Solving the Stalling Issue for Multicasting Video On-Demand

A Design Project Report Presented to the Engineering Division of the Graduate School of Cornell University in Partial Fulfillment of the Requirements for the Degree of Master of Engineering (Electrical)

by
Chaoping Deng
Project Advisor:Professor Aaron B. Wagner
Degree Date: August 2015

Abstract

Master of Electrical & Computer Engineering Program
Cornell University
Design Project Report

Project Title:
Solving the Stalling Issue for Multicasting Video On-Demand

Author:
Chaoping Deng

Project Advisor:
Professor Aaron B. Wagner

Multicasting is a network information transmission technique which significantly reduces network traffic. Different from unicasting, for which a central server sends out information to different users separately, multicasting allows the server to distribute only one copy of the content to various clients. The broadcasting nature of multicasting ensures it to be able to send out the same amount of traffic over every link as if the information is transmitted to a single client. In a "video on demand' scenario where different users arrive in the network at different times to watch the same video, unicasting technique will resend the entire video content whenever a new user arrives, while multicasting will enable the newly-arrived user to reuse the cached data in the network used by the previous user.

There are various multicasting schemes developed over the years. There are two main types: those with servers that keep track of the full state information of clients by receiving feedbacks from them and those with servers that are completely ignorant of the client states. The former works the best with a small amount of users and the latter works the best for a large group of clients. It is possible to develop a scheme that will work well for both scenarios, by utilizing a built-in mechanism to select partial feedback based on the number of users in the network. Approximate Dynamic programming techniques has been used to tackle the problem this way.

Previous projects have successfully implemented the approximate DP algorithm described by the paper " Multicasting for wireless video on-demand" . In particular,past implementation and test have shown the approximate DP algorithm implemented performed well relative to unicast. This design project focuses on exploring ways of solving the stalling issue presented by the previous project, as well as providing a better testing environment by fixing existing bugs and providing better packet tracking techniques.

Executive Summary

The Video Aware Wireless Network has been an ongoing project focusing on implementing efficient multicasting algorithms that could serve multiple users concurrently. A video is broadcasted to different users who join the network at different times from different locations.Various multicast algorithms have been developed and compared with the unicast algorithm. The latest Approximate Dynamic Programming algorithm developed by VAWN has shown promising future in industry with its unique feature of combining the merits of both client-oblivious and feedback-based algorithms.

This design project is a continuation of the implementation of Approximate Dynamic Programming algorithm. The ADP algorithm developed by previous master students outperformed unicast algorithm when the number of client in the network is small.All the previous tests have been run against at most 5 concurrent users in the network. When the number of user reaches a relatively large number(>5),users start to experience the “stalling issue” ,which causes videos received for the clients starting to lag.This indeed weakens the incentive to use multicast over unicast when there are a lot of users in the network concurrently. This design project focuses on understanding the reason behind the stalling as well as developing a solution to the problem.

In summary, the cause of “stalling issue”, or throttling is determined as a server-side problem,which refers to that fact that the scheduler could not keep up issuing video segments to different users when the user number is large. A direct approach to the problem is to reduce the number of segments of a video so that the search space will be reduced as number of users increases. A more sophisticated approach involves approximating most needed packets and re-sending their adjacent packets as “larger segments”. This modification is made to the algorithm and profound results have been achieved.

The design project has also made tracking and monitoring packet information a lot easier by reconstructing packet so that each video segment is easily located and studied with the use of Wireshark.Original video segment as defined in the program,when transferring through wireless protocols, is divided into pieces. This has made it nontrivial to track each segment with a third-party monitor. The reconstructed packet, however, provides user with clear tracking information and timestamp.

Lastly, the design project has fixed multiple compiling bugs produced by the original code and altered several setup command so that the program is more user-friendly.

Table of Contents

1.Introduction

With the fast increasing demand of wireless service nowadays, streaming data becomes expensive since bandwidth is generally limited. Traditional data transmission method such as unicasting, namely, transferring data from a server to a user, is inefficient. This leads to incentives to develop better data distribution technology that could release the burden on wireless networks.

The Video Aware Wireless Networks project aims at developing highly efficient data transferring model by adopting multicasting. Traditional unicasting requires data being transferred from server to single users separately. Multicasting works on the fact that different users usually need the same piece of information in the network. In a multicast setting, this “common piece” could be used by different users at the same time, which is much more efficient than unicast. Furthermore , the VAWN project tackles the issue of asynchronous viewing, which refers to the fact that for video streaming different users will join the network at different time, the algorithm developed must have the ability to ensure newly joined users could also retrieve the information from the beginning of the video. The challenge of the project is to develop such an algorithm that could not only most efficiently serve different user in the illustrated way above, but also provide best user experience for watching the video.

Past projects have successfully implemented different multicasting algorithms in order to achieve such goal.On one hand, the client-oblivious algorithms such as pyramid coding, which requires no client feedback, works the best when the number of users in the network is big, on the other hand, the feedback-based algorithms which require client to send back feedback packets to the server to help it make decisions, work the best when the number of user in the network is relatively small. The latest developed ADP algorithm, by combining the advantages of client-oblivious and feedback-based algorithms, theoretically scales well in regardless of the number of users. It indeed provides satisfying result when the number of user in the network is small; however it failed to achieve high performance due to the “stalling issue” . In order to release its full potential in industrial application. This research paper provides solutions to the limitation of the ADP algorithm.

The paper will focus on the following sections:1. summary of previously implemented algorithms: overviews of previously implemented client-oblivious system such as Pyramid Coding and client-feedback system such as Earliest Needed First scheduling. Overviews of error-correction codes and a brief introduction of the ADP algorithms. 2.Explanation of the “stalling issue” ,the implementation of its solution and performance matrix of the newly modified code.3. Refined packet structure and methods to track each data segment.

