RCP: A Reinforcement Learning-Based Retransmission Control Protocol for Delivery and Latency Sensitive Applications

Yu Wang and Alhussein A. Abouzeid
ECSE Department
Rensselaer Polytechnic Institute
Troy, NY, USA
wangy52@rpi.edu, abouzeid@ecse.rpi.edu

***Abstract*—This paper presents the design and performance evaluation of a new machine learning-based transport protocol called Reinforcement learning-based retransmission Control Protocol (RCP). Unlike prior transport protocols that aim at either providing guaranteed end-to-end delivery, e.g., TCP, or minimizing the end-to-end delay, e.g., UDP, RCP aims to optimize any achievable combination of these objectives, as specified by an application layer utility function. The utility function in this paper can be quite general, and can capture a combination of delay and packet delivery metrics. RCP can be thought of as an intelligent middle-ground between UDP and TCP, that maps application layer objectives subject to what is learned about the network state. It is window-based like TCP, but retransmissions are decided based on a reinforcement learning algorithm to maximize the application layer utility function. RCP employs a reinforcement learning method, double Q-Learning network, to learn the best strategy in real-time from a history of packet transmission/re-transmission experiences. No assumption on the shape of the utility function is needed. RCP is evaluated under a wide range of network settings, and is found to outperform UDP, TCP, and ARQ for almost all settings. The performance is also evaluated with respect to TCP-friendliness and network stability.**

***Index Terms*—Reinforcement Learning, UDP, ARQ, TCP, Delivery Guarantee, Latency**

I. Introduction

Most Internet traffic is transmitted using TCP [1], [2]. Nevertheless, the increasing popularity of streaming services [3], P2P applications [4], and protocols for real-time transmission (e.g., webRTC) has increased the usage of UDP. If an application requires delivery guarantee, such as file transmission (FTP), TCP (or UDP with ARQ implementation at a higher layer) is preferred. For delay-sensitive applications that require a minimal delay, such as streaming services, UDP is preferred. However, there are also applications requiring some level of delivery guarantee while still targeting the lowest possible latency, e.g., [5], [6]. One typical example is the periodic avatar position updating in multiplayer online games. A player can tolerate sudden position updating of the controlled avatar due to packet loss and slow reaction due to transmission latency. Choosing TCP to get rid of only packet loss would worsen the latency. On the other hand, choosing UDP to decrease the avatar reaction time would cause more frequent avatar teleportation, which is also an undesirable effect. Thus, such applications would require a combination of these metrics to be achieved, and such a transport service is not supported by either TCP or UDP.

Providing a desired end-to-end service is particularly challenging when taking into account the network dynamics. When the end-to-end network state is less congested (under-load), both TCP and UDP achieve high packet delivery probability and low latency. However, as the network becomes more loaded, and thus varying, or even congested, the differences, in terms of achieved packet delivery ratio and delay, between TCP and UDP are more pronounced.

In this paper, we present the design of a new transport protocol called RCP, a novel Reinforcement learning-based retransmission Control Protocol, that considers both the desired quality of service of the application and the dynamic changes of network state. RCP has no specific assumption on the shape of the utility used by the application and is designed to adapt its (re)transmission strategy to maximize the application-layer utility function. Since the target applications are considered sensitive to both delivery and latency, the utility function takes in two inputs, the packet loss probability and the average transmission latency. RCP monitors the network state, such as *round-trip-time* (RTT) and the current packet loss probability, from acknowledged (ACKed) packets. The network state information is also used to adjust the transmission strategy.

Once a packet is lost/timed-out, RCP solves a sequential decision-making problem to decide whether to retransmit or permanently ignore the packet loss. Retransmitting a packet may increase the packet delivery probability but might harm (increase) the average latency, while ignoring a packet achieves the opposite effect. Since the decision may affect the future state of the packet, i.e., being retransmitted again, the problem is shown to be a Markov Decision Process (MDP) problem. We propose to use a reinforcement learning method, and in particular, *Double Q-learning Network* (DQN), to formulate this sequential decision problem, and thus design a utility-

This work is supported by grant R01EB005807 from the National Institute of Biomedical Imaging and Bioengineering of the National Institutes of Health.

maximizing protocol.

The contributions of this paper are summarized as follows:

- We propose RCP, a novel reinforcement learning-based retransmission control protocol that learns the best (re)transmission policy based on the provided utility function and the learned network state.
- RCP has no restriction on the shape of the utility function and learns the strategy that maximizes the achieved utility.
- RCP performs better than UDP, TCP, or ARQ for almost all network settings. It achieves significant utility improvement for applications that care about a combination of delivery and delay metrics.

The remainder of this paper is organized as follows. We briefly present related work in Sec. II. In Sec. III, we formulate the retransmission decision problem as an MDP. Our proposed protocol is then described in detail in Sec. IV. Numerical simulations comparing RCP with UDP, ARQ, and TCP are presented in Sec. V. Finally, we discuss avenues for future work in Sec. VI.

II. Related Work

Several prior works have attempted to address the problem of TCP latency. Examples include the earlier versions of TCP, e.g., TCP Tahoe [7], as well as more recent TCP innovations, e.g., TCP-Cubic [8] and Compound TCP [9]. Optimizations of the congestion control algorithm have been made to achieve high throughput and quick steady state convergence. Other than that, there are also studies on optimizing TCP for real-time transmission. Zhang *et al.* [10] proposed a new congestion control algorithm where the congestion window is adjusted by both the transmitter and the receiver based on the data receiving rate. Therefore, the transmission rate is better controlled based on information known by both the transmitter and the receiver. Liang *et al.* introduced TCP-RTM [11] in which a new framing mode is designed for multimedia applications. They propose an out-of-order packet handling method to skip over discarded packets. Their work was later extended to a design for the last-hop wireless link scenario [12]. Pekez *et al.* [13] mathematically evaluated the possibility of using TCP for short-distance audio communication. The paper concludes that TCP requires more memory and higher processing speed to achieve a better performance than UDP.

