QTCP: Adaptive Congestion Control with Reinforcement Learning

Wei Li, Fan Zhou, Kaushik Roy Chowdhury, *Senior Member, IEEE*, and Waleed Meleis

Abstract—Next generation network access technologies and Internet applications have increased the challenge of providing satisfactory quality of experience for users with traditional congestion control protocols. Efforts on optimizing the performance of TCP by modifying the core congestion control method depending on specific network architectures or apps do not generalize well under a wide range of network scenarios. This limitation arises from the rule-based design principle, where the performance is linked to a pre-decided mapping between the observed state of the network to the corresponding actions. Therefore, these protocols are unable to adapt their behavior in new environments or learn from experience for better performance. We address this problem by integrating a reinforcement-based Q-learning framework with TCP design in our approach called QTCP. QTCP enables senders to gradually learn the optimal congestion control policy in an on-line manner. QTCP does not need hard-coded rules, and can therefore generalize to a variety of different networking scenarios. Moreover, we develop a generalized Kanerva coding function approximation algorithm, which reduces the computation complexity of value functions and the searchable size of the state space. We show that QTCP outperforms the traditional rule-based TCP by providing 59.5 percent higher throughput while maintaining low transmission latency.

Index Terms—Reinforcement learning, TCP congestion control, function approximation, dynamic generalization, Kanerva coding

1 Introduction

Rapid advancements in wired and wireless technologies has triggered the emergence of new network architectures, such as 60 GHz mmWave WiFi [1], cognitive radio networks [2], [3], [4] and data center networks [5]. At the same time, the proliferation of new applications such as video streaming, cloud storage, on-line gaming, creates higher performance requirements for the data transmission environment and poses new challenges on the design of congestion control protocols.

In contrast to these new and evolving networking scenarios and user applications, the same transport protocol design has been employed over the past three decades, with TCP NewReno being one of the de-facto congestion control standards. Despite effort expended to develop new congestion control protocols (such as Vegas, FAST, see Section 6 for more details), these protocols broadly share a common limitation of not being able to perform well across a wide range of networking scenarios, and hence, are seldom deployed in real world networks. This limitation stems from the fact that these protocols are built on the common concept of relying on pre-configured rules to guide the behavior of end hosts (e.g., how to change the congestion window size) given specific observations of the surrounding environment (e.g., measured throughput, RTT). For example, the NewReno protocol uses the well-known additive increase, multiplicative decrease (AIMD) strategy, and Cubic adopts a well-crafted function to adjust the congestion window size (*cwnd*) given feedback from the receiver. This rule-based design can cause two problems: *First*, it causes congestion control protocols to be unable to adapt to new scenarios when a network environment changes. Since different kinds of networks differ in significant ways with respect to bandwidth, delay and network topology, a given TCP flavor that works well for a specific network might not work in another. *Second*, the rules of operation are usually built upon standard assumptions or the network model. When either changes, the fixed mapping between observation and actions means that TCP does not intelligently adjust its behavior by learning from experience. As a result, the protocol repetitively adopts the same *cwnd* changing rules that bring sub-optimal performance, without the flexibility to adjust behaviors for better performance (Section 2).

Proposed Approach. In this work, we use reinforcement learning (RL) to design a congestion control protocol called QTCP (Q-learning based TCP) that can automatically identify the optimal congestion window (*cwnd*) varying strategy, given the observation of the surrounding networking environment in an on-line manner. It does not need for manually-crafted rule sets or time-consuming off-line training process. RL enables agents to adjust their behavior based on real-time feedback, and avoid repeating the same mistakes by discouraging ineffective behavior. We utilize this capability in QTCP to allow senders to dynamically learn different strategies to better adapt to varying networking scenarios, instead of mechanically following fixed rules. Specifically, QTCP continuously updates the values of possible *state-action pairs* of the protocol, based on the measurement of performance metrics collected from a networking environment, and uses

• *The authors are with the Department of Electrical and Computer Engineering, Northeastern University, Boston, MA 02115. E-mail: {li.wei, zhou.fan1}@husky.neu.edu, {krc, meleis}@ece.neu.edu.*

Manuscript received 15 Oct. 2017; revised 19 Mar. 2018; accepted 3 May 2018. Date of publication 10 May 2018; date of current version 11 Sept. 2019. (Corresponding author: Wei Li.)
Recommended for acceptance by O. B. Akan.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TNSE.2018.2835758

Q-learning algorithm to search for the best action, i.e., how to adjust the *cwnd* in specific states so that the long term *reward* of the sender is maximized.

Challenges and Innovations. While RL has been shown to perform well on many hard problems (e.g., Go, automatic driving), applying it to TCP congestion control is particular challenging due to the problem's continuous, high-dimensional state space. The size of the state space can grow exponentially with the dimension of the state space, causing an significant increase in the size of the table needed to store the state-action values. It is usually very time-consuming to update entries in such a large table, which results in unacceptably long training time. To speed up the learning process and make QTCP tractable, we apply *function approximation* [6], which is an effective way to reduce the size of the state space needed to search and explore using an abstract state representation.

While there are many function approximation algorithms available, we choose Kanerva coding [7], also known as Sparse Distributed Memories (SDMs), because of its low complexity, quick convergence and its effectiveness in solving problems with large, high-dimensional and continuous state spaces. The idea of Kanerva coding considers such a setting that the whole state space is represented by a carefully selected subset of the state space based on which trained values are stored and derived policies are evaluated, thus reducing the memory consumption and computation complexity of value trainings significantly. However, we found that the performance of original Kanerva coding is not satisfactory in practice due to the improper selection of the subset of the state space. To solve this problem, we propose a novel approach, generalization-based Kanerva coding, that can adjust the level of abstraction for each entry of the subset of the state space and thus dynamically reallocate the subset to find its near-optimal structure when exploring the state space. Our approach allows the granularity of the state abstraction to be changeable based on visited states, where less important entries of the subset with improper levels of generalization are examined and replaced with ones that provide better generalization. This overcomes the limitations of the classic Kanerva coding algorithm and its variants, enabling QTCP to have faster convergence and better overall learning performance.

In summary, we make following two contributions:

- We describe QTCP, a Q-learning based congestion control protocol that automatically learns the effective strategies for adjusting the *cwnd* to achieve high throughput and low delay in an on-line manner. This fundamentally changes the design of previous NewReno-like TCP variants that require fixed, manually selected rules.
- We propose a new kind of Kanerva coding algorithm that scales well when applied to large complex state spaces and greatly speeds up convergence and provides stable performance. Our algorithm allows the learned values no longer to be stored in a tabular form and thus eliminates a vital limitation, e.g., unable to handle enormous states, of RL technique when applied to large-scale problem domains.

This paper is organized as followings. In Section 2, we give an overview of congestion control problems and the limitations of rule-based TCP variants (using NewReno as an example). We describe the design of QTCP in Section 3. In Section 4, we introduce the principles of function approximation used in RL, discuss the issues of existing Kanerva coding approaches, and then propose our generalization-based Kanerva coding algorithm. We present experimental results in Section 5. We show related work in Section 6 and finally conclude our work in Section 7.

2 Background and Motivation

In this section, we quantitatively describe the problem with classic additive increase multiplicative decrease rule used by TCP NewReno. Then we discuss the limitations of rule-based TCP and motivate the need for a more adaptive strategy to control congestion in a network.

2.1 Congestion Control: Problems and a Classical Solution

The goal of congestion control is to allow senders to share limited bandwidth fairly, without overwhelming the network. The congestion control algorithm does this by maintaining a *cwnd* that limits the maximum number of packets each sender can safely inject into the network without causing traffic congestion. In general, using larger *cwnd* allows more packets to be sent into the network that can potentially give higher throughput. However, if every sender tries to greedily maximize its own throughput and keeps increasing *cwnd*, increased congestion can severely degrade every sender's performance.

While many other factors (such as packet size, sender and receiver buffer size, etc) can influence the value of the *cwnd*, the key problem in congestion control is to enable each sender to independently tune the *cwnd* that maximizes its own throughput while co-existing fairly with other competing flows in the network. For example, the sender should increase the *cwnd* to increase the link utilization and reduce the *cwnd* if the packet loss rate or the queuing delay increases when the network is congested. The challenge is that TCP is an end-to-end protocol as it works only on two end hosts. The sender does not has ground truth information about path characteristics, such as bottleneck link bandwidth or the current number of flows that are sharing the same link. Nor can different senders coordinate or share information with one another. The end-to-end paradigm significantly increases the complexity of congestion control protocol design. While there are some TCP variants that break this rule by assuming the existence of a centralized controller or the availability of a global view of the entire network, such designs are limited in generality and can only be applied to small-scale scenarios such as enterprise or data center networks.

