Arianpoo and Leung *EURASIP Journal on Wireless Communications and Networking* (2016) 2016:278
DOI 10.1186/s13638-016-0773-3

EURASIP Journal on Wireless Communications and Networking

RESEARCH Open Access

How network monitoring and reinforcement learning can improve tcp fairness in wireless multi-hop networks

Nasim Arianpoo* and Victor C.M. Leung

Abstract

Wireless mesh network (WMN) is an emerging technology for the last-mile Internet access. Despite extensive research and the commercial implementations of WMNs, there are still serious fairness issues in the transport layer, where the transmission control protocol (TCP) favors flows with a smaller number of hops to flows with a larger number of hops. TCP unfair behavior is a known anomaly over WMN that attracts much attention in recent years and is the focus of this paper. In this article, we propose a distributed network monitoring mechanism using a cross-layer approach that deploys reinforcement learning techniques (RL) to achieve fair resource allocation for nodes within the wireless mesh setting. In our approach, we deploy Q-learning, a reinforcement learning mechanism, to monitor the dynamics of the network. The Q-learning agent creates a state map of the network based on the medium access control (MAC) parameters and takes actions to enhance TCP fairness and throughput of the starved flows in the network. The proposal creates a distributed cooperative mechanism where each node hosting a TCP source monitors the network and adjusts its TCP parameters based on the network dynamics. Extensive simulation results and testbed analysis demonstrate that the proposed method significantly improves the TCP fairness in a multi-hop wireless environment.

Keywords: TCP, Fairness, Wireless mesh networks, Reinforcement learning, Distributed

1 Introduction

Enhancing transmission control protocol (TCP) fairness over wireless mesh networks (WMN) is a significant research area that attracts researchers' attention for the past decade. TCP was first designed for wired networks and performs well over wired infrastructure; as such, when wireless networks were introduced, TCP was adopted to wireless environment. However, the fundamental differences between wireless and wired mediums result in substandard performance of TCP over wireless networks. Wireless multi-hop networks are especially affected by TCP unfairness as TCP favors flows with smaller number of hops in WMNs [1]. In a wired setting, the network topology is well defined and each node has a comprehensive knowledge of the available network resources. Therefore, TCP is capable of assigning a fair share of network resources to each flow. However, in a wireless multi-hop network, each node has a partial knowledge of the available network topology and creating a unified map of the links and collision domains using feedback messages is extremely costly in terms of overhead. As such, TCP is not capable of a fair resource allocation over WMN [1–5].

TCP unfairness increases remarkably in uplink traffic where nodes transmit data to the gateway [1]. Findings of [1] over a real WMN and many similar studies motivated us to propose a fair resource allocation mechanism in transport layer over WMN despite an unfair MAC layer. Our proposal uses a distributed mechanism to monitor the network anomalies in resource allocation and tune TCP parameters accordingly. Each TCP source models the state of the system as a Markov decision process (MDP) and uses Q-learning to learn the transition probabilities of the proposed MDP based on the observed variables. To maximize TCP fairness, each node hosting a TCP source takes actions according to the recommendations of the Q-learning algorithm and adjusts TCP parameters autonomously. Our algorithm preserves autonomy

*Correspondence: nasima@ece.ubc.ca
Department of Electrical and Computer Engineering, University of British Columbia, 2356 Main Mall, Vancouver, Canada

of each node in decision-making process and does not require a central control mechanism or control message exchange among nodes. Unlike the existing machine learning solutions, i.e., TCP ex Machina, our proposal is compatible with the computational capacity of the current infrastructure. We call our approach Q-learning TCP. The contributions of this paper can be summarized as:

- Modeling the multi-hop network as an MDP in each TCP source and using Q-learning algorithm to monitor and learn the dynamics of the network and the proposed MDP.
- Finding a cross-layer distributed and scalable solution for TCP fairness over multi-hop networks with no extra overhead. Our proposal enhances TCP fairness over multi-hop networks in favor of flows traversing a longer number of hops with negligible impact on flows with a shorter number of hops via changing TCP parameters cooperatively based on the recommendation of the Q-learning algorithm.
- Enhancing TCP fairness by a factor of 10 to 20% without any feedback messaging and no incurred overhead to the medium.

The rest of this paper is organized as follows: an overview of the available TCP solutions is given in Section 2. Section 3 is a detailed description of our algorithm followed by the implementation specifics in Section 4. Performance evaluation of our proposed algorithm is presented in Section 5 via extensive simulation and testbed experimentation. A discussion on implications of Q-learning TCP along with a comparison with available fairness techniques is presented in Section 6. Throughout this paper, we measure fairness using Jain's fairness index.

2 Related works

Different approaches have been proposed in the literature to address the problem of TCP unfairness over wireless mesh networks [2–11]. The existing literature on TCP fairness solutions over multi-hop wireless networks can be categorized into cross-layer designs e.g [12–21], and layered proposals, e.g., [22–34]. While layered proposals aim to keep the end-to-end semantic of TCP intact, the cross-layer designs use the information from different layers to adjust TCP parameters.

Random early detection (RED) is among one of the first proposals to enhance TCP fairness over wired connections. Neighborhood RED (NRED) is a cross-layer adaptation of RED for wireless multi-hop settings [15]. NRED is implemented in MAC and uses channel utilization from physical layer to calculate the probability of packet dropping. The main disadvantage of NRED is the use of broadcast messages to inform the neighboring node about packet dropping probabilities. In [16], the gateway uses a centralized cross-layer explicit congestion notification algorithm (ECN) to notify the nearby TCP sources that it favors the farther flows and allows them to use more network resources [16]. The fact that the gateway uses a centralized mechanism along with feedback messages makes ECN less favorable for WMN [16]. In [13], a cross-layer solution is proposed that uses network coding in each hop with a different rate to improve TCP throughput, the computational over head of network coding along with the hop-by-hop feedback and rate estimation creates a bottle neck for computational resources of the network. In another instance of cross-layer approach [14], authors used a hop-by-hop congestion control mechanism for fair resource allocation; however, the mechanism in [14] introduces a heavy inter-node and intra-node control traffic. In [35], another hop-by-hop mechanism is discussed that collects information from physical layer and MAC layer to approximate channel utilization in each hop. The estimation of channel utilization is based on carrier sensing which might not be very accurate in some scenarios.

CoDel is another cross-layer example that uses active queue management on selected bottle neck links, i.e., links with large queuing delay. CoDel uses spacing in transmission times as the queue management method over bottle neck links. CoDel has to run on all the hops/routers within TCP flow path; hence, the implementation requires infrastructural change which is not practical. D^2TCP is a recent variation of TCP for on-line data intensive applications that uses varying rate based on a deadline set by the application and the congestion condition of the network reported by ECN [20]. D^2TCP is a cross-layer approach that needs compatibility in both application layer and routing protocol to perform effectively which is a huge disadvantage.

TCP-AP, another instance of the cross-layer method, attempts to eliminate the reliance of TCP fairness solution on feedback messages [18]. The approach of [18] relies on the information from the physical layer to infer the fair share of each node of network resources. The main drawback of TCP-AP is the reliance on received signal strength indication (RSSI) which is not an accurate estimate of receiver power level. As such, TCP-AP still needs to get feedback from the receiver to function effectively. In [21], a cross-layer algorithm, MHHP, is proposed that assigns higher priority to flows traversing larger number of hops. Although MHHP improves TCP fairness in multi-hop environment, the design does not accurately reflect the dynamic nature of the wireless environment.

Unlike cross-layer approaches, layered solutions preserve the end-to-end semantic of the open systems interconnection model (OSI). According to [22], the binary back-off mechanism, request-to-send/clear-to-send signaling (RTS/CTS), and the end-to-end congestion mechanism play key roles in TCP unfairness. In [22], a MAC layer solution is proposed in which each access point calculates the fair share of incoming and outgoing flows in terms of flow rates and broadcasts the rates to other nodes. Reference [24] is another MAC layer approach in which nodes negotiate to set up a fair transmission schedule throughout the network. The proposed methods in [22] and [24] rely on feedback messages which increases the network overhead significantly. In [26], a layered approach is proposed that uses the TCP advertised window and delayed ACK to control flow rates of different flows. The algorithm in [26] only works in scenarios where all flows are destined to the gateway. TCP Veno [32], another instance of layered solution for TCP, gained a lot of attention in recent years due to its better performance over wireless settings; Veno is basically an integration of TCP Vegas into TCP Reno and does not contribute to the fairness significantly.

