HACC：Hierarchical Automatic Selection of Congestion Control Algorithms

Yujie Peng, Jin Wang, Youyang Wang, Jing Wang and Jinbin Hu

Changsha University of Science and Technology, Changsha410114, China

Abstract. Most congestion control mechanisms work well in certain network environments, and no one can adapt well to consistently deliver good performance in all scenarios. The recently proposed frameworks based on reinforcement learning can flexibility select congestion control algorithms to be resilient to the dynamic changes for network status. However, frequent alterations to the congestion control mechanisms during the relatively stable period of the network actually lead to network instability and unnecessary computational overhead. In this paper, we present a hierarchical adaptive congestion control algorithm (HACC) to solve this problem. HACC dynamically selects the appropriate congestion control mechanism only when the current congestion control algorithm is not suitable for the current network state, rather than changing congestion control scheme every training cycle to ensure network stability. The simulation results show that HACC reduces flow completion time (FCT) and increases throughput significantly in stable network status. For example, ACC achieves throughput rates that are 47%，35%, 23% and 8% over Cubic, Reno, BBR and Antelope.

Keywords: Congestion Control, Deep Reinforcement Learning, Transport Protocols.

1 Introduction

Since the introduction of TCP, various congestion control (CC) mechanisms have been proposed. However, it's important to note that no single mechanism can consistently deliver peak network performance across all environments and user demands. This can be attributed to two primary factors. First, most algorithms are typically tailored for specific network environments. Second, a single congestion control strategy often struggles to maintain optimal performance in dynamically changing situations.

Machine learning-based congestion control methods hold the potential to autonomously adapt to a wide range of network scenarios, offering promising solutions to the challenges posed by network congestion. There are machine learning-based CC mechanisms like RemyCC [1], PCC-Vivace [2], DeepCC [3], and Orca [4], each using different inputs and techniques to adjust sending rates.

However, their practical implementation within real networks proves intricate due to the idiosyncrasies of individual network environments and the continuous learning demands, which undermine their dependability in novel network setups. To mitigate these challenges, researchers have introduced a novel approach leveraging deep learning to intelligently select appropriate congestion control policies based on the prevailing network conditions. This approach, exemplified by Antelope [5], eschews direct transmission rate configuration, instead dynamically fine-tuning congestion control for each data flow in response to the current network state. However, this adaptive strategy introduces potential complexities, as traditional congestion control mechanisms work gradually to stabilize networks, and frequent adjustments may precipitate network instability, consequently impeding overall performance.

This paper presents a new congestion control system called HACC. It uses a flexible hierarchical structure and machine learning to choose the best congestion control method based on the network's current status. This involves two parts: a decision module and a policy module. The decision module decides when the current congestion control isn't suitable for the network. When needed, the policy module creates a new congestion control method that fits the current network conditions.

2 Motivation

Using deep reinforcement learning to adapt the congestion control (CC) policy based on network conditions can pose challenges, and frequent adjustments to the CC policy may lead to network instability, affecting overall system performance. We conducted evaluations in a simulated network using the Ns-3 simulation tool. In this setup, we created a network with a client, server, and background traffic node using the Ns-3 topology module. Configuration settings were applied to application layer protocols and traffic patterns to initiate communication between the client and server. We then introduced an additional background traffic node and gradually increased its traffic load to induce network congestion over time.

We initiate a gradual increase in background traffic, observing a concurrent decrease in throughput for both Antelope and BBR [6]. Notably, BBR exhibit a more pronounce reduction in throughput compared to Antelope. However, for a specific small period of time, Antelope's performance even lower than that of BBR. This decline is attributed to Antelope's frequent shifts in CC control policies.

3 System Design

3.1 HACC Overview

In the context of the HACC architectural framework, as depicted in Figure 1, the system's functioning necessitates the acquisition and observation of data inputs for assessing the current state of the Reinforcement Learning (RL) agent. This assessment involves comparing the current state to the last state at which an action was executed. The system initiates an action only when it discerns a significant disparity, signaling the activation of the policy generator module to execute the next action. Furthermore, the RL agent maintains a connection to the environment, allowing for the retrieval of updated state and reward values. This connectivity serves to augment the agent's knowledge and enhances its overall performance.

