

TCP Reno over Adaptive CSMA

CHEN, Wei

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
in
Information Engineering

©The Chinese University of Hong Kong

August 2010

The Chinese University of Hong Kong holds the copyright of this thesis. Any person(s) intending to use a part or the whole of the materials in this thesis in a proposed publication must seek copyright release from the Dean of the Graduate School.

Abstract of thesis entitled:

TCP Reno over Adaptive CSMA

Submitted by CHEN Wei

for the degree of Master of Philosophy

at The Chinese University of Hong Kong in August 2010

An interesting distributed adaptive CSMA MAC protocol, called adaptive CSMA, was proposed recently to schedule any strictly feasible achievable rates inside the capacity region. Of particular interest is the fact that the adaptive CSMA can achieve a system utility arbitrarily close to that is achievable under a central scheduler. However, a specially designed transport-layer rate controller is needed for this result. An outstanding question is whether the widely-installed TCP Reno is compatible with adaptive CSMA and can achieve the same result. The answer to this question will determine how close to practical deployment adaptive CSMA is. Our answer is yes and no. First, we observe that running TCP Reno directly over adaptive CSMA results in severe starvation problems. Effectively, its performance is no better than that of TCP Reno over legacy CSMA (IEEE 802.11), and the potentials of adaptive CSMA cannot be realized. Fortunately, we find that multi-connection TCP Reno over adaptive CSMA with active queue management can materialize the advantages of adaptive CSMA. NS-2 simulations demonstrate that our solution can alleviate starvation and achieve fair and efficient rate allocation. Multi-connection TCP can be implemented at either application or by inserting an intermediate layer. Intermediate layer implementation requires no application modification, and is compatible with today's applications making it readily deployable in networks running adaptive CSMA.

Acknowledgments

First and foremost, I am heartily thankful to my supervisor of this project, Prof. Minghua Chen and Prof. Soung Chiang Liew, whose patient encouragement, guidance and support from the initial to the final level enabled me to develop an understanding of the subject. Their sage advice, and insightful criticisms aided the writing of this thesis in innumerable ways.

Besides, I would like to thank Dr. Yue Wang whose steadfast support of this project was greatly needed and deeply appreciated.

Finally, an honorable mention goes to my families and friends for their understandings and supports on me in completing this project. Without helps of the particular that mentioned above, I would face many difficulties while doing this project.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Contributions	2
1.3	Thesis Organization	3
2	Related Work	4
2.1	Previous Work on Rate Control and link Scheduling in Wireless Networks	4
2.2	Previous Work on Multi-connection TCP	6
2.3	Previous Work on AQM	6
3	Problem Settings	7
3.1	Network Modeling	7
3.2	Capacity Region of Wireless Networks and Throughput-optimal Scheduling	9
3.3	Throughput-optimality of A-CSMA	10
3.4	TCP Reno Congestion Control Modeling	11
4	Starvation of TCP Reno over L-CSMA and A-CSMA	13
4.1	TCP Reno Starves over L-CSMA	13
4.2	TCP Reno Starves over A-CSMA	15
4.2.1	Simulations	15

4.2.2	Observations and Explanations	17
5	Analysis and Our Proposed Solution	19
5.1	Proposed Solution: Multi-connection TCP Reno Scheme	19
5.2	Implementation	25
5.3	Discussion	28
5.3.1	Achieve Arbitrary Utility	28
5.3.2	Extension to Networks with Both Wired and Wireless Links	28
5.3.3	Impact of ACK Traffic	30
5.3.4	Tradeoff between performance and overhead	31
5.3.5	Overhead of Multi-connection TCP	32
6	Simulations	37
6.1	Single-hop Wireless Networks Scenario	38
6.1.1	Fairness and Throughput	38
6.1.2	Impact of Measuring Queue Length in Number of Bytes for n-ACK	42
6.1.3	Impact of Dummy Packets	43
6.1.4	Impact of Product $k^2\beta$	45
6.1.5	Effects of Parameter β	47
6.1.6	Effects of Parameter k	49
6.1.7	Overhead of n-ACK Solution	50
6.2	Multihop Wireless Networks Scenario	52
6.3	Multihop Networks with Wireless and Wired Links Scenario	53
7	Conclusions and Future Work	56
7.1	Conclusions	56
7.2	Future Work	57
A	Explanation to Starvation of TCP Reno over A-CSMA	58

B TCP Reno over A-CSMA with AQM	60
B.1 TCP Reno starves	60
B.2 Explanation	61
Bibliography	64

List of Figures

4.1	A wireless network of four links. Its conflict graph is shown on the right hand side.	14
4.2	Throughput under 1) L-CSMA: NS-2 simulation of TCP Reno over L-CSMA, 2) A-CSMA: NS-2 simulation of TCP Reno over A-CSMA, 3) our solution: NS-2 simulation of multi-connection TCP Reno over A-CSMA with AQM, 4) optimal: optimal throughput computed from problem MP	16
4.3	Transmission aggressiveness under A-CSMA in NS-2 simulation.	18
5.1	Scheme Diagram	27
6.1	Link Conflict Graphs of single-hop network topologies.	38
6.2	Throughput under 1) L-CSMA: NS-2 simulation of TCP Reno over L-CSMA, 2) h-ACK: NS-2 simulation of multi-connection TCP Reno over A-CSMA with AQM and TCP ACKs sent back instantaneously, 3) n-ACK: NS-2 simulation of multi-connection TCP Reno over A-CSMA with AQM and TCP ACKs operated normally, 4) optimal: optimal throughput computed from problem MP	40
6.3	Simulation results for topology <i>a</i> under multi-connection TCP Reno over A-CSMA with AQM when h-ACK is used : curves of transmission aggressiveness and number of connections.	41

6.4	Simulation results for topology a under multi-connection TCP Reno over A-CSMA with AQM when n-ACK is used : curves of transmission aggressiveness and number of connections. . . .	42
6.5	Simulation throughput in topology a by n-ACK with queue length measured in number of bytes and number of packets. . .	43
6.6	Number of connections in topology a by n-ACK with queue length measured in number of bytes and number of packets. . .	43
6.7	Transmission Aggressiveness by n-ACK with queue length measured in number of bytes.	44
6.8	Number of packets in queue by n-ACK with queue length measured in number of bytes.	44
6.9	Simulation throughput in topology a by: a) h-ACK without dummy packets, b) h-ACK with dummy packets, c) n-ACK without dummy packets and d) n-ACK with dummy packets. . .	45
6.10	Throughput distribution by different k and β when the product $k^2\beta$ is fixed.	46
6.11	Cumulative packets by different β	47
6.12	Simulation throughput in topology a when: a) $k = 10$, b) $k = 15$, and c) $k = 20$	49
6.13	Transmission aggressiveness and number of queue length under different k	51
6.14	Multihop networks scenarios: (a) topology d and its associated conflict graph. (b) throughput (c) curves of transmission aggressiveness. (d) curves of number of connections.	54
6.15	Multihop networks with wired and wireless links scenario: (a) topology e (b) throughput	55
B.1	Simulation Results: TCP Reno over A-CSMA with AQM. . . .	61

List of Tables

2.1	Rate control and Scheduling Solution Classes	5
3.1	Key Notation	8
5.1	Notation Used in Computing Overhead	33
6.1	Utility gap ΔU under different schemes	39
6.2	Overall throughput under different schemes (packets/second) . .	41
6.3	Overall throughput and Utility gap under different β	48
6.4	Overall throughput and Utility gap under different k	50
6.5	Efficiency of all flows in topology a when n-ACK is used	52

Chapter 1

Introduction

1.1 Motivation

The Carrier Sense Multiple Access (CSMA) protocol is widely used in local-area networks, including Wi-Fi. As the deployment of Wi-Fi spreads, it is now common to find multiple co-located Wi-Fi networks with partially overlapping coverage. In such networks, each link can only sense a subset of other links, and characterizing the link throughput achievable by CSMA is challenging.

Refs. [28] and [15] show that the saturated link throughput achieved by CSMA can be studied via a time-reversible Markov chain and can be expressed in a product form. Further, authors in [15] showed that the product form results are insensitive to the packet length and back-off time distributions given the ratio of their means.

Based on these results on the CSMA link throughput, Jiang and Walrand [12], and later others [5, 17, 19, 21], present a distributed adaptive CSMA (A-CSMA) scheme that can achieve desired link throughput according to some system utility objective. It is shown that the distributed A-CSMA scheme can achieve almost the entire capacity region that can be achieved under a central scheduler. Given its attractiveness and impenetrability [14], an outstanding question is how close A-CSMA is to practical deployment. As originally studied in [12], specially designed transport-layer rate controllers were required to

inter-work with A-CSMA.

In view of the large installed base of TCP Reno, a question is whether TCP Reno over A-CSMA works well. We answer this question in this thesis.

1.2 Contributions

The main contributions of this thesis are as follows:

- We observe that running TCP Reno directly over A-CSMA results in severe starvation problems. Effectively, its performance is no better than that of TCP Reno over legacy CSMA (IEEE 802.11) [28] [15] [22] [25]. Our analysis indicates that the root cause is TCP Reno over A-CSMA results in system instability.
- We propose a multi-connection TCP Reno solution to inter-work with A-CSMA in a compatible manner. Although multi-connection TCP has been explored before in the context of video transmission over cellular and wired networks [6, 27], this thesis applies it to solve starvation problem in TCP Reno over A-CSMA. Our solution is provably optimal under the Network Utility Maximization (NUM) framework. Specifically, by adjusting the number of TCP Reno connections for each session, any system utility can be realized. NS-2 simulations demonstrate that this solution can address starvation problems and achieve fair and efficient rate allocation. The scheme is applicable to single-hop wireless networks, wired networks as well as multihop networks with wired and wireless links.
- Our multi-connection TCP Reno scheme can be implemented at either the transport layer or the application layer. Application-layer implementation requires no kernel modification to TCP Reno, and the solution can be readily deployed over networks with A-CSMA.

1.3 Thesis Organization

In this thesis, we aim at providing answers to the open question mentioned in the motivation section. In this work, we assume that a wireless link is associated with a fixed bandwidth. We show that the joint rate control and scheduling problem in wireless network can be formulated as a concave optimization problem defined by [12], [5]. We approach the problem by using Markov approximation technique described in the work [5] with primal-dual algorithm, which later we show that it is the same as the behavior of multiple connection TCP and A-CSMA with AQM.

The rest of the thesis is organized as follows. We present related work and background studies in Chapter 2. The problem setting and network model are given in Chapter 3. We discuss and illustrate the starvation of TCP Reno over L-CSMA and A-CSMA in Chapter 4 through an illustrative example. We then propose our solution in the Chapter 5, and evaluate its performance through NS-2 simulations in Chapter 6. Finally, Chapter 7 concludes our thesis.

Chapter 2

Related Work

In this chapter, we first classify the existing work on rate control and scheduling problem in wireless network into four classes from a layering perspective. We give a quick review of each class of solution on this problem and we find the the solution 3 which has less modification and more efficiency. We also review some work on the multiple connection TCP/TFRC and work on AQM in this chapter.

2.1 Previous Work on Rate Control and link Scheduling in Wireless Networks

Rate control (i.e., congestion control) is important for data transmission over both wired and wireless networks. Among those rate control scheme, the widely deployed one is TCP Reno currently. To put our study in context, we classify the solutions on rate control and scheduling problem into four classes, as shown in Table 2.1, where we refer to non-adaptive CSMA, i.e., the one used in IEEE 802.11 DCF MAC, as legacy CSMA (L-CSMA) and the modified TCP as new designed rate control scheme.

	L-CSMA	modified-MAC
TCP Reno	class 1	class 2 (our solution)
modified TCP	class 3	class 4

Table 2.1: Rate control and Scheduling Solution Classes

Class 1 solutions use TCP Reno over L-CSMA. There have been extensive work studying the fairness problem of this scheme [2, 15, 28]. They reveal that TCP Reno over L-CSMA suffers from severe starvation problem. The main reason is that L-CSMA is not an efficient scheduling algorithm and it can only schedule a fraction of the capacity region.

Class 3 solutions, for instance [22, 25], do not make any modifications to the widely used IEEE 802.11 DCF MAC. In contrast, they limit the design space to within the scope of transport layer. In [22], the authors propose an AIMD-based rate control protocol called WCP Wireless Control Protocol (WCP) to address the efficient and fair rate allocation over L-CSMA networks. WCP halves the rates of other flows if they are within interference range of the congested link. Hence, queues of other fast transmitting links are driven to empty, giving the congested link opportunity to transmit. However, WCP requires message passing among the interfered links. Meanwhile, each link must monitor the Round-trip time (RTT) of every flow that traverses it. EWCCP [25] is designed to be proportional-fair by periodically broadcasting the average queue information to coordinate among interference links. These periodical broadcasted message consumes the precious wireless bandwidth.

There have been many efforts belonging to class 4. Cross-layer designs based on new TCP and new MAC are adopted [4, 12, 16, 17, 19, 21]. Cross-layer designs can improve the overall system performance and make interaction between layers more transparent, but raise the hurdle to practical deployment.

We prefer class 2 solution because, first, it is a layered approach rather than a cross-layer one, reducing the implementation difficulty, secondly, widely

installed base of TCP Reno in practice makes it is preferable to avoid modifying TCP Reno.

2.2 Previous Work on Multi-connection TCP

In our solution, we apply multiple connection TCP Reno to combat starvation problem. There has been work on using multiple connections to transfer data. For example, multiple TFRC connections for streaming video [7] and multiple TCP Reno connections for data and multimedia streaming [6, 27] have been discussed. These works focus on wired networks and wireless cellular networks. In contrast, our work focuses on CSMA networks and therefore solves a different set of challenges, including the spatial wireless inference that was not considered in [6, 7, 27].

2.3 Previous Work on AQM

Our solution also utilizes active queue management (AQM) in the MAC. AQM techniques have been proposed before to both alleviate congestion control problems [9] as well as to implement the provisioning of quality of service [8]. The classical example of an AQM policy is RED [9]. In our study, however, we apply a different AQM policy: the packet dropping probability is proportional to queue length. In our work, by applying this AQM, our scheme converges to the optimal solution.

Chapter 3

Problem Settings

In this chapter, we provide problem settings and necessary background on A-CSMA and TCP Reno. The hidden-node free wireless network is attractive because it can reduce the collision thus improve the system performance. We also show the link throughput (capacity region) of this hidden-node free wireless network. Finally, we introduce an efficient scheduling algorithm called A-CSMA (Adaptive CSMA), which is proposed recently. People are attracted to A-CSMA these days because it eliminates the starvation problem appeared in L-CSMA (legacy CSMA).

3.1 Network Modeling

We model a (single-channel) CSMA wireless network by a graph $G = (V, E)$, where V is the set of nodes (i.e., wireless transmitters and receivers) and E is the set of links. We consider the scenario where multiple users communicate over the wireless network. Each user is associated with a flow over a pre-defined route between a source and a destination. The key notations used in this thesis are listed in Table 3.1. We use bold symbols to denote vectors and matrices of these quantities, e.g., $\mathbf{x} = [x_s, s \in S]$.

In CSMA wireless networks, transmitter applies a carrier-sensing mechanism to prevent collisions. In particular, a transmitter listens for an RF carrier

Notation	Definition
V	the set of nodes
E	the set of links
\mathcal{S}	the set of source-destination pairs
x_s	rate of flow s
T_s	round trip time of flow s
n_s	number of connection on flow s
y_l	average rate of link l
τ_i	time portion of independent set i
r_l	transmission aggressiveness (TA) of link l
p_l	price of link l
$l \in i$	link l belongs to independent set i
$l \in s$	flow s passes through link l
\mathcal{I}	the set of independent sets

Table 3.1: Key Notation

before sending its data. If a carrier is sensed, the transmitter defers its transmission attempt, enters a countdown process in which it waits for a random amount of time before making another transmission attempt. As such, two links are allowed to transmit simultaneously only if the simultaneous transmissions are under the CSMA carrier-sensing operations.

