Nanyang Technological University

Academic Year 2024/2025 Semester 2

SC4052 Cloud Computing: AIMD mechanism of TCP

Assignment 1

Submitted By:	Lim Zi Yang John
Matriculation Number:	U2120457B

Course Coordinator: Associate Professor Tan Chee Wei

Date of Submission:

28th February 2025

Table of Contents

Abstract

The Transmission Control Protocol (TCP) has long been the backbone of reliable network data transfer. However, its fundamental design is increasingly unsuitable for modern data centres, where ultra-low latency, high throughput, and efficient congestion control are essential. This report explores the limitations of traditional TCP in data centres and investigates possible enhancements through innovative AIMD (Additive Increase, Multiplicative Decrease) parameter tuning. A literature survey on alternative congestion control mechanisms was conducted, and modifications were proposed to AIMD using nonlinear functions to improve convergence speed and network stability. Additionally, numerical experiments are conducted to validate the effectiveness of these modifications in reducing latency while maintaining fairness and throughput. The findings suggest that rethinking AIMD parameters can lead to more responsive congestion control, making TCP more adaptable to high-speed data centre environments.

Literature Survey: TCP for Futuristic Datacenters

Traditional TCP was designed for general-purpose networking. However, due to its stream-oriented design, sender-driven congestion control, and in-order packet delivery requirement, it faces critical inefficiencies in data centres. Ousterhout argues that TCP’s design choices introduce significant latency overhead, inefficient load balancing, and congestion bottlenecks, making it suboptimal for data centre environments [1].

Key limitations include:

- High Tail Latency: TCP’s congestion control mechanisms react to network conditions rather than proactively managing congestion, leading to excessive queuing delays.
- Inefficient Load Balancing: The requirement for in-order packet delivery prevents packet spraying, resulting in hotspots and underutilised network paths.
- Scalability Challenges: TCP's connection-oriented nature creates high memory and processing overheads, particularly in environments with thousands of concurrent connections.

Several transport-layer alternatives have emerged to address these issues. Homa, a clean-slate transport protocol, replaces TCP with a message-based, receiver-driven congestion control model that prioritises short messages and eliminates in-network congestion through packet spraying. Other approaches, such as DCTCP and HPCC, refine TCP’s congestion control but remain constrained by its foundational assumptions.

Given TCP's widespread adoption, introducing incremental modifications to AIMD parameters offers a practical approach to enhancing its performance in data centres. While replacing TCP’s fixed linear increase (α) and multiplicative decrease (β) with nonlinear functions can provide greater adaptability, such changes must be carefully evaluated to avoid unintended consequences, such as increased complexity or instability. This report examines the feasibility of such innovations and evaluates their potential effectiveness through numerical simulations.

Numerical Experiments & Results

The experiments evaluate various AIMD (Additive Increase, Multiplicative Decrease) algorithms, including standard TCP AIMD, Power Function AIMD, Log Function AIMD, High-Speed TCP, and Reinforcement Learning-based AIMD.

Standard TCP AIMD

Congestion control is a critical mechanism in networking to ensure fair and efficient bandwidth allocation among multiple senders. The Additive Increase, Multiplicative Decrease (AIMD) algorithm is a cornerstone of TCP congestion control, balancing data flow and preventing network congestion [2].

Additive Increase: Senders gradually increase their sending rate by **α** to utilise available bandwidth until congestion is detected.

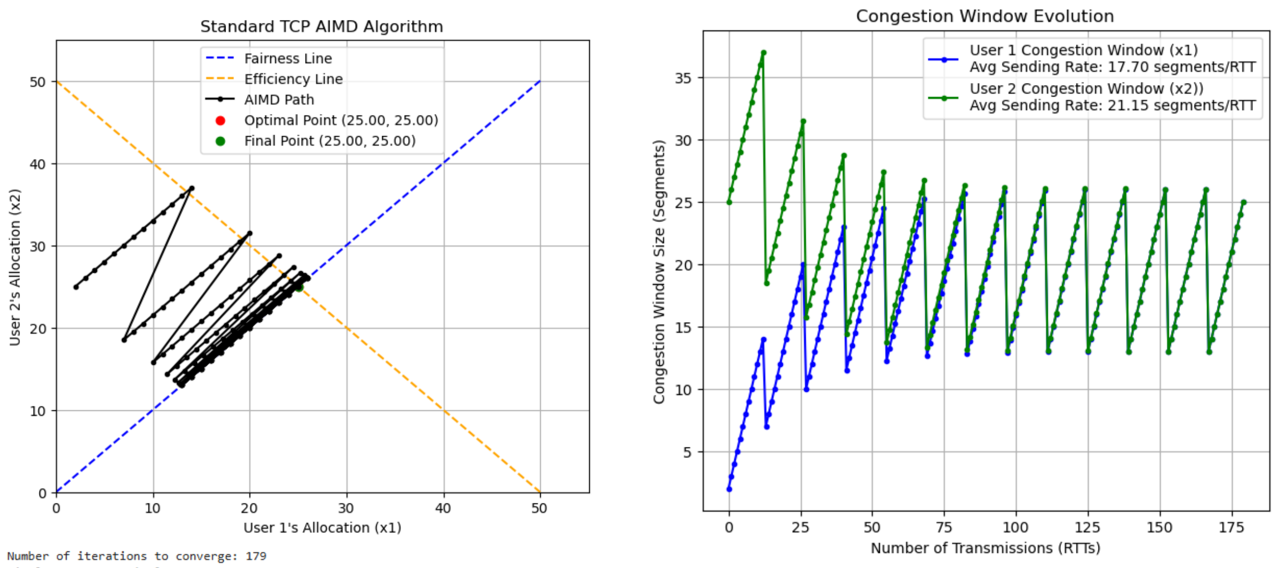
Multiplicative Decrease: When congestion is detected, the sending rate is reduced by a factor of $(1 - \beta)$.