TCP's window based congestion control protocol variants rely on acknowledgments (or lack thereof). The retransmission strategy thus belongs to the Automatic repeat request (ARQ) family of protocols. Almost all ARQ protocols are variations of the three well-known types, Stop-and-wait ARQ, Go-Back-N ARQ, and Selective Repeat ARQ. All three types of ARQ maintain a sliding window but with different updating strategies. Among TCP variants, TCP NewReno uses Go-Back-N ARQ that acknowledges the last consecutive received packet. In contrast, some other variations, such as TCP SACK, implement Selective Repeat ARQ that has a strategy to identify which packet is negatively acknowledged. Combining UDP with ARQ is also a commonly used method. A well known example is the TFTP protocol [14] which implements a data file transmission protocol using UDP with a receiver acknowledgment mechanism. Even though a few works propose TCP variations that tolerate some packet loss [11], TCP is still considered to provide almost 100% delivery guarantee.

Recently, as machine learning methods have shown significant advantages in solving complex optimization problems, recent work has also considered combining machine learning with network protocol design. In [15], machine learning tools are used to select a good TCP variant and configure a good initial congestion window size (IW) instead of using a fixed default value. Features about a client, such as location, internet service provider, device type, subnet, etc., are collected from the TCP handshake phase. Then, the selection of the optimal IW is modeled as a data mining problem based on user history data. The protocol selection problem is modeled as a time-variant multi-arm bandit problem and is solved by a reinforcement learning algorithm. Li *et al.* proposed *QTCP* in [16], a reinforcement learning-based congestion control algorithm design. The congestion window *cwnd* is adjusted by a Q-learning network other than hard-coded rules. *TCP-Drinc* presented in [17] is a deep reinforcement learning-based congestion control algorithm that aims to guarantee fairness among multiple transmitters by making a balance between throughput and delay. However, these TCP protocols still offer the same kind of service to the application, i.e., 100% reliability, and do not offer a way to trade-off delivery ratio for better network delay, which is achieved in our proposed RCP protocol.

Utility-based protocol design is also an intensively studied topic, especially in wireless sensor networks and cellular networks. Four different mobility models are studied in [18]. Theoretical bounds on throughput for each of the models under the delay constraint and their feasible coding-scheduling algorithms are also provided. Soret *et al.* characterized the achievable rate under the latency and reliability constraint in a cellular network [19]. The trade-off between energy consumption and throughput is studied in [20] for ultra-reliable low latency communication. The optimal number of retransmission attempts is derived under the energy consumption constraint. In task scheduling algorithms designed for computer processors, Time/Utility Function (TUF) is a commonly used utility function to capture the trade-off between delay and task completion in soft or hard real-time applications [21], [22].

In this paper, the packet retransmission problem is modeled as an MDP problem, and we propose to solve the problem with a model-free reinforcement learning algorithm. One advantage of using a learning-based algorithm is that no assumption on the shape of the utility function is needed. Other than that, a model-free algorithm has a better generalization ability to work with various scenarios.

III. Network setting and Problem Statement

In this section, we describe the network setting for the studied real-time application and formulate the packet retransmission problem as an MDP. We then discuss the limitations of using UDP and TCP for these applications and motivate

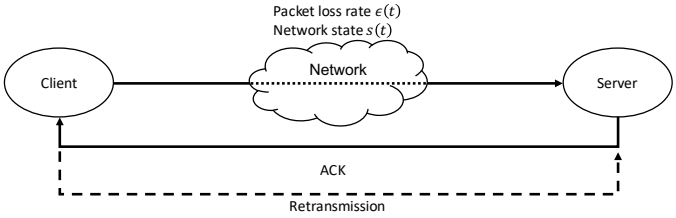

Fig. 1: Network model in RCP. The client sends packets to the server through the network, and the server responds with acknowledgment packets for packet receival. For a packet not being acknowledged for a time exceeding some time out value, RTO, the client decides whether to retransmit the lost packet or to ignore that packet loss.

a protocol that can adapt its strategy to the system utility function and network state.

A. Network Setting

Consider a real-time application that adopts the client-server architecture. The client periodically sends fix-sized packets to the server through the Internet. These real-time packets are sensitive to both packet loss and transmission latency, and the sensitivity is characterized by a utility function of both the packet loss probability and average transmission delay. The network setting is illustrated in Fig. 1. In the transport layer, UDP and TCP are the two commonly used protocols for data transmission. If the transmission goes through UDP, data packets are sent without tracking the success or failure of the packet delivery (i.e., no acknowledgments are sent by the receiver). The packet loss probability reflects the end-to-end network congestion level. The transmission latency is the one-way delay/trip time. The achieved utility may be low if the packet loss is high. If the transmission chooses TCP, and we further assume a window-based TCP, the server responds to each received packet with an ACK packet. The client tracks the number of transmitted (and un-acknowledged) packets and keeps retransmitting a lost packet until its delivery is acknowledged by the server. TCP guarantees delivery, but the retransmission may cause a large delay, especially when multiple retransmissions are attempted and the RTT is large. Thus, TCP may also suffer low utility due to high average delay, even though it has a high delivery ratio.

The motivation of RCP is that, by treating UDP and TCP as two extreme protocols that optimize only one performance metric, a protocol that strikes a balance between delivery and latency based on the preference of the application, expressed as a utility function, may achieve higher utility. RCP improves on standard ARQ protocols in two ways. First, it decides whether or not to retransmit packets. Whenever a packet is lost, RCP decides whether to retransmit the packet or not by evaluating the potential effect of the decision on the utility. Second, it is adaptive, and tries to learn how the decisions affect the end-to-end network performance.

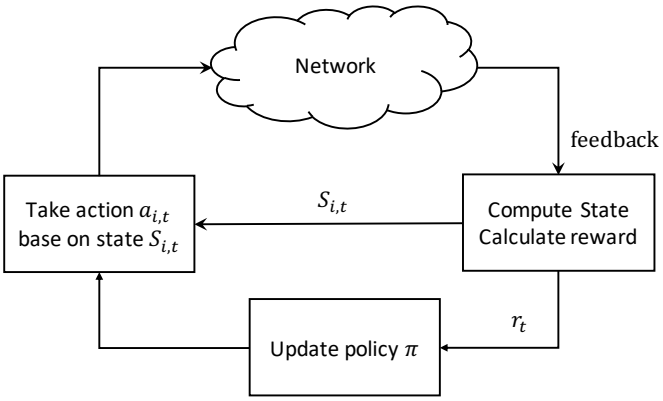

Fig. 2: Reinforcement learning framework in RCP. The state for packet i at t is $S_{i,t}$, and RCP follows policy π to take action $a_{i,t}$. When the state of a packet changes, an instantaneous reward r_t is calculated, which is then used to update the policy.