If not properly designed, simultaneous transmissions allowed by the carrier-sensing operations may still interfere with each other, resulting in the hidden-node problem [11]. Hidden-node-free CSMA network designs are attractive, because, importantly, their throughput analysis is tractable [28] [15]. In particular, the stationary distribution of the system state is in a product form that is fundamental to the design of adaptive CSMA [12]. As studied by [3], any CSMA wireless network can always be made to be hidden-node-free by setting the carrier sensing range properly.

In this thesis, we study the problem of end-to-end rate control and link scheduling in hidden-node-free CSMA wireless networks.

3.2 Capacity Region of Wireless Networks and Throughput-optimal Scheduling

We use the conflict graph approach in [10] to model the relationship of wireless link interference. We denote the conflict graph of G by $G_c = (V_c, E_c)$. The vertices of G_c correspond to links E in the communication graph G , i.e., $V_c = E$. An edge between two vertices means that the two corresponding links can carrier-sensing each other. An independent set corresponds to a feasible schedule which is a set of links in the communication graph that can be active simultaneously.

Let \mathcal{I} denote the set of all independent sets, τ_i denote the fraction of time the system is in schedule $i \in \mathcal{I}$. Assuming that all links have unit capacity, the capacity region of link rate is given by

$$\Pi = \{\mathbf{y} \geq 0 | \forall l, y_l \leq \sum_{i:l \in i} \tau_i, \sum_{i \in \mathcal{I}} \tau_i = 1, \boldsymbol{\tau} \geq 0\}. \quad (3.1)$$

A rate vector outside the capacity region Π is not attainable because all scheduling policy would lead to unbounded expected queue length. A special interest is to design a scheduling policy that can attain any arrival rate vector \mathbf{y} that belongs to the interior of the capacity region. Note that finding all the independent sets is NP-hard [1]. Consequently, characterizing the capacity region in (3.1) is thus very challenging, leaving alone attaining it in practice.

Definition 1 (Throughput-optimal) *A scheduling policy is said to be throughput-optimal if it stabilizes the system for any arrival rate vector that are strictly within the capacity region Π .*

In other words, throughput-optimal scheduling policy can make expected queue length bounded for all arrival rate within capacity region.

3.3 Throughput-optimality of A-CSMA

Refs. [2], [28] and [15] show that the link scheduling in CSMA networks can be studied via a Markov chain, in which the states are all independent sets. In this model, the transmitter of link l counts down an exponentially distributed random period of time with mean $\exp(-\beta r_l)$ before transmission. When the link l starts a transmission, it keeps the channel for an exponentially distributed period of time with mean one¹. This CSMA Markov chain converges to its product form stationary distribution, which is given by

$$p(i, \mathbf{r}) = \frac{\exp(\sum_{l \in i} \beta r_l)}{\sum_{i' \in \mathcal{I}} \exp(\sum_{l \in i'} \beta r_l)}, \quad \forall i \in \mathcal{I}, \quad (3.2)$$

where β is the a positive constant. The average portion of time link l is active, denoted by y_l , achieved in steady state is given by

$$y_l(\mathbf{r}) = \sum_{i: l \in i} p(i, \mathbf{r}) = \frac{\sum_{i: l \in i} \exp(\sum_{l \in i} \beta r_l)}{\sum_{i' \in \mathcal{I}} \exp(\sum_{l \in i'} \beta r_l)}, \quad \forall l \in E. \quad (3.3)$$

The authors in [12] introduce a fully distributed adaptive CSMA (A-CSMA) algorithm as follows:

Algorithm A-CSMA

$$\dot{r}_l = \alpha [a'_l - d'_l]_{r_l}^+, \quad (3.4)$$

where a'_l and d'_l are the empirical average arrival rate and service rate of link l , α is constant step size and notationally,

$$[g(z)]_z^+ = \begin{cases} \max(g(z), 0), & \text{if } z \leq 0 \\ g(z), & \text{otherwise} \end{cases} \quad (3.5)$$

Theorem 1 ([12]) *Assume arrival rate vector \mathbf{y}' is in interior of capacity region Π . Then with Algorithm A-CSMA, \mathbf{r} converges to \mathbf{r}^* , with probability one, where \mathbf{r}^* satisfies that $y'_l \leq y_l(\mathbf{r}^*)$.*

¹Without loss of generality, we normalize the mean of transmission time to one.

Remark 1: it is not difficult to verify that r_l is proportional to its local queue length of link l . Therefore, A-CSMA scheduling Algorithm can be implemented in a fully distributed way. In A-CSMA, before transmitting, any link l sets up a random exponentially distributed counter with mean equal to $\exp(-\beta r_l)$. When link l obtains the channel, it transmits a packet with length which is exponentially distributed with mean equal to one. Note A-CSMA differs from L-CSMA in that the link's countdown timer is a function of its queue length, as a result, A-CSMA gives priority to links with heavy loads. We remark that A-CSMA is simple to implement by slight modification to today's CSMA, and there have been efforts in doing so [14].

Remark 2: Theorem 1 shows that the product-form distribution $\mathbf{y}(\mathbf{r})$ achieved by A-CSMA is larger than any interior feasible rate vector. This means A-CSMA stabilizes the system for all feasible arrival rate vector, i.e., A-CSMA scheduling algorithm is throughput-optimal. Therefore, A-CSMA has superior performance as compared to L-CSMA.

For more details of A-CSMA, interested readers can refer to the work in [5, 12, 17, 19, 21].

3.4 TCP Reno Congestion Control Modeling

TCP Reno adjusts the flow rate by controlling the congestion window size W . During the congestion avoidance phase, TCP Reno increments its congestion window W by one every round trip time if no loss is observed. It halves its window whenever packet loss is detected.

In [23], the authors model this dynamic of the congestion avoidance phase of TCP Reno with the following dynamic equation:

$$\dot{x}_s = \frac{x_s^2}{2} \left(\frac{2}{T_s^2 x_s^2} - \sum_{l \in s} p_l \right) \quad (3.6)$$

where T_s is round-trip-time(RTT) of flow s , and p_l denotes the price (probability a packet will be dropped) of link l . When link l applies drop-tail queue,

$$p_l = \frac{(\sum_{s:l \in s} x_s - y_l)^+}{\sum_{s:l \in s} x_s}, \quad (3.7)$$

where y_l denotes average throughput of link l , and $[\cdot]^+$ denotes $\max(0, \cdot)$.

Chapter 4

Starvation of TCP Reno over L-CSMA and A-CSMA

Fairness is one of the key problems that must be considered in designing any MAC, which allows contending links to share the wireless channel fairly. The random backoff algorithm in the L-CSMA network gives each host equal average count-down counter to grab the channel during transmission. This random count-down algorithm works fine in a symmetric environment where all hosts connected to a single access point. However, in asymmetric settings, it fails to achieve the fair channel accessing. Even though a fair higher layer rate control protocol such as TCP cannot solve this MAC layer fairness problem.

4.1 TCP Reno Starves over L-CSMA

Extensive work, such as [2, 15, 28], studied the fairness issue of L-CSMA (IEEE 802.11 DCF) networks. They study capacity of L-CSMA (IEEE 802.11) via a Markov chain, with all links count down with equal mean time. Hence, for simplicity, we define $\rho = \exp(\beta r_l)$, where r_l s are equal for each link l . From (refeq:csma.stable.rate), the throughput of link l achieved by L-CSMA is then given by

$$y_l = \frac{\sum_{i:l \in i} (\prod_{l \in i} \rho)}{\sum_{i' \in \mathcal{I}} (\prod_{l \in i'} \rho)}, \quad \forall l \in E. \quad (4.1)$$

Since ρ is constant value (refeq:lcsma.rate), L-CSMA only can schedule a small fraction of capacity region, and is therefore not throughput-optimal.

We use a simple example shown in Fig. 4.1 to illustrate the starvation of TCP Reno over L-CSMA networks. In this example, each of the four links runs L-CSMA MAC algorithm. On the left hand side figure, it shows a communication graph with four wireless links. The right hand side figure gives the conflict graph of the networks. Each link carries a TCP Reno session. This network topology is first studied in [15]. We run NS2 simulations to study the rate distribution of TCP Reno flows. The channel bandwidth is 11Mbps and packet payload is 1000 bytes. Fig. 4.2 plots the throughput distribution of all TCP Reno flows. We observe that throughput of link 2 is almost zero, and contrarily, other three links with very high throughput.

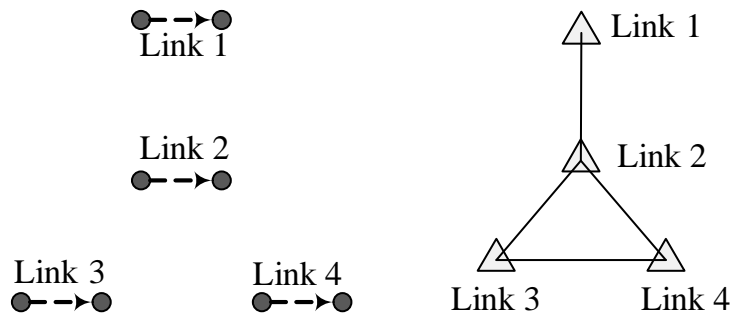


Figure 4.1: A wireless network of four links. Its conflict graph is shown on the right hand side.

The starvation phenomenon of L-CSMA is caused by the fundamental inefficiency of L-CSMA, and can be intuitively explained as follows. When link 1 is transmitting, link 2 will freeze its count down process because it is within the carrier sensing range of link 1. While link 2 is frozen, either link 3 or link 4 can transmit. This will continuously freeze link 2 after link 1 finishes its transmission. Link 1, link 3 and 4 take turns to occupy the channel, leaving link 2 very small chance to transmit.

In particular, the independent sets for the example in Fig. 4.1 are \emptyset , $\{1\}$, $\{2\}$, $\{3\}$, $\{4\}$, $\{1, 3\}$ and $\{1, 4\}$, and throughput of link 2 can be computed by

(4.1) as follows

$$y_2 = \frac{\rho}{2\rho^2 + 4\rho + 1}. \quad (4.2)$$

where $\rho \approx 2.24$ in our simulation settings [28]. Plugging ρ into (4.2), we get $y_2 \approx 0.11$, which is much smaller compared to the throughput of other links. As such, the TCP Reno session running over link 2 starves. For more discussions on the starvation of TCP Reno over L-CSMA, please refer to [15, 22, 25, 28].

4.2 TCP Reno Starves over A-CSMA

As previously stated in the Chapter 3.3, A-CSMA scheduling algorithm is throughput optimal and can be easily implemented by slightly modifying the existing L-CSMA [5, 12, 17, 19, 21]. Observing the large installed base of TCP Reno, a natural question to ask before actual deployment of A-CSMA network is how well TCP Reno can perform over A-CSMA networks.

In this section, we study the performance of TCP Reno directly running over A-CSMA network. Surprisingly, our simulation reveals that TCP Reno over A-CSMA suffers from starvation problem. In particular, it is no better than TCP Reno over L-CSMA network. By applying TCP Reno, the potential advantages of A-CSMA cannot be realized.

4.2.1 Simulations

We carry out NS-2 simulation to demonstrate the starvation problem of running TCP Reno directly over A-CSMA. We study the same network topology in Fig. 4.1. Each link carries a single TCP Reno session. The simulation settings are as follows: 1) update step size and update interval for r are 0.05 and 2.0 respectively; 2) β : weight of entropy term is 800; 4) data rate of wireless networks is 11Mbps; 5) TCP payload is 1000 byte. To prevent the countdown

from being too aggressive¹, we set the upper bound for r to be $r_{\max} = 0.01$. We remark that our simulation results hold as long as βr_{\max} is bounded. To get a clear understanding, we only consider the forward link (the link that transmits TCP DATA) in the analysis by letting the backward link send back the TCP ACK with higher priority than forward link TCP DATA. To achieve this goal, we set the backward link with a fixed maximum transmission aggressiveness r_{\max} . Consequently, most of the packets are buffered in the queue of the forward links.

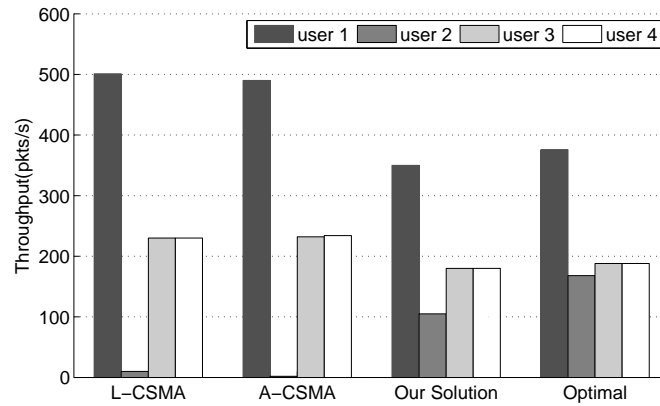


Figure 4.2: Throughput under 1) L-CSMA: NS-2 simulation of TCP Reno over L-CSMA, 2) A-CSMA: NS-2 simulation of TCP Reno over A-CSMA, 3) our solution: NS-2 simulation of multi-connection TCP Reno over A-CSMA with AQM, 4) optimal: optimal throughput computed from problem **MP**.

Fig. 4.2 plots the throughput achieved under TCP Reno over A-CSMA. It shows that they have almost the same performance. In both cases, user 2 has less throughput than other users does and therefore starves, which reveals that TCP Reno over A-CSMA also result in severe starvation. Effectively, its performance is no better than that of TCP Reno over L-CSMA, and the potential of A-CSMA cannot be realized by running TCP Reno directly over it. Because of the large installed base of TCP Reno, many Internet services such as HTTP and FTP that relies on TCP will suffer over A-CSMA networks.

¹If transmission aggressiveness are too large, $\exp(-\beta r_l)$ should be very tiny, this will result in zero countdown in digital system.

4.2.2 Observations and Explanations

The explanation of this starvation phenomenon is as follows. Initially all links have the same r and compete the channel with the same level of transmission aggressiveness. Under the link interference relationship, links 1, 3 and 4 will take turns to freeze link 2, and link 2 is not able to obtain its fair share. This is the same as the origin of the starvation problem of L-CSMA discussed in Section 4.1. To avoid starvation, link 2 must be able to count down with larger transmission aggressiveness than other links, so as to compete more aggressively for the fair channel share against the other three (less aggressive) links.

However, with TCP Reno running over A-CSMA, link 2 actually obtains a smaller transmission aggressiveness than other links, exacerbating its starvation. We explain this as follows. TCP Reno increases its rate more slowly when it experiences larger RTT, and vice versa. Initially, link 2 suffers from freezing of its backoff countdown for longer time because of the transmissions of its neighbors. Consequently, TCP Reno over link 2 experiences larger RTT and increases its congestion window more slowly than those over other links. In our example, the TCP congestion window size roughly equals to the buffer length, thus link 2's queue increases more slowly than those of other links. Since A-CSMA sets the transmission aggressiveness to be proportional to the queue length, link 2 ends up having the smallest transmission aggressiveness among all the four links. This explanation is verified by the simulation results on transmission aggressiveness in Fig. 4.3.

This is a positive feedback loop. Larger transmission aggressiveness will lead to shorter RTT. Shorter RTT makes TCP Reno increase its rate faster, thus the larger congestion window. Larger congestion window leads to longer queues, thus larger transmission aggressiveness. This loop continues until the transmission aggressiveness of all the links reach the same upper limit r_{\max} .

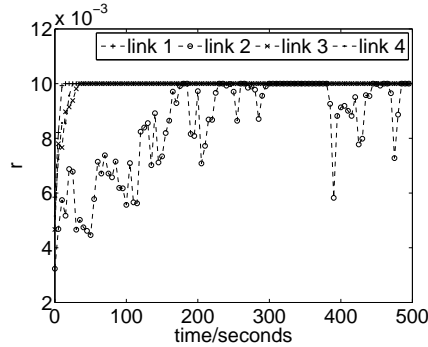


Figure 4.3: Transmission aggressiveness under A-CSMA in NS-2 simulation.