Among the end-to-end solutions to enhance TCP performance, there are few that use machine learning as an interactive tool to observe network dynamics and change TCP parameters based on some prediction of the network behavior. Sprout is a good example of the interactive transport layer protocols [33]. Sprout uses the arrival time of packets to predict how many bytes the sender can successfully transmit without creating congestion in the network while maximizing network utilization. The main disadvantage of Sprout is the fact that it needs to run over CoDel to outperform other TCP variants; as such, it requires changes to the infrastructure to enable active queue management. TCP ex Machina, also known as Remy, is another end-to-end interactive solution that uses machine learning to create a congestion control algorithm which controls packet transmission rate based on the perceived network model by the learning algorithm [34]. The main disadvantage of TCP ex Machina is its resource-intensive nature that results in lengthly learning time. It takes almost forever for a single Linux machine to come up with a congestion algorithm suitable for a specific network using TCP ex Machina. Hence, TCP ex Machina requires a separate set of nodes with extensive computational ability to learn the network model and then the model has to be loaded into any nodes within the network. Any changes in the network dynamics requires TCP ex Machina to repeat the costly re-learn process.

Q-learning TCP uses reinforcement learning algorithm (RL) to monitor and learn the characterstics of the network; however, it distinguishes itself from [12, 16–34] by covering the unique characteristics of WMN. First, it does not use any feedback messages unlike the majority of the above fairness solutions which is crucial for saving the valuable bandwidth in WMN. Second, it uses RL algorithm to learn the characteristics of the network using minimal computational resources contrary to TCP ex Machina or Sprout. Third, it only requires minimal changes in the source node which is vital for practicality/feasibility of the design. None of [12, 15–34] address all the unique characteristics of WMN stated above in one solution.

3 Q-learning TCP architecture

In our approach, each TCP source is equipped with a Q-learning agent that sees the world as an MDP. Q-learning is a class of reinforcement learning algorithms (RL) first introduced by Watkins et al. [36]. RL algorithms are based on the basic idea of learning via interaction. The learner module is called an *agent*, and the interactive object which is the subject of learning is the *environment*. During the learning process the time is divided into decision epochs. In each decision epoch, the agent receives network statistics in the form of state space variables. The agent uses the received information to determine the state of the MDP; then, the agent takes an action via fine-tuning TCP parameters. In the next decision epoch, the environment responds with the new state. The learning agent uses a reward function to receive feedback on the consequences of the taken action on TCP fairness.

The interaction between the agent and the environment helps the agent to construct a mapping of the possible states-actions. The agent mapping is called the *policies* and shows how the agent changes its decisions based on the different responses from the environment. As time passes, the mapping gets more inclusive and the agent learns almost all the possible $(state, action)$ pairs and their associated reward/penalty values and can cope with any changes in the environment. The memory of the agent of the possible rewards based on the $(state, action)$ pairs is kept in a matrix called Q. The rows of Q represent the current state of the system, and the columns represent the possible actions leading to the next state. At the beginning of the learning process, the learning agent does not know about the rules of the MDP other than the states and the possible actions; therefore, Q is initialized to 0. After each decision epoch, Q is updated as in Eq. (1).

$$\overbrace{Q_{t+1}(s_t,a_t)}^{\textit{new value}} = (1-\alpha)\,\overbrace{Q_t(s_t,a_t)}^{\textit{old value}} + \overbrace{\alpha[\underbrace{r(s_t,a_t)}_{\textit{reward observed after performing } a_t \textit{ in } s_t} + \overbrace{\gamma \max_a Q_t(s_t,a_t)]}^{\textit{estimated future value}}}^{\textit{learned value}} \tag{1}$$

Equation (1) is called the Q-learning or Ballard rule [37]. The objective of the Q-learning rule is to inform the agent of the possible future rewards along with the immediate rewards of the latest action. α is the learning rate of the agent and determines the importance of the newly acquired reward. α varies between 0 and 1. A factor of 0 causes the agent not to learn from the latest $(action, state)$

pair, while a factor of 1 makes the agent to only consider the immediate rewards without considering the future rewards. γ is the discount factor and determines the importance of future rewards. γ varies between 0 and 1. A 0 discount factor prohibits the agent from acquiring future rewards, while a factor of 1 pushes the agent to only consider future rewards. We use a polynomial learning rate $\left(\alpha = \frac{1}{(1+t)^2}\right)$ as it has a faster convergence rate [38]. A discount factor of $\gamma = 0.9$ is suggested by the literature to encourage the agent into a comprehensive discovery of the $(action, state)$ map [36]. The learning process continues as long as the network is up and running; it basically works as a memory that can be adjusted according to network changes.

In the following sub-sections, we present a detailed overview of the key factors of Q-learning TCP including states, action set, reward function, and transition probabilities.

3.1 States

The state space of our proposed Q-learning algorithm in each TCP source is in the form of $S = (fairness\ index, aggressiveness\ index)$. To measure fairness index in each decision interval, the agent uses Jain's fairness index as in Eq. (2):

$$J_k^t(x_1, x_2, ..., x_n) = \frac{\left(\Sigma_{i=1}^{i=n} x_i\right)^2}{n \times \Sigma_{i=1}^{i=n} x_i^2}, \quad (2)$$

where x_i is the data rate of flow i, n is the number of flows that are originated from or forwarded by node k, and J_k^t is the Jain's fairness index at node k at decision epoch t. The Jain's fairness index is a continuous number that varies between 0 and 1; with 0 the worst fairness index and 1 an absolute best fairness condition. To tailor the Jain's fairness for a discrete state space, we divided the $[0, 1]$ interval to p sub-intervals $[0, f_1], (f_1, f_2], \ldots, (f_{p-1}, 1]$. Instead of using a continuous fairness index, we quantize it to have manageable number of states. Number of states is important in convergence of the learning algorithm.

The aggressiveness of each TCP source in each decision epoch is measured as in Eq. (3):

$$G(i) = \frac{number\ of\ packets\ originated\ from\ node\ i}{total\ number\ of\ packets\ forwarded\ by\ node\ i}. \quad (3)$$

The aggressiveness index is a continuous amount that varies between 0 and 1. To tailor the aggressiveness index for discrete state space, we divided the $[0, 1]$ interval to q sub-intervals $[0, g_1], (g_1, g_2], \ldots, (g_{q-1}, 1]$. As such, the state space of the MDP is in the form of Eq. (4) with the size of $p \times q$.

$$S = \{(f_t, g_t) | f_t \in \{0, f_1, \ldots, f_p\}\ and\ g_t \in \{0, g_1, \ldots, g_q\}\} \quad (4)$$

Choosing a suitable value for p and q is a critical task. A small p or q shrinks the state space and positively affects the convergence rate; however, larger quantization intervals disturb the transparency of the changes in the system to the reward function. Reward function uses the state of the system as a decision criterion to reward or penalize the latest $(state, action)$ pair. Our extensive simulations and testbed experimentation show that choosing $3 \leq q \leq 4$ and $3 \leq p \leq 4$ provides the Q-learning TCP with enough number of states to significantly increase the fairness index of the network and convergence rate.

Fairness index obviously is a good indicator of how well the Q-learning TCP is doing in terms of enhancing the fairness. However, fairness index is not enough to make a decision on fairness of the TCP source. Therefore, we define aggressiveness index to indicate if a TCP source is fair to other TCP sources or not. The aggressiveness index calculates the share of the TCP source located on a specific node in the outgoing throughput of the node. A high-aggressiveness index along with a low fairness index in a TCP source triggers the learning agent to make changes to the TCP parameters and to force the TCP source to be more hospitable towards other flows. The desirable state for the learning agent is the state with a high fairness index and an aggressiveness index of T_{fair}. T_{fair} is the fairness threshold of the node. The fairness threshold depends on the number of flows originating and passing through the node and the priority of each flow. As an example, if three flows are passing via a node, and two other flows are originating from the node, assuming the same priority for all five flows, the fair share of the node from network resources and the fairness threshold is $\frac{2}{5}$. Any aggressiveness index above fairness threshold is an indication of the unfair resource allocation by the TCP source on the node.

Both fairness index and aggressiveness index are calculated based on the number of packets received or transmitted in each decision epoch in the TCP source. As such, both variables are accessible in each node and there is no need to get a feedback from other nodes. Let us emphasize that the objective of our MDP is to enhance TCP fairness cooperatively and accomplish this objective via moving towards the goal state; therefore, choosing fairness index and aggressiveness index as state variables of our MDP is justified.

