Precise and Adaptable: Leveraging Deep Reinforcement Learning for GAP-based Multipath Scheduler

Binbin Liao*†,Guangxing Zhang*,Zulong Diao* and Gaogang Xie‡
*Institute of Computing Technology, Chinese Academy of Sciences, China
†University of Chinese Academy of Sciences, China
‡Computer Network Information Center, Chinese Academy of Sciencecs, China
{liaobinbin, guangxing, diaozulong}@ict.ac.cn, xie@cnic.cn

***Abstract*—Using multiple network interfaces to accelerate data transfer is an attractive feature on multi-homed endhost. The key component in a multipath system such as MPTCP is the scheduler, which determines how to distribute the packets over multiple paths. In this paper, we propose GAPS, a new multipath scheduler aiming at decreasing the out-of-order queue size (OQS) under heterogeneous paths. In GAPS, a deep reinforcement learning agent monitors the network states, adjusts the GAP value of each path, and maximizes MPTCP's reward utility. GAPS is precise in that it searches the optimal action policy, and only has 1.2% to 3.3% deviation from the true GAP. It is adaptable in that it performs better in varying network conditions and congestion control algorithms. In the controllable and realistic experiments, GAPS decreases the subflows' 99th percentile OQS by up to 68.3%. It allows an increase of 12.7% in application goodput with bulk traffic while reducing application delay by 9.4% as compared to the state-of-the-art schedulers. For the short and long MPTCP flows, GAPS has the flow completion time (FCT) reduction by 13% and 21%, respectively.**

I. Introduction

Currently, the most widely used multipath solution is MPTCP [1], which enables unmodified applications to leverage multiple network interfaces, such as Cellular, WiFi, and Ethernet. It has already been implemented in the linux kernel and supported by many commercial products [2]. MPTCP adds a shim layer between TCP and Application layer, which can establish several TCP subflows over each network interface. The scheduler determines the amount of packets to distribute to each subflow from a sending queue. However, the heterogeneity of TCP subflows makes it challenging to design a good scheduler. As discussed in [3]–[6], a wrong scheduler causes severe packet out-of-order arrival at the receiver side, when the packets scheduled on the faster path have to wait for the packets on the slower path, and thus to arrive in a shared out-of-order queue of the receiver. This phenomenon is also known as head-of-line (HoL) blocking [7]. Because of delaying the packets delivery, HoL makes the application less interactive and results in poor user experience.

Endhost must maintain a large buffer size to reorganize the out-of-order packets. If the host buffer is limited, it will cause a dramatic reduction in application performance, as the receive buffer has to exclude some packets. Besides, a data-level acknowledgement (Data ACK) of the blocking packets in the slower subflow will cause bursting traffic [6], as the faster subflow accumulates huge free send windows (SWND) during the waiting time. If the in-network buffer is not bigger enough to store these bursting packets, it will cause severe packet loss and congestion window capping.

Several works have been made to address these problems. DAPS [8] firstly proposes the concept of out-of-order sending for in-order arrival. BLEST [9] calculates the waiting packets of the faster subflow based on the properties of paths, such as round-trip-time (RTT), congestion windows (CWND), and SWND. Inheriting these ideas, ECF [4] adds a deviation to calibrate the value of waiting packets and achieves better performance than DAPS and BLEST. STMS [6] finally maintains a GAP value for each subflow to preallocate future packets, which performs better than ECF. Through simple formula-derivation in Section.II-B, we find that DAPS, BLEST, and ECF are also the GAP-based scheduler, which transforms the uncontrollable OQS problem from the receiver side to the adjustable GAP value at the sender side.

Our experiment shows that GAP-based schedulers decrease the OQS with their improvement of algorithms. However, they are inadaptable to various network environments, especially when the packet loss rate (PLR), the available bandwidth, and default buffer [9] are changeable. As none of the GAP-based schedulers detect these factors to distribute the sending packets. Moreover, the congestion controller (CC) algorithms greatly affect the OQS, since most schedulers take CWND as the main parameter to adjust their GAP value. We also observe that the GAP values are imprecise when MPTCP maintains more than three active subflows because their human experiences only consider partial factors to calculate the blocking packets and suit the specific scenarios.

In this paper, we design GAPS, which leverages a Deep Deterministic Policy Gradient (DDPG) [10] to train four deep neural networks (DNNs) through wide experiences in state space and generates the continuous control policy for adjusting GAP value. For each round of Data ACK, it gets OQS as the reward utility to optimize the overall performance of MPTCP. Extensive experiments show that GAPS significantly outperforms the state-of-the-art schedulers in various network conditions. The main contributions and results of this paper are summarized as follows.

- We present a DDPG-based deep reinforcement learning (DRL) framework, which realizes our experience-driven philosophy on MPTCP packet scheduling. We utilize DNNs as the function approximator to learn the GAP

ISBN 978-3-903176-28-7

adjusting actions according to the runtime states without relying on any accurate mathematical model.
- We integrate a Transformer network [11] as the representation layer of the actor-critic networks to encode the raw states of dynamic subflows. With positional encoding, each subflow obtains a unique label to determine its GAP value. The relevant degree between any two subflows is calculated by the self-attention layer, which increases the accuracy of GAP.
- We add a new flag-bit in the reserved area of Data Sequence Signal (DSS) option for each Data ACK. Cooperating with the A-bit in the MP_CAPABLE option, the OQS is carried back from the receiver side to the scheduler at the sender side as the reward utility of critic network.
- We implement GAPS in the Linux kernel and evaluate it over controllable and realistic conditions. The experiments show that GAPS decreases the subflows' 99th percentile OQS by 68.3%, improves the aggregated goodput by up to 12.7%, and reduces the application delay by 9.4% as compared to the state-of-the-art schedulers.

II. Background and Motivation

A. State-of-the-practice schedulers

Whenever the MPTCP stack is ready to send data, its scheduler is invoked to execute two procedures: (a) choosing an available subflow among the set of TCP subflows and (b) deciding which segment to send considering the properties of the subflow. We describe a modular scheduler function analogous to [2]. The pseudo-code implementation of this behavior is illustrated in Algorithm.1, which allows us to design the scheduler in a modular infrastructure.

