AUTO: Adaptive Congestion Control Based on Multi-Objective Reinforcement Learning for the Satellite-Ground Integrated Network

Xu Li, Feilong Tang, and Jiacheng Liu, *Shanghai Jiao Tong University;* Laurence T. Yang, *St. Francis Xavier University;* Luoyi Fu and Long Chen, *Shanghai Jiao Tong University*

https://www.usenix.org/conference/atc21/presentation/li-xu

This paper is included in the Proceedings of the 2021 USENIX Annual Technical Conference.

July 14–16, 2021

978-1-939133-23-6

Open access to the Proceedings of the 2021 USENIX Annual Technical Conference is sponsored by USENIX.

AUTO: Adaptive Congestion Control Based on Multi-Objective Reinforcement Learning for the Satellite-Ground Integrated Network

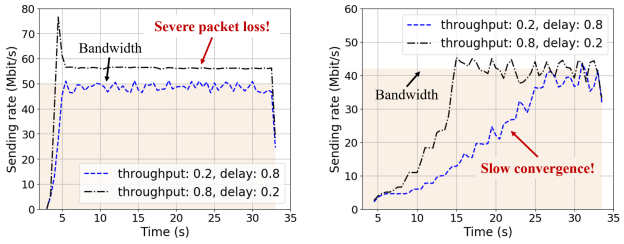
Xu Li†, Feilong Tang†*, Jiacheng Liu†, Laurence T. Yang‡, Luoyi Fu†, Long Chen†
† *Department of Computer Science, Shanghai Jiao Tong University*
‡ *Department of Computer Science, St. Francis Xavier University*

Abstract

The satellite-ground integrated network is highly heterogeneous with diversified applications. It requires congestion control (CC) to achieve consistent high performances in both long-latency satellite networks and large-bandwidth terrestrial networks and cope with different application requirements. However, existing schemes can hardly achieve these goals, for they cannot balance the objectives of CC (i.e., throughput, delay) adaptively and are not objective-configurable. To address these limitations, we propose and implement a novel adaptive CC scheme named AUTO, based on Multi-Objective Reinforcement Learning (MORL). It is *environment-adaptive* by training a MORL agent and a preference adaptation model. The first can generate optimal policies for all possible preferences (i.e., the relative importance of objectives). The latter automatically selects an appropriate preference for each environment, by taking a state sequence as input to recognize the environment. Meanwhile, AUTO *can satisfy diversified application requirements* by letting applications determine the input preference at will. Evaluations on emulated networks and the real Internet show that AUTO consistently outperforms the state-of-the-art in representative network environments and is more robust to stochastic packet loss and rapid network changes. Moreover, AUTO can achieve fairness against different CC schemes.

1 Introduction

To achieve global coverage and meet the excessive custom demand for data access, the satellite-ground integrated network has attracted intensive research interest [1]. It is highly heterogeneous where a node can send data to different peers through both satellite networks located at different orbits, and ground networks including the Internet and cellular networks [2]. To achieve consistent high user experiences in all these environments with widely varying packet loss rates, round-trip time (RTT) and available bandwidth, *adaptive Congestion Control (CC)* is one of the key technologies [3]. Basically, it determines sending rates, convergence speeds, and the reactiveness to network fluctuations of underlining flows. It has two competing objectives, i.e., optimizing link utilization (high throughput) while avoiding congestion (low queuing delay) [4].



(a) Inter-datacenter network (b) Satellite network

Figure 1: Sending rates of two CC schemes that adopts two different preferences. A fixed preference can only achieve high performance in specific types of environments.

The two objectives have to be balanced adaptively in different environments [5], since a fixed preference (i.e., the relative importance of the objectives) can only achieve high performance in specific types of environments. To illustrate this phenomenon, we conduct an experiment by first training two Reinforcement Learning (RL) based CC schemes using the same training ground with two different preferences. We evaluated their performances in the emulated inter-datacenter network and the satellite network WINDs [6] based on Pantheon [7]. The former is high-speed and has a short buffer while the latter has a long latency. As shown in Figure 1, putting higher weight on "throughput" achieves good performance in the satellite network but meanwhile suffers from severe packet loss in the ground network. The main reason is that a saturated buffer only has little punishment on the final reward and the trained model cannot detect the congestion. In contrast, putting higher weight on "delay" suffers from slow convergence in the satellite network because it is too sensitive to delay variation.

Meanwhile, a CC schemes for the integrated network

*Feilong Tang is the corresponding author of this paper.

will serve diversified applications at the same time. Delay-sensitive applications such as online meetings require higher weight to be put on "delay", while throughput-sensitive applications such as file synchronization require higher weight to be put on "throughput". To cope with all these requirements, the CC scheme should be able to let applications determine the preference at will.

Lots of CC schemes have been proposed in recent years, which can be classified into *heuristic-based* or *learning-based* schemes. Unfortunately, they cannot achieve the consistent high performance in all environments nor cope with diversified application requirements [7]. We analyze the reasons in what follows.

Heuristic-based methods hard-wire actions with predefined events or signals (e.g., packet loss, delay variation) based on the analysis of network characteristics. However, the analysis is not suitable for all kinds of network environments [8]. For example, *loss-based* methods such as TCP Cubic [9] take packet loss as the sign of congestion and is not suitable for unreliable lossy scenarios, while the *delay-based methods* such as Copa [4] don't work well in the short buffer scenarios for the delay fluctuates in a very small range [10]. Meanwhile, the hard-wire mapping fundamentally limits the objective configurability.

Existing learning-based methods can only achieve high performance in specific network environments [5, 8]. One of the main reasons is that they convert the objectives into a single reward function (RL) [5, 11–15] or utility function (online learning) [16–18] by introducing fixed empirical preferences and determine actions accordingly. Although training a series of models with different preferences to cope with different environments may alleviate this problem, the lack of an environment recognition method makes them unable to switch models adaptively.

Based on above analyses, we in this paper propose an *Adaptive congestion control method based on mUlTi-Objective reinforcement learning*, abbreviated as AUTO. It first divides time into consecutive monitoring intervals and formulates the CC as an extended version of the multi-objective Markov decision process. It then trains a *MORL agent* and a *preference adaptation model*, based on which it adjusts the sending rate at the end of each interval. Compared with existing approaches, it has following characteristics.

Firstly, AUTO is *environment-adaptive*. It automatically balances the two objectives in different environments where the preference adaptation model can adaptively select an environment-adaptive preference for users, by taking a state sequence as input to recognize the network environment. Meanwhile, the policy agent can generate optimal policies for all possible network states and preferences. Combining the two components, AUTO achieves the consistent high performance in different environments.

Secondly, AUTO is *objective-configurable*. Since the MORL agent only needs to be trained once and can deal with any input preferences, AUTO can cope with diversified application requirements.

Moreover, AUTO is *competitive-configurable*. The input preference determines the AUTO's delay-sensitivity and further influence its competitiveness against other flows. Therefore, AUTO can achieve fairness against different competing CC schemes and allows users to explicit adjust the priority of different flows, by adjusting the preference.

Main contributions are summarized as follows.

1. We propose an efficient and practical *Multi-Objective Reinforcement Learning (MORL) framework*. It trains a single MORL agent that can recover optimal policies for all possible preferences.
2. We propose a novel *environment-adaptive preference selection method*. We first propose a policy similarity model, based on which we find the best preference for each training environment by comparing with the expert policy. Then we train a preference adaptation model to automatically select preferences.
3. We propose and implement AUTO based on the above framework and method. It is easy to deploy for it requires sender-only modification. It consistently outperforms than the state-of-the-art in representative scenarios and is more robust to network changes and packet loss. Moreover, it can achieve fairness against other CC methods.
4. We develop an emulation-based training suite Pantheon-Gym for MORL-based CC. It eases the training by providing standard OpenAI Gym interfaces for researchers.

2 Related Work

We classify related work on learning-based CC methods into the following three categories.

RL-based methods train a policy model with the goal of maximizing a reward function. QTCP [13] adopted the Q-Learning framework and formulate the reward function as the proportional fairness [19] of throughput and delay. RL-TCP [12] adopted a SARSA [20] based RL framework and added the packet loss rate in the reward function. Their actions specify how to change the congestion window in response to variations in the network environments and they are trained on simulators NS2 [21] and NS3 [22] respectively. Aurora [5] formulate the reward function as the weight sum of throughput and delay. Meanwhile, it implements a simulation-based gym environment. To train the agent on real network environments, authors in [14] adopt the same reward function and designed Park, which is an open platform for learning-augmented computer systems. Considering the delayed action phenomenon, an asynchronous RL framework called MVFST-RL was proposed in [11]. To combine reinforcement learning and traditional CC schemes, TCP-RL [15] took different traditional CC schemes as the action space and trained the model through emulations built by the Linux tool. To solve the CC problem

for multi-path TCP, authors in [23] proposed SmartCC, which calculates the reward according to throughput, latency, and delay jitter. Authors in [24] proposed DRL-CC, which adopts the goodput as rewards. To solve the centralized congestion control problem, Iroko [25] formulate the reward function as a function of the bandwidth utilization and the latency of all links. Its action is to regulate the sending rates of all nodes. However, the trained policy model can only be applied to the fixed network topology and is intractable for large networks.

