## 1. Introduction:

The Transmission Control Protocol (TCP) is one of the main protocols of the Internet protocol suite. TCP operates on top of the Internet Protocol(IP) and is responsible for reliable and ordered data transfer between applications running on hosts communicating via an IP network. TCP can be divided into 3 phases, Connection establishment, Data Transfer and Connection Termination. However, TCP alone does not address the shared nature of network resources. Multiple senders could overwhelm a network, leading to congestion, delays, and data loss.

Congestion is a state occurring in the network layer when the message traffic is so heavy that it slows down network response time. TCP Congestion Control addresses this issue by controlling the entry of data packets into the network, enabling a better use of a shared network infrastructure and avoiding congestive collapse. These are essential goals TCP congestion Control aims to achieve:

- High Throughput: Maximize the data transfer rate without overloading the network.
- Fairness: Ensure all competing flows share network resources equitably.
- Quick Transient Response: Fast response following a sudden change or disturbance in the number of hosts and/or packets

In this assignment, we explore the tuning of parameters for Additive Increase, Multiplicative Decrease (AIMD), a widely used Congestion Control algorithm, and provide insights on other Congestion Control algorithms.

## 2. AIMD Principles:

Conventional AIMD first uses the slow-start algorithm until either a packet loss is detected, the receiver's advertised window (rwnd) becomes the limiting factor, or slow start threshold (ssthresh) is reached. This algorithm allows the hosts to swiftly transmit packets at the maximum pace the network can allow. The AIMD process then starts after one of the conditions stated is met. In this report, we will focus solely on the AIMD process.

The Additive Increase, Multiplicative Decrease (AIMD) algorithm combines linear growth of the congestion window when there is no congestion with an exponential reduction when congestion is detected. Multiple flows using AIMD congestion control will eventually converge to an equal usage of a shared link.

Principles of AIMD:

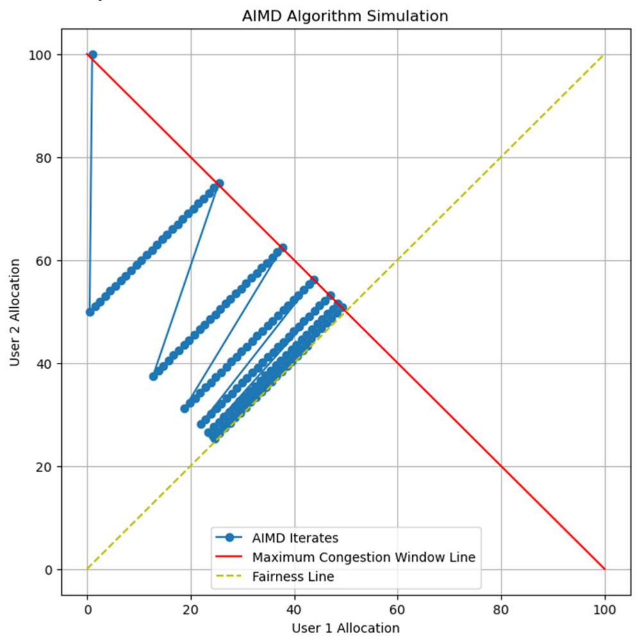
- Slow start: Starts cautiously with small data packets to probe the network capacity.
- Congestion window: Limits the amount of data in transit at any given time for each host.
- Additive increase, multiplicative decrease: Based on feedback like packet loss, the window size increases slowly (parameter Alpha) during good times but decreases significantly (parameter Beta) during congestion.

