NANYANG TECHNOLOGICAL UNIVERSITY

SCHOOL OF COMPUTER SCIENCE & ENGINEERING

TCP AIMD Congestion Control Simulation Report

Chan Eu Ching
U2021000F

Introduction

Transmission Control Protocol (TCP) depends on packets drops as a signal of congestion, detected by duplicated ACKs. If duplicated ACKs are detected, it means a congestion occurred. AIMD is an algorithm to control congestion occurring in TCP. It manages the flow of data across networks to avoid congestion and ensure efficient and reliable data transmission. It aims to achieve fairness bandwidth allocation in a network.

In short, the congestion window size (cwnd) increases by a fixed amount α every round-trip time (RTT), probing for usable bandwidth, until packet loss occurs. When packet loss occurs, cwnd decreases by a factor of β. The aim of TCP AIMD is to achieve efficient and fair congestion control.

This report will cover the simulation of TCP flow link utilizing AIMD algorithm. Mainly the update of α and β parameters will be tweaked to observe the cwnd behaviour of the flow link(s). The algorithm will also be tested on its scalability on higher number of flow links sharing the same bottleneck capacity c.

Assumptions Made

There are a few assumptions made in this exploration. The simulation assumes that every packet is sent at a constant RTT, with ideal conditions i.e. no delays or reordering of packets. Every packet sent over the network has a fixed size and both sender and receiver have perfect knowledge of network conditions. It is worthy to note that cwnd is typically stored and manipulated as an integer value to align with packet transmission process. In this simulation, floating-point representations are used for more precise calculations.

AIMD Algorithm

In the AIMD algorithm, the cwnd will update every RTT as follows:

$$cwnd = \begin{cases} cwnd + \alpha, & cwnd <= capacity \\ cwnd \times \beta, & cwnd > capacity \end{cases}$$

α refers to the aggressiveness of the probing of usable bandwidth while β represents the responsiveness to congestion detection. For standard AIMD TCP flow, the α and β parameters are set default at 1 and 0.5 respectively.

There can be more than 1 flow link sharing the same network, hence affecting the bandwidth allocation. The worst-case scenario is when all the flows simultaneously increase and decrease their cwnd resulting in poor link utilization. The best-case scenario would be when the cwnd of each flow changes periodically with evenly distributed decreasing moments for each flow.

Exploration

In this section, every system has a corresponding diagram found in images folder in Github.

Base Case (α=1, β=0.5).

BaseCaseCWND shows the sawtooth pattern of TCP AIMD. This is due to the additive increase and multiplicative decrease mechanism of the algorithm. In systems where both flows experience similar network conditions (same RTT, same packet loss rate) and same α and β values, they are likely to achieve fairness in bandwidth allocation. The system also reaches a stable balanced state (~25RTT). Subsequent explorations will be done in different network conditions and compared with *BaseCaseCWND* as reference.

BaseCaseAllocation shows the bandwidth allocation through RTT. The allocations converges towards the fairness and iterate along the line. This is in line with the goal of fairness and stability of the algorithm.

We will now explore systems with different network conditions.

System No.	No. of Flows n [F1, …, Fn]	RTT	Capacity cwnd c	Initial cwnd	Additive Increase α	Multiplicative Decrease β	Observation and Explanation
1	2	50	10	F1 = 2 F2 = 8	3	0.5	System 1 reaches a balanced state earlier than Base Case. This might because a more aggressive probes the usable bandwidth more in lesser RTT, resulting it to reach maximum congestion window faster hence quicker attainment of balanced state. However, it yields a smaller throughput due to its aggressiveness.
2	2	50	10	F1 = 2 F2 = 8	1	0.2	This system has a lower beta value which results in a more responsive reaction when congestion is detected as the flow decreases more aggressively. This allows more additive phases to occur. The throughput yield should be higher.
3	2	50	10	F1 = 2 F2 = 8	1	0.8	This system shows a less responsive reaction when congestion is detected. As the cwnd decrease only by 0.2 whenever congestion is detected. With less abrupt changes in cwnd, the throughput is highly likely to be higher than System 2, though it reaches balanced state later.
4	2	50	10	F1 = 2 F2 = 8	$\frac{10}{2^{\text{cwnd}}}$	0.5	The aggressiveness in probing of usable bandwidth is inversely proportional to cwnd i.e. flows probes more aggressively at lower cwnd values. The system also reaches a balanced state quicker than Base Case due to the adaptive α value to adjust according to the current cwnd hence the usable bandwidth is only probed minimally when congestion is about to occur. This additive increase results in a higher throughput compared to Base Case as the cwnd increases more aggressively per RTT at lower values. However, this system does not work well with more flow links.
5	2	50	30	F1 = 2 F2 = 8	$\frac{2}{\log_{10} \text{cwnd}}$	0.5	This system's concept is similar to System 4 however instead of exponential, logarithm is used. This means that at