Online learning-based methods try to fine-tune the sending rate and decide the adjustment direction according to a utility function and they also split the time into consecutive monitoring intervals. PCC-Allegro [16] formulates the utility function as a function of the sending rate and the loss rate, while Vivace [17] puts delay into consideration on this basis. PCC Proteus [18] defines two kinds of utility functions. It can behave as a primary protocol for primary flows or an effective "scavenger" that only utilizes the residue bandwidth of primary flows. Compared with the RL-based methods, online learning-based methods cannot learn the prevailing network regularities and have been shown to have poorer performance than RL-based methods [5].

Inverse RL-based methods seek to avoid the empirical preference setting by finding a reward function from the expert demonstrations. Indigo [7] trained a model to adjust the sending rate every 10 ms by directly imitating expert behaviors. This kind of method is not objective-configurable and cannot cope with different application requirements. Meanwhile, both the RL-based and inverse RL-based methods cannot achieve fairness against other flows [5], since their competitiveness is not configurable and strongly related to the training environments.

To cope with the heterogeneity of the integrated network, we in this paper propose a new kind of learning-based method, i.e., *MORL-based method*. It addresses the limitations of existing methods and is environment-adaptive, objective-configurable, and competitive-configurable. Thus, it can achieve consistent high user experience for flows passing through different environments and with different performance requirements.

3 Multi-objective Reinforcement Learning based Congestion Control

In this section, we propose practical methods for training the MORL agent that can generate optimal policies for all possible preferences.

3.1 Congestion Control Formulation

In order to apply the MORL framework, we first formulate the congestion control as a *Multi-Objective Markov Decision Process (MOMDP) with delayed actions*. Formally, it can be described by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \Omega, \gamma, f_\Omega)$, where

- $\mathcal{S} \subseteq \mathbb{R}^n$ is the continuous state space;
- $\mathcal{A}$ is the discrete action set;
- $\mathcal{P}$ is the Markovian transition model;
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^2$ is the set of two reward functions and corresponds to the two objectives for CC;
- $\Omega \subseteq \mathbb{R}^2$ is the preference space;
- $\gamma \in [0,1)$ is the discount factor;
- f_Ω is the set of preference functions that convert the reward vector into one scalar according to the chosen preference $\omega \in \Omega$.

In this work, preference $\omega \in \Omega$ is a vector of two values, i.e, $\omega = [\omega_t, \omega_d]$ is the weight on throughput and delay respectively. Meanwhile, we consider functions in f_Ω are linear functions, i.e.,

$$f_\omega(\mathcal{R}(s,a)) = \omega^{\mathrm{T}} \mathcal{R}(s,a), \forall s \in \mathcal{S}, \forall a \in \mathcal{A} \tag{1}$$

In the MOMDP, a policy π is associated to an expected return for a given preference $\omega \in \Omega$, which is denoted as

$$J_\omega^\pi = \mathbb{E}_{s_t \sim \mathcal{P}, a_t \sim \pi} \left[\sum_{t=0}^{T} \gamma \cdot f_\omega(\mathcal{R}(s_t, a_t)) \right] \tag{2}$$

where $s_t \sim \mathcal{P}$ means state s_t is sample from $\mathcal{P}$ and $a_t \sim \pi$ means action a_t is selected according to policy π.

The goal for solving MOMDP is to find the set of Pareto-optimal policies for all possible preferences, which is depicted as

$$\Pi^* = \{\pi | \exists \omega \in \Omega, \nexists \pi' : J_\omega^{\pi'} > J_\omega^\pi\}. \tag{3}$$

MOMDP with delayed actions is an extension of the MOMDP, which means that actions won't take effect immediately and therefore the state s_{t+1} is not only influenced by action a_t and state s_t, but also influenced by the previous actions and states. It is caused by two reasons. Firstly, the apply of the action is delayed due to the policy lookup time [11]. Since the environment is running asynchronously, it would keep transmitting data based on the old action a_{t-1} during the policy looking up. Secondly, the state observation is delayed since the states are updated based on packet ACKs. After action a_t is applied, it takes about $0.5 \cdot \texttt{rtt}_t$ for receiver to receive the packets sent based on a_t. It takes another $0.5 \cdot \texttt{rtt}_t$ to response the ACKs. Thus, the observed network state is influenced by the action made $\texttt{rtt}_t$ before.

In what follows, we describe the formulation of state space, reward functions, the action set and the Monitoring Interval (MI) in detail.

State space considering delayed actions ($\mathcal{S}$). To accurately identify the network congestion without explicit network information, we select the following features to model network states. To further improve the robustness and generalization of the trained model, we try to use relative values instead of absolute values such as latency and sending rate.

- *Latency Ratio* r^t_{lat} [19]. Latency ratio is the ratio of the current MI's mean latency to minimum observed mean latency, which is one of the most important features to detect congestion.
- *Sending Ratio* r^t_{send} [5]. Sending ratio is the ratio of the number of sent packets to the number of acknowledged packets. It helps agents adjust the sending rate.
- *Packet Loss Rate* r^t_{loss}. The r^t_{send} is influenced by the inflight ACKs. To prevent the agent from overestimating the congestion level, we add r^t_{loss} in the state.
- *Latency Gradient* g^t_{lat} [17]. Latency gradient is the derivative of latency with respect to time. The agent can infer that the sending rate is greater than the capacity if $g^t_{lat} > 0$.
- *Action* a_t. Considering that the state is influenced by the previous actions (the delayed action phenomenon), the action history should be put in the state to fasten convergence [11].

The delayed action phenomenon makes the reinforcement learning more challenging, since it is hard for agents to catch the real transition for the MOMDP. Therefore, we let each state be a history of the network statistics so that the agent can capture the influence of previous action on future states. We use h to represent the history length and the state $s_t \in \mathcal{S}$ can be formulated as

$$s_t = \{r^{t-i}_{lat},\ r^{t-i}_{send},\ r^{t-i}_{loss},\ g^{t-i}_{lat},\ a_{t-i} \mid i \in [0,h]\}. \tag{4}$$

Multi-Objective Reward Functions ($\mathcal{R}$). The goal of the CC is to achieve both high throughput and low queuing delay. Correspondingly, we adopt the average throughput, denoted as $\texttt{throughput}_t$, and the negative mean RTT as reward functions, i.e.,

$$\mathcal{R}(s_t, a_t) = [\texttt{throughput}_t,\ -\texttt{rtt}_t]. \tag{5}$$

RTT-adaptive monitoring interval. In AUTO, the state and the rewards are calculated at the end of each MI. A short MI is not enough for capturing the real transition in long-RTT scenarios, while setting the MI too long reducing the real-time responsiveness to the network dynamics in short-RTT scenarios. To cope with the heterogeneous network environment and considering that each action takes about $0.5 \cdot \texttt{rtt}_{t-1}$ to take effects, we adopt the RTT-adaptive MI and set the length of each MI to

$$|\texttt{MI}_t| = \min\{0.5 \cdot \texttt{rtt}_{t-1}, 1.5 \cdot \texttt{rtt}^{min}_r\} \tag{6}$$

where $\texttt{rtt}^{min}_r$ is the observed minimum RTT. It guarantees that the MI is longer enough and meanwhile let CC be able to quick react to congestion.

Action set ($\mathcal{A}$). In our proposed CC scheme, each action corresponds to an adjustment on the sending rate. To adapt to heterogeneous environments with variant bandwidth, we seek to adjust the sending rate proportionally and adopt an action set containing seven discrete values:

$$\mathcal{A} = \{\div 2, \div 1.3, \div 1.1, \times 1, \times 1.1, \times 1.3, \times 2\} \tag{7}$$

where "$\div x$" means "dividing the sending rate by x", and "$\times x$" means "multiplying the sending rate by x".

Since the reward function is the combination of throughput and delay, the agent will observe the equivalently bad rewards when the sending rate raises to 50x or 100x bandwidth and the buffer queue is saturated. In the training stage, bonding the sending rate to reasonable limits will produce the same transitions and meanwhile decrease CPU overheads caused by packet generation and sending. Thus, we set the sending rate in MI_t, denoted as sr_{t+1}, to

$$sr_{t+1} = \min(\text{apply}(sr_t, a_t), 500\text{Mbps}) \tag{8}$$

where $\text{apply}(sr_t, a_t)$ is a function that applies action a_t on sending rate sr_t.

3.2 Parallel MORL Training Framework

The goal of the MORL agent is to achieve consistent high performance in diverse environments under different preferences. Since the optimal policy in these scenarios varies, the MORL agent has to be trained in a series of environments with different network characteristics and input preferences. It results in the *catastrophic forgetting problem* [26] and *extended training time*, where catastrophic forgetting means that the knowledge for previously learned environment is abruptly destroyed by the training in the new environment.

