CS 6043 – Computer Networking

Report

A Simulation-based Study of Additive increase/multiplicative decrease (AIMD)

Submitted by:
Gautham Pothana
Jithesh Kumar Kasi

1 Introduction

The TCP transport layer uses the AIMD feedback control technique, which is employed by TCP protocols. When there is no congestion identified, the AIMD Additive Increase phase is what causes the sender's congestion window to grow linearly. When congestion is identified, the multiplicative decrease phase is what causes the sender's congestion window to exponentially shrink. AIMD seeks to distribute available resources in a just and effective manner. There are three further combinations: AIAD, MIAD, and MIMD. However, they don't guarantee justice. On the other hand, AIMD guarantees fairness while vying for network bandwidth with other TCP flows. When there is no network congestion, MIMD expands the congestion window exponentially. As a result, the likelihood of congestion grows, and this specific flow consumes a significant amount of the available bandwidth, negatively affecting the performance of other flows. Congestion happens substantially more frequently as a result of its exponential increase. All TCP flows do not converge to utilizing the available network bandwidth, MIMD and AIMD do not achieve stability.

Formula for updating the load value in AIMD algorithm:

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

Fig. 1(a) Updating load values

Where,

t = current time value

$x_i(t)$ = i-th user’s transmission rate at time t

a = additive increase parameter

bD = multiplicative decrease parameter

2 Characteristics of the algorithm

There are two characteristics for the AIMD algorithm:

1. **Efficiency**: For loads with different users, ideally the load should stay on the efficiency line (grid capacity) of the graph. All elements below the line are underloaded and all elements above are overloaded.

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

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

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

where:

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

Fig. 2(a) Efficiency formula

2. **Fairness:** In an ideal world, different users should contribute the same amount to the total network load, or "same bottleneck".

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

where:

$$\begin{aligned} x_i &= \text{the } i\text{-th user's transmission rate} \\ n &= \text{the number of users on the network} \end{aligned}$$

Fig. 2(b) Fairness formula

3 Detailed Description

Our aim is to get our load values as near to ideal efficiency and fairness as feasible [i.e. an allocation of (50, 50) in the case of two users when the threshold is 100].

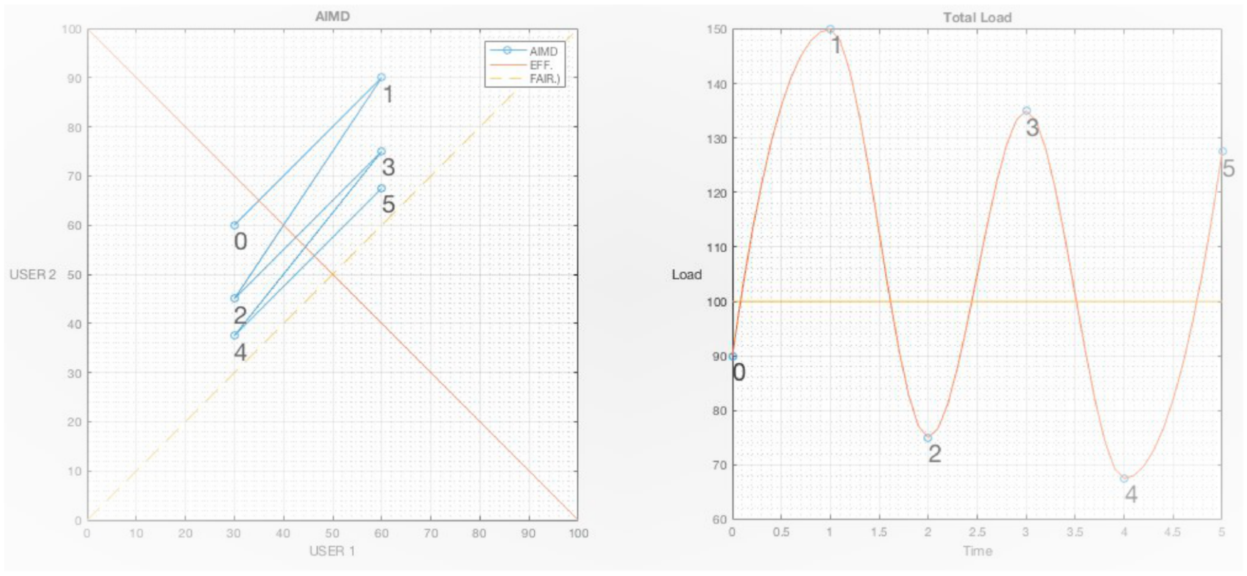

Fig. 3(a) Output graph for 2 users

Suppose there are two users in the network and the initial load values of the two users is (30, 60) and the additive increase parameter used by AIMD (a) is 30 and the multiplicative decrease parameter (b) is 0.5. And if we run the AIMD algorithm with these values we get the graph shown in figure 3(a).