With $r_l = r_{\max}$ for $l = 1, 2, 3, 4$, the performance of TCP Reno over A-CSMA falls back to the performance of TCP Reno over L-CSMA, and link 2 suffers starvation as discussed in Section 4.1.

To break the positive feedback loop, we can adopt an AQM policy: each wireless link monitors its local queue length, and drops the incoming packets with probability proportional to queue length. In the following, we only consider this AQM policy. With this AQM policy, a link with larger transmission aggressiveness has larger packet loss ratio (transmission aggressiveness is proportional to queue length). Therefore, the TCP Reno traverses this link reduces its rate due to it experiences high packet losses. This decreases the link queue length. And thus the link transmission aggressiveness decreases, which give more chance to other lower transmitted links for transmitting. Through this analysis, we suspect that TCP Reno over A-CSMA with AQM can solve this starvation problem. However, this solution also suffers severe starvation problem.

The theoretical explanation to starvation of TCP Reno over A-CSMA can be found in Appendix A and simulation and explanation to starvation of TCP Reno over A-CSMA with AQM can be found in Appendix B.

Chapter 5

Analysis and Our Proposed Solution

In this chapter, we propose to use a layered approach of multi-connection TCP Reno over A-CSMA with AQM as a practical solution to achieve fairness. This solution is provably optimal under NUM framework. Interestingly, it can be extended to wired network and multihop networks with wired and wireless links without any change. In this solution, we stick to today's TCP Reno instead of designing a new TCP for the following reasons. First, many Internet services (e.g. HTTP, FTP) currently rely on the most widely-installed TCP Reno. Second, TCP Reno has been well-tested in the Internet scale. Last, compatibility concern seems challenging and not well-understood yet [26]. A new TCP needs to be carefully designed to be compatible with today's TCP Reno.

5.1 Proposed Solution: Multi-connection TCP Reno Scheme

We observe that TCP Reno interacts with A-CSMA through both RTT and packet loss rate. First, any loss based TCP will interact with the queue-based A-CSMA through packet loss rate. For the interaction to behave properly, we

always need to employ AQM to calibrate the mapping between the link queue-length and the link loss rate. Second, it has been observed and discussed in the previous Chapter that the reason of the flow starvation is that it suffers long RTT, which is caused by the long MAC access delay. To remove TCP bias on RTT, one way is to open multiple number of connections proportional to the RTT that TCP experiences. In this way, TCP Reno with large RTT that will open more connections to compensate for the small congestion window of each connection. As a result, there are more outstanding packets filling in the queue of the link. This makes A-CSMA allocate more airtime to the link, reducing its MAC accessing delay of the link and thus the RTT of the flow.

Inspired by the above observations, we propose to use multi-connection TCP Reno to remove the RTT interaction between TCP Reno and A-CSMA, and use AQM to calibrate their packet loss interaction. In our solution, every user s monitors the round trip time T_s , and opens $n_s = kT_s$ (where k is a constant) TCP Reno connections to remove the RTT bias. We remark that now the user rate x_s is the aggregate rate of n_s TCP Reno connections. We present the overall scheme as the following dynamic system:

$$\begin{cases} n_s = kT_s, \forall s \in \mathcal{S}; & (5.1) \\ \dot{x}_s = \frac{x_s^2}{2n_s} \left(\frac{2n_s^2}{T_s^2 x_s^2} - \sum_{l \in s} r_l \right)_{x_s}^+, \forall s \in \mathcal{S}; & (5.2) \\ \dot{r}_l = \alpha \left[\sum_{s: l \in s} x_s - \sum_{i: l \in i} \tau_i(\mathbf{r}) \right]_{r_l}^+, \forall l \in L, & (5.3) \end{cases}$$

where

$$\tau_i(\mathbf{r}) = \frac{\exp(\sum_{l \in i} \beta r_l)}{\sum_{i' \in \mathcal{I}} \exp(\sum_{l \in i'} \beta r_l)}, \forall i \in \mathcal{I}. \quad (5.4)$$

We remark that how T_s is updated is irrelevant to our solution as we always compensate for its impact by opening n_s connections. The multi-connection TCP Reno dynamic equation can be expressed in the form of (5.2)¹ [6] when

¹Recall that the single connection TCP Reno congestion window dynamic equation is,

$$\dot{W}_s = \frac{1}{T_s} - \frac{W_s^2}{2T_s} \left(\sum_{l \in s} p_l \right), \quad (5.5)$$

the end-to-end price (e.g., packet loss rate) is $\sum_{l \in s} r_l$. This link price can be fed back by packet loss based technique like Active Queue Management (AQM). In this paper, we adopt an AQM policy in which each link drops or ECN-marks packet with probability equal to r_l . We remark that the independent sets distribution satisfies (5.4) as long as each link applies CSMA mechanism and backoff with mean equal to $\exp(-\beta r_l)$. Dynamic equations (5.3) are exact to A-CSMA algorithm introduced in Section 3.3 and r_l is proportional to queue length.

The above dynamic system achieves certain system utility. With the time-scale separation assumption, it is stable and converges to the unique equilibrium. We summarize the result as the following theorem.

Theorem 2 1. Equilibrium of dynamic system in (5.1)-(5.3) solves the following Network Utility Maximization problem² :

$$\begin{aligned} \text{MP: } \max_{\mathbf{x} \geq 0, \boldsymbol{\tau} \geq 0} \quad & \sum_s -\frac{k^2}{x_s} - \frac{1}{\beta} \sum_{i \in \mathcal{I}} \tau_i \log \tau_i \\ \text{s.t.} \quad & \sum_{s: l \in s} x_s \leq \sum_{i: l \in i} \tau_i, \quad \forall l \in E, \end{aligned} \quad (5.8)$$

$$\sum_{i \in \mathcal{I}} \tau_i = 1, \quad (5.9)$$

where T_s is the round trip time and $\sum_{l \in s} p_l$ is the end-to-end packet loss rate experienced by the TCP Reno. If flow s opens n_s number of TCP connection, the flow rate x_s can be expressed as

$$x_s = \frac{n_s W_s}{T_s}. \quad (5.6)$$

From (5.5) and (5.6), we now have

$$\dot{x}_s = \frac{x_s^2}{2n_s} \left(\frac{2n_s^2}{T_s^2 x_s^2} - \sum_{l \in s} p_l \right). \quad (5.7)$$

²This formulation is similar to rate control over TDMA network except that there is an additional entropy term in the objective function. As shown in [5, 12], this leads to distributed implementation by using A-CSMA.

2. With the time-scale separation, i.e., the A-CSMA Markov chain converges to its stationary distribution instantaneously, the dynamic system (5.1)-(5.3) globally asymptotically converges to the unique optimal solution of problem **MP**.

Remark: at the equilibrium, the utility function $-\frac{1}{x_s}$ guarantee an α -fairness among users with $\alpha = 2$ [18]. One of its implications is that *no user will starve at the optimal solution at the equilibrium*. This provides theoretical justification that our proposed solution will effectively address the TCP-Reno over A-CSMA starvation problem.

Proof: 1) By relaxing the first set of inequalities (5.8) of problem **MP**, we get its partial Lagrangian as follows:

$$L(\mathbf{x}, \boldsymbol{\tau}, \mathbf{r}) = \sum_{s \in \mathcal{S}} k^2 U_s(x_s) - \frac{1}{\beta} \sum_{i \in \mathcal{I}} \tau_i \log \tau_i - \sum_{l \in E} r_l \left(\sum_{s: l \in s, s \in \mathcal{S}} x_s - \sum_{i: l \in i} \tau_i \right), \quad (5.10)$$

where $\mathbf{r} = [r_l, l \in E]$ is the vector of Lagrange multipliers. We notice that $\sum_{l \in E} r_l \sum_{i: l \in i} \tau_i = \sum_{i \in \mathcal{I}} \tau_i \sum_{l \in i} r_l$. Since the problem **MP** is a concave optimization problem and Slater's condition holds, the optimal solution of problem **MP** can be found by solving the following problem successively in $\boldsymbol{\tau}$, \mathbf{x} and \mathbf{r} :

$$\begin{aligned} \min_{\mathbf{r} \geq 0} \max_{\mathbf{x} \geq 0, \boldsymbol{\tau} \geq 0} & \sum_{s \in \mathcal{S}} k^2 U_s(x_s) - \sum_{l \in E} r_l \sum_{s: l \in s} x_s \\ & + \sum_{i \in \mathcal{I}} \tau_i \sum_{l \in i} r_l - \frac{1}{\beta} \sum_{i \in \mathcal{I}} \tau_i \log \tau_i \\ \text{s.t.} & \sum_{i \in \mathcal{I}} \tau_i = 1. \end{aligned} \quad (5.11)$$

Define

$$g_\beta(\mathbf{r}) \triangleq \frac{1}{\beta} \log \left(\sum_{i \in \mathcal{I}} \exp(\beta \sum_{l \in i} r_l) \right). \quad (5.12)$$

The conjugate of $g_\beta(\mathbf{y})$ is defined as $g^*(z) = \sup_{\mathbf{y} \in \text{dom}g} (z^T \mathbf{y} - g(\mathbf{y}))$. Therefore, the conjugate of $g_\beta(\mathbf{r})$ is given by

$$g_\beta^*(\boldsymbol{\tau}) = \begin{cases} \frac{1}{\beta} \sum_{i \in \mathcal{I}} \tau_i \log \tau_i & \text{if } \boldsymbol{\tau} \geq 0 \text{ and } \mathbf{1}^T \boldsymbol{\tau} = 1 \\ \infty & \text{otherwise.} \end{cases} \quad (5.13)$$

$$(5.14)$$

The conjugate of its conjugate is itself, i.e., $g_\beta(\mathbf{r}) = g_\beta^{**}(\mathbf{r})$. Therefore, we have

$$\begin{aligned} g_\beta(\mathbf{r}) = \max_{\boldsymbol{\tau} \geq 0} & \sum_{i \in \mathcal{I}} \tau_i \sum_{l \in i} r_l - \frac{1}{\beta} \sum_{i \in \mathcal{I}} \tau_i \log \tau_i \\ \text{s.t.} & \sum_{i \in \mathcal{I}} \tau_i = 1, \end{aligned} \quad (5.15)$$

with the corresponding unique optimal solution

$$\tau_i(\mathbf{r}) = \frac{\exp\left(\sum_{l \in i} \beta r_l\right)}{\sum_{i' \in \mathcal{I}} \exp\left(\sum_{l \in i'} \beta r_l\right)}, \quad \forall i \in \mathcal{I}. \quad (5.16)$$

(Note this is the exact stationary distribution that CSMA Markov chain achieves.

Hence, A-CSMA can be used to solve this subproblem.) From (5.12) and

(5.15), formula (5.11) will be

$$\begin{aligned} \min_{\mathbf{r} \geq 0} \max_{\mathbf{x} \geq 0} & \sum_{s \in \mathcal{S}} k^2 U_s(x_s) - \sum_{l \in E} r_l \sum_{s: l \in s} x_s \\ & + \frac{1}{\beta} \log \left(\sum_{i \in \mathcal{I}} \exp\left(\beta \sum_{l \in i} r_l\right) \right). \end{aligned} \quad (5.17)$$

Define

$$\begin{aligned} L_2(\mathbf{x}, \mathbf{r}) \triangleq & \sum_{s \in \mathcal{S}} U_s(x_s) - \sum_{l \in E} r_l \sum_{s: l \in s} x_s \\ & + \frac{1}{\beta} \log \left(\sum_{i \in \mathcal{I}} \exp\left(\beta \sum_{l \in i} r_l\right) \right). \end{aligned} \quad (5.18)$$

The saddle point of above formula is the optimal primal and dual variable.

Therefore, the partial derivative is equal to zero, we have

$$\frac{\partial L_2(\mathbf{x}, \mathbf{r})}{\partial x_s} = k^2 U'_s(x_s) - \sum_{l \in s} r_l = 0, \quad (5.19)$$

$$\frac{\partial L_2(\mathbf{x}, \mathbf{r})}{\partial r_l} = - \left(\sum_{s: l \in s} x_s - \sum_{i: l \in i} \tau_i(\mathbf{r}) \right) = 0. \quad (5.20)$$

When $U_s(x_s) = -\frac{1}{x_s}$, a primal-dual algorithm that solves above problem is

$$\begin{cases} \dot{x}_s = f(x_s) \left(\frac{2k^2}{x_s^2} - \sum_{l \in s} r_l \right)_{x_s}^+, \quad \forall s \in \mathcal{S}; \end{cases} \quad (5.21)$$

$$\begin{cases} \dot{r}_l = \alpha(r_l) \left[\sum_{s: l \in s} x_s - \sum_{i: l \in i} \tau_i(\mathbf{r}) \right]_{r_l}^+, \quad \forall l \in L. \end{cases} \quad (5.22)$$

Plus the (5.16), we now have that the equilibrium of dynamic system in (5.1)-(5.3) solves **MP**.

2) With such time-scale separation assumption, the proof can use the same set of standard Lyapunov elaborated in [5]. We construct a Lyapunov function,

$$V(\mathbf{x}, \mathbf{r}) = \sum_{s \in S} \int_{x_s^*}^{x_s} \frac{1}{f_s(\delta)} (\delta - x_s^*)^+ d\delta + \sum_{l \in L} \int_{r_l^*}^{r_l} \frac{1}{\alpha(\xi)} (\xi - r_l^*)^+ d\xi, \quad (5.23)$$

where $(\mathbf{x}^*, \mathbf{r}^*)$ is the optimal solution. We assume the A-CSMA Markov chain converges to its stationary instantaneously. So we have the equilibrium condition

$$\sum_{s:l \in s} x_s^* = \sum_{i:l \in i} \tau_i \quad (5.24)$$

$$\frac{2k^2}{x_s^{*2}} = \sum_{l \in s} r_l^* \quad (5.25)$$

The Lyapunov function $V(\mathbf{x}, \mathbf{r})$ is continuous and positive definite. Because $f(x_s) > 0, \alpha(r_l) > 0$, for $x_s > 0, r_l > 0$, it is easy to check that $V(\mathbf{x}, \mathbf{r}) \geq 0$ and $V(\mathbf{x}, \mathbf{r}) = 0$ if and only if $(\mathbf{x}, \mathbf{r}) = (\mathbf{x}^*, \mathbf{r}^*)$. And also $V(\mathbf{x}, \mathbf{r}) \rightarrow \infty$ if $(\mathbf{x}, \mathbf{r}) \rightarrow \infty$. Therefore, Lyapunov function V is an energy-like function.