To cope with these issues, we utilize an efficient parallel MORL training framework as shown in Figure 2. Basically, it is an extended version of the Asynchronous Advantage Actor-Critic framework [27] and has three key points. Firstly, it is *asynchronous*, which means multiple agents are used to train actors in parallel and updates the global parameters periodically. To deal with the catastrophic forgetting problem, we train the agent in different environments simultaneously, by letting each agent interacts with a specific type of environment (e.g., long latency, low bandwidth, high throughput weight). Secondly, it is based on the actor-critic paradigm [28]. In the training phase, the actor updates the policy parameters in the direction suggested by the critic while the critic updates the value function parameters. Thirdly, it uses the *advantage* function instead of the raw value of an action [29], which helps us better compare the actions for a given state and makes the training process more stable.

The adopted framework greatly improves learning efficiency. On the one hand, it fastens the interaction through parallel training. On the other hand, it improves the sample efficiency by adopting the off-policy RL approach, which reuses any past episodes through the experience replay mechanism. Furthermore, it improves the exploration efficiency since the

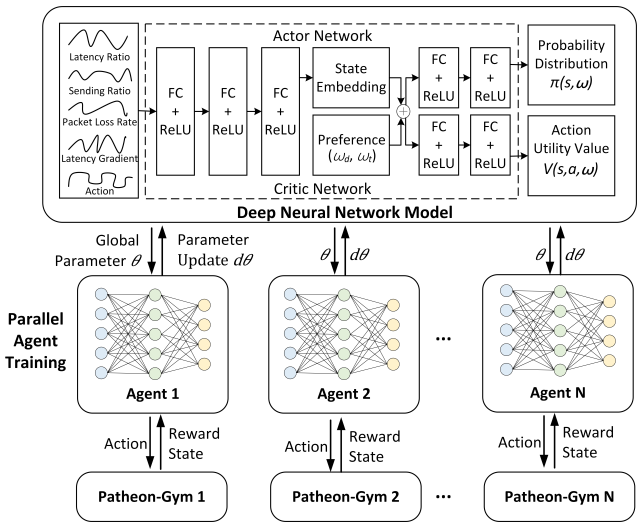


Figure 2: MORL Training Framework.

adopted off-policy brings better explorations for sample collection, by following a behavior policy that is different from the target policy.

Deep Neural Network Model. As shown in Figure 2, we use two deep neural networks with state s and preference ω as input and $2|\mathcal{A}|$ Q-values as output. The first is the actor network, which outputs the probability distribution over the action space. The second is the critic network, which outputs the utility value of the action. The parameters in two networks are denoted as θ_π and θ_v respectively.

The two networks share three fully connected layers, each of which uses a rectified linear activation unit (ReLU) for feature extraction from raw inputs. The extracted features are then concatenated with the preference value and fed into different fully connected layers for output.

3.3 Agent Training Strategies

To train the MORL agent with multiple reward functions, we update the neural network model according a generalized version of bellman equation [30]

$$\mathbf{Q}^{\pi}(s,a,\omega) = \mathop{\mathbb{E}}_{s'\sim P}\left[\mathcal{R}(s,a) + \gamma \mathop{\mathbb{E}}_{a'\sim\pi}\left[\mathbf{Q}^{\pi}\left(s',a',\omega\right)\right]\right] \tag{9}$$

where $\mathbf{Q}$ is the Multi-Objective Q-value (MOQ-value) function. Its key idea is to use the vectorized value function to perform envelope updates. It allows our method to quickly align one preference with optimal rewards and trajectories that may have been explored under other preference. Thus, it can learn a single parametric representation for optimal policies over the whole preference space.

Based on the above equation, we can calculate the target for a given transition $(s,a,s',\boldsymbol{r})$ at step k by

$$\boldsymbol{y} = \mathbb{E}[\boldsymbol{r} + \gamma \arg_{\mathbf{Q}} \max_{\omega'}[\omega^{\intercal}\mathbf{Q}(s',a,\omega';\theta_k)] \tag{10}$$

where s and s' are the previous state and the current state, a is the adopted action, $\boldsymbol{r}$ is the reward vector, ω' is sampled according to a fixed distribution, θ_k is the network parameters at step k and $\mathbf{Q}(\cdot;\theta)$ is the MOQ-value function parameterized by θ. Then, we can use the mean square error (MSE) between the prediction returned by the neural networks and the target,

$$L^{\mathtt{A}}(\theta) = \mathbb{E}_{s,a,\omega}\left[\|\boldsymbol{y} - \mathbf{Q}(s,a,\omega;\theta)\|_2^2\right] \tag{11}$$

3.3.1 Homotopy optimization

Unfortunately, directly optimizing $L^{\mathtt{A}}$ is challenging in practice since it is non-smooth considering that the optimal frontier contains a large number of discrete solutions. To solve this problem, we adopt homotopy optimization [31] similar to [30]. To be specific, we construct an auxiliary loss function $L^{\mathtt{B}}$ that directly optimize the scalarized Q value as,

$$L^{\mathtt{B}}(\theta) = \mathbb{E}_{s,a,\omega}[|\omega^{\intercal}\boldsymbol{y} - \omega^{\intercal}\mathbf{Q}(s,a,\omega;\theta)|]. \tag{12}$$

Thus, the overall loss function is formulated as

$$L(\theta) = (1-\lambda)\cdot L^{\mathtt{A}}(\theta) + \lambda \cdot L^{\mathtt{B}}(\theta) \tag{13}$$

where λ slowly increases from 0 to 1 in the training process and shifts the loss function from $L^{\mathtt{A}}$ to $L^{\mathtt{B}}$.

Finally, we can calculate the stochastic gradient of network parameters by

$$d\theta_\pi = \frac{1}{N_\omega N_\tau}\sum_{i,j} T_{ij} \tag{14}$$

$$d\theta_v = (1-\lambda)\cdot\nabla_{\theta_v} L^{\mathtt{A}}(\theta_v) + \lambda\cdot\nabla_{\theta_v} L^{\mathtt{B}}(\theta_v) \tag{15}$$

where N_ω and N_τ are the minibatch sizes for sampling transitions and preferences respectively and

$$T_{ij} = \left[\omega_i^{\intercal}(\boldsymbol{V}_{ij} - \boldsymbol{V}(s_j,\boldsymbol{\omega}_i;\theta_v))\right]\nabla_{\theta_\pi}\log\pi(a_j|s_j,\boldsymbol{\omega}_i;\theta_\pi).$$

Here, $\boldsymbol{V}(s_j,\boldsymbol{\omega}_i;\theta_v)$ is the baseline for calculating the advantage [29] and $\boldsymbol{V}_{ij}$ is the estimated optimal value, which can be calculated by

$$\boldsymbol{V}_{ij} = \begin{cases} \boldsymbol{r}_j \ \text{ if } done \\ \boldsymbol{r}_j + \gamma \arg_V \max\limits_{\omega'\in W} \omega_i^{\intercal}\boldsymbol{V}(s_{j+1},\omega';\theta) \ o.w. \end{cases} \tag{16}$$

3.3.2 Early termination trick

For training efficiency, the *needle-in-the-haystack problem* [14] has to be solved, which is described as follows. In the exploration stage of the training, the agent performs random walks and may be trapped in congestion states, i.e., when the sending rate is above the available network bandwidth and the buffer is saturated, the agent can only observe equivalently bad rewards (which is the combination of throughput and latency). It provides meaningless gradients and results in extremely inefficient training.

To solve this problem and stabilize the training process, we propose a simple but useful method called *early termination trick*. The basic idea is to increase the training step length according to the episode index. In this work, we adopt a linear function to calculate the termination index of each training step, which is depicted as

$$f_{et}(i) = \delta_a + (i \bmod \delta_b) \tag{17}$$

where δ_a is the initiate step length and δ_b is the step growth speed.

Algorithm 1: Agent Training Algorithm

Input: Minibatch sizes N_τ and N_ω; network parameters θ_π and θ_v; λ for homotopy optimization; δ_a and δ_b in the early termination trick.

Output: Trained parameters θ_π and θ_v.

1 Initialize replay buffer $\mathcal{D}_\tau = \{\}$.
2 Set $\mathcal{D}_\omega = U\{\omega_t + \omega_d = 1 | \omega_t \in (0,1), \omega_d \in (0,1)\}$.
3 **for** *episode* = $1, \ldots, M$ **do**
4 Synchronize parameters $\theta'_v = \theta_v$ and $\theta'_\pi = \theta_\pi$.
5 Sample a preference $\omega \sim \mathcal{D}_\omega$.
6 **for** $t = 0, \ldots, N-1$ **do**
7 **if** $t \geq f_{et}(episode)$ **then**
8 **break**
9 Calculate state s_t according to Eq. (4).
10 Sample an action $a_t \sim \pi(a_t | s_t, \omega; \theta'_\pi)$.
11 Get $(\boldsymbol{r}_t, s_{t+1}, done)$ from the environment.
12 Set $\mathcal{D}_\tau = \mathcal{D}_\tau \cup (s_t, a_t, \boldsymbol{r}_t, s_{t+1})$.
13 **if** $|\mathcal{D}_\tau| \geq N_\tau$ **then**
14 Sample $(s_j, a_j, \boldsymbol{r}_j, s_{j+1}) \sim \mathcal{D}_\tau$.
15 Sample $W = \{\omega_1, \omega_2, \ldots, \omega_{N_\omega}\} \sim \mathcal{D}_\omega$.
16 Calculate $d\theta_\pi$ and $d\theta_v$ using Eq. (14)(15)
17 Perform asynchronous update of θ_v using $d\theta_v$ and of θ_π using $d\theta_\pi$.
18 **return** θ_π and θ_v..