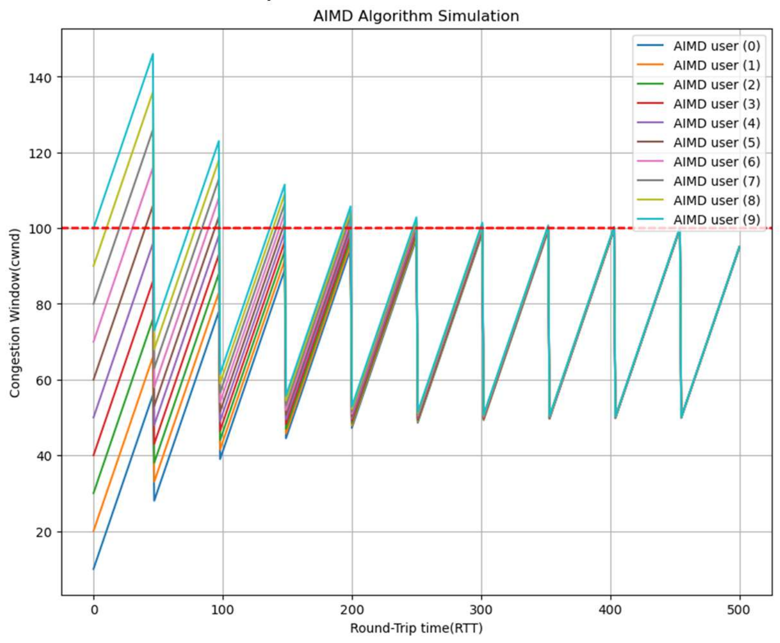
### 2.1 Conventional AIMD

After the slow-start process, conventional AIMD follows these rules to update the congestion window which indicates the among of packets to send at time t:

- Let w(t) be congestion window size at time t, a ($a>0$) be the additive increase parameter and b ($0<b<1$) be the multiplicative decrease factor
    - If congestion is not detected, then $w(t+1) = w(t) + a$
    - If congestion is detected, then $w(t+1) = w(t) * b$

Conventional AIMD typically sets additive increase parameter a to be 1, and multiplicative decrease factor b to be ½. The diagram below, **<Figure 1>**, shows AIMD convergence with their initial starting parameters. The left diagram shows host 2 utilizing the entire traffic before host 1 connects to the network while the right diagram shows 10 different hosts, with different initial starting congestion windows, converging along the fairness line after numerous Round-Trip times. In **<Figure 2>**, different parameters of a and b were used but all hosts share the same parameters





**<Figure 1>**: 2 Hosts congestion window convergence (left), 10 Hosts congestion window convergence with respect to Round-Trip time (right)

| With reference to diagram folder on github | Type, [a,b] | Number of Round-Trip time | Number of Hosts | Network maximum congestion window | Initial congestion window,e.g. [Host 1, …, Host x] | Additive increase parameter a | Multiplicative decrease factor b | Observation & Insights |
|---|---|---|---|---|---|---|---|---|
| **Diagram 1.1 (Figure 1)** | [Linear, Linear] | 175 | 2 | 100 | [1,100] | 1 | 0.5 | Conventional AIMD |
| **Diagram 1.2 (Figure 1)** | [Linear, Linear] | 500 | 10 | 1000 | [10,20,30,...,100] | 1 | 0.5 | Conventional AIMD |
| Diagram 2.1 | [Linear, Linear] | 75 | 2 | 100 | [1,100] | 4 | 0.5 | Faster Convergence |
| Diagram 2.2 | [Linear, Linear] | 200 | 10 | 1000 | [10,20,30,...,100] | 4 | 0.5 | Faster Convergence |
| Diagram 3.1 | [Linear, Linear] | 175 | 2 | 100 | [1,100] | 1 | 0.8 | Fast Recovery |
| Diagram 3.2 | [Linear, Linear] | 500 | 10 | 1000 | [10,20,30,...,100] | 1 | 0.8 | Fast Recovery |

**<Figure 2>**: Table showing different parameters a and b shared by all hosts

### 2.1.1 AIMD benefits

AIMD remains a popular and effective congestion control algorithm for TCP due to multiple key benefits. Here are some examples of those benefits:

- Simplicity and predictable behaviour
  - AIMD's reliance on well-defined parameters like "a" and "b" allows for straightforward configuration and analysis
- Stability and Fairness
  - AIMD achieves a good balance between maximizing network utilization and ensuring fairness among competing flows.

    - The multiplicative decrease during congestion events effectively penalizes misbehaving flows, promoting cooperation and preventing any single flow from dominating the bandwidth.
- Flexibility and Adaptability
    - AIMD can be adapted to different network conditions and requirements through parameter adjustments.
        - Adjust “a” and “b” to a non-linear equation that takes into account a host’s current congestion window size **(Diagram 4.1 to 4.8)**.
        - As in the case of Scalable TCP, the value of “b” is increased from ½ to ⅞ which means that each packet loss decreases the congestion window by a small fraction. However, the value of “a” reduces from 1 to 1 per 100 successful acknowledgements (0.01) **(Diagram 5.1 and 5.2)**.
        - Adjust “a” and “b” values for different hosts. If hosts do not share the same “a” and “b” values, the hosts will not have equal bandwidth allocation, but rather proportionate shares of it **(Diagram 6.1)**.

