AN EXPLORATION OF THE EFFECTS OF VARYING PARAMETERS IN AN AIMD SYSTEM

JACK CAI

1. Introduction

Among common Internet protocols, TCP (Transmission Control Protocol) is often favored for its features such as error-checking and retransmission as well as being connection-oriented, all of which offer it significant advantages in terms of reliability. Additionally, TCP implements a host of other features such as a full-duplex communication, ordered delivery, cross-network compatibility, and congestion control, the last of which will be the focus of our present discussion.

We begin by providing some background and context on AIMD, after which we investigate the effects of modifying various parameters in an AIMD system.

2. Background and Preliminaries

TCP congestion control is implemented through a mechanism aptly named Additive Increase Multiplicative Decrease (AIMD). In this system, the host probes for available bandwidth among its clients by increasing additively increasing the number of packets it sends (denoted by CWND) until a loss is detected, upon which it multiplicatively decreases said number of packets.

Definition (Additive Increase Multiplicative Decrease (AIMD))**.** Let α be a strictly positive integer and β be a real number in the range $(0, 1)$. Let r be a client of host H and let **cwnd** be the number of packets H sends to r at once at each communication exchange. Then we have:

while Connection **do**
  while No Loss Detected **do**
    cwnd $\leftarrow$ **cwnd** $+ \alpha$
  end while
  cwnd $\leftarrow$ **cwnd** $* \beta$
end while

For a system with two clients, we would see a corresponding graph of their respective allocations (**cwnd**) similar to the one below (Figure 1):

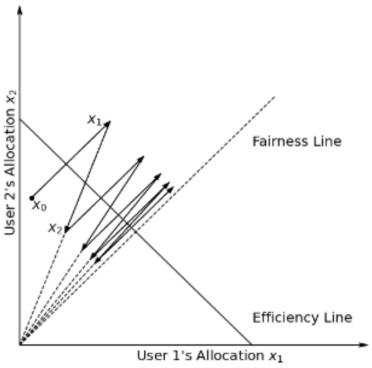

FIGURE 1. Example of a CWND graph for an AIMD TCP system with two users

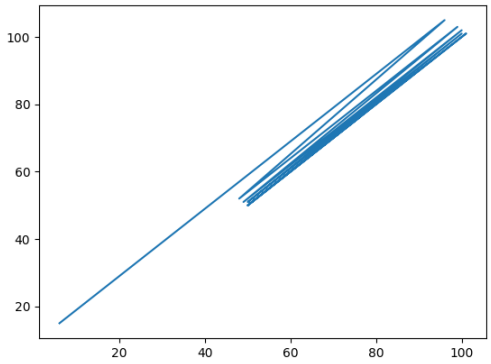

FIGURE 2. Simulation of AIMD with 2 clients and default α, β

We can simulate such a system with the code here (alternatively, check Code Snippet 1 in the Appendix). Running this code gives us the following output:

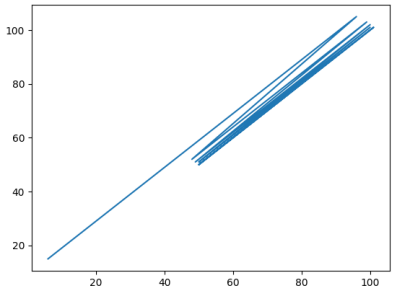

FIGURE 3. Simulation of AIMD with 2 clients and default α, β

Theorem (Asymptotic Behavior of AIMD)**.** For a given TCP system W implementing AIMD with n clients, let $w_i(k)$, $1 \leq i \leq n$ denote the congestion window size (**cwnd**) allocated by W to the i^{th} client immediately before the k^{th} congestion event. Additionally, let α_i, β_i respectively denote the values of the additive increase and multiplicative decrease allocated by W to w_i. Finally, we use the shorthand $W(k) = \begin{bmatrix} w_1(k) & w_2(k) & \dots & w_n(k) \end{bmatrix}^T$. Then

$$\lim_{k \to \infty} W(k) = C \begin{bmatrix} \frac{\alpha_1}{1-\beta_1} \\ \dots \\ \frac{\alpha_n}{1-\beta_n} \end{bmatrix}$$

for some real and positive constant C.

