Analysis of the "Additive Increase Multiplicative Decrease" Model for Congestion Avoidance

Math 610: Projects in Applied Mathematics, Spring 2017

Analysis of the "Additive Increase Multiplicative Decrease" Model for Congestion Avoidance

John Gaylan
Surabhi Maheshwari
Andrew Whitener

March 14, 2017

Abstract

As technology advances, network congestion avoidance becomes increasingly more important. For this project, we attempted to analyze and optimize the Additive Increase, Multiplicative Decrease (AIMD) algorithm to gather data for our partner company. We created our own AIMD models and ran simulations to find and compare relevant data.

1 Project Background

Our project is primarily based on internet traffic control, specifically congestion avoidance in networks. It has been proposed by *Grier Forensics*.

Grier Forensics is a digital forensics consulting firm led by Mr. Jonathan Grier. Besides providing consulting services to clients, they also research, study and publish new forensics methods. Based on the information provided in the company's website, their clients include: Federal Reserve Bank of New York, DARPA, CUNY, and MIT's Lincoln Laboratory.

The project proposed by our partner company is relatively open ended. We were provided with the description of the company and one research paper to start with, titled 'Analysis of the Increase and Decrease, Algorithms for Congestion Avoidance in Computer Networks' by Dah-Ming Chiu and Raj Jain [1].

The Chiu-Jain paper discusses all possible algebraic algorithms for congestion avoidance, namely: Additive Increase, Additive Decrease; Additive Increase, Multiplicative Decrease; Multiplicative Increase, Additive Decrease; and Multiplicative Increase, Multiplicative Decrease.

From the paper's emphasis, we knew that we were expected to focus on the method of Additive Increase, Multiplicative Decrease algorithms, a congestion avoidance scheme emphasized in detail in the paper. Our primary focus thus far has been to work on AIMD and create a workable code using *MATLAB* for the same. Once the basic structure of the code has been fabricated, we try to incorporate more complex ideas into it, such as automation, allowing for dynamic values of X_{Goal}, and many other factors and variables.

AIMD Algorithm: The AIMD algorithm suggested by Chiu and Jain uses a variable to increase each user's allocation when the network capacity has not been reached. When the network capacity overloads, a variable attempts to restore each user's allocation to below that capacity (see Equation 1).

(Equ. 1)

$$x_i(t+1) = \begin{cases} a_I + x_i(t) & \text{if } y(t) = 0, \text{ Increase} \\ b_D * x_i(t) & \text{if } y(t) = 1, \text{ Decrease} \end{cases}$$

where:

t = the current time value
$x_i(t)$ = the i-th user's transmission rate at time t
a_I = the additive increase parameter
b_D = the multiplicative decrease parameter

Efficiency: One of the best ways to gauge how well the AIMD algorithm functions is the *efficiency characteristics* of the code. In an ideal scenario, the load stays on the efficiency line (the network capacity) of the graph for loads being used by different users. Anything below the line is underloaded while anything above it is overloaded (see Equation 2).

(Equ. 2)

$$X(t) = \sum x_i(t)$$

$$X(t) > X_{goal} = overload$$

$$X(t) < X_{goal} = underload$$

where:

$x_i(t)$ = the i-th user's transmission rate
$X(t)$ = the sum of every user's transmission rates
X_{goal} = desired load level

Fairness: Another way to judge an AIMD algorithm's worth is by its *fairness characteristic*. In an ideal case, different users should have an equivalent share, or "the same bottleneck", of the total amount of load put on the network (see Equation 3).

(Equ. 3)

$$F(x) = \frac{(\sum x_i)^2}{n(\sum x_i^2)}$$

where:

x_i = the i-th user's transmission rate
n = the number of users on the network

Let us look at a hypothetical network scenario with two users sharing the load (see Figure 1). Let point 0 be the starting point. At this point, the load by $User1$ is 30 and that by $User2$ is 60. Thus, at point 0, the total load on the network (i.e. 30 + 60 = 90) is less than the total efficiency of the network [X_{goal}, assumed to be 100], and the system is underloaded (under-utilized). By principle of additive increase, we prompt the users to add a certain amount a_I (say, 30) to both loads. Now, we are at point 1, where the load allocation is (60, 90).

The total load at this point is more than our X_{goal} and hence the system is over-loaded (over-utilized). Now, by principle of multiplicative decrease, we multiply both our load values by a constant b_D, between 0 and 1 (say, 0.5).

The algorithm consists of the load following a similar pattern of the algorithm, indefinitely.

The goal is to bring our load values as close to being perfectly efficient and fair [i.e. an allocation of (50, 50)] as possible. Note that the load values are constantly changing and thus cannot be brought to the ideal point and stopped. As the load is subjected to multiple iterations of AIMD , we see that the movement of our load values is bound by a zig-zag pattern, with the values oscillating around the efficiency line, each line lower to the previous line by an angle of 45 degrees. There are two important characteristics of this zig-zag motion of the loads that help us determine the efficiency of the process of congestion control/avoidance (see **Convergence to Efficiency** and **Smoothness**).