Within the modular framework, we begin with a simple round-robin scheduler (RR), which fully utilizes the capacity of each path but ignores the external influences on subflows. If any active subflow is poor or broken, RR suffers a long-term blocking or even severe packet retransmission. Considering these heterogeneous subflows where significant delay differences are observed between them [12], [13], we discuss the popular delay-based schedulers [3], [14], including Minimum RTT First (MinRTT) with Retransmission Penalization (RP) and Bufferbloat Mitigation (BM) mechanism.

Algorithm 1 An MPTCP connection ready to send data. mptcp→sched represents the specific callback of schedulers

```
1: subflow = mptcp→sched→get_subflow();
2: while subflow != NULL do
3:     data = mptcp→sched→get_data(subflow);
4:     while data != NULL do
5:         send_data(subflow, data);
6:         data = mptcp→sched→get_data(subflow);
7:     end while
8:     subflow = mptcp→sched→get_subflow();
9: end while
```

MinRTT reduces the delay for the interactive applications but still suffers out-of-order arrival [14], head-of-line blocking [7], and receive-window limitation [15]. Through calculating RTT difference, RP and BM reinjects the blocking segment and drains the in-network buffer by capping CWND. Due to their opportunistic nature, RP and BM can not always reduce the RTT to the optimal value but cause a recurrence of blocking [9]. Their frequent reinjection also wastes available bandwidth and transmission time. Besides, MPTCP bursting pattern exacerbates the in-network bufferbloat, which is unsolved and detailedly discussed in [6]. To overcome these challenges, we take more factors into account to explore the following GAP-based schedulers, including DAPS, BLEST, ECF, and STMS.

B. State-of-the-art schedulers

We demonstrate a case with only two active subflows and denote CWND_f, CWND_s, RTT_f, RTT_s as the available CWND and RTT of the faster subflows and slower subflows. Assuming that there are 100 packets in the sending queue, which have not been assigned to any subflow. As shown in Fig.1, if RTT_f has redundant CWND_f, the packets are scheduled to it. If RTT_f does not have available space, the packets are scheduled to the RTT_s with CWND_s. Rather than taking packets whose sequence numbers are right after those transmitted on the faster path, the slower subflow sends packets with bigger sequence numbers. It leaves a sequencing GAP for the faster path to send the corresponding packets in the future. By the time the packets from the slower path arrive, all packets from the faster path (including GAP) have already arrived without any interval.

Delay Aware Packet Scheduler (DAPS) aims for in-order arrival at the receiver to prevent its buffer from blocking. It leverages RTT and the available CWND to calculate the GAP value. The sequence number of the packets transmitted on the slower path is determined by the ratio between the two paths, and the available congestion window on the faster path. Then the GAP value of DAPS is estimated as:

$$GAP_f = min(\lfloor \frac{RTT_s}{RTT_f} \rfloor, CWND_f - unACK_f) \quad (1)$$

BLocking ESTimation-based scheduler (BLEST) proposes a proactive scheduler to decide at packet scheduling time whether sending packets over the slower subflow blocks the SWND or not. It assumes that all segments will occupy space in SWND for at least one RTT_s, if they are sent on the slower subflow. Therefore, the amount of data that be sent on the faster subflow during RTT_s is the value of GAP, and derived as:

$$GAP_f = (CWND_f + \frac{\frac{RTT_s}{RTT_f}}{2}) * (\frac{RTT_s}{RTT_f} - 1) \quad (2)$$

Earliest Completion First scheduler (ECF) monitors subflow not only RTT estimates but also the bandwidths. By determining whether using a slower path for the injected traffic will cause faster paths to become idle. ECF more efficiently

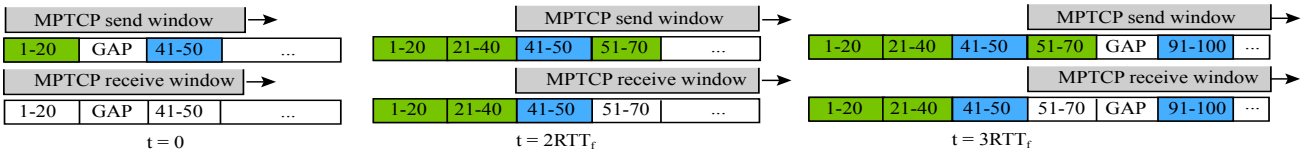

Fig. 1. The demonstration of GAP concept with green and blue for packets over the faster and slower subflows respectively.

utilizes faster paths by removing the idle periods. Thus, the GAP value is the packets injection of faster subflow during the idle time. By using a hysteresis value β and a standard deviation δ of RTT, the formulation of ECF is then transformed as:

$$GAP_f = \frac{(2 * RTT_f + \delta) * CWND_s}{RTT_s} - \frac{((1+\beta) * (RTT_s + \delta) - RTT_f) * CWND_f}{RTT_f} \quad (3)$$

Slide Together Multipath Scheduler (STMS) realizes the Data ACKed reflects the degree of OQS. Denoting $delta\_gap$ and $adjust\_interval$ as the adjusting step and interval, STMS monitors the $data\_acked$ for each Data ACK, then if $data\_acked$ is bigger than two and sent from the slower path, $delta\_gap$ = $data\_acked$; otherwise, $delta\_gap$ = - $data\_acked$. An exponentially weighted moving average (EWMA) of $delta\_gap$ over $adjust\_interval$ adjusts its value as below:

$$GAP_f += EWMA(delta\_gap, adjust\_interval) \quad (4)$$

After passing GAP as an argument to the callback function $get\_data(subflow, GAP)$ in Algorithm. 1, the design difficulty is then translated to adjust the GAP value. Any bias from its TRUE value will cause packets out-of-order arrival. We rebuild the controllable testbed, which has been setup in STMS or ECF. The RTT$_f$, RTT$_s$ are randomly set with [20ms,50ms] and [50ms,100ms]. The bandwidths of both paths are set to 50Mbps. The packet loss rate are set to 0.01%. Based on [13], [16], the in-network buffer of routers are set to 100 packets for WiFi and 3000 packets for LTE. With a coupled balia [17], both receive and send buffers are set to the Linux default size (6MB). Unless otherwise noted, these initial setups keep the same in this paper.