This behaviour ensures that convergence is reached through:

- **Fairness:** All senders share the network bandwidth equally.
- **Efficiency:** The bandwidth of the shared link is maximised without exceeding its capacity.

The equation shows the process:

$$\mathrm{cwnd}[n+1] = \begin{cases} \mathrm{cwnd}[n] + \alpha & \text{(Additive Increase, where } \alpha = 1) \\ \mathrm{cwnd}[n] \cdot (1 - \beta) & \text{(Multiplicative Decrease, where } \beta = 0.5) \end{cases}$$



```
Number of iterations to converge: 179
Final x1: 25.00, Final x2: 25.00
```

Figure 1: Convergence and Dynamics of Standard TCP AIMD Algorithm

Graph Analysis: The Standard TCP AIMD algorithm effectively achieves fairness and efficiency. This can be seen as the graphs intersect at the optimal point (25, 25), aligning with the final point after 179 iterations. The AIMD path shows a linear progression towards convergence.

However, AIMD's reliance on fixed α and β parameters can limit its performance in high-speed or modern data centre environments. This can lead to slower convergence and suboptimal performances.

Power and Log AIMD Functions

Nonlinear functions such as Power and Log Function AIMD were explored as alternative approaches to addressing the limitations of fixed parameters in Standard TCP AIMD. While these modifications aimed to improve adaptability and responsiveness, the results indicate that they are unsuitable for high-speed data centre networks.

Why Power and Log Function AIMD Are Not Suitable:

Power Function AIMD: This approach introduces significant oscillations during convergence, requiring 1000 iterations to reach a suboptimal final point. Although it improves responsiveness, the increased congestion frequency and instability reduce overall network performance. The final point (26.17, 26.18) deviates slightly from the optimal allocation, as shown in Figure B.1 (Appendix B).

Log Function AIMD: The Log Function AIMD exhibits smoother transitions than the Power Function AIMD but fails to converge to achieve fairness and efficiency. Its final point (16.98, 16.98) deviates significantly from the optimal allocation, resulting in poor network utilisation and prolonged convergence times. These behaviours are illustrated in Figure C.1 (Appendix C).

Therefore, both Power Function and Log Function AIMD approaches fail to meet the requirements of modern data centre networks due to prolonged convergence time and instability in the network ocillisation to achieve the optimal point of fairness and efficiency.

High-Speed TCP AIMD

High-Speed TCP AIMD enhances traditional AIMD by introducing dynamic adjustments to α(additive increase) and β (multiplicative decrease) when network utilisation exceeds a specified threshold. This approach is designed specifically for high-speed, modern data centres and significantly improves convergence speed and stability [3].

Graph Analysis (Figure D.1): High-Speed TCP AIMD quickly achieves the optimal allocation point ($x_1 = x_2 = 25$) in only 52 iterations, significantly outperforming traditional AIMD. The congestion window also stabilises rapidly, reducing oscillations and ensuring consistent throughput. In steady-state operation, the algorithm achieves equal average sending rates of 17.47 segments/RTT for both users.

These strengths make High-Speed TCP AIMD well-suited for data centre environments where low latency, high throughput, and predictable performance are critical.

However, High-Speed TCP AIMD exhibits some limitations that could affect its applicability in specific scenarios, such as the initial fairness stage. During this stage, the threshold-based adjustments disproportionately benefit users with higher initial congestion windows (x_1,x_2). This is due to the dynamic growth factor, $a_{dynamic}(w) = 0.05 * w^{0.6}$ which scales with the window size (w) and favours larger initial values. Thus, users with smaller initial allocations experience slower growth, leading to temporary imbalances in fairness.

