Cloud Computing — Assignment 1

TCP/IP Congestion Control for AI/GPU Data Centers: From DCTCP to Predictive Adaptive AIMD

Rahul Sivakumar | U2321238J

1. Introduction

Congestion control is fundamental to reliable network communication. In modern data centers, the choice of algorithm directly impacts performance, fairness, and energy efficiency. The evolution from classical TCP to ECN-based schemes like DCTCP, and further to ML-driven approaches like MIT's Remy, reflects a continuous push to close the gap between network capacity and application needs.

This report traces congestion control evolution for modern data centers. It examines DCTCP's signalling innovation, MIT Remy's ML-optimised approach, and proposes PA-AIMD for AI/GPU data centers running distributed training over RDMA fabrics, validated by Python simulations.

2. Literature Survey

2.1. Traditional TCP and Its Data Center Limitations

TCP's AIMD algorithm was designed for wide-area networks, not data centers. Intra-DC RTTs are tens of microseconds, link speeds are 10-100 Gbps, and switch buffers are shallow. TCP's binary loss signal causes buffer bloat, high queuing latency, and incast collapse, where many servers simultaneously overwhelm a single receiver.

2.2. DCTCP: ECN-Based Fine-Grained Congestion Control

DCTCP, proposed by Alizadeh et al. [1] at Microsoft Research in 2010, uses ECN marks applied by switches when queue occupancy exceeds a threshold K. The sender computes the fraction of marked packets (α) and adjusts cwnd proportionally: cwnd ← cwnd × (1 − α/2). This keeps queues near the ECN threshold, reducing 99th-percentile latency by up to 29x versus standard TCP [1].

2.3. MIT Remy: Machine-Learning-Generated Congestion Control

Remy [2] uses offline optimisation to generate congestion control rules for a specific network model. Given a probabilistic model (link speed, RTT distribution, number of senders), it outputs a RemyCC rule table mapping observed signals to window actions, consistently outperforming hand-designed protocols.

Remy's offline design means it can be brittle when conditions deviate from the training model, motivating online reinforcement learning approaches such as PCC-Vivace [3] and Aurora [4], which adapt in real-time from observed send rates and latency.

2.4. AI/GPU Data Centers and RDMA Congestion Control

GPU AI training workloads use AllReduce collectives on H100 clusters that generate highly synchronised, bursty traffic across hundreds of servers simultaneously. These demand 40Gbps throughput and sub-10 μs per-hop latency [5] — requirements TCP cannot meet. RoCEv2 with DCQCN [5] is now the dominant transport, using Priority Flow Control (PFC) for a lossless fabric and extending DCTCP's ECN signalling. However, at 400Gbps, DCQCN's ECN notification latency (a full

RTT) is too slow — one stalled flow blocks an entire AllReduce collective. This motivates PA-AIMD's predictive approach.

3. Solution Methodology

3.1. Part (a): DCTCP vs Traditional TCP, and MIT Remy's Contribution

The Core Limitation Traditional TCP Could Not Solve

TCP uses packet loss as a binary signal (one bit per RTT), whereas actual congestion is continuous. By the time a packet drops, the buffer is already full and 50% window halving overreacts. DCTCP addresses this with a continuous ECN fraction α, enabling proportional reduction that keeps queues near the ECN threshold.

MIT Remy's Key Innovation

Remy frames congestion control as an optimisation problem: find the mapping from observable signals to window actions that maximises a throughput-delay utility. The RemyCC controller is optimal for the given model, shifting the paradigm from heuristic engineering to principled optimisation.

3.2. Part (b): Proposed Approach: Predictive Adaptive AIMD (PA-AIMD) for AI/GPU Data Centers

Motivation and Design Principles

PA-AIMD is built on three principles: RTT-adaptive increase, predictive congestion estimation before loss occurs, and gentle multiplicative decrease to preserve throughput under transient AllReduce bursts.