2.Design Problem

2.1 Previous work

2.1.1 **Pyramid Coding**

One of the first multicasting schemes implemented by VAWN is Pyramid Coding, a client-oblivious multicast scheme that broadcasts data in a fixed order without any client feedback information. The video is divided into segments for transmitting, the segment size grows exponentially bigger as you proceed to later part of the video. Each segment is divided into packets which correspond to a base unit for transmission. The Pyramid scheduler takes turns broadcasting these segments into the network , a legacy example to demonstrate how pyramid coding is shown as below (figure 1). As illustrated in Dunn's paper about Pyramid Scheduler "a video is broken up into three segments, each twice as large as the previous. For each time step in the broadcast, one packet from each of the segments is broadcast. Within each segment, packets packet is sent and the process starts again from the first packet creating a circular pattern. As such, packets in earlier segments are broadcast more frequently than those in later segments as there are fewer packets in earlier segments. The lower portion of Figure 1 shows the transmission scheme for the example video. By looking across each row, you can see the carousel broadcast manner of each segment."

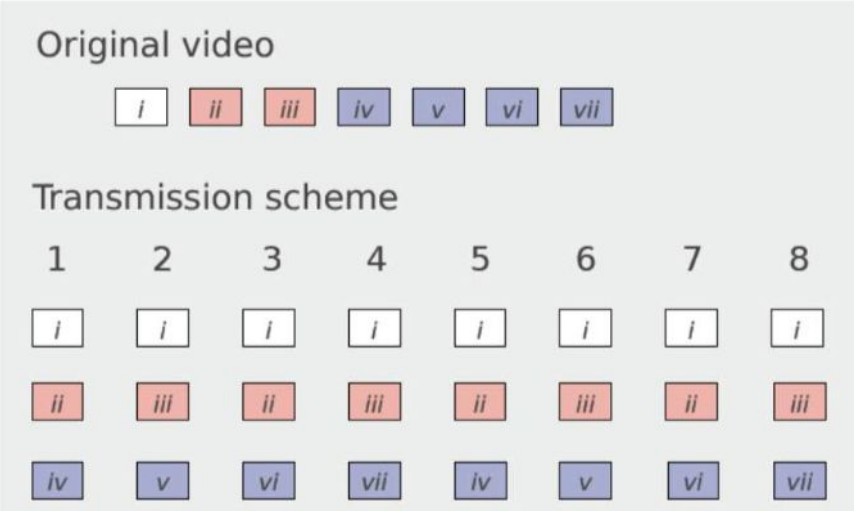

Figure 1. Pyramid Coding Example

The Pyramid Coding scheme takes the asynchronous arrival of clients into full consideration. Later clients will receive earlier segments much more frequently than later segments so that they can start immediately after they join the network. Pyramid Coding scheme, as demonstrated by previous MENG student, outperforms unicast when the number of clients is large. One key

drawback of pyramid coding is that its performance is weakened by packet loss. If a single packet is lost through the transmission process, a client has to wait for the scheduler to rotate a full transmission cycle in order to retrieve the missing packet. This increases the average stalling time of each client by a significant amount. In order to solve the related problem, LT error correction code is introduced in order to reduce losses.

2.1.2 LT Error Correcting Codes

Luby Transform Codes are error correcting codes that can reduce error rate by creating cross information between packets. LT codes allow partial redundant information to be embedded into each packet so that when some packets are lost through transmission, information contained in the lost packets could be reconstructed through the remaining packets so that the entire message could be received through the communication channel. LT correction code is in nature a simple algorithm based on exclusive or(xor) operation to encode and decode messages.The LT code process is presented as below:

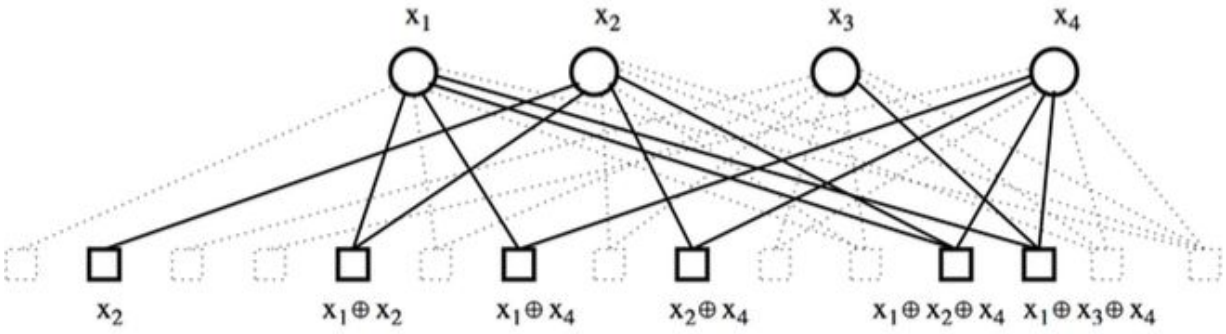

Figure 2. LT encoding

The encoding and decoding process is:

Encoding

1. Choose degree d from a distribution
2. Pick d neighbors uniformly at random.
3. Compute XOR of the d neighbors, for the value of the encoding symbol.

Decoding:

1. Identify code but of remaining degree
2. Recover corresponding input bit.
3. Update neighbors of this input bit.
4. Delete edges.

5. Repeat.

2.1.3 Earliest Needed First

Earliest Needed First scheduling algorithm, different from Pyramid Coding, is a client-feedback based scheme that requires the server to keep track of every client's state. Each client will constantly report back to the server about its current location so that the server could make decision about which client to server first. As suggested by its name, the Earliest Needed First algorithm will let its server send packet to the client with the least time playable left before stalling.