We monitor the OQS of MPTCP as the waiting bytes in the out-of-order queue [18]. Fig. 2 shows that the average OQS of GAP-based schedulers is smaller than the MinRTT scheduler under various network settings. The finer GAP-adjusting in STMS earns the smallest average OQS, which indicates that its GAP value is more accurate and makes packets arrive in-order. However, we also observe that PLR significantly affects the OQS, as none of the GAP-based schedules detect the PLR to schedule the packets just like [19].

We consider the effects of different CCs implemented in [2]. ECF, BLEST, and DAPS are susceptible to CCs, as their GAP-values directly derives from CWND. STMS has the smallest deviation since it is CC agnostic [6]. For each Data ACK, the PDR, receive windows (RWND), and Data ACKed packets affect the OQS in a similar CCs fashion. If we consider all these factors, the GAP value cannot be confirmed by a simple mathematical model but with a high-dimensional tensor. This is why the existing schedulers ignore some important attributes of a heterogeneous network.

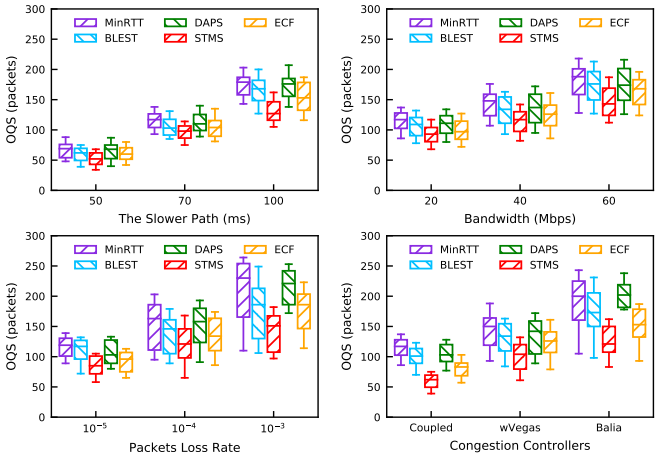

Fig. 2. The distribution of OQS under various network environments.

To clarify the performance of GAP-based schedulers, we measure the OQS by varying the number of active subflows. As shown in Fig. 3, when MPTCP establishes more than three subflows, MinRTT has a smaller OQS than all GAP-based schedulers. Besides, we adjust the GAP value of each subflow manually and search their TRUE values by inspecting the minimum OQS. Fig. 3 shows that the above scheduler's GAP values deviate from the TRUE values by about 10% to 15% with the increasing active subflows. We explain that the active subflows for each round have to be sorted solely by their RTT, which mistakes the property of subflows and cumulates the error of each GAP. Moreover, the design of the above GAP-based schedulers involves parochial human experience, which assumes a specific network environment with no more than two subflows.

In this paper, we dissect deep reinforcement learning techniques, which is the subfield of machine learning concerned with decision making and action control. It studies how an agent can learn to achieve goals in a complex, uncertain environment. DRL has achieved excellent results in many challenging environments with advances in DNNs: DeepMind's Atari results [20] and AlphaGo [21] used DRL algorithms, which make few assumptions about their environments and thus can be generalized in other settings. Inspired by these

results, we are motivated to enable DRL for automatic GAP adjusting to adapt the heterogeneous network environments.

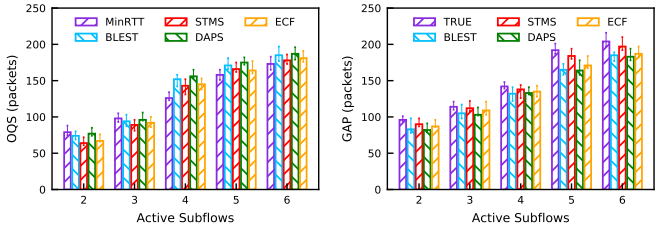

Fig. 3. The relationship between OQS and GAP under multiple active subflows.

III. Approach

DRL is a deep version of regular reinforcement learning (RL), which combines insights from recent advances in deep learning and RL. Considering a standard RL system, which consists of an agent interacting with an environment in discrete timesteps. At each timestep t, the agent receives an observation of the measurement state s_t of the uncertain environment, takes actions a_t according to its policy π, and receives a reward r_t. The goal in RL is to learn a policy $\pi(s_t)$, which maps its state to a deterministic action or to a probability distribution over actions, such that the cumulative discounted reward $\sum_{t=0}^{T} \gamma^t r(s_t, a_t)$ is maximized, where $\gamma \in [0, 1]$ is the discounting factor.

For most practical problems, it is infeasible to learn all possible state-action pairs as the high-dimensional state space. Thus function approximation technique [22] is commonly used to learn the action policy. A function approximator is parameterized by a vector θ, whose size is much smaller (thus mathematically tractable) than the number of all possible state-action pairs. The function approximator can have many forms. Deepmind designs a Deep Q-Networks (DQN) [20] to play Atari game better than human experts, which extends the traditional Q-learning with a DNN approximator.

