that indicates whether $CM_i$ is mapped (1), or not mapped (0) to channel $j$.

The final remap function, remap3, was developed to provide a simple approximation of the remap2 LP function. The average demand of each flow was again used as the basis of the remapping. To build a new channel map, based on current average demands, a simple greedy bin-packing algorithm was used. Each flow was packed, in turn, into the existing channels using, if necessary, multiple channels per flow up to the maximum number of channels in the CM. As each channel is filled the algorithm moves to the next channel until all channels are filled, or all flows are covered. After the packing is complete, the algorithm round robin assigns additional channels to each flow, as needed, to reach the maximum channels in the CM.

```
/* pack the channels */
ch_index = 0;
clear_channel_map();
for (i=0; i<num_flows; i++)
{
     while (average_demand[i] > 0 &&
            channels_per_cm[i] < cm_channels[i])
     {
          if (average_demand[i] <= remaining_capacity[ch_index])
          { /* remaining demand will fit in current channel */
               remaining_capacity[ch_index] -= average_demand[i];
               demand_assigned += average_demand[i];
               average_demand[i] = 0;
               channel_map[i][ch_index] = 1;
          }
          else
          { /* remaining demand must be split */
               demand_assigned += remaining_capacity[ch_index];
               average_demand[i] -= remaining_capacity[ch_index];
               remaining_capacity[ch_index] = 0;
               channels_per_cm[i]++;
               channel_map[i][ch_index] = 1;
          }
          if (remaining_capacity[ch_index] <= 0)
          {
               ch_index++;
               if (ch_index >= num_channels)
                    ch_index = 0;
          }
     } /* end while */
}  /* end for */

/* round robin assign un-used channel mappings */
ch_index = 0;
for (i=0; i<num_flows; i++)
     while (channels_per_cm[i] < cm_channels[i])
     {
          if (channel_map[i][ch_index] != 1)
          {
               channel_map[i][ch_index] = 1;
               channels_per_cm[i]++;
          }
          ch_index++;
          if (ch_index >= num_channels)
               ch_index = 0;
     }
```

Figure 5.7: Pseudo Code – remap3.

### 5.3 Results

Seven scenarios were initially built to test the operation of the remapping algorithm. A combination of scenarios was used where some would overload the total capacity of the channels and some would not. All scenarios used four channels each with a capacity of 40 Mbps and ten flows with varying demands. The initial channel map placed flows 1-5 on channels 1 and 2 and flows 6–10 on channels 3 and 4. All scenarios had two hundred 500ms timeslots.

Scenario 1 places a demand of 4 Mbps on every flow during all timeslots. This provides a load of less than 25% of the total bandwidth. Scenario 2 places a demand of 10 Mbps on every flow during all timeslots, 120% of the total capacity. Scenario 3 places a demand of 10 Mbps on flows 1–5 and no demand on flows 6–10 for two consecutive timeslots, then the 10 Mbps are switched to flows 6 – 10 and flows 1–5 have no demand for two timeslots. This cycle repeats through the run, using approximately 65% of total bandwidth.

Scenarios 4 and 5 each use two different repeating patterns of varying demands across the ten flows. The total bandwidth used for both scenarios is approximately 80% of capacity. Scenario 6 uses a pattern that repeats all of the demand patterns from scenarios 2-5 producing a total demand of approximately 85% of capacity. Scenario 7 uses a different repeating pattern from scenario 6, providing a total demand exceeding the total capacity by approximately 400 Mbps (2.5%).

After using these first seven scenarios to test under and overloaded conditions, two additional scenarios, 8 and 9, were added to test the edge conditions. For these two scenarios there are 4 channels, each with a 10 Mbps capacity, and 4 flows. Each flow can access 2 channels. The demands on flows 0 and 1 are 15Mbps and are initially assigned to channels 0 and 1. The demands on flows 2 and 3 are 5 Mbps and are initially assigned to channels 2 and 3. For scenario 8 these demands are constant throughout the run. This provides a total demand exactly equal to the total capacity of the channels.