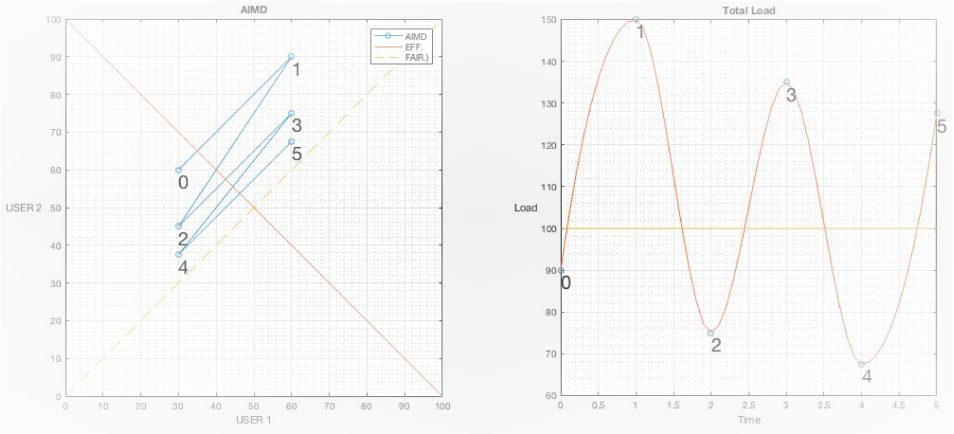


Figure 1: Graphical Representation of the AIMD Algorithm

Load Distribution: Another requirement is that the centralized scheme should have complete knowledge of the system. Obviously for real networks, the most viable option is to restrict the knowledge of users to the current state of loading (i.e Underloaded/ Overloaded), otherwise it may overload the network.

Convergence: Irrespective of the starting point of our load distribution, we require a congestion avoidance algorithm to converge our distribution towards the efficiency and the fairness line of the graph. Two factors that determine the quality of the convergence are the *time* taken to converge to a point of maximum fairness and the *smoothness* of the algorithm around the network capacity once the maximum fairness is reached.

Convergence to Efficiency The *Convergence to Efficiency*, or the *Responsiveness*, is a measurement of the time it takes from the initial coordinate of the n users' positions to a point where their positions oscillate as close to the efficiency line as possible with the given parameters (see Equation 4). Responsiveness is best explained as the time taken by the load values to reach the "desired" location.

(Equ. 4)

$$t_e = \begin{cases} \frac{log\left(\frac{an+(b-1)X_{goal}}{an+(b-1)X(0)}\right)}{log(b)} & b > 0 \\ \\ \frac{X_{goal}-X(0)}{an} & b = 0 \end{cases}$$

where:

a = the additive increase parameter
n = the number of users on the network
b = the multiplicative decrease parameter
X_{goal} = the total network capacity
$X(0)$ = the sum of every user's original transmission rate

Smoothness The *Smoothness* of a certain algorithm is the maximum distance, or the maximum oscillation, over the efficiency line once the maximum efficiency convergence has occurred. A larger oscillation would result in poorer network performance because the load would be further above the network capacity (see Equation 5). In other words, the smoothness is the magnitude of oscillations of the load values around the efficiency line as they move towards the 'desired' location.

(Equ. 5)

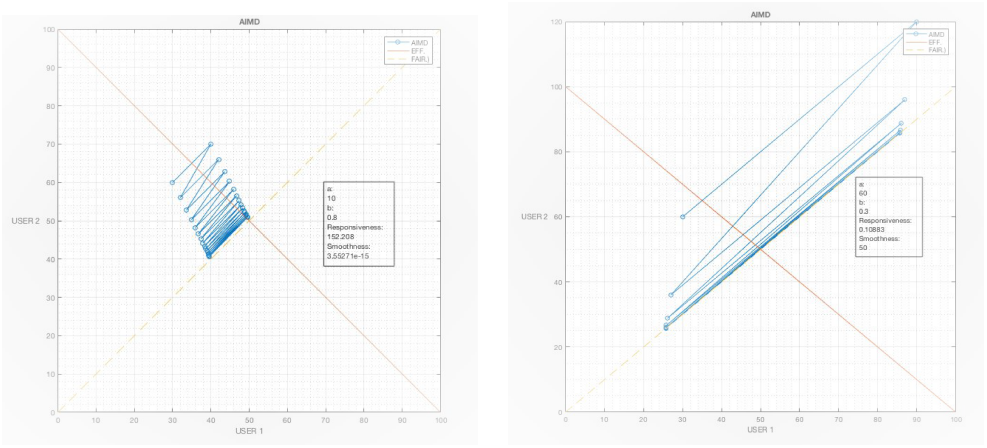
$$s_e = |an + (b-1)X_{goal}|$$

where:

a = the additive increase parameter
n = the number of users on the network
b = the multiplicative decrease parameter
X_{goal} = the total network capacity