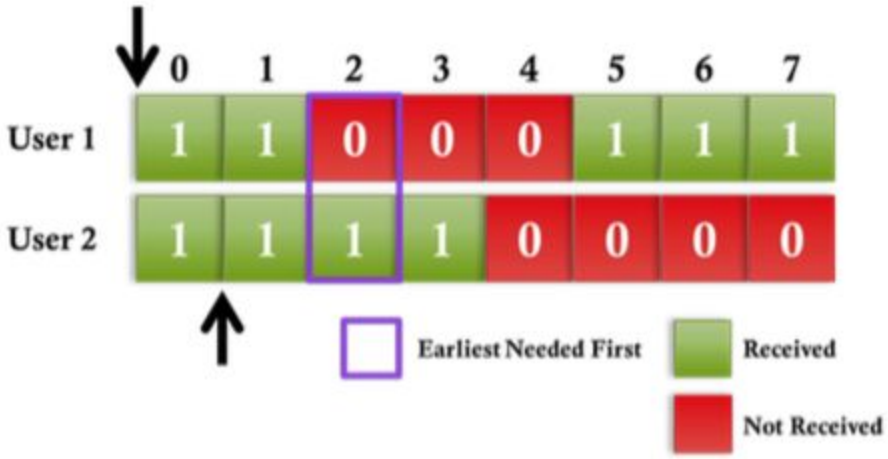

Figure 3. Earliest Needed First Example

As shown in the figure above, Packet 2 will be sent to user 1 for the next time slot.

Previous project has demonstrated a better performance than Pyramid Coding when the number of client is small, however, due to the fact that each client needs to send feedback to the server, the scheme faces a scaling problem when the number of client is large. Therefore ,developing a scheme that can work equally well in regardless of client number is crucial for the project.

2.1.4 Approximate Dynamic Programming

Model

The basic setup could be modeled as an MDP problem. For a video transmitting scenario, a total of I different users arrive in the network asynchronously.(The server-to-user broadcasting channel is a typical packet-erasure channel for which a user either receives the full information of an information segment at a certain time slot or receives nothing) A full-length video is divided into N different information segments and is sent out by a transmitter. We also assume that the information about a user arrives or departs is immediately available to the server and the feedback delivery time is ignored.We now define the following parameters:

L_i^j: *number of segments user i have at time j, taking value from -1 to N-1, where -1 represents the user is not in the system.*

C_i^j: *is the state of the server-to-user-i channel, taking value r or e. Users with value r will recevie the segment at the end of each time slot from the transmitter.*

$\boldsymbol{V}_i^j$: *is a vector of binary values of length N with $V_i^j(k) = 1$ representing for user i, he/she has received segment k at time j and $V_i^j(k) = 1$ representing he/she has not.*

A user i at the end of time slot j would be

$U_i^j = (L_i^j, C_i^j, \boldsymbol{V}_i^j)$

Policy Space

Assume we have J times slots and the policy space about which information to transmit for the next time slot could therefore be described as

$$\pi : \{\{-1,0,1,...N-1\} * \{e,r\} * \{0,1\}^N\}^I * \{0,1,2,....J-1\} -> \{1,2,3,4,....N\}^R$$

where R is the ratio between the transmission ratio of server transmission rate and video data rate, both in units of bit per second.

Value Function and DP recursion

Define the average stall time of all the users as

$$E(s) = 1/J \sum_{j=0}^{j-1} (1/I \sum_{I=1}^{I} \epsilon(U_i^j, U_i^{j+1}))$$

where we denote $\alpha(U_i^j, U_i^{j+1}) = 1/I \sum_{I=1}^{I} \epsilon(U_i^j, U_i^{j+1})$.

where J is the total number of time slots, I is the time number of users and

$\epsilon(U_i^j, U_i^{j+1}) = 1$ if $L_i^j \neq -1$ and $L_i^j = L_i^{j+1}$, $\epsilon(U_i^j, U_i^{j+1}) = 0$ otherwise.

We try to find a policy π^* such that E(s) is minimized .

We can define a input a(j) to represent which segment to distribute in the system at time j. The probability of U^j becoming U^{j+1} is $p(U^j, U^{j+1})$. We can setup the DP recursion as

$$\Phi(U^j, j) = min_{a(j) \to A} \Phi(U^j, U^{j+1}, a(j)) \quad (1)$$

$$\Phi(U^j, U^{j+1}, a(j)) = \sum \{p(U^j, U^{j+1})(\Phi(U^{j+1}, j+1) + \alpha(U_i^j, U_i^{j+1}))\} \quad (2)$$

with terminal cost $\Phi(U^J, J) = 0$ for all J.

Approximation

The current DP model has some problems addressing the computational complexity of equation (1) and (2) .Based on the model above, researchers have developed ways to approximate the original model. Here are some examples:To approximate (1), instead of allocating segments jointly for each time slot, we modify the original problem to a simpler version with the assumption that the server can only send out one information segment per time slot.

To compute (2), in order to find the optimal a^j at time j, we need to compute $\Phi(U^j, U^{j+1}, a(j))$ for each a^j, we can approximate the $\Phi(U^{j+1}$,j+1)term in (2) using $1/I \sum_{i=1}^{I} \Phi_i^*((U^{j+1}, J+1)$ where $\Phi_i^*((U^{j+1}, J+1)$ is the value function where we assume the user i is the only user in the system at the moment,and the transmission ratio is R=1, which means the server just keeps up with the video. This assumption is made because the coupling between different users upon next time slot is ignored.

3.Overall Project Requirement

The ADP scheduling scheme indeed performs well when the number of clients in the network is small. At most 5 clients have been tested against the scheme and great performance has been achieved comparing to unicast. However, as number of users increases, the “stalling issue” emerges where clients start to experience lags and distortion. Therefore, it is important to figure out why the problem occurs and provide a solution. In particular , we would like to find a way to modify the algorithm so that average stalling time for each user is significantly reduced.

4. Design Task and Solution Specifications

4.1 Throttling Caused by Router

The original ADP algorithm shows a large discrepancy between rate of multicast from client and server. With server broadcasting at a rate around 100Mbps and router’s ability to broadcast at 12Mbps without significant losses. At this point , unicast performs much better than multicast since large portion of data is lost through transmission.

Solution

Modification has been made to the code such that the maximum throughput from the server is limited to 9Mbps, so the limit set by the apple router we use is no longer an issue.