- **Load Distribution:** A prerequisite is that the centralized system must be fully aware of the load in the system.
- **Convergence:** Whatever your initial starting point for load balancing, you need a congestion avoidance method that helps you converge on the efficiency-fairness line of the graph.
- **Responsiveness:** It is the amount of time it takes for load values to arrive at the "desired" location.

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

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

Fig. 3(b) Responsiveness

- **Smoothness:** The degree to which the load value oscillates around the efficiency line as it approaches the "target" position.

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

where:

$$\begin{aligned} a &= \text{the additive increase parameter} \\ n &= \text{the number of users on the network} \\ b &= \text{the multiplicative decrease parameter} \\ X_{goal} &= \text{the total network capacity} \end{aligned}$$

Fig. 3(c) Smoothness

Responsiveness and smoothness are inversely proportional to each other.

4 Comparison of plots achieved based on initial points of different fairness

Consider the AIMD algorithm for 2 users with varied initial fairness values- (30, 60), (50, 50), (10, 90) and additive increase a = 30 and multiplicative decrease value b = 0.5. We get the output graphs as shown in Figures 4(a), 4(b), 4(c).

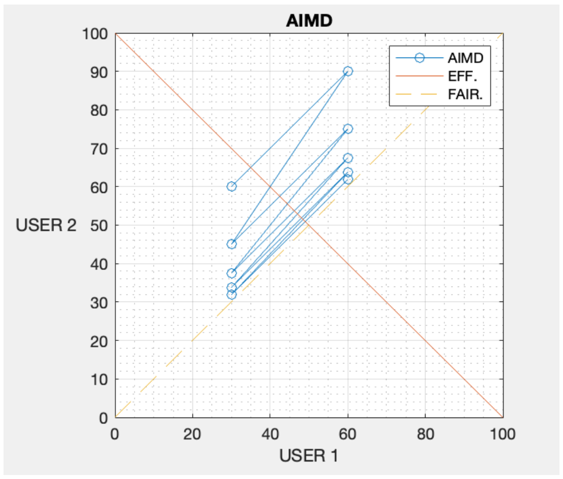

Fig. 4(a) (30, 60)

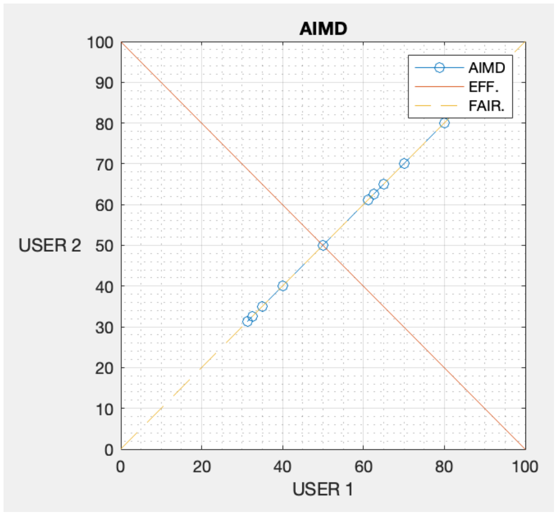

Fig. 4(b) (50, 50)

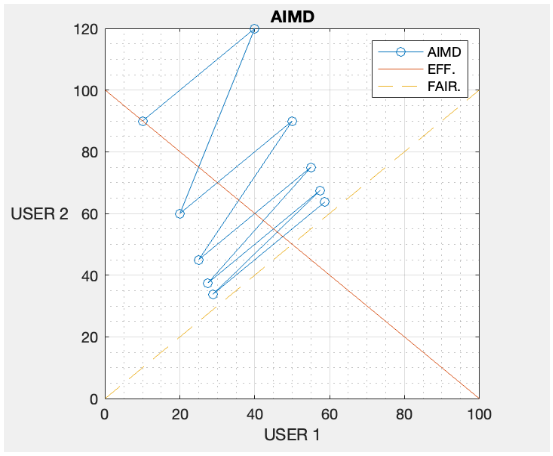

Fig. 4(c) (10, 90)

As we can observe in the figures 4(a), 4(b), 4(c) we can observe that the first figure took a medium amount of time to converge to the fairness line and second point took no time to converge since it is already in the ideal fair and efficient point. Whereas, the third load value took the highest amount of time to converge towards the fairness line. This infers that as the initial load values' fairness decreases, the amount of time and iterations needed to converge towards the fairness line increases.