As an example of how different parameters of the AIMD algorithm affect the time to efficiency convergence and smoothness, Figure 2 contains two graphs of the algorithm using different values for the additive increase and multiplicative decrease parameters. Ideally, the AIMD algorithm should converge quickly to the maximum efficiency. Once maximum efficiency is achieved, the algorithm should not oscillate too far beyond the maximum network capacity. However, the convergence to efficiency is inversely proportional to the smoothness of the algorithm. A quick convergence to efficiency translates into greater oscillations, whereas smaller oscillations create longer convergence times.

In Figure 2(a), a relatively small value of a_I (10) and a relatively large value of b_D (0.8) translates into smoother network usage for the users, with a maximum oscillation just over 0. However, it takes over 30 iterations of the code to converge, or respond. In Figure 2(b), a relatively large value of a_I (60) and a relatively small value of b_D (0.3) converges in about 10 iterations. These values cause the user's usage to oscillate wildly, with a smoothness value of 50. This illustrates the inverse proportionality of the *Convergence to Efficiency* and *Smoothness* functions.



(a) Better Smoothness

(b) Better Convergence

Figure 2: Smoothness vs. Convergence

2 Current State of Project

To begin the project and to gain some insight into the Additive Increase, Multiplicative Decrease (AIMD) algorithm, hand sketches were configured with set values of *30* for the additive increase parameter and *0.5* for the multiplicative decrease parameter. As a control source, a hand sketch starting on the fairness line at coordinates (50, 50) was created as a way to compare the different values used in each sketch (see Figure 3). The coordinates refer to the position of user x_1 and user x_2, respectively.

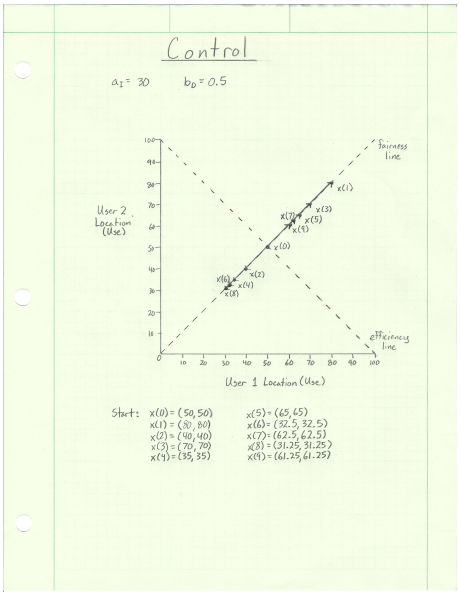
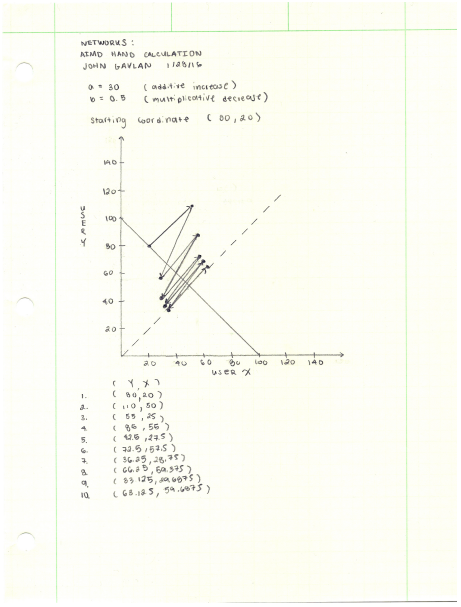


Figure 3: Control

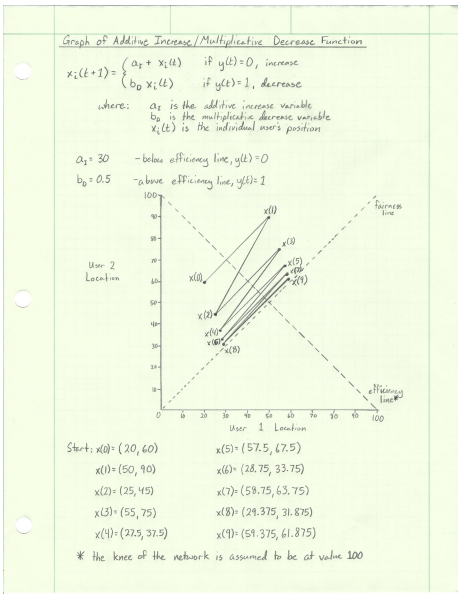
Figure 4 is comprised of each of the three hand sketches of the AIMD algorithm. Figure 4(a) begins at the coordinates (20, 80). Figure 4(b) begins at the coordinates (20, 60). Figure 4(c) is the calculation page of the sketch featured in Figure 4(d), which begins at the coordinates (40, 60).