B. Utility Function

For real-time applications, the utility function $U(\epsilon, l)$ takes into account both: a) the average packet loss probability (including any possible retransmission) ϵ and the average delivery latency l (averaged over the delivered packets). Notice that the latency of a packet is the time between its generation by the transmitter, and its reception at the server. This includes any retransmission delays. Furthermore, if a packet is lost, its delay is infinity, so it is not counted in the computation of the delay – rather, it is reflected in the first parameter, the packet loss probability.

In general, $U(\epsilon, l)$ is regarded as a continuous monotonically decreasing function of ϵ and l, but we have no need for any assumption on concavity or curvature of the function.

We consider the following utility functions with tunable weights $\beta \in [0, 1]$ for simulation and plotting in Sec. V:

$$U(\epsilon, l) = -\beta \cdot \big(n_\epsilon(\epsilon)\big)^\alpha - (1-\beta) \cdot \big(n_l(l)\big)^\alpha. \qquad (1)$$

β is the weight adopted to change the emphasis on delivery or latency. Normalization functions $n_\epsilon()$ and $n_l()$ are employed to scale both ϵ and q to a comparable range, i.e. $[0, 1]$, and $\alpha \in [0, \infty)$ is an exponent used to adjust the curvature of the utility function. We will discuss the detailed implementation of both normalization functions and the effect of α in Sec. V.

C. Reinforcement Learning Problem Formulation

In the reinforcement learning problem formulation, there are two entities, an agent and the environment. The agent interacts with the environment by taking actions a based on the current state s with policy $\pi(s)$. After taking an action, the environment changes to a new state and responds with an instantaneous reward r. In our problem, the agent is the RCP protocol running at the client side. The system includes the network and the RCP server. Action a is whether to retransmit a timeout packet or not. RCP is supposed to learn a good policy $\pi(S)$ such that the expected reward is maximized.

Let $a_{i,t}$ denote the action taken by the agent for packet i at time t. $a_{i,t}$ is chosen from action space $\mathscr{A} = \{0, 1\}$, where $a_{i,t} = 0$ indicates not retransmitting the packet and $a_{i,t} = 1$ indicates retransmitting the packet.

The state of packet i at time t, $S_{i,t}$, is a vector in $R^{1\times 5}$ including the following quantities: (1) the number of transmission attempts of packet r_i; (2) queuing time in the transmitter buffer $q_{i,t}$; (3) current network RTT estimation $\hat{\tau}_t$; (4) current packet loss probability of the link estimation $\hat{\epsilon}_c$; (5) the current delivery latency estimation $\hat{l}_t$. Thus,

$$S_{i,t} = [r_i, q_{i,t}, \hat{\tau}_t, \hat{\epsilon}_c, \hat{l}_t]. \tag{2}$$

Since the ground truth ϵ and l are known by the server rather than the client, the client needs to maintain estimations $\hat{\epsilon}_t$ and $\hat{l}_t$ to update the state. The detailed estimation method is discussed in Sec. IV-B.

To make a good decision, reinforcement learning compares the expected reward of an action in $\mathscr{A}$. The expected reward at state s by taking action a is evaluated by value Q-function $Q(S, a)$. Once the function $Q(S, a)$ is fully characterized, the optimal action to take at $S_{i,t}$ is determined by

$$\pi(S_{i,t}) = a^*_{i,t} = \arg\max_a Q(S_{i,t}, a), a \in \mathscr{A}. \tag{3}$$

In our problem, once a packet is considered as lost (timeout), RCP generates the current state of the packet and then uses the Q-function to choose the action that results in the highest reward.

There are two situations where instantaneous reward r_t is calculated: when a packet is confirmed to be delivered and when a packet is decided not to be retransmitted. Since the client is in charge of the reward calculation, while the real performance information is known at the server, the client maintains the estimation of both the packet deliver probability $\hat{\epsilon}$ and the delivery latency $\hat{l}$.

Consider a packet i that was generated at t_0.

- If the packet is received at time t,

$$r_t = U(0, t - t_0). \tag{4}$$

- If the packet is decided not to be retransmitted,

$$r_t = U(\hat{\epsilon}, \hat{l}). \tag{5}$$

When a packet is delivered, using $r_t = U(0, t - t_0)$ as the instantaneous reward is straightforward because it is the system utility if we only consider the packet. When a packet is decided to be ignored, the selection of r_t is less straightforward, which is described henceforth. If a packet is not retransmitted, there will be no influence on the delivery latency by this packet. From the perspective of the client, $\hat{l}$ is still the same as before. A non-retransmission of a packet increases the packet loss probability, so r_t considers the updated $\hat{\epsilon}$. Note that, $U(1, \hat{l})$ is not a good choice for r_t . Since $U(\epsilon, l)$ is a decreasing function of ϵ and l, $U(1, \hat{l})$ reflects the penalty for increasing the packet loss probability by setting ϵ to 1. However, $\hat{l}$ cannot fully reflect the reward of preventing the average delivery latency from being enlarged, especially when the estimation $\hat{l}$ is inaccurate.

To guide the decision-making algorithm to achieve the global optimum rather than a local optimum, an action should be chosen based on the expected reward rather than the instantaneous reward. By applying the Bellman equation with a discount factor $\gamma \in [0, 1]$, the Q-value can be updated from instantaneous reward following

$$Q(S_t, a_{i,t}) = r_{t'} + \gamma Q(S_{t'}, a_{i,t'}), \tag{6}$$

where $S_{t'}$ is the new state after taking $a_{i,t}$ at S_t. Thus, the instantaneous reward acquired at time t' can sequentially propagate to its previous state S_t and update $Q(S, a)$ to the expected reward. For RCP, γ is set to be 0.9 as many other reinforcement learning implementations [16].