Initially the mapping will cause an overload on channels 0 and 1, while there is an excess capacity on channels 2 and 3. After 5 timeslots the wasted bandwidth will accumulate to the X1 threshold and cause a remapping. At this point the flows will be remapped such that the demand will fully load all four channels and it will not be possible to draw down the unsatisfied demand accumulated during those initial 5 timeslots. Scenario 9 is initially setup identical to scenario8 but after those 5 timeslots, immediately after the channels are remapped, the flow 3 demand switches to 15 Mbps and the flow 1 demand switches to 5 Mbps, once again causing an imbalance in the demands.

Figure 5.8 shows the threshold graph for scenario 9 using a X1 threshold and remap3. The wasted bandwidth grows until the threshold is exceeded. A remapping occurs when the wasted bandwidth level drops to zero, which occurs twice in this run, before the second remapping keeps the wasted bandwidth at 0 for the remainder of the run.

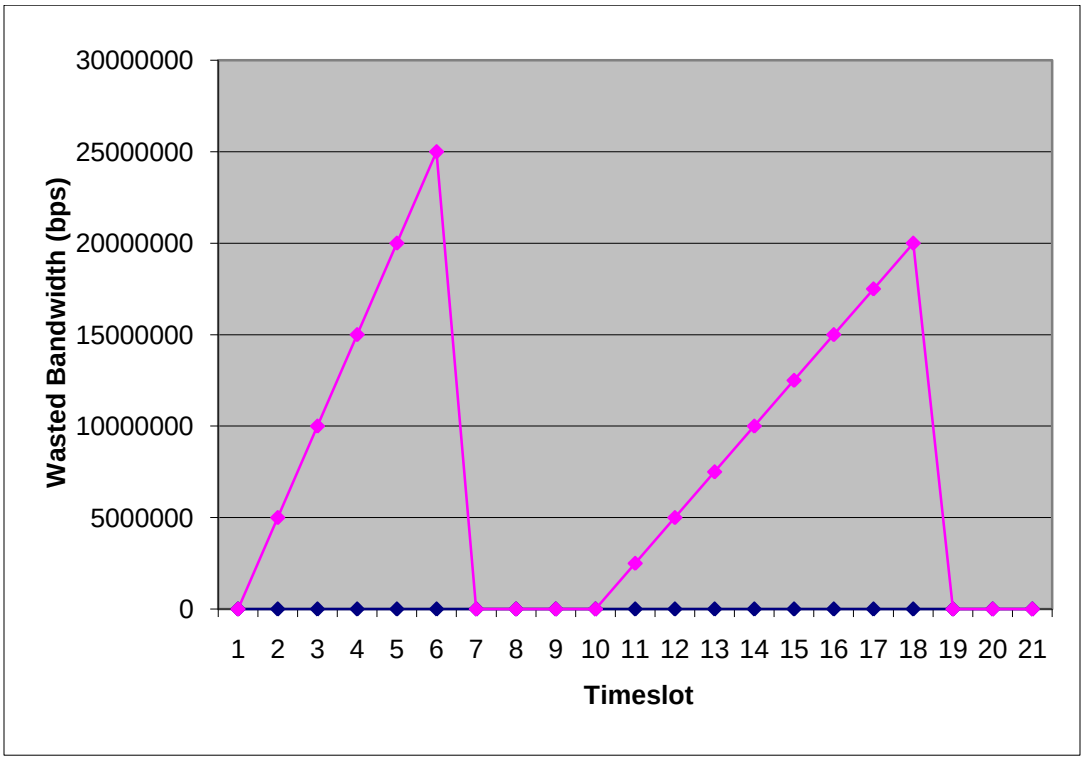


Figure 5.8: Scenario 9 Threshold Graph.