From the hand sketching, a pattern emerged. As the value of t increases, the coordinates for user's x_1 and x_2 oscillate around the efficiency line. Furthermore, the coordinates also approach the fairness line, with the network sharing the capacity more equally between the two users with each increase in time.

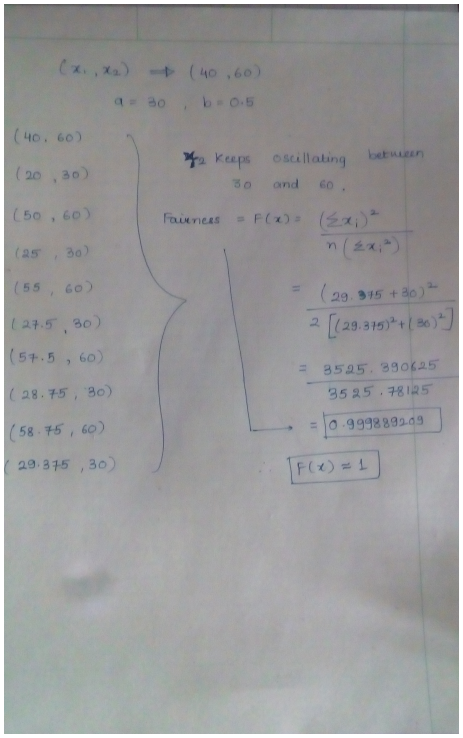
Once the hand sketching brought some familiarity to the project, two codes were created in the MATLAB language. They both perform similarly, with some exceptions. Two codes were created as a way to check the values and to ensure they are both performing optimally as a type of failsafe. The computer-generated graphs of the original graphing code can be found in Figure 5. The computer-generated graphs tend to follow the same pattern as the hand sketches.



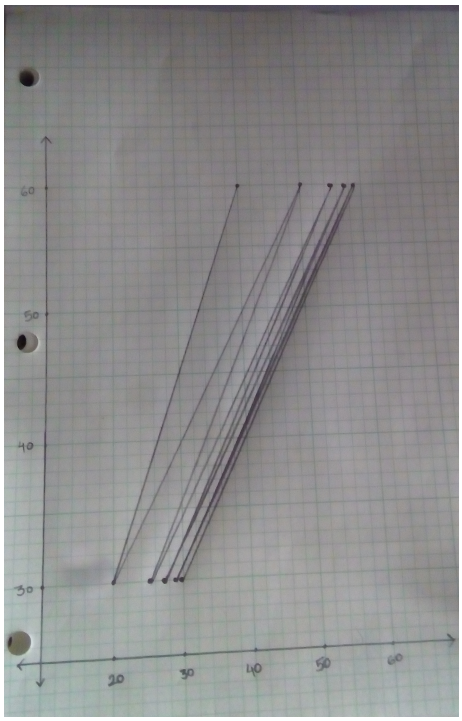
(a) Start: (20, 80)



(b) Start: (20, 60)

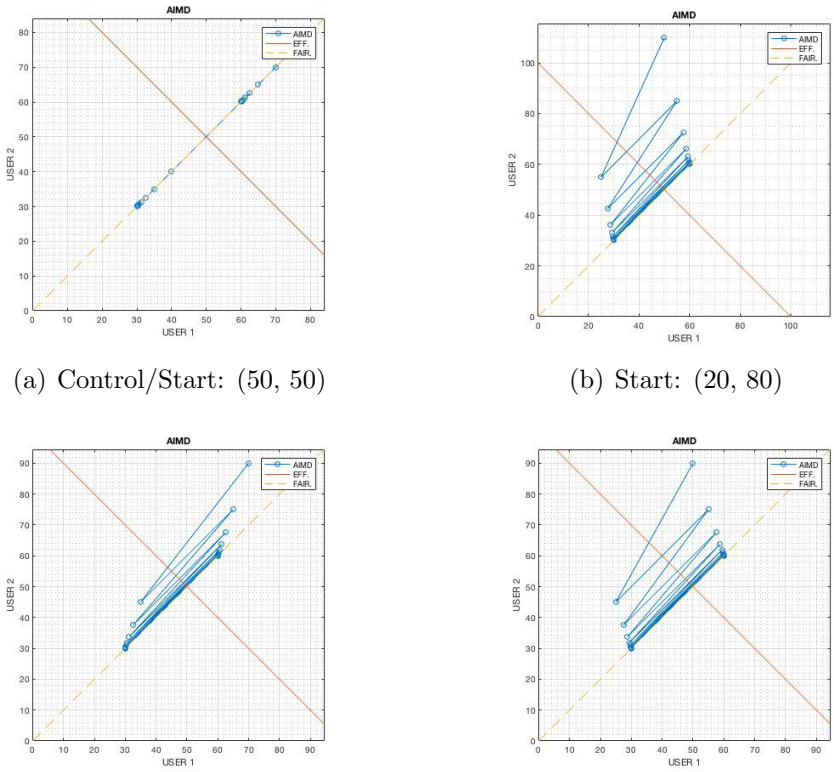


(c) Calculations of (40, 60)