A classic congestion control protocol is NewReno [8], which is also one of the most widely used TCP today. It uses the AIMD rule to control the *cwnd*. The basic idea of AIMD is to first slowly increase the *cwnd* until the bottleneck link buffer has been saturated and the packets are dropped by the router (additively increase). Information about packet drops is conveyed back to the sender with duplicate ACKs indicating network congestion. After that, sender reduces its sending rate by halving its *cwnd* (multiplicative decrease). In summary, the behavior of NewReno can be broadly modeled as using the following three stages:

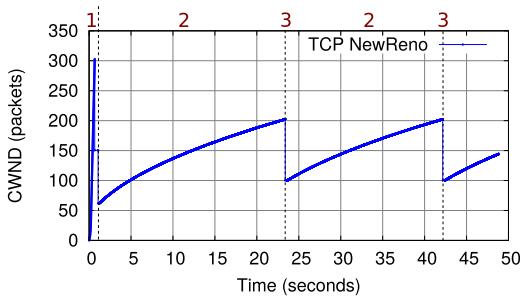


Fig. 1. TCP cwnd graph for a NewReno flow: (1) slow start (2) congestion avoidance and (3) fast recovery after duplicate ACK.

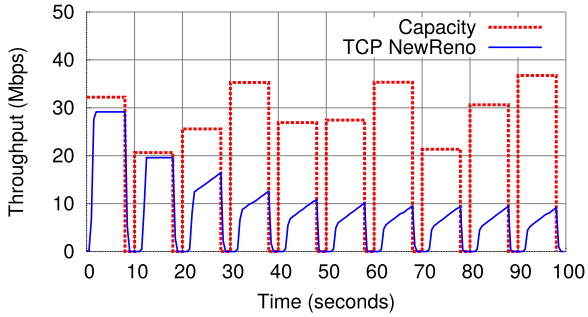
- *Slow start: cwnd = cwnd* + 1 for every ACK received
- *Congestion avoidance: cwnd = cwnd* + 1/*cwnd* for every ACK received
- *Fast recovery (duplicate ACK): cwnd = cwnd*/2

The phases of NewReno are shown in Fig. 1. During slow start (stage 1), the *cwnd* ramps up quickly, approximately doubling every RTT until it reaches the slow start threshold. This allows the new flows to quickly acquire bandwidth. During congestion avoidance (stage 2), the increasing speed of *cwnd* slows down (approximately by 1 MSS every RTT). This is to enable the sender to cautiously detect the congestion point of the network. Once the sender receives packet loss information, it halves *cwnd* when it receives duplicate ACKs (stage 3) or reduces *cwnd* to 1 when severe congestion is observed and the sender fails to receive any feedback from the receiver. It has been shown that AIMD can guarantee convergence to a policy that optimally shares bottleneck bandwidth with respect to both efficiency and fairness [9].

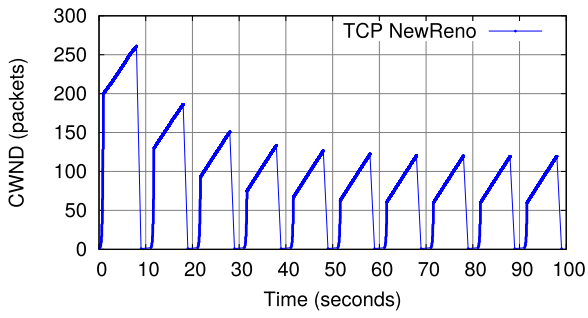
2.2 Limitation of Rule-Based TCP Protocols

In addition to NewReno, many other congestion control protocols have been described to further optimize TCP's performance for specific networks or applications. However, we point out that there are two key limitations faced by these rule-based congestion control protocols:

- *Inability to adapt to new or unseen network scenarios:* Most existing protocols do not adapt their congestion-control strategies as the network evolves over time. For example, NewReno will always increase *cwnd* by one over one RTT during congestion avoidance stage, even though it might be too slow for the flow to fully utilize the bandwidth in some modern Internet routes where the link bandwidth is high and the round trip time (RTT) is long.
- *Inability to learn from historical knowledge:* Every time a flow starts, the classical approach assumes no prior information of the links. However, much better performance can be achieved if TCP can adjust its behavior based on previously learned information when the same path was previously explored. For example, if NewReno already has knowledge learned from previous interactions with path characteristics such as bandwidth, delay, packet loss rate, it may be able to adjust its behavior with more flexibility: it may speed up the *cwnd* more aggressively during congestion avoidance to increase the link utilization on less congested links.



(a) Link capacity and flow throughput.



(b) Varying cwnd of NewReno.

Fig. 2. The real time throughput and *cwnd* variation of a TCP NewReno flow. Bottleneck bandwidth is uniformly distributed between [20,40] Mbps. RTT = 100ms, Buffer size = 150 packets.

We use the following *ns-3* simulation with dynamic spectrum access links in cognitive radio (CR) network to study a typical bandwidth-varying scenario that highlights the above problems. CR allows opportunistic use of under-utilized licensed spectrum allocated to primary users, where the CR user has to sense interferences and periodically switch to a new spectrum to avoid interfering with the primary user. This spectrum sensing and switching process may lead to short transmission stall and sudden changing in the available bandwidth, making effective congestion control more challenging [10], [11].

We can show the performance of NewReno with varying network conditions (bandwidth, frequent interruptions caused by PU activity/sensing). We start one NewReno flow from one source to the CR mobile device. We model the consecutive transmission *On* period and the disconnection caused by spectrum sensing/switching as the *Off* period. We fix the "On" period to be 9s and the "Off" period to be 1s. The varying range of channel capacity is chosen between 20 Mbps and 40 Mbps. Thus, the bandwidth availability changes by uniformly picking a value between 20 Mbps and 40 Mbps every 10 seconds.

As shown in Fig. 2a, the performance of NewReno is far from satisfactory in this network scenario—it only achieves about 30 percent of link bandwidth. Fig. 2b reveals two key reasons behind the low link utilization: (1) the *cwnd* will drop to one due to the timeout event triggered every time the CR user is performing spectrum sensing/switching, and (2) the slow increase of *cwnd* during the convergence avoidance does not allow the sender to acquire sufficient network resources between two transmission interruptions. Even worse, this pattern is repeated in each cognitive cycle, indicating that a rule-based protocol is unable to learn from previous experience and adapt its behavior to achieve

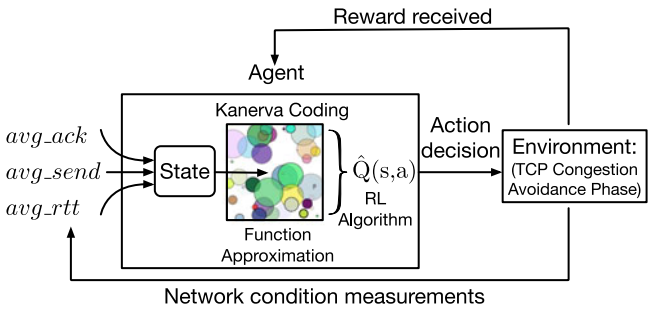


Fig. 3. Solution framework of our RL-based TCP congestion control design.

consistent good performance in evolving practical network nowadays. While a larger initial *cwnd* might improve performance, we will show that our adaptive algorithm can achieve improved performance even when starting with a suboptimal initial *cwnd*.

3 QTCP: Apply Q-Learning to TCP Congestion Control

In this section, we explore the use of RL to automatically design congestion control strategies. RL has the potential of overcoming the problems of rule-based TCP described above as it can enable agent to learn from past experience, without the need for manually settled rules or prior knowledge of the networking scenarios. Specifically, we discuss how to apply Q-learning, a classical RL algorithm to the domain of congestion control problem and propose QTCP: a new congestion control protocol that allows sender to learn the optimal *cwnd* changing policy through interaction with the network scenarios.

3.1 Overview of QTCP

The framework of QTCP is shown in Fig. 3. The learning agent (sender) interacts with the network environments and keeps exploring the optimal policy by taking sequential actions (e.g., varying the *cwnd*) given feedback as it works to achieve its desired goal, i.e., large throughput and low latency. Like any typical RL problem, QTCP consists of the following elements:

- *States:* defined as informative perceptions or measurements that an agent can obtain from the outside environment. Here, the state is a unique profile of the network conditions evaluated through selected performance metrics (Section 3.2).
- *Actions:* chosen by an agent at each time step, after perceiving its current state, according to a *policy*. In the context of congestion control, the action is the decision to increase, decrease, or leave unchanged the current *cwnd* (Section 3.3).
- *Reward:* this reflects the desirability of the action picked. As we describe below, the reward is further specified by the value of a *utility function*, which is computed based on the measurement of flow throughput and latency. Higher throughput and lower latency translates into a higher utility value and vice-versa (Section 3.4).
- *Training algorithm:* The purpose of the training algorithm is to learn the optimal policy to select certain action for each state. This is the central module of QTCP as it is responsible for developing the congestion control strategies (Section 3.5).

The performance of QTCP depends on appropriate selection and design of above mentioned elements, and we will further discuss them in the following sections. In general, QTCP works by checking the values of selected state variables and passing these state values to the currently trained policy to generate an action to adjust *cwnd*. Then QTCP observes the new state and the reward and uses them as an input to the training algorithm that evaluates and improves the *cwnd* changing policies. Since we choose Q-learning as the training algorithm, the above process can be conducted in an on-line manner.