DQN takes a state-action pair (s_t, a_t) as the inputs, and outputs the corresponding Q-value $Q(s_t, a_t)$, which represents the expected discounted cumulative reward $Q(s_t, a_t; \theta) = E[R_t|_{s_t,a_t}; \theta]$. Then, the optimal policy $\pi^*(s)$ can be derived by applying a ϵ-greedy strategy that follows the greedy strategy with probability 1–ϵ, and selects a random action with probability ϵ. Since DQN refers to a neural network function approximator with weights vector θ^Q as a Q-network. The Q-network can be trained or updated by minimizing a sequence of loss functions $L(\theta^Q)$ that changes at each iteration.

$$L(\theta^Q) = E[(Q(s_t, a_t; \theta^Q) - y_t)^2] \tag{5}$$

where y_t is the target value for each state-action pair derived from the Bellman equation.

$$y_t = r(s_t, a_t) + \gamma * Q(s_{t+1}, \mu(s_{t+1})|_{\theta^Q}) \tag{6}$$

Although neural network or DNN has already been used as the function approximator before, it is known that such a non-linear function approximator is not stable and even leads to oscillation or divergence in the parameters. To improve the stability of training, DQN introduces two effective techniques: experience replay and the target network. By using experience replay, a DRL agent stores historical state transition samples into a replay buffer, and then updates the DNN by a mini-batch sampling from the replay buffer instead of the immediately collected transition. DRL agent thus breaks correlations in the observation sequence, and learns from a more independently and identically distributed past experience. Moreover, a separate target network is proposed for calculating target values y_t, which shares the same network structure as the original Q-network, and copies the weights $\theta^{Q'}$ of the target network from θ^Q.

However, DQN can only handle discrete and low-dimensional action spaces. Many tasks of interest, such as our GAP adjusting, have continuous and high dimensional action spaces. As the popular approach to continuous control is the policy gradient [23]. The actor-critic based approach called Deep Deterministic Policy Gradient (DDPG) [10] for DRL, which has integrated both DQN and the emerging deterministic policy gradient for continuous control. The basic idea behind DDPG is to maintain four DNNs simultaneously. Two DNNs, critic $Q(s_t, a_t; \theta^Q)$ and actor $\mu(s_t; \theta^\mu)$ with weights θ^Q and θ^μ, are trained on sampled mini-batches of replay buffer, where each item represents an experienced transition tuple (s_t, a_t, r_t, s_{t+1}) during the agent interacts with the environment. The other two copying DNNs, target actor $\mu(s_t; \theta^{\mu'})$ and target critic $Q(s_t, a_t; \theta^{Q'})$ are used for smooth updates of the actor and critic networks. For a probability state distribution ρ and the start distribution J, the parameters θ^Q and θ^μ of above DNNs are updated by the following gradient.

$$\begin{aligned}\nabla_{\theta^Q} L(\theta^Q) =& E_{(s_t,a_t)\sim\rho}[(y_t - Q(s_t, a_t; \theta^Q))* \\ & \nabla_{\theta^Q} Q(s_t, a_t; \theta^Q)]\end{aligned} \tag{7}$$

$$\begin{aligned}\nabla_{\theta^\mu} J \approx& E_{s_t\sim\rho}[\nabla_a Q(s, a; \theta^Q)|_{s=s_t,a=\mu(s_t)}* \\ & \nabla_{\theta^\mu} \mu(s; \theta^\mu)|_{s=s_t}]\end{aligned} \tag{8}$$

The target networks of actor and critic are then updated by having them slowly track the two learned deep neural networks with $\tau \ll 1$, as below.

$$\theta' \leftarrow \tau\theta + (1 - \tau)\theta' \tag{9}$$

Combining with the powerful DDPG, several works have achieved excellent performance in network transmission systems, such as adjusting flow priority of datacenter in AUTO [24], adapting bitrate of video streaming in Pensieve [25], and classifying packets of high-speed switches in NeuroCuts [26]. With similar designing challenges, we carefully select the state space, action space, and reward function, which are critical to the ultimate performance of our system.

Because adding more parameters into the state space does not necessarily result in noticeable performance improvement, which instead increases the data collection overhead and training complexity. From the experiments in section II-B, we know that RTT, CWND, PDR, and PLR are correlated

with GAP adjusting. Data ACKed and the host buffer size are proved to be related to OQS-latency in [6]. Then, the value of RTT, CWND, PDR, PLR, Data ACKed, and RWND construct the attributes of each state input.

State space of an MPTCP flow at epoch t is s_t = [s_{t1},$\cdots$, s_{ti},$\cdots$, s_{tN}], and s_{ti} = [d_{ti}, c_{ti}, b_{ti}, l_{ti}, w_{ti}, k_{ti}], where s_{ti} is the state of subflow i of an MPTCP connection at t; d_{ti}, c_{ti}, b_{ti}, l_{ti}, w_{ti}, and k_{ti} are the corresponding RTT, CWND, PDR, PLR, RWND and DATA ACKed from the subflow i respectively; and N is the total number of the fluctuant subflows, which are established by the MP_JOIN option or closed by the RST option.

Action space of the active subflows at epoch t is a_t = [g_{t1},$\cdots$, g_{ti},$\cdots$, g_{tN}], where g_{ti} specifies how much change needs to be made to the GAP value of subflow i at t. We denote the minimum tuning unit as one TCP packet (about 1KB), which is finer than the STMS. Then, the positive, negative, and zero actions lead to increasing, reducing, and staying at the same GAP value size, respectively. Note that at each epoch t, GAPS takes actions on one target MPTCP flow of N TCP subflows.

Reward function of the critic network at epoch t is r_t = $\sum_i^N f(i,t)$, where $f(i,t)$ gives the utility function of active TCP sublfow i. Because OQS at the receiver side is the goal of optimizing, which is expected to construct the reward function. As shown in Table. I, we add a Q-bit flag in the reserved area of DSS option for each Data ACK. Cooperating with the flag A-bit in the MP_CAPABLE option, we occupy the two octets of CHECKSUM area to return the current OQS to the sender side when Q&A = 1. Then we denote r_t = -OQS since all active subflows share the out-of-order queue.

TABLE I
DATA ACK WITH Q-BIT FLAG

Kind	Length	Subtype	Reserved	Q	F	M/m	A/a
Data-level Acknowledgment							
Data Sequence Number							
Subflow Sequence Number							
Data-level Length			Out-of-order Queue Size				

Because of the cumulative ACK mechanism, each Data ACK may free huge space and consume several subflow-level RTT, which satisfies the time cost of packets scheduling round from the sending queue at epoch t.

