Maximizing Bandwidth Utilization in Downstream DOCSIS 3.0 Channel Bonded Networks

Scott Moser, Brian C. Dean, Jim Martin, James M. Westall
School of Computing
Clemson University
Clemson, South Carolina
{smoser, bcdean, jim.martin, westall}@clemson.edu

Abstract—The addition of bonding groups in DOCSIS 3.0 allowed the bandwidth to be increased to the end user. Max-min fair packet scheduling provides for fairness among those users, but due to the restrictions of the bonding groups, available bandwidth may go unused. This research investigates the dynamic remapping of the flows to allow more effective utilization of the system bandwidth. An online algorithm was developed, using the concept of competitive analysis, to monitor unutilized bandwidth and current demands and make adjustments in the flow-to-channel mappings to improve the utilization of available system bandwidth. We show that our algorithm provides near optimal results and greatly improves bandwidth utilization over the case without remapping.

Keywords—Channel Bonding; Competitive Analysis; DOCSIS; Online Algorithms; Scheduling

I. INTRODUCTION

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. The newest version of this standard, DOCSIS 3.0 [1], adds channel bonding capability which allows two or more channels to be combined to provide a single logical pipe. Downstream and upstream bandwidth is managed centrally by the Cable Modem Termination System (CMTS). The set of channels over which the CMTS schedules the information of a service flow (either downstream or upstream) 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 the 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.

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 large unused bandwidth 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 Fig. 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.

![Figure 1. Unbalanced Example](image-url)

This work was supported in part by Cisco Systems, Inc. via Cisco Research Award DOCSIS 3.0 Scheduling Algorithms and in part by NSF through awards CCF-0845593 and ECCS-0948132.
In this example channels 1 and 2 are fully loaded and flows 2 – 9 have unsatisfied demand. Channel 3 is less than 50% loaded and channel 4 is unused. In this case shifting some of the flows 2 – 9 to bonding groups 2 and 3 can provide all flows with their full demands.

Borodin and El-Yaniv [2] 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 our requirement is satisfying 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.

II. BACKGROUND

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, \( s(t), \ i \leq t \leq m \), 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 [3] 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 [4] 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

\[
A(I) \leq c \cdot OPT(I)
\]

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 [5] 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-competitive.

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.

III. 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.

A. 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 \( 1 \ldots n \)
- Channels \( 1 \ldots m \)

The following inputs were required by the program.

- Incoming traffic (demand) \( a_{it} \) for CM \( i \) at time \( t \)
- Max channels per CM \( M_i \) for CM \( i \)
- Capacity per unit time \( C_j \) for channel \( j \)

\( a_{it} \) is 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 Fig. 2 below.

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 \( t_0 \) to \( t_1 \) since channels can’t be switched until time \( t_1 \). The possibility does exist for demand that is unsatisfied between \( t_0 \) and \( t_1 \) 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.

\[
\text{OPT} = \text{Min } \sum_{i=1}^{n} e_{iT}
\]

Subject to:

\[
\forall i,t: \quad \sum_{j=1}^{m} Z_{ijt} \leq M_i \quad \text{(1)}
\]

\[
\forall i,j,t: \quad X_{ijt} \leq Z_{ijt} \quad \text{(2)}
\]

\[
\forall i,j,t: \quad X_{ijt} \leq X_{ijt-1} \quad \text{(3)}
\]

\[
\forall i,t: \quad \sum_{j=1}^{m} X_{ijt} \leq 1 \quad \text{(4)}
\]

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

\[
\forall i,t: \quad 0 \leq X_{ijt} \leq 1 \quad \text{(6)}
\]

\[
\forall i,t: \quad Z_{ijt} \in \{0, 1\} \quad \text{(7)}
\]

\[
\forall i,t: \quad e_{it} \geq 0 \quad \text{(8)}
\]

Figure 2. Offline Algorithm
B. 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, $B_{\text{max}}$, a simple greedy bin packing algorithm 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 [6] was used to calculate the actual bandwidth assignments, $B_{\text{actual}}$. The wasted bandwidth is $B_{\text{max}} - B_{\text{actual}}$. 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. Appendix 1 shows pseudo code for the main program loop that monitors the wasted bandwidth and determines when the threshold is reached and remapping is required.