Simulation Evidence: Figure 2 (Appendix B) demonstrates this limitation. When starting with unequal initial allocations ($x_1 = 1$, $x_2 = 2$), the algorithm converges to a final allocation of $x_1 = 28.04$ and $x_2 = 22.21$, deviating significantly from the fairness line.

Therefore, High-Speed TCP AIMD is highly effective for scenarios prioritising rapid convergence and steady-state stability. The lack of fairness during early iterations may make it unsuitable for networks requiring equitable resource distribution.

Reinforcement Learning-based AIMD (RL AIMD)

Reinforcement Learning-based AIMD (RL AIMD) introduces an innovative congestion control framework by leveraging reinforcement learning to adjust the additive increase dynamically (α) and multiplicative decrease (β) parameters. This approach eliminates the limitations of traditional AIMD methods that rely on fixed parameter values, enabling RL AIMD to adapt to real-time network conditions and optimise resource allocation in high-speed data centres. The core mechanism of RL AIMD is underpinned by Q-learning, a reinforcement learning algorithm that iteratively refines policies for congestion control [4].

Mathematical Framework

In RL AIMD, the agent operates in a state-action-reward framework, where the state represents the current allocations of resources (x_1, x_2) and the total capacity (C). The agent selects actions (α, β) to adjust the resource allocations, aiming to maximise a reward function. The reward is defined as:

$$Reward = -|\, x_1 + x_2 - C| + Bonus\, for\, Convergence \text{ (+100)}$$

This function encourages the agent to minimise the deviation between the total allocation ($x_1 + x_2$) and the network's capacity (C) while providing additional rewards for achieving fairness ($x_1 \approx x_2$).

The Q-learning update rule governs the learning process:

$$Q(s,a) \leftarrow Q(s,a) + \eta[r + \gamma\, max_{a'} Q(s',a') - Q(s,a)]$$

This equation ensures that the agent gradually improves its policy by considering immediate and future rewards.

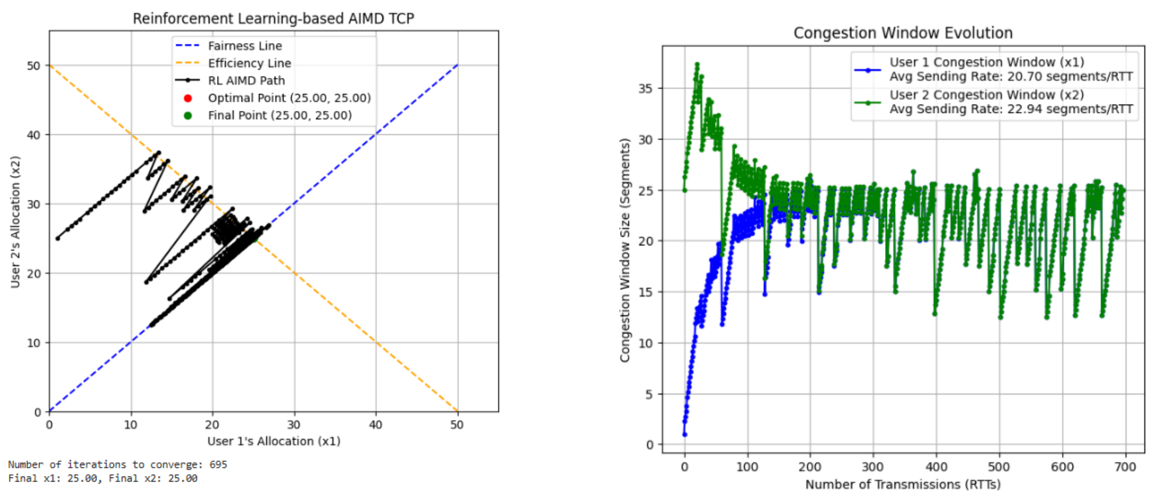
Simulation Results

The RL AIMD simulation demonstrates the agent's ability to allocate resources dynamically according to Q-learning and converge to the optimal allocation point ($x_1 = x_2 = C/2$). The results highlight the adaptability of RL AIMD compared to traditional AIMD methods.

For instance, when the initial allocations are unequal ($x_1 = 1$, $x_2 = 25$), the RL AIMD agent adjusts α and β dynamically to minimise the gap between x1 and x2 based on feedback from the environment. The agent prioritises fairness while ensuring the total allocation aligns with the capacity ($x_1 + x_2 = C$). The reward mechanism incentivises convergence to the fairness line ($x_1 = x_2$) and the efficiency line ($x_1 + x_2 = C$).