The time derivative of Lyapunov function is

$$\dot{V}(\mathbf{x}, \mathbf{r}) = \sum_{s \in S} \dot{x}_s (x_s - x_s^*)^+ + \sum_{l \in L} \dot{r}_l (r_l - r_l^*)^+ \quad (5.26)$$

Plug equations (5.1)-(5.3) into above equation, we have

$$\begin{aligned} \dot{V}(\mathbf{x}, \mathbf{r}) &= \sum_{s \in S} \left(\frac{2k^2}{x_s^2} - \sum_{l \in s} r_l \right) x_s (x_s - x_s^*)^+ \\ &\quad + \sum_{l \in L} \left(\sum_{s:l \in s} x_s - \sum_{i:l \in i} \tau_i \right) r_l (r_l - r_l^*)^+ \\ &\leq \sum_{s \in S} \left(\frac{2k^2}{x_s^2} - \sum_{l \in s} r_l \right) (x_s - x_s^*) \\ &\quad + \sum_{l \in L} \left(\sum_{s:l \in s} x_s - \sum_{i:l \in i} \tau_i \right) (r_l - r_l^*) \\ &= \sum_{s \in S} x_s^* \left(\sum_{l \in s} r_l^* - \frac{2k^2}{x_s^2} \right) + \sum_{s \in S} x_s \left(\frac{2k^2}{x_s^2} - \sum_{l \in s} r_l^* \right) \\ &\quad + \sum_{l \in L} r_l \left(\sum_{s:l \in s} x_s^* - \sum_{i:l \in i} \tau_i \right) \end{aligned} \quad (5.27)$$

Plug (5.24) and (5.25) into (5.27), we get

$$\dot{V}(\mathbf{x}, \mathbf{r}) = - \sum_{s \in S} \frac{2k^2 (x_s - x_s^*)^2 (x_s + x_s^*)}{x_s^2 x_s^{*2}} \leq 0. \quad (5.28)$$

Therefore $V(\mathbf{x}, \mathbf{r})$ is non-increasing along system updating. The dynamic system in (5.1)-(5.3) converges to the optimal point $(\mathbf{x}^*, \mathbf{r}^*)$ astronomically. \square

Theoretically, under the fluid model analysis, the proof of convergence can still be established under some mild conditions on parameters. The ideas are very similar to those expounded in [13, 24].

Without such time-scale separation assumption, the dynamic system turns into a stochastic dynamic system, where the link rate measured by link l does not satisfy equations (5.4). Under small step size and update interval, the resulted stochastic dynamic system is shown to converge to a bounded neighborhood of the same optimal solutions with probability one [5, 24].

We validate the convergence of the algorithm in packet-level simulations later in Chapter 6.

5.2 Implementation

Inspired by the observations from the dynamic systems in (5.1)-(5.3). We can solve the problem by running multi-connection TCP Reno over A-CSMA with AQM. We stress multi-connection is important in our solution in that it removes the RTT interaction with A-CSMA. Without removing such RTT bias on RTT, TCP Reno over A-CSMA with AQM also results in starvation problem.

The multi-connection TCP Reno scheme can be implemented in two ways, without modification to the transport layer. One method is to insert an intermediate layer between the application layer and transport layer, called Multi-connection API. It provides the universal API to the application layer, and functions to maintain kT_s TCP Reno connections from monitored RTT. This solution requires no modification to TCP/IP stack and is compatible with today's applications (like FTP, HTTP, etc.).

The other implementation method is to rewrite the applications. The application keeps monitoring RTT and opens multiple sockets. To obtain the RTT, we can simply set up a UDP connection to measure the RTT [6, 7].

We prefer the first implementation in which we insert a multi-connection API in between. This implementation does not require any modification to today's applications and hence simplifies programming. From the structure of the dynamic system in (5.1)-(5.3), it can be implemented in a layered manner as follows, with the diagram shown in Fig. 5.1:

- in the MAC layer, we run A-CSMA to schedule the link transmissions. This is directly obtained from dynamic equations (5.3). A-CSMA maintains transmission aggressiveness vector \mathbf{r} to be proportional to the lengths of links' queue.
- we run AQM at each link that drops or marks packets with a probability proportional to \mathbf{r} , and thus proportional to the queue lengths. In this way, the prices of using individual links can be fed back to end users via packet losses or ECN marks. We remark that AQM is required if we want to run loss based TCP (e.g. Reno) over A-CSMA.
- in the transport layer: we perform rate control based on sum of the prices of individual links, which is fed back in a form of packet losses or ECN marks. This result is directly obtained from equations (5.2). When loss ratio of link l is r_l , the total loss ratio that TCP Reno sees is $1 - \prod_{l \in s} (1 - r_l) \approx \sum_{l \in s} r_l$, when r_l is small.
- multi-connection layer, we maintain $n_s = kT_S$ connections to remove the RTT bias.

We remark that AQM is minimally required in order to run loss-based TCP (e.g. TCP Reno) over A-CSMA. The pseudo code of A-CSMA with AQM is shown in Alg. 1.

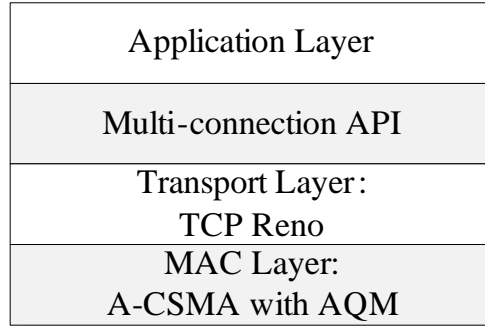


Figure 5.1: Scheme Diagram

Algorithm 1 : Pseudo code of A-CSMA with AQM.

```

1: // Updating the transmission aggressiveness:
2: if at the end of time unit  $t - 1$  then
3:   Transmission aggressiveness  $r_l \leftarrow \alpha \cdot Q_l(t - 1)$ .
4: end if
5: // Setting the backoff window:
6: if before transmitting a packet with size  $PKT$  then
7:   Sets the backoff window following exponential distribution with mean
   equal to  $PKT * \exp(-\beta r_l)$ .
8: end if
9: // Adjusting the queue length:
10: if a packet is served then
11:   Queue length  $Q_l(t) \leftarrow \max(Q_l(t) - 1, 0)$ .
12: end if
13: if a new packet is inserted into queue then
14:   (a) Discards tail packet with probability  $\min(r_l, 1)$ .
15:   (b)  $Q_l(t) \leftarrow Q_l(t) + 1$ .
16: end if
17: if no packet to serve in the queue then
18:   Generates a dummy packet.
19: end if

```

In practice, user s can only open an integer number of connections. By setting $n_s = \text{int}(kT_s)$, user s can react to the change in T_s in fine granularity if k is large, and in coarse granularity for small k . While large k is preferred in theory, in practice user s may need to maintain a large number of connections if k is too large. Large k also make the n_s adjustment more responsive to the changes in T_s , so it induces more overhead due to frequently opening and

closing TCP Reno connections.

Packet loss ratio should be far less than one, otherwise it will waste considerable bandwidth for dropping large number of packets. Besides, the total loss ratio $1 - \prod_{l \in s} (1 - r_l)$ does not hold when r_l is large. We can scale r_l so as to make it much smaller than one.

5.3 Discussion

We discuss the following three questions in this subsection: 1) By using multi-connection TCP Reno, can we achieve arbitrary utility function? 2) Is our solution applicable to wired network as well as multihop networks with wired and wireless links? 3) How to define the overhead of our solution?

5.3.1 Achieve Arbitrary Utility

For the first question, our answer is yes. Let $U_s(x_s)$ be arbitrary utility function of problem **MP**. Consider our solution in (5.1)-(5.3) but replacing the updating equation of n_s as follows:

$$n_s = k \sqrt{\frac{U'_s(x_s) T_s^2 x_s^2}{2}}. \quad (5.29)$$

It is not difficult to verify that the equilibrium of the modified solution solves problem **MP** with the specified utility function $U_s(x_s)$.

In particular, our proposed solution can be understood as a special case of the above approach with the specified utility function being $U_s(x_s) = -\frac{1}{x_s}$.

5.3.2 Extension to Networks with Both Wired and Wireless Links

Our answer to the second question is also yes. We consider a multihop network composed of wired and wireless links. Let l denote the wireless link and w

denote the wired link. Our dynamic system to solve the extension problem of networks composed of wired and wireless links is as follows:

$$\begin{cases} n_s = kT_s, \forall s \in \mathcal{S}; & (5.30) \end{cases}$$

$$\begin{cases} \dot{x}_s = \frac{x_s^2}{2n_s} \left(\frac{2n_s^2}{T_s^2 x_s^2} - \sum_{l \in s} r_l - \sum_{w \in s} p_w \right)_{x_s}^+, \forall s \in \mathcal{S}; & (5.31) \end{cases}$$

$$\begin{cases} \dot{r}_l = \alpha \left[\sum_{s:l \in s} x_s - \sum_{i:l \in i} \tau_i(\mathbf{r}) \right]_{r_l}^+, \forall l \in E', & (5.32) \end{cases}$$

where

$$\tau_i(\mathbf{r}) = \frac{\exp(\sum_{l \in i} \beta r_l)}{\sum_{i' \in \mathcal{I}} \exp(\sum_{l \in i'} \beta r_l)}, \forall i \in \mathcal{I}. \quad (5.33)$$

$$p_w = \frac{(\sum_{s':w \in s'} x_{s'} - C_w)^+}{\sum_{s':w \in s'} x_{s'}}, \forall w \in E_w, \quad (5.34)$$

and E_w denotes the set of wired links, E_l denotes the set of wireless links, C_w stands for the capacity of wired link w , and p_w is the packet loss ratio of wired link (when it applies drop-tail queue).

Theorem 3 Equilibrium of dynamic system in (5.30)-(5.32) solves the following Network Utility Maximization problem:

$$\begin{aligned} \mathbf{EP:} \quad \max \quad & \sum_s -\frac{k^2}{x_s} - \frac{1}{\beta} \sum_{i \in \mathcal{I}} \tau_i \log \tau_i \\ & - \sum_{w \in E_w} \int_0^{\sum_{s':w \in s'} x_{s'}} \frac{(y - C_w)^+}{y} dy \end{aligned} \quad (5.35)$$

$$s.t. \quad \sum_{s:l \in s} x_s \leq \sum_{i:l \in i} \tau_i, \quad \forall l \in E', \quad (5.36)$$

$$\sum_{i \in \mathcal{I}} \tau_i = 1, \quad (5.37)$$

$$\mathbf{x} \geq 0, \mathbf{\tau} \geq 0, \quad (5.38)$$

where the utility function is $U_s(x_s) = -\frac{1}{x_s}$, $\forall s \in \mathcal{S}$ and

$$\int_0^{\sum_{s':w \in s'} x_{s'}} \frac{(y - C_w)^+}{y} dy \quad (5.39)$$

is the penalty function which penalizes the arrival rate for exceeding the wired link capacity c_w .

This theorem can be proved using similar method of the proof of Theorem 2.

From the above extension dynamic system, we have the following observations: 1) in MAC layer, wireless links perform A-CSMA with AQM; 2) link dropping ratio at wireless link is exact equal to packet dropping probability at today's routers (with drop-tail queue). Therefore, our solution discussed in Section 5.2 can be directly extended to networks with wired and wireless links without modification to any network components.

5.3.3 Impact of ACK Traffic

In order to be consistent with the dynamic equations in (5.1)-(5.3), we can set transmission aggressiveness of backward link that carries TCP ACK traffic to r_{max} . Thus, TCP ACK is sent back instantaneously. Together with small size of ACK packet makes the ACK backward link effects negligible. However, such operation, in reality, increases burden of MAC layer. For example, MAC layer needs to examine the type of the arrival packet, and transmits ACK packet with highest priority by setting large transmission aggressiveness. So we are inclined to a simple solution which does not require such examine process in MAC layer. In this part, we discuss about two settings: 1) ACK with high priority (**h-ACK**) and, 2) normal ACK (**n-ACK**).

In h-ACK setting, all links measure queue length in data bytes. However, in normal ACK setting, we need to consider two queue length measurements for all links (backward and forward links), one is in number of packets, and the other is in data Bytes. If all links' queue length are measured in data bytes, it will result in the backward link with very small transmission aggressiveness, because ACK packet size is far smaller than TCP packet. Then forward link can transmit all data packets in the queue because backward link is not complete with forward link. When forward link queue is empty, it generates a dummy packet with packet size still larger than ACK packet size. This

makes the forward link's transmission aggressiveness still larger than backward link's, which leads to backward starvation problem and so as to degrade the total throughput. Thus, we employ the first queue length measurement methods in our simulation. Actually, the dynamic system in (5.1)-(5.3) can be transformed into

$$\begin{cases} n_s = kT_s, \forall s \in \mathcal{S}; & (5.40) \\ \dot{x}_s = \frac{x_s^2}{2M_s n_s} \left(\frac{2M_s^2 n_s^2}{T_s^2 x_s^2} - \sum_{l \in \mathcal{S}} r_l \right)_{x_s}^+, \forall s \in \mathcal{S}; & (5.41) \\ \dot{r}_l = \alpha \left[\sum_{s: l \in \mathcal{S}} \frac{x_s}{M_s} - \frac{\sum_{i: l \in i} \tau_i(\mathbf{r})}{\bar{M}_s} \right]_{r_l}^+, \forall l \in L, & (5.42) \end{cases}$$

where M_s is the packet size of flow s , and \bar{M}_s is the average packet size on link l and satisfies

$$\bar{M}_s = \frac{\sum_{s: l \in \mathcal{S}} \frac{x_s}{M_s}}{\sum_{s: l \in \mathcal{S}} x_s}. \quad (5.43)$$

$\sum_{s: l \in \mathcal{S}} \frac{x_s}{M_s}$ is the total number of arrival packets per second at link l and $\frac{\sum_{i: l \in i} \tau_i(\mathbf{r})}{\bar{M}_s}$ approximately equal to the average number of packets being served per second on link l . It is not difficult to verify that r_l is proportional to number of packets in queue. When we move \bar{M}_s out of brackets, equation (5.42) is the same to equation (5.3). This proves we can also update transmission aggressiveness proportional to queue length.

5.3.4 Tradeoff between performance and overhead

From the problem formulation **MP**, our dynamic system in (5.1)-(5.3) does not achieve utility $-\frac{1}{x}$ exactly, because there is an entropy term in the objective function. Theoretically, the optimal solution depends on the product $k^2\beta$. Further, larger $k^2\beta$ produces better performance. When $k^2\beta$ approaches to infinity, the entropy term is zero, so the dynamic system exactly achieves the utility function. However, these two parameters cannot be set arbitrarily large. Practically, large β leads to the following problems:

- short-term starvation problem: Because r is very large, the backoff interval is very small. Once a link selects a large backoff interval, other links transmit in burst. Its countdown timer is frozen for a long time and hence it suffers short-term starvation.
- cannot be implementable in the MAC layer: If β is too large, links wait such short time that may be smaller than the clock cycle of the digital system. Hence, the collision is not avoidable because every link does not wait before transmitting.

We cannot use too large k also. Although more connections implies large transmission aggressiveness and therefore better performance, it leads to large overhead in the number of connections. It also can be verified from problem formulation **MP** that the smaller weight of the entropy term gives better performance. Hence, there is always a tradeoff between overhead and performance. A proper chosen k and β is important in achieving both acceptable overhead and performance.

We will verify these analysis that the performance is the same as long as $k^2\beta$ is fixed in the simulation chapter. Also we will study the effects of the parameter k and β through simulations.

5.3.5 Overhead of Multi-connection TCP

In our solution, applications need to operate multiple TCP connections. A natural concern is what is the overhead of maintaining and frequently opening and closing multiple TCP Reno connections. In this study, we will consider following types of overhead in multi-connection TCP:

1. signaling
2. port number usage
3. memory cost

4. CPU cycles

The notations used in discussing overhead is listed in Table. 5.1.

Notation	Definition
M	memory resource consumption per TCP connection
M_c	total memory resource consumption in multi-connection TCP
d	update interval of n_s
D_e	the duration for establishing a TCP connection
D_c	the duration for closing a TCP connection
D_p	the duration for sending a DATA packet
Ω	CPU overhead of connection management
Thr	user throughput
Co	the total CPU overhead

Table 5.1: Notation Used in Computing Overhead

Recall that TCP uses packet-loss to signal the network congestion. In our multi-connection TCP solution, we employ AQM to calibrate the mapping between queue-length based A-CSMA and loss-based TCP Reno. The overhead of retransmitting due to packet-loss, can be computed by $(Thr \cdot r_l)$, when AQM policy is employed.

Each connection is associated with four elements, which are local IP, local port number, remote IP and remote port number, so the available connection numbers can be much larger than local available numbers of port. Port numbers range from 0 to 65535 can be used in currently used TCP/IP stack. Theoretically, at most 65536 source port number can be used, so the total number of available connections is 65536. This is a very large number, so the insufficient in port number is unlikely to occur.