Max allocation is 40 Mbps (20 Mb per 500 ms timeslot). Actual allocation given the channel map is 30 Mbps (15 Mb per timeslot). This gives us 20Mb - 15Mb (5Mb) per timeslot of buildup towards the threshold. The X1 threshold is the amount of data lost during a 500 ms channel switch or 20Mb. Therefore after four timeslots we are at the limit, the fifth timeslot pushes us over it and causes a remap. With the new map the wasted bandwidth again increases, causing an additional remap, after another 12 timeslots, Figure 5.9 shows the original mapping and the two remaps.

| Initial Map | First remap | Second remap |
|---|---|---|
| 1 1 0 0 | 1 1 0 0 | 1 1 0 0 |
| 1 1 0 0 | 0 1 1 0 | 0 1 1 0 |
| 0 0 1 1 | 1 0 0 1 | 1 0 1 0 |
| 0 0 1 1 | 0 1 0 1 | 0 0 1 1 |

Figure 5.9: Scenario 9 Remaps.

Looking at the first remap, given the greedy bin-packing approach this is as expected. The 15M from flow 0 goes 10M to channel 0 and 5M to channel 1. The 15M for flow 1 goes 5M to channel 1 and 10M to channel 2. The 5M for both flows 2 and 3 go to channel 3. We then round robin to fill each flow out to their 2 channel limits, with flow 2 going to channel 0 and flow 3 going to channel 1.

All nine scenarios were run both without a remapping algorithm and with remap1, remap2 and remap3 and with all three thresholds. Appendix A shows the unsatisfied demand, throughput and total number of remaps for all nine scenarios for each of the three threshold levels and all three remap strategies.

Table 5.1 shows the unsatisfied demand for each of the nine scenarios for no remapping, our remap3 online algorithm, and for the optimal result produced by the offline algorithm. It is readily apparent that the use of remapping improves the servicing of the offered demand, and therefore the bandwidth utilization, except in cases where the overall demand exceeds the total available bandwidth.

It should be noted that there is one case, scenario 9, where remapping does not cause an improvement in performance. In this scenario the run begins with flows 0 and 1, each with 15 Mbps of demand, both on channels 0 and 1. Flows 2 and 3, both with 5 Mbps of demand, are on channels 2 and 3. This provides 30 Mbps of demand on 20 Mbps of channel capacity, and 10 Mbps of demand on 20 Mbps of channel capacity. This situation only lasts for 5 timeslots, 2.5 seconds. After that time the demands change such that there is 40 Mbps of demand spread across 40 Mbps of channel capacity. Therefore the case with no remapping is able to satisfy all demand requests except during those first 5 timeslots. In the remap case, after those 5 timeslots the mapping is changed such that the load is balanced evenly, just as the demands are changing. It requires 12 timeslots for another remapping to correct the situation and rebalance the loads once again. Therefore the remap algorithm loses 17 timeslots rather than the 5 timeslots in the no remapping case.

Our remapping algorithm produces results very close to optimal, the exceptions being the edge conditions of scenarios 8 and 9. It can be seen, as expected, that remapping provides little benefit if the available bandwidth is exceeded, since as much data will be transmitted as is possible. In this case the remapping threshold is never exceeded, since the maximum possible allocation is no better than the actual allocation, and the remap function is never called.

| | No remap | Remap3 | Optimal |
|---|---|---|---|
| Scenario 1 | 0 | 0 | 0 |
| Scenario 2 | 4 000 000 000 | 4 000 000 000 | 4 000 000 000 |
| Scenario 3 | 20 000 000 | 0 | 0 |
| Scenario 4 | 10 000 000 | 0 | 0 |
| Scenario 5 | 2 000 000 | 0 | 0 |
| Scenario 6 | 2 000 000 | 0 | 0 |
| Scenario 7 | 210 000 000 | 200 000 000 | 200 000 000 |
| Scenario 8 | 1 000 000 000 | 25 000 000 | 2 500 000 |
| Scenario 9 | 25 000 000 | 45 000 000 | 2 500 000 |

Table 5.1: Unsatisfied Demand.