Proof. The idea is that $x = \begin{bmatrix} \frac{\alpha_1}{1-\beta_1}, \dots, \frac{\alpha_n}{1-\beta_n} \end{bmatrix}^T$ is a Perron eigenvector of the matrix A such that $W(k+1) = AW(k)$. Being a Perron eigenvector of A, it follows the eigenvector λ_x for

x is real, unique, and of largest magnitude for A. Moreover, $\lambda_x = 1$. For a complete proof, visit [1]. □

3. Further Investigations

In the context of a data center, one would reasonably expect significantly more than just 2 clients. Suppose we modified our code to run with 200 clients instead of 2 for 1000 iterations (Appendix: Code Snippet 2). This gives us the output in Figure 4: It is immediate that

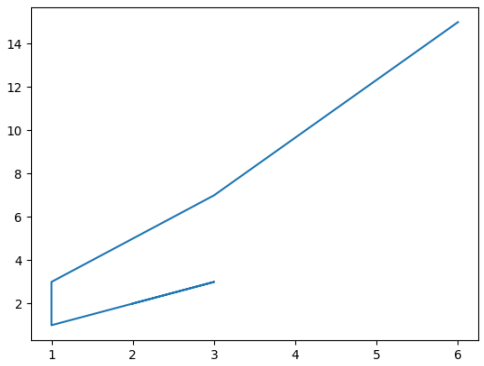

Figure 4. Graph of two clients in an AIMD simulation with 200 clients, maximum window size of 200, and default α, β

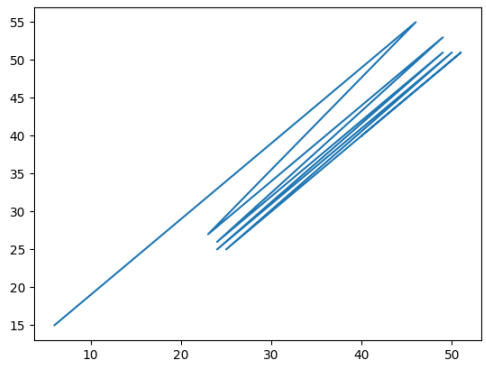

Figure 5. Graph of two clients in an AIMD simulation with 200 clients, maximum window size of 5000, and default α, β

the resulting figure is not at all alike the saw-toothed plot we are used to. However, we may notice that although the number of clients increased, the maximum bandwidth supported by our data center did not. Indeed, fixing our window size at 5000 (Appendix: Code Snippet 3) gives us back the graph we are familiar with (Figure 5).

It may be reasonable that a data center serving numerous clients simultaneously would want to process its requests as fast as possible. Increasing α to 10 (Appendix: Code Snippet 4), we see that our system reaches equilibrium much quicker (Figure 6).

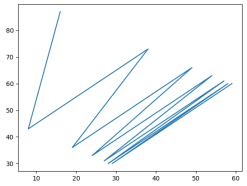

Figure 6. Graph of two clients in an AIMD simulation with 200 clients, maximum window size of 5000, and $\alpha = 10, \beta = 0.5$

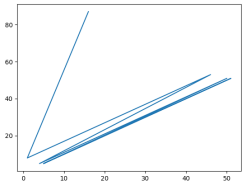

Figure 7. Graph of two clients in an AIMD simulation with 200 clients, maximum window size of 5000, and $\alpha = 1, \beta = 0.1$

Responsiveness can also be achieved by achieved by decreasing β. Setting $\beta = 0.1$ (Appendix: Code Snippet 5), we see that equilibrium is reached in as few as 6 steps (Figure 7).

Such a configuration would understandably lead to improved fairness, as each client would quickly receive its optimal bandwidth. However, one may also imagine that this may come at the cost reliability and throughput; clients may find, for instance, that their download speed is subject to significant variation from one moment to the next.