Except for the above considerations, the major obstacles of GAPS focuses on three points: (a) the number of active subflows is dynamically changed, which varies the attributes of subflows; (b) the correlation degree between any two subflows with their attributes is the premise of calculating the GAP value; (c) each subflow must have a unique symbol position to map its GAP value. Recurrent Neural Networks (RNN) or Long Short-Term Memory (LSTM) has been widely used to memory and encode the variable-length input sequences for DNN in [27]–[30]. However, RNN-based methods suffer from massive training and processing complexity [11], which becomes critical and causes poor correlation degree at longer input sequence lengths, as memory constraints limit batching across neural layers. Google's Transformer model [11] allows for significantly more parallelization and deals with the above three obstacles by using Input-Embedding [31], Multi-head Attention, and Positional-Encoding [32], respectively. We, thus, integrate the stacked Transformer's encoders as the representation network of the DDPG framework.

IV. SYSTEM DESIGN

As illustrated in Fig.4, we present the system design and implementation of GAPS. The core idea behind our system is to train a DRL agent to perform accurate GAP adjusting actions for active TCP subflows on an MPTCP connection, and maximize the cumulative reward utility (i.e., minimizing the OQS). As mentioned before, all the TCP active subflows are fluctuant and only refer to a single sending buffer, where GAPS is running between them.

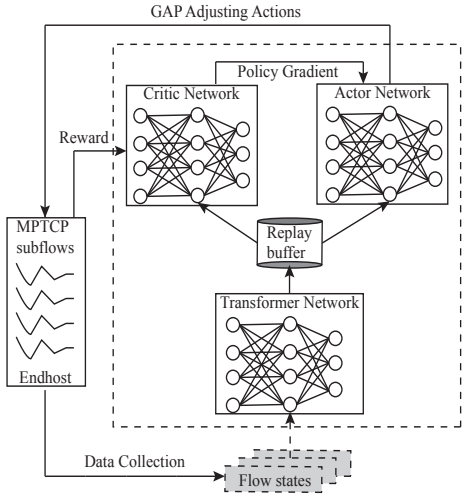

Fig. 4. The system diagram of GAPS.

To realize this formulation, we design the architecture of GAPS, which consists of the following two core elements.

Transformer network takes the raw state s_t of all active TCP subflows as the input at each decision epoch t, and generates output (i.e., a matrix with attached information), which are then used by the actor-critic networks for deriving actions.

As mentioned above, the major obstacle of GAPS is to deal with the situation, in which the active subflows may change over time. Most DNN needs to have a fixed input sequence. A straightforward way to use a neural network here is to zero-pad or exclude some attributes, which lead to poor training. Considering that Transformer-model encodes a variable input sequence into a normalized matrix. We thus choose Transformer encoders to process the states of dynamic subflows for the DNNs. As illustrated in Fig. 5, each encoder is composed of two sub-layers. The first is a multi-head self-attention layer, and the second is a simple position-wise fully connected feed-forward network. Two normalized layer [33] are deployed as a residual connection [34] around each of the encoder. Each state s_{ti} is embedded into a vector through the

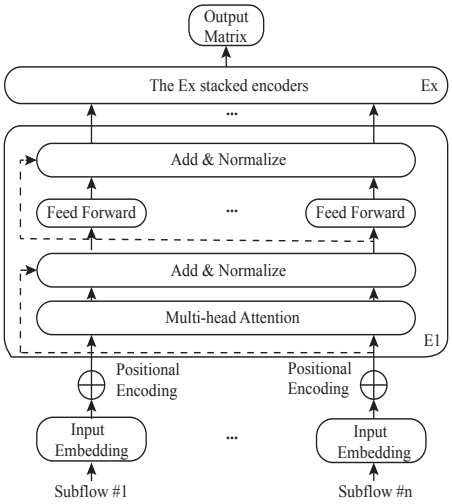

Fig. 5. The stacked Transformer encoders.

Input Embedding module. The unique position of each state s_{ti} is determined by a positional encoding. The vectors of s_t are then passed to the self-attention module with long-range dependencies [35]. After processed by E_x stacked encoders, the output is returned as the input of the actor-critic networks.

Actor critic network is trained through Algorithm.2, which is determined by Equation.6,7,8,9.

Algorithm 2 Actor critic algorithm for multipath scheduler

1: Randomly initialize critic network $Q(s,a;\theta^Q)$ and actor policy $\mu(s;\theta^\mu)$ with weights θ^Q and θ^μ;
2: Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu\prime} \leftarrow \theta^\mu$;
3: Initialize replay buffer $R \leftarrow \phi$;
4: Initialize a random process $\mathcal{N}$ for action exploration;
5: Receive initial state s' from Transformer network;
6: **for** each received Data ACK **do**
7: Select action $a_t = \mu(s_t;\theta^\mu) + \mathcal{N}_t$;
8: Execute a_t and observe reward r_t and state s_{t+1};
9: Store transition (s_t, a_t, r_t, s_{t+1}) into R;
10: Sample transitions (s_i, a_i, r_i, s_{i+1}) from R;
11: Calculate $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1};\theta^{\mu'})|\theta^{Q'})$;
12: Update critic network by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i,a_i;\theta^Q))^2$;
13: Update the actor policy by using the gradient: $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a;\theta^Q)\nabla_{\theta^\mu}\mu(s;\theta^\mu)$;
14: Update the target networks: $\theta^{Q'} \leftarrow \tau\theta^Q + (1 - \tau)\theta^{Q'}$, $\theta^{\mu\prime} \leftarrow \tau\theta^\mu + (1 - \tau)\theta^{\mu\prime}$;
15: **end for**

Firstly the algorithm randomly initializes parameters θ^μ of actor-network and θ^Q of the critic-network. The target networks of actor and critic networks copy their structures with initial parameters $\theta^{\mu'}$, and $\theta^{Q'}$. In order to sample the state transitions, we initialize the replay buffer R. As the major challenge in continuous action spaces is exploration. We initialize a noise process [36] $\mathcal{N}$ for action exploration. Then the Transformer network receives the initial observation state s' from the TCP subflows layer and decodes it to R. After that, we enter a training loop until the neural network is convergence. For each received Data ACK, the action is selected from $a_t = \mu(s_t;\theta^\mu) + \mathcal{N}_t$ according to the current actor policy and exploration noise. Then the action a_t is applied to the active TCP subflows at the sender side, the reward r_t and new state s_{t+1} are observed from the environment. The replay buffer R stores the newest state transition (s_t, a_t, r_t, s_{t+1}) and removes the oldest tuples when the buffer is full. And then a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) is sampled from the buffer R to update the neural networks. The target value y_i is calculated according to Equation.6. Then the critic network is updated according to Equation.7 by minimizing the loss function. According to Equation.7, the actor network is updated by using the sampled policy gradient with N transitions. Finally, the target networks are slowly updated according to Equation.9.