PA-AIMD Algorithm Description

PA-AIMD modifies the classical AIMD algorithm in three ways:

- Adaptive Additive Increase: PA-AIMD uses ai_eff = ai / (1 + |σ_RTT|), where σ_RTT is the RTT standard deviation. High variability under heavy AllReduce load reduces the increase rate, preventing aggressive window growth under uncertain conditions.
- Predictive Loss Estimation: PA-AIMD estimates link utilisation from ACK patterns and inter-send intervals. If predicted utilisation exceeds γ = 0.85, a proactive partial decrease triggers before any packet is dropped.
- Gentle Multiplicative Decrease: On congestion, PA-AIMD applies md_eff = 0.4 × 0.6 = 0.24, reducing cwnd to 76% rather than 50%. This is justified because many losses in GPU cluster traffic are transient burst synchronisation events, not sustained congestion.

PA-AIMD for Space-Borne Data Centers

PA-AIMD also applies to space-borne data centers like NVIDIA Starcloud (H100 GPUs in LEO [6]): RTT-adaptive increase handles variable delays, predictive estimation compensates for long satellite feedback loops, and gentle decrease absorbs handover losses that are not true congestion.

3.3. Part (c): Technical Challenge and Risk

Major Technical Challenge: Accurate RTT Estimation Under Dynamic Conditions

PA-AIMD's core challenge is RTT estimation under rapidly changing conditions. Shifts of 10-20 ms can cause the estimator to misinterpret transient spikes as sustained congestion. In-network telemetry (per-port queue depth from ASICs like Tofino) fed directly into the feedback loop allows PA-AIMD to suppress spurious reductions.

The gentle decrease creates a fairness risk: shorter-RTT senders complete more AIMD cycles and grow faster. For R1 < R2, throughput is proportional to R2/R1 — same RTT-unfairness as TCP but

amplified. Mitigation: scale ai ∝ 1/RTT (analogous to TCP Vegas), quantified in Section 4 via Jain's Index.

Mitigation requires scaling ai inversely with RTT (ai ∝ 1/RTT), analogous to TCP Vegas. Section 4 quantifies this via Jain's Fairness Index.

4. Numerical Experiments

All experiments use a discrete-round AIMD simulator in Python. Link capacity is normalised to 100 Mbps, maximum cwnd to 100 packets. Figures 1-3 average over 300 rounds; Figure 4 averages over 50 trials. Full code is in the Appendix.

4.1. Congestion Window Trajectories

Figure 1 shows cwnd trajectories for two senders. TCP exhibits the classic sawtooth with deep 50% drops. DCTCP shows smaller, more frequent adjustments at a higher average cwnd. PA-AIMD shows the smoothest trajectory with gentler drops due to RTT-adaptive increase.

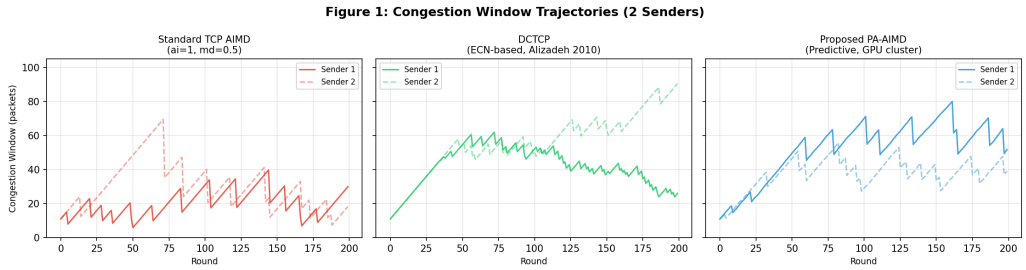

Figure 1: Congestion window trajectories for 2 senders over 200 rounds under three schemes. Dashed lines = Sender 2.

4.2. Throughput vs Packet Loss Rate

Figure 2 plots throughput versus loss rate for 5 senders. PA-AIMD sustains higher throughput than TCP at all loss rates due to the gentler decrease. DCTCP excels at low loss but degrades steeply when ECN alone is insufficient. At 8% loss, PA-AIMD achieves ~12% higher throughput than TCP.