Since all unsatisfied demand is carried forward to the next timeslot in the simulation program the indication is that all demand can always be serviced if the total capacity of all channels is not exceeded. Queue sizes will determine the ability to achieve this in practice, but this result should be possible if there is no packet drop.

Our remap3 implementation therefore provides an online algorithm to dynamically remap channels such that the total system bandwidth can be more efficiently utilized in attempting to satisfy changing demands.

# CHAPTER SIX

# CONCLUDING REMARKS

In this dissertation, we have studied the problem of managing downstream bandwidth in a DOCSIS 3.0 based cable network that supports bonded channels. Our work assumes that max-min fair is the desired allocation objective and that the essence of the well studied GPS based fair queuing model is appropriate. We developed a flow network to model the channel bonded network. Using this model we developed an offline algorithm that calculated the max-min fair allocations based on the flow demands, channel capacity and current channel map.

We modified the standard DRR and SCFQ scheduling disciplines to operate in a channel bonded network and coded the implementation into our *ns* DOCSIS module. Simulations were run using these channel bonded DRR and SCFQ implementations and were compared against the max-min results from the flow network implementation. Our analysis suggests the following:

- A simple round robin based packet scheduler, such as one based on DRR, might encounter situations where it can not preserve the correct (as defined by max-min fairness) service ordering.
- While it is possible to develop more complex round robin based schedulers that potentially offer an appropriate compromise between algorithm complexity and consistent fairness, we have shown that time stamped based algorithms, such as SCFQ, naturally avoid any complexity due to bonded channels.

- Finally, we have provided simulation-based evidence suggesting that multichannel packet scheduling algorithms must maintain state not only on a per flow basis, but also on a per channel basis. In all experiments that we have performed, we never found a scenario where channel scheduled SCFQ failed to converge to the max-min allocation.

We developed and implemented an online algorithm to remap the flows, real time, to maximize the utilization of the available system bandwidth. An offline algorithm was implemented using an integer linear program to determine the optimal bandwidth utilization. A competitive analysis approach was used to quantitatively analyze the effectiveness of our online algorithm. We showed that remapping will improve greatly the bandwidth utilization.

Future efforts in the resource allocation area are numerous. Further study of multichannel packet scheduling could focus on worst case fairness and delay bounds, and computational complexity issues. The broadest effort would be to add upstream scheduling to the system. This would be an extensive effort that could involve far more variables than the downstream problem.

In the online remapping area several enhancements can be made. To make the implementation more applicable to practical networks several additions could be made to the algorithm. Fixed queues could be added to the simulation to assess the effects of packet drops. Rather than allowing any flow to be mapped to any channel, remapping could be confined to only existing bonding groups, or to newly created bonding groups.

The algorithm could be modified to operate on channel modules where switching channels would only lose bandwidth on the channel switched, or the channels within the switched module. Perhaps the most interesting future direction is to develop an optimization approach that must not only minimize unsatisfied demand, but also one that maintains fairness objectives.

# APPENDICES

# Appendix A

## Threshold and Remapping Data

| Threshold | Remap1 | Remap2 | Remap3 |
|---|---|---|---|
| X1 | 0 | 0 | 0 |
| X2 | 0 | 0 | 0 |
| X3 | 0 | 0 | 0 |

Table A.1: Scenario 1 Unsatisfied Demand.

| Threshold | Remap1 | Remap2 | Remap3 |
|---|---|---|---|
| X1 | 4000000000 | 4000000000 | 4000000000 |
| X2 | 4000000000 | 4000000000 | 4000000000 |
| X3 | 4000000000 | 4000000000 | 4000000000 |

Table A.2: Scenario 2 Unsatisfied Demand.

| Threshold | Remap1 | Remap2 | Remap3 |
|---|---|---|---|
| X1 | 0 | 0 | 0 |
| X2 | 0 | 0 | 0 |
| X3 | 0 | 0 | 0 |

