Transport Layer Fairness and Congestion Control in Wireless Ad Hoc Access Networks

By

Hao Zhang

A thesis submitted to
The Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of

Master of Applied Science

Ottawa-Carleton Institute of Electrical and Computer Engineering

Department of Systems and Computer Engineering
Carleton University
Ottawa, Canada

July 2006

The undersigned recommend to the Faculty of Graduate Studies and Research

acceptance of the thesis

Transport Layer Fairness and Congestion Control in Wireless Ad Hoc Access Networks

Submitted by **Hao Zhang**, M.A.Sc

In partial fulfillment of the requirements for
the degree of Master of Applied Science

Dr. Thomas Kunz
Thesis Supervisor

Chair, Department of Systems and Computer Engineering

Carleton University

July 2006

ABSTRACT

Transmission Control Protocol (TCP) is a reliable, end-to-end transport protocol, which is most widely used for data services and is very efficient for wired networks. However, experiments and research showed that TCP's congestion control algorithm performs very poorly over Wireless Ad Hoc Networks with degraded throughputs and severe unfairness among flows.

This thesis studies fairness and throughput issues presented by TCP in Wireless Ad Hoc Access Networks, which are Wireless Ad Hoc Networks with supporting infrastructure, i.e. gateway, to send and receive packets from wired networks, and concentrates on designing an applicable congestion control algorithm based on the characteristics of the Wireless Ad Hoc Access Networks. We proposed using DCCP (Datagram Congestion Control Protocol) with a specially designed congestion control algorithm and implemented this congestion control algorithm on NS2. Simulations were performed and the results show the improvements on fairness and throughput achieved by using the designed congestion control algorithm. The two flows with the new congestion control tested in the simulation had Jain's fairness index greater than 0.95 in all combinations, where TCP flows may have a fairness index less than 0.6. The aggregate throughputs of the two flows with our new congestion control algorithm also increased.

ACKNOWLEDGEMENT

It is a great honor for me to be supervised by Professor Thomas Kunz. I cannot fully express my gratitude to the continuous, superb guidance and valuable comments Professor Kunz had given to me during my study and thesis preparation. I will never forget, not only his knowledge, which guided me during my study, but also his great personality and positive attitudes towards work and life.

I would like to thank my family, especially my husband, for their understanding, support, and pressure-free encouragements during my study, thanks for always being there for me.

Table of Contents

List of Figures

List of Tables

List of Appendices

List of Acronyms

ACK - Acknowledgement

AIMD – Additive Increase and Multiplicative Decrease

ATP – Ad Hoc Transport Protocol

ECN – Explicit Congestion Notification

BER – Bit Error Rate

CCID – Congestion Control Identifier

CSMA/CA – Carrier Sense Multiple Access/Collision Avoidance

CTS – Clear To Send

DCCP – Datagram Congestion Control Protocol

DCF – Distributed Coordination Function

DIFS – DCF Inter Frame Space

ECN – Explicit Congestion Notification

ELFN – Explicit Link Failure Notification

FIFO – First In First Out

ICMP – Internet Control Message Protocol

IEEE – Institute of Electrical and Electronics Engineers

IP – Internet Protocol

MAC – Media Access Layer

MANET – Mobile Ad-Hoc Network

MSDU – MAC Service Data Unit

NS 2 – Network Simulator (version 2)

OOO – Out-of-Order

RTO	Retransmission Time Out
RTS	Request to Send
RTT	Round Trip Time
SACK	Selective ACK
SIFS	Short Inter Frame Space
SLoPS	Self Loading Periodic Stream
TCP	Transmission Control Protocol
TCP-AP	TCP with Adaptive Pacing
TOPP	Trains of Packet Pairs
TIBET	Time Interval based Bandwidth Estimation Technique
UDP	User Datagram Protocol
WLAN	Wireless Local Access Network

Chapter 1

Introduction

Wireless Ad Hoc Networks are multi-hop wireless networks. A Wireless Ad Hoc Network consists of mobile platforms (e.g., a router with multiple hosts and wireless communication devices) – herein simply referred to as “nodes” – which are free to move about arbitrarily [8]. Wireless Ad Hoc Networks can be easily deployed with or without the support of existing infrastructure. Figure 1-1 is an example Ad Hoc network which uses gateways as the connection between the wired and wireless parts, which is referred to as a Wireless Ad Hoc Access Networks in the thesis. With the easy deployments, Wireless Ad Hoc Networks meet the requirements of many applications, such as in military battlefield and emergencies, etc. Extensive research has been conducted concerning media access, routing and transport protocols for such networks. Transport layer protocols, which are specifically modified or designed for Wireless Ad Hoc Access Networks, are discussed in this thesis.