5 Results/ Observations/ Insights

- **Better Smoothness:** Smaller a value and higher b value gives low oscillations which means better smoothness for the network users. This translates to larger convergence time as it takes more iterations of the code to converge.

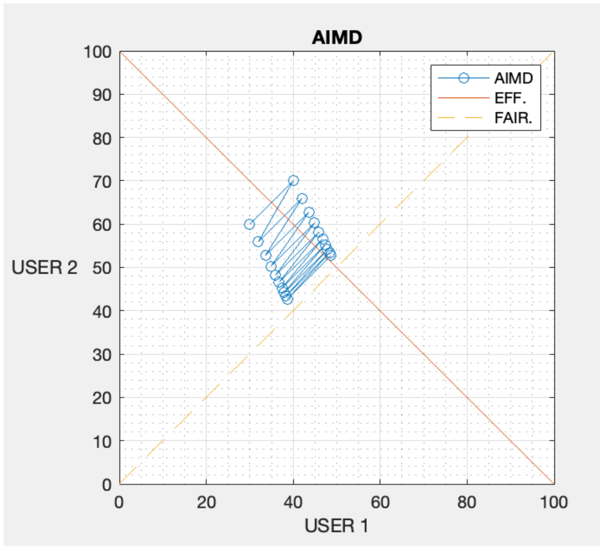

Fig 5(a) Better Smoothness

- **Better Convergence:** Larger a value and lower b value gives higher oscillations of user's network usage. This gives less smoothness and faster convergence time and less iterations of code.

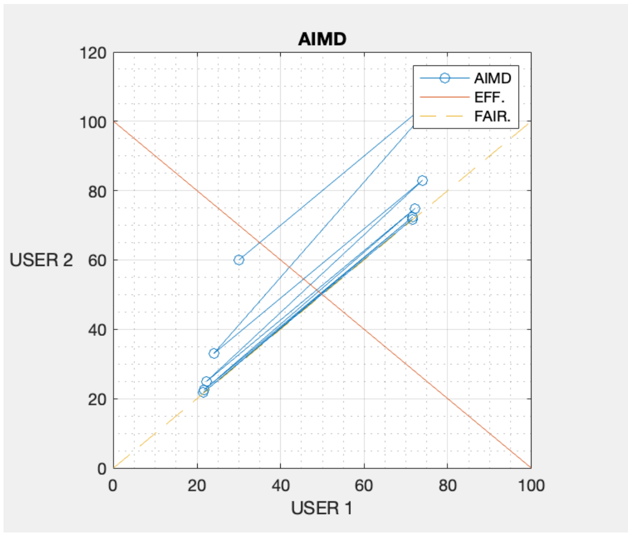

Fig 5(b) Better Convergence

As we can infer, Convergence and smoothness are both inversely proportional to each other. So, if you want better smoothness, you would require more time to converge and similarly, if you want faster convergence, it would be less smooth. But when both a and b were reduced, a smooth trade-off with convergence can be achieved.

6 Summary

Overall, AIMD is an effective congestion control algorithm. And it has multiple use cases: it can be used in transport layer of OSI model. It is also used in TCP congestion control. And it is also used in Stream Control Transmission Protocol (STCP). Sometimes, the Responsiveness formula produced zero or negative values. Another algorithm which is similar to AIMD that is also gaining a lot of popularity is CUBIC algorithm. CUBIC is a network congestion avoidance method for TCP that, in comparison to earlier algorithms, can establish high bandwidth connections over networks more rapidly and consistently in the presence of high latency. It uses a CUBIC expression.

7 References

[1] https://www.geeksforgeeks.org/aimd-algorithm/

[2] https://www.cs.princeton.edu/courses/archive/fall16/cos561/papers/Cubic08.pdf

[3] https://www.cse.wustl.edu/~jain/papers/ftp/cong_av.pdf

[4] https://en.wikipedia.org/wiki/Additive_increase/multiplicative_decrease

[5]
https://www.researchgate.net/publication/255591510_The_new_AIMD_congestion_control_algorithm

8 Contributions

Both Gautham Pothana and Jithesh Kumar Kasi have equally contributed in the completion of this project.