(d) Graph of (40, 60)

Figure 4: Hand Sketches of AIMD Algorithm



(a) Control/Start: (50, 50)

(b) Start: (20, 80)

(c) Start: (40, 60)

(d) Start: (20, 60)

Figure 5: Computer-Generated Graphs

In addition to the graphs, we have begun to incorporate the convergence formulas for fairness and smoothness into our codes. A new MATLAB code has also been created to deal with them exclusively. This code takes an input for the multiplicative decrease variable and runs additive increase variables from 1 to 99 through it and puts the data into an Excel spreadsheet (see **Appendix C**). While this code still needs work, it is a much quicker method than computing one at a time. The current iteration of the code that generated the above graphs can be found in **Appendix A.1**. The other similar code can be found in **Appendix B**.

As an explanation of the process toward data analysis through variable automation the following explanation delves into the development of a code separate from the initial code created. For the initial code, each iteration would increase or decrease depending on the desired X_{goal} and all iterations were observed. Fairness as well as responsiveness along with smoothness was

calculated for at the beginning of the code depending on the values set for A and B. The current code which creates this file is derived from the initial code created which has a set A, B, *user 1* and *user 2* start values. The second code created, separate from the initial code, has the initial code nested in two for loops which varies the values for A and B.

After each iteration, the A, B, *responsiveness*, *smoothness* and the mean values for X_1 and X_2 are written into a .csv file. A and B can be changed as desired but the ranges chosen are for graphical reasons pertaining to the report. In the table located in Appendix A.1, the first column is parameter A which ranges from 40 to 60, incremented by 10 and the second column is variable B from 0.1 to 0.9 incremented by 0.1. Some of the values are not properly calculated for and result in a *NaN*, short for not-a-number, which is possible numbers calculated to infinity. The third column calculates for *responsiveness*, the fourth, *smoothness*, the fifth, mean value for X_1 and the sixth, mean value for X_2 (see **Appendix A.3**). This will prove helpful for data analysis and will direct our project toward a more specified goal.

3 Direction of Research

To understand the basis of our project, we began by referring to algorithms, generally explained above, upon which data is distributed within a network. Also, seen above, we have begun the process by actualizing the algorithms into code in which we calculate certain data points depending on desired inputs that we intend on collecting for analysis.

For this project and the world outside this classroom, AIMD is commonplace for congestion control. As the project progresses and we collect data and research on our topic, we will begin to focus our scope on how certain parameters will affect the model in hopes of yielding enlightening information to prescribe some finding toward a solution. To reach some kind of a conclusion we will have built a code to model this network.

We coded past the base of our project which is the AIMD model and gathered data on variations of the base parameters included, which will then be analyzed against other parameters derived from the model. Our general direction will be constantly changing and pointing toward any relevant data that seems useful to our project and contribution to congestion control as a whole.

This data has been compiled into tables which show A and B variations with corresponding values for responsiveness, smoothness and the mean values for both users after x amount of iterations. This provides some hard data and

contributes greatly to the understanding of how different values for the additive increase and multiplicative decrease variables affect the overall effectiveness of the algorithm. With this data, we may be able to start narrowing down the variables that we can run through our future codes for research down the road.

As per the company representative, the focus of our project will begin to transition to a network that incorporates more users to better simulate real-world situations. To do this, we need to develop our code in a way that we can change both the number of users and each of their starting positions (see **Appendix A.2**).

Additionally, we have begun research on a more advanced algorithm called CUBIC [2]. As can be ascertained from the name, the main function of the CUBIC algorithm is a cubic equation. This allows for a quicker approach to network efficiency and also equates to better smoothness around the most efficient network state. We are currently researching this algorithm.

4 Case Study

Using the code in **Appendix A.2**, we were able to accomplish a case study. We wanted to examine the effects of changing the parameters of the additive increase (a) and multiplicative decrease (b) following a sudden increase in network traffic. For the control group, the users did not change a and b in response to increase in network traffic. For the experimental groups, the users responded to increase in traffic by changing a and b. 15 iterations of increase/decrease in transmission rates were observed. Initially, there were 4 users on the network, with initial additive increase factor set at $a = 35$ and initial multiplicative decrease factor set at $b = 0.5$. After 5 iterations, the number of users was increased to 7, and thus the load on the network suddenly increased.