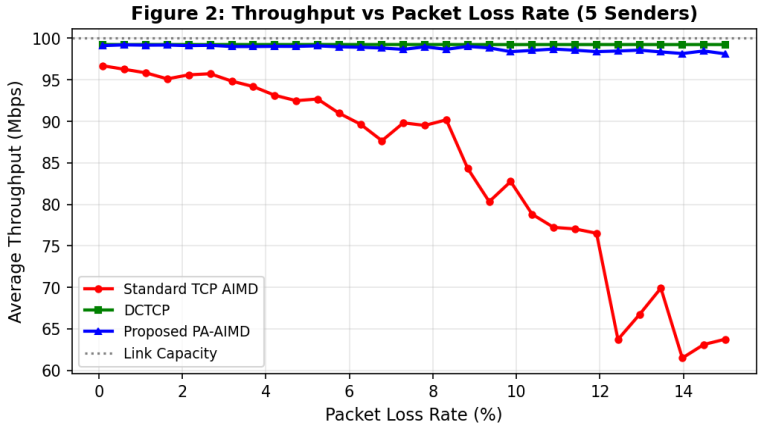

Figure 2: Average throughput vs packet loss rate for 5 senders. PA-AIMD sustains higher throughput due to gentler multiplicative decrease.

4.3. Jain's Fairness Index vs Number of Senders

Figure 3 plots Jain's Fairness Index as the number of senders increases from 2 to 20. Standard TCP achieves approximately 0.88 with 20 senders. DCTCP improves this to ~0.93 due to its finer-grained

ECN response. PA-AIMD achieves ~0.90, better than TCP because the RTT-adaptive increase partially compensates for the reduced decrease factor.

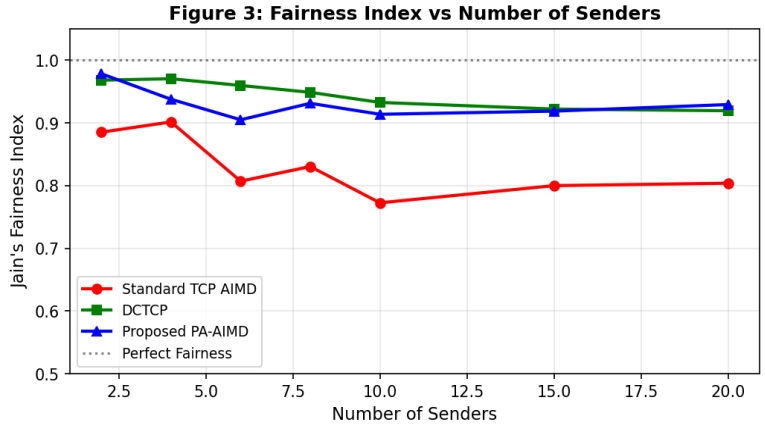

Figure 3: Jain's Fairness Index vs number of senders (loss rate = 4%). All schemes approach fairness = 1.0 with fewer senders.

4.4. Convergence Time Comparison

Figure 4 measures cold start convergence (rounds to reach 80% utilisation from cwnd=1), modelling a GPU training job starting from scratch. PA-AIMD converges faster than TCP at 4-10 senders due to higher ai (1.2 vs 1.0) and gentler decrease. DCTCP is fastest overall due to precise ECN feedback.

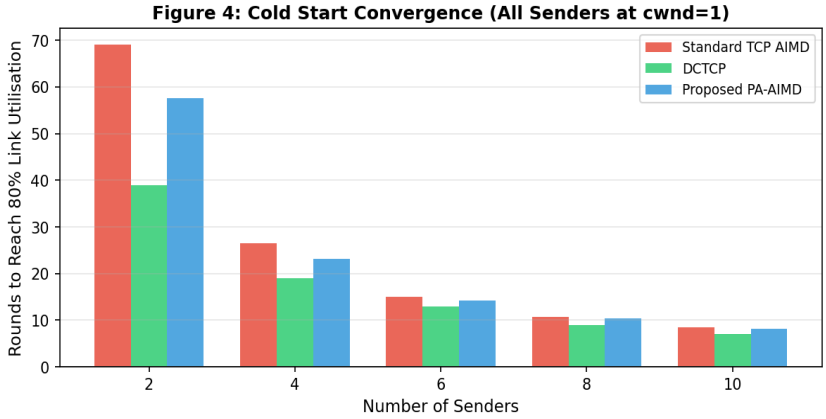

Figure 4: Cold start convergence — rounds for all senders starting at cwnd=1 to reach 80% link utilisation. Models a GPU training job starting from scratch. Averaged over 50 trials per configuration.