Initially, the reinforcement learning algorithm has no knowledge about the environment. Exploring the environment rather than sticking to known solutions helps to have a better understanding of the environment. A commonly used method to force the algorithm exploration is to implement an Epsilon-Greedy algorithm. Each time an action is to be taken, the agent randomly chooses an action from $\mathscr{A}$ with probability ϵ_0 and employs the reinforcement learning algorithm to choose the action with probability $1-\epsilon_0$. As more data is collected and the reinforcement learning algorithm has an accurate estimation of the expected reward, ϵ_0 should keep scaling down. We set initial $\epsilon_0 = 0.3$, and henceforth ϵ_0 is decayed by 0.7 per 100 times of learning.

IV. RCP Protocol

In this section, we describe the Double Q-learning Network that RCP uses to approximate the Q function. Then, we describe how we estimate the packet and network states.

When the application has new packets to transmit, RCP sends out packets immediately, just as UDP. Packets that are sent out are kept for tracking in a transmitter window buffer. A packet stays in the window until being acknowledged or being decided not to be (re)transmitted. When a packet has not been acknowledged for a time exceeding RTO τ_r, the packet is considered to be timeout and the reinforcement learning algorithm decides whether to retransmit it or not.

A. DQN used by RCP

Acquiring $Q(S, a)$ or finding a good approximation of Q is the key step of a Q-learning based reinforcement learning algorithm. For RCP, we choose *Double Q-learning Network* (DQN) to approximate the Q function because of its good and reliable empirical performance. DQN was first introduced in [23], and its stability and performance are improved by [24], [25]. DQN maintains an *evaluation network* and a *target network*, both sharing the same neural network architecture but using independent weights. Both networks are expected to approximate the Q function, but the evaluation network is updated more frequently from history experiences, while the target network is updated by copying the weights from the evaluation network.

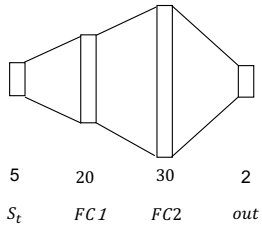

Fig. 3: The neural network architecture used in DQN. "FC" represents a fully-connected layer, and there are two hidden layers. We use the sigmoid function as the activation function for layer FC1.

When DQN is asked to choose an action based on state S_t, the target network computes the Q value for each action in the action space, and the action is chosen following (3). Each time an instantaneous reward is achieved, an experience $(S_t, a_{i,t}, r_{t'}, S_{t'})$ is stored in a memory of DQN. When DQN is asked to learn, a small batch of experiences are randomly chosen from the memory, and the target network is updated by minimizing the distance between its estimation of Q value and the calculated Q value following (6) from experiences. Similar to almost all neural networks, weight update is achieved by back-propagation. In RCP, we use the mean-square distance metric.

Limited by the amount of data RCP can acquire, the neural network in DQN must be shallow. A long training process also hurts the overall performance. In this paper, we choose a two-layer network shown in Fig. 3. After the first fully-connected layer, we use sigmoid function as the activation function.

B. Packet and Link State Estimation

RCP keeps estimating the current packet and link states and makes decisions based on that. The estimations are achieved by using an autoregressive model, which is

$$\hat{x}_+ = (1-\alpha)\hat{x} + \alpha x, \tag{7}$$

where $\hat{x}_+$, $\hat{x}$, and x are the latest estimation result, previous estimation result, and the latest observation, respectively. $\alpha \in [0,1]$ is a weight that controls the percentage of the previous result to keep in the estimator. In general, if we prefer a sensitive estimator, set α to a relatively large weight; if the observation is noisy and we need a stable estimation, set α to a small weight.

RTT τ_t is the round-trip time of the link, and RTO τ_r is the maximum period after which a packet not acknowledged is considered timeout. As suggested in RFC [26], $\hat{\tau}_t$ is estimated by

$$\hat{\tau}_t = 7/8 \cdot \hat{\tau}_t + 1/8 \cdot x, \tag{8}$$

and its variance $V(\hat{\tau}_t)$ is estimated by

$$V(\hat{\tau}_t) = 3/4 \cdot V(\hat{\tau}_t) + 1/4 \cdot |V(\hat{\tau}_t) - \tau_t|. \tag{9}$$

Then, RTO is computed by

$$\tau_r = \hat{\tau}_t + \max(1, 4 \cdot V(\hat{\tau}_t)). \tag{10}$$

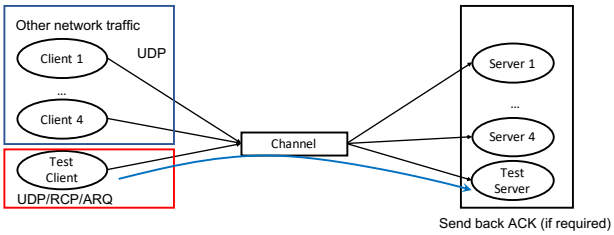

Fig. 4: Client 1,2,3, and 4 are designed to simulate network flows other than the tested client. For each run, the test client chooses one protocol to test, and its test server sends back ACK packets if needed. The performance is evaluated at the test server.

We also use the auto-regression model to estimate packet loss probability $\hat{\epsilon}$, link packet drop probability $\hat{\epsilon}_c$, and current delivery latency $\hat{l}_t$, all using $\alpha = 0.01$ to track a longer period. Both $\hat{\epsilon}$ and $\hat{\epsilon}_c$ take in observation 1 if a packet is delivered. 0 is fed to $\hat{\epsilon}$ when a packet is no longer being retransmitted (decided by the RCP), while 0 is fed to $\hat{\epsilon}_c$ if there is a packet lost. Once an acknowledgment packet is received, the transmission latency of the delivered packet is computed and then used to updated $\hat{l}_t$.