## 2.2 AIMD params change results

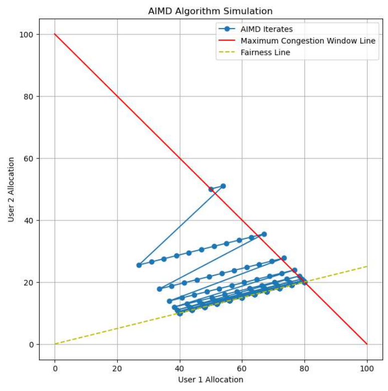
| With reference to diagram folder on github | Type, [a,b] | Number of Round-Trip time | Number of Hosts | Network maximum congestion window | Initial congestion window,e.g. [Host 1, …, Host x] | Additive increase parameter a | Multiplicative decrease factor b | Observation & Insights |
|---|---|---|---|---|---|---|---|---|
| Diagram 4.1 | [Exponential, Linear] | 75 | 2 | 100 | [1,100] | 2 * power(current congestion window, 1/3) | 0.3 (set low to observe exponential increase) | Rate of “a” increases as the current congestion window is high. Throughput is likely low as allocated bandwidth quickly reaches the congestion limit |
| Diagram 4.2 | [Exponential, Linear] | 75 | 10 | 1000 | [20,40,60…,200 ] | 2 * power(current congestion window, 1/3) | 0.3 (similar to 4.1) | Similar to 4.1. Hosts converge to fair line quickly due to higher “a” |
| Diagram 4.3 | [log, Linear] | 75 | 2 | 100 | [1,100] | 2 * log(current congestion window + 1) | 0.3 (set low to observe logarithmic increase) | Rate of “a” decreases as the current congestion window is high. Throughput is likely low as allocated bandwidth quickly reaches the congestion limit |
| Diagram 4.4 | [log, Linear] | 75 | 10 | 1000 | [20,40,60…,200 ] | 2 * log(current congestion window + 1) | 0.3 (similar to 4.3) | Similar to 4.3. Hosts converge to fair line quickly due to higher “a” |
| Diagram 4.5 | [inverse-log, Linear] | 100 | 2 | 100 | [1,100] | 5 / log(current congestion window + 1) | 0.5 | Rate of “a” is high when congestion window is low. Rate decreases as congestion window increases. |
| Diagram 4.6 | [inverse-log, Linear] | 300 | 10 | 1000 | [10,20,30,...,100 ] | 5 / log(current congestion window + 1) | 0.5 | Similar to 4.5 |
| Diagram 4.7 | [linear,log] | 250 | 2 | 100 | [1,100] | 1 | 1 - (1 / math.log(cwnd + 10)) | “b” value is only used at max congestion window before packet loss occurs. “b” value is close 1 when congestion window is large |
| Diagram 4.8 | [linear,log] | 650 | 10 | 1000 | [10,20,30,...,100 ] | 1 | 1 - (1 / math.log(cwnd + 10)) | Similar to 4.6 |
| Diagram 5.1 | Scalable TCP (linear) | 150 | 2 | 100 | [32,64] (assume slow start, host 1 | 1 (Up from 0.01 for better visualisation) | 7/8 | In standard TCP, congestion window is increases by (1/cwnd), but scalable TCP |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | started late by 1 unit time) | | | increases by 0.01. For simplicity both are scaled to 1. High throughput especially for high speed networks |
| Diagram 5.2 | Scalable TCP | 500 | 10 | 1000 | [10,20,30,...,100 ] | 1 | 7/8 | Similarly to 5.1. |