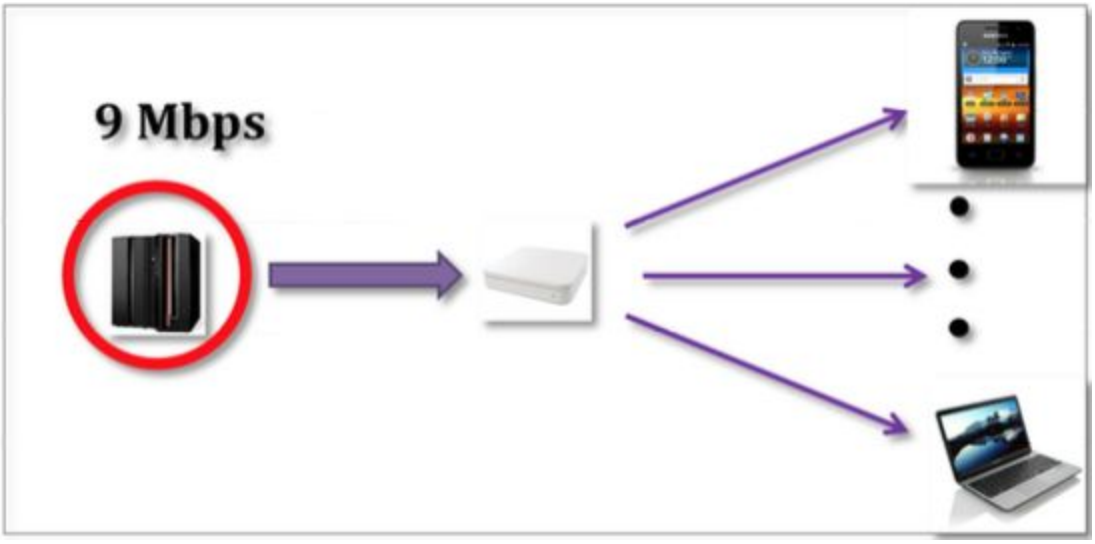

Figure 4. Solve Router Throttling

4.2 Video Stalling with 5 users

The ADP algorithm developed by the previous project has shown vast improvement over unicast when the client number is small. Ideally, when the client number is 5 the following graph shows how stall time should be looking like for unicast and multicast,respectively

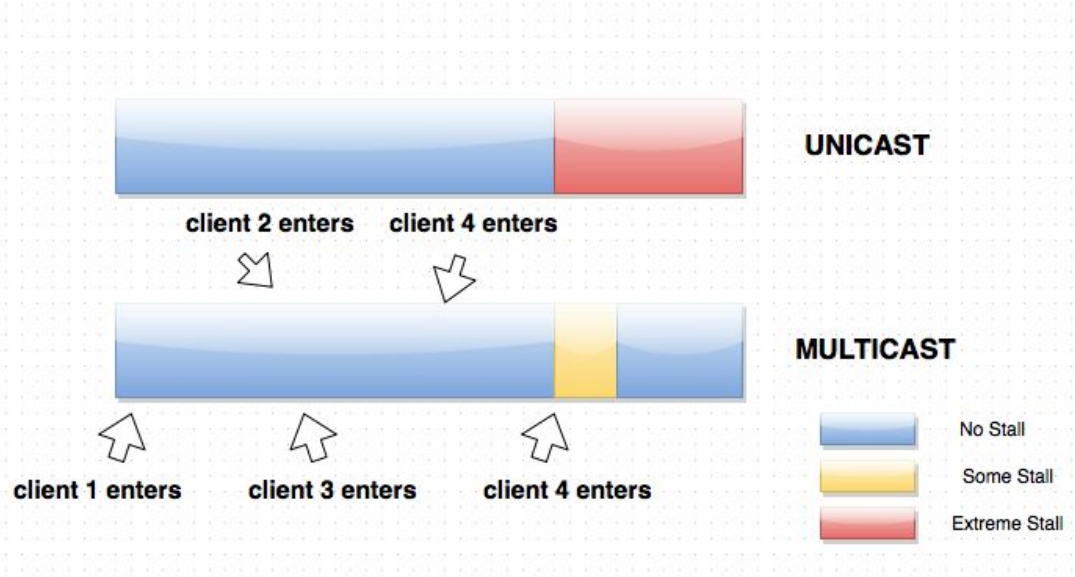

Figure 5. ADP with 5 users

As shown in the above graph, unicast performs well when the number of client is less than 5. However, after the fifth user enters, the video begins to experience extreme stalling and distortion since the bandwidth of the network is used up. Multicast is able to cope with 5 users and only experiences some stall when the fifth user enters. After a short period of time, no stall is experienced by any user.

4.3 Video Stalling with more than 5 users

When the 6th user enters the network,even under the multicast scheme, the user experiences extreme stalling. A visual demonstration is shown as below

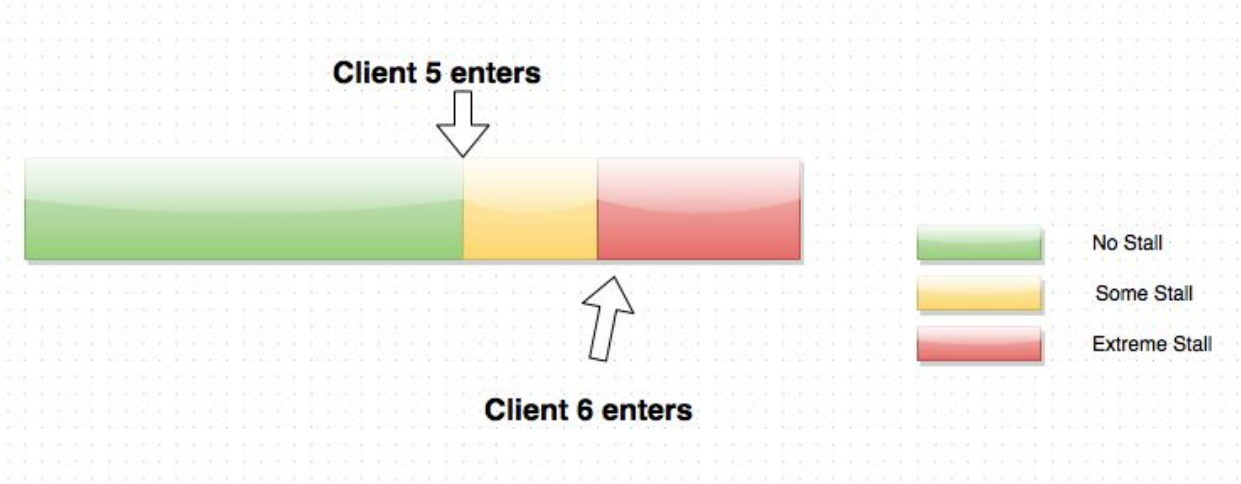

Figure 6. ADP with 6 users

Here are several possibilities that could cause the issue:

Router Malfunction
All the clients stall when the 6th user enters the network. This is not ideal since the ADP algorithm is desgined to scale well even when the client number is large. The causes of this problem can be on server side ,the router or the client side so we need a clear understanding about how this problem occurs. This problem still exists even when we run everything locally, which means that router is not a cause of the problem.

Client Malfunction
Client's jobs are receiving the information packets sent by the server and sending out feedback packets. If the stalling issue happens on the client side, given the algorithm works as it was described in the paper, the only way it could happen is that either packets sent from the server are lost or the mechanism of sending feedback packets from client is broken, resulting in either feedback packets not being able to reach the server , corrupted or not sent according to plan. Again, in order to get rid of the possiblity that feedback packets are lost or corrupted through the transmission process, we switched to local setting as well as setting up the client debug file that outputs all the the recived messages from the client. Unfortunately the stalling issue still occurs. Judging from the debugging files , it seems that the scheduler function well throughout the process. The previous reasoning of both client and router leads to the only possible cause of the problem, the server.

Server Malfunction

One intuitive assumption of why the server is not functioning as we expect is that the scheduler simply could not make scheduling decisions fast enough as client number increases. As shown in previous section, the full MDP model has a high computational complexity even with its approximate assumptions which were intended for complexity reduction. Therefore, with more concurrent clients in the network, we expect the scheduling decision to be made slowlier and eventually, affecting the video quality as packets could not be sent in time and the video is stalled.

In order to prove the assumption is indeed what happens to the scheme, a direct approach is to switch to a more powerful computer.With higher computational power ,the scheduler will be able to make scheduling decision faster and therefore, prevents the stalling issue from happening when client number reaches 6. Test has been done switching from a dual-core 1.8GHZ macbook air to a 2.7GHZ quad-core Imac, the stalling issue is resolved for 6 concurrent clients in the network. However, with more client joining in, the stalling issue happens again. There is an incentive to develop a way to modify the program such that we can cope with the increasing computational complexity of the scheduler.

5 Range of Solutions

5.1 SETUP

We would like to run all our experiments on an apple computer locally, the setup detail is in APPENDIX II, we assume clients arrive following a 4-second interval, with server running by itself for an initial 4 second before the first client joins.

5.2 Change Number of Segments

In ADP algorithm, a video is divided into smaller segments where each segment contains only one network packet. Consider the complexity of the Approximate DP algorithm which is $O(NU)$,where N is the number of segments in the video and U is the number of concurrent users in the network. A direct way to reduce the complexity is by linearly scaling down N, the number of segments dividing the video as client number increases . The following graph shows the improvement of changing packet size from 1400bytes to 3000 bytes and 5000bytes

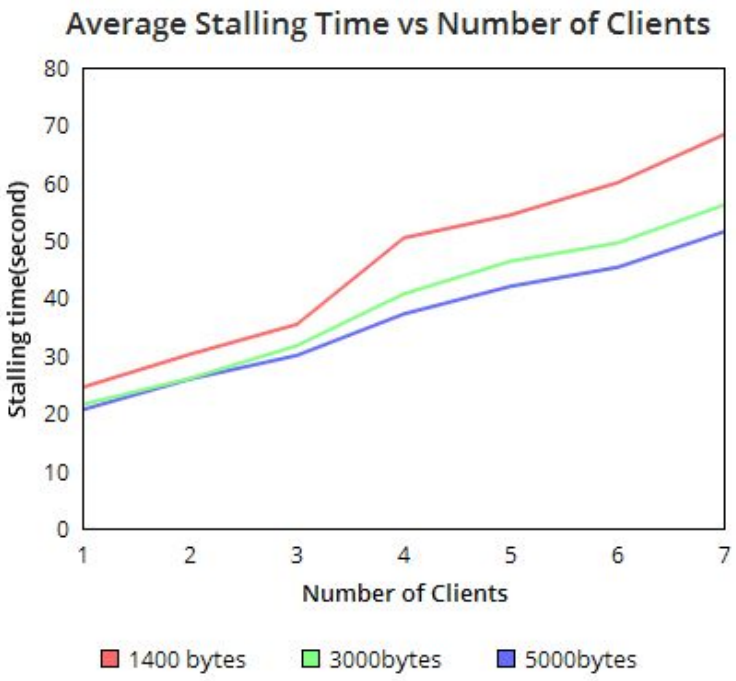

Figure 7. Stalling time vs Number of Clients

Discussion

The above graph shows a somewhat linear relationship between number of clients and stalling time for the setup. As we can notice, an increase in segment size indeed brings down the stalling time by a significant amount. Therefore, for a somewhat large video , we can manually increase the packet size for the stalling issue to be resolved.

However ,this brings up a new problem. Given a short video,as number of clients increases, it requires a huge segment size to fully avoid the stalling issue. Dividing the video into just several segments is something we do not want. Missing one of the segments will cause a huge distortion of video quality.

5.3 Segment Redefinition

Instead of treating one network packet as a segment, we would like to embed several packets into one segment as shown below:

Figure 8. A Multi Packet Segment

The segment is constructed by 5 network packets. The scheduler will function as if they were still the single-packet segment; however , whenever a packet is missing, all the packets which are defined to be in the same segment as the missing one will be resent by the server.
The benefit of this is,the concept of segment is only used when some packet is missing. The server will interpret the missing packet as a larger missing segment and try to resend the whole "segment" all over again. This will indeed potentially reduce the computation done by the scheduler as well as avoiding the scenario as stated in the previous section, where one segment only contains one packet,the packet gets too big so that a video is only divided into a couple of packets(segments).