As we would expect, increasing β (Appendix: Code Snippet 6) results in slower changes in client **cwnd** sizes, and thus stabler transfer times (albeit at the cost of fairness) (Figure 8):

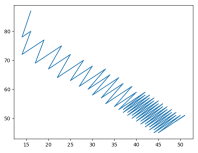

FIGURE 8. Graph of two clients in an AIMD simulation with 200 clients, maximum window size of 5000, and $\alpha = 1, \beta = 0.9$

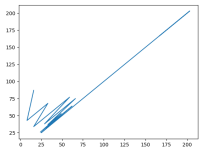

FIGURE 9. Graph of two clients in an AIMD simulation with 200 clients, maximum window size of 5000, arithmetically increasing α, and $\beta = 0.9$

We can further experiment with varying α by increasing it arithmetically in a similar fashion to each receiver's window size (Appendix: Code Snippet 7). This produces the plot in Figure 9.

We may be surprised to notice the aberrative point at the top right corner along the equilibrium line. This is because as the number of iterations increases, the corresponding increases in α become extremely large. Thus arithmetic increases in α is not particularly effective.

On the other hand, if we were to geometrically vary α (Appendix: Code Snippet 8) we get the following output, which we may observe to not be significantly different from ones we obtained previously:

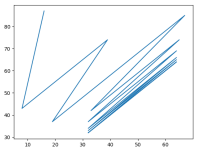

FIGURE 10. Graph of two clients in an AIMD simulation with 200 clients, maximum window size of 5000, arithmetically increasing α, and $\beta = 0.5$

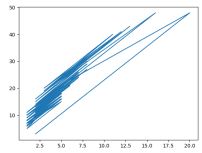

FIGURE 11. Graph of two clients in an AIMD simulation with $\{n, q_{\max}, \alpha_1, \beta_1, \alpha_2, \beta_2\} = \{200, 5000, 2, 0.3169740839374581, 5, 0.491574497075565\}$

Thus it would appear that the specific method of varying α (and likely too β) do not matter

so much as its effective proportion with respect to the other parameters (ie. maximum window size, number of clients, client window sizes, etc.). We may also observe that the slope of the equilibrium line in each simulation is always identical (in fact it is one). This is because, according to the theorem outlined in Section 2, the equilibrium window size w for a particular client c satisfies

$$\lim_{k\to\infty} w_c(k) = C\frac{\alpha_c}{1-\beta_c}$$

Since in our case α_c and β_c are identical for all clients, we have that the slope $m_{2,1}$ of our equilibrium line between clients 1 and 2 satisfies

$$m_{2,1} = \lim_{k\to\infty} \frac{w_2(k)}{w_1(k)} = \lim_{k\to\infty} \frac{C\frac{\alpha_2}{1-\beta_2}}{C\frac{\alpha_1}{1-\beta_1}} = \frac{C\frac{\alpha}{1-\beta}}{C\frac{\alpha}{1-\beta}} = 1$$

If we modify our code to provide proprietary α_i and β_i to each client i (Appendix: Code Snippet 9), we get the output in Figure 11.

If we sum the window sizes over the last 100 iterations for each client, we get an average window size of 2.01 and 6.54 for clients 1 and 2 respectively. As we know $\lim_{k\to\infty} w_i(k) = C\frac{\alpha_i}{1-\beta_i}$, we can evaluate

$$\begin{aligned}\lim_{k\to\infty} \frac{w_2(k)}{w_1(k)} &= \frac{C\frac{\alpha_2}{1-\beta_2}}{C\frac{\alpha_1}{1-\beta_1}} \\ &= \frac{\frac{5}{1-0.49157449707556}}{\frac{2}{1-0.3169740839374581}} \\ &= 3.3585348892503184 \approx 3.253731343283582 = \frac{6.54}{2.01}\end{aligned}$$

which is consistent with what we expect.

References

[1] Avi Berman, Robert Shorten, Nui Maynooth, Douglas Leith. Hamilton Institute, (2003). *Positive Matrices Associated With Synchronised.*

Cheriton School of Computer Science, University of Waterloo
Email address: jack.cai@uwaterloo.ca