Table A.3: Scenario 3 Unsatisfied Demand.

| Threshold | Remap1 | Remap2 | Remap3 |
|---|---|---|---|
| X1 | 0 | 0 | 0 |
| X2 | 0 | 0 | 0 |
| X3 | 0 | 0 | 0 |

Table A.4: Scenario 4 Unsatisfied Demand.

| Threshold | Remap1 | Remap2 | Remap3 |
|---|---|---|---|
| X1 | 0 | 0 | 0 |
| X2 | 0 | 0 | 0 |
| X3 | 0 | 0 | 0 |

Table A.5: Scenario 5 Unsatisfied Demand.

| Threshold | Remap1 | Remap2 | Remap3 |
|---|---|---|---|
| X1 | 0 | 0 | 0 |
| X2 | 0 | 0 | 0 |
| X3 | 0 | 0 | 0 |

Table A.6: Scenario 6 Unsatisfied Demand.

| Threshold | Remap1 | Remap2 | Remap3 |
|---|---|---|---|
| X1 | 200000000 | 200000000 | 200000000 |
| X2 | 200000000 | 200000000 | 200000000 |
| X3 | 200000000 | 200000000 | 200000000 |

Table A.7: Scenario 7 Unsatisfied Demand.

| Threshold | Remap1 | Remap2 | Remap3 |
|---|---|---|---|
| X1 | 25000000 | 25000000 | 25000000 |
| X2 | 45000000 | 45000000 | 45000000 |
| X3 | 65000000 | 65000000 | 65000000 |

Table A.8: Scenario 8 Unsatisfied Demand.

| Threshold | Remap1 | Remap2 | Remap3 |
|---|---|---|---|
| X1 | 502500000 | 47500000 | 45000000 |
| X2 | 25000000 | 25000000 | 25000000 |
| X3 | 25000000 | 25000000 | 25000000 |

Table A.9: Scenario 9 Unsatisfied Demand.

| Threshold | Remap1 | Remap2 | Remap3 |
|---|---|---|---|
| X1 | 8000000000 | 8000000000 | 8000000000 |
| X2 | 8000000000 | 8000000000 | 8000000000 |
| X3 | 8000000000 | 8000000000 | 8000000000 |

Table A.10: Scenario 1 Throughput.

| Threshold | Remap1 | Remap2 | Remap3 |
|---|---|---|---|
| X1 | 16000000000 | 16000000000 | 16000000000 |
| X2 | 16000000000 | 16000000000 | 16000000000 |
| X3 | 16000000000 | 16000000000 | 16000000000 |

Table A.11: Scenario 2 Throughput.

| Threshold | Remap1 | Remap2 | Remap3 |
|---|---|---|---|
| X1 | 9999982336 | 9999981056 | 9999981056 |
| X2 | 9999984640 | 9999981568 | 9999981568 |
| X3 | 9999985664 | 9999982592 | 9999982592 |

Table A.12: Scenario 3 Throughput.

| Threshold | Remap1 | Remap2 | Remap3 |
|---|---|---|---|
| X1 | 12000007424 | 12000029568 | 12000029568 |
| X2 | 12000021504 | 12000028672 | 12000028672 |
| X3 | 12000015360 | 12000026624 | 12000026624 |

Table A.13: Scenario 4 Throughput.

| Threshold | Remap1 | Remap2 | Remap3 |
|---|---|---|---|
| X1 | 12000004608 | 12000020480 | 12000020480 |
| X2 | 11998006272 | 12000010240 | 12000010240 |
| X3 | 11998007296 | 12000006144 | 12000006144 |

Table A.14: Scenario 5 Throughput.

| Threshold | Remap1 | Remap2 | Remap3 |
|---|---|---|---|
| X1 | 13500008320 | 13500000000 | 13500012544 |
| X2 | 13500004352 | 13500013568 | 13500013568 |
| X3 | 13500000256 | 13500003328 | 13500008448 |