V. Implement

In our implementation, the Transformer network is constructed by E_x = 6 stacked encoders. The Input Embedding of each encoder is a vector of size 512. The actor-network has two fully-connected hidden layers with 48 and 48 neurons. The rectified linear function [37] is used for activation in the two hidden layers. The hyperbolic tangent function is used for activation in its output layer. The critic-network has two hidden layers same as the actor-network except for one extra output layer, which has only one linear neuron (with no activation function). During training, we use Adam [38] for learning the neural network parameters with a learning rate of 10^{-4} and 10^{-3} for the actor and critic, respectively. The soft target update is set to τ = 0.001. We set the default weight α = 0.5 to and use a discount factor γ = 0.99, which implies that 100 future time steps will influence current rewards. For compatibility, we leveraged the TFLearn deep learning library's TensorFlow API [39] to declare the neural network during both training and testing. Thus, in the operation of the GAPS system, it requires interaction between the kernel and userspace. To simplify the execution logic, we implement the packet scheduler in the kernel that executes the GAP adjusting of each subflows from the actor-network in the userspace by a system call $setsockopt()$. Besides, we call $getsockopt()$ to capture the raw network states and OQS metrics. For the rest parts, we compile MPTCP v0.95 [2] into two endhosts with Ubuntu 16.04 by making kernel configure, which includes the recent comparable schedulers and congestion control algorithms. An Openwrt [40] router installed with TC [41] connects the two endhosts and simulates the network settings in the controllable lab. The endhosts are equipped with three kinds of interface in the wild, including Ethernet (80Mbps), WiFi (50Mbps), and LTE (30Mbps, USB cellular modem). We maintain at most 8 subflows (each with 6 attributes) except for the wire to wire one.

VI. Evaluation

A. Evaluation in the lab

We first focus on the out-of-order queue size at the receiver side with multiple active subflows. Fig. 6 shows the CDF of OQS at the same endhost with different schedulers. GAPS causes the smallest OQS than other schedulers. The average OQS of STMS is around 0.31MB, but the 99th percentile of MPTCP connection in GAPS has OQS less than 100KB, which makes about 68.3% reduction. Simulating a poor network environment with 1% PLR and 50KB in-network buffer, We further test the GAP-value deviation under different active subflows. Fig. 7 demonstrates that GAPS only has 1.2% to 3.3% deviation from the inspecting true GAP, which is ten times more accurate than the existing schedulers.

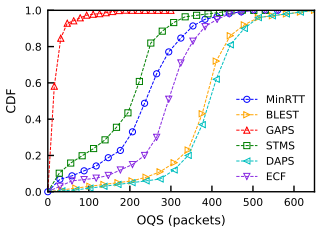

Fig. 6. The distribution of OQS.

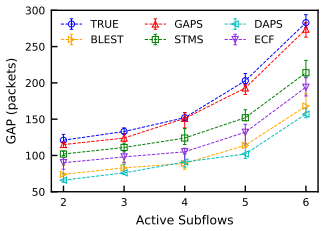

Fig. 7. The accuracy of GAP.

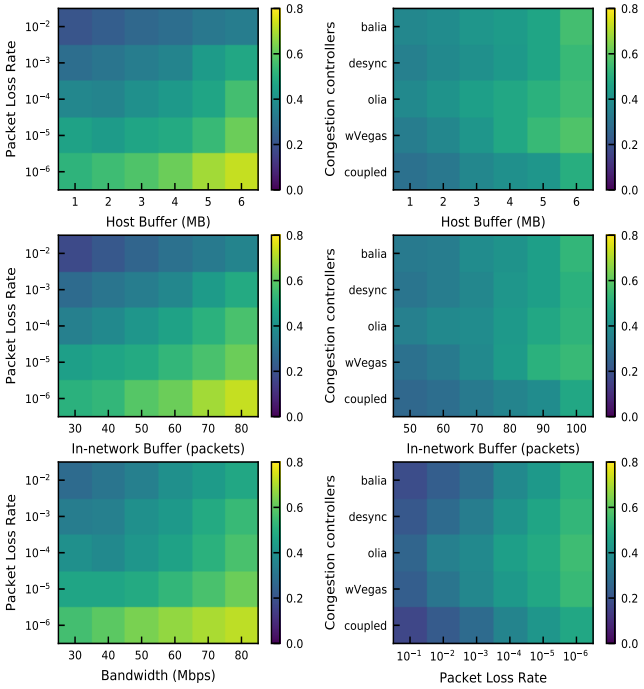

Fig. 8. The normalized OQS of GAPS relative to that of STMS.

By varying the default settings, we normalize the OQS of GAPS relative to that of STMS. As the heat map is shown in Fig. 8, GAPS is more adaptable to a heterogeneous network. Because it has no more than half of the OQS than STMS in the lossy environment (PLR $>$ 0.01%). When we sysctl different CCs, GAPS is also CCs agnostic as its OQS stays a stable normalized ratio compared with STMS.

Because the OQS latency affects the packet delivery, we also test the application delay of different schedulers. The result that established 5 subflows is shown in Fig 9. It demonstrates that GAPS has a smaller application delay in different network settings. The reduction of its application delay reaches up to 9.4% as compared to STMS. When the network environment is poor, GAPS reduces the application delay by up to 16.3% as compared to DAPS.