3.2 Action set

The Q-learning agent uses the action set to constrain TCP aggressive behavior. Findings of [1] suggested that the maximum TCP congestion window size plays a crucial role in the aggressive behavior of TCP. However, putting a static clamp on TCP congestion window size might cause an under-utilization of the network resources. Therefore, to avoid any under/over-utilization of network resources, we use a dynamic clamp on TCP congestion window size. The learning agent dynamically changes the maximum congestion window size of TCP without interfering with the congestion control mechanism via the action set. As such, TCP uses the standard congestion control mechanism along with a dynamic clamp on the congestion window to limit any aggressive window increase. The agent uses the action functions in Algorithm 1 to change TCP parameters in each decision epoch.

Algorithm 1 Q-learning action set

```
1: if action = decrease then
2:     $\delta = 50\%$
3:     decreases the TCP maximum window size by $\delta$
4: else if action = increase then
5:     increases the TCP maximum window size by $\delta$
6: else
7:     no change to maximum congestion window size
8: end if
```

The Q-learning TCP does not interfere with the congestion control mechanism of TCP, only changes the maximum TCP window size; the maximum window size can be decreased up to slow start threshold. In our algorithm, we used δ as 50% of the latest increase/decrease of the current TCP maximum window size. The above action set, which resembles a binary search behavior, serves perfectly in a dynamic environment. At the beginning of the learning process, the maximum congestion window size is set either to 65536 bytes [39] or the amount allowed by the system. The learning agent starts searching for the optimized maximum TCP window size by halving the current maximum TCP window size. As such, the search space decreases into half. The agent chooses either half randomly due to the random nature of the search algorithm in the beginning of the learning process. The agent starts swiping the search space using the *decrease*( ), *increase*( ), and *stay*( ) functions and uses the reward function as the guide to the optimum state. During the learning process, the agent develops a memory and uses its memory after convergence as a series of policies to handle any changes in the dynamics of the system. As a result, in any state, the learning agent knows how to find its way to the optimum clamp. The Q-learning agent converges when all the available action series and their associated reward are discovered.

3.3 Transition probabilities

Another element of the MDP is the transition probabilities which are unknown in our design. When the transition probabilities of an MDP are unknown, RL methods such as Q-learning are used to calculate the transition probabilities. $P(s_t|s_{t-1}, a)$ is the transition probability from state s_{t-1} to state s_t taking action a. In an MDP, the states are Markovian and Eq. (5) holds.

$$p(s_t|s_{t-1}, s_{t-2}, \ldots, s_0, a) = p(s_t|s_{t-1}, a) \tag{5}$$

In our proposed model, the state space is in the form of $S = (\textit{fairness index}, \textit{aggressiveness index})$. Both fairness and aggressiveness indices depend on the number of packets transmitted or received in the most recent decision epoch. Therefore, any state transition only depends on the latest state of the system. As such, all states are Markovian and Eq. (5) holds for our model.

3.4 Reward function

According to [40], an efficient reward function should have the following conditions:

- the reward function must have a uniform distribution for states far from the goal state.
- the reward function must point in the direction of the greatest rate of increase of reward in a zone around the goal state.

Choosing a Gaussian reward function in the form of Eq. (6) complies with the above conditions. The Gaussian function is almost uniform in states far from the goal state, and has an increasing gradient in a belt around the goal state that directs the learning agent towards the desirable state.

$$R(s'|s, a) = \beta e^{-\frac{d(s', s^*)}{2\sigma^2}} \tag{6}$$

The reward function that we use for our model is in form of Eq. (7), which is a summation of fairness reward function and network utilization reward function.

$$R(s, a) = \beta_u e^{-\frac{d(u_s, u_{s^*})^2}{2\sigma_u^2}} + \beta_f e^{-\frac{d(f_s, f_{s^*})^2}{2\sigma_f^2}} \tag{7}$$

u_s and u_{s^*} are the the network utilization factor in states s and s^*. Network utilization factor is the accumulative throughput of all the incoming and outgoing flows in a node. f_s and f_{s^*} are the Jain's fairness index in states s and s^*. s^* is the goal state.

There is always a trade off between fairness and the throughput/efficiency of the network. In a highly effective network, the utility function is focused on maximizing the aggregated throughput of the network which might not be optimized based on fairness. In our scheme, we optimize

TCP performance based on both fairness and throughput. To create a balance between the two, we use the aggressiveness index (throughput control factor) and fairness index (fairness control factor). The aggressive nodes have to compromise the throughput in favor of starved nodes to increase the fairness index of the network. In our mechanism, the reward function includes both fairness and throughput. One has to keep in mind that optimizing TCP performance based on both fairness and throughput results in some compromise on throughput to increase the fairness. Moreover, quality of service (QoS) can be added to our scheme via the reward function. Putting different weights on either throughput or fairness via changing the coefficients in reward function ($\beta_u, \beta_f, \sigma_u, \sigma_f$) can result in different levels of QoS.

4 Integration of Q-learning agent and TCP

The Q-learning process interacts with TCP via the action set. The learning agent receives the statistics of the network in each decision epoch and based on the memory of the learning process and the state of the system, the agent selects an action. The interaction between the Q-learning agent and TCP can be best described in an example. Assume that the learning agent chooses the action series in Fig. 1 and decreases the TCP maximum window size to a specific amount.

As a result of the new clamp in each decision epoch, the learning agent receives a reward or penalty from the reward function. The learning agent tries all the possible combinations during the learning process and creates an inclusive map of the action series (polices) and their respective rewards. After the learning process converges, any changes in the network dynamics can be handled instantly using the memory map of the agent. As such, Q-learning TCP adapts the TCP parameters instantly based on the changes in traffic condition and network conditions. Our proposed mechanism is presented in Algorithm 2.

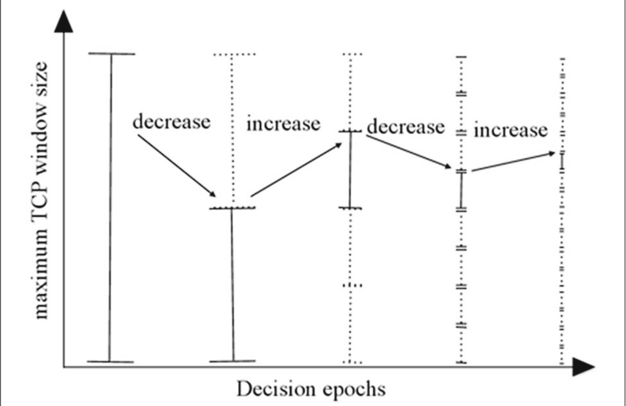

Fig. 1 An example of the Q-learning agent interaction and TCP maximum congestion window size

Algorithm 2 Q-learning TCP algorithm of node i

```
1: action set ={decrease,increase,stay}
2: absorbing $state = (f_{goal}, g_{goal})$
3: **while** not in goal state **do**
4:     **for** every t seconds **do**
5:         get the number of sent packet by node $i$
6:         **for** each flow being forwarded by node $i$ **do**
7:             get the number of sent packets
8:         **end for**
9:         calculate the fairness index $f_t$, based on (2)
10:        calculate the aggressiveness index $g_t$, based on (3)
11:        determine the state according to (4)
12:        calculate the reward $r_t$ based on (7)
13:        update the $Q$ matrix according to (1)
14:        choose the action with maximum $Q$ value
15:        take the action (inform TCP)
16:    **end for**
17: **end while**
```

As depicted in Algorithm 2, the MAC layer collects the number of sent packet of each flow passing through the node and sends them to the learning agent which is located in TCP source every t seconds (t is the length of decision epoch). The learning agent uses the latest information to calculate the aggressiveness and fairness index and determines the current state of the system. The reward function module uses the current state of the system, previous state of the system, and the latest action to calculate the immediate reward of the agent based on the latest $(state, action)$ pair. When the agent obtains the immediate reward, it updates the Q-matrix based on the Ballard equation, Eq. (1). Finally, the agent chooses an action based on the recent Q-matrix and informs TCP to adjust its maximum window size based on the chosen action. We are using a delayed reward system because the agent has to wait for the system to settle down in a specific state to figure out the instant reward.

5 Performance evaluations

Q-learning TCP is specifically aimed toward a wireless mesh network environment; as such, all the simulations and testbed are designed to present the interaction of TCP and the unique characteristics of the wireless mesh setting. In this section, we present the numerical results of our proposed method and demonstrate the effectiveness of our fairness mechanism compared to TCP, TCP-AP, and TCP ex Machina.

We first evaluate the performance of the Q-learning TCP in a multi-hop setting in which all the nodes are located in the same wireless domain and participate in the optimization process in Section 5.1., Section 5.3 presents the performance of Q-learning TCP over a testbed with real data in an office environment.