Memory is precious resource in digital system and multiple TCP connections consumes more memory resource than single one. Suppose to maintain a TCP connection consumes M memory resource. From RFC [20], all persistent TCP-specific data about a connection is stored in a single large structure, the transmission control block (TCB). Typically, M equals to size of TCB, which is 512 bytes. Let $t_1, t_2, \dots, t_k, \dots$ be the point in time to update n_s . Because

n_s is updated every d seconds, so $d = t_{k+1} - t_k$. During interval t_k and t_{k+1} , the connection number is $n_s(t_k)$. Therefore, the memory resource, denoted by $Mc(t_k)$, needed from time t_k to t_{k+1} is given by,

$$Mc(t_k) = n_s(t_k) \cdot M. \quad (5.44)$$

Assuming the running time is R , the average memory overhead can be expressed as

$$\bar{M}c = \frac{\sum_{k=1}^{R/d} M \cdot n_s(t_k)}{R/d}, \quad (5.45)$$

where R/d is the number of times that n_s is updated. It is easily to verify that $\bar{M}c$ converges to $M \cdot n_s^*$ as n_s converges to n_s^* . Since M is a small value compared to memory installed in today's digital system, the extra memory overhead of maintaining multiple TCP connections is small.

For intermediate layer implementation, M is equal to TCB data size. Hence, the memory overhead is very small in this case.

For application layer implementation, each connection is a thread in application. Thread management and scheduling consumes CPU and memory resources, so it uses more resource than intermediate layer implementation does. In practice, we find opening every 100 more connections consumes 2.4 Mb memory. Besides, the number of threads an application can open is limited by system resources, and it cannot be sufficiently large, as such limits the number of connections. In typical, it is 1024 in Linux platform (defined by variable `PTHREAD_THREADS_MAX` in file `/usr/include/bits/local_lim.h`). These limitations are another reasons we prefer intermediate layer implementation.

We now consider CPU overhead in multi-connection TCP. CPU overhead can be divided into two parts. One part is CPU cycles in opening/closing connections, called connection management overhead. This overhead is small when n_s changes smoothly and becomes large when n_s changes dynamically. The other part is CPU cycles in transmitting data packet. Note that this part is independent of connection number (n_s), since transmitting k packets using

multiple TCP connections consumes the same CPU resources as transmitting them using single TCP connections. This part is the common overhead of multi-connection TCP and single-connection TCP.

We denote the CPU duration of establishing a TCP connection by D_e . This procedure consists of socket initialization and the TCP three-way-handshake. And we denote the CPU duration of closing a TCP connection by D_c . Let $n_s(t_{k-1})$ and $n_s(t_k)$ be the number of connections opened at time t_k and t_{k+1} respectively. Thus, the CPU overhead $\Omega(t_k)$ of managing connections at time t_k is

$$\Omega(t_k) = \begin{cases} [n_s(t_k) - n_s(t_{k-1})] D_e & \text{if } n_s(t_{k-1}) < n_s(t_k) & (5.46) \\ [n_s(t_{k-1}) - n_s(t_k)] D_c & \text{if } n_s(t_{k-1}) > n_s(t_k) & (5.47) \\ 0 & \text{otherwise.} & (5.48) \end{cases}$$

At beginning, no TCP connection is established, so $n_s(t_0) = 0$. When transmission finishes and the finishing time is t_e , all connections should be closed, so $n_s(t_e) = 0$. If $n_s(t_1) < n_s(t_2)$, $[n_s(t_2) - n_s(t_1)]$ extra connections will be opened, the overhead is mainly on the establishment of TCP Reno connections. If $n_s(t_1) > n_s(t_2)$, the overhead comes from closing the extra TCP Reno connections. When $n_s(t_1) = n_s(t_2)$, the overhead is zero because no CPU time is needed on closing or opening TCP connections.

In a PC with Intel P4 processor 3.2Ghz, 1Mb L2 cache and 1Gb SDRAM, we find the average duration of initializing a socket is nearly $12\mu s$ and TCP three-way-handshake is nearly $406\mu s$. Closing a connection takes $12\mu s$, and sending a packet with payload 1000bytes takes $14\mu s$. In this setting, D_e is $418\mu s$, D_c is $12\mu s$ and D_p is $14\mu s$. We observe that the duration of establishing a connection is much longer than other durations, mainly due to overhead of TCP three-way-handshake is large. Suppose the average throughput is Thr (packets/s) and running time is R , the total CPU overhead Co is

$$Co = \sum_{k=1}^{R/d+1} \Omega(t_k) + Thr \cdot R \cdot D_p, \quad (5.49)$$

where d is update interval for n_s . In Eq.(5.49), $\sum_{k=1}^{R/d+1} \Omega(t_k)$ is connection management CPU overhead. It is the extra overhead of our solution from running single-connection TCP. The second part of Eq.(5.49), $(Thr \cdot R \cdot D_p)$, is the time CPU spend on transmitting data packet. This part is the same for multi-connection TCP and single-connection TCP. So we define efficiency as the ratio of this common overhead to the total overhead, which is given by,

$$\eta = \frac{Thr \cdot R \cdot D_p}{C_o}. \quad (5.50)$$

When the efficiency η is larger, the connection management overhead in our solution is relatively small compared to total overhead. Note that the efficiency is one when the number of connections n_s converges.

We will take a closer look at these overheads in Chapter 6.

Chapter 6

Simulations

We evaluate the performance of multi-connection TCP Reno over A-CSMA stated in Chapter 5.1 with NS-2 simulation. We have implemented A-CSMA with AQM in NS-2 following the description of Algorithm 1 by modifying the IEEE 802.11 module. In the following, we refer to this scheme with ACK sent back instantaneously as h-ACK, and refer to this scheme without such ACK operation as n-ACK.

Unless specified otherwise, the simulation parameters are the same. All simulations are conducted with zero channel losses, so as to give us a clear observation. By default, 1) k : the coefficient for n_s , is 10; 2) update step size α and update interval for r are 0.1 and 1.0second, respectively; 3) the weight of entropy term $\beta = 200$; 4) data rate is 11Mbps; 5) carrier sensing range and transmission range are 800m and 250m respectively to avoid hidden-node problem. We do not wish to vary the number of TCP Reno connection too dynamically. In the simulation we update the connection number every 5 seconds. In the following subsections, we will investigate our solution under three different scenarios: single-hop networks, multihop pure wireless networks and multihop networks with wireless links and wired links.

6.1 Single-hop Wireless Networks Scenario

In this part, we consider single-hop wireless network scenario. To understand the performance of h-ACK and n-ACK, we examine three topologies, with their conflict graph shown in Fig. 6.1. All topologies consist of four links, with each link carrying a single user data flow. Each of these three topologies has the starvation problem when TCP Reno is running over L-CSMA networks, which has been studied in [15]. In topology *a*, which has been studied in previous sections, TCP flow running over link 2 starves. In topology *b*, link 1 starves and links 2, 3 and 4 obtain the whole portion of channel access. In topology *c*, links 2 and 3 starve, link 1 and link 4 get the half of the channel bandwidth.

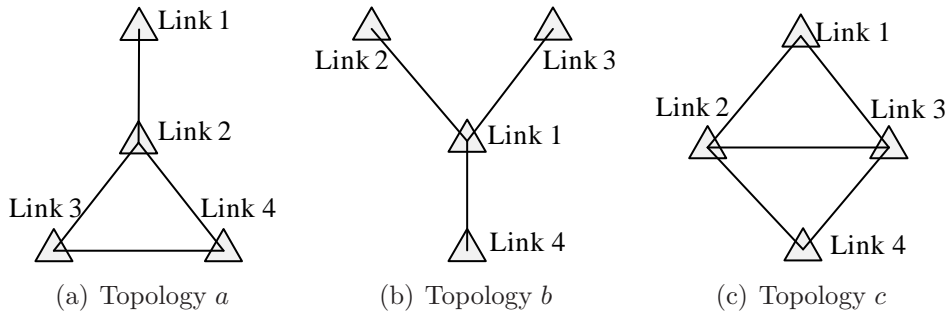


Figure 6.1: Link Conflict Graphs of single-hop network topologies.

6.1.1 Fairness and Throughput

We measure the throughput of each user flow by L-CSMA, h-ACK and n-ACK, and compare these throughput with the optimal throughput in Fig. 6.2. The optimal throughput is computed by solving the utility maximization problem **MP**. We observe an excellent agreement between simulation result of our solution and optimal. To understand the fairness index among all schemes, we define utility gap $\Delta U = \sum_s (U(x_s) - U(x_s^*))$ as the difference between utility achieved by simulation and the optimal utility. When ΔU is close to zero,

the system achieves fair throughput allocation. Table. 6.1 compares the utility gap in three topologies by L-CSMA, n-ACK and h-ACK. The aggregate throughput and fairness is always a tradeoff. This is the reason we observe that the aggregate throughput achieved by L-CSMA is the largest because it has the lowest fairness. We observe that the utility gap of both n-ACK and h-ACK are very close to zero. Therefore, n-ACK and h-ACK have superior performance compared to L-CSMA. Because of overhead in countdown before transmitting ACK packets, n-ACK gets poorer utility gap.

	Topo. <i>a</i>	Topo. <i>b</i>	Topo. <i>c</i>
L-CSMA	-268.5	-541.4	-190.8
h-ACK	-1.6	-1.6	-2.0
n-ACK	-4.1	-4.2	-5.6

Table 6.1: Utility gap ΔU under different schemes

For a better understanding of our solution, we have measured in simulation detailed information about each flow and link. In particular, we plot evolution of transmission aggressiveness r and the number of connections n_s in the topology a by h-ACK and n-ACK in Fig. 6.3 and 6.4. We observe that r and n_s are stable by both h-ACK and n-ACK. Note that packet loss ratio r of each link, around $0.005 \sim 0.01$ by h-ACK and $0.002 \sim 0.006$ by n-ACK, is very small, so throughput does not degrade due to dropping too many packets. User 2 opens 20 connections for both cases in the simulation, which is the most among all flows. This number can be further reduced by using a smaller k . Larger k ensures finer granularity in adjusting n_s to be more responsive to the change in T_s , and therefore lead to better system performance, but at the cost of consuming more device resources. We discuss the overhead of the n-ACK in next subsection. In practice, both link countdown procedure and transmission delay affects RTT, result in very dynamic in RTT. This is why we see sharp oscillation of n_s in Fig. 6.4 and 6.3, because n_s is tuned to remove RTT bias. We can smooth n_s by taking the average of the current and last value, but

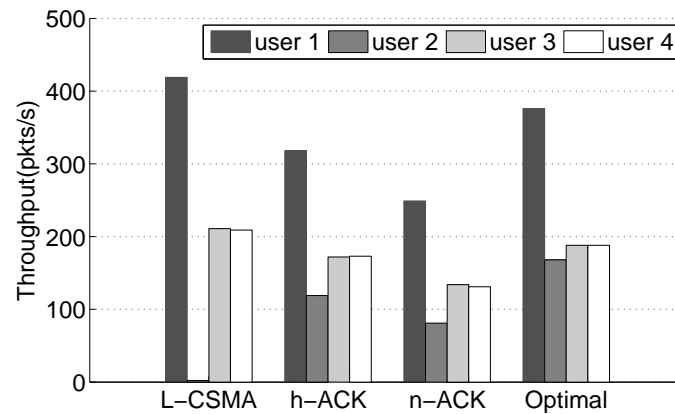
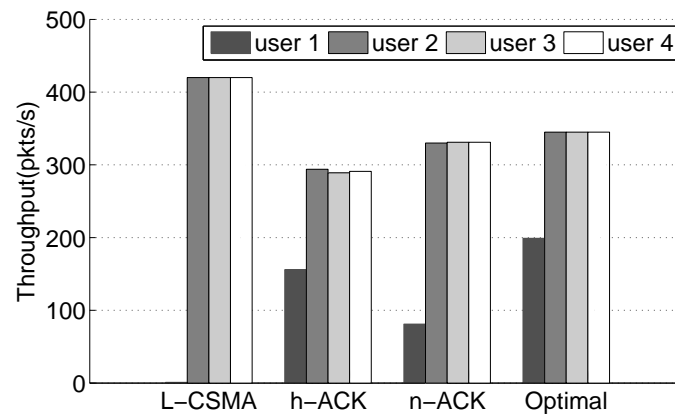
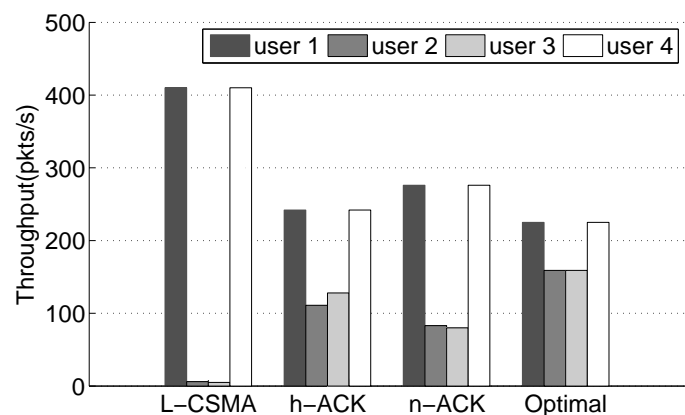
(a) Topology *a*(b) Topology *b*(c) Topology *c*

Figure 6.2: Throughput under 1) L-CSMA: NS-2 simulation of TCP Reno over L-CSMA, 2) h-ACK: NS-2 simulation of multi-connection TCP Reno over A-CSMA with AQM and TCP ACKs sent back instantaneously, 3) n-ACK: NS-2 simulation of multi-connection TCP Reno over A-CSMA with AQM and TCP ACKs operated normally, 4) optimal: optimal throughput computed from problem **MP**.

there is a tradeoff between smooth of n_s and responsive to the change in T_s .

	Topo. <i>a</i>	Topo. <i>b</i>	Topo. <i>c</i>
L-CSMA	841	1261	831
h-ACK	780	1030	723
n-ACK	739	1073	715

Table 6.2: Overall throughput under different schemes (packets/second)

Another observation from Fig. 6.3 and 6.4 is that r by h-ACK is larger than r by n-ACK. Large r leads to short countdown, and therefore high throughput. We compute the overall simulation throughput in Table.6.2. It verifies our analysis that overall throughput by h-ACK is higher. The other reason leads to higher throughput by h-ACK is that air time is further saved by TCP ACK being sent back instantaneously.

From throughput and fairness perspective, both h-ACK and n-ACK achieve fair and efficient rate allocation among different users in the single hop scenario. They both have solved the starvation problem of L-CSMA and therefore have superior advantage over L-CSMA solution. Compared to h-ACK, n-ACK has lower throughput, but it is a simpler and practical solution.

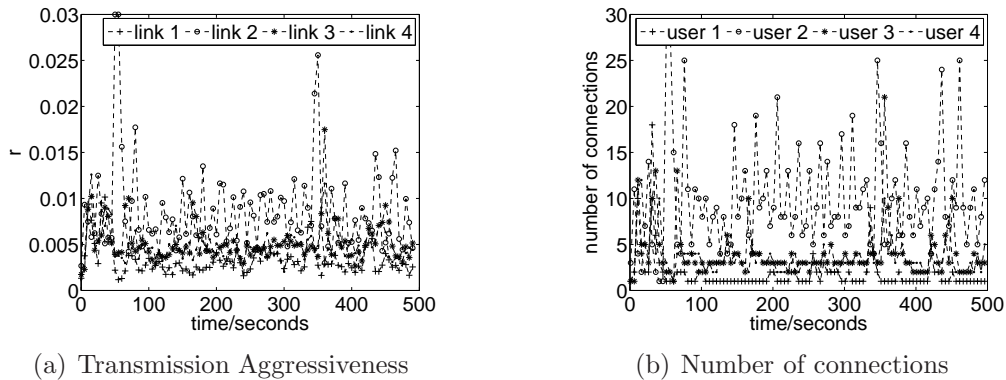


Figure 6.3: Simulation results for topology *a* under multi-connection TCP Reno over A-CSMA with AQM when **h-ACK** is used: curves of transmission aggressiveness and number of connections.