Table A.15: Scenario 6 Throughput.

| Threshold | Remap1 | Remap2 | Remap3 |
|---|---|---|---|
| X1 | 15400000512 | 15400002560 | 15400002560 |
| X2 | 15389999104 | 15400002560 | 15400002560 |
| X3 | 15400000512 | 15400000512 | 15400000512 |

Table A.16: Scenario 7 Throughput.

| Threshold | Remap1 | Remap2 | Remap3 |
|---|---|---|---|
| X1 | 3975000000 | 3975000000 | 3975000000 |
| X2 | 3955000000 | 3955000000 | 3955000000 |
| X3 | 3935000000 | 3935000000 | 3935000000 |

Table A.17: Scenario 8 Throughput.

| Threshold | Remap1 | Remap2 | Remap3 |
|---|---|---|---|
| X1 | 3497499424 | 3952500000 | 3955000000 |
| X2 | 3975000000 | 3975000000 | 3975000000 |
| X3 | 3975000000 | 3975000000 | 3975000000 |

Table A.18: Scenario 9 Throughput.

| Threshold | Remap1 | Remap2 | Remap3 |
|---|---|---|---|
| X1 | 0 | 0 | 0 |
| X2 | 0 | 0 | 0 |
| X3 | 0 | 0 | 0 |

Table A.19: Scenario 1 Number of Remaps.

| Threshold | Remap1 | Remap2 | Remap3 |
|---|---|---|---|
| X1 | 0 | 0 | 0 |
| X2 | 0 | 0 | 0 |
| X3 | 0 | 0 | 0 |

Table A.20: Scenario 2 Number of Remaps.

| Threshold | Remap1 | Remap2 | Remap3 |
|---|---|---|---|
| X1 | 2 | 1 | 1 |
| X2 | 2 | 1 | 1 |
| X3 | 2 | 1 | 1 |

Table A.21: Scenario 3 Number of Remaps.

| Threshold | Remap1 | Remap2 | Remap3 |
|---|---|---|---|
| X1 | 4 | 1 | 1 |
| X2 | 2 | 1 | 1 |
| X3 | 2 | 1 | 1 |

Table A.22: Scenario 4 Number of Remaps.

| Threshold | Remap1 | Remap2 | Remap3 |
|---|---|---|---|
| X1 | 3 | 1 | 1 |
| X2 | 2 | 1 | 1 |
| X3 | 1 | 1 | 1 |

Table A.23: Scenario 5 Number of Remaps.

| Threshold | Remap1 | Remap2 | Remap3 |
|---|---|---|---|
| X1 | 2 | 5 | 1 |
| X2 | 2 | 1 | 1 |
| X3 | 2 | 2 | 1 |

Table A.24: Scenario 6 Number of Remaps.

| Threshold | Remap1 | Remap2 | Remap3 |
|---|---|---|---|
| X1 | 2 | 1 | 1 |
| X2 | 1 | 1 | 1 |
| X3 | 1 | 1 | 1 |

Table A.25: Scenario 7 Number of Remaps.

| Threshold | Remap1 | Remap2 | Remap3 |
|---|---|---|---|
| X1 | 1 | 1 | 1 |
| X2 | 1 | 1 | 1 |
| X3 | 1 | 1 | 1 |

Table A.26: Scenario 8 Number of Remaps.

| Threshold | Remap1 | Remap2 | Remap3 |
|---|---|---|---|
| X1 | 22 | 2 | 2 |
| X2 | 0 | 0 | 0 |
| X3 | 0 | 0 | 0 |

Table A.27: Scenario 9 Number of Remaps.

# REFERENCES


[1] http://www.cablelabs.com/specifications/CM-SP-MULPIv3.0-I11-091002.pdf

[2] Abramson N., *The Aloha System – Another Alternative for Computer Communications*, Proceedings of the Fall Joint Computer Conference, Jan 1970.