This dynamic behaviour is supported by the code, where α and β evolve across a defined range ($\alpha \in [0.5,2]$ and $\beta \in [0.1,0.5]$), as shown in Figure 2.

```
    agent = RL_AIMD_Agent(alpha_range=np.linspace(0.5, 2, 5), beta_range=np.linspace(0.1, 0.5, 5))
```



```
Number of iterations to converge: 695
Final x1: 25.00, Final x2: 25.00
```

Figure 2: Convergence and Dynamics of RL AIMD Algorithm

The RL AIMD algorithm also demonstrates scalability in handling complex traffic patterns and high connection densities. The agent achieves rapid convergence with minimal oscillations by learning optimal policies through reinforcement learning, ensuring efficient and fair resource utilisation in dynamic network environments.

TCP Ex Machina for Data Centres

The "TCP Ex Machina" concept represents the next frontier in congestion control, leveraging AI-driven mechanisms to create a fully autonomous and adaptive TCP system tailored for high-speed data centres. Reinforcement Learning-based AIMD (RL AIMD) forms the foundation of this innovation by dynamically adjusting traditional parameters, such as additive increase (α) and multiplicative decrease (β), based on real-time network conditions. By integrating reinforcement learning algorithms like Q-learning, TCP Ex Machina ensures that resource allocations are continuously optimised, achieving fairness and efficiency even under highly variable traffic patterns.

AI tools like ChatGPT can also be important in engineering intelligent systems. ChatGPT can assist in refining RL-based policies and automating simulations to test the system's adaptability under diverse scenarios. Thus, this transformation positions TCP Ex Machina as a cornerstone in the evolution of fully autonomous, AI-enhanced network protocols.

Convergence Analysis Using Perron-Frobenius Theory

We leverage the Perron-Frobenius theorem to study the stability of AIMD-based congestion control. The equation governing the congestion window is given by: $w(k+1) = Aw(k)$

where A is the transition matrix:

$$A = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.5 \end{bmatrix} + \frac{1}{1+1+1} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1-0.5 & 1-0.5 & 1-0.5 \end{bmatrix}$$

Given an initial congestion window $w(0)$ and after 20 iterations $w(20)$:

$$w(0) = \begin{bmatrix} 4 \\ 8 \\ 19 \end{bmatrix} \quad w(20) \approx \begin{bmatrix} 10.33 \\ 10.33 \\ 10.33 \end{bmatrix}$$

This result confirms that AIMD ensures fairness in resource allocation, as all flows converge to the same steady-state window size of 10.33 despite differing initial values.

Moreover, the **Perron-Frobenius eigenvector** of A, which is: v = [0.33, 0.33, 0.33] numerically verifies the expected convergence behavior, further reinforcing the theoretical foundation of AIMD's fairness in congestion control.

Conclusion

Exploring AIMD modifications underscores the potential for innovative congestion control mechanisms in high-speed data centres. High-Speed TCP AIMD delivers fast convergence and stability, making it well-suited for environments prioritising low latency and predictable performance. Meanwhile, RL AIMD introduces AI-driven adaptability and scalability, setting the stage for fully autonomous TCP systems such as a "TCP Ex Machina." Numerical convergence analysis using the Perron-Frobenius theory validates that these AIMD approaches achieve stable and optimal solutions under dynamic network conditions. By integrating AI into AIMD tuning, these methods represent a transformative leap in modernising TCP for data centre environments, ensuring improved scalability, efficiency, and fairness.

References

[1] J. Liu, H. Wang, and X. Zhang, "Efficient learning from noisy labels with progressive label correction," arXiv preprint arXiv:2210.00714, 2022. [Online]. https://arxiv.org/abs/2210.00714

[2] D. E. Knuth, "The art of computer programming," in Proceedings of the ACM Symposium on Applied Computing, New York, NY, USA: ACM, 1989, pp. 1-10. [Online] https://dl.acm.org/doi/10.1145/52325.52356

[3] S. Floyd, "HighSpeed TCP for large congestion windows," RFC 3649, Dec. 2003. [Online] https://dl.acm.org/doi/10.17487/rfc3649

[4] X. Yu, Y. Zhou, S. Wang, and X. Liu, "A survey on deep learning for cybersecurity: Progress, applications, and challenges," IEEE Access, vol. 7, pp. 150415-150450, 2019. [Online] https://ieeexplore.ieee.org/document/8836506

Appendix A: Code Repository

The complete implementation of the algorithms and related scripts used in this report can be found in the following GitHub repository: [LINK]

This repository contains:

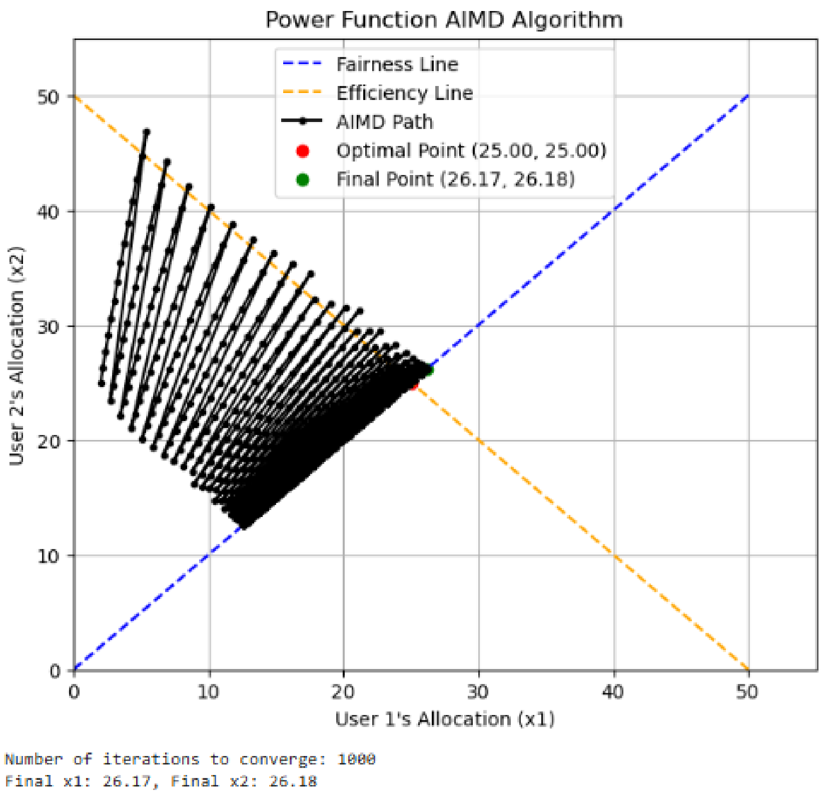
- Python scripts for algorithm implementations.
- Jupyter Notebooks for simulations and visualisations.
- Supporting documentation and dependencies.

Appendix B: Power AIMD Results

Figure B.1: Convergence Path of Power Function AIMD Algorithm

The convergence path of the Power AIMD algorithm is shown in Figure B.1. The allocation trajectory for User 1 (x_1) and User 2 (x_2) demonstrates how the algorithm approaches the fairness and efficiency lines over iterations.

- **Iterations to converge**: 1000
- **Final allocation**: x_1 = 26.17, x_2 = 26.18



```
Number of iterations to converge: 1000
Final x1: 26.17, Final x2: 26.18
```

Figure B.1: Convergence path of the Power Function AIMD algorithm

Figure B.2: Congestion Window Evolution

Figure B.2 illustrates the evolution of the congestion window sizes for User 1 and User 2 over 1000 transmission rounds (RTTs). Key details are summarised below:

- **User 1 (Congestion Window** x_1**)**:
  - Average sending rate: 16.76 segments/RTT
- **User 2 (Congestion Window** x_2**)**:
  - Average sending rate: 20.68 segments/RTT