5.1 Chain opology

The chain topology is a great scenario to evaluate the effectiveness of Q-learning TCP over WMN, because it can create a competitive environment for flows with different number of hops which is the main feature of each wireless mesh topology. We use Jain's fairness index and the flow throughput as the comparison metric parameters. We set up a multi-hop network of three nodes located in a chain topology with 150 m spacing between neighboring nodes, Fig. 2. Each node is equipped with a 802.11b network interface. Nodes 1 and 2 are equipped with an FTP source that transmits packets to node 3. The transmission range is around 250 meters, and the carrier sensing range is around 550 m for each wireless interface. The data rate for IEEE 802.11b is 2 Mbps and the RTS/CTS mechanism is on. Each TCP packet carries 1024 bytes of application data.

For the Q-learning scheme, we have to determine the state space of the system and the reward function. As mentioned in Section 3, the fairness index and aggressiveness index are values between 0 and 1. For fairness index, we divided the $[0,1]$ interval into four sub-intervals, $f = \{[0,0.5), [0.5,0.8), [0.8,0.95), [0.95,1]\}$. For aggressiveness index, we split $[0,1]$ interval into three sub-intervals, $g = \{[0,0.5), [0.5,0.8), [0.8,1]\}$. We can split both fairness index and aggressiveness index into smaller sub-intervals and provide more control for the learning agent to tune TCP parameters for more desirable results. Although having smaller intervals facilitates the learning agent decision making, an increase in number of intervals increases the state space size and slows down the learning process convergence rate. After the quantization, the state space reduces to (8):

$$s = \{(f,g)|f = \{0, 0.5, 0.8, 0.95\}, g = \{0, 0.5, 0.8\}\} \quad (8)$$

The above state space provides each node in the network with a realistic understanding of the resource allocation of the neighboring nodes. The immediate reward function that we use for our Q-learning TCP is in the form of Eq. (7). We run each simulation for 30 times for 95% confidence interval. The length of each simulation is 1000 s.

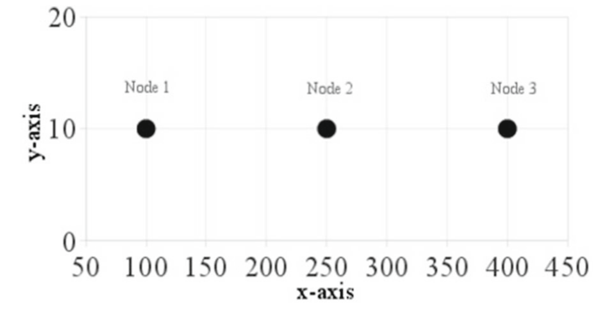

Fig. 2 Chain topology – three nodes, two flows

Figure 3 shows the throughput changes during the learning process. At the beginning of the learning process, the Q values are all zero; therefore, the agent starts a systematic search to determine the effect of each action on the state of the system. Because we choose a Gaussian reward function, the Q-learning agent gradually moves to states adjacent to the goal state. The systematic search behavior exists during the simulation, but the range of the search circle diminishes as the learning process converges to the goal state. As depicted in Fig. 3, at the beginning of the learning process, the throughput of both flows fluctuates. As time goes by, the fluctuation of both flows dwindles to negligible amount. As the learning process progresses, the agent visits each state sufficient number of times to find the best policy (the best maximum TCP window size) to maximize its acquired rewards. Eventually the learning agent in node 2 converges to the $s = (0.95, 0.5)$, where 0.95 is the fairness index and 0.5 is the aggressiveness index of node 2.

To investigate the convergence of the learning algorithm, we calculated the average learning rate, Fig. 5. The average learning rate of the process is calculated as $\frac{1}{(E\{n(s,a)\})}$, where $E\{n(s,a)\}$ is the average number of times each $(state, action)$ pair is visited by the agent. According to [36], a deceasing average learning rate is an indication of the Q-learning convergence process.

Figure 5 shows that the learning rate approaches 0 which guarantees the convergence of the learning algorithm. Figure 6 shows the changes of network utilization factor as the learning progress. As depicted in Fig. 6, the network utility factor fluctuates widely at the beginning of the learning process. However, the network utility factor settles to a value within the desirable range when the learning process converges. The same pattern can be seen in Jain's fairness index of the network, as demonstrated in Fig. 4. The wide fluctuations of the fairness index are visible during the systematic search of the learning agent in the beginning of the learning process. As the learning process converges to the desirable state, the Jain's fairness index of the network settles and the fluctuations become negligible.

We compare our scheme with TCP-AP [18] and TCP ex Machina [34]. We implemented TCP-AP based on the algorithm in [18]. In TCP-AP, the sending rate is limited based on the changes in RTT.

Figure 7 graphs the performance of TCP-AP, TCP, Q-learning TCP, and TCP ex Machina. TCP-AP performs closely to our learning method in terms of the

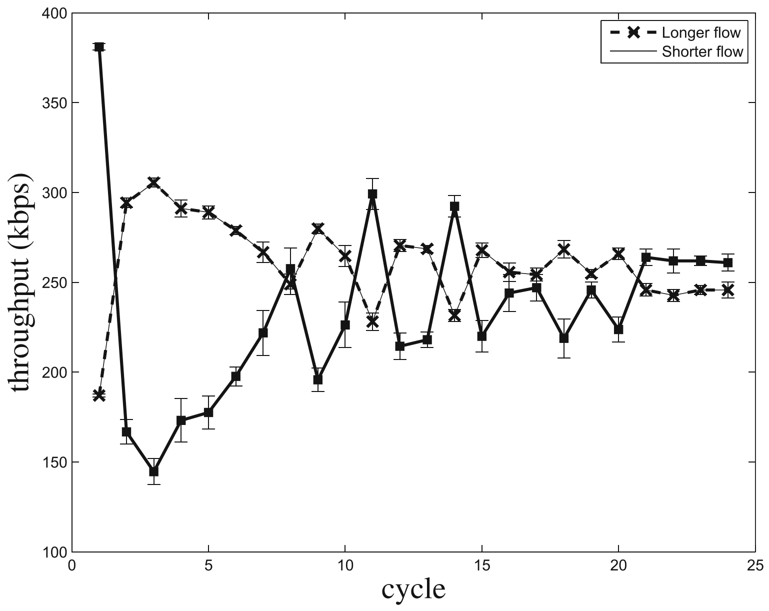

Fig. 3 Throughput changes of the flows in the chain topology of Fig. 2. Each cycle is equivalent of 40 s

fairness index of the network. However, the fair resource allocation in TCP-AP comes at the cost of network utility. The drop in the network utility factor in TCP-AP is caused by the frequent pauses in data transmission. TCP ex Machina, on the other hand, performs similar to standard TCP; the reason behind this behavior of TCP ex Machina over the multi-hop wireless setting is that the learning mechanism in TCP ex Machina is optimized for wired settings and the re-learning process requires a great deal of computational resources which almost is impossible to be done on the current wireless nodes within a reasonable amount of time. Table 1 represents the comparison of Q-learning with TCP-AP and TCP ex Machina.

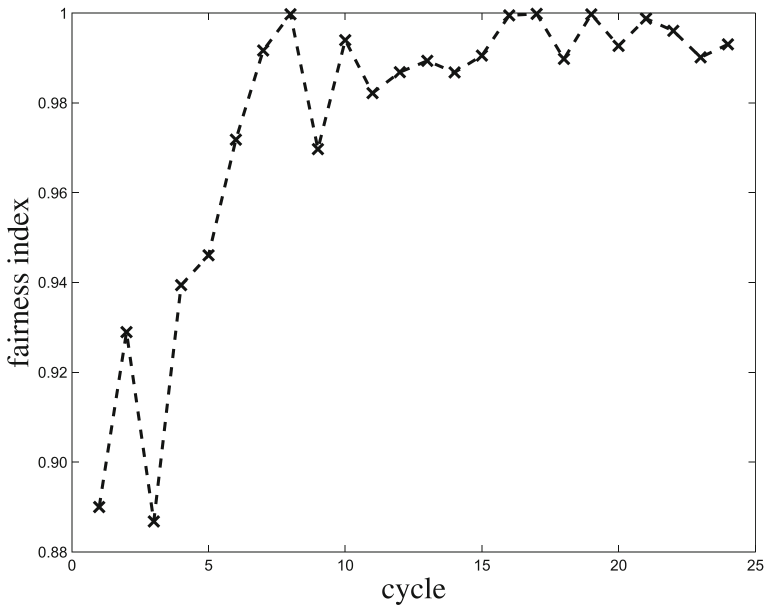

Fig. 4 Jain's fairness index in the chain topology of Fig. 2. Each cycle is equivalent of 40 s