[3] Ahuja R., Magnanti T. and Orlin J., Network Flows: Theory, Algorithms and Applications, Prentice Hall, 1993.

[4] Ahuja V., *Routing and Flow Control in Systems Network Architecture*, IBM Systems Journal, vol. 18, no. 2, pp. 298-314, 1979.

[5] Bennett C. and Zhang H. *$WF^2Q$ : Worst-case Fair Weighted Fair Queuing*. Proceedings IEEE INFOCOM'96

[6] Borodin A., and El-Yaniv R., Online Computation and Competitive Analysis, Cambridge University Press, 1998.

[7] Bushmitch D., Mukherjee S., Narayanan S., Ratty M. and Shi Q. *Supporting MPEG Video Transport on DOCSIS-Compliant Cable Networks*. IEEE Journal on Selected Areas in Communications. Vol.18, Issue 9, September 2000, pp. 1581-1596

[8] Cho S., Kim J. and Park S. *Performance Evaluation of the DOCSIS 1.1 MAC Protocol According to the Structure of a MAP Message,* http://viplab.hanyang.ac.kr/pdf/5-20.pdf, 2001.

[9] Comer D., Computer Networks and Internets, Prentice Hall, 2009.

[10] Cormen T., Leiserson C., Rivest R. and Stein C., Introduction to Algorithms, McGraw-Hill, second edition, 2001.

[11] Davies D., *The Control of Congestion in Packet-Switching Networks*, IEEE Transactions on Communications, vol. COM-20, June 1972.

[12] Droubi M., Idirene N. and Chen C. *Dynamic bandwidth allocation for the HFC DOCSIS MAC protocol*. Ninth International Conference on Computer Communications and Networks. Las Vegas, NV, October 2000, pp. 54-60.

[13] Ford L. and Fulkerson D., *Maximal Flow through a Network*, Canadian Journal of Mathematics, 1956

[14] Gallo G., Grigoriadis M. and Tarjan R., *A fast parametric maximum flow algorithm and applications*, SIAM Journal of Computing, vol. 18, no. 1, pp. 30-55, 1989.

[15] Gerla M. and Kleinrock L., *Flow Control: A Comparative Survey*, IEEE/ACM Transactions on Communications, 28(4), pp. 553–574, 1980.

[16] Gray J., *Network Services in Systems Network Architecture*, IEEE Transactions on Communications, vol. COM-25, pp. 104-116, 1977.

[17] Golestani S., *A Self-Clocked Fair Queueing Scheme for Broadband Applications*. Proceedings IEEE INFOCOMM 94, pp. 636-646, June 1994.

[18] Golmie N., Mouveaux F. and Su D. *Differentiated Services over Cable Networks*. Proceedings of Global Telecommunications Conference GlobeCom99, December 1999.

[19] Hayden H., Voice Flow Control in integrated Packet Networks, Master's Thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering, Cambridge, MA, 1981.

[20] Hawa M. and Petr D. *Quality of Service Scheduling in Cable and Broadband Wireless Access Systems*. Tenth IEEE International Workshop on Quality of Service, May 2002.

[21] Heyaime-Duverge C. and Prabhu V. *Traffic-Based Bandwidth Allocation for DOCSIS Cable Networks*. Proceedings, Eleventh International Conference on Computer Communications and Networks, October 2002.

[22] Huston G. Internet Performance Survival Guide QoS Strategies for Multiservice Networks. John Wiley & Sons, Inc., 2000.

[23] Jaffe J., *Bottleneck Flow Control*, IEEE Transactions on Communications, 29(7), pp. 954-962, 1981.

[24] Ju H. and Liao W. *Adaptive Scheduling in DOCSIS-based CATV Networks*. Proceedings, Eleventh International Conference on Computer Communications and Networks, October 2002.

[25] Karlin A., Manasse M., Rudolph L. and Sleator D. *Competitive snoopy caching*. Algorithmica, 3:79-119, 1988.