V. Numerical Simulation

In this section, we conduct four sets of numerical simulations to evaluate RCP. We choose UDP and UDP with ARQ (referred to as ARQ) as benchmark protocols. We include TCP in the first experiment but not for the other three, since TCP yields a very poor performance in comparison, as explained henceforth.

A. Experiment Settings

We build a simulation environment and simulate a link (representing an end-to-end path), servers, and clients with Python 3 and PyTorch. Simulation time is discretized into time slots. [1]. The link simulates the propagation delay and adopts a drop-tail-queue model with random packet drop probability to simulate packet loss when the network is congested. The link has a processing rate λ pkts/slot. In a time slot, each client generates new packets and needs to send packets to its server. The server is set to respond with ACK packets if its client requires. The dumbbell network topology used by experiment 1, 2, and 3 is shown in Fig. 4, and the topology for experiment 4 is shown in Fig. 10

TABLE I: Settings for Experiment 1, 2, and 3

Link	Propagation Delay (slots)	100
	Packet Drop Probability	0.1
	Processing Rate λ (pkts/slot)	3
	Queue Size (pkt)	300
Client 1, 2, 3, 4	Protocol	UDP
	New Packet Rate (pkt/slot)	1
Test Client	Protocol	UDP/ARQ/RCP/TCP
	New Packet Rate (pkt/slot)	1

[1] code is available from https://github.com/qizhu8/DRL-RCP-py

	ϵ	l
UDP	0.405	101.070
ARQ	0.034	611.000
TCP NewReno	0.99968	2414.607

TABLE II: Performance of UDP, ARQ, and TCP

TABLE III: Parameters used by RCP

ϵ_0	0.3
ϵ decay	0.7
γ	0.9
Target Net update period	100 rounds
Learning rate	0.01
Batch size	32
Memory size	100000

B. Experiment 1: Effect of β on RCP Performance

Since the performance of UDP, ARQ, and TCP NewReno are not affected by β, we run tests for the three protocols first. Their performances are illustrated in Tab. II. Then, we use the packet loss probability and latency of UDP and ARQ to design the normalization function n_ϵ and n_l, for comparison reasons, i.e.,

$$n_\epsilon(\epsilon) = 1 - \frac{(1-\epsilon) - 0.9(1-\epsilon_{UDP})}{1.1(1-\epsilon_{ARQ}) - 0.9(1-\epsilon_{UDP})}, \quad (11)$$

$$n_l(l) = \frac{l - l_{UDP}}{1.1(l_{ARQ} - l_{UDP})}. \quad (12)$$

The utility function takes $\alpha = 2$. The other parameters used by RCP are listed in Tab. III. By changing β from 0 to 1 with step size 0.1, the performance of RCP under different β is shown in Tab. IV. The achieved utility of UDP, ARQ, and RCP are depicted in Fig. 5.

As shown in Fig. 5, for almost all β values, RCP achieves utility no worse than the better of UDP and ARQ. We plot the average retransmission probability of the last 25% slots (4000 slots) of RCP in Fig. 6. As β increasing from 0 to 1, the retransmission probability increases from almost 0 (UDP) to almost 1 (ARQ). When $\beta = 0.5$, the utility of UDP and ARQ are equally good (or bad), and RCP achieves a significantly better utility by retransmitting 69.9% timeout packets.

In this experiment, we verify that RCP is able to adapt its strategy to maximize the system utility. For scenarios where focusing only on either latency or delivery is not ideal, RCP is able to achieve a dominantly better performance.

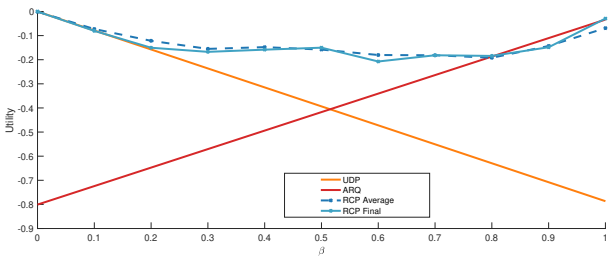

Fig. 5: Utility Comparison of RCP when $\alpha = 2$ with UDP and ARQ for $\beta \in [0, 1]$

TABLE IV: Performance of RCP when $\alpha = 2$ for $\beta = 0, 0.1, \cdots, 1$.

β	Average			Final			Retrans
	ϵ	l	Utility	ϵ	l	Utility	Prob
0	0.401	108.151	-0.001	0.395	101.960	0.000	0.030
0.1	0.379	120.276	-0.073	0.408	104.967	-0.080	0.124
0.2	0.332	152.928	-0.122	0.394	101.895	-0.150	0.304
0.3	0.297	176.941	-0.155	0.322	151.357	-0.167	0.475
0.4	0.229	210.283	-0.148	0.249	191.906	-0.158	0.676
0.5	0.209	220.929	-0.158	0.205	209.509	-0.150	0.699
0.6	0.209	223.975	-0.181	0.237	198.449	-0.207	0.715
0.7	0.191	239.361	-0.182	0.196	209.534	-0.182	0.757
0.8	0.186	246.981	-0.192	0.184	217.140	-0.184	0.766
0.9	0.158	305.936	-0.171	0.188	215.276	-0.209	0.850
1	0.076	483.942	-0.069	0.028	541.687	-0.029	0.969

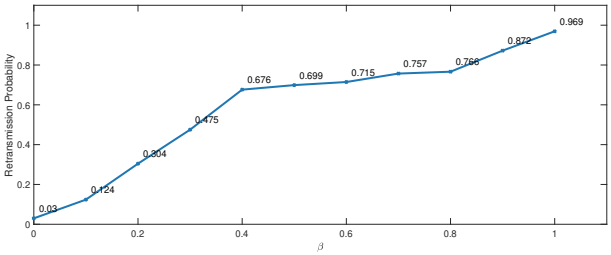

Fig. 6: Packet Retransmission Probability of RCP in the last 10% simulation rounds when $\alpha = 2$

C. Comparison to TCP

The performance of TCP NewReno in Experiment 1 is shown in Table II. TCP shows a long latency and almost one packet loss probability. The reason is an interplay between the "back-off" strategy in the congestion control and the RTO updating algorithm. The initial congestion window size for TCP NewReno is between 2 and 4 based on the SMSS [27], and we choose 4 here. Since the packet loss probability is at least 0.1, and there also exist competitions from Clients 1, 2, 3 and 4, the probability of seeing a packet loss is high. Therefore, TCP transmitter is almost always in back-off mode (slow-start) with the initial congestion window. As TCP NewReno requires the receiver acknowledges the largest consecutive packet sequence number, even with the help of *tri-dup-ack*, the sender is still unable to efficiently know which packet has been delivered. Therefore, an overestimated RTT estimation is made, which further causes a larger RTO estimation. When RTO becomes longer than the simulation period, no more packets will be considered timed-out and hence and no retransmissions take place during the simulation period. Therefore, the TCP has almost zero throughput. Since the experiment setting is quite unfavorable to TCP (because the setting is latency sensitive), we do not include TCP in further experiments.