The results of the time to convergence and smoothness were observed. The data was entered into a table that can be viewed in **Appendix A.4**. We observed that an increase in a and decrease in b converged first, but the output was not very smooth. Also, an optimal trade-off between smoothness and convergence occurred when both a and b were decreased.

5 Future Work

In the future, we would like to add a graphical user interface (GUI) to our AIMD models. The GUI would have entry points where certain parameters

and conditions may be entered into the model, such as network capacity and/or the number of users on the network. Once these values are entered, our model will run through the calculations in the background. The results would then print to the GUI. This would make our models easier to use and can be used by Grier Forensics to aid in the application of the AIMD algorithm. They will be able to compare network conditions against AIMD parameters quickly to determine the best settings for their real-world applications.

Furthermore, a list of values for the parameters would also be quite useful. This list would dynamically change based on the entered values and results of the aforementioned GUI. Preferably, a range of values centered around the values entered into the GUI would be calculated for and printed on the list. This way, a more suitable network situation may be found and utilized. Grier Forensics could use this list as a quick-reference guide for their applications.

A model based on the CUBIC algorithm would also be created. This model could then be compared with our models of the AIMD algorithm to see if similar values produce similar results. This would provide Grier Forensics with a better array of tools to tackle real-world networks.

6 Conclusions

Based upon on our case study, we found that an increase in the additive increase parameter (a) and a decrease in the multiplicative decrease parameter (b) caused the model to converge quicker, but the resulting output had decreased smoothness. This builds upon the theory that Dah-Ming Chiu and Raj Jain proposed, that time to converge is inversely proportional to the smoothness of the output [1].

Additionally, from our case study we can conclude that the optimal tradeoff between the time to convergence and the smoothness of the output occurs when both the additive increase (a) and multiplicative decrease (b) parameters decrease.

This project also generated questions that we were unable to answer. For instance, the convergence of efficiency formula (see Equation 4) sometimes resulted in negative or non-existent values. While this makes sense mathematically, a negative or non-existent time does not make sense for network convergence. Our research has proven unfruitful so far, and would require more research and simulation to formulate a reasonable hypothesis.

Overall, we attempted to gain valuable insight into a very broad topic. While we were unable to provide several definitive answers, we did find information that can provide a valuable foundation to build future research upon.

PIC Math is a program of the Mathematical Association of America (MAA) and the Society for Industrial and Applied Mathematics (SIAM). Support is provided by the National Science Foundation (NSF grant DMS-1345499).

References

[1] Dah-Ming Chiu and Raj Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN systems*, 17(1):1–14, 1989.

[2] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new tcp-friendly high-speed tcp variant. *ACM SIGOPS Operating Systems Review*, 42(5):64–74, 2008.

Appendices

A AIMD Code #1

A.1 AIMD Code #1A

```
1   function[Fairness, Responsiveness, Smoothness, X1, X2, T] =
    AIMDJG2(AdditiveIncreaseConstant, MultiplicativeDecreaseConstant,
    User1StartValue, User2StartValue, Iterations, Datacap)
2   % AIMD Additive Increase/Multiplicative Decrease
3   % Mathematical Model Used For Network Congestion
4
5   %   Author:  John Gaylan
6   %   Date:  1/25/17
7   %   Class:  MATH 610
8   %   Project:  NETWORKS
9
10  %% SETUP
11  a = AdditiveIncreaseConstant;
12  b = MultiplicativeDecreaseConstant;
13  x1 = User1StartValue;
14  x2 = User2StartValue;
15  n = Iterations;
16  c = DataCap;
17  Xgoal = x1+x2;
18  eff = 0:c;
19  EFF = c:-1:0;
20  fair = 0:c;
21  FAIR = 0:c;
22
23  %% INITIAL PARAMETERS
24  Responsiveness = (log(((a*2)+(b-1)*Xgoal)/((a*2)+(b-1)*Xgoal)))/log(b);
25  Smoothness = abs((a*2)+((b-1)*Xgoal));
26
27  %% MAIN
28  for i = 2:n
29    y(i) = x1(end) + x2(end);
30    if y(end) <= Xgoal
31      x1(i) = a + x1(end);
32      x2(i) = a + x2(end);
33    else y(end) > Xgoal;
```

```
34      x1(i) = b * x1(end);
35      x2(i) = b * x2(end);
36    end
37  end
38
39    y(end+1) = x1(end) + x2(end);
40    TotalLoad = y(2:end);
41
42  %% PLOT
43  X1 = x1';
44  X2 = x2';
45  T = TotalLoad';
46  xx = 0:0.1:n;
47  YY2 = [0 TotalLoad];
48  xx1 = [0 x1];
49  xx2 = [0 x2];
50  g = pchip(0:n,YY2,xx);
51  yy = pchip(0:n,xx1,xx);
52  yy2 = pchip(0:n,xx2,xx);
53  subplot(2,2,1)
54  plot(x1,x2,'-o',eff,EFF,fair,FAIR,'--')
55  title ('AIMD')
56  xlabel ('USER 1')
57  ylabel ('USER 2','Rotation',0,'VerticalAlignment','middle',
    'HorizontalAlignment','right')
58  axis square
59  grid on
60  grid minor
61  legend('AIMD','EFF.','FAIR.')
62  subplot(2,2,2)
63  plot(1:n,TotalLoad,'o',xx,g)
64  title ('Total Load')
65  xlabel ('Time')
66  ylabel ('Load','Rotation',0,'VerticalAlignment','middle',
    'HorizontalAlignment','right')
67  grid on
68  grid minor
69  subplot(2,2,3)
70  plot(1:n,x1,'o',xx,yy)
71  title('User 1')
```

```
72  xlabel('Time')
73  ylabel('Load','Rotation',0,'VerticalAlignment','middle',
    'HorizontalAlignment','right')
74  grid on
75  grid minor
76  subplot(2,2,4)
77  plot(1:n,x2,'o',xx,yy2)
78  title('User 2')
79  xlabel('Time')
80  ylabel('Load','Rotation',0,'VerticalAlignment','middle',
    'HorizontalAlignment','right')
81  grid on
82  grid minor
83
84  %% CALCULATED PARAMETERS
85  Fairness = (sum(x1)+sum(x2))^2/(2*((sum(x1))^2+(sum(x2))^2));
```