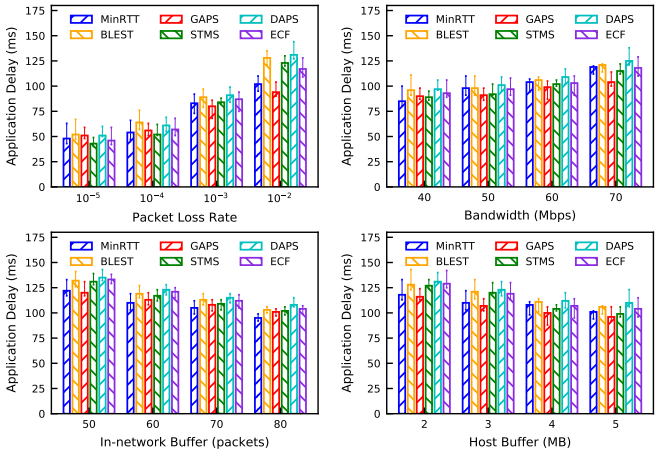

Fig. 9. The application delay under various network environments.

We then investigate how our scheduler improves the aggregated goodput under different buffer sizes. Fig.10 shows the result. When the in-network buffer is limited, our scheduler can improve the average goodput by about 12.7% as compared to STMS. When the host buffer is extremely limited, GAPS still has no blocking packets, and gets about 5.8% goodput improvement than STMS. When the PLR and bandwidth are changeable, GAPS still achieves significant goodput improvements.

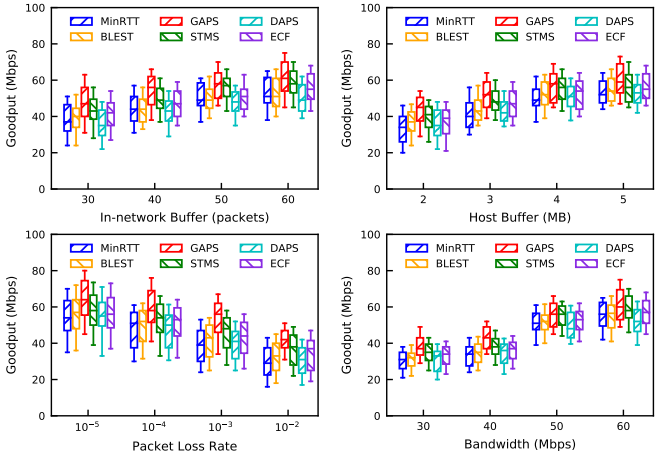

Fig. 10. The goodput under various network environments.

B. Evaluation in the wild

We implement a web service at the server endhost, which provides files download service with different sizes. The client endhost has installed a browser, which is located inside four different locations of our campus: Office, Library, Dorm, and canteen. Fig 11 shows the improvement of aggregated goodput and application delay in the wild, which maintains at least three subflows. GAPS allows an increase of 11.4% in application goodput with bulk traffic as compared to STMS while reducing application delay by 6.3%.

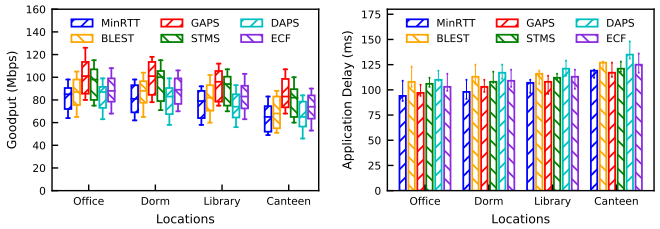

Fig. 11. The performance improvement in different locations.

We measure the FCT of downloading files with different sizes 32MB and 512MB as the short and long MPTCP flow, respectively. The result is shown in Fig. 12. As compared to STMS, GAPS reduces the flow completion time by up to 13%, 21% for 32MB, and 512MB files download.

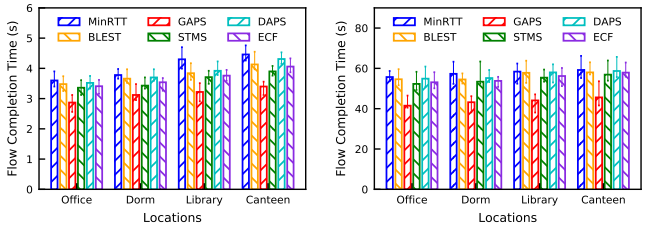

Fig. 12. The flow completion time in different locations.

VII. Related work

Several advanced schedulers have been proposed to optimize the performance of MPTCP in different application scenarios. ReMP [42] sends duplicated data to improve reliability in the cost of high redundancy. eMPTCP [43] considers energy consumption to make scheduling decisions. MultiMob [44] extends the MPTCP handshake to enable immediate retransmissions. Lim et al. [45] propose a cross-layer optimization model for multipath scheduling. DEMS [5] develops theoretical analyses that making all subflows complete at the same time is a necessary condition for achieving the optimal performance. Different from the existing solutions, GAPS is an experience-driven scheduler, which is self-adaptive to various networks. To the best of our knowledge, it is the first time that DDPG and Transformer are integrated into the design and implementation of a realtime traffic scheduler for MPTCP.

VIII. Conclusion

In this work, we derive the GAP-based MPTCP schedulers under heterogeneous path conditions. To inherit a precise and adaptable scheduler, we design a learning-based system, which feeds variable states of TCP subflows into a deep reinforcement learning agent and adjusts the appended GAP-value of each subflow accurately. We implement GAPS in the Linux kernel and evaluate it over both controlled and real network conditions. Our experimental results show that GAPS outperforms state-of-the-art schedulers in diverse congestion controllers and network settings, especially when the path is lossy and the buffer is limit.

Acknowledgments

This work was supported by the National Key R&D program of China (Grant NO. 2019YFB1802800), the National Science Fund for Distinguished Young Scholars (Grant NO. 61725206), and the Key Projects of National Key R&D program of China (Grant NO. 2018YFF0214706).

References

[1] A. Ford, C. Raiciu, M. Handley, O. Bonaventure, *et al.*, "Tcp extensions for multipath operation with multiple addresses, draft-ietf-mptcp-multiaddressed-09," *Internetdraft, IETF (March 2012)*, 2012.