**<Figure 3>**: Table showing parameters "a" and "b" taking into account of current congestion window

## 2.3 AIMD fairness with Perron-Frobenius Eigenvector

Fairness in TCP aims to ensure competing flows share the available bandwidth equitably. This does not necessarily mean equal bandwidth allocation, but rather proportionate shares based on factors like round-trip time, flow duration, or application needs. For conventional AIMD, different hosts can be given different "a" and "b" parameters. This causes different hosts to get proportionate shares based on a convergence point. This convergence point can be deduced from the Perron-Frobenius Eigenvector. The entries in the Perron-Frobenius Eigenvector represent the steady-state probabilities of each flow being in different congestion window states. If all entries are equal, it suggests perfect fairness in the long run, meaning all flows will eventually reach the same average window size. With unequal entries, a larger spread between the entries suggests that some flows might have significantly higher or lower window sizes on average and vice versa.

| With reference to diagram folder on github | Type, [a,b] | Number of Round-Trip time | Number of Hosts | Network maximum congestion window | Initial congestion window,e.g. [Host 1, …, Host x] | Additive increase parameter a | Multiplicative decrease factor b | Observation & Insights |
|---|---|---|---|---|---|---|---|---|
| Diagram 6.1 | [linear ,linear ] | 250 | 2 | 100 | [50,50] | [4,1] | [0.5,0.5] |  Start both host at [50,50], Converges at Host 1 with a higher weight calculated by the Perron-Frobenius Eigenvector |

**<Figure 4>**: Table and Diagram showing proportional fairness between 2 Hosts

The convergence point is the intersection point between the maximum available bandwidth before packet loss and the fairness line. The fairness line can be deduced via the power iteration method on the Perron-Frobenius Eigenvector. This is proven to converge to alpha/(1-beta). In the case above, Host 1 has a proportional weight of 8 while Host 2 has a proportional weight of 2. This results in Host 1 reaching a maximum of 80 without packet loss while Host 2 is left with 20 congestion window.

### 2.3.1 Perron-Frobenius Eigenvector Limitations

While the Perron-Frobenius Eigenvector is simple to calculate, it overlooks other factors and has these limitations:

- The Perron-Frobenius analysis solely reflects the long-term behavior of the system and doesn't capture transient fairness or short-term fluctuations.
- It assumes all flows have identical characteristics (e.g., round-trip time), which might not hold true in real-world scenarios.
- It doesn't account for different fairness definitions or considerations like max-min fairness or proportional fair share.

Hence, while the Perron-Frobenius analysis can provide valuable qualitative insights into fairness tendencies, it should combined with other fairness metrics and knowledge of the network conditions for a thorough fairness evaluation.

### 2.4 AIMD Limitations

While AIMD is a widely used and generally effective congestion control algorithm for TCP, it does have some limitations that researchers are actively exploring:

- **Slow Start**
  - The initial slow start phase can be inefficient, especially for high-speed networks, leading to underutilization of bandwidth.
- **Bursty Behavior**
  - AIMD's sawtooth pattern of window size adjustments can cause bursty traffic, potentially impacting network stability and application performance.
- **Limited Fairness**
  - While AIMD promotes some level of fairness, it might not be ideal for scenarios with diverse flow characteristics or prioritize specific flows
- **Not Adaptive**
  - AIMD reacts passively to congestion events, limiting proactive congestion avoidance capabilities

To cater to high-speed networks such as data centers, research efforts have developed alternative algorithms like CUBIC, BBR, and HyStart++ that address these limitations of AIMD, aiming for higher throughput, improved fairness, and faster congestion response. For example, in CUBIC algorithm, the window size is dependent on the last congestion event and not RTT. Moreover, due to the uprising of Artificial Intelligence (AI) in recent times, researchers are starting to leverage machine learning for congestion prediction and control for more proactive and adaptive congestion control mechanisms.

## 3. Conclusion

Despite AIMD prevalence in TCP/IP networks, high-speed networks nowadays have adopted other algorithms, instead of simply modifying AIMD parameters, in consideration of their trade-offs in specific network environments and application requirements. Additionally, it is also possible to improve the network performance through other means. Network optimizations like reducing latency, increasing bandwidth, and optimizing routing can significantly improve performance without necessarily modifying TCP itself. Dynamic traffic management techniques like traffic shaping and congestion avoidance can also contribute to efficient network utilization and congestion control.

### 4. Extra AIMD Analysis

**Diagram 7.1:** Illustrates a change in max congestion window at RTT = 400.
**Diagram 7.2:** Illustrates new users entering the network traffic at RTT = 600.