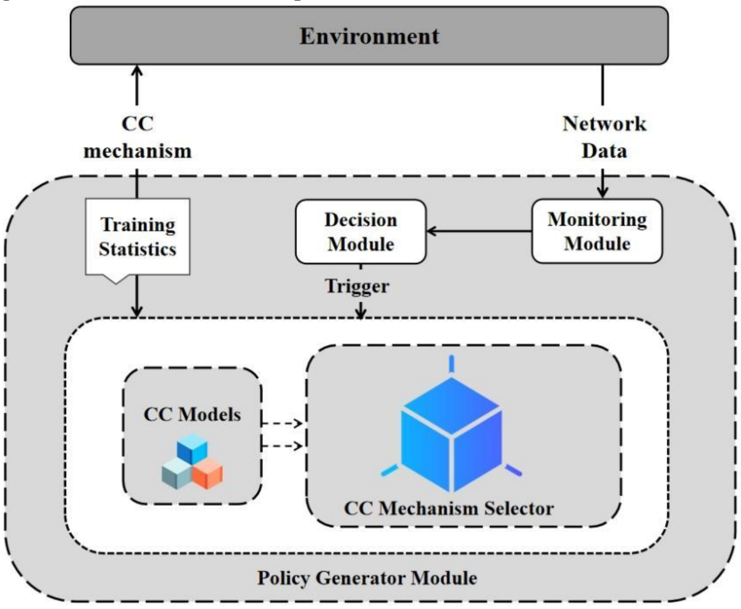


Fig. 1. HACC overview

3.2 Design Details

HACC comprises three crucial modules: the Monitoring Module, the Decision Module, and the Policy Generator Module. The Monitoring Module is responsible for retrieving environmental data and transmitting it to the Decision Module for assessing the ongoing suitability of the current CC control policy for the network environment. If adjustments to the CC control policy are deemed necessary, the Policy Generator Module utilizes the current status information to formulate the most appropriate CC control policy.

Monitoring Module: In the RL agent's architectural design, the pivotal role of gathering present network data from the environment is entrusted to the monitoring module, as illustrated by the statistical metrics detailed in Table 1.

Table 1. Statistics generated by the Monitoring Module.

Category	Meaning
Rtt	The mean value of the smoothed RTT
number	The number of ACK packets

loss	The number of lost packets
throughput	The mean value of sending throughput
$\text{delay}_{\text{min}}$	The minimum packet delay so far
pacing_rate	The maximum pacing rate so far

Decision Module: Within the dynamic network context, the Decision Module assesses the present CC situation and determines if an adjustment to the CC policy is warranted. It is noteworthy that each instance when the Decision Module triggers the activation of the Policy Generator Module, both the current state and the implemented CC policy are meticulously documented for reference.

Policy Generator Module: In our RL agent architecture, the policy generator module plays a central role. This module includes sub-models for different CC control strategies, allowing it to dynamically choose the right one for the current network conditions. For instance, if we want to predict the reward for a specific flow using the BBR control strategy, we use the BBR sub-model. This prediction helps the RL agent select the most suitable CC control mechanism by choosing the one with the highest expected reward.

Reward Function: In the realm of deep reinforcement learning, the reward function assumes paramount significance, exerting a substantial influence on the overall system performance. Within our research, we introduce Equation 1 as a quantitative metric, denoted as the reward function, to assess the efficacy of each congestion control (CC) mechanism.

$$\widehat{R} = R/R_{max} = \left(\frac{throughput - \eta' \times loss}{delay'}\right) / \left(\frac{pacing\_rate_{\ max}}{delay_{\ min}}\right) - \alpha \times \ trigger \qquad (1)$$

In a related vein, Giessler's investigation introduces a compelling metric termed "$power$," defined as throughput divided by latency. Additionally, we incorporate a " $loss$ " parameter to fine-tune the reward function, with parameter η' playing a pivotal role in calibrating the influence of packet loss on this function.

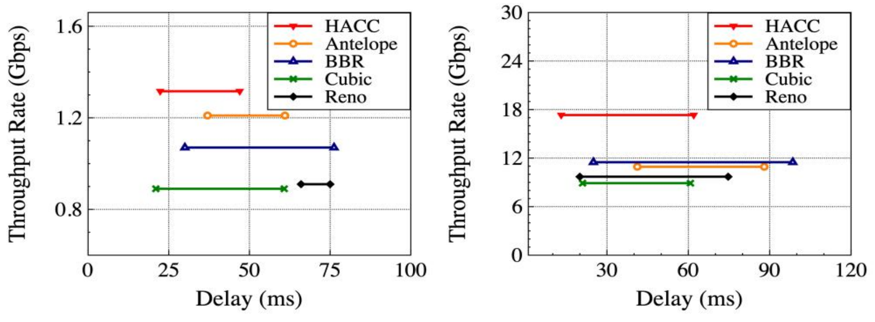
The hyperparameter α holds a central role in our model training, with a fixed value of 0.05. This parameter governs the imposition of penalties during training, where an increase α value reduces policy generator activations, emphasizing system stability. The decision module exercises discretion in triggering the policy generator base on α. Conversely, a decrease in α leads to more frequent activations, affording finer control over network dynamics.