5.3.1 Direct Approach

A direct approach is to modify the scheduler so that each time it encounters a missing packet, it will mark all the adjacent packets as missing. This way, we manually reconstructed a "larger segment"
from the original packets.

	packet n	packet n+1	packet n+2	packet n+3	packet n+4	packet n+5
client 1	1	missing	1	1	1	1
client 2	1	1	1	1	1	1
client 3	1	1	1	missing	1	1
client 4	1	1	1	1	1	1
client 5	1	1	1	1	1	1
client 6	1	1	missing	1	1	1

Table 1. State Matrix
All the red packets are marked as missing and needed to be considered as missing packet

The result of this approach is worse than the original algorithm. With a segment size of 5 packets, it provides an average of 75 seconds for 6 clients joining the network with 4 seconds interval, which is around 15 seconds worse than the average of 60 seconds for the original protocol.

The reason why this is not working as expected is that the modification to the scheduler does not really reduce the computational complexity and instead increases the burden of the server to send out additional packets. Many of these packets are not optimal to be sent since the missing packets are not really missing for the scheduler state matrix.

This also raises a practical concern: the scheduler makes its decision based on its estimation of the each client's state. Therefore whenever a packet is missing, it is possible for the server to not send out the exact missing packet as required by the client. Instead the server will send out a group of packets (group of 60) based on its stall calculation. The group is a mixture of requested packets of the newly arrived clients as well as approximated missing packets from the existing clients.

5.3.2 Refined Approach: Approximated Segment Reconstruction

As discussed above, the scheduler provides us a group of packets which are the approximated packets that are mostly needed by the network. It is ambiguous which exact packet within the selected group from the scheduler is corresponding to the missing packet. Therefore, we would like to develop an arbitrary " approximated segment reconstruction" mechanism to build larger segments.

Consider the same setting with 6 clients. The stalling issue occurs when the last cient joins the network, demanding early packets while the other clients are requesting later packets. This leads to burden on the scheduler , who is trying to come up with the optimal packet to serve everyone. When the scheduler can not keep up with the server , the stalling occurs. When all 6 clients join at the same time or reach the same progress, stalling issue is resolved because all clients demand similar packets. Our problem now is to make all 6 clients obtain the same progress before stalling occurs.

Consider the group of 60 packets sent out by the scheduler every time slot, in order to make all 6 clients reach the same progress as soon as possible (which will reduce scheduler's work later on) , as well as not creating extra burden for the scheduler, it is reasonable to follow the steps below:

1. Arbitrarily approximate most needed packets by choosing packets towards the beginning and end of the group of packets provided by the scheduler each time slot.
2. Treat those packets as missing portion of a larger segment size N, and resent all those segments(all the adjacent packets).

It is illustrated in the graph below:

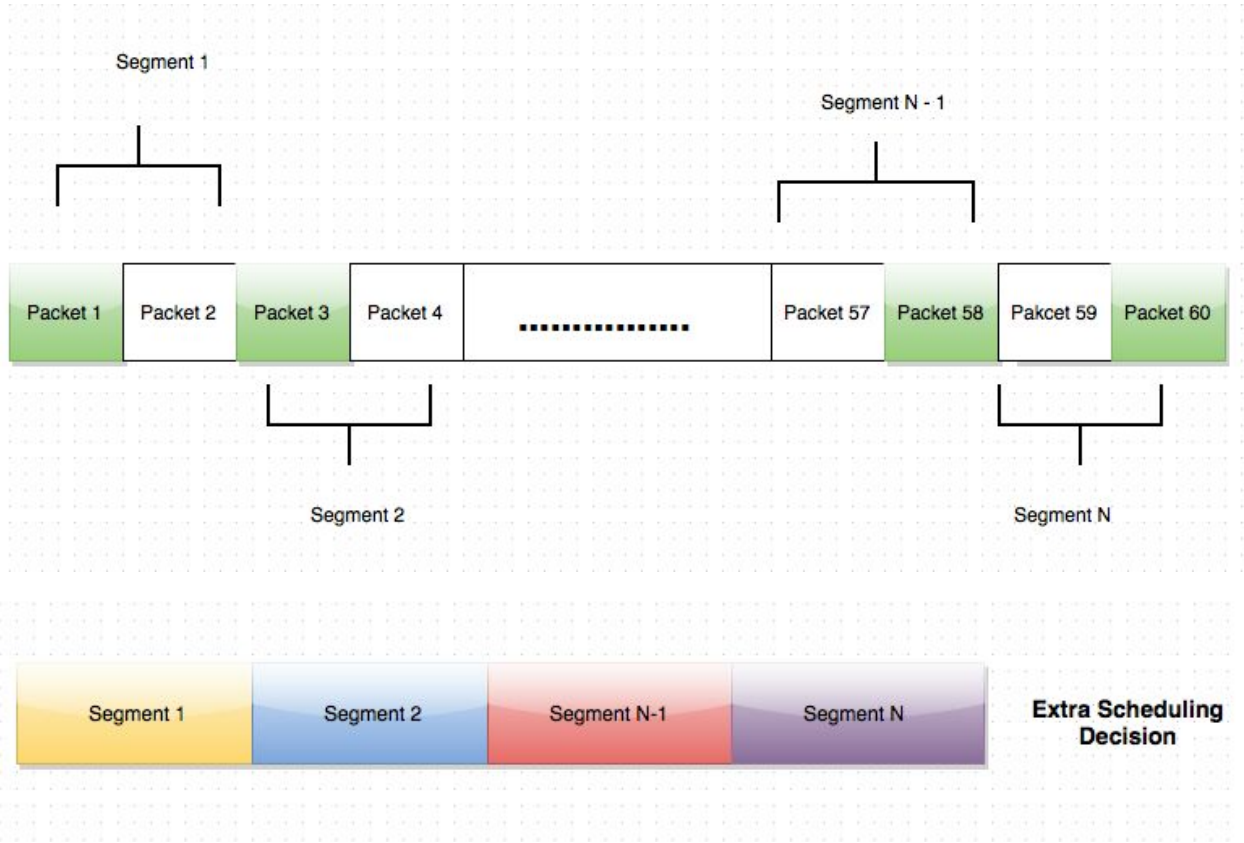

Figure 9. Appending Extra Scheduling Decision

Packet 1, 3, …… 58 60 are considered as “most needed packets” and the server will resent packet 1 2 3 4 57 58 59 60 , since in this case 2 packets are considered to be contained in one segment. We could also interpret it as the scheduler makes some extra scheduling decision by sending out segment 1,2, N-1 and N again.