							higher values, the increase is more aggressive than in System 4. The usable bandwidth is utilized more efficiently. This α is also experimented with higher c and managed to achieve stable state in a very short time. Since α is adjusted with c as well, it is highly scalable. However, the limitation is that cwnd cannot be 1 and the system cannot start with a congestion being detected.
6	10	65	65	F1 = 2 . . F10 = 11	$\frac{2}{\log_{10} \text{cwnd}}$	0.5	System 5 is scalable with increasing number of hosts as well. However, the minimum c for 10 flow links must be at least the sum of all initial cwnd flows to ensure that when a decrease happens, the updated cwnd does not fall to 1.
7	2	50	10	F1 = 2 F2 = 8	1	$\frac{1}{2^{\text{cwnd}-1}}$	This system calls for a more responsive decrease when the cwnd is high and vice versa. The system managed to attain stable state very quickly compared to Base Case as well. However, it yields a lower throughput due to its higher responsiveness to congestion.
8	10	50	60	F1 = 1 . . F10 = 10	1	$\frac{1}{2^{\text{cwnd}-1}}$	System 7 is also highly scalable with increasing number of hosts. There is no limit on c and cwnd unlike System 6. The throughput is also slightly higher compared to Base Case with 10 Flow Links
9	2	50	10	F1 = 2 F2 = 8	$\frac{2}{\log_{10} \text{cwnd}}$	$\frac{1}{2^{\text{cwnd}-1}}$	System 9 combines both α and β updates in System 6 and 7 respectively. The cwnds for both links converge near to c/2.
10	2	100	50	F1 = 2 F2 = 100	$\frac{2}{\log_{10} \text{cwnd}}$	$\frac{1}{2^{\text{cwnd}-1}}$	System 9 works well with high initial bandwidth and capacity. Compared to Base Case, it reaches a stable state faster. However, the throughput might be lower due to more aggressive increase.

Possible explorations

There are many other ways to explore the potential of AIMD function.

1. Higher speed network (refer to *HighSpeedNetwork*)
   - *HighSpeedNetwork* shows the default α and β parameters performed in high-speed network. Stable state is only achieved at ~400RTT compared to Base Case at ~30RTT. Even if logarithm or exponential function is used, since at larger values, their step increase is not significant, it does not help the system to converge to balanced state.

- Slow start can be used to implement more aggressive probing of usable bandwidth before a certain threshold. As seen in *SlowStart*, it takes lesser RTT to reach a balanced state a higher throughput is yielded as well.

2. Flows with different priorities (refer to *DifferentPriorities*)
   - Flow links with higher priorities will have a higher α value. At balanced state, flow links will not be distributed the same amount of cwnd due to its different priorities.
3. Flows with different RTT (refer to *DifferentRTT*)
   - For flows with different RTT, their updates will be at different timing. Flows with larger RTT will typically have larger initial cwnd to compensate for their longer round-trip times. *DifferentRTT* shows default AIMD but with two flows of different RTT values.

Perron Frobenius Theory

Perron Frobenius Theory serves as a mathematical framework to study the properties of positive matrices which can give insights into the stability and convergence of AIMD congestion control.

Firstly, a state transition matrix P to represent the behaviour of AIMD algorithm can be obtained. Eigenvalues and eigenvectors can be computed for P. Eigenvalues represents the scaling factors by which the eigenvectors are altered during matrix multiplication. If all eigenvalues are less than 1, then the system is said to be stable and cwnd sizes converge to a steady-state solution. The largest eigenvalue determines the convergence rate of the system. A dominant eigenvalue significantly smaller than 1 suggests faster convergence.

These values can be compared to determine which system has faster convergence to stable and balanced state.

Limitations and Solutions

1. TCP AIMD's multiplicative decrease phase might result in slow responsiveness to network congestion especially in high-speed networks and network with large RTTs. This can result in suboptimal throughput and increased latency.
   Solution: Develop a faster congestion detection mechanism that can detect congestion events more quickly like TCP BRR that uses RTT and packet delivery rate information to estimate network congestion.
2. AIMD might underutilize network capacity especially in higher bandwidths.
   Solution: Use adaptive congestion control algorithms that updates cwnd based on network conditions.
3. Congestion collapse may occur when multiple flows simultaneously increase their cwnd aggressively, causing performance degradation.
   Solution:Implement congestion control algorithms that prevent congestion collapse while efficiently utilizing bandwidth. TCP Reno can quickly recover from congestion events and maintain network stability.

TCP Ex Machina

With technology advancing, advanced artificial intelligence (AI) and machine learning (ML) are used to enhance the performance, scalability, and efficiency of TCP in data centre. TCP Ex Machina addresses the challenges like high-speed network, large-scale deployments, and complex network traffic. It then optimizes the TCP behaviour to improve network performance. From assisting in the design and optimization of TCP congestion control algorithms to detecting anomalies to predict network congestion, AI/ChatGPT can be employed to engineer TCP Ex Machina.

Conclusion

In conclusion, the experimentation of TCP AIMD provides insights to its behaviour, performance, and limitations to network congestion. Through experimenting, it is certain that there are many trade offs in the algorithm such as stability vs responsiveness, fairness vs efficiency. It is essential that the algorithm aims to balance these properties to achieve an effective congestion control algorithm.