3.3.3 Training algorithm

Combine the above components, we get the agent training algorithm as shown in Algorithm 1, which works as follows. We first initialize the replay buffer as an empty set (line 1) and adopt a uniform distribution for preference sampling (line 2). At the beginning of each episode, the agent synchronizes the network parameters with the global parameters (line 4). Next, it interacts with the environment to collect the trajectory for the replay buffer (lines 5-12). Considering the needly-in-the-haystack problem, the step length is calculated based on the early termination trick (lines 7-8). Then the stochastic gradient of the parameters for the actor network ($d\theta_\pi$) and for the critic network ($d\theta_v$) are calculated according to the sampled transitions and the homotopy update trick (lines 13-16). Finally, the global parameters are updated asynchronously based on $d\theta_\pi$ and $d\theta_v$ (line 17).

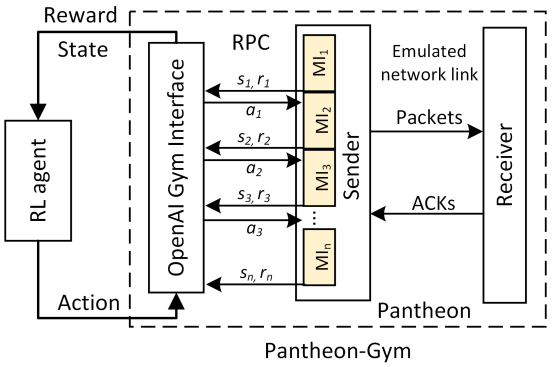


Figure 3: Pantheon-Gym architecture.

Essentially, Algorithm 1 is to iteratively apply the Bellman optimality operator $\mathcal{T}$ on $\mathbf{Q}$. Since $\mathcal{T}$ is a contraction mapping on the complete pseudo-metric space, Algorithm 1 will finally terminate with a MOQ-value function $\mathbf{Q}$ that is equivalent to the preferred optimal value function $\mathbf{Q}^*$ based on the Multi-Objective Banach Fixed-Point Theorem [30].

3.4 Pantheon-Gym: MORL Training Suite For CC

To train the models and to facilitate further research, we present Pantheon-Gym, which is an asynchronous MORL suite for CC based on the OpenAI Gym interface [32] and Pantheon [7] as shown in Figure 3. It supports MORL by returning a reward vector rather than a single scalar. Meanwhile, it interacts with the MORL agent with the standard OpenAI Gym interface, letting researchers design their own agent in an easy way.

Besides the features adopted in our state space, it also supports various kinds of features including 1) sending rate; 2) receiving rate; 3) 95th percentile latency; 4) the exponentially-weighted moving average (EWMA) of the queuing delay; 5) the EWMA of the sending rate; 6) the EWMA of the receiving rate. Users can conveniently customize their own reward functions and state space.

Compared with Aurora [5] which proposes a training suite based on a simulated network environment, Pantheon-Gym provides more realistic training data by adopting an emulated network environment. To promote bandwidth utilization and training efficiency, Pantheon-Gym trains models in an asynchronous fashion similar to MVFST-RL [11]. The agents, Gym interface and the emulated network work in different processes and communicate with each other through remote procedure calls. Therefore, the environment steps are not blocked by the policy lookup and the forward-pass in the training stage.

4 Environment-Adaptive Preference Selection Method

In this section, we propose a novel environment-adaptive preference selection method by training the preference adaptation model with the help of the expert policy. When the preference is not determined by the upper-level application, it automatically selects a suitable preference by taking the state sequences as input to recognize the network environment.

4.1 Expert Policy

In the training stage, since we know the link capacity, we can construct an expert policy π^* that adjusts the sending rate to the link capacity as quickly as possible. More specifically, assuming c is the link capacity, then the action produced by the expert policy at each MI is depicted as

$$\pi^* : a_t = \arg_{a\in\mathcal{A}} \min(|\text{apply}(sr_t, a) - c| + c * \mathbb{1}_{\text{apply}(sr_t,a)>c}) \tag{18}$$

where $\mathbb{1}_{\text{apply}(sr_t,a)>c}$ is a binary function that outputs 1 if $\text{apply}(sr_t, a)$ is larger than the link capacity c. It guarantees that actions that do not cause excessive sending rate will be prioritized.

4.2 Policy Similarity Model

We next propose a policy similarity model based on the cumulative reward distributions [33], which is used to quantify the similarity between the expert policy and the policy for a given preference ω. Based on it, we find the best preference for each training environment. Compared with ordinary methods that only use the expectations to estimate the policy, such a method achieves higher accuracy.

The cumulative reward for policy π is defined as

$$\hat{\mathcal{R}}^\pi = \sum_{t=0}^{T} \gamma^t \mathcal{R}(s_t, a_t). \tag{19}$$

Due to the transition probability, $\hat{R}^\pi$ is a random variable in $\mathbb{R}^2$. For a given policy π, we further define its Markov chain as $\mathcal{M}(\pi) = \{\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \pi\}$. Then the cumulative distribution of $\hat{\mathcal{R}}^\pi$ can be formulated as,

$$P^\pi(\boldsymbol{\varepsilon} \mid s, a) = \Pr(\hat{\mathcal{R}}^\pi \le \boldsymbol{\varepsilon} \mid s, a, \mathcal{M}(\pi)) \tag{20}$$

By assuming this distribution follows a multi-variable Gaussian distribution $G(\boldsymbol{\varepsilon}; \theta)$ parameterized by θ, and adopting a Monte Carlo sampling technique to collect dataset $\mathcal{D}$, we can get the distribution parameters by,

$$\Theta^* = \arg_\Theta \max \mathbb{E}_\mathcal{D}(G(\boldsymbol{\varepsilon}; \Theta)) \tag{21}$$

We then use the Kullback-Leibler divergence [34] between the expert and the policy generate by different preference as the similarity measurement, which is formulated by

$$D_{\text{KL}}(G^* \| G_\omega) = \int_{-\infty}^{\infty} G(\boldsymbol{\varepsilon}; \Theta^*) \log\left(\frac{G(\boldsymbol{\varepsilon}; \Theta^*)}{G(\boldsymbol{\varepsilon}; \Theta_\omega)}\right) d\boldsymbol{\varepsilon} \tag{22}$$

where $G^* = G(\varepsilon; \Theta^*)$ and $G_\omega = G(\varepsilon; \Theta_\omega)$ are the Gaussian distributions for the expert policy and the policy generated by preference ω. A larger D_{KL} means that the similarity between the two policies is lower.

4.3 Preference Adaptation Model

Based on the policy similarity model, we first find the most suitable preference ω for each training environment, under which the reward distribution is most similar to that of the expert policy (i.e., $D_{\text{KL}}(G^* \| G_\omega)$ is minimized). Then, by running a set of experiments using a indicator preference $\tilde{\omega}$, we can collect the training dataset $\mathcal{D}_p = \{(\boldsymbol{S}_0, \omega_0^*), (\boldsymbol{S}_1, \omega_1^*), \ldots (\boldsymbol{S}_n, \omega_n^*)\}$, where $\boldsymbol{S}_i$ is a state trajectory, and ω_i^* is the most suitable preference for $\boldsymbol{S}_i$.

Build upon this dataset, the policy adaptation problem can be formulate as a classification problem where each candidate preference can be seen as a class. To solve this problem, we train a *preference adaptation model* named *AdaM* using supervised learning. *AdaM* is a three-layer neural network. The first two layers of it are Long Short-Term Memory (LSTM [35]) layers that can efficiently handle the sequence data while the last layer is a fully connected layer.

We adopt the cross entropy loss [36] to train *AdaM*, which is formulated as

$$\mathcal{L} = -\sum_{i=0}^{N} \omega_i \log(\omega_i^*) \tag{23}$$

where ω_i^* is the ground truth for training sample i, while ω_i is the prediction. We omit the details of the training procedure since it is a classical supervised learning.

Algorithm 2: Preference Adaptation Algorithm

Input: Trained model *AdaM* and π; indicator preference $\tilde{\omega}$; sample count *SC*.
Output: Optimal ω for the current environment.