[2] C. Paasch, S. Barré, *et al.*, "Multipath tcp v0.95 in the linux kernel," *Available from {https://github.com/multipath-tcp/mptcp/tree/mptcp_v0.95/net/mptcp.}*, 2013.

[3] C. Paasch, S. Ferlin, O. Alay, and O. Bonaventure, "Experimental evaluation of multipath tcp schedulers," in *Proceedings of the 2014 ACM SIGCOMM workshop on Capacity sharing workshop*, pp. 27–32, ACM, 2014.

[4] Y.-s. Lim, E. M. Nahum, D. Towsley, and R. J. Gibbens, "Ecf: An mptcp path scheduler to manage heterogeneous paths," in *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, pp. 147–159, ACM, 2017.

[5] Y. E. Guo, A. Nikravesh, Z. M. Mao, F. Qian, and S. Sen, "Accelerating multipath transport through balanced subflow completion," in *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, pp. 141–153, ACM, 2017.

[6] H. Shi, Y. Cui, X. Wang, Y. Hu, M. Dai, F. Wang, and K. Zheng, "{STMS}: Improving {MPTCP} throughput under heterogeneous networks," pp. 719–730, ACM, 2018.

[7] M. Scharf and S. Kiesel, "Nxg03-5: Head-of-line blocking in tcp and sctp: Analysis and measurements," in *IEEE Globecom 2006*, pp. 1–5, IEEE, 2006.

[8] N. Kuhn, E. Lochin, A. Mifdaoui, G. Sarwar, O. Mehani, and R. Boreli, "Daps: Intelligent delay-aware packet scheduling for multipath transport," in *2014 IEEE International Conference on Communications (ICC)*, pp. 1222–1227, IEEE, 2014.

[9] S. Ferlin, Ö. Alay, O. Mehani, and R. Boreli, "Blest: Blocking estimation-based mptcp scheduler for heterogeneous networks," in *2016 IFIP Networking Conference (IFIP Networking) and Workshops*, pp. 431–439, IEEE, 2016.

[10] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[12] S. Deng, R. Netravali, A. Sivaraman, and H. Balakrishnan, "Wifi, lte, or both?: Measuring multi-homed wireless internet performance," in *Proceedings of the 2014 Conference on Internet Measurement Conference*, pp. 181–194, ACM, 2014.

[13] H. Jiang, Y. Wang, K. Lee, and I. Rhee, "Tackling bufferbloat in 3g/4g networks," in *Proceedings of the 2012 Internet Measurement Conference*, pp. 329–342, ACM, 2012.

[14] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley, "How hard can it be? designing and implementing a deployable multipath {TCP}," in *9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*, pp. 399–412, 2012.

[15] C. Paasch, R. Khalili, and O. Bonaventure, "On the benefits of applying experimental design to improve multipath tcp," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pp. 393–398, ACM, 2013.

[16] J. Sommers and P. Barford, "Cell vs. wifi: on the performance of metro area mobile connections," in *Proceedings of the 2012 Internet Measurement Conference*, pp. 301–314, ACM, 2012.

[17] Q. Peng, A. Walid, J. Hwang, and S. H. Low, "Multipath tcp: Analysis, design, and implementation," *IEEE/ACM Transactions on networking*, vol. 24, no. 1, pp. 596–609, 2014.

[18] L. Li, K. Xu, T. Li, K. Zheng, C. Peng, D. Wang, X. Wang, M. Shen, and R. Mijumbi, "A measurement study on multi-path tcp with multiple cellular carriers on high speed rails," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pp. 161–175, ACM, 2018.

[19] D. Ni, K. Xue, P. Hong, and S. Shen, "Fine-grained forward prediction based dynamic packet scheduling mechanism for multipath tcp in lossy networks," in *2014 23rd International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–7, IEEE, 2014.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[21] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[22] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[23] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," in *International Conference on Machine Learning*, pp. 2829–2838, 2016.

[24] L. Chen, J. Lingys, K. Chen, and F. Liu, "Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pp. 191–205, ACM, 2018.

[25] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 197–210, ACM, 2017.

[26] E. Liang, H. Zhu, X. Jin, and I. Stoica, "Neural packet classification," in *Proceedings of the ACM Special Interest Group on Data Communication, SIGCOMM 2019, Beijing, China, August 19-23, 2019*, pp. 256–269, 2019.

[27] I. Sutskever, O. Vinyals, and Q. Le, "Sequence to sequence learning with neural networks," *Advances in NIPS*, 2014.

[28] 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.

[29] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[30] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[31] O. Press and L. Wolf, "Using the output embedding to improve language models,"

[32] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning,"

[33] L. J. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *CoRR*, vol. abs/1607.06450, 2016.

[34] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[35] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber, *et al.*, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.

[36] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the brownian motion," *Physical review*, vol. 36, no. 5, p. 823, 1930.

[37] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

[38] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[39] "Tflearn. [online].," *Available from {http://tflearn.org/.}*, 2013.

[40] "Openwrt. [online].," *Available from {https://openwrt.org/.}*, 2017.

[41] T. Graf, G. Maxwell, and R. Mook, "Linux advanced routing & traffic control," *Online document http://lartc. org*, 2004.

[42] A. Frommgen, T. Erbshäußer, A. Buchmann, T. Zimmermann, and K. Wehrle, "Remp tcp: Low latency multipath tcp," in *2016 IEEE International Conference on Communications (ICC)*, pp. 1–7, IEEE, 2016.

[43] Y.-s. Lim, Y.-C. Chen, E. M. Nahum, D. Towsley, R. J. Gibbens, and E. Cecchet, "Design, implementation, and evaluation of energy-aware multi-path tcp," in *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, p. 30, ACM, 2015.

[44] Q. De Coninck and O. Bonaventure, "Tuning multipath tcp for interactive applications on smartphones," in *2018 IFIP Networking Conference (IFIP Networking) and Workshops*, pp. 1–9, IEEE, 2018.

[45] Y.-s. Lim, Y.-C. Chen, E. M. Nahum, D. Towsley, and K.-W. Lee, "Cross-layer path management in multi-path transport protocol for mobile devices," in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pp. 1815–1823, IEEE, 2014.