Given only 6 clients, this approach successfully gets rid of the stalling issue for each user. However, with all 6 clients reaching the same progress there is no need for the server to still treat packets as larger segments. Therefore, a timer is deployed and after a certain period of time, ideally when all clients have reached similar progress, the above approximate segmentation mechanism will no longer be active.

In the test, a random sequence is chosen for the scheduling group . In particular, for each group, packets with index 1,2,3,50,49,47 are chosen as packets that are part of some larger segment and will be present along with their adjacent packets.A timer of 150 counts has been set for the effect to be active and segment size could be modified by user. Given the upper limit of 10 clients, we have tested 7 concurrent clients who join in order.

The following graph shows its comparison against original scheduler.

*number in bracket refers to timeslots during which the mechanism is active

Figure 10. Performance Curve

Discussion

We have seen improvement as we apply the approximate segment construction technique as introduced above. There are several points to notice:

1. There is an optimal segment size that could provide us with the best performance. There is a tradeoff between speed boost provided by this technique as discussed above and the added computational complexity for reconstructing a new scheduling group. Among all 3 segment size tested, a segment of 5 packets provides us with the best performance.
2. The “approximate segmentation” technique scales well especially when the number of clients in the system is large. As number of clients exceeds 5, the new mechanism provides steadier growth of total stalling time for all clients in the network.The one with the best parameters beat the original protocol by almost 10 seconds when client number reaches 7.

3. The timer is used to keep the mechanism active until all clients have reached similar progress. The timer value choice, similar to the sequence choice as defined above, is arbitrary. We do not want a large timer, because if the mechanism is still active when all clients reach similar states, it is a waste of bandwidth to resend all the packets all over again. We do not want a small timer neither, since we want to make sure the timer will be effective before all clients have reached similar progress.

6.Conclusion

We have generally solved the stalling issue by providing two methods: direct segment reduction method and approximate segmentation method. It is possible to improve the latter one with better sequence choice as well as timer value choice. Experiment results have shown that the second method provides us with better scaling when number of concurrent clients in the network is large.

7.Future improvement

One way to improve the scheme is to ask scheduler to keep track of all the missing packets and feedback an array containing all the indexes of these packets. Given that for each time slot up to only 60 scheduling decision is made, it is important to establish a formula to return indexes of the most important missing packets (not packets for new clients) and let the server send their adjacent packets all over again. It would provide a more efficient scheme than sending out random packets towards the beginning and end of the scheduling packet array.

Another way to reduce the computational complexity is to modify the scaling laws to do a cheaper "stall" calculation for each user. That requires us to choose □ $\sqrt{U}$ who are close to stalling and therefore reduces the computational complexity from $O(NU)$ to O(N $\sqrt{U}$)

Finally, we can change the packet and timer value to find the best combination that could produce the optimal performance curve.

Appendix I: A fast way to track packets

We would like to track packet information effectively. However, whenever a packet is sent through internet. HTTP protocol tends to break down overly large segment. A network packet generally contains 1412 bytes ,so a 1400 payload size for the packet is typically a safe limit to not let the packet break. It is important to reconstruct each packet and make it easy to be tracked in case it breaks.

A newly reconstructed packet is as following

```
struct Packet {

        uint32_t feedbackClient; // Client ID number of client we want feedback from
        uint32_t packetNumber; // Number of packet in chunk data has come from
        uint32_t time_stamp;
        uint8_t payload[maxPayloadSize]; // The segment's data

        void hton();
        void ntoh();
    };
```