1 Initialize sequence buffer $\boldsymbol{S} = \{\}$.
2 Observe the current state s according to Eq. (4).
3 Get instruction policy $\pi_{\tilde{\omega}}$
4 **for** *k from 1 to SC* **do**
5 Sample an action $a_t \sim \pi_{\tilde{\omega}}(s)$.
6 Observe a new state s'.
7 Set $\boldsymbol{S} = \boldsymbol{S} \cup s'$.
8 Set $s = s'$.
9 **return** $\omega = AdaM(\boldsymbol{S})$.

Once we train the *AdaM*, we can use Algorithm 2 to infer the most suitable preference. It first initializes the state sequence buffer and observes the initial state (lines 1-2). Then, it collects a state sequence by interacting with the environment using the policy generated by the indicator preference $\tilde{\omega}$ (lines 3-8). Finally, we infer the best preference using the trained adaptation model *AdaM* (line 9).

5 Implementation and Evaluations

5.1 System Settings

Model Training and AUTO Implementation. We adopted the same training parameters as Aurora [5] to train the MORL agent as shown in Table 1. All parameters are sampled uniformly except the queue size, for which the log is sampled uniformly. The history length is set to 10. To train the preference adaptation model, we set the sample count *SC* to 10 and set the indicator preference $\tilde{\omega}$ to [0.3,0.7], i.e., the weights of throughput and delay are 0.3 and 0.7 respectively.

Table 1: Training Parameters

Bandwidth	Latency	Queue size	Loss rate
100-500 pps	50-500 ms	2-2981 packets	0-5%

We train models with 16 workers in parallel with different sampled preferences. It take around 7 days on a server with Intel Xeon Gold 6148 CPU and 128G memory to converge. The whole model takes about 0.5 milliseconds on our server. Using the two trained model, we implemented a user-space prototype on top of Pytorch [37] and the UDT framework [38]. It only requires modifications on the sender side and it works smoothly when any other CC scheme on the receiver-side sends per-packet ACK.

Evaluation environments. We evaluate performances in both emulated environments and the real Internet through following tools.

- Pantheon [7]. The Pantheon is a community evaluation platform. It supports performance evaluations on the Internet and meanwhile can generate *calibrated network emulators* that capture the diverse performance of real Internet paths. It only supports point-to-point topology and can run one scheme at once.
- CoCo-Beholder [39]. The CoCo-Beholder is an emulator based on Pantheon. It emulates multiple flows with different CC schemes on a dumbbell topology.

Compared CC algorithms. We compare AUTO with state-of-the-art methods that have been proven to achieve high performance in the heterogeneous network, including heuristic-based CC algorithms FillP [40], Copa [4], BBR [41] and LEDBAT [42], online learning based method PCC-Allegro [16] and Vivace [17], inverse RL based method Indigo [7], RL-based method Aurora [5], and statistic-based method TaoVA [43]. We further adopt classic TCP variants including TCP Cubic [9] and TCP Vegas [44] as baselines.

5.2 Achieve Consistent High Performance in Different Environments

We first show AUTO *achieves consistent high performance in different network environments* by evaluating its performance in three real-world, representative, and very different environments and a hybrid network environment.

5.2.1 Satellite network

The first representative environment is the satellite network, where links typically have longer RTT. We evaluate the performance on Pantheon parameterized with the real-world measurements of the WINDs satellite system [6], replicating an experiment from the PCC-Allegro paper [16]. The link has 800 ms RTT, 42 Mbps capacity, 0.74% stochastic loss rate, and a queue size of 1500. Part of the comparison algorithms have been shown to achieve high performance in this environment, including PCC-Allegro [16], Vivace [17] and Copa [4].

Figure 4(a) shows as expected, AUTO achieves different trade-off results between delay and throughput by inputting different preferences. Therefore, AUTO can cope with different application requirements. In contrast, other methods only obtain a single trade-off result.

For each comparison algorithm, there is always a preference under which AUTO achieves a better performance. Compared with methods that work well in satellite network environments, AUTO (0.8, 0.2) [1] achieves similar throughput but 14% shorter delay compared to PCC-Allegro, and 30.0% higher throughput and 20.6% shorter delay than Vivace, while AUTO (0.5, 0.5) achieves 6.9% higher throughput and 12.2% shorter delay than Copa. Compared with Indigo, the throughput and the delay of AUTO (0.2, 0.8) are 19.9% higher and 5.0% shorter respectively. Compared with BBR, AUTO (0.2, 0.8) achieves 153.5% higher throughput and 5.7% shorter delay. Compared with schemes that achieve short delay, the throughput of AUTO (0.05,0.95) is about 1.1 times of FillP, 3.1 times of Aurora, 4.0 times of TaoVA, 15.3 times of TCP Cubic, 29.2 times of TCP Vegas, and 86 times of LEDBAT.

In this environment, (0.5, 0.5) is selected as the default preference by the preference adaptation model. Its corresponding packet loss rate is shown in the blue bar in Figure 5, which is 79.0% and 85.9% lower than Copa and PCC-Allegro that achieve comparable throughput. With much higher throughput, the packet loss rate of AUTO (0.3, 0.7) is also lower than FillP, BBR, Vivace, and Aurora.

5.2.2 Cellular network

The second representative environment is the cellular network, where the links have time-varying speeds and are usually modeled as Poisson Point Processes (PPP). We emulate this environment on Pantheon parameterized with the real-world measurements of the path from Amazon Web Services (AWS) in Brazil to the Colombia cellular network, replicating an experiment from Pantheon [7]. The link has 260 ms RTT, 3.04 Mbps capacity, 0.6% stochastic loss rate, and a queue size of 426.

With different preferences, AUTO outperforms other meth-

[1] We use AUTO (ω_t, ω_d) to represent AUTO with preference set to $[\omega_t, \omega_d]$.

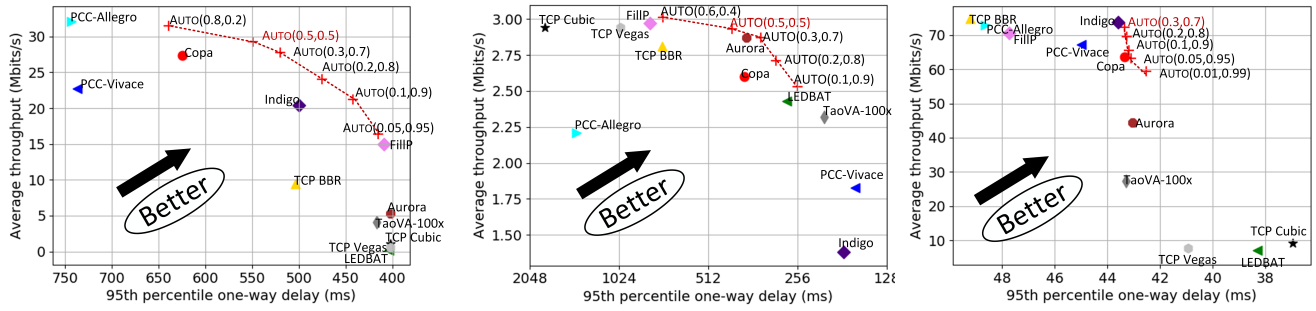


Figure 4: Throughput v.s. 95th percentile one-way delay in three representative environments. AUTO achieves consistent high performance and can cope with different application requirements by achieving a series the Pareto dominate results.

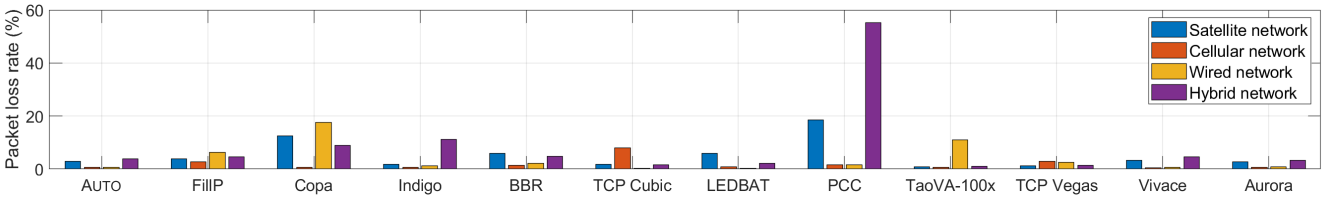


Figure 5: Packet loss rate in the satellite, cellular, wired, and hybrid network environments

ods in different aspects as shown in Figure 4(b). By setting the preference to (0.6, 0.4), AUTO achieves the highest throughput. Compared with other methods that prefer high throughput, it achieves 61.1%, 28.2% and 9.7% shorter delay than TCP Cubic, TCP Vegas and FillP respectively. By putting more weight on delay, AUTO achieves shorter delay at the expense of lower throughput. Statistically, AUTO (0.3, 0.7) achieves 5.7% higher throughput and 16.3% shorter delay than Copa. With almost the same throughput, it achieves 10.2% lower latency compared with Aurora. AUTO (0.1, 0.9) achieves 4.1% higher throughput and 8.3% shorter delay than LEDBAT. For other methods that achieve low link utilization in this environment, AUTO (0.1, 0.9) achieves 9.1%, 14.5%, 38.3% and 83.3% higher throughput than TaoVA, PCC-Allegro, Vivace and Indigo respectively.