A.2 AIMD Code #1B - Multiple Users

```
1   function[a, b, Responsiveness, Smoothness, MeanX] =
    AIMDJGN (AddIncreaseStart, increaseAdd, AddIncreaseEnd,
    MulDecreaseStart, increaseDec, MulDecreaseEnd, Iterations,
    Datacap, varargin)
2   Developed by John Gaylan and Surabhi Maheshwari
3
4   %% SETUP
5   UserStartValues = cell2mat(varargin);
6   x = UserStartValues;
7   a1= AddIncreaseStart;
8   a2=AddIncreaseEnd;
9   k=increaseAdd;
10  b1=MulDecreaseStart;
11  b2=MulDecreaseEnd;
12  p=increaseDec;
13  n=Iterations;
14  c=Datacap;
15  Xgoal= sum(x);
16
```

```
%% MAIN
fileID = fopen('USERDATANUSERS.csv','w');
fprintf(fileID, 'a, b, t, s, MeanX \n');
for a=a1:k:a2
  for b=b1:p:b2
    x = UserStartValues;
    y = [];
    z = [ 60 120 ];
    j = length(UserStartValues);
    Responsiveness = (log(((a*j)+(b-1)*c)/((a*j)+(b-1)*
    (sum(UserStartValues))))/log(b));
    Smoothness = abs((a*j)+((b-1)*c));
    for i=2:n+1
      y(i-1) = sum(x);
      if y(end)<c
        x = a + x;
      else y(end)>=c
        x = b * x;
      end
      if(i==n+1)
        x
      end
    end
  y
  MeanX = mean(UserStartValues);
  SILVER = table([a],[b],[Responsiveness],[Smoothness],[MeanX])
  m = [a, b, Responsiveness, Smoothness, MeanX];
  M = m';
  fprintf(fileID, '%4.2f, %4.2f, %4.4f, %4.2f, %4.2f\n',M);
  end
end
```

A.3 Table Generated From AIMD Code #1A

A	B	Resp.	Smooth.	X1	X2
40.00	0.10	-1.000	0 10.0	0 46.7	9 50.1 2
40.00	0.20	Inf	0 .00 31	.43 35	.00
40.00	0.30	0.4407	10.00	37.94	42.02
40.00	0.40	0.2863	20.00	46.24	51.00
40.00	0.50	0.2224	30.00	57.14	62.85
40.00	0.60	0.1866	40.00	56.74	62.88
40.00	0.70	0.1634	50.00	55.14	62.66
40.00	0.80	0.1469	60.00	55.71	65.48
40.00	0.90	0.1346	70.00	56.08	70.81
50.00	0.10	0.2788	10.00	31.95	35.13
50.00	0.20	0.2091	20.00	38.39	41.96
50.00	0.30	0.1742	30.00	46.40	50.49
50.00	0.40	0.1525	40.00	56.61	61.38
50.00	0.50	0.1375	50.00	70.00	75.71
50.00	0.60	0.1263	60.00	58.59	64.46
50.00	0.70	0.1177	70.00	67.05	74.56
50.00	0.80	0.1107	80.00	64.25	73.88
50.00	0.90	0.1049	90.00	62.16	76.74
60.00	0.10	0.1139	30.00	37.71	40.88
60.00	0.20	0.1133	40.00	45.36	48.93
60.00	0.30	0.1088	50.00	54.87	58.95
60.00	0.40	0.1040	60.00	66.98	71.75
60.00	0.50	0.0995	70.00	58.72	63.61
60.00	0.60	0.0955	80.00	60.53	66.32
60.00	0.70	0.0919	90.00	65.09	72.13
60.00	0.80	0.0887	100.0	0 63.4	4 72.7 7
60.00	0.90	0.0859	110.0	0 65.7	6 80.2 1