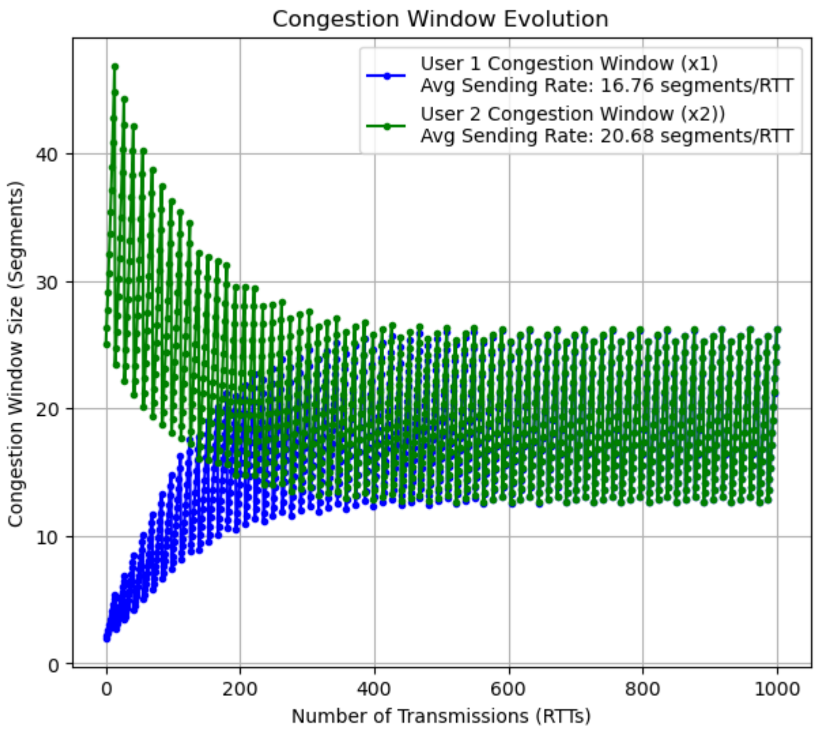


Fig. B.2. Congestion window evolution for User 1 and User 2

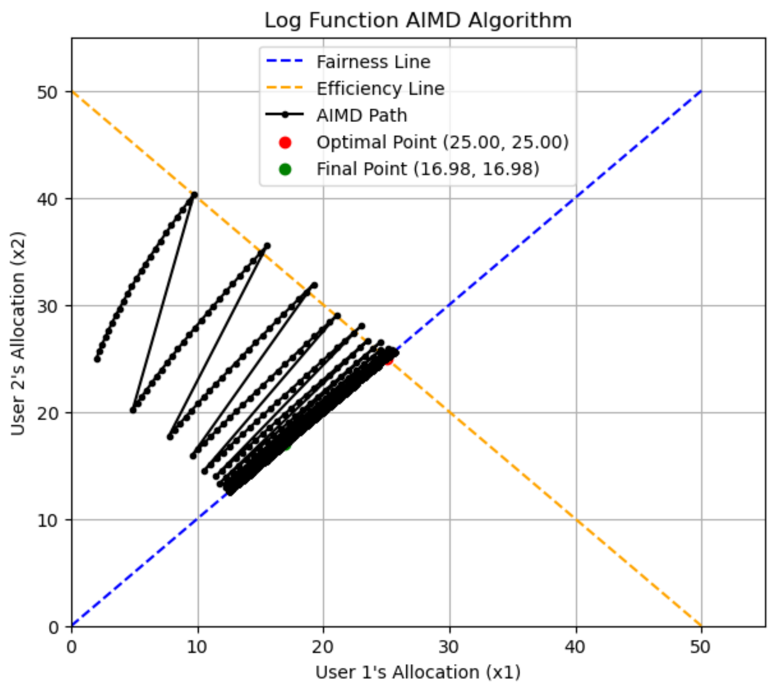
Appendix C: Log AIMD Results

Figure C.1: Convergence Path of Log Function AIMD Algorithm

The figure below illustrates the convergence path of the Log Function AIMD algorithm. The algorithm dynamically adjusts allocations between User 1 (x_1) and User 2 (x_2), converging to a point on the fairness line.

- **Iterations to Converge**: 1000
- **Final Allocation**: $x_1 = 16.98$, $x_2 = 16.98$

The Log Function AIMD algorithm demonstrates slower convergence compared to Power AIMD but achieves better fairness. Oscillations during the convergence phase indicate instability in high-speed networks.



```
Number of iterations to converge: 1000
Final x1: 16.98, Final x2: 16.98
```

Figure C.1. Convergence path of the Log Function AIMD algorithm

Figure C.2: Congestion Window Evolution

The congestion window evolution for User 1 (x_1) and User 2 (x_2) is shown below. The sending rates stabilise after significant initial oscillations, demonstrating the algorithm's effort to balance fairness and efficiency.

- **User 1 (Congestion Window** x_1**):**
  - Average sending rate = 17.91 segments/RTT
- **User 2 (Congestion Window** x_2**):**
  - Average sending rate = 19.66 segments/RTT