The default preference in this environment is (0.5, 0.5). Under this preference, the packet loss rate of AUTO is 0.64% and is lower than all other methods except Vivace, which achieves 0.56% packet loss rate but very low link utilization.

5.2.3 Wired network

The third representative environment is the wired network, where the links have higher bandwidth and shorter buffer. We evaluate the performance in this environment on the real Internet by utilizing two servers located at Shanghai and Hong Kong respectively. The link between the two server has about 36.2 ms RTT and 80 Mbps capacity.

Although the link bandwidth in the evaluate environment is far beyond the training range [1.12 Mbps, 5.6 Mbps], AUTO still achieves a set of Pareto dominant results as shown in Figure 4(c). This is because the adopted state space achieves a good generalization of the model. By setting the preference to (0.3, 0.7), AUTO achieves comparable throughput as FillP, Copa, PCC-Allegro, Vivace and Indigo and meanwhile achieves shorter delay. Due to the stochastic packet loss, TCP Cubic, TCP Vegas and LEDBAT have poor performance in this test. Due to the slow convergence, the throughput of Aurora is 38.5% lower than AUTO (0.3, 0.7).

In this test, the queuing delay fluctuate in a small range because of the small queue size. If setting the throughput weight too large (≥ 0.4), AUTO will neglect the small queuing delay, which will result in high packet loss rate. Therefore, we set the throughput weight no higher than 0.3. In contrast, the delay-based method Copa cannot detect congestion correctly and therefore has large packet loss rate as shown in Figure 5.

Similar to the previous results, the default preference setting (0.3, 0.7) achieves lower packet loss rate than other CC algorithms that achieve the similar throughput. Quantitatively, the packet loss rate of AUTO (0.3, 0.7) is 88.9% lower than FillP, 96.0% lower than Copa, 42.1% lower than Indigo, 57.3% lower than PCC-Allegro and is 2.7% lower than Vivace.

5.2.4 Hybrid network

To show AUTO can cope with the heterogeneous satellite-ground integrated network, we next evaluate the performance on an emulated hybrid network path using Pantheon. The hybrid path contains a satellite link with 800 ms RTT and 42 Mbps bandwidth, and a terrestrial link with 40 ms RTT and

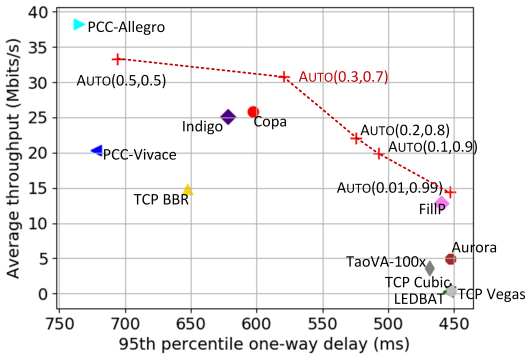


Figure 6: Throughput v.s. 95th percentile one-way delay in the hybrid network environment.

60 Mbps bandwidth.

For congestion control, the hybrid path can be abstracted as a single special link. Its delay and packet loss rate are the total delay and the overall packet loss rate of the path respectively, while its bandwidth is the bandwidth of the bottleneck link on the path. Since the bandwidth-delay ratios of hybrid paths are covered by representative environments, AUTO can well cope with it. As shown in Figure 6, AUTO achieves both high throughput and low latency. With similar latency, AUTO(0.01, 0.99) achieves 14.36 Mbps throughput, corresponding to a gain of 52.2×, 30.9×, 18.9×, 2.9×, 1.9×, and 12.5% over LEDBAT, TCP Vegas, TCP Cubic, TaoVA, Aurora, and FillP respectively. With similar throughput, it also improves the delay by 30.7% compared with BBR. AUTO(0.1, 0.9) achieves similar delay with Vivace but its delay (507.2 ms) is 29.8% shorter. Compared with Copa, AUTO(0.3, 0.7) improves the throughput and delay by 18.6% and 3.9% respectively. Meanwhile, it achieves 22.5% higher throughput and 6.9% lower delay compared with Indigo.

From the perspective of packet loss rate, the default preference setting (0.3, 0.7) performs better than all other methods that achieve a link utilization rate higher than 15%. As shown in Figure 5, it still achieves 15.0%, 16.9%, 18.9%, 57.0%, 65.5% lower packet loss rate than FillP, Vivace, BBR, Copa, and Indigo respectively. Although PCC-Allegro achieves the highest throughput in this scenario, it fails to detect congestion since the chosen preference is not suitable for this scenario. It suffers from 55.11% packet loss rate, which is 13.3× higher than AUTO(0.3, 0.7).

5.2.5 Summary

Existing CC methods cannot adapt to diversified environments. For example, BBR and FillP suffer from low throughput or long delay in the satellite, cellular, and the hybrid network environments. Due to the fixed monitoring interval and lack of consideration on delayed action phenomenon, Indigo can only achieve high performance in the wired network environment. Due to the empirical preference settings, PCC suffers from unacceptably high packet loss in the satellite environment, Vivace suffers from low throughput in both satellite and cellular network environments, while Aurora has poor performance in the satellite and the wired network environments.

In contrast, AUTO adapts to different network environments by automatically adjusting the preference. Although we adopt a relatively small training range, the trained policy model achieve high performance since there is no absolute value in the state space. Its performance can be further improved by adding more training environments. Further, by finding the Pareto-optimal frontier, AUTO can cope with different application requirements.

5.3 Adapt to Rapid Network Changes

We next show AUTO *achieves high reactiveness and adapts to the rapid network changes* by emulating a link with dynamic available bandwidth, which changes every 5 seconds with a uniform distribution ranging from 2 Mbps to 10 Mbps. The link delay and loss rate are set to 50 ms and 1% respectively.

Figure 7(a) shows that AUTO achieves highest average throughput. Quantitatively, it reaches 5.95 Mbps and 92.3% link utilization, corresponding to a gain of 2.6%, 5.1%, 7.0%, 11.8%, 16.9%, 37.4%, 45.5%, 75%, 1.58×, 2.79×, 3.92× over FillP, Vivace, BBR, Copa, TaoVA, Indigo, PCC-Allegro, Aurora, LEDBAT, TCP Vegas and Cubic, respectively. To further demonstrate AUTO's reactivity, we show AUTO achieves shorter delay comparing to other algorithms that achieve comparable throughput in the blue bar. More specifically, compared with FillP, Vivace and BBR, the delay of AUTO is 25.4%, 96.6%, and 74.7% shorter.

Figures 7(b)-(m) illustrate the behavior of different CC methods across time. As shown in Figure 7(b), AUTO's sending rate fluctuates around the available bandwidth since it adjusts the sending rate mainly according to delay variation. It tries to increase the sending rate when the latency ratio is small and to decrease otherwise. FillP and BBR also achieve good performance but have longer converge time when the bandwidth suddenly increases. Although Vivace achieves high throughput, it cannot cope with the bandwidth decrease and result in extremely long delay and high packet loss rate. Due to the slow convergence, Aurora has low reactiveness and suffers from poor bandwidth utilization. Other algorithms such as Copa, Indigo, Cubic, Vegas, and LEDBAT cannot cope with the varient bandwidth or misestimate the network status due to the stochastic packet loss rate. Their and suffer from low throughput.

5.4 Resilient to Stochastic Packet Loss

Then, we show that AUTO *is resilient to stochastic packet loss*, which is challenging for CC algorithms since the stochastic packet loss caused by poor link quality could let CC algorithms mistakenly think that congestion occurred and further