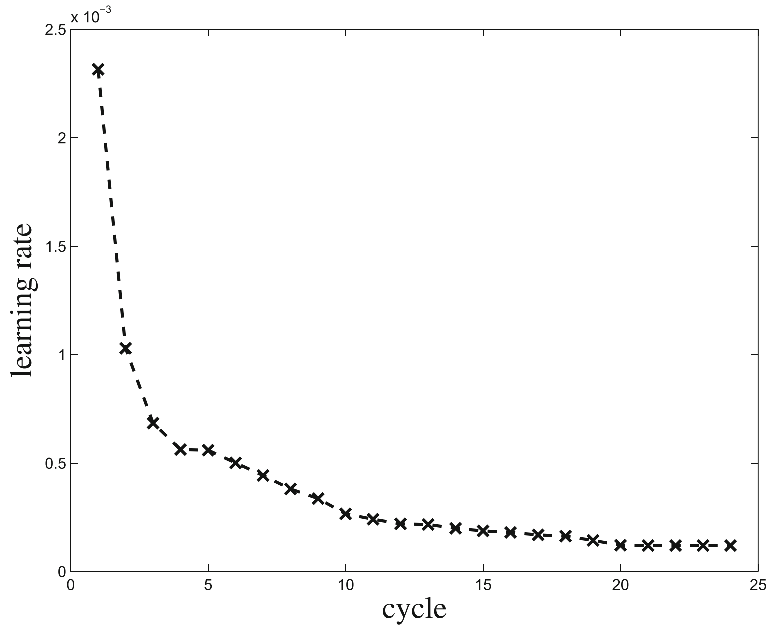

Fig. 5 Learning rate (proof of convergence) in topology of Fig. 2. Each learning cycle consists of 40 s

5.2 Larger scale WMN

To evaluate our proposed algorithm further with non-static traffic pattern and a random mesh topology, we generate a random topology in ns2, as illustrated in Fig. 8. There exist six flows in the network; all flows are destined towards node 0 and are originated from nodes 8, 3, 5, 7, 1, and 4. To study the performance of Q-learning TCP under non-static data traffic, we programmed all the sources to generate data for random length intervals and intermittently.

As depicted in Fig. 9, the resource allocation in legacy TCP is severely unfair as nodes 8, 1, and 4 are starving while nodes 3 and 7 aggressively consume the bandwidth. However, the Q-learning TCP pushes node 3 to

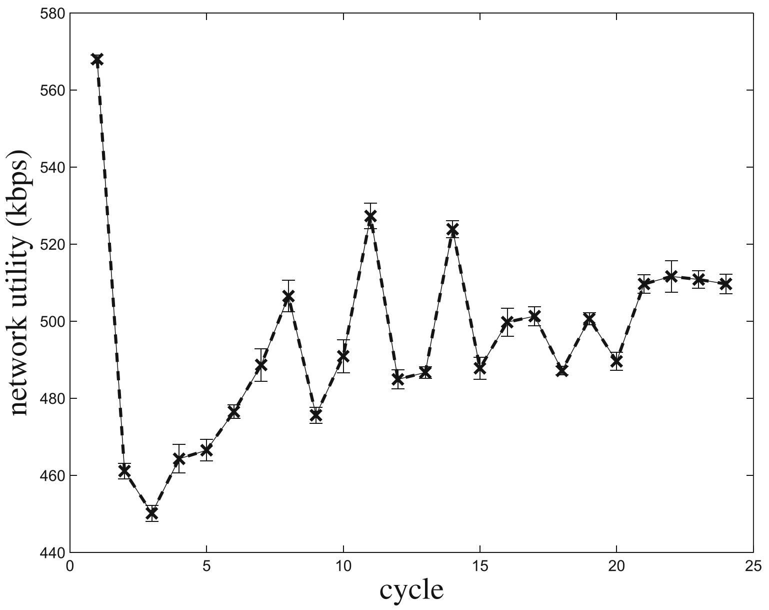

Fig. 6 Network utilization factor (kbytes/s)in topology of Fig. 2. Each learning cycle consists of 40 s

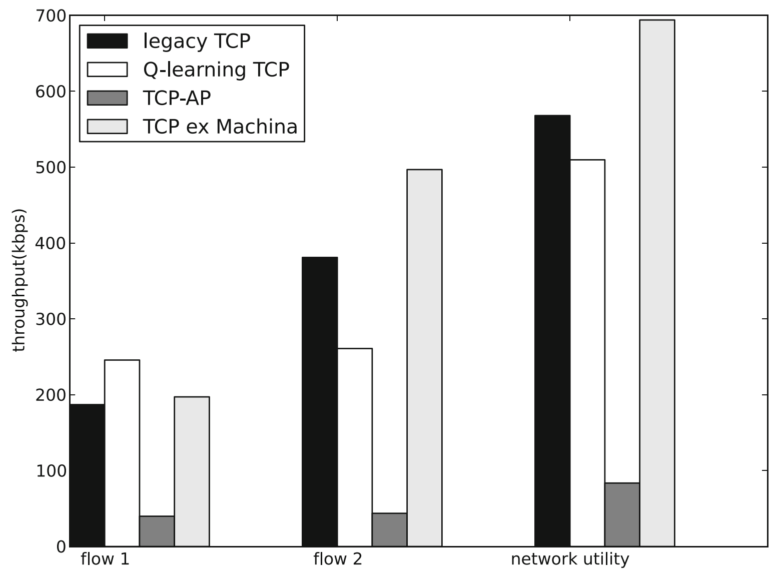

Fig. 7 Performance comparison between legacy TCP, Q-learning TCP, TCP-AP, and TCP ex Machina

decrease its network share and provide more transmission chance for other nodes. The learning agent of node 5 also indicates an unfair resource allocation and forces node 5 to slow down the data sending rate. On the other hand, node 8 experiences an undergrowth of the congestion window size and starts to increase the sending rate as node 5 decreases the sending rate. Node 3 and node 5 cooperatively provide other nodes with more sending opportunities by decreasing the sending rate which results in an increase in node 8 sending rate. On the other side of the network, node 7 consumes a bigger portion of the bandwidth compared to node 1; therefore, the Q-learning TCP forces node 7 to decrease its sending rate and provide other nodes with more sending opportunities. A comparison of TCP, TCP-AP, TCP ex Machina and Q-learning TCP is presented in Table 2.

TCP-AP outperforms both TCP and Q-learning TCP in fair resource allocation in the scenario of Fig. 8. However, the high fairness index of TCP-AP comes at the cost of drastic decrease in network utilization by a factor of 50%. The reason behind the extreme decrease in network utility factor in TCP-AP is the over-estimation of RTTs and excessive overhead caused by feedback messages. TCP ex Machina performs as poor as legacy TCP and causes one of the flows to starve completely.

To investigate the convergence of the learning process, we graphed the learning rate of the learning process for nodes 3, 5, and 7 in Fig. 10. The agent learning rate for all three nodes converges to 0 as the learning process progresses. The learning rate of node 3 is higher that the two other nodes, the reason for this behavior is that node 3 has to make more changes to get to the optimum state. More changes translate into more state transitions and consequently a higher convergence rate.