The key challenge of applying learning algorithm to congestion control is sifting through the overwhelming number of state combinations used to model the environment. To solve this problem, QTCP takes advantages of advanced function approximation technique to learn the value functions of encountered state-action pairs and optimal policies when observing new states from the network environment. Specifically, we first choose the adaptive Kanerva-coding as a base-line algorithm and then propose an improved generalization-based mechanism to further speed up the learning process and optimize the training qualities (Section 4).

3.2 States

Network topologies can be complicated and traffic can undergo dramatic changes, especially when considering competition among multiple data flows and arbitrary bandwidth changes. The continuous, high-dimensional state space used to represent the network can generate a nearly infinite number of states. Many state variables can describe the characteristics of the network environment, such as the most-recent sample RTT, average time between the timestamps when sending packets, average inter-arrival time between newly received ACKs, the average throughput in a time interval, the average RTT in a time interval, the threshold, the immediate *cwnd* or the average *cwnd* during a past time interval, etc. A high-dimensional state space that consists a large set of state variables would not only exponentially enlarge the size of the state space to explore, but also significantly delay convergence. It makes sense to reduce the set of state variables and only focus on those features of the environment that relate to the agent's goal. We therefore need to identify the appropriate state variables that can capture the performance of actions taken by QTCP and guarantee the tractability of the learning process.

We consider the state space used in Remy [12] and choose our three state variables described as followings:

- *avg_send:* the average interval between sending two packets.
- *avg_ack:* the average interval between receiving two consecutive ACKs.
- *avg_rtt:* the average RTT.

We calculate *avg_send* by taking the average of several packet-sending intervals in a time window (one RTT) to reduce the estimation bias. The *avg_send* and *avg_rtt* are calculated in a similar way. All values are represented in milliseconds and commercially rounded to nearest integer values.

TABLE 1
cwnd Modification Options

Change in *cwnd*	Extent of change (bytes)
Increase	10
Decrease	-1
No change	0

We use these three state variables because they are significantly affected by network congestion and can be seen as efficient "congestion probes". For example, *avg_send* characterizes the traffic sending rate at the sender side and *avg_ack* reflects the real goodput measured at the receiver side. If there is no congestion, then ideally *avg_send* should be equal with *avg_ack*. On the other hand, $avg\_send < avg\_ack$ indicates a high possibility of congestion and the sender should slow down its sending rate.

Note that although we reduce the number of state variables to three, the state space is still huge because of the large number of different values these variables can take. For example, assuming the range for avg_ack and avg_send is from 1 to 175 and the range for avg_rtt is from 120 to 155, then there will be a space of 1,115,136 states. Such a large space poses a great challenge to the design of QTCP, as it can make the learning process very slow and could make convergence impossible. We will further discuss this problem in Section 4.

3.3 Actions

The selection of actions is the key to the QTCP's performance. An action specifies how QTCP should change its $cwnd$ in response to variations in the network environments. While there are an infinite number of actions the QTCP can take, we only allow three actions to simplify our action space (shown in Table 1). The first action increases the *cwnd* by 10 bytes. The second action reduces the size of *cwnd* by 1 byte making it possible to reduce the congestion issue in the network flow and the last action does nothing to the size of *cwnd* letting the *cwnd* remains the same as before. The reason why we assign a relatively large value, i.e., 10, to increase the size of $cwnd$ and at the same time assign a relatively small value, i.e., 1, to reduce the size of $cwnd$ is that we intend to encourage the agent to quickly increase the $cwnd$ to utilize the bandwidth while still offering an option to decrease the sending rate when necessary. We pre-set increase/decrease values for corresponding actions in our design. We found that these values can achieve satisfactory performance across different networking scenarios since our RL-based QTCP can adaptively learn a serial of optimal actions to accommodate the new network scenarios. Note that in QTCP, the control decisions are learned from actual experience and thus it eliminates the need for necessary pre-coded rules to adapt to various scenarios of the network environment. The set of available actions could also be related to particular network conditions, e.g., using a function of the packet size to direct the rate of increase/decrease of $cwnd$. However, we have found that this approach does not lead to significant performance improvements.

Note that when the size of *cwnd* is changed by one action, this *cwnd* will remain unchanged and be applied to the sender each time it receives an ACK until learning agent makes another action decision to change the size of *cwnd*. This is different from how NewReno updates the *cwnd*. In NewReno, whenever a new ACK is received, the *cwnd* will be adjusted by a numeric value provided by some predefined strategies. However, when applying our learning algorithm to TCP congestion control, our learning agent makes the action decisions in every $t_{interval}$ time interval, allowing the action taken in previous state have enough time to occur on the network flow which also allows the agent accurately measure resulted throughput and RTT since it takes certain time for the sender to count ACKs received (the ACKs are used to measure the throughput and RTT by our learning agent) with respect to those latest sent packets.

3.4 Utility Function and Reward Definition

A utility function specifies the objective of QTCP. The goal of the QTCP is to find the decision policy of *cwnd* that can maximize the value of the utility function for each sender. While QTCP can take a variety of different objectives, we choose proportional fairness and define the utility function as follows [12]:

$$Utility = \alpha \times \log(\text{throughput}) - \delta \times \log(\text{RTT}), \quad (1)$$

where α and δ control the relative weight or importance of throughput and RTT. Intuitively, the function emphasizes that every flow should try to maximize its throughput while minimizing delay. The *log* function ensures that the network can converge to a proportional fairness allocation of bandwidth resources when multiple users compete for the same bottleneck link.

While the utility function is the true objective that the QTCP attempts to optimize, we find that its value cannot be used to indicate the desirability of the action taken in particular state and hence cannot be directly used as a *reward* for the learning process. To see why this is the case, assume that the $cwnd$ adjusted by a learning agent has already converged to the optimal *cwnd* w_1 that achieves 100 percent link utilization. Then suppose a new flow comes in, now the available bandwidth for each flow is 50 percent of the total bandwidth. In this scenario, the optimal action for the first flow should halve the $cwnd$ to $w_1/2$ to fully utilize the current available 50 percent of the total bandwidth. However, this is not the case for the original flow if using utility value as a reward signal. Instead of choosing actions that involve an amount of decreases to the $cwnd$, the learning agent would still retain the preference of previously adjusted *cwnd* w_1 since its resulted utility value, while being reduced to a smaller value, could still be a positive value and thus continue enhancing the preference of taking previous action to change the $cwnd$ that proves unsatisfactory in the new case. The ambiguity in action evaluation comes from the unique dynamic network environment the learning agent is interacting with, and it means we cannot simply take the utility value as the reward.

Instead, we consider the difference between consecutive utility values to define the reward. This is because an increase in the utility value indicates an improvement and hence the corresponding action should be encouraged, regardless of the original value of the utility function. This naturally overcomes the ambiguity in action evaluation brought by varying network scenarios. Therefore, we define the reward as follows:

- a, if $U_t - U_{t-t_{interval}} > \epsilon$
- b, if $U_t - U_{t-t_{interval}} < -\epsilon$

where a is a positive value and b is a negative value both of which are used to indicate the reward (a reinforcement signal) given the direction of changes between two newly observed consecutive utility values. The ϵ sets a tolerance of the changes between utility values. It is a tunable parameter that sets the sensitivity of the learning agent to changes in the utility values.

3.5 Training Algorithm

The training algorithm attempts to find a policy that selects actions under specific states to maximize the long term reward received by the agent. In QTCP, the reward is decided by the utility function, so the goal of the training policy is to automatically develop the *cwnd* varying strategy to maximize the throughput while minimizing delay. Therefore, the learning speed and quality of the training algorithm is the key to the performance of QTCP.

3.5.1 Why Q-Learning

Many training algorithms can be used to solve the RL problems. For example, dynamic programming (DP) methods are proved to be able to compute optimal policies in RL given a complete and accurate model (i.e., state transition probabilities) of the environment that can be described as a finite Markov decision process (MDP). However, DP's particular need of a perfect model of transition probabilities about network dynamics makes it unsuitable to solve our problem since initially in learning, we have no prior knowledge on the network environment. Alternatively, two model-free methods, Monte Carlo and temporal-difference (TD) learning methods, have been proposed. Unlike DP's dependence on a model to calculate the value function of each state, Monte Carlo methods average all returns starting from a state to calculate its value function when an episode terminates. However, learning the value function and optimal policy on this episode-by-episode basis in Monte Carlo method makes it infeasible to apply to our congestion control task since the congestion control task is so time-sensitive that it needs fine-grained step-by-step incremental updates on value functions to develop satisfactory policies. Moreover, as a continuing task, it is impractical to divide the task into a sequence of episodes for learning.

Nowadays, the most widely used technique in RL is the class of TD learning methods that are not only model-free but also can be incrementally implemented on a step-by-step basis and with minimal computational cost. Q-learning, one popular TD method that has been successfully applied to many practical applications, can effectively learn value functions and optimal policies in an on-line manner. Therefore, we use Q-learning as the training algorithm in our learning-based TCP congestion control design.