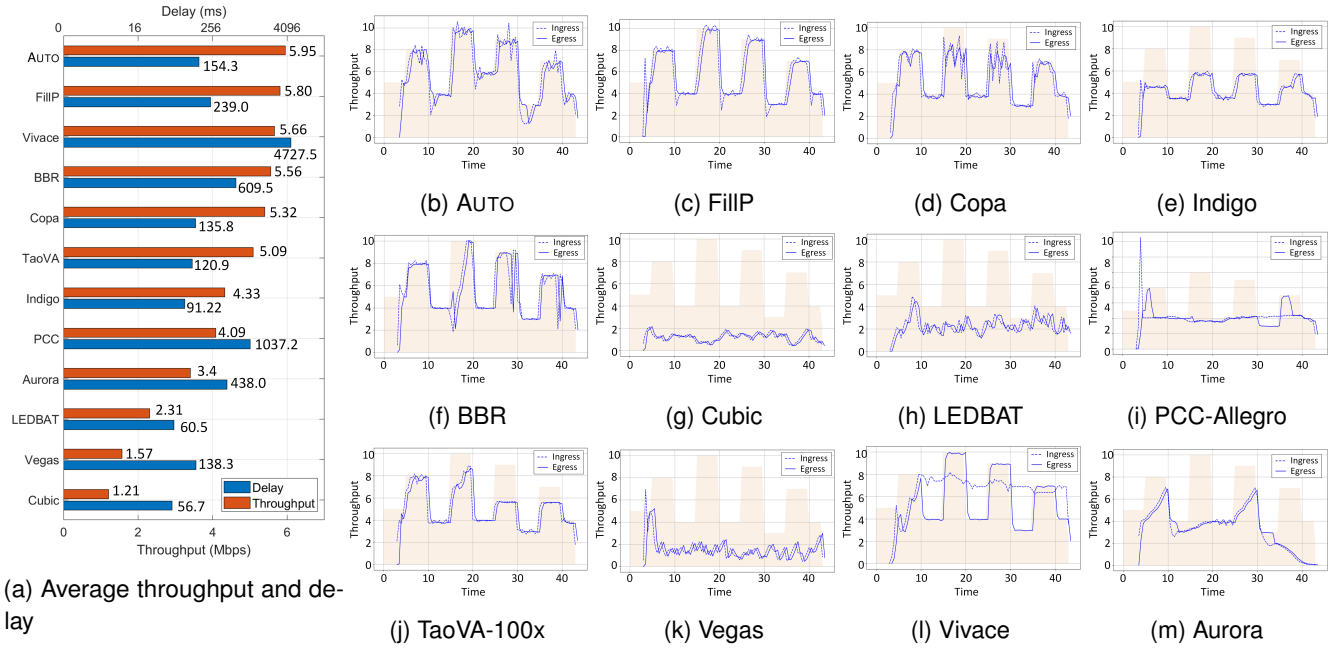


Figure 7: Performance evaluations in the rapidly changing network scenario.

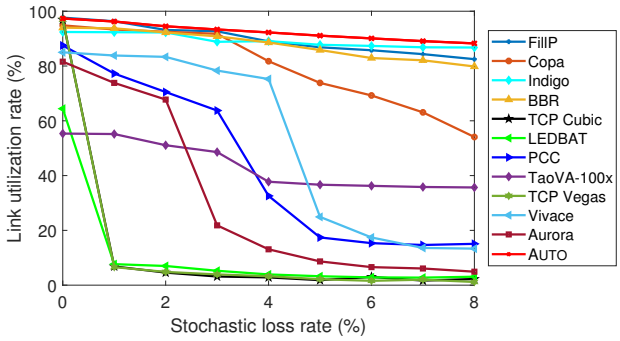


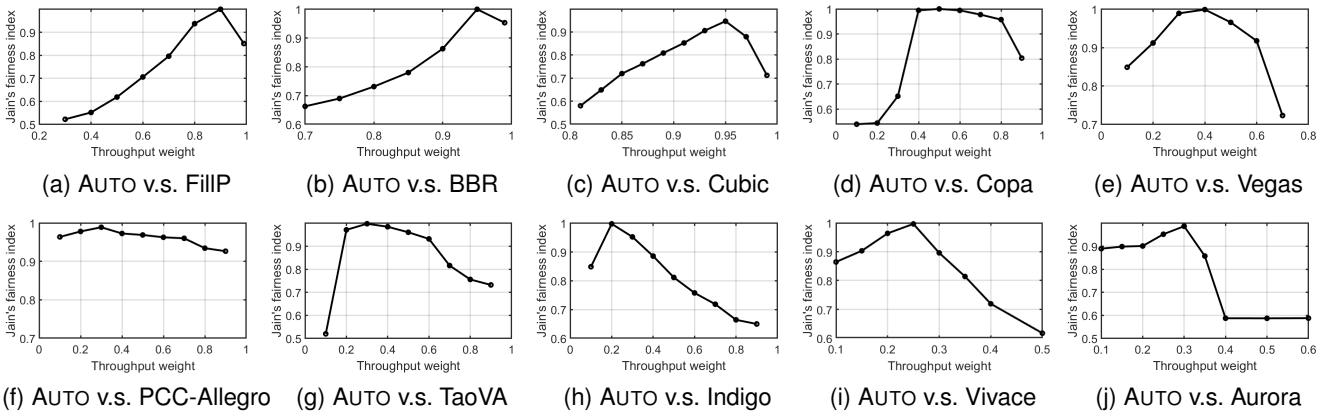
Figure 8: Link utilization v.s. stochastic loss rate.

decrease the sending rate. We test AUTO against other algorithms under an emulated link with a rate of 12 Mbps bandwidth and an RTT of 200 ms. To further test the robustness of the trained policy model, we let the stochastic loss rate of the link range from 0% to 8%, which is beyond our training range [0%, 5%].

As shown in Figure 8, AUTO achieves consistent high link utilization under all stochastic loss rates. Even when the stochastic loss rate is set to 8%, which is outside the training range, AUTO achieves 88.25% link utilization, which is 6.9% higher than the second-highest method FillP. The main reason is that AUTO judges whether congestion occurs by jointly considering the latency ratio, the latency gradient, the sending ratio, and the loss rate. Since the stochastic loss rate only influences the last two features, AUTO still can detect congestion accurately. With a similar reason, Indigo also achieves high link utilization. In contrast, the link utilization of Aurora falls sharply when the stochastic loss rate larger than 2% due to the fixed preference. BBR adjusts the sending rate according to the estimated bandwidth and minimum latency, thus, it is less affected by the stochastic loss rate. Since packet loss is not considered in the packet arrival model, the performance of Copa degrades when the stochastic loss rate is larger than 3%. TaoVA fails to consider lots of real network parameters such as stochastic loss rate and link buffer size and its link utilization ranges from 35% to 55%. For PCC-Allegro and Vivace that adopt the fixed weight on the packet loss rate in the utility function, their performance is poor when the stochastic packet loss rate is large since the calculated utilities are always low. As for TCP Cubic, TCP Vegas and LEDBAT, they are designed for reliable network environments and are sensitive to packet loss. They achieve relatively high link utilization when there is no stochastic packet drop on the link, and their link utilization drops to about 3% when the stochastic loss rate increases to 8%.

5.5 Fairness Against Different CC Schemes

Last, we show that AUTO *is competitive-configurable and can achieve fairness against different CC schemes with different competitiveness*, which solves the pain point of existing offline learning-based CC methods [5]. We first set up two flows on a dumbbell topology using CoCo-Beholder [39] with one flow using AUTO and competing with another flow using other CC schemes. The bottleneck link has 100 ms RTT, 20 Mbps capacity, and a queue size of 200. After 30 seconds running, we evaluate the Jain's Fairness Index (JFI) of the



(a) AUTO v.s. FillP (b) AUTO v.s. BBR (c) AUTO v.s. Cubic (d) AUTO v.s. Copa (e) AUTO v.s. Vegas

(f) AUTO v.s. PCC-Allegro (g) AUTO v.s. TaoVA (h) AUTO v.s. Indigo (i) AUTO v.s. Vivace (j) AUTO v.s. Aurora

Figure 9: JFI between AUTO and other CC schemes v.s. the throughput weight (ω_t). The delay weight is set to $1-\omega_t$.

two flows, which is calculated by $\bar{\boldsymbol{x}}^2/\overline{\boldsymbol{x}^2}$ and $\boldsymbol{x}=[x_{\text{AUTO}},x_c]$ is the throughput of AUTO and the competing scheme respectively. The JFI ranges from 0.5 (worst case, the throughput of one flow is 0) to 1 (best case, the two flows have the same throughput).

As shown in Figure 9, the JFI first increases with the throughput weight and then decreases. This is because the competitiveness of AUTO increases along with the throughput weight so that the JFI increases before AUTO achieves the same competitiveness as the competing flow and decreases after. More specifically, with a small throughput weight, AUTO is extremely delay-sensitive and tends to decrease the sending rate when the competing flow causes the increase of the queuing delay. On the contrary, if higher weight is put on throughput, AUTO will be more competitive and decrease the sending rate only if the latency ratio is large.

Therefore, AUTO can always find a suitable preference such that the two flows achieve comparable throughput on the shared link. When competing with CC schemes with high competitiveness, AUTO achieves the fairness by increasing the throughput weight. For example, FillP, BBR and TCP Cubic have high competitiveness for the first two adjust the sending rate based on the estimated bandwidth and the last reduces the congestion window size only when there is packet loss. As shown in Figures 9(a)-(c), by setting preference to (0.9, 0.1), (0.95, 0.05) and (0.95, 0.05), AUTO achieves the fairness against FillP, BBR and TCP Cubic respectively. In contrast, by putting higher weight on delay, AUTO can play well with delay-sensitive CC algorithms. Among the selected CC schemes, Indigo, Vivace and Aurora are more delay-sensitive due to their selected features or the rate adjusting mechanism. As shown in Figure 9(h)-(j), AUTO achieves the fairness against these three algorithms by setting the preference to (0.2, 0.8), (0.25, 0.75), and (0.3, 0.7) respectively.