D. Experiment 2: Effect of the shape of the Utility function on RCP performance

In this section, we use different α values to see how RCP performs on utility functions with different shapes. We change α to 1 and 4 while keeping the other parameters unchanged.

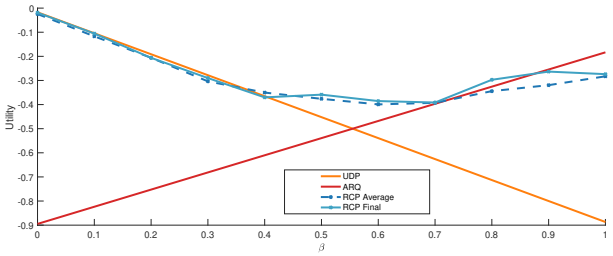

Fig. 7: Comparison of Utility between RCP, UDP and ARQ when $\alpha = 1$ and $\beta \in [0, 1]$

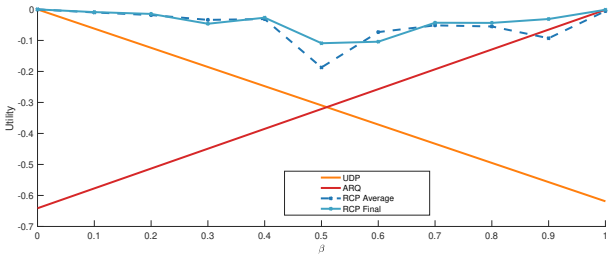

Fig. 8: Comparison of Utility between RCP, UDP and ARQ when $\alpha = 4$ and $\beta \in [0, 1]$

The comparisons of utilities achieved by RCP, UDP, and ARQ are depicted in Fig. 7 and Fig. 8, respectively.

When $\alpha = 1$, the utility function is a linear function of ϵ and l. The utility versus β curve of RCP is closer to the linear utility achieved by UDP or ARQ. Since the (ϵ, l) pair achieved by either UDP or ARQ is not affected by β, the utility with β curve for either UDP or ARQ is a linear function that is also unaffected by α. As α increases, the utility achieved by the same (ϵ, l) pair also increases. Thus, a protocol is better rewarded than performing similar to either UDP or ARQ if it is able to achieve $\epsilon_{ARQ} < \epsilon < \epsilon_{UDP}$ and $l_{UDP} < l < l_{ARQ}$. RCP is then able to achieve high utility improvements for most of the β values.

E. Experiment 3: Effect of the Normalization Function

The experiments in Sec. V-B and Sec. V-D use elaborated tuned normalization functions (11) and (12) that require prior knowledge of UDP and ARQ performance, which is infeasible in real applications. In this section, we conduct another experiment to show that RCP can still perform well with a less carefully designed normalization function.

In utility function $U(\epsilon, q)$, ϵ is a probability that is always in range $[0, 1]$ while latency $q \in (0, \infty)$. To make the two terms involving ϵ and q comparable, scaling down q is system-dependent. We choose the following two new normalization functions

$$n_\epsilon(\epsilon) = \epsilon, \tag{13}$$

$$n_l(l) = \frac{l}{\tau_{r,\max}}. \tag{14}$$

$\tau_{r,\max}$ is the maximum RTO assumed to be acquired from previous transmission data. Here we approximate $\tau_{r,\max}$ by

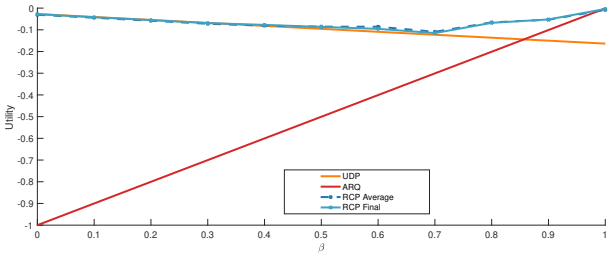

Fig. 9: Utility verses β curves for RCP, UDP and ARQ using new normalization functions and $\alpha = 2$. RCP is still able to overperform UDP and ARQ, but the utility improvement is less significant than the result in Sec. V-B

q_{ARQ}. In practice, $\tau_{r,\max}$ can be replaced by a sufficiently large number based on the network condition. The utility versus β performance for all three protocols is depicted in Fig. 9.

In Fig. 9, RCP is still able to achieve better utility than either UDP or ARQ for almost all β values. However, it also shows that the range of β in which RCP performs significantly better is smaller than the setup in Sec. V-B. We can then conclude that the choice of the normalization function is going to affect the dynamical range of β in which RCP significantly outperforms UDP and ARQ.

F. Experiment 4: Effect of RCP on Network Stability

In this section, we show that the learning algorithm in RCP actually helps to constrain the retransmission attempts such that it is not going to take up all link resources for retransmission. Initially, we configure the link to have a sufficiently large processing rate. When the network becomes stable, we decrease the link processing rate such that the network becomes congested. By checking the behavior of RCP before and after the decrease of the link processing rate, we are able to see whether RCP is able to learn the change in the network and adjust its strategy in response to this onset of congestion.

The network topology is shown in Fig. 10, and the experiment setting is included in Table V. The total simulation time is $25,000$ time slots.