A.4 Table Generated From AIMD Code #1B

RESPONSE: CHANGE IN a AND b					
	Control	Exp. #1	Exp. #2	Exp. #3	Exp. #4
a	35	55	55	15	15
b	0.5	0.3	0.7	0.3	0.7

Results:					
s	195	315	355	35	75
$t_{99\%}$	12	8	n/a	10	n/a

RESPONSE: CHANGE IN a OR b					
	Control	Exp. #5	Exp. #6	Exp. #7	Exp. #8
a	35	35	35	55	15
b	0.5	0.7	0.3	0.5	0.5

Results:					
s	195	215	175	335	55
$t_{99\%}$	12	15	10	9	11

B AIMD Code #2

```
% Additive Increase Multiplicative Decrease Code
% Created by Andrew Whitener for the MATH 610 'AIMD' Project

function[G] = aimd5(a1, a2, x1, x2, n, s)

a = a1;
b = a2;
x = x1;
y = x2;
 n = number of iterations;
 signal = s;

 for f = x + y
   for t = 1:1:n
     while t <= n
       if signal >= f
         prev1 = x;
         prev2 = y;
         x = a + x;
```

```
20          y = a + y;
21          user1 = x;
22          user2 = y;
23          f = x + y;
24        elseif signal < f
25          prev1 = x;
26          prev2 = y;
27          x = b * x;
28          y = b * y;
29          user1 = x;
30          user2 = y;
31          f = x + y;
32        end
33        x_coord(1,:)  = x;
34        x_coord(isnan(x_coord(:,1)),:)  = [];
35        User_1(:,1) = unique(x_coord);
36        y_coord(1,:)  = y;
37        y_coord(isnan(y_coord(:,1)),:)  = [];
38        User_2(:,1) = unique(y_coord);
39        User_1_vs_User_2(t,:)  = [t User_1 User_2];
40        xx = 0:s;
41        yy = -xx + s;
42        xxx = 0:s;
43        yyy = xxx;
44        H = plot( prev1, prev2, 'k.', xx, yy, 'k-', xxx, yyy, 'k:');
45        hold on;
46        line([prev1 user1], [prev2 user2]);
47        hold on;
48        title('Additive Increase, Multiplicative Decrease Graph');
49        xlabel('User 1 Allocation');
50        ylabel('User 2 Allocation');
51        Z1 = ('Allocation Sum');
52        M1 = string('Efficiency');
53        M2 = string('Fairness');
54        legend(H, Z1, M1, M2);
55        t = n;
56        break
57      end
58    end
59  echo off
```

```
60   end
61
62   % Responsiveness
63
64   for b > 0
65     QT = log(((a*2) + (b-1) * signal)/((a*2) + (b-1) * (x1 + x2)));
66     VV = (x1 + x2);
67       if signal > VV
68         tsube = -(QT)/log(b);
69       elseif signal == VV
70         tsube = 0;
71       elseif signal < VV
72         tsube = (QT)/log(b);
73       elseif signal < 0
74         tsube = NaN; 75       end
76   elseif b==0
77     tsube = ((signal-(x+y))/(a*2));
78   end
79
80   % Smoothness
81
82   ssube = abs((a*2)+(b-1)*signal);
```

C Convergence/Smoothness Code

```
1   % Convergence to Efficiency(Responsiveness)/Smoothness Code
2
3   % Created by Andrew Whitener, Math 610 Networks Project
4
5 function[T] = SmoothEff( b, x, y, s)
6
7   for a = 1:1:99
8     if b > 0
9       tsube = ((log(((a*2)+((b)-1)*signal)/((a*2)+((b)-1)*(x+y))))/log((b)));
10       ssube = abs((a*2)+((b)-1)*signal);
11     elseif b==0
12       tsube = ((signal-(x+y))/(a*2));
13       ssube = abs((a*2)+((b)-1)*signal);
```

```
14      end
15      tsube(1,:)  = tsube;
16      ssube(1,:)  = ssube;
17      se(a,:)  = [a b tsube ssube];
18    end
19
20      T1 = table(se)
21      writetable(T1,'SmoothEff.xlsx')
```