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.

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 following variables were defined for remap2:

<table>
<thead>
<tr>
<th>CMs</th>
<th>1 \ldots n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channels</td>
<td>1 \ldots m</td>
</tr>
<tr>
<td>$X_{ij}$</td>
<td>Percent of channel $j$ used by $CM_i$</td>
</tr>
<tr>
<td>$C_j$</td>
<td>Capacity of channel $j$</td>
</tr>
<tr>
<td>$d_i$</td>
<td>Average demand of $CM_i$</td>
</tr>
<tr>
<td>$Z_{ij}$</td>
<td>Mapping of $CM_i$ to channel $j$;</td>
</tr>
<tr>
<td>$M_i$</td>
<td>Maximum number of channels on $CM_i$</td>
</tr>
</tbody>
</table>

The integer linear program for the remap2 function is shown in Fig. 3.

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 $Z_j$, 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 algorithm. 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. Appendix 2 shows pseudo code for the remap3 algorithm.

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

Subject to:

\[
\forall \ i: \sum_{j=1}^{m} X_{ij} C_j \leq d_i \tag{1}
\]

\[
\forall \ i: \sum_{j=1}^{n} X_{ij} \leq 1 \tag{2}
\]

\[
\forall \ i,j: X_{ij} \leq Z_{ij} \tag{3}
\]

\[
\forall \ i: \sum_{j=1}^{m} Z_{ij} = M_i \tag{4}
\]

\[
\forall \ i,j: 0 \leq X_{ij} \leq 1 \tag{5}
\]

\[
\forall \ i,j: Z_{ij} \in \{0, 1\} \tag{6}
\]

Figure 3. remap2 Algorithm
IV. 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. 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 scenario 8 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.

Fig. 4 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.

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.

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 scenario 8 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.

Fig. 4 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.

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, Fig. 5 shows the original mapping and the two remaps.

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 packet from flow 1 and one packet from flow 2. Therefore, DRR will not provide max-min fair scheduling.

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 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.

<table>
<thead>
<tr>
<th>Initial Map</th>
<th>First remap</th>
<th>Second remap</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 0 0</td>
<td>1 1 0 0</td>
<td>1 1 0 0</td>
</tr>
<tr>
<td>1 1 0 0</td>
<td>0 1 1 0</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>1 0 0 1</td>
<td>1 0 1 0</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>0 1 0 1</td>
<td>0 0 1 1</td>
</tr>
</tbody>
</table>

Figure 5. Scenario 9 Remaps

![Scenario 9 Threshold Graph](image-url)
algorithm to dynamically remap channels such that the total possible allocation is no better than the actual allocation, and remapping threshold is never exceeded, since the maximum much data will be transmitted as is possible. In this case the little benefit if the available bandwidth.

Table I. Unsatisfied Demand (Mbps)

<table>
<thead>
<tr>
<th>Scenario</th>
<th>No remap</th>
<th>Remap3</th>
<th>Optimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>4000</td>
<td>4000</td>
<td>4000</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>210</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>25</td>
<td>2.5</td>
</tr>
<tr>
<td>9</td>
<td>25</td>
<td>45</td>
<td>2.5</td>
</tr>
</tbody>
</table>

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.

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.

V. Conclusions and Future Work

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 and that our online algorithm performs nearly as well as the optimal.

Possible future efforts will deal with making the implementation more applicable to practical networks. 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.

Acknowledgment

We acknowledge our sponsors at Cisco, in particular Alon Bernstein, in providing insight into the DOCSIS 3.0 scheduling problem.

References

Pseudo code; When to remap

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 */
    }
}
/* 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;
        }