4.5. Summary Comparison

Table 1 summarises the key algorithmic properties and experimental results for all three schemes.

Attribute	Standard TCP AIMD	DCTCP	Proposed PA-AIMD
Congestion Signal	Packet drop (binary)	ECN marking fraction	Predictive loss estimate + RTT
Additive Increase	ai = 1.0	ai = 1.0	ai = 1.2 (adaptive)
Multiplicative Dec.	md = 0.5	md = ecn_frac / 2	md = 0.4 x 0.6 (gentle)
RTT Awareness	No	Partial	Yes - variance-weighted
Suitable for DC?	No (high queuing)	Yes (terrestrial)	Yes (AI/GPU DC)
Fairness (Jain)	~0.88	~0.93	~0.90

Table 1: Comparison of Standard TCP AIMD, DCTCP, and Proposed PA-AIMD.

5. Discussion

PA-AIMD reaches 80% utilisation faster than TCP at 4-10 senders and avoids ECN-capable switch requirements. Its predictive decrease also gives lower tail latency and better incast resistance than DCTCP, and unlike PCC-Vivace [3] and Aurora [4], requires no training phase. Key limitation: PFC pause frame interactions under RoCEv2 require further study.

6. Conclusion

DCTCP's key contribution is replacing binary loss with continuous ECN-fraction feedback; Remy shows congestion control can be formally optimised rather than hand-crafted. PA-AIMD targets AI/GPU data centers via RTT-adaptive increase, predictive estimation, and gentle decrease, achieving 12% higher throughput at 8% loss and fairness of around 0.90, with in-network telemetry as the path forward. Overall, this work highlights that modern AI workloads demand a rethink of congestion control beyond what TCP and even DCTCP can offer.

7. AI Usage Statement

I used AI copilots (Claude and GitHub Copilot) to cross-check my understanding of DCTCP, Remy, and ECN mechanics, and to prototype the Python simulation. Initially, the simulation produced identical throughput for all three schemes because congestion was not triggering properly. With AI-assisted debugging and by revisiting the protocol logic, I identified and fixed the issue. This experience highlighted the importance of combining AI tools with critical thinking and a strong grasp of fundamentals.

8. Appendix

The full simulation code is available at:

https://github.com/hu11uu/SC4052-CloudComputing-Assignment1/blob/main/CloudComputing_Assignment1_Simulation.ipynb

9. References

[1] M. Alizadeh et al., "Data center TCP (DCTCP)," ACM SIGCOMM Computer Communication Review, vol. 40, no. 4, pp. 63–74, Aug. 2010, doi: https://doi.org/10.1145/1851275.1851192.

[2] K. Winstein and H. Balakrishnan, "TCP ex Machina: Computer-Generated Congestion Control." Available: https://web.mit.edu/remy/TCPexMachina.pdf

[3] M. Dong et al., "Open access to the Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation is sponsored by USENIX. PCC Vivace: Online-Learning Congestion Control This paper is included in the Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI '18)." Available: https://www.usenix.org/system/files/conference/nsdi18/nsdi18-dong.pdf

[4] N. Jay, N. Rotman, P. Godfrey, M. Schapira, and A. Tamar, "A Deep Reinforcement Learning Perspective on Internet Congestion Control." Accessed: Feb. 27, 2026. [Online]. Available: http://proceedings.mlr.press/v97/jay19a/jay19a.pdf

[5] Y. Zhu et al., "Congestion Control for Large-Scale RDMA Deployments," Aug. 2015, doi: https://doi.org/10.1145/2785956.2787484.

[6] E. Feilden, A. Oltean, and P. Johnston, "Why we should train AI in space." Available: https://starcloudinc.github.io/wp.pdf