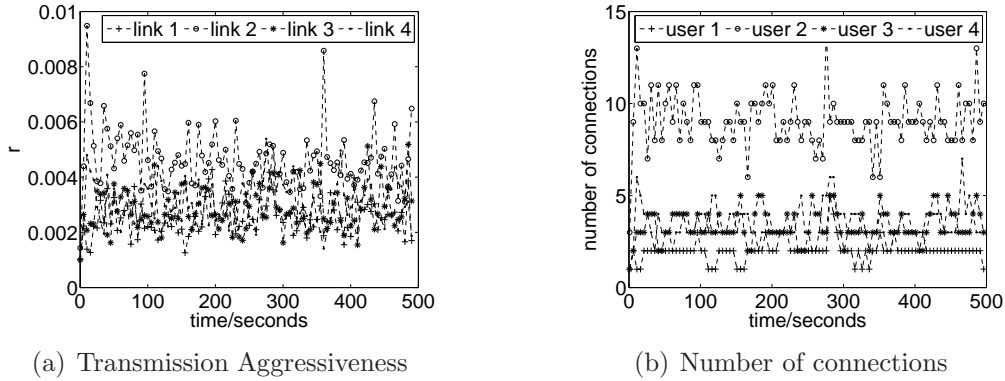


Figure 6.4: Simulation results for topology *a* under multi-connection TCP Reno over A-CSMA with AQM when **n-ACK** is used: curves of transmission aggressiveness and number of connections.

6.1.2 Impact of Measuring Queue Length in Number of Bytes for n-ACK

In Chapter 5.3.3, we discussed two types of queue length measuring methods in n-ACK, one is in number of bytes, the other is in number of packets. In our analysis, if all links' queue length are measured in bytes of data, the throughput degrades. We validate our analysis in NS-2 simulation. Fig. 6.1.2 plots the throughput in topology *a* by n-ACK with queue length measured in number of bytes and packets respectively. The overall throughput is 579 packets/s by queue length measured in number of bytes, and 595 packets/s by queue length measured in number of packets. The simulation results agree with our analysis that queue length measured in number of bytes gets lower throughput. However, the throughput degradation is not significant. Because backward link accumulate enough ACK packets, which makes the backward link get comparable transmission aggressiveness. RTT will increase due to too many ACK accumulated in backward link, and thus the number of connections will increase. These can be observed from Fig. 6.6, 6.7 and 6.8. Simulation results shows that the number of packets in backward link queue length is 15 times larger than that of forward link. Therefore, the number of connections

sharply increases to 100. The overhead will be significant, whereas measuring queue length in number of packets does not suffer such problem.

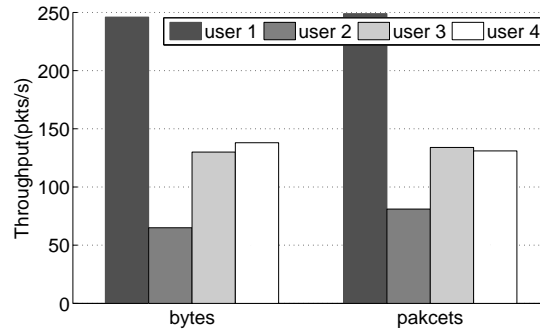


Figure 6.5: Simulation throughput in topology *a* by n-ACK with queue length measured in number of bytes and number of packets.

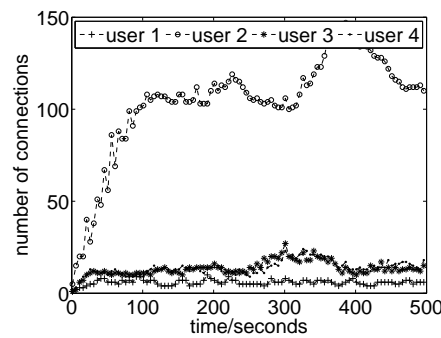


Figure 6.6: Number of connections in topology *a* by n-ACK with queue length measured in number of bytes and number of packets.

6.1.3 Impact of Dummy Packets

In Algorithm 1, when the queue of a link is empty, it generates a dummy packet to saturate the traffic so that the CSMA Markov chain is satisfied. In practice, however, transmitting dummy packets consumes precious bandwidth and causes overall throughput degradation.

We validate it through NS-2 simulations. The simulation throughput in topology *a* by h-ACK and n-ACK with/without dummy packets are plotted in Fig.6.1.3. The simulation results agree with our analysis that throughput

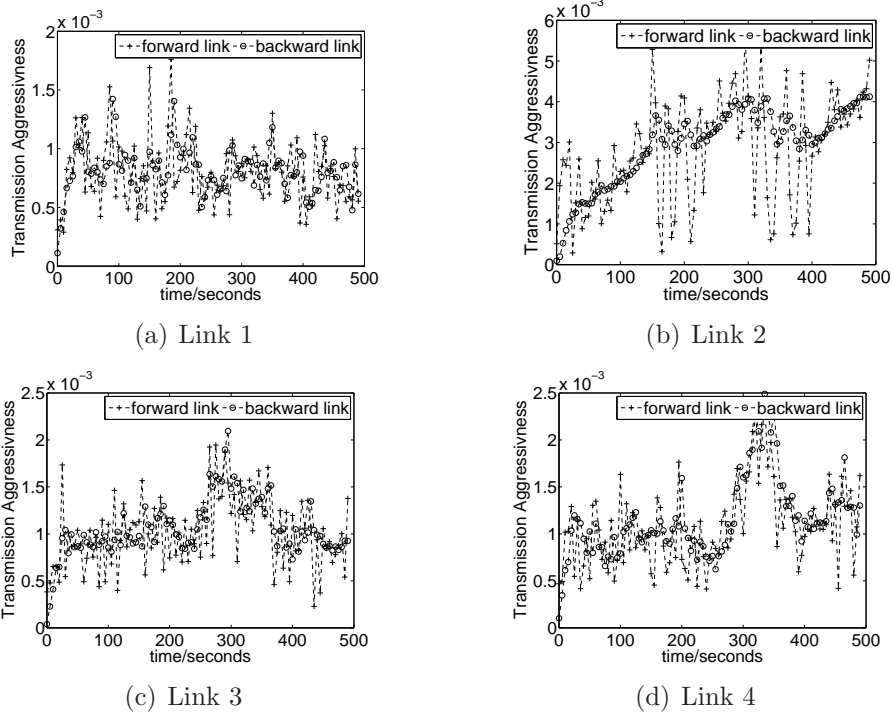


Figure 6.7: Transmission Aggressiveness by n-ACK with queue length measured in number of bytes.

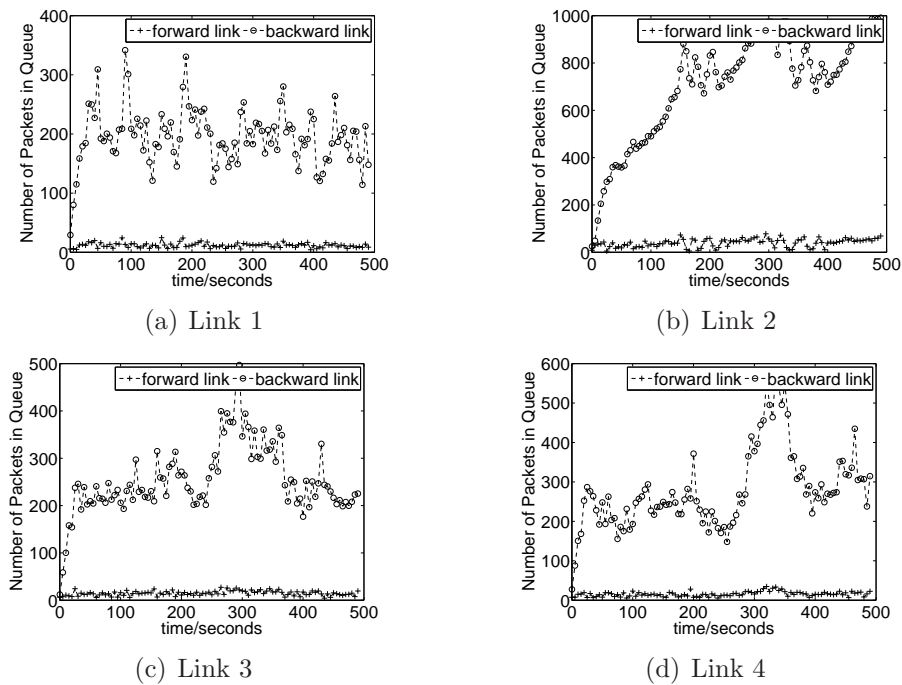


Figure 6.8: Number of packets in queue by n-ACK with queue length measured in number of bytes.

performance degrades with dummy packets, but not significant. The reason is that all links are saturated with traffic, very few dummy packets are generated due to empty queue. Hence, the dummy packet has little effect on throughput. In practice, links are not always saturated. This will lead to a queue of the link is empty and produce a large number of dummy packets, leading to decreased throughput. So in actual deployment, dummy packets function can be removed.

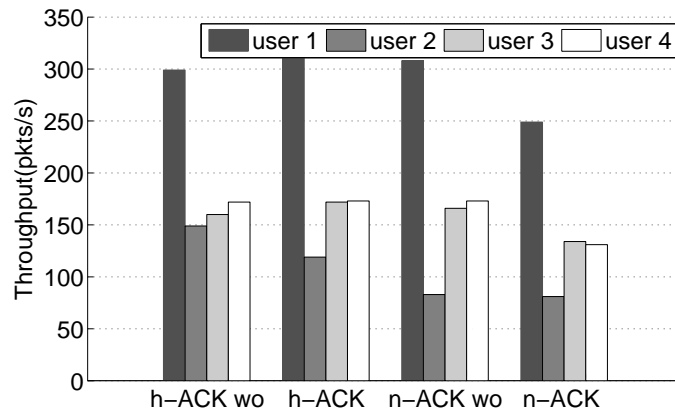


Figure 6.9: Simulation throughput in topology *a* by: a) h-ACK without dummy packets, b) h-ACK with dummy packets, c) n-ACK without dummy packets and d) n-ACK with dummy packets.

6.1.4 Impact of Product $k^2\beta$

We validate that the system performance depends only on $k^2\beta$ through simulation in this section. The conflict graph of simulation topology under study is still the one shown in Fig. 6.1. We choose a set of parameters but $k^2\beta = 20000$ keeps constant and these parameters are listed in the Fig. 6.10.

The throughput by simulations are plotted in Fig. 6.10. When $k = 28.3$, $k = 20$, $k = 14$, and $k = 10$, the throughput distribution achieved are very similar, which verifies our analysis that the optimal solution only depends on $k^2\beta$. However, we also observe that the throughput distribution achieved by $k = 44.7, \beta = 10$ and $k = 5, \beta = 800$ are different from those obtained

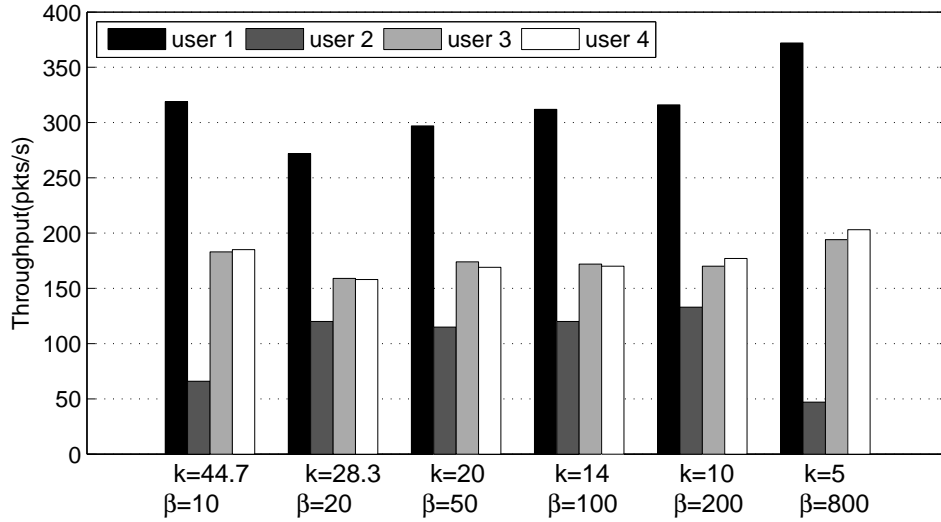


Figure 6.10: Throughput distribution by different k and β when the product $k^2\beta$ is fixed.

under other settings, even though the product $k^2\beta$ is the same. We discuss the problem from two aspects.

When k is large, we find that the performance difference is caused by large packet loss ratio. Because the number of connections is very large when k is large, the queue length accumulates dramatically. The packet loss ratio is proportional to queue length, thus performance loss is very large. In simulation, we observe when $k = 44.7$, the number of loss packets by all users are [5, 16, 13, 13]pkts/s. But when $k = 20$, the number of loss packets is [1, 3, 2, 2]pkts/s. This observation verifies that the throughput distribution difference is caused by large loss ratio.

When k is small, the performance difference is caused by the coarse granularity in number of connections. In this case, when $k = 5$, the average number of connections opened by all users are [1, 4, 1, 1]. The number of connections opened by user 3 and user 4 should be larger than that opened by 1, due to small k , the number of connections opened by user 1 should at least be one. Therefore, it results in the solution difference.

6.1.5 Effects of Parameter β

In our analysis that large β can produce better performance but too large β leads to short-term fairness problem. We conduct simulations to study this effects. To observe the short-term fairness clearly, we plot curves of the first 20s. In the simulation, we fixed all other parameters except β . We plot the aggregate packets the link transmits upon each time by different β in Fig. 6.11. The line slope indicates the instant throughput. If the line slope is zero, it implies the throughput at this moment is zero. Therefore, the short-term starvation can be observed from the line slope. If the line slope stays zero for a long interval, it implies the user experience starvation.

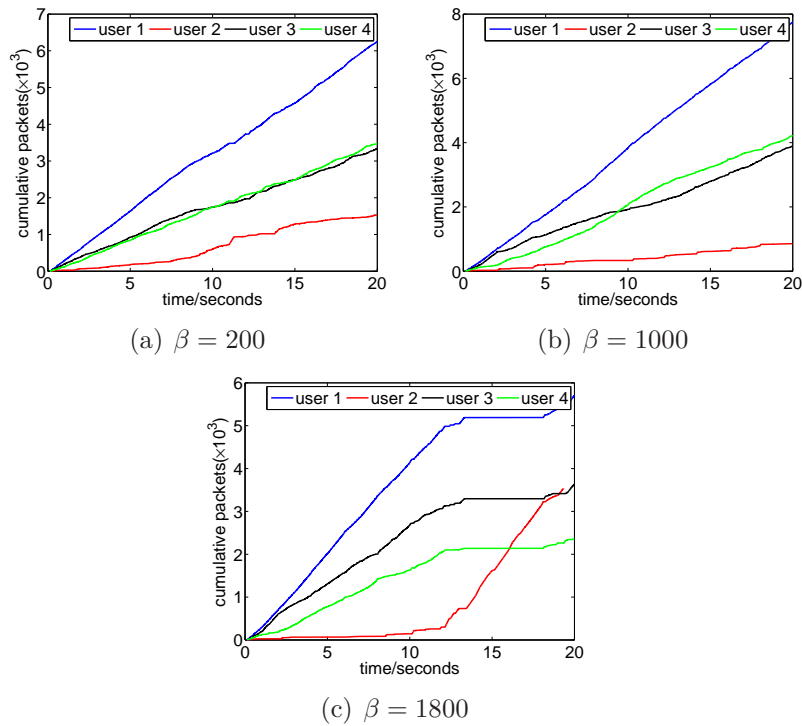


Figure 6.11: Cumulative packets by different β .