3.5.2 Apply Q-Learning in QTCP

Q-learning uses a simple value iteration update process. At time step t, for each state s_t and each action a_t, the algorithm calculates an update to its expected discounted reward, or action-value function $Q(s_t, a_t)$ as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t(s_t, a_t)[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)], \quad (2)$$

where r_{t+1} is an immediate reward, $\alpha_t(s_t, a_t)$ is the learning rate such that $0 \leq \alpha_t(s_t, a_t) \leq 1$, and γ is the discount factor such that $0 < \gamma \leq 1$. Q-learning stores $Q(s_t, a_t)$ values as entries in a table, called the Q-table.

Q-learning gives good learning performance and fast convergence for problem domains with small state spaces. However, Q-learning scales very poorly with large-scale state spaces, especially when the state spaces consist of continuous variables. In this case, the enumerated states are infinite and it is impractical to use Q-table to record value function for each state or state-action pair. To reduce the memory required to store large Q-tables and the training time required to update values of visited states or state-action pairs, generalization technique, also called function approximation, that relates values among similar states has been proposed and widely used when applying RL algorithm to domains with high-dimensional, continuous state spaces.

4 Practical QTCP with Function Approximation

As described above, it is very challenging, if not impossible, for the Q-learning algorithm to successfully train the high-performance policy given the extremely large state space. In this section we discuss how to make QTCP practical with *function approximation* based on Kanerva coding. Kanerva coding is an effective approach to reduce the number of states needed for trainings, making the Q-learning tractable in the presence of a high-dimensional, continuous state space. We first introduce the basic adaptive Kanerva coding algorithm. Then, based on the limitation of the original algorithm, we describe a new generalization-based Kanerva coding scheme that achieves higher function approximation ability and more stable policy training performance.

4.1 Basic Algorithm

Kanerva Coding (SDMs). In the architecture of Kanerva coding [7], a set of *prototype states* (prototypes) is selected and used to approximate the value functions, where the state or state-action values are estimated by a linear combination of values of local prototypes. In each time step, only prototypes that are adjacent to the input sample data are updated. A prototype is described by a sequence of state variables and each state variable has a range of numerical values. In Kanerva coding, a collection of k prototypes is selected before learning. Given a state s and a prototype p_i, $||s - p_i||$ represents the number of state variables whose values differ between them. A state s and a prototype p_i are said to be *adjacent* if s and p_i differ by less than a given amount, e.g., differing in at most one state variable or dimension. We define the *membership grade* $\mu(s, p_i)$ of state s with respect to prototype p_i to be equal to 1 if s is adjacent to p_i, and 0 otherwise. A value $\theta(p_i, a)$ is maintained for the i_{th} prototype p_i and action a, and $\hat{Q}(s, a)$, an approximation of the value of a state-action pair (s, a), is then the sum of the θ-values of the adjacent prototypes of state s with action a, defined as follows:

$$\hat{Q}(s,a) = \sum_i \theta(p_i,a)\mu(s,p_i). \quad (3)$$

When the learning agent takes action a in a state s, receives a reward r and transitions to next state s' where a new action a' is chosen, the θ-value for each prototype is updated as follows:

$$\theta(p_i,a) \leftarrow \theta(p_i,a) + \frac{\mu(s,p_i)}{M}\alpha(s,a)[r + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a)], \quad (4)$$

where M is the number of adjacent prototypes of the state s.

Adaptive Kanerva Coding. If the set of prototypes is chosen wisely, Kanerva coding works well [13]. However, its ability to estimate the value functions accurately is sensitive to the size and allocation of the set of prototypes [14]. If prototypes are not well distributed across the appropriate regions of the state space, many input sample data, in this case, the visited states, will not be adjacent to sufficient prototypes, or even worse, they will not have any adjacent prototypes at all to estimate their values. This scenario is caused by poorly selected prototypes and greatly reduces the accuracy of value function estimation. As a result, the learning quality and convergence can be affected.

To solve this problem, an adaptive Kanerva-based approach [15] was proposed and also effectively used by [16] when solving an Xpilot-AI video game task that has a high-dimensional continuous state space. This approach starts with an initial set of randomly selected prototypes and periodically deletes poorly-performing/rarely utilized prototypes and generates corresponding new prototypes (e.g., ones that are adjacent to most frequently utilized prototypes) to gradually adjust the allocation of original set of prototypes to cover the region of interest. However, the scheme that generates new prototypes based on certain prototypes found in the initial prototype set limits its ability to further explore better layout of the set of prototypes. Moreover, prototypes should not be permanently deleted based on their periodically-observed performance since their performance may fluctuate and also their long-term importance may be as yet unperceived. In addition, prototype deletions could lose previously-learned information and newly-generated prototypes need time to retrain both of which would slow down convergence time.

4.2 Dynamic Generalization Kanerva Coding Algorithm

We propose a generalization-based Kanerva coding technique that provides a general methodology for learning agents to automatically adjust and explore potential space abstractions and manage the levels of generalization for each prototype in a flexible and adjustable manner.

4.2.1 *Prototypes' Varying Abilities to Generalize*

The size of the state space covered by a prototype, also called the receptive field of the prototype, corresponds to the number of similar/neighboring states that are adjacent to the prototype. During the learning process, prototypes encounter differing numbers of similar/neighboring states and thus have varying sizes of receptive fields that may partially overlap with one another. Fig. 4 shows that prototypes P1 – P14 have varying sizes of receptive fields within which

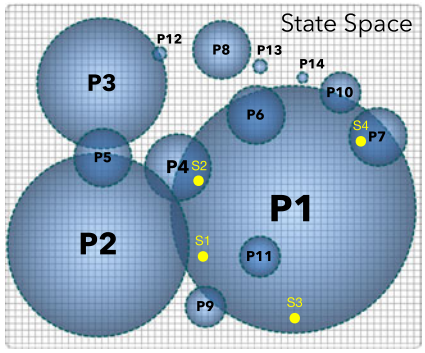


Fig. 4. Various sizes of receptive fields of 2-dimensional prototypes. Note that the size of the receptive field indicates the level of generalization for each prototype and is measured based on the periodically observed visit frequency of each prototype.

states S1, S2, S3 and S4 in P1 share the values learned by the prototype P1 that covers them. The receptive field of each prototype shows how many adjacent states each prototype is able to share its value with. Note that the size of receptive field of each prototype indicates the prototype's ability to generalize since its acquired knowledge can be applied to adjacent states. Empirical results show that prototypes in Kanerva-based approaches usually have varying levels of generalization. As shown in Fig. 4, prototypes have different generalization abilities and their levels of generalization decrease from prototype P1 to prototype P14. Note that we use the number of encountered states, not the total number of states, that are adjacent to the prototype to estimate the level of generalization for that prototype.

Generalization, a natural feature of state abstractions, enables adjacent states to reuse experience gained from previously learned state samples. To achieve good state abstraction in Kanerva-based solvers, we must carefully determine the level of knowledge sharing or generalization. Since the transportation of packets in the network is complex and incorrect decisions may cause the network environment to degrade, our learning process can be interrupted in certain situations that are handled by standard TCP mechanism such as slow start and fast recovery. Once these mechanisms have completed, additional time is needed to restore the network environment and resume the learning process. To mitigate the effects of this problem, we want our learning algorithm to converge quickly. The algorithm should have quick convergence through a relatively small number of trials, and should achieve satisfactory performance in a short time period.

Inappropriate generalization, such as over- and under-generalization from the set of prototypes, would result in poor abstractions and would ultimately degrade the learning quality. For example, if the receptive field of one prototype, e.g., P1 in Fig. 4, is very large, making P1 relate to too many states, all states that reside in P1's receptive field share the same gained experience from P1 and in turn use their immediate rewards to update P1's value during the value update phase. This allows each state residing in P1 to be affected by any value updates from other states that also reside in P1. Since P1's receptive field is large and P1 is adjacent to many states, P1's high generalization can deteriorate the value updates to many adjacent states in P1.

To make the explanation more clear, we use the states S1 to S4 in prototype P1 in Fig. 4 as an example. Since S4 is adjacent

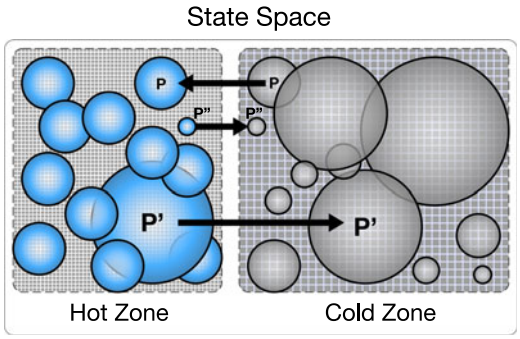


Fig. 5. Illustration of prototype migration between the hot zone and cold zone.

to P1, all knowledge gained by P1 will transfer to S4. In this case, all P1's adjacent states when encountered, i.e., S1, S2, S3, etc., would apply their rewards to P1, which would then again be transferred to S4. Since S4 shares gained knowledge from other prior state samples and it is very likely that S4 may favor totally different gained knowledge, its value/knowledge can be learned very coarsely or arbitrarily making its value less useful, even after additional learning takes place. We refer to this situation as over-generalization and we seek to use much finer-grained generalization to explore the large area that is covered by P1's receptive field. In our algorithm, prototypes with over-generalization are replaced with moderately-generalized prototypes that provide much finer-grained generalization and hence faster convergence. If the receptive fields of prototypes, e.g., P14 in Fig. 4, are very small, they have little or no generalization abilities and should be replaced with prototypes that provide higher generalization and more coarsely-grained discrimination.