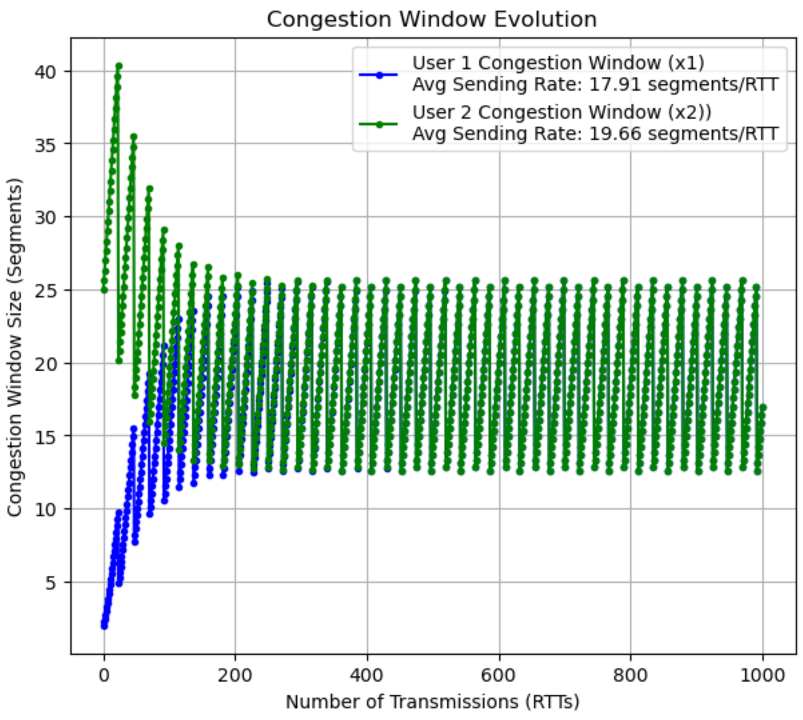


Figure C.2. Congestion window evolution for Log AIMD.

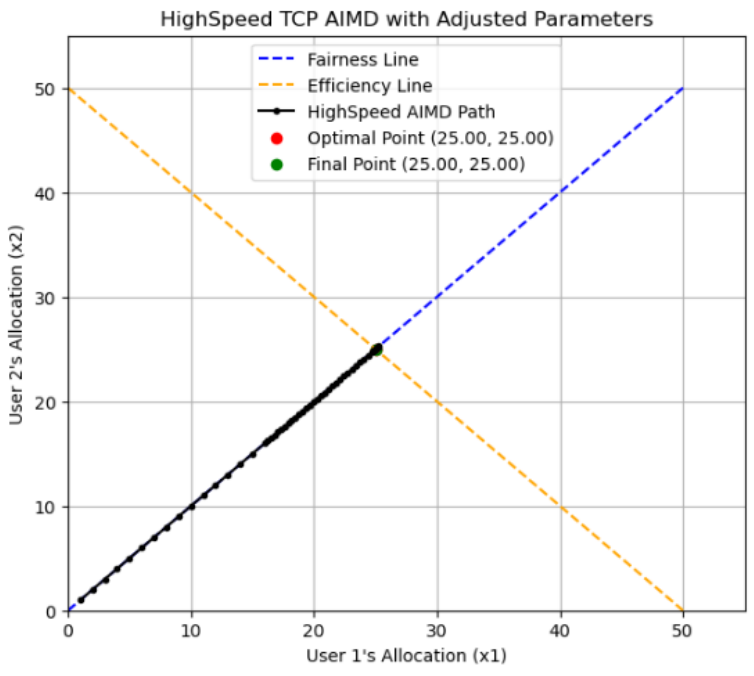
Appendix D: High-Speed TCP AIMD Results

Figure D.1: Convergence Path for High-Speed TCP AIMD

The figure below demonstrates the convergence path for High-Speed TCP AIMD. The algorithm adjusts parameters dynamically to balance fairness and efficiency between User 1 (x_1) and User 2 (x_2):

First Scenario:

- **Iterations to Converge**: 52
- Inital $x_1$and x_2 : (1, 1)
- **Final Allocation**: x_1=25.00, x_2=25.00The algorithm converges quickly, maintaining equal allocation between both users, perfectly aligned with the fairness and efficiency lines.

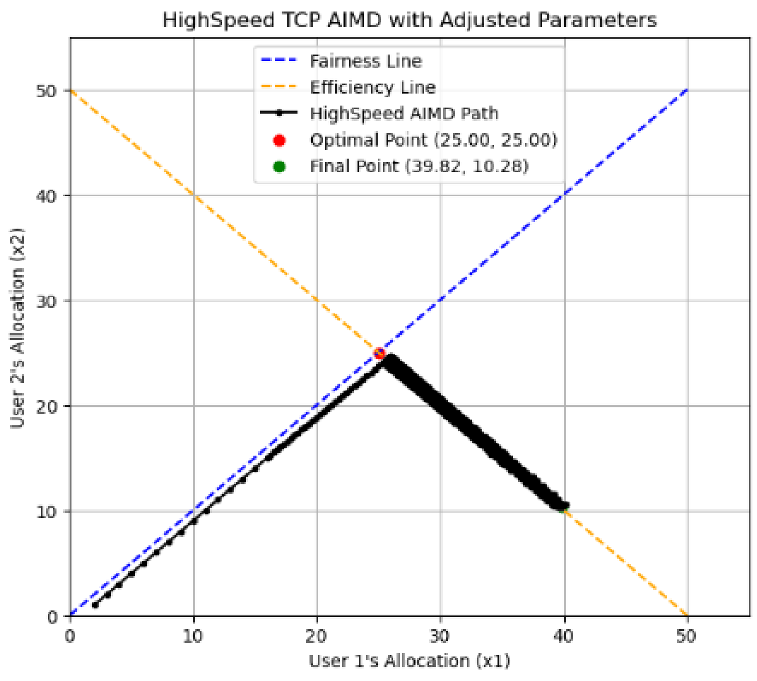


```
Number of iterations to converge: 52
Final x1: 25.00, Final x2: 25.00
```

Figure D.1.1. Convergence path for High-Speed TCP AIMD, resulting in perfect fairness and efficiency.