[26] Keshav S. An Engineering Approach to Computer Networking: ATM Networks, the Internet, and the Telephone Network. Addison-Wesley, 1997.

[27] Kuo W., Kumar S. and Kuo C.-C. *Bandwidth Allocation and Traffic Scheduling for DOCSIS Systems with QoS Support*. Global Telecommunications Conference, GLOBECOM Vol. 2, November 2002.

[28] Kuo W., Kumar S. and Kuo C.-C. *Improved Priority Access, Bandwidth Allocation and Traffic Scheduling for DOCSIS Cable Networks*. IEEE Transactions on Broadcasting, Vol. 49, No. 4, December 2003.

[29] Lambert J., Van Houdt B. and Blondia C. *Dimensioning the contention channel of DOCSIS cable modem networks*. Networking 2005: 4th International Ifip-Tc6 Networking Conference, May 2005.

[30] Lin Y., Huang C. and Yin W. *Allocation and Scheduling Algorithms for IEEE 802.14 and MCNS in Hybrid Fiber Coaxial Networks*, IEEE Trans. Broadcasting. vol.44, pp. 427-435, December 1998.

[31] Megiddo N., *Optimal Flows in Networks with multiple Sources and Sinks*, Mathematical Programming, 7:97-107, 1974.

[32] Namman N. and Rom R. *Analysis of Transmission Scheduling with Packet Fragmentation*. Discrete Mathematics and Theoretical Computer Science 4, 2001.

[33] Namman N. and Rom R. *Packet Scheduling with Fragmentation*. Proceedings, Twenty-first Annual Joint Conference of the IEEE Computer and Communications Societies INFOCOMM 2002, June 2002.

[34] Namman N. and Rom R. *Scheduling Constant Bit Rate Flows in Data over Cable Networks*. Proceedings, Seventh International Symposium on Computers and Communications, ISCC 2002.

[35] Namman N. and Rom R. *Bandwidth Scheduling for Multi-Channel Packet Cable Telephony*. Proceedings, Eleventh International Conference on Computer Communications and Networks, October 2002.

[36] Parekh A. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks. PhD dissertation, Massachusetts Institute of Technology, February 1992

[37] Rabbat R. and Siu K. *QoS Support for Integrated Services over CATV,* IEEE Communications, Vol. 37, No. 1, pp. 64-68 January 1999.

[38] Sdralia V., Smythe C., Tzerefos P. and Cvetkovic S. *Performance Characterisation of the MCNS DOCSIS 1.0 CATV Protocol with Prioritised First Come First Served Scheduling,* IEEE Transactions on Broadcasting, vol. 45, No.2, pp. 196-205, June 1999.

[39] Shreedhar M. and Varghese G. *Efficient Fair Queuing Using Deficit Round Robin,* IEEE/ACM Transactions on Networking, Vol. 4, No. 3, pp. 375-385 June 1996.

[40] Sleator D. and Tarjan R. *Amortized Efficiency of List Update and Paging Rules,* Communication of the ACM, 28:202-208, 1985.

[41] Tzerefos P., Sdralia V., Smythe C. and Cvetkovic S. *Delivery of Low Bit Rate Isochronous Streams Over the DOCSIS 1.0 Cabel Television Protocol,* IEEE Transactions on Broadcasting, vol. 45, No.2, pp. 206-214, June 1999.

[42] Varghese G. Network Algorithmics An Interdisciplinary Approach to Designing Fast Networked Devices. Elsevier, 2005

[43] Yin W. and Lin Y. *Statistically Optimized Minislot Allocation for Initial and Collision Resolution in Hybrid Fiber Coaxial Networks*. IEEE Journal on Selected Areas in Communications, Vol. 18, No. 9, September 2000.

[44] Yin W., Wu C. and Lin Y. *Two-Phase Minislot Scheduling Algorithm for HFC QoS Services Provisioning*, IEICE Trans. on Communications, Vol. E85-B, No. 3, 582-593, Mar. 2002.