Our generalization-based prototype optimization algorithm combines the following steps to construct desired generalization abilities for considered prototype set. First, we select a number n of states from the state space as a set of prototypes $\vec{p_1}$ for learning. Second, in every k time steps, i.e., a time period during which k number of states are encountered, we calculate the level of generalization for each prototype. To make the set of prototypes' current level of generalization dynamically follow an appropriate range, we first calculate the prototype set's average level of generalization V_{avg} based on all k encountered states during current time period. Then we remove prototypes whose generalization levels are much bigger or smaller than V_{avg} (see Line 15 in Algorithm 1 for details). Then the same number of state prototypes from the candidate prototype set $\vec{p_2}$ that have moderate (see Line 17 in Algorithm 1 for details) levels of generalization abilities are selected and introduced to the prototype set $\vec{p_1}$.

4.2.2 Hot Zone and Cold Zone

One contribution of our technique is that it automatically identifies prototypes with inappropriate levels of generalizations and replaces those prototypes with alternative ones, giving agent the ability to adjust the levels of generalization from coarse-grained to finer-grained over time.

However, some problems arise when dealing with newly introduced prototypes. First, after new prototypes are added to the original set of prototypes, their associated θ-values are typically set to 0 or other values borrowed from similar states. Since these θ-values are not actually learned by the new prototypes, it takes a number of learning cycles to correct the initial bias and approach to the correct θ-values. It would be useful to avoid this cost to relearn the values of new prototypes and be able to use their sufficiently learned values immediately after they are introduced to the prototype set.

Algorithm 1. QTCP with Generalization-Based Kanerva Coding

Input: $\vec{p}$: a set of prototypes, $\vec{\theta}$: θ_i is the value maintained for p_i, $\vec{g}$: g_i indicates the level of generalization of p_i
Output: π: action decisions to adjust the $cwnd$

1 **Procedure** `Main()`
2 Randomly select a set of prototypes $\vec{p}$ from state space and initialize each θ_i and g_i to 0
3 Divide $\vec{p}$ into two sets with different sizes, the set $\vec{p1}$ with a small size is hot zone and the set $\vec{p2}$ with a large size is cold zone
4 **for** *each new ACK received in congestion avoidance mode in TCP NewReno* **do**
5 **if** *reward is updated* **then**
6 **if** *encountering accumulatively k number of states* **then**
7 `PrototypeOptimization` ($\vec{p1}$, $\vec{p2}$, $\vec{g}$)
8 `TraditionalKanervaCoding` (*pre_s*, *pre_a*, $\vec{p1}$, $\vec{\theta}$, $\vec{g}$)
9 Also update prototypes' θ-values and levels of generalization in cold zone $\vec{p2}$
10 **else**
11 Take action *pre_a* and apply it to *cwnd*
12 **Procedure** `PrototypeOptimization` ($\vec{p1}$, $\vec{p2}$, $\vec{g}$)
13 V_{avg} = average level of generalization in $\vec{p1}$
14 **for** *each prototype* $\vec{p1}_i$ *in* $\vec{p1}$ **do**
15 **if** $g_i < V_{avg} * (1 - \beta)$ or $g_i > V_{avg} * (1 + \beta)$ **then**
16 Migrate $\vec{p1}_i$ to $\vec{p2}$
17 Select one prototype $\vec{p2}_j$ in $\vec{p2}$ if $V_{avg} * (1 - \beta) < g_j < V_{avg} * (1 + \beta)$
18 Then migrate $\vec{p2}_j$ to $\vec{p1}$

Second, we note that the quality of newly introduced prototypes can be poor. We want the new prototypes to be useful and effective in the learning process; otherwise, their introduction may reduce the quality of the abstraction. Therefore, generating qualified new prototypes is vital part of the reallocation of the set of prototypes in any Kanerva-based approach.

Finally, we observe that deleting prototypes can unnecessarily eliminate previously-gained knowledge from the system. The performance of prototypes can vary over time, and prototypes that are not effective at one time may be useful later. Regenerating and training prototypes can take a significant amount of time.

Our new algorithm solves the three problems described above. Prototype migration, demonstrated in Fig. 5, uses two sets of states referred to as the *hot zone* and the *cold zone*. The hot zone is the regular set of prototypes used to represent the state abstraction for learning, used in the traditional Kanerva-based approach. We apply our generalization-based prototype removal and introduction on the hot zone. The cold zone consists of a number of random prototypes and prototypes that have been removed from use in the hot zone. Prototypes in the cold zone are continuously trained (i.e., their values continue to be updated for possible future

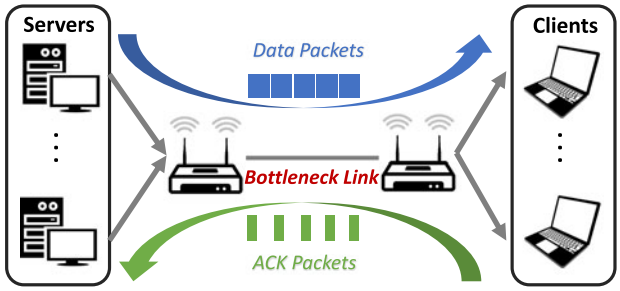


Fig. 6. Overview of the network topology.

use). The cold zone is used to provide qualified prototypes that have desired levels of generalization and already sufficiently leaned knowledge.

It is worth noting that there is a trade-off between introducing the cold zone for the sake of unqualified prototypes retraining and discarding them entirely to achieve less computational overhead. The design of the cold zone inevitably introduces certain costs, i.e., extra memory space to store prototypes and learning cycles to train these prototypes. However, our results show that the use of a cold zone improves the performance of function approximation enough to compensate for the additional computation and memory costs incurred.

4.2.3 QTCP with Generalization-Based Kanerva Coding

We apply the regular learning processes to the hot zone, including updating prototypes' θ-values, calculating the approximated Q-values from prototypes, and calculating prototypes' levels of generalization. At the same time, prototypes' values in the cold zone are also updated and their levels of generalization are calculated. Note that prototypes in the cold zone are not used to do state abstractions and are therefore not directly involved in the estimation of Q-values.

During the prototype optimization phase, we migrate prototypes that meet the migration criterion between the hot zone and the cold zone. Prototypes in the hot zone whose generalizations are undesirable are migrated to the cold zone and prototypes in the cold zone that have desired generalization abilities are selected and migrated to the hot zone to maintain the same amount of prototypes in hot zone. Through this mechanism, when we need new prototypes to supply to the hot zone, qualified prototypes that have desired generalization abilities can be easily found in the cold zone and when they are migrated to the hot zone, their θ-values are already preset. These updated prototypes can be immediately used for state abstractions and to approximate Q-values. In addition, since inappropriate prototypes are migrated to the cold zone instead of being deleted, we reduce the risks of permanently losing previously learned values as well as deleting prototypes with as-yet undiscovered potential.

Algorithm 1 describes how our learning agent handles inappropriate generalization of the set of prototypes and automatically adjusts the set's generalization for better abstraction in QTCP. In our algorithm we combine the generalization-based prototype optimization and the prototype migration between the hot zone and the cold zone together. The goal is to employ appropriate levels of generalization for states that should further share gained knowledge, and prevent states that favor finer-grained discrimination or different action strategies from being inappropriately generalized. In addition, we argue that our technique offers the potential to allow learners to manage and modify the levels of generalization on the set of prototypes in a flexible way, making it easily adaptive to other contexts. Note that in Line 5, the reward is updated in every $t_{interval}$ time interval. This parameter is adjusted by the designer to get accurate measurements when updating the rewards in the given context. In Line 15, $\beta \in \mathbb{R}$ (where $\beta \in (0, 1)$) is the generalization tolerance factor we defined to set a customized range of levels of generalization.

TABLE 2
Simulation Parameters

Parameter	Value setting
Learning rate α	initially 0.95, reduced by a factor of 0.995 per second
Exploration rate	initially 0.1, reduced by a factor of 0.9995 per second
Discount factor γ	0.9
Generalization tolerance factor β	0.8
Reward update time $t_{interval}$	0.23 second
Simulation time	800 seconds
RTT	120 ms
Buffer size	200 packets

5 Evaluation

We implement QTCP and evaluate its performance under various networking scenarios in the *ns-3* simulator. We focus on showing that QTCP is able to automatically learns the right *cwnd* varying strategy to achieve high flow throughput while maintaining low delay without manually crafted fixed rules.