$$\text{delay}' = \begin{cases} \text{delay}_{\ min} (\text{ delay}_{\ min} \leq \text{ delay } \leq \delta \times \text{ delay}_{\ min}) \\ \text{delay o.w.} \end{cases} \qquad (2)$$

However, optimizing both maximum throughput and minimal latency concurrently presents a complex challenge. Larger flows tend to prioritize throughput, while smaller flows prioritize latency. To tackle this issue, we introduce a factor δ (where $\delta \geq 1$) in Equation 2. Initially set to *2* when the connection is established, δ grows exponentially as data units are received, with each unit doubling δ value. Consequently, the reward function's focus shifts from latency sensitivity to throughput sensitivity. It is worth noting that a similar approach is adopted in the Orca framework.

4 Performance Evaluation and Analysis

In this section, our objective is to conduct an assessment of the effectiveness and practical feasibility of HACC using NS-3 simulations. We create a network topology with two nodes named h1 and h2, where h1 acts as a server node and h2 acts as a client node, and we establish a point-to-point connection between them. We create three processes in h2, which send data to h1 at 10ms intervals from the start time. The link latency and bandwidth are normalized to 1000ms and 40Gbps. We perform a series of experiments with both long and short flow sizes. For long flows, file requests range from roughly 10MB, while for short flows, the size is around 1MB.



(a) short flows (b) long flows

Fig. 2. Performance under simulation network

As depicted in Figure 2, The icon shows the average delay, and the line end represents the highest delay value. The Antelope algorithm generally demonstrates superior performance when compared to alternative congestion control mechanisms like Cubic [7] and BBR. However, its susceptibility to network fluctuations becomes more pronounced with short flows, resulting in performance variability, particularly in scenarios involving short flows where the overall performance lower that of Cubic and BBR.

HACC consistently demonstrates superior and stable performance, consistently outperforming leading congestion control algorithms such as Cubic. HACC achieves throughput rates that are 47%， 35%, 23% and 8% over Cubic, Reno [8], BBR and Antelope, respectively. Its flexible layered control architecture, particularly its isolation of the policy module from the agent, proves advantageous as the network stabilizes. This approach ensures a thorough assessment of the network environment before making frequent adjustments to the CC mechanism, thereby enhancing stability and preventing network disruptions cause by sudden CC changes, which can negatively affect overall network performance.

5 Conclusion

This paper introduces HACC, an innovative congestion control solution using Deep Reinforcement Learning (DRL) for dynamic mechanism selection based on assessments by the decision module. HACC's flexible hierarchical control structure, comprising a decision module and a policy generation module, minimizes the need for frequent policy adjustments, enhancing network stability. Empirical assessments reveal that HACC significantly outperforms existing methods, achieving substantial latency reductions and noteworthy throughput improvements, with gains of 47%, 35%, 23%, and 8% over Cubic, Reno, BBR, and Antelope, respectively.

References

1. Winstein, K., & Balakrishnan, H.: Tcp ex machina: Computer-generated congestion control. In ACM SIGCOMM Computer Communication Review, pp.123-134, (2013).
2. Dong, M., Meng, T., Zarchy, D., Arslan, E., Gilad, Y., Godfrey, B., & Schapira, M.: {PCC} Vivace:{Online-Learning} Congestion Control. In 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), pp. 343-356, (2018).
3. Abbasloo, S., Yen, C. Y., & Chao, H. J.: Make tcp great (again?!) in cellular networks: A deep reinforcement learning approach. In arXiv preprint arXiv:1912.11735. (2019).
4. Abbasloo, S., Yen, C. Y., & Chao, H. J.: Classic meets modern: A pragmatic learning-based congestion control for the internet. In Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication, pp. 632-647, (2020).
5. Zhou, J., Qiu, X., Li, Z., Tyson, G., Li, Q., Duan, J., & Wang, Y.: Antelope: A framework for dynamic selection of congestion control algorithms. In 2021 IEEE 29th International Conference on Network Protocols (ICNP), pp. 1-11. IEEE. (2021)
6. Cardwell, N., Cheng, Y., Gunn, C. S., Yeganeh, S. H., & Jacobson, V.: BBR: Congestion-based congestion control. In Communications of the ACM, 60(2), pp. 58-66, (2017).
7. Ha, S., Rhee, I., & Xu, L.: CUBIC: a new TCP-friendly high-speed TCP variant. In ACM SIGOPS operating systems review, 42(5), pp. 64-74, (2008).
8. Henderson, T., Floyd, S., Gurtov, A., & Nishida, Y.: The NewReno modification to TCP's fast recovery algorithm. In Technical Report, No. rfc6582, (2012)