TABLE V: Experiment 4 Settings

Link	Propagation Delay (slots)	100
	Packet Drop Probability	0
	Queue Size (pkts)	1000
	Processing Rate λ when $t <= 5000$ (pkts/slot)	7
	Processing Rate λ when $t > 5000$ (pkts/slot)	3
RCP	β	0.8
	α	2
RCP and ARQ	Initial RTO (slots)	30

We depict the changes of throughput (delivered packets) over time of Client 3 (RCP), 4 (ARQ), and 5 (UDP) in Fig. 11 and provide a closer look at the throughput curves after the changes of λ in Fig. 12. Since the initial RTO of both ARQ and RCP are set to be smaller than the link propagation delay, when $t < 2500$, the throughput of both

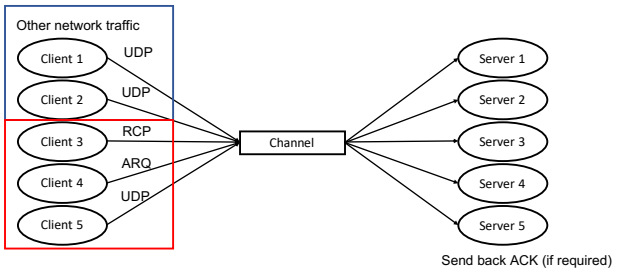

Fig. 10: There are five clients and five corresponding servers indexed from 1 to 5. Each client needs to send one new packet in each slot to its server. Client 1 and Client 2 are used to simulate background traffic in the network, and Clients 3, 4, and 5 choose RCP, ARQ, and UDP respectively. The link processing rate λ is initially set to 7 pkts/slot and then decreases to 3 pkts/slot at $t = 5000$.

TABLE VI: Performance of UDP, ARQ, and RCP in Experiment 4

	ϵ	l	$U(\epsilon, l)$
UDP	0.443	273.110	-0.531
ARQ	0.151	1551.363	-0.267
RCP	0.288	577.592	-0.200

ARQ and RCP fluctuate and then stabilize to 1 pkt/slot. When λ decreases to 3 pkts/slot at $t = 5000$, all three clients suffer a decrease in throughput. According to Fig. 13, we see that the retransmission attempts keep increasing until around $t = 9000$ because RCP needs to first update its experience memory and then learn from the new experiences. The retransmission attempts become stable after $t = 12000$. The final performance results of the three protocols are shown in Table VI. Similar to the result in Sec. V-B, RCP is still able to achieve the best utility.

Therefore, we conclude from our experimental observations that RCP, as expected, is able to learn and adjust its strategy based on the network state and will not lead to network instability in a congested network.

VI. Conclusion and Future Work

In this paper, we focus on the design of an intelligent transport layer protocols for applications that are not sensitive exclusively to only one quality of service, such as delivery

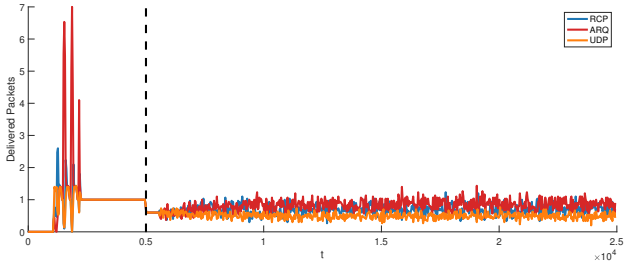

Fig. 11: We present the changes of throughput over time of Client 3, 4, and 5 when λ decreases from 7 to 5 at $t = 5000$.

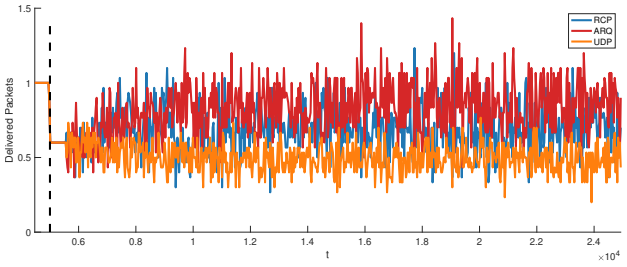

Fig. 12: This plot is a close look at the throughput of Client 3, 4, and 5 after $t = 5000$. Since RCP is configured to have $\beta = 0.8$, RCP acts similar to ARQ, which can be verified from the plot that RCP and ARQ have similar averaged throughput.

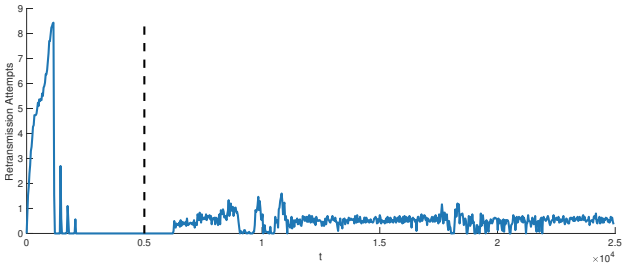

Fig. 13: Retransmission attempts of RCP (Client 5) over time in Experiment 4. When λ decreases from 7 pkts/slot to 3pkts/slot at $t = 5000$, the retransmission attempts increase because of the increasing number of timeout packets due to the link congestion. RCP learns the changes in the meanwhile. The modified strategy starts to work at around $t = 13000$, which is verified by a decrease in retransmission attempts.

ratio or packet delay, but rather they are sensitive to a combination of such metrics. We design RCP, a reinforcement learning-based transport layer protocol, that maps the service requirement, represented as a utility function, to a sequential (re)transmission strategy that maximizes the objective utility function. RCP trades-off packet delay and packet delivery ratio, in order to provide optimal service for the application. For example, by deciding to no retransmit some lost packets, it can improve on the average delay for delivered packets, which may be beneficial to the utility. The decision of whether to give up or retransmit a packet is made by a reinforcement learning network, DQN. Without hard-coding the system utility function to the retransmission policy design, RCP is able to learn from the history and maximize the overall system utility. Extensive simulation results show that RCP is almost always the best protocol compared to UDP, TCP and ARQ. For scenarios where the application layer cares about both delay and delivery, RCP is able to achieve a significant utility improvement by achieving an optimal balance between delivery and latency. Furthermore, the experiments demonstrate that RCP can adjust its strategy in case of sudden onset of network congestion, maintaining network stability.