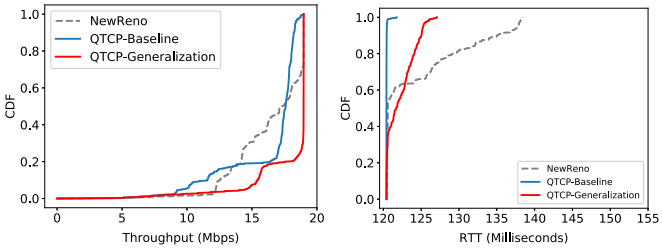
5.1 Experimental Setup

Fig. 6 shows the network topology we used in our experiments. The topology represents a typical dumbbell-like network model, where multiple flows compete for the bottleneck link's bandwidth. The congestion control protocol must dynamically adjust the *cwnd* of each sender to enable them sharing the bandwidth resources fairly with each other. We first evaluate QTCP with fixed bottleneck link bandwidth to demonstrate the characteristic features of learning and state abstraction in this simpler networking environment (Section 5.2). Then we extend the evaluation to varying bottleneck link bandwidth situation to demonstrate the ability of QTCP to adapt to more complex, non-stationary network scenarios. (Section 5.3).

We focus the comparison of following three approaches:

- *NewReno:* classical and default congestion control protocol in use today
- *QTCP-Baseline:* QTCP with the original adaptive Kanerva coding algorithm, which is a state of art function approximation method that serves as a baseline
- *QTCP-Generalization:* QTCP with our newly proposed generalization-based Kanerva coding

Parameter Setting. The simulation parameters used in our experiments are summarized in Table 2. Note that besides the parameters shown in Table 2, we also vary the values of three



(a) Real-time throughput (b) Real-time RTT

Fig. 7. CDF comparisons in fixed bandwidth network.

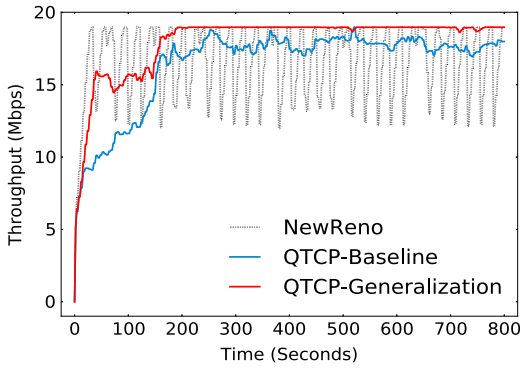


Fig. 8. Real-time throughput comparisons in fixed bandwidth network.

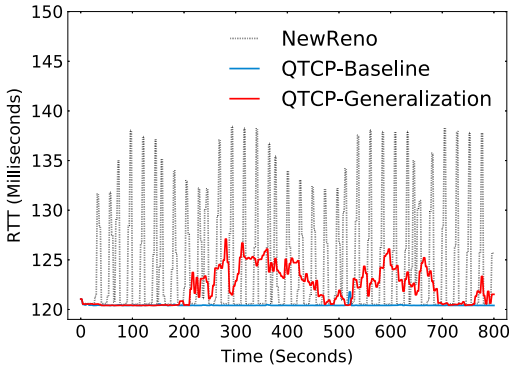


Fig. 9. Real-time RTT comparisons in fixed bandwidth network.

parameters, generalization tolerance factor β, reward update time $t_{interval}$ and exploration rate, to see their impacts on the throughput performance of QTCP-Generalization algorithm (see Figs. 10 and 11).

To run the experiments on adaptive Kanerva coding algorithm, we randomly generate an original set of prototypes $\vec{P}_1$ with 300 randomly selected states and initialize their corresponding θ-values to 0. Our generalization-based Kanerva coding algorithm uses an original set of prototypes (hot zone) to enable state abstraction and implement the regular learning process. The algorithm also uses another larger set of states (cold zone) to provide well-trained prototypes with desired levels of generalization. To fairly compare both algorithms, our generalization-based Kanerva coding algorithm uses the same original set of prototypes $\vec{P}_1$ used by adaptive Kanerva coding. We randomly generate a set of 900 states from the state space to construct the cold zone.

5.2 Fixed Bottleneck Bandwidth Network

Fig. 7a shows the CDF of real-time flow throughput with different congestion control algorithms in a fixed bandwidth

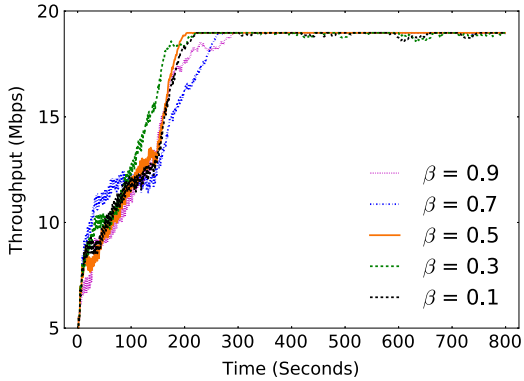
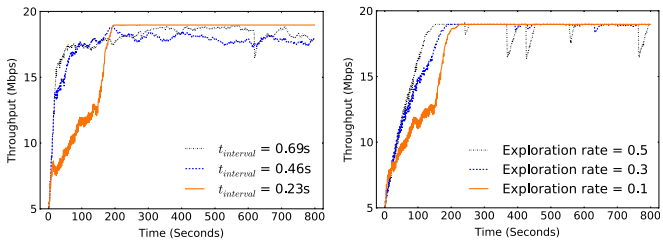


Fig. 10. Real-time throughput comparisons of QTCP-Generalization with various values of generalization tolerance factor β in fixed bandwidth network.

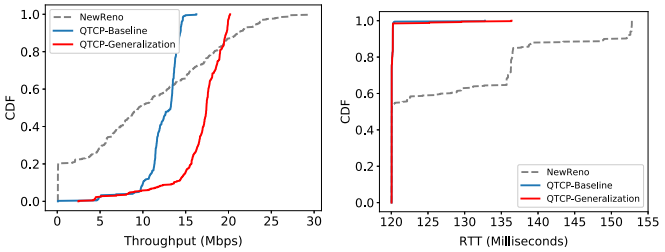


(a) Varying reward update time $t_{interval}$ (b) Varying exploration rate

Fig. 11. Real-time throughput comparisons of QTCP-Generalization with various values of reward update time $t_{interval}$ and exploration rate in fixed bandwidth network.

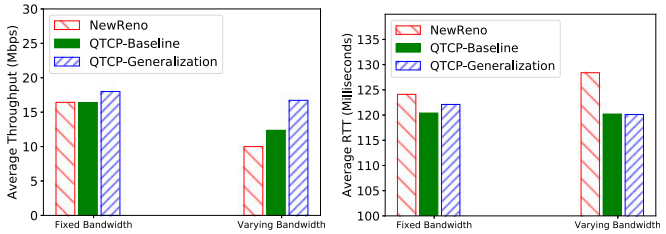
network. We set the bottleneck bandwidth in this network to be 40 Mbps. We observe that QTCP-Generalization achieves better performance than the alternatives—the median throughput reaches close to 20 Mbps, which is 14.7 and 14.9 percent higher than QTCP-Baseline and NewReno, respectively. Fig. 8 shows the real-time flow throughput averaged between two senders. The results show that our QTCP-Generalization outperforms QTCP-Baseline during the entire learning process. QTCP-Generalization and QTCP-Baseline outperform NewReno in terms of both average throughput and stability, especially after the learning process has converged (after 200 seconds).

From Fig. 8 we can make two further observations: *First*, the instantaneous throughput of NewReno fluctuates and cannot remain stable. The main reason is that the fixed AIMD rule used by NewReno forces the *cwnd* to be halved when packet losses are observed, which results in low and unstable average throughput. On the other hand, QTCP could learn optimal control behavior by leveraging the outcomes of different decisions made to interact with the network environment and eventually gain the ability to appropriately change *cwnd* and avoid repetitively taking ineffective actions. *Second*, equipped with an effective prototype optimization strategy, QTCP-Generalization achieves even higher and more stable throughput than QTCP-Baseline. As shown in Fig. 8, QTCP-Generalization learns the decision policy quicker than QTCP-Baseline, and is able to achieve and retain a very high (nearly optimal) and stable throughput until the experiment ends. In comparison, QTCP-Baseline converges slower and achieves worse throughput. The key reason is that once the baseline approach deletes rarely-visited prototypes, it generates new



(a) Real-time throughput (b) Real-time RTT

Fig. 12. CDF comparisons in varying bandwidth network (periodically switching between 30Mbps and 60Mbps).



(a) Average throughput (b) Average RTT

Fig. 13. Average throughput and RTT comparisons in fixed and varying bandwidth network.

prototypes by splitting those most-visited prototypes. As a result, many frequently-visited prototypes with similar layouts may be added. These prototypes dominate the set of prototypes, causing very coarse-grained generalization and insufficient detailed exploration on the state space. Since both Kanerva-based function approximation approaches start with the same original set of prototypes, the superior performance of QTCP-Generalization comes from the generalization-based prototype optimization that introduces prototypes with fine-grained generalization to reallocate the set of prototypes in order to guarantee sufficient complexity of needed approximation to function values for each visited state.