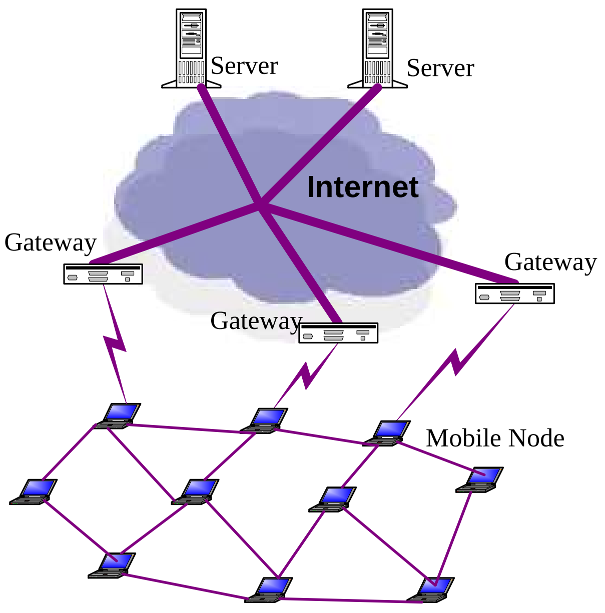


Figure 1-1 An Example Wireless Ad Hoc Access Network

Wireless Ad Hoc Access networks use wireless links for transmission in the wireless networks, their bandwidth is significantly lower than that in the wired networks and the transmission is more prone to errors due to the wireless nature. All nodes in the network are free to move arbitrarily, which may cause temporary route failures and route changes, leading to packet losses.

1.1 Motivation

TCP/IP is the protocol suite that defines the Internet. Transmission Control Protocol (TCP) is a reliable end-to-end transport protocol most widely used for data services, which is primarily designed for wired networks and became very efficient and robust with years of enhancements. However, experiments and research showed that TCP's congestion control algorithm performs very poorly over Wireless Ad Hoc networks with degraded throughputs and severe unfairness among flows [11]. This presents the need to design an applicable congestion control algorithm based on the special characteristics of the Wireless Ad Hoc networks.

Currently, the vast majority of the traffic in the Internet relies upon the congestion control mechanism provided by TCP. There are more applications appearing in the past few years, such as streaming video and Internet Telephony, which prefer timeliness to reliability. For those applications, only data arriving within a certain deadline are useful. The reliability and in-order delivery algorithm provided by TCP often results in arbitrary delay, and TCP's rate control AIMD (Additive Increase and Multiplicative Decrease) algorithm causes very sharp bandwidth change upon the detection of one packet loss, which can often be compensated for by those applications. Because of these undesirable properties of TCP, many new applications often choose UDP, with either their own congestion control mechanisms implemented on top of it or none at all. These long-lasting UDP flows without any congestion control mechanism present a potential threat of network collapse to the Internet. Also, congestion control mechanisms are difficult to implement and may behave incorrectly. This presents the need for a common base transport protocol, which is able to provide different congestion control algorithms to suit the needs of different applications. It should also provide capabilities for applications to easily implement their own congestion control algorithm under special conditions. DCCP (Datagram Congestion Control Protocol) is a general-purpose transport-layer protocol, which maintains end-to-end connections and has built-in mechanisms to provide choices of congestion control algorithm selection or implementation.

1.2 Thesis Contributions

This thesis concentrates on the potential use of DCCP with a specially designed congestion control algorithm for multi-hop Wireless Ad Hoc Access Networks. In this thesis, DCCP and other transport layer protocols are discussed, congestion control algorithms and bandwidth estimation techniques are studied in Chapter 3, the design of our congestion control algorithm and our evaluation criteria are discussed in Chapters 4 and 5.

We implemented a simplified DCCP protocol with our proposed congestion control algorithm in NS2 and performed experiments to study the throughput and fairness. Because DCCP does not provide reliability by default, additional packet loss detection and retransmission is implemented to compare the proposed congestion control algorithm with TCP. Simulation results are presented in Chapter 6 and in Chapter 7, which show that flows with the new congestion control tested in the simulation had very good inter-flow fairness, indicated by Jain's fairness index greater than 0.95 in all tests, where TCP flows may have a fairness index less than 0.6. The aggregate throughput of the two flows with our new congestion control algorithm also increased. Those improvements in fairness and bandwidth utilization show that DCCP with the designed congestion control algorithm can potentially replace TCP and UDP for transmitting audio and video applications, and with the implementation of reliability similar to what TCP provides, the reliable DCCP with the designed congestion control algorithm has the potential to replace TCP for Wireless Ad Hoc Access Networks.

1.3 Thesis Organization

This thesis is organized as follows:

Chapter 1: Introduction

Chapter 1 contains the thesis introduction, motivation, and contribution

Chapter 2: TCP and TCP in Wireless Ad Hoc Access Networks

Chapter 2 explains how TCP works, and what problems TCP encounters when it operates in Ad Hoc Networks.

Chapter 3: Recent Transport Layer Protocols and Bandwidth Estimation Algorithms

Chapter 3 studies several recent transport layer protocols, which shed light on the issues of Wireless Ad Hoc Networks and the bandwidth estimation algorithms for congestion control.