Several avenues of future work exist. First, the concept here, while promising, needs to be investigated in more detail

before it could be recommended for wide-scale deployment, e.g., through testbed experimentation. Second, compared to TCP, RCP does not have an explicit design of a congestion control function, and the current design has not been assessed for TCP-friendliness in general settings. Finally, due to the implementation of a learning-based algorithm, the convergence speed needs to be further investigated. The memory size of DQN is hard-coded in the numerical simulation section and needs to be generalized. The proper memory size that balances the learning efficiency and accuracy should also be considered in future work.

REFERENCES

[1] M. Fomenkov, K. Keys, D. Moore, and K. Claffy, "Longitudinal study of internet traffic in 1998-2003," in *Proceedings of the Winter International Synposium on Information and Communication Technologies*, ser. WISICT '04. Trinity College Dublin, 2004, pp. 1–6.

[2] W. John and S. Tafvelin, "Analysis of internet backbone traffic and header anomalies observed," in *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '07. New York, NY, USA: Association for Computing Machinery, 2007, pp. 111–116.

[3] P. Pan, Y. Cui, and B. Liu, "A measurement study on video acceleration service," in *2009 6th IEEE Consumer Communications and Networking Conference*. IEEE, 2009, pp. 1–2.

[4] W. John, S. Tafvelin, and T. Olovsson, "Trends and differences in connection-behavior within classes of internet backbone traffic," in *International Conference on Passive and Active Network Measurement*. Springer, 2008, pp. 192–201.

[5] V. Clincy and B. Wilgor, "Subjective evaluation of latency and packet loss in a cloud-based game," in *2013 10th International Conference on Information Technology: New Generations*. IEEE, 2013, pp. 473–476.

[6] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Claypool, "The effects of loss and latency on user performance in unreal tournament 2003®," in *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, 2004, pp. 144–151.

[7] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "Tcp selective acknowledgment options," RFC 2018, Tech. Rep., 1996.

[8] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A compound TCP approach for high-speed and long distance networks," in *Proceedings-IEEE INFOCOM*, 2006.

[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] C. Zhang and V. Tsaoussidis, "TCP-Real: Improving real-time capabilities of TCP over heterogeneous networks," in *Proceedings of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, ser. NOSSDAV '01. New York, NY, USA: Association for Computing Machinery, 2001, pp. 189–198.

[11] S. Liang and D. Cheriton, "TCP-RTM: Using TCP for real time multimedia applications," in *International Conference on Network Protocols*. Citeseer, 2002.

[12] I. Radovanovic, R. Verhoeven, and J. Lukkien, "Improving TCP/IP Performance over Last-hop Wireless Networks for Streaming Video Delivery," in *2007 Digest of Technical Papers International Conference on Consumer Electronics*, 2007, pp. 1–2.

[13] N. Pekez, A. Popović, and J. Kovačević, "Performance analysis on TCP/IP audio streaming in point-to-point communication," in *2019 Zooming Innovation in Consumer Technologies Conference (ZINC)*. IEEE, 2019, pp. 70–75.

[14] K. Sollins, "The TFTP protocol (revision 2)," Internet Requests for Comments, RFC Editor, Tech. Rep., 7 1992. [Online]. Available: https://tools.ietf.org/html/rfc1350

[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] 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, 2019.

[17] K. Xiao, S. Mao, and J. K. Tugnait, "TCP-Drinc: Smart Congestion Control Based on Deep Reinforcement Learning," *IEEE Access*, vol. 7, pp. 11 892–11 904, 2019.

[18] L. Ying, S. Yang, and R. Srikant, "Optimal Delay-Throughput Tradeoffs in Mobile Ad Hoc Networks," *IEEE Transactions on Information Theory*, vol. 54, no. 9, pp. 4119–4143, 2008.

[19] B. Soret, P. Mogensen, K. I. Pedersen, and M. C. Aguayo-Torres, "Fundamental tradeoffs among reliability, latency and throughput in cellular networks," in *2014 IEEE Globecom Workshops (GC Wkshps)*, 2014, pp. 1391–1396.

[20] J. P. Battistella Nadas, O. Onireti, R. D. Souza, H. Alves, G. Brante, and M. A. Imran, "Performance Analysis of Hybrid ARQ for Ultra-Reliable Low Latency Communications," *IEEE Sensors Journal*, vol. 19, no. 9, pp. 3521–3531, 2019.

[21] E. D. Jensen, C. D. Locke, and H. Tokuda, "A time-driven scheduling model for real-time operating systems." in *Rtss*, vol. 85, 1985, pp. 112–122.

[22] P. Li, H. Wu, Binoy Ravindran, and E. D. Jensen, "A utility accrual scheduling algorithm for real-time activities with mutual exclusion resource constraints," *IEEE Transactions on Computers*, vol. 55, no. 4, pp. 454–469, 2006.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[24] J. Foerster, N. Nardelli, G. Farquhar, T. Afouras, P. H. S. Torr, P. Kohli, and S. Whiteson, "Stabilising experience replay for deep multi-agent reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. International Convention Centre, Sydney, Australia: PMLR, 06–11 Aug 2017, pp. 1146–1155.

[25] S. Omidshafiei, J. Pazis, C. Amato, J. P. How, and J. Vian, "Deep decentralized multi-task multi-agent reinforcement learning under partial observability," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. International Convention Centre, Sydney, Australia: PMLR, 06–11 Aug 2017, pp. 2681–2690.

[26] M. Sargent, J. Chu, D. V. Paxson, and M. Allman, "Computing TCP's retransmission timer," Internet Requests for Comments, RFC Editor, RFC 6298, 5 2011. [Online]. Available: https://tools.ietf.org/html/rfc6298

[27] M. Allman, V. Paxson, and E. Blanton, "TCP congestion control," Internet Requests for Comments, RFC Editor, RFC 5681, 9 2009. [Online]. Available: https://tools.ietf.org/html/rfc5681