Fig. 7b shows that the high throughput achieved by QTCP does not sacrifice RTT performance. About 98th percentile RTTs of QTCP-Generalization are between 120 ms to 126 ms while only 68th percentile RTTs of NewReno are in this range. QTCP-Baseline basically does not introduce queuing delays. Fig. 9 shows the real-time RTT of all three algorithms. Like throughput, the RTT of NewReno flow also suffers from periodic fluctuations due to the fixed rule-based policy. QTCP-Generalization achieves better performance by accumulating only a small number of queuing packets, but still keeps queuing delays relatively small. Note that QTCP-Baseline achieves the optimal performance in terms of delay at the cost of lower throughput. As we show in Fig. 8, the QTCP-Baseline does not fully utilize the link due to its insufficient optimization on the layout of prototype set and conservative sending policy (but still achieves better throughput than NewReno).

In both Figs. 10 and 11, we evaluate the sensitivity of the performance of QTCP-Generalization to variations in the settings of important parameters, i.e., generalization tolerance factor β, reward update time $t_{interval}$, and exploration rate (the fraction of time when the agent chooses a random action instead of the currently learned best one to explore more possibilities), and we also investigate the convergence of the algorithm.

Fig. 10 shows parameter β's sensitivity evaluation on the performance measures. We observe that the learning methods with smaller β values generally converge faster than ones with larger β values, e.g., β = 0.1, 0.3 and 0.5 improve the performance faster than β = 0.7 and 0.9. In addition, we observe that β values equal to or larger than 0.5 can give more stable and slightly better throughput performance. We conclude that for QTCP-Generalization, it is efficient to merely migrate prototypes in the hot zone whose generalization abilities are far above or below the average level of generalization.

As shown in Fig. 11a, the methods with larger $t_{interval}$ values converge faster but end up with inferior overall performance. The value of $t_{interval}$ should not be too big; otherwise, the agent may not have sufficient opportunity to explore the action space since a large $t_{interval}$ reduces the frequency at which the learning algorithm is triggered.

Fig. 11b shows that when the value of the exploration rate is equal to 0.5, the learning method converges fastest, and when this value decreases, convergence becomes slower. However, although an exploration rate of 0.1 makes the agent converge slower than ones with 0.3 and 0.5, it eventually gives more stable and better throughput performance.

5.3 Varying Bandwidth Network

We next extend our evaluation to more complex scenarios where the bottleneck link bandwidth can vary significantly. This model is used in mobile networks when the user switches between different wireless accessing links with different capacities (e.g., LTE or 60 GHz WiFi). In the varying bandwidth network, the bandwidth switches alternately between 30 Mbps and 60 Mbps. When the 60 Mbps bandwidth is triggered, it stays at that value for 40 seconds, and when switching to the 30 Mbps bandwidth it stays at that value for 10 seconds. The varying bandwidth network challenges both Kanerva-based learning algorithms since it generates more complex dynamics in the network environment and the effectiveness of function approximation technique is the key to learn a practical policy in such a large-scale state space.

The CDF comparisons of our results in the varying bandwidth network are shown in Fig. 12. In Fig. 12a, we show that approximate 85th percentile throughputs gained by our approach are between 15 Mbps and 20 Mbps while there are only 20th percentile throughputs gained by NewReno are in this range and almost none of the throughputs of QTCP-Baseline have reached this range. The delay comparisons are shown in Fig. 12b. The figure shows that the delay gained by our approach is almost the same as the link RTT (120 ms) and can be maintained at this value nearly until the end of the experiments, while there are about 30th percentile RTTs of NewReno larger than 136 ms that are much worse than our QTCP-Generalization approach.

We repeat our simulation 5 times for each algorithm in both fixed and varying bandwidth network scenarios and report the average throughput and delay in Fig. 13a and 13b. Fig. 13a shows that in the fixed bandwidth network, the average throughput of QTCP-Generalization is 18.0 Mbps that outperforms both QTCP-Baseline and NewReno by 9.6 percent. In the more complex varying bandwidth network, the average throughput of QTCP-Generalization is 16.7 Mbps that outperforms QTCP-Baseline by 35.2 percent and is much better than NewReno with improvements of 59.5 percent. As

shown in Fig. 13b, QTCP-Generalization has comparable average RTT with QTCP-Baseline in both network scenarios.

5.4 Discussion of Efficient Value Estimation in QTCP-Generalization

We now analyze the efficiency of value approximation for the proposed generalization-based Kanerva coding method in QTCP-Generalization. In our experiments, we found that even when employing a relatively small set of prototypes, (300 prototypes, which is 0.0269 percent of the possible state space), generalization-based Kanerva coding can provide effective state abstraction, and improvements in learning performance have been observed in all experimental evaluations. We argue that it is safe to use a small set of prototypes with the generalization-based Kanerva coding approach since the key component that affects the performance of function approximation is not the large number of prototypes but the reasonable layout of those prototypes. In fact, as long as the number of prototypes are not too few, the learning results with varying numbers of prototypes show no statistically significant difference [16]. Therefore, the superior performance of our proposed generalization-based Kanerva coding, as demonstrated by our experimental results, is mainly a result of the fine-grained levels of generalization for all prototypes that are dynamically adjusted based on visited state samples by our RL learner.

6 Related Work

6.1 Congestion Control Protocols

TCP is a well explored topic in both wired and wireless networking. For years, many end-to-end congestion control mechanisms have been proposed. For example, Cubic uses a cubic function to tweak the *cwnd*, and is known for its ability to aggressively search for spare bandwidth. Vegas [17] uses delay as a congestion control indication and starts to decrease *cwnd* when the measured RTT exceeds expected value. Other well known end-to-end congestion control protocols include Compound [18], Fast [19] and BBR [20]. While these protocols all have their own unique properties, they share the similar idea of using some fixed functions or rules to change *cwnd* to handle network conditions. As we introduced, the limitation of this fixed-rule strategy is that they cannot adapt to the complexity and rapid evolution of modern data networks. They do not learn from experience or history and are not able to predict the consequences of each action taken. Even if an action reduces performance, the algorithm would still mechanically and repeatedly select the same action.

Meanwhile, we notice that a number of techniques have been explored by the research community to solve the limitations of traditional TCP protocols. For example, Remy [12] uses off-line training to find the optimal mapping from every possible network condition to the behavior of the sender. Remy works well when prior assumptions about the network given at design time are consistent with the network situations in experiments. Performance may degrade when real networks violate the prior assumption [21]. The mappings stored in the lookup table are pre-calculated, which, as with other traditional TCP variants, cannot adapt to continuously varying network environment. In Remy's approach, the lookup table must be recomputed (which may take days to train the model) when new network conditions apply.

PCC [22] is a recently proposed protocol that can rapidly adapt to changing conditions in the network. PCC works by aggressively searching for better actions to change the sending rate. However, its performance may diminish in some cases since its greedy exploration could be trapped at a local optimum, requiring certain strategy to approach to the globally optimal solution. Both Remy and PCC regard the network as a black box and focus on looking for the change in the sending rate that can lead to the best performance, without directly interpreting the environment or making use of previous experience.

6.2 Reinforcement Learning and Its Applications

RL has achieved a lot of success in solving sequential decision problems and has been effectively applied to a variety of applications. The advantage of RL is its ability to learn to interact with the surrounding environment based on its own experience. For example, [23] proposed to learn channel assignment decisions with a linear bandit model to minimize the total switching cost in a multichannel wireless network. [24] used deep reinforcement learning (DRL) to handle the large complex state space when solving the cloud resource allocation and power management problem. [25] proposed a DRL-based framework for power-efficient resource allocation in cloud RANs.

Moreover, many reinforcement learning-based schemes have been proposed to improve the quality of service for network applications. For example, [26] proposed a RL-based algorithm to generate congestion control rules to optimize the QoE specifically for multimedia application. [27] formulated a network resource allocation problem in a multi-user video streaming domain as a DEC-POMDP model and applied a distributed RL algorithm to solve the problem. However, the work in [27] did not provide practical technique to help RL algorithm adapt to complex network topologies that have continuous state spaces. [28] used a RL algorithm to adaptively change parameter configuration and thus improve the QoE of video streaming. Its limitation arises from its use of a tabular-based algorithm that directly stores and updates value functions as entries in a table, confining its application to large, continuous domains.

All above mentioned schemes are task-driven. They are designed for particular applications and cannot be directly applied to congestion control problem. In fact, to the best of our knowledge, QTCP is the first proposed solution applying RL to TCP congestion control protocol design directly.

6.3 Related Function Approximation Techniques

Many approximating approaches has been developed to abstract and compress full state spaces in RL tasks that have enormous number of states. One effective approach is function approximation [6], which can reduce the size of the state space by representing it with an abstracted and parameterized function. The explicit table that stores value functions is replaced by an approximate and compact parameterized version. Many function approximation techniques have been developed, including tile coding (also known as CMAC) and its variants, i.e., adaptive tile coding [29], and

tree-based state partitions [30]. However, when solving practical real-world problems, limitations arise from the coding schemes used in those approaches, e.g., requiring task-dependent criteria-based heuristics, spending impractical large computation expenses to explore each dimension for partitions, and the size of state space and function approximation complexity increases exponentially with the number of dimensions. All the limitations prevent considered approaches' applications to domains that are very large, have high dimensions, or that have continuous state spaces.