Meanwhile, the competitive-configurable characteristic also supports users to explicitly set the priority for different flows. For example, users can adjust the throughput weight of flows that require high bandwidth (e.g., file downloading) to larger than that of delay-sensitive flows (e.g., online meeting). By doing so, the performance of delay-sensitive can be guaranteed while other flows only use the residue bandwidth.

6 Conclusion

We proposed AUTO, an adaptive CC algorithm based on multi-objective reinforcement learning for the heterogeneous satellite-ground integrated network. It doesn't target at finding the optimal result for a single preference, but instead aims for finding *optimal policies for all possible preferences*. Thus, AUTO can cope with heterogeneous network environments and diversified application requirements using a single agent, by taking different preferences as input. To adjust the preference adaptively in different environments, we next proposed an environment-adaptive preference selection method by training a preference adaptation model. To train the agent on emulated environments and to facilitate further research, we developed a training suite Pantheon-Gym for MORL-based CC. Evaluation results show that AUTO achieves consistent high performance in representative network environments, adapts to rapid network changes, is resilient to stochastic packet loss, and can achieve fairness against different CC schemes.

Acknowledgments

We would like to thank the anonymous reviewers for their thoughtful comments. This work was supported in part by the National Natural Science Foundation of China projects (Nos. 61832013), in part by STCSM (Science and Technology Commission of Shanghai Municipality) AI project (No.19511120300), in part by the National Key Research and Development Program of China (No. 2019YFB2102204), and in part by the Huawei Technologies Co., Ltd projects (Nos. YBN2019105155 and YBN2020085026). Feilong Tang is the corresponding author of this paper.

References

[1] J. Liu, Y. Shi, Z. M. Fadlullah, and N. Kato, "Space-air-ground integrated network: A survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 2714–2741, 2018.

[2] F. Tang, "Dynamically adaptive cooperation transmission among satellite-ground integrated networks," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1559–1568.

[3] M. Polese, F. Chiariotti, E. Bonetto, F. Rigotto, A. Zanella, and M. Zorzi, "A survey on recent advances in transport layer protocols," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3584–3608, 2019.

[4] V. Arun and H. Balakrishnan, "Copa: Practical delay-based congestion control for the internet," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2018, pp. 329–342.

[5] N. Jay, N. H. Rotman, P. Godfrey, M. Schapira, and A. Tamar, "Internet congestion control via deep reinforcement learning," *arXiv preprint arXiv:1810.03259*, 2018.

[6] H. Obata, K. Tamehiro, and K. Ishida, "Experimental evaluation of tcp-star for satellite internet over winds," in *Proceedings of the International Symposium on Autonomous Decentralized Systems (ISADS)*. IEEE, 2011, pp. 605–610.

[7] F. Y. Yan, J. Ma, G. D. Hill, D. Raghavan, R. S. Wahby, P. Levis, and K. Winstein, "Pantheon: the training ground for internet congestion-control research," in *USENIX Annual Technical Conference (ATC)*, 2018, pp. 731–743.

[8] S. Abbasloo, C.-Y. Yen, and H. J. Chao, "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*, 2020, pp. 632–647.

[9] S. Ha, I. Rhee, and L. Xu, "Cubic: a new tcp-friendly high-speed tcp variant," *ACM SIGOPS operating systems review*, vol. 42, no. 5, pp. 64–74, 2008.

[10] R. Al-Saadi, G. Armitage, J. But, and P. Branch, "A survey of delay-based and hybrid tcp congestion control algorithms," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3609–3638, 2019.

[11] V. Sivakumar, T. Rocktäschel, A. H. Miller, H. Küttler, N. Nardelli, M. Rabbat, J. Pineau, and S. Riedel, "Mvfst-rl: An asynchronous rl framework for congestion control with delayed actions," *arXiv preprint arXiv:1910.04054*, 2019.

[12] Y. Kong, H. Zang, and X. Ma, "Improving tcp congestion control with machine intelligence," in *Proceedings of the 2018 Workshop on Network Meets AI & ML*, 2018, pp. 60–66.

[13] W. Li, F. Zhou, K. R. Chowdhury, and W. Meleis, "Qtcp: Adaptive congestion control with reinforcement learning," *IEEE Transactions on Network Science and Engineering*, vol. 6, no. 3, pp. 445–458, 2018.

[14] H. Mao, P. Negi, A. Narayan, H. Wang, J. Yang, H. Wang, R. Marcus, M. K. Shirkoohi, S. He, V. Nathan *et al.*, "Park: An open platform for learning-augmented computer systems," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019, pp. 2490–2502.

[15] X. Nie, Y. Zhao, Z. Li, G. Chen, K. Sui, J. Zhang, Z. Ye, and D. Pei, "Dynamic tcp initial windows and congestion control schemes through reinforcement learning," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1231–1247, 2019.

[16] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, "Pcc: Re-architecting congestion control for consistent high performance," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2015, pp. 395–408.

[17] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and M. Schapira, "Pcc vivace: Online-learning congestion control," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2018, pp. 343–356.

[18] T. Meng, N. R. Schiff, P. B. Godfrey, and M. Schapira, "Pcc proteus: Scavenger transport and beyond," 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*, 2020, pp. 615–631.

[19] K. Winstein and H. Balakrishnan, "Tcp ex machina: Computer-generated congestion control," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 123–134, 2013.

[20] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[21] T. Issariyakul and E. Hossain, "Introduction to network simulator 2 (ns2)," in *Introduction to network simulator NS2*. Springer, 2009, pp. 1–18.

[22] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, "Network simulations with the ns-3 simulator," *SIGCOMM demonstration*, vol. 14, no. 14, p. 527, 2008.

[23] W. Li, H. Zhang, S. Gao, C. Xue, X. Wang, and S. Lu, "Smartcc: A reinforcement learning approach for multipath tcp congestion control in heterogeneous networks," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 11, pp. 2621–2633, 2019.

[24] Z. Xu, J. Tang, C. Yin, Y. Wang, and G. Xue, "Experience-driven congestion control: When multi-path tcp meets deep reinforcement learning," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1325–1336, 2019.

[25] F. Ruffy, M. Przystupa, and I. Beschastnikh, "Iroko: A framework to prototype reinforcement learning for data center traffic control," *arXiv preprint arXiv:1812.09975*, 2018.

[26] T. L. Hayes, K. Kafle, R. Shrestha, M. Acharya, and C. Kanan, "Remind your neural network to prevent catastrophic forgetting," in *European Conference on Computer Vision*. Springer, 2020, pp. 466–483.

[27] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of the International conference on machine learning (ICML)*, 2016.

[28] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," *IEEE Transactions on Neural Networks*, vol. 16, pp. 285–286, 2005.

[29] j. schulman, p. moritz, s. levine, I. M. Jordan, and p. abbeel, "High-dimensional continuous control using generalized advantage estimation," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[30] R. Yang, X. Sun, and K. Narasimhan, "A generalized algorithm for multi-objective reinforcement learning and policy adaptation," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019, pp. 14 610–14 621.

[31] T. L. Watson and T. R. Haftka, "Modern homotopy methods in optimization," *Computer Methods in Applied Mechanics and Engineering*, pp. 289–305, 1989.

[32] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[33] T. Morimura, M. Sugiyama, H. Kashima, H. Hachiya, and T. Tanaka, "Nonparametric return distribution approximation for reinforcement learning," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2010.

[34] S. Kullback and R. A. Leibler, "On information and sufficiency," *Annals of Mathematical Statistics*, vol. 22, pp. 79–86, 1951.

[35] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, pp. 1735–1780, 1997.

[36] J. Shore and R. P. Johnson, "Properties of cross-entropy minimization," *IEEE Trans. Inf. Theory*, vol. 27, pp. 472–482, 1981.

[37] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in neural information processing systems*, 2019, pp. 8026–8037.

[38] Y. Gu, *UDT: a high performance data transport protocol*. University of Illinois at Chicago, 2005.

[39] E. Khasina, "The coco-beholder: Enabling comprehensive evaluation of congestion control algorithms," *arXiv preprint arXiv:1912.10531*, 2019.

[40] T. Li, K. Zheng, K. Xu, R. A. Jadhav, T. Xiong, K. Winstein, and K. Tan, "Tack: Improving wireless transport performance by taming acknowledgments," in *Proceedings of the ACM SIGCOMM Conference*, 2020, pp. 15–30.

[41] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control," *Queue*, vol. 14, no. 5, pp. 20–53, 2016.

[42] D. Rossi, C. Testa, S. Valenti, and L. Muscariello, "Ledbat: the new bittorrent congestion control protocol," in *Proceedings of the International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2010, pp. 1–6.

[43] A. Sivaraman, K. Winstein, P. Thaker, and H. Balakrishnan, "An experimental study of the learnability of congestion control," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 479–490, 2014.

[44] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "Tcp vegas: New techniques for congestion detection and avoidance," in *Proceedings of the ACM SIGCOMM Conference*, 1994, pp. 24–35.