We observe that when $\beta = 200$, the curves of user 3 and 4 almost coincide, and all users' line slope are not equal to zero. User 3 and 4 transmit almost equal number of packets all the time. So users do not suffer short-term fairness because the cumulative packets of all users keep increasing. When $\beta = 1000$,

we observe that user 3's cumulative packets is larger than user 4's in the first 9 seconds, while in the next 10 seconds, user 4's exceeds user 3's cumulative packets. In the long run, their throughput will be equal on average. This short-term starvation is caused by using large β . When $\beta = 1800$, user 2's cumulative packets is constant in the first ten seconds, which implies user 2 suffers short-term starvation. From the 13 ~ 18 seconds, users 1, 3 and 4's cumulative packets stay constant, while user 2's cumulative packets keep increasing. Because user 2 keeps occupying the channel, other users are unable to transmit. Therefore, large β can result in short-term fairness problem. For example, without loss of generality, we assume user 1 and user 2 have relatively equal transmission aggressiveness, i.e., $\frac{r_1}{r_2} \approx 1$, and $r_1 < r_2$. The value of $\frac{\exp(-\beta r_1)}{\exp(-\beta r_2)}$ can be very large if β is large. Note $\exp(-\beta r_1)$ is the average waiting time of link 1. Therefore, this causes that link 1 (with smaller r) waits very long time. The link 2 (with larger transmission aggressiveness) can transmit a great number of packets before link 1 starts transmission. It leads to the short-term starvation problem when β is very large.

However, with large β , the system performance is better. We list the overall throughput and utility gap by different β in the Table.6.3. When $\beta = 1800$, the utility and overall throughput are the largest. Both the overall throughput and utility gap can be improved by using large β .

	$\beta = 200$	$\beta = 1000$	$\beta = 1800$
Overall Throughput	737	821	841
Utility Gap	-4.1	-2.6	-1.0

Table 6.3: Overall throughput and Utility gap under different β

With these observations, although large β leads to better throughput and utility, it results in short-term fairness problem and slow convergence. In practical, to choose a proper β that can achieve good performance and low short-term fairness is important.

6.1.6 Effects of Parameter k

We conduct simulations to study the tradeoff between performance and overhead in Chapter 5.3.4. The parameter β is fixed and set to 200. So the parameter k increases, the weight of entropy term increases, which theoretically gives better performance. We compare the simulation throughput and overhead in different k . The throughput by simulation are shown in Fig. 6.1.6 and the evolution of transmission aggressiveness and number of connections are plotted in Fig. 6.13.

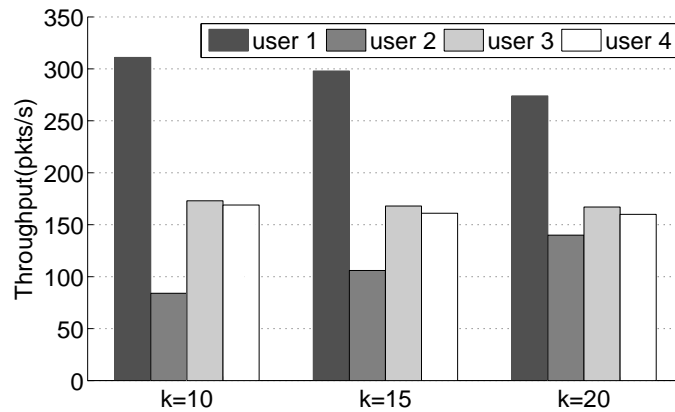


Figure 6.12: Simulation throughput in topology a when: a) $k = 10$, b) $k = 15$, and c) $k = 20$.

We observe that user 2 transmits more packets when $k = 20$, therefore, the solution achieve fairer throughput distribution for large k . To examine the performance, we compute the utility gap for different k and list them in the Table.6.4. The overall throughput is almost the same for all k , which is around 735 packets/s. The utility gap when $k = 20$ is the largest among all other k , while utility gap when $k = 10$ is the smallest. It verifies that the system performance is good when using large k .

However, we also notice from Fig. 6.13, larger k results in more number of connections. When $k = 10$, the average number of connections is around 9, while when $k = 20$, it is around 26. This implies the overhead increases with large k . From analysis of previous chapter, more connections implies

	$k = 10$	$k = 15$	$k = 20$
Overall Throughput	737	733	741
Utility Gap	-4.1	-3.1	-2.1

Table 6.4: Overall throughput and Utility gap under different k

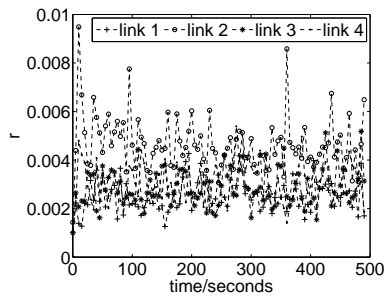
more packets generated, so transmission aggressiveness is large. This also can be seen from Fig. 6.13. When $k = 10$, all links' transmission aggressiveness are around 0.002, 0.005, 0.003 and 0.003 respectively. When $k = 15$, all links' transmission aggressiveness are around 0.002, 0.006, 0.004 and 0.004. While $k = 20$, all links transmission aggressiveness increases to a relatively large value, 0.002, 0.010, 0.005 and 0.005. The simulation results agree with our analysis, that the transmission aggressiveness increases as k increases. Large transmission aggressiveness leads to links transmit more aggressively and therefore higher throughput. This is another reason why large k can produce better throughput.

Hence, there is a tradeoff between small number of connections and better throughput. However, when k doubles, the throughput improvement is not significant, while the number of connections increases drastically. So it is not always suitable to sacrifice the overhead for throughput performance.

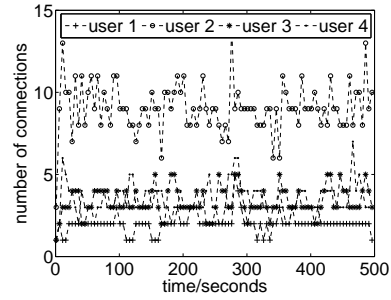
6.1.7 Overhead of n-ACK Solution

In this subsection, we examine the overhead of our solution when n-ACK is used. The results show that the number of connections opened by user 2 is roughly 20 and by other users are less than 10 connections. The port number used is much smaller than the total available port number, so multi-connection TCP has less impact on the insufficient port number problem.

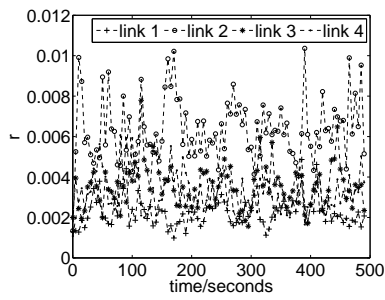
The average r measured is $[3.7, 9.4, 4.5, 4.9] * 10^{-3}$, so overhead of retransmitting can be computed by $(Thr \cdot r_l)$. The vector of number of retransmitted packets is $[1.1, 1.4, 0.7, 0.8]$ packets/s. All users drop very few packets seconds,



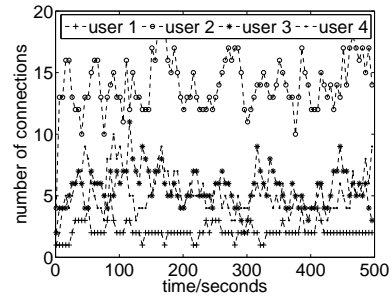
(a) $k = 10$



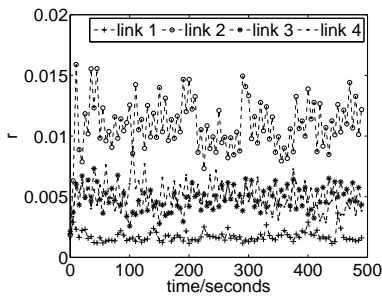
(b) $k = 10$



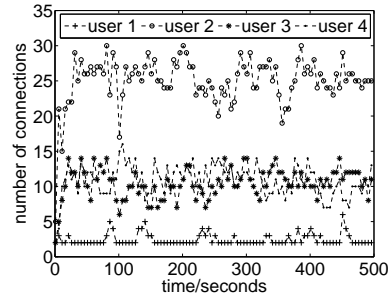
(c) $k = 15$



(d) $k = 15$



(e) $k = 20$



(f) $k = 20$

Figure 6.13: Transmission aggressiveness and number of queue length under different k .

this indicates our solution has very less overhead on retransmission.

We characterize the CPU overhead via the efficiency η defined in the Chapter 5.3.5. In a PC with Intel P4 processor 3.2Ghz, 1Mb L2 cache and 1Gb SDRAM, T_e (the duration to open a TCP Reno connection) is $418\mu s$, T_c (the duration to close a connection) is $12\mu s$. In our simulation, the update interval of n_s denoted by d is 5s, the packet payload is 1000bytes and T_p is $14\mu s$. In topology a , from Eq. (5.50), we give the efficiency of all flows in topology a in Table. 6.5. From the results, the efficiencies of all users are above 94%, which means among all the CPU resource needed in the solution, 6% of which is used for managing connections and 94% is used for data transmitting. User 1 even have 98.8% efficiency which is the highest among all users, the reason is that the curve of connection numbers n_s is not too dynamic, which can be seen from Fig. 6.4(b). On the contrary, user 2 has the highest overhead due to its n_s is relatively dynamic, which proves our analysis in Chapter 5.3.5. Our solution has very small connection management CPU overhead from these results.

	user 1	user 2	user 3	user 4
Efficiency η	98.8%	94.0%	97.5%	97.8%

Table 6.5: Efficiency of all flows in topology a when n-ACK is used

6.2 Multihop Wireless Networks Scenario

In this part, we consider multiple hop networks scenario and n-ACK is used. The simulated topology and its associated conflict graph under study is shown in Fig. 6.14(a). It is a nine-node wireless network with six links represented by dashed line. There are three user flows whose paths are represented by solid lines.

Fig. 6.14(b) plots the optimal throughput and throughput achieved by multi-connection TCP Reno scheme and TCP Reno over L-CSMA. The middle

TCP flow starves when running TCP Reno over L-CSMA. In contrast, multi-connection TCP Reno over A-CSMA with AQM scheme assigns 86 packets to middle TCP flow, which is within 24% of the optimal achievable rate for this topology.

Fig. 6.14(c) and 6.14(d) give the evolution of transmission aggressiveness r and number of connection n_s . They are both stable in this scenario. The average number of connections around 20. And link transmission aggressiveness is below 0.01, which indicates the packet loss ratio is very small.

The results indicate that our multi-connection TCP over A-CSMA with AQM scheme is also applicable for multiple hop networks scenario.

6.3 Multihop Networks with Wireless and Wired Links Scenario

In this subsection, we validate our multi-connection TCP Reno solution for multihop networks with wired and wireless links scenario. The simulated topology under study is shown in Fig. 6.15(a). It is a nine-node wireless network with two wireless APs denoted by node d and node h . Wireless client nodes a , b and c are associated with AP d while wireless client node i is associated with AP h . Nodes e , f and g are wired nodes. In the graph, wireless links are represented by dashed line and wired links are represented by solid line. There are three user flows whose paths are from node a to node g , node b to node c and node i to g respectively. The wired link f -to- g has the link capacity of 2Mb and other wired links have the link capacity of 20 Mb. So the wired bottleneck is on the link f -to- g . The wireless link bandwidth is 11 Mb.

Fig. 6.15(b) plots the simulation throughput when n-ACK is used and optimal throughput. We observe TCP Reno does not suffer server starvation over L-CSMA in this topology. However, the utility gap of our solution is -0.9

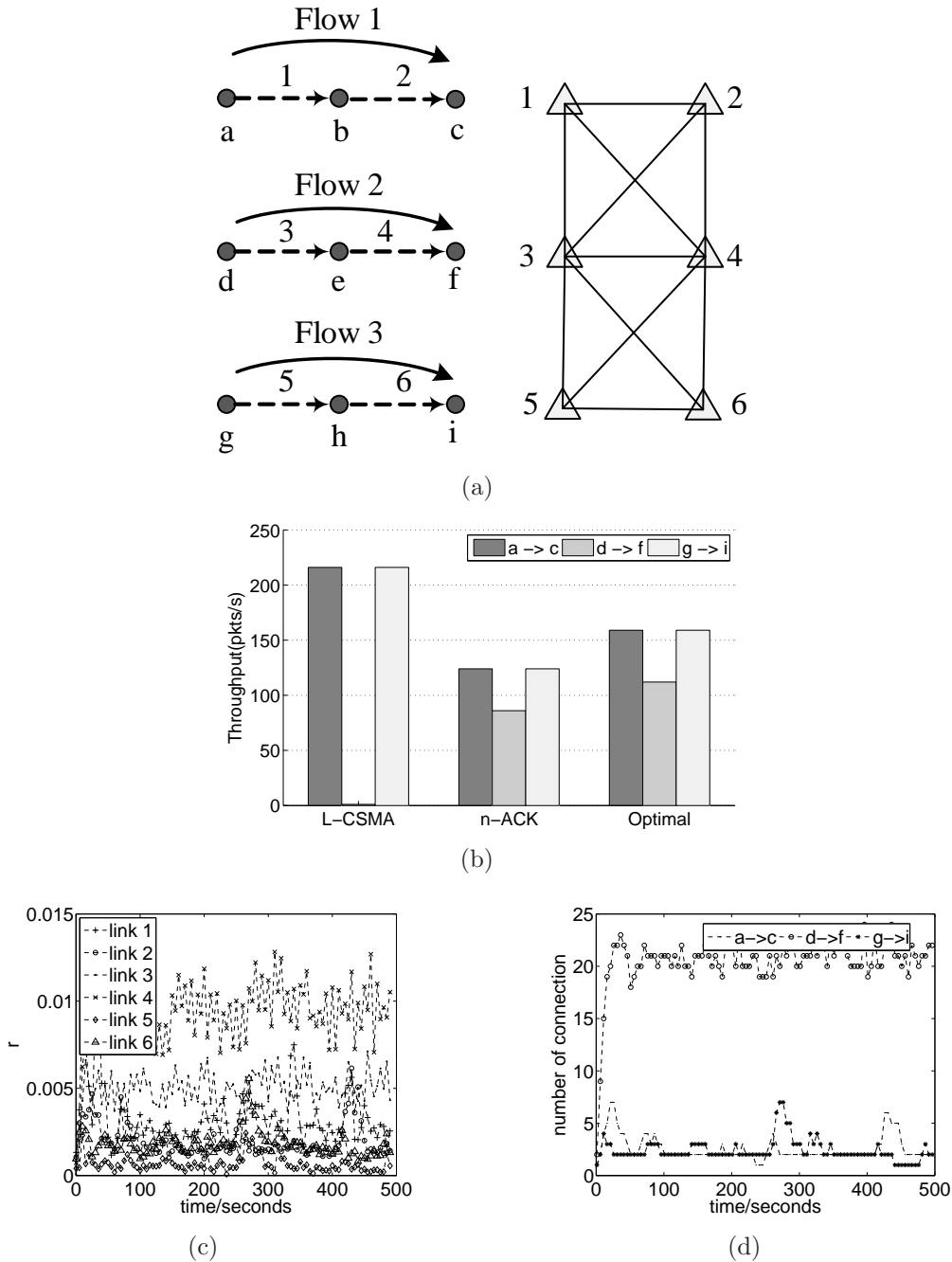
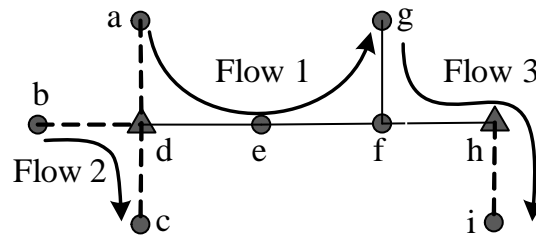
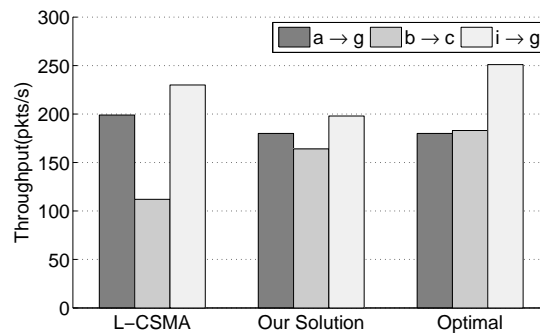


Figure 6.14: Multihop networks scenarios: (a) topology d and its associated conflict graph. (b) throughput (c) curves of transmission aggressiveness. (d) curves of number of connections.