However, Kanerva coding technique proves to scale well with high-dimensional, continuous problem domains [7]. Our proposed generalization-based Kanerva coding approach further improves its approximation ability and reduces the convergence time by dynamically optimizing the layout of the prototype set and providing a finer-grained discrimination on the explored state areas.

7 Conclusion

Our work describes QTCP, an effective RL-based approach that derives high-quality decision policies and successfully handles highly complex network domains with a broad set of characteristics. Unlike preprogrammed rule-based TCP, QTCP uses the reinforcement signals (rewards) to learn the congestion control rules from experience, needing no prior knowledge or model of the network dynamics. This allows our approach to be widely applicable to various network settings. Moreover, our learning agent applies a novel generalization-based Kanerva coding approach to reduce the training time and necessary state space to search. This approach reformulates original function approximator with adjustable generalization granularity across states, making it possible to abstract sufficient information from a wide range of environment signals and even use a very small subset of state space to accurately approximate the whole state space.

Our QTCP-Generalization achieved better throughput and delay performance than both NewReno and QTCP-Baseline (a learning-based TCP with best currently-existing Kaneva-based function approximator) in our evaluations. We found that the average throughput of QTCP-Generalization outperformed QTCP-Baseline by 35.2 percent and outperformed NewReno by 59.5 percent. Our approach also has a slightly better RTT performance than NewReno. We conclude that QTCP with generalization-based Kanerva coding can be used to manage congestion in a wide range of network conditions, and that the technique enables quick on-line policy development with minimal computation and memory expenses.

References

[1] T. Yilmaz and O. B. Akan, "State-of-the-art and research challenges for consumer wireless communications at 60 ghz," *IEEE Trans. Consum. Electron.*, vol. 62, no. 3, pp. 216–225, Aug. 2016.

[2] O. B. Akan, O. B. Karli, and O. Ergul, "Cognitive radio sensor networks," *IEEE Netw.*, vol. 23, no. 4, pp. 34–40, Jul.-Aug. 2009.

[3] Y. Li, Y. Li, B. Cao, M. Daneshmand, and W. Zhang, "Cooperative spectrum sharing with energy-save in cognitive radio networks," in *Proc. Global Commun. Conf.*, 2015, pp. 1–6.

[4] E. S. Hosseini, V. Esmaeelzadeh, R. Berangi, and O. B. Akan, "A correlation-based and spectrum-aware admission control mechanism for multimedia streaming in cognitive radio sensor networks," *Int. J. Commun. Syst.*, vol. 30, no. 3, 2017.

[5] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *Proc. ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, 2010, pp. 63–74.

[6] L. Frommberger, *Qualitative Spatial Abstraction in Reinforcement Learning*. New York, NY, USA: Springer Science & Business Media, 2010.

[7] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: Bradford Books, 1998.

[8] S. Floyd, A. Gurtov, and T. Henderson, "The NewReno modification to TCP's fast recovery algorithm," RFC 3782, 2004.

[9] D.-M. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Comput. Netw. ISDN Syst.*, vol. 17, no. 1, pp. 1–14, 1989.

[10] F. Zhou, M. Di Felice, B. Drozdenko, and K. Chowdhury, "Towards fast flow convergence in cognitive radio cellular networks," in *Proc. IEEE Global Commun. Conf.*, 2017, pp. 1–6.

[11] K. R. Chowdhury, M. Di Felice, and I. F. Akyildiz, "TCP CRAHN: A transport control protocol for cognitive radio ad hoc networks," *IEEE Trans. Mobile Comput.*, vol. 12, no. 4, pp. 790–803, Apr. 2013.

[12] K. Winstein and H. Balakrishnan, "Tcp ex machina: Computer-generated congestion control," in *Proc. ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, 2013, pp. 123–134.

[13] C. Wu and W. Meleis, "Function approximation using tile and kanerva coding for multi-agent systems," in *Proc. Adaptive Learn. Agents Workshop (ALA) in AAMAS*, 2009.

[14] P. W. Keller, S. Mannor, and D. Precup, "Automatic basis function construction for approximate dynamic programming and reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2006, pp. 449–456.

[15] C. Wu and W. M. Meleis, "Adaptive Kanerva-based function approximation for multi-agent systems," in *Proc. 7th Int. Joint Conf. Autonomous Agents Multiagent Syst.-Volume 3*, 2008, pp. 1361–1364.

[16] M. Allen and P. Fritzsche, "Reinforcement learning with adaptive kanerva coding for xpilot game ai," in *Proc. IEEE Congr. Evol. Comput.*, 2011, pp. 1521–1528.

[17] L. S. Brakmo and L. L. Peterson, "Tcp vegas: End to end congestion avoidance on a global internet," *IEEE J. Select. Areas Commun.*, vol. 13, no. 8, pp. 1465–1480, 1995.

[18] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A compound TCP approach for high-speed and long distance networks," in *Proc. 25th IEEE Int. Conf. Comput. Commun.*, 2006, pp. 1–12.

[19] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "FAST TCP: Motivation, architecture, algorithms, performance," *IEEE/ACM Trans. Netw.*, vol. 14, no. 6, pp. 1246–1259, Dec. 2006.

[20] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control," *Queue*, vol. 14, no. 5, 2016, Art. no. 50.

[21] A. Sivaraman, K. Winstein, P. Thaker, and H. Balakrishnan, "An experimental study of the learnability of congestion control," in *Proc. ACM Conf. SIGCOMM*, 2014, pp. 479–490.

[22] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, "PCC: Re-architecting congestion control for consistent high performance," in *Proc. 12th USENIX Conf. Netw. Syst. Des. Implementation*, 2015, pp. 395–408.

[23] T. Le, C. Szepesvari, and R. Zheng, "Sequential learning for multi-channel wireless network monitoring with channel switching costs," *IEEE Trans. Signal Process.*, vol. 62, no. 22, pp. 5919–5929, Nov. 2014.

[24] N. Liu, Z. Li, J. Xu, Z. Xu, S. Lin, Q. Qiu, J. Tang, and Y. Wang, "A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 372–382.

[25] Z. Xu, Y. Wang, J. Tang, J. Wang, and M. C. Gursoy, "A deep reinforcement learning based framework for power-efficient resource allocation in cloud rans," in *Proc. IEEE Int. Conf. Commun.*, 2017, pp. 1–6.

[26] O. Habachi, H.-P. Shiang, M. van der Schaar, and Y. Hayel, "Online learning based congestion control for adaptive multimedia transmission," *IEEE Trans. Signal Process.*, vol. 61, no. 6, pp. 1460–1469, Mar. 2013.

[27] M. Hemmati, A. Yassine, and S. Shirmohammadi, "An online learning approach to qoe-fair distributed rate allocation in multi-user video streaming," in *Proc. 8th Int. Conf. Signal Process. Commun. Syst.*, 2014, pp. 1–6.

[28] J. van der Hooft, S. Petrangeli, M. Claeys, J. Famaey, and F. De Turck, "A learning-based algorithm for improved bandwidth-awareness of adaptive streaming clients," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage.*, 2015, pp. 131–138.

[29] S. Whiteson, M. E. Taylor, P. Stone, et al., "Adaptive tile coding for value function approximation," Comput. Sci. Dep., Univ. Texas Austin, Tech. Rep. AI-TR-07–339, 2007.
[30] S. Chernova and M. Veloso, "Tree-based policy learning in continuous domains through teaching by demonstration," in *Proc. Workshop Modeling Others Observ.*, 2006, pp. 24–31.

Wei Li received the BS degree in control engineering from the University of Electronic Science and Technology of China, and the MS degree in computer engineering from Northeastern University, in 2012. He is working toward the PhD degree in the ECE Department, Northeastern University. He is doing his research under the instruction of Prof. Waleed Meleis. His current research interests include reinforcement learning algorithms, function approximation technique, and learning-based network applications.

Fan Zhou received the BS and MS degrees from Hohai University, in 2011 and Beijing University of Posts and Telecommunications, in 2014. He is working toward the PhD degree in the ECE Department, Northeastern University. He is currently working in the Next Generation Networks and Systems Lab under the supervision of Prof. Kaushik Chowdhury. His research interests include different layers of wireless networking, with a specific focus on the design and implementation of high performance data transfer architecture.

Kaushik Roy Chowdhury received the MS degree from the University of Cincinnati, in 2006, and the PhD degree from the Georgia Institute of Technology, in 2009. He was an assistant professor from 2009 to 2015 with Northeastern University, where he is now an associate professor with the Electrical and Computer Engineering Department. He received Presidential Early Career Award for Scientists and Engineers (PECASE), in 2017, ONR Director of Research Early Career Award, in 2016 and the NSF CAREER Award, in 2015. His current research interests include dynamic spectrum access, wireless RF energy harvesting and IoT and in the area of intra/on-body communication. He is a senior member of the IEEE.

Waleed Meleis received the BSE degree in electrical engineering from Princeton University and the MS and PhD degrees in computer science and engineering from the University of Michigan. He is an associate professor and associate chair with the Department of Electrical and Computer Engineering, Northeastern University. His research interests include applications of algorithm design, combinatorial optimization and machine learning to diverse engineering problems, including cloud computing, spectrum management, high-performance compilers, instruction scheduling, parallel programming and network management.