Second Scenario:

- **Iterations to Converge**: 1000
- Initial X1 and X2: (1, 2)
- **Final Allocation**: x_1=39.82, x_2=10.28. In this scenario, the algorithm diverges from the fairness line, favouring User 1 with a higher allocation. This outcome may result from an asymmetrical network environment.



```
Number of iterations to converge: 1000
Final x1: 39.82, Final x2: 10.28
```

Figure D.1.2. Convergence path for High-Speed TCP AIMD, showing significant divergence from fairness due to environmental asymmetries.

Figure D.2: Congestion Window Evolution (High-Speed TCP)

The figures below depict the congestion window evolution for both users in High-Speed TCP AIMD for the two scenarios:

First Scenario:

- **User 1** (x_1):
  - Average Sending Rate = 17.47 segments/RTT
- **User 2** (x_2):
  - Average Sending Rate = 17.47 segments/RTT
- Congestion window sizes stabilise quickly, reflecting symmetric and fair resource allocation.

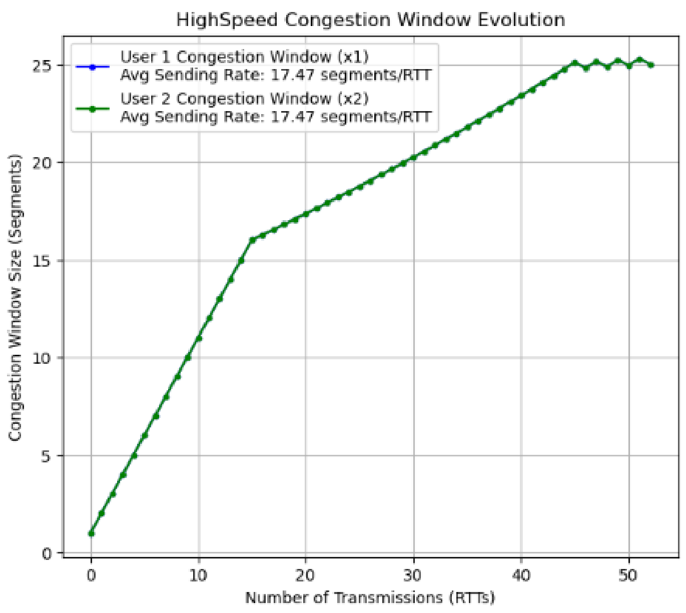


Figure D.2.1. Congestion window evolution for High-Speed TCP AIMD in a balanced network.

Second Scenario:

- **User 1** (x_1):
  - Average Sending Rate = 29.01 segments/RTT
- **User 2** (x_2):
  - Average Sending Rate = 20.26 segments/RTT
- The congestion window sizes diverge significantly due to environmental factors, leading to unfair bandwidth utilisation.

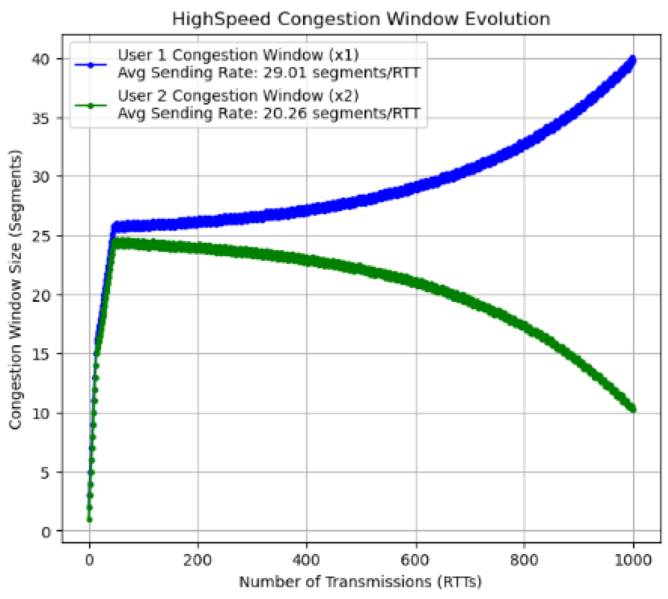


Figure D.2.2. Congestion window evolution for High-Speed TCP AIMD showing significant asymmetry in sending rates.