(a)



(b)

Figure 6.15: Multihop networks with wired and wireless links scenario: (a) topology *e* (b) throughput

which is closer to zero than that of TCP Reno over L-CSMA, whose utility gap is -1.8 . This result validates our solution has better performance than that of TCP Reno over L-CSMA. The results validate multi-connection TCP Reno can work properly in multihop networks with wired and wireless links.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this chapter, we conclude our work and give the further work. In this thesis, we study whether the widely-installed TCP Reno is compatible with adaptive CSMA. We find that running TCP Reno directly over adaptive CSMA suffers from the same severe starvation problems as TCP Reno over legacy CSMA (IEEE 802.11 DCF). The reason the potentials of adaptive CSMA cannot be realized by TCP Reno is that the overall system is not stable.

We then propose a multi-connection TCP Reno solution to inter-work with A-CSMA in a compatible manner. By adjusting the number of TCP connections for each session, our solution can achieve arbitrary system utility. Simulation results also indicate our solution has low overhead.

Our solution can be implemented at the application layer or by inserting an intermediate layer. Intermediate layer implementation requires no modification to today's applications, and the solution can be readily deployed in networks running adaptive CSMA. Finally, we show that our solution is applicable to single-hop wireless networks as well as multihop networks with wired and wireless links.

7.2 Future Work

In the practical situation, the wireless channel is inclined to suffer channel errors. For example, multiple TFRC connections for streaming video [7] and multiple TCP Reno connections for data and multimedia streaming [6,27] have been discussed to solve this link error problem. So in a A-CSMA network with both interference and packet loss due to channel errors, our future work is to find a solution to solve both problems in one shot.

Looking into the future, it would be interesting to implement our proposed scheme into real systems and evaluate its performance over the testbed running A-CSMA [14].

Appendix A

Explanation to Starvation of TCP Reno over A-CSMA

We give theoretical reason why TCP Reno over A-CSMA network results in starvation problem. In the example stated in Chapter 4.2, there are four TCP Reno sessions $s = 1, 2, 3, 4$, each running over a wireless link $l = 1, 2, 3, 4$. With single connection TCP running over A-CSMA wireless networks, the dynamic system of TCP Reno over A-CSMA networks is:

$$\begin{cases} \dot{x}_s = \frac{x_s^2}{2} \left(\frac{2}{T_s^2 x_s^2} - p_s \right)_{x_s}^+, \quad \forall s = 1, 2, 3, 4; & (\text{A.1}) \\ \dot{r}_s = \alpha \left(x_s - \sum_{i:s \in i} \tau_i(\mathbf{r}) \right)_{r_s}^+, \quad \forall s = 1, 2, 3, 4; & (\text{A.2}) \end{cases}$$

where

$$p_s = \frac{(x_s - \sum_{i:s \in i} \tau_i(\mathbf{r}))^+}{x}, \quad (\text{A.3})$$

$$\tau_i(\mathbf{r}) = \frac{\exp(\sum_{s \in i} \beta r_s)}{\sum_{i' \in \mathcal{I}} \exp(\sum_{s \in i'} \beta r_s)}, \quad \forall i \in \mathcal{I}, \quad (\text{A.4})$$

and p_x is packet loss ratio that user s experiences. When there is no packet loss on user s , we have $x_s = \sum_{i:s \in i} \tau_i(\mathbf{r})$. Thus, p_s is non-negative and it does not equal to zero value, otherwise, according to Eq.(A.1), $\dot{x}_s > 0$ and rate x_s will keep increasing. Therefore, we have $x_s > \sum_{i:s \in i} \tau_i(\mathbf{r})$. In this situation,

r_s diverge to infinity because $\dot{r}_s > 0$ according to Eq. (A.2). So the dynamic system by TCP Reno over A-CSMA does not have an stable equilibrium.

The divergence of TCP Reno over A-CSMA leads to starvation problem. Because r will diverge to infinity, we set an upper bound on r , denoted by r_{max} (for the digital system cannot assign a too small back-off time which is $\exp(-\beta r)$). The consequence is that r of all users increases to r_{max} . Thus, with all links with equal $r = r_{max}$, TCP Reno over A-CSMA reduces to TCP Reno over L-CSMA and results in the starvation problem.

This analysis explanation can be justified by the simulation results in Fig. 4.3, where all users' r keeps increasing until hitting r_{max} .

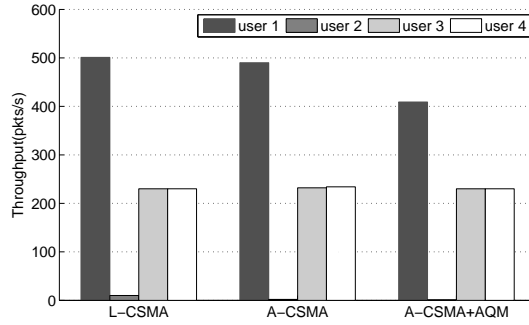
Appendix B

TCP Reno over A-CSMA with AQM

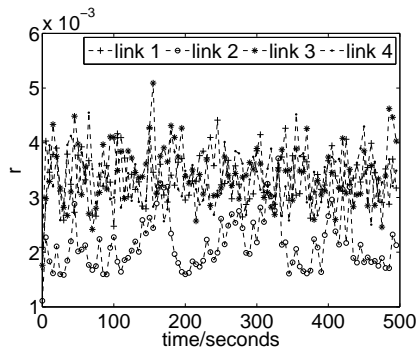
In this single TCP Reno over A-CSMA with AQM solution, the only modification needed is each transmitter of link l applies an AQM policy that each link drops the incoming packet with probability proportional to r_l . We study the performance of the solution and give the explanation in this part.

B.1 TCP Reno starves

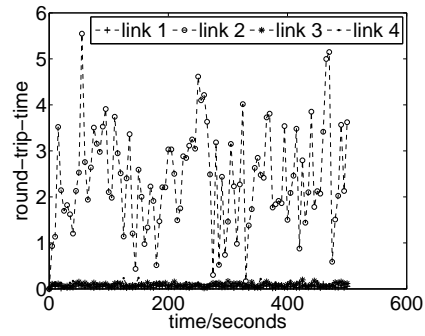
We conduct an NS-2 simulation to demonstrate the problem. The simulation topology and setup are the same as that in Fig. 4.1. But each link l drops incoming packets with probability proportional to r_l . We also give the backward links higher priority. The simulation results are shown in Fig. B.1(a), from which we can see that user 2 starves. Fig. B.1(b) and B.1(c) show the transmission aggressiveness of the links, and RTT of TCP Reno sessions, respectively. We observe that all links have closely equal transmission aggressiveness. Under this situation, by now it should be clear that link 2 will starve.



(a) Throughput by L-CSMA: TCP over L-CSMA, A-CSMA: TCP over A-CSMA, and A-CSMA + AQM: TCP over A-CSMA with AQM policy



(b) Transmission Aggressiveness



(c) Round-trip-time

Figure B.1: Simulation Results: TCP Reno over A-CSMA with AQM.

B.2 Explanation

We present the analysis as follows. In this example, there are four TCP Reno sessions $s = 1, 2, 3, 4$, each running over a wireless link $l = 1, 2, 3, 4$. With single connection TCP running over A-CSMA with AQM wireless networks, the algorithms in (5.2)-(5.3) turns into:

$$\begin{cases} \dot{x}_s = \frac{x_s^2}{2} \left(\frac{2}{T_s^2 x_s^2} - r_s \right)_{x_s}^+, \quad \forall s = 1, 2, 3, 4; & \text{(B.1)} \\ \dot{r}_s = \alpha \left[x_s - \sum_{i:s \in i} \tau_i(\mathbf{r}) \right]_{r_s}^+, \quad \forall s = 1, 2, 3, 4, & \text{(B.2)} \end{cases}$$

where

$$\tau_i(\mathbf{r}) = \frac{\exp(\sum_{s \in \mathcal{I}} \beta r_s)}{\sum_{i' \in \mathcal{I}} \exp(\sum_{s \in \mathcal{I}} \beta r_s)}, \quad \forall i \in \mathcal{I}, s \quad (\text{B.3})$$

$$T_s = \frac{1}{\alpha} \frac{r_s}{x_s}, \quad \forall s = 1, 2, 3, 4. \quad (\text{B.4})$$

The first equation is the TCP Reno updating equation, as discussed in (B.1). The second correspond to A-CSMA transmission aggressiveness adjusting equation.

The Eq. (B.4) characterizes the RTT user s experiences. As we mentioned in the simulation settings, to simplify the analysis, we only consider the forward link by giving the TCP ACK higher priority to transmit. Therefore, the waiting time of TCP ACK is very short. Most of the packets are buffered at the queue of forward link. Thus, $W_s \approx Q_s$ (recall W_s is the congestion window size of TCP Reno session of user s), and round trip time T_s can be written as

$$T_s = \frac{W_s}{x_s} \approx \frac{Q_s}{x_s}, \quad (\text{B.5})$$

where Q_s is the queue length of link s used by user s . According to A-CSMA, the transmission aggressiveness is proportional to the queue length. Thus we obtain (B.4):

$$T_s = \frac{Q_s}{x_s} = \frac{1}{\alpha} \frac{r_s}{x_s}, \quad \forall s = 1, 2, 3, 4. \quad (\text{B.6})$$

The RTT equation reveals a subtle interaction between TCP Reno and A-CSMA, which in fact causes the starvation. We explain it as below.

At the equilibrium of the dynamic system described in (B.1)-(B.2), the derivatives are all zero. Hence,

$$r_s = \frac{2}{T_s^2 x_s^2}, \quad \forall s = 1, 2, 3, 4. \quad (\text{B.7})$$

Solving (B.6) and (B.7), we arrive at the following observations:

$$r_s = 2^{1/3} \alpha^{2/3}, \quad \forall s = 1, 2, 3, 4. \quad (\text{B.8})$$

Thus every link s has the same r_s and competes the channel with the same level of transmission aggressiveness. Under this situation, our previous discussions

indicate that links 1, 3 and 4 will take turns to freeze link 2, and hence the TCP Reno over link 2 will starve.

Bibliography

- [1] B. Baker. Approximation algorithms for np-complete problems on planar graphs. *Journal of the ACM*, pages 153–180, 1994.
- [2] R. Boorstyn, A. Kershenbaum, B. Maglaris, and V. Sahin. Throughput analysis in multihop CSMA packet radio networks. *IEEE Transactions on Communications*, pages 267–274, 1987.
- [3] C. Chau, M. Chen, and S. Liew. Capacity of large-scale CSMA wireless networks. In *Proceedings of MobiCom*, pages 97–108, 2009.
- [4] L. Chen, S. Low, and J. Doyle. Joint congestion control and media access control design for ad hoc wireless networks. In *Proceedings of IEEE INFOCOM*, 2005.
- [5] M. Chen, S. Liew, Z. Shao, and C. Kai. Markov approximation for combinatorial network optimization. In *Proceedings of IEEE INFOCOM*, 2010.
- [6] M. Chen and A. Zakhor. Flow control over wireless network and application layer implementation. In *Proceedings of IEEE INFOCOM*, 2006.
- [7] M. Chen and A. Zakhor. Multiple TFRC connections based rate control for wireless networks. *IEEE Transactions on Multimedia*, pages 1045–1062, 2006.
- [8] D. Clark and W. Fang. Explicit allocation of best-effort packet delivery service. *IEEE/ACM Transactions on Networking*, pages 362–373, 1998.

- [9] S. Floyd and V. Jacobson. Random early detection gateways for congestion control. *IEEE/ACM Transactions on Networking*, pages 397–412, 1993.
- [10] K. Jain, J. Padhye, V. Padmanabhan, and L. Qiu. Impact of interference on multi-hop wireless network performance. *Wireless Networks*, pages 471–487, 2005.
- [11] L. Jiang and S. Liew. Improving throughput and fairness by reducing exposed and hidden nodes in 802.11 networks. *IEEE Transactions on Mobile Computing*, pages 34–49, 2008.
- [12] L. Jiang and J. Walrand. A distributed CSMA algorithm for throughput and utility maximization in wireless networks. In *Proceedings of Communication, Control, and Computing*, pages 1511–1519, 2008.
- [13] L. Jiang and J. Walrand. Convergence and stability of a distributed CSMA algorithm for maximal network throughput. *UC Berkeley, Mar*, pages 2009–43, 2009.
- [14] J. Lee, J. Lee, Y. Yi, S. Chong, A. Proutiere, and M. Chiang. Implementing Utility-Optimal CSMA. In *Proceedings of Allerton Conference*. Citeseer, 2009.
- [15] S. Liew, C. Kai, J. Leung, and B. Wong. Back-of-the-envelope computation of throughput distributions in CSMA wireless networks. *IEEE Trans Mobile Computing*, 2010.
- [16] X. Lin and N. Shroff. Joint rate control and scheduling in multihop wireless networks. In *IEEE CDC*, 2004.
- [17] J. Liu, Y. Yi, A. Proutiere, M. Chiang, and H. Poor. Towards utility-optimal random access without message passing. *Wireless Communications and Mobile Computing*, pages 115–128, 2010.

- [18] J. Mo and J. Walrand. Fair end-to-end window-based congestion control. *IEEE/ACM Transactions on Networking*, pages 556–567, 2000.
- [19] J. Ni, B. Tan, and R. Srikant. Q-CSMA: Queue-Length based CSMA/CA algorithms for achieving maximum throughput and low delay in wireless networks. In *IEEE INFOCOM Mini-Conference*, 2010.
- [20] J. Postel. Transmission control protocol: DARPA Internet program protocol specification. *RFC 793, IETF*, 1981.
- [21] S. Rajagopalan and D. Shah. Distributed algorithm and reversible network. In *Proceedings of CISS*, pages 498–502, 2008.
- [22] S. Rangwala, A. Jindal, K. Jang, K. Psounis, and R. Govindan. Neighborhood-centric congestion control for multi-hop wireless mesh networks. *IEEE/ACM Transactions on Networking*, 2009.
- [23] S. Shakkottai and R. Srikant. Network optimization and control. In *Foundations and Trends in Networking*, pages 271–379, 2007.
- [24] Z. Shao, M. Chen, A. S. Avestimehr, and S. Li. Cross-layer Optimization for Wireless Networks with Deterministic Channel Models. In *Proceedings of IEEE INFOCOM*, 2010.
- [25] K. Tan, F. Jiang, Q. Zhang, and X. Shen. Congestion control in multihop wireless networks. *IEEE Transactions on Vehicular Technology*, pages 863–873, 2007.
- [26] A. Tang, J. Wang, S. Low, and M. Chiang. Equilibrium of heterogeneous congestion control: Existence and uniqueness. *IEEE/ACM Transactions on Networking*, page 837, 2007.

- [27] S. Tullimas, T. Nguyen, R. Edgecomb, and S. Cheung. Multimedia streaming using multiple TCP connections. *ACM TOMCCAP*, pages 1–20, 2008.
- [28] X. Wang and K. Kar. Throughput modelling and fairness issues in CSMA/CA based ad-hoc networks. In *Proceedins of IEEE INFOCOM*, 2005.