5.3 Testbed

To evaluate the performance of Q-learning TCP in real world setting, we set up a testbed in a real office environment along with other network users in the office, Fig. 11. The blue nodes in Fig. 11 are the employees with their laptop (MacBook Pro or MacBook Air) who connect to the Internet via Router R1 (extender) or R2. Routers R1 and R2 connect to the Internet through the gateway (purple node). We add node B which is both a source and forwarder node. Node B can forward to R1 and R2; moreover, we set up a static routing table inside node B that forwards

Table 1 Network metrics parameters for different TCP variations of Fig. 2

TCP variation	Jain's fairness index	Network utility
Legacy TCP	89%	509 Kbps
TCP-AP	99%	83 Kbps
TCP ex Machina	84%	694 Kbps
Q-learning TCP	99%	468 Kbps

Table 2 Network metrics parameters for different TCP variations of scenario 8

TCP variation	Jain's fairness index	Network utility
Legacy TCP	75%	430 Kbps
TCP-AP	97%	240 Kbps
TCP ex Machina	71%	436 Kbps
Q-learning TCP	83%	403 Kbps

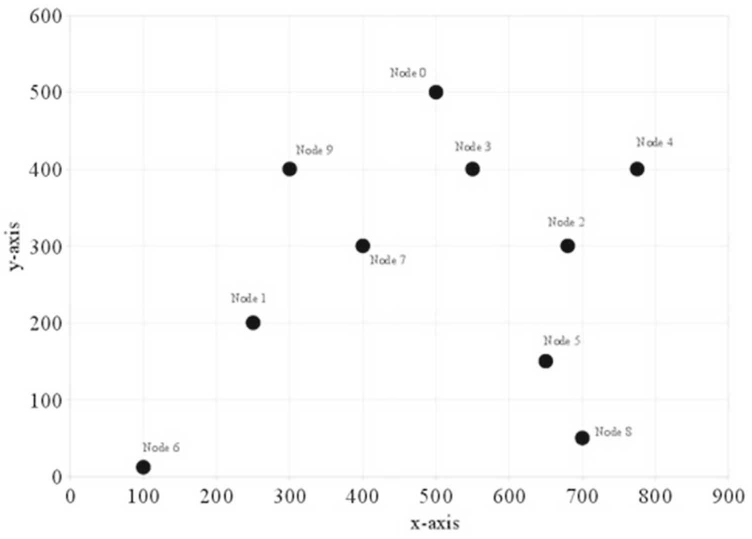

Fig. 8 Random network topology – 10 nodes

packets from node A to node C without forwarding them to R1 or R2. We implement the Q-learning mechanism as a python suite based on Algorithm 2 in node B and node A. The Q-learning mechanism communicates with the transport layer via action functions in Section 3.2 in order to tune TCP maximum congestion window size. While all the users over the network continue their day to day activity.

For node A, B, and C, we use raspberry pi as wireless nodes in our testbed. Raspberry pi [41] is a tiny affordable computer that can host multiple wired/wireless interfaces and can be used either as a client node, a server node, a router, or a forwarder node based on the available application. We use FreeBSD [42] on client and server nodes (nodes A and C) and Wheezy [43], another flavor of Linux, on nodes acting as router (node B). FreeBSD only supports WiFi adapters in client mode and not host or forwarder mode. As such, we use Wheezy for nodes in the middle that has to play a role in forwarding/routing packets. We use TP-LINK nano USB adapters as *802.11 g* WiFi interface [44], as they are cheap and they work well with FreeBSD and Wheezy.

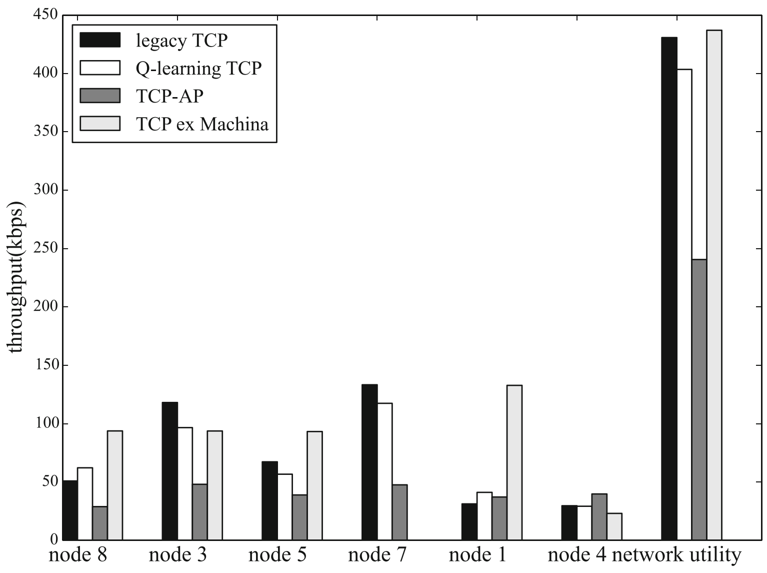

Fig. 9 TCP flow throughput and network utility in kbytes/s for scenario of Fig. 8

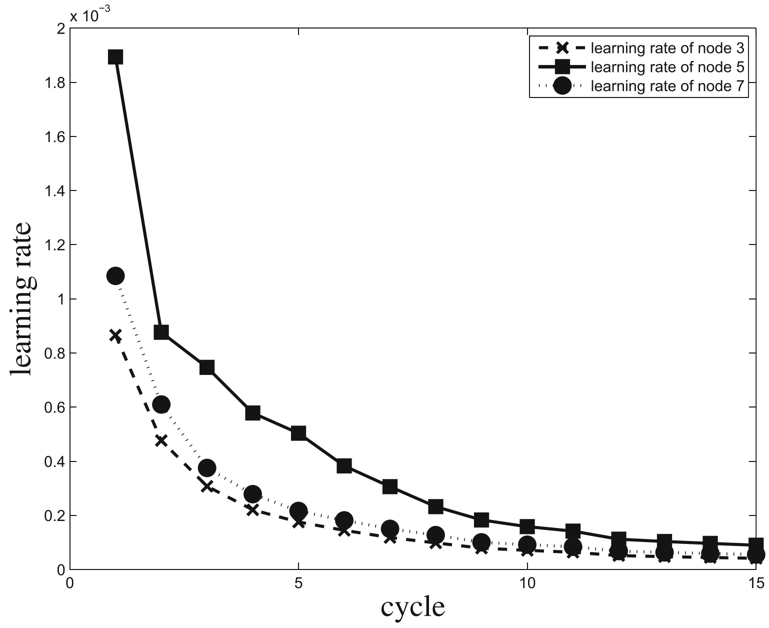

Fig. 10 Learning rate of nodes 3, 5, and 7 in scenario of Fig. 8

To generate real-world traffic profile over the testbed, we use findings of Brownlee et al. in [45]. According to measurements of [45] over two real-life networks, Internet streams can be classified into two groups: dragonflies and tortoises. Dragonflies are session with lifetime of less than 2 s while the tortoises are the sessions with long lifetimes, normally over 15 min. Authors of [45] showed that although the lifetime of sessions are variable; the lifetime distribution shape is the same for both Tortoise and dragonflies and it does not not experience rapid changes over time. Findings of [45] are critical to the design of Q-learning TCP and its interaction in real world; the fact that the distribution of the lifetime of the streams does not change rapidly over time fits well with the characteristics of Q-learning. Based on [45], 45% of the streams have a lifetime of less than 2 s, 53% have a lifetime between 2 s and 15 min, and the rest has life times more than 15 min (usually in the order of hours). We use these findings to generate traffic on node A and B along with the other existing day-to-day traffic within the office network.

Table 3 shows the result of our measurement over the testbed of Fig. 11. TCP Q-learning outperforms TCP Reno on node A with the big margin of 85% of

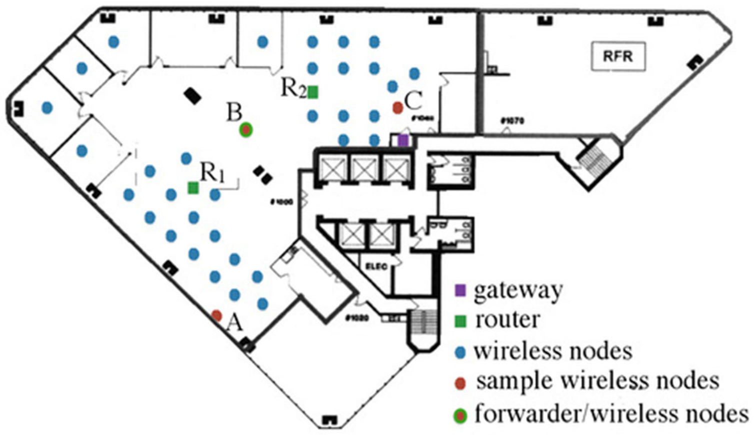

Fig. 11 Testbed topology

Table 3 A comparison between Q-learning TCP and TCP Reno

TCP variation	Source	Throughput (Kbits/s)	95% Confidence interval (Kbits/s)
TCP Reno	node A	883	less than 100
TCP Reno	node B	4003	less than 400
Q-learning TCP	node A	1549	less than 400
Q-learning TCP	node B	3877	less than 1000

increase in the average throughput. Node B which acts as a source and a forwarder node has to compromise its throughput to enhance the fairness of the network. The average throughput of node B decreases by 3% to accommodate a 85% boost in throughput of node A which is a drastic increase with a minimal compromise.

The larger confidence interval in Q-leanring TCP is caused by the changes of the maximum congestion window size according to the optimal policy of Q-learning during the discovery. As a side note, the initial convergence time for the testbed is a few hours; however, after the convergence, adjustment to any changes in the network is almost immediate as the agent has already discovered all the possible states.

6 Discussion and comparison

To propose an effective fairness solution for TCP over wireless multi-hop networks one has to consider the dynamic nature of the environment as an important design factor. Meaning that a dynamic solution that changes strategy based on the network condition is required. Therefore, the effective solution requires two characteristics: (a) monitoring/learning network conditions (b) choosing the correct strategy based on the perceived condition. Reinforcement learning methods meet the design characteristics in (a) and (b). Among various reinforcement learning methods, Q-learning fits our needs the best, as it is a model-free technique. Meaning that Q-learning can be used to find an optimal strategy-selection policy for any given finite state Markov decision process. The above reasoning justifies our choice of fairness solution for TCP over wireless multi-hop settings.

The main purpose of Q-learning in this paper is to decide which kind of action to take in the next time slot, i.e., increase, decrease or maintain the maximum congestion window size. Note that the best application of Q-learning is to learn action-reward functions for stationary settings, which can be proved to converge. It is true that Q-learning can still get results in non-stationary environment, such as wireless settings, but the Q-learning agent will take more time to be aware of the changes. Due to the time-varying network conditions, sometimes rapidly, the stationary assumption cannot always hold and it can make Q-learning less suitable for wireless networks. However, there are ways to make sure that the convergence rate stays within an acceptable range for dynamic environments. Using a learning rate of $\alpha = \frac{1}{(1+t)^2}$ brings down the convergence rate to a polynomial in $\frac{1}{(1-\gamma)}$, where γ is the discount factor [38]. We use a learning rate of $\alpha = \frac{1}{(1+t)^2}$ to ensure that our proposal complies with the dynamic nature of the environment. We have to emphasize that once the Q-learning TCP converges, the agent does not need to re-learn the environment as it has already discovered all the state-action pairs and their associated rewards and can cope with any changes in the environment. In the event of a drastic change in the environment, a fast convergence rate helps the Q-learning to adjust its memory fairly quickly.

Another consideration while using Q-learning for any scenario is the computational overhead. Assuming that in a specific scenario action a_i alleviates the aggressive behavior of a specific flow while keeping the throughput at its maximum possible, and τ iteration is needed before the algorithm converges. Then the overhead of the action to the learning node is:

$$Overhead = \frac{1}{n-1} \Sigma_{t=1}^{\tau} \Sigma_{i=1, i \neq j}^{n} P_i(t) O(a_i) \tag{9}$$

where $P_i(t)$ is the probability of choosing action i at iteration t and depends on the values in the Q matrix, and the reward function. $O(a_i)$ is the overhead of performing action a_i and τ is the convergence time. n is the number of states in the underlying MDP. Based on [38], in a Q-learning scenario with a polynomial learning rate, the convergence time τ depends on covering time L. Covering time indicates the number of iterations needed to visit all action-state pairs with the probability of at least 0.5 starting from any pair. The convergence time τ is in the order of $\Omega\left(L^{2+\frac{1}{\omega}} + L^{\frac{1}{1-\omega}}\right)$ with the smallest amount at $\omega = 0.77$. In our fairness mechanism, we have limited number of states and in each state the agent has three actions to choose from; as such, both covering time and convergence time are tractable in our mechanism.

Comparing the Q-learning TCP with other well-known existing fairness methods [8, 46, 47], the Q-learning TCP does not incur any overhead to the network with the expense of extra computation at each node. The focus of LRED [8] is on enhancing TCP throughput over WMN and fairness enhancement is not one of the design objectives. However, the pacing mechanism of LRED enhances the fairness as a side-effect at the cost of excessive transmission delay. The extra transmission delay in LRED pacing mechanism alleviates the hidden terminal issue; however, the imposed delay is fixed in size and is not adjustable to the dynamic nature of the WMN. NRED uses a dropping mechanism to decrease the competition and provide more resources for the starved flows. However, the dropping probabilities are calculated and broadcasted constantly which incurs a heavy overhead to the shared medium. TCP-AP uses the received signal strength indicator (RSSI) to infer information regarding the hidden terminal issue; however, RSSI causes an over-estimation of transmission delay in its pacing mechanism and decreases the TCP throughput drastically. As such, TCP-AP still requires feedback messaging from the neighboring nodes for hidden terminal distance calculations. TCP ex Machina, another comparable mechanism, requires excessive computational resources for its congestion control mechanism optimization which is not compatible with current network infrastructure resources. Our method achieves the fairness enhancement of TCP at a cost of reasonable extra computation of the machine learning approach in each node. In WMN that the shared medium is extremely valuable, flooding the network with excessive feedback messages or under-utilizing the links with excessive non-dynamic transmission delays to enhance the fairness is not very cost efficient. However, Q-learning TCP trades the computational simplicity in each node for TCP fairness. In a mesh setting, since the mobility of each node is very minimal, increasing the computational capacity of the nodes is not very costly. A brief comparison of LRED, NRED, TCP-AP, TCP ex Machina, and Q-learning TCP is presented in Table 4.

It is noteworthy to mention that the complexity of Q-learning TCP is polynomial with the number of states and the convergence rate is tractable with a suitable state space size as confirmed with the simulation and testbed experiments.

Our findings confirm that in a wireless multi-hop setting, each TCP source has to cooperate with others to ensure a fair share of network resources for all end-users. TCP allocates resources with the assumption of an inclusive knowledge of the network topology and a reliable permanent access to the medium for all the end-users. However, in a wireless mesh setting, the end-user knowledge of the network topology is partial and the access to the medium is intermittent. As such, TCP needs to collect information about other nodes to compensate for the short comings of the underneath layers. The learning agent provides the TCP source with insight on existing competition for network resources from other nodes. The insight provided by the learning agent compensate for the unfair behavior of the MAC and TCP in the wireless multi-hop environment by suppressing the aggressive response of TCP. It is noteworthy that Q-learning TCP inter-works well with any variation of TCP on the other end-point because the changes to the TCP protocol stack are only in the sender side and the learning mechanism does not need any feedback from the receiver. The Q-learning TCP only relies on the information collected by the learning agent.

7 Conclusions

We have proposed a cross-layer monitoring and learning mechanism for fair resource allocation of TCP over WMN that uses the information obtained from the MAC layer to optimize TCP parameter to enhance the end to end fairness within a network. The learning agent uses the local fairness index and the aggressiveness index of each node to decide if the node is starving or abusing the network resources. We have used a reward function to guide the learning agent in taking correct actions which eventually allows us to solve the fairness problem in a distributed manner. We have compared our learning method with legacy TCP and TCP-AP, and TCP ex Machina via extensive ns2 simulations. Simulation results have demonstrated the superiority of our proposed method within wireless network. Moreover, we have studied the performance of Q-learning TCP in a testbed. Testbed measurements have proved that Q-learning TCP can be a great candidate for transport protocol over current wireless multi-hop networks with minimal changes only in TCP source.

Table 4 A comparison between Q-learning TCP and other well-known TCP solutions

Fairness solution	Fairness enhancement	Throughput enhancement	Disadvantage
LRED [8]	Slight increase	5 to 30% increase	Overhead caused by broadcast messages and fixed transmission delay
NRED [46]	Effective increase (Jain's fairness index of 99% in a chain topology)	Up to 12% increase	Excessive overhead caused by broadcast messages (over 60%)
TCP-AP [47]	Effective increase (Jain's fairness index of 99% in a chain topology)	Drastic decrease (up to 50%)	Reliance on RSSI and excessive transmission delay
TCP exMachina [47]	Decrease (Jain's fairness index of 84% in a chain topology)	Slight increase	Excessive learning time and computational resource requirement
Q-learning TCP	Effective increase (Jain's fairness index of 99% in a chain topology)	Slight decrease	Medium computational overhead

Competing interests
This work was supported in part by the Canadian Natural Sciences and Engineering Research Council (Grants RGPIN-2014-06119 and RGPAS-462031-2014), and by the National Natural Science Foundation of China (Grant 61671088).

Received: 18 April 2016 Accepted: 11 November 2016
Published online: 05 December 2016

References

1. M Franceschinis, *et al*, Measuring tcp over wifi: a real case. 1st Workshop on Wireless Network Measurements (Winmee), Riva Del Garda, Italy (2005)
2. B Francis, V Narasimhan, A Nayak, I Stojmenovic, in *32nd International Conference on Distributed Computing Systems Workshops (ICDCSW 2012)*. Techniques for enhancing TCP performance in wireless networks, (2012), pp. 222–230
3. G Jakllari, S Eidenbenz, N Hengartner, SV Krishnamurthy, M Faloutsos, Link positions matter: a noncommutative routing metric for wireless mesh networks. IEEE Trans. Mob. Comput. **11**(1), 61–72 (2012)
4. DJ Leith, Q Cao, VG Subramanian, Max-min Fairness in 802.11 Mesh Networks. IEEE/ACM Trans. Networking. **20**(3), 756–769 (2012)
5. XM Zhang, WB Zhu, NN Li, DK Sung, TCP congestion window adaptation through contention detection in ad hoc networks. IEEE Trans. Veh. Technol. **59**(9), 4578–4588 (2010)
6. J Karlsson, A Kassler, A Brunstrom, in *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks Workshops, (WoWMoM 2009)*. Impact of packet aggregation on TCP performance in wireless mesh networks, (2009), pp. 1–7
7. R De Oliveira, T Braun, A smart TCP acknowledgment approach for multihop wireless networks. IEEE Trans. Mob. Comput. **6**(2), 192–205 (2007)
8. Z Fu, H Luo, P Zerfos, S Lu, L Zhang, M Gerla, The impact of multihop wireless channel on TCP performance. IEEE Trans. Mob. Comput. **4**(2), 209–221 (2005)
9. K Nahm, A Helmy, C-C Jay Kuo, in *Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing. MobiHoc '05*. TCP over multihop 802.11 networks: issues and performance enhancement (ACM, New York, NY, USA, 2005), pp. 277–287
10. Y Su, P Steenkiste, T Gross, in *16th International Workshop on Quality of Service (IWQoS 2008)*. Performance of TCP in multi-Hop access networks, (2008), pp. 181–190
11. IF Akyildiz, X Wang, W Wang, Wireless mesh networks: a survey. ELSEVIER Comput. Netw. **47**(4), 445–487 (2005)
12. A Al-Jubari, M Othman, B Mohd Ali, N Abdul Hamid, An adaptive delayed acknowledgment strategy to improve TCP performance in multi-hop wireless networks. Wirel. Pers. Commun. **69**(1), 307–333 (2013)
13. CY Huang, P Ramanathan, Network layer support for gigabit tcp flows in wireless mesh networks. IEEE Trans. Mob. Comput. **14**(10), 2073–2085 (2015)
14. Y Cai, S Jiang, Q Guan, FR Yu, Decoupling congestion control from tcp (semi-tcp) for multi-hop wireless networks. EURASIP J. Wirel. Commun. Netw. **2013**(1), 1–14 (2013)
15. K Xu, M Gerla, L Qi, Y Shu, in *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking (ACM MobiCom 2003)*. Enhancing TCP fairness in ad hoc wireless networks using neighborhood RED (ACM, New York, NY, USA, 2003), pp. 16–28
16. J Ye, J-X Wang, J-W Huang, A cross-layer TCP for providing fairness in wireless mesh networks. Int. J. Commun. Syst. **24**(12), 1611–1626 (2011)
17. A Raniwala, D Pradipta, S Sharma, in *26th IEEE International Conference on Computer Communications (INFOCOM 2007)*. End-to-end flow fairness over IEEE 802.11-based wireless mesh networks, (2007), pp. 2361–2365
18. SM ElRakabawy, C Lindemann, A practical adaptive pacing scheme for TCP in multihop wireless networks. IEEE/ACM Trans. Networking. **19**(4), 975–988 (2011)
19. H Xie, A Boukerche, AAF Loureiro, in *IEEE International Conference on Communications (ICC 2013)*. TCP-ETX: A cross layer path metric for TCP optimization in wireless networks, (2013), pp. 3597–3601
20. B Vamanan, J Hasan, T Vijaykumar, Deadline-aware datacenter TCP. ACM SIGCOMM Comput. Commun. Rev. **42**(4) (2012)
21. N Arianpoo, P Jokar, VCM Leung, in *International Conference on Computing, Networking and Communications (ICNC 2012)*. Enhancing TCP performance in wireless mesh networks by cross layer design, (2012), pp. 177–181
22. V Gambiroza, B Sadeghi, EW Knightly, in *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*. MobiCom '04. End-to-end performance and fairness in multihop wireless backhaul networks (ACM, New York, NY, USA, 2004), pp. 287–301
23. S Shioda, H Iijima, T Nakamura, S Sakata, Y Hirano, T Murase, in *Proceedings of the 5th ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks*. PM2HW2N '10. ACK pushout to achieve TCP fairness under the existence of bandwidth asymmetry, (2010), pp. 39–47
24. C Cicconetti, IF Akyildiz, L Lenzini, FEBA: A bandwidth allocation algorithm for service differentiation in IEEE 802.16 Mesh Networks. IEEE Trans. Netw. **17**(3), 884–897 (2009)
25. C Cicconetti, IF Akyildiz, L Lenzini, in *Proceedings of 26th IEEE International Conference on Computer Communications (ICC 2007)*. Bandwidth balancing in multi-channel IEEE 802.16 wireless mesh networks, (2007), pp. 2108–2116
26. T-C Hou, C-W Hsu, C-S Wu, A delay-based transport layer mechanism for fair TCP throughput over 802.11 multihop wireless mesh networks. Int. J. Commun. Syst. **24**(8), 1015–1032 (2011)
27. S Fowler, M Eberhard, K Blow, Implementing an adaptive TCP fairness while exploiting 802.11e over wireless mesh networks. Int. J. Pervasive Comput. Commun. **5**, 272–294 (2009)
28. T Li, DJ Leith, V Badarla, D Malone, Q Cao, Achieving end-to-end fairness in 802.11e based wireless multi-hop mesh networks without coordination. Mob. Networks Appl. **16**(1), 17–34 (2011)
29. K Xu, N Ansari, Stability and fairness of rate estimation-based AIAD congestion control in TCP. IEEE Commun. Lett. **9**(4), 378–380 (2005)
30. KLE Law, W-C Hung, Engineering TCP transmission and retransmission mechanisms for wireless networks. Pervasive Mob. Comput. **7**(5), 627–639 (2011)
31. S Ha, I Rhee, L Xu, Cubic: a new tcp-friendly high-speed tcp variant. ACM Spec. Interes. Group Oper. Syst. Rev. (ACM SIGOPS 2008). **42**(5), 64–74 (2008)
32. CP Fu, SC Liew, TCP Veno: TCP enhancement for transmission over wireless access networks. IEEE J. Sel. Areas Commun. **21**(2), 216–228 (2003)
33. K Winstein, A Sivaraman, H Balakrishnan, in *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. Stochastic forecasts achieve high throughput and low delay over cellular networks, (2013), pp. 459–471
34. K Winstein, H Balakrishnan, in *ACM Special Interest Group on Data Communication Conference 2013 (SIGCOMM 2013)*. TCP ex machina: computer-generated congestion control, (2013), pp. 123–134
35. Q Jabeen, F Khan, S Khan, MA Jan, Performance improvement in multihop wireless mobile adhoc networks. J. Appl. Environ. Biol. Sci. (JAEBS). **6**, 82–92 (2016)
36. CCH Watkins, P Dayan, Technical note: Q-Learning. Mach. Learn. **8**(3–4), 279–292 (1992)
37. SD Whitehead, DH Ballard, Learning to perceive and act by trial and error. Mach. Learn. **7**(1), 45–83 (1991)
38. E Even-Dar, Y Mansour, Learning rates for q-learning. J. Mach. Learn. Res. **5**, 1–25 (2004)
39. V Jacobson, R Braden, D Borman, TCP Extensions for High Performance. **RFC 1323** (1999). doi:10.17487/RFC1323. http://www.rfc-editor.org/info/rfc1323
40. L Matignon, GJ Laurent, N Le Fort-Piat, in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. Improving reinforcement learning speed for robot control, (2006), pp. 3172–3177
41. RaspberryPi: Model B. https://www.raspberrypi.org/
42. The FreeBSD Project. https://www.freebsd.org/
43. Debian Wheezy. https://www.debian.org/releases/wheezy/
44. TPLINK: 150Mbps Wireless N Nano USB adapter. http://www.tp-link.com/lk/products
45. N Brownlee, KC Claffy, Understanding internet traffic streams: dragonflies and tortoises. IEEE Commun. Mag. **40**(10), 110–117 (2002)
46. K Xu, M Gerla, L Qi, Y Shu, in *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking*. MobiCom '03. Enhancing tcp fairness in ad hoc wireless networks using neighborhood red (ACM, New York, NY, USA, 2003), pp. 16–28
47. SM ElRakabawy, C Lindemann, A practical adaptive pacing scheme for TCP in multihop wireless networks. IEEE/ACM Trans. Networking. **19**(4), 975–988 (2011)