In order to track it , setup Wireshark and record network traffic

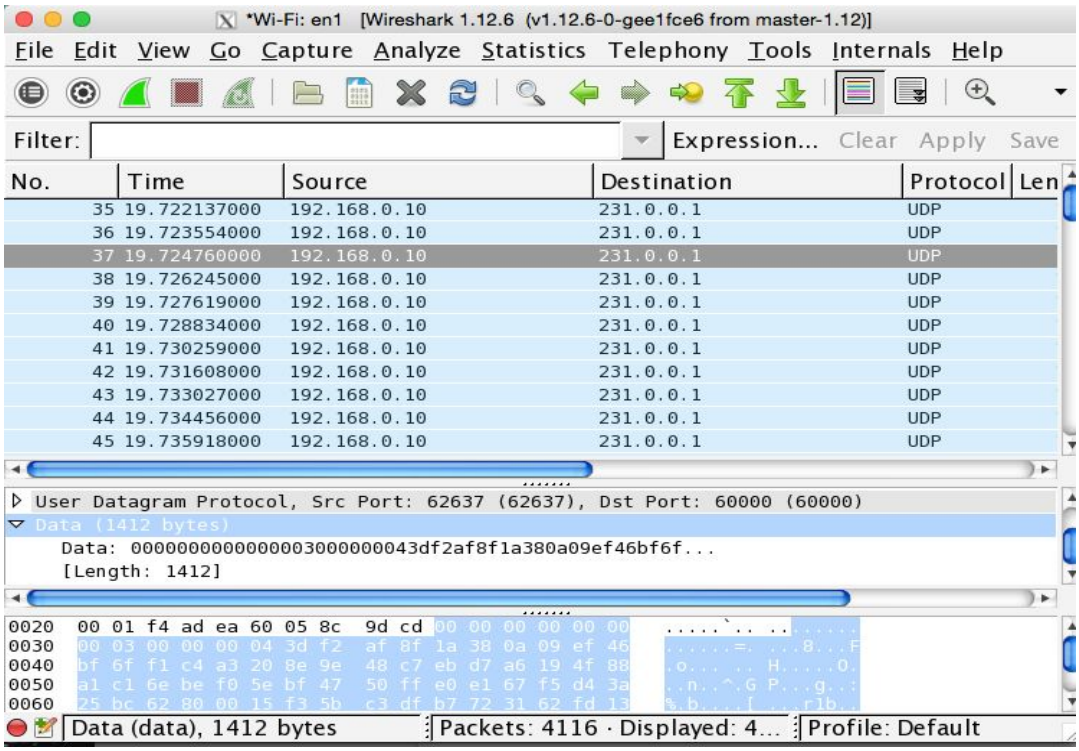

Figure 11. Wireshark Console Overview

In order to track highlighted packet- packet no.37, we would like to look into its Data section. In order to retrieve this specific packet by packet number as defined in the packet structure. Simply type:

data[8:4] ==00:00:00:04

8 means packet number starts after the 8th byte and 4 refers to the size of the packet number (4bytes)

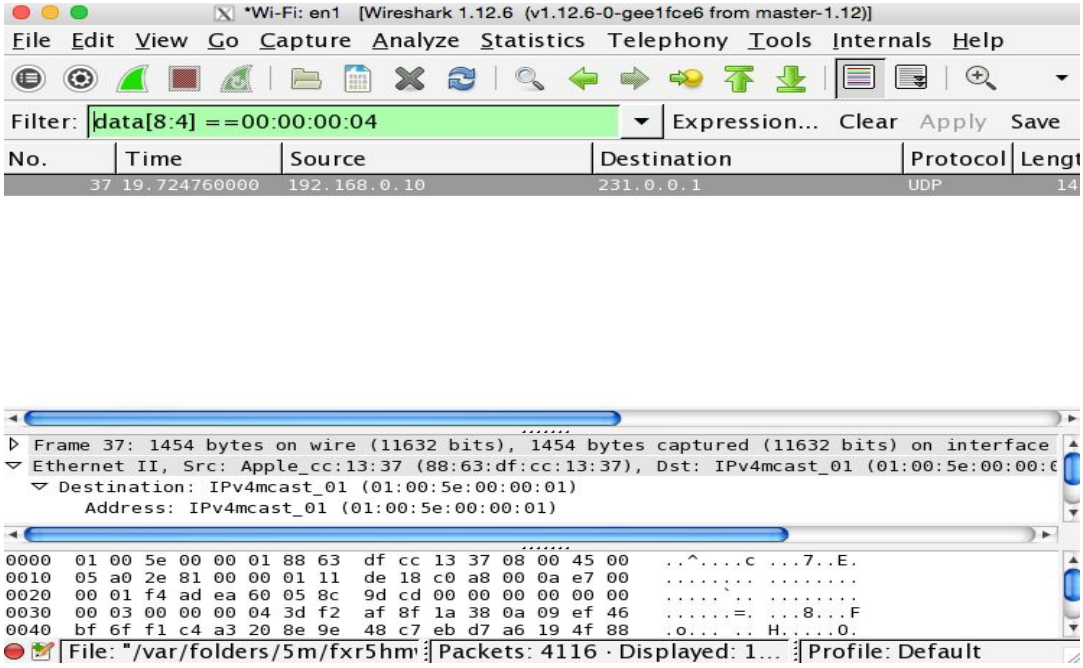

Figure 12. Track a single packet

Appendix II:Setup

Detailed Setup guide for windows and VLC player is covered in reports from previous years. Here is a quick setup guide for apple computer:

1.Compile the program

Open up Terminal in mac, go to directory and type “make all”.

2.Run server

Open up a separate console, go under vawn directory and follow the format
./server <scheduling_type(0:earliestNeededfirst,1:Ratelimited)> <schedulerdebug(boolean)> <Segmentsize(int)>
For example
For example ./ server 1 true 5
It means: run ratelimited with scheduler debug on and set approximate segmentation size to be 5 packets. Set it to be 0 if no segmentation mechanism is used.

3.Run Client

Open up a separate console, go under vawn directory and follow the format
./client <hostname> <port> <clientDebug(boolean)> <FeedbackDebug(boolean)> > data.ts
For example, ./client 192.168.0.10 54321 true true > data4.ts which means we would like to run client listening to broadcasting IP address 192.168.0.10 through port 54321. The client debug and feedback debug are both turned on and all the data is output to data.ts.
There is no need to change 54321 since it is a default port number we use for the program.

4.To find out about broadcasting address

type:
ifconfig | grep "inet" | grep -v 127.0.0.1
In this case, for example it returns line:
inet 192.168.0.10 netmask 0xffffff00 broadcast 192.168.0.255
wehre 192.168.0.10 is the address client needs to listen to.

A successful connection is shown as below
Server

```
ServerMulticastConnection initialized to "231.0.0.1" on port 60000, with a bitra
te throttled to about 1250000 B/s
ServerMulticastConnection initialized a client
Server initialized a client
Removed user 1
```

Client

```
 success!
MulticastConnection initialized to "231.0.0.1" on port 60000
My client id is: 1
The received bitrate size is: 38016

ScheduleDecoder initialized with 14593 packets of size 1400 bytes, except for th
e last which is 596 bytes
Finished receiving everything!
```

5.Setup wireshark

Simply download the program and type wireshark in the console. Make sure X11 is properly installed.

References

1.Md. Saifur Rahman & and Aaron B. *Wagner Multicasting for Wireless Video-on-Demand,* 2013 ,Cornell University
2.Aditya Ramesh, *Multicasting for Asynchronous Video-on-Demand*,2013,Cornell University
3.David Dunn,*Multicasting Video on-Demand*,2013,Cornell University
4.Christopher James Halabi, *Pyramid Coding for Video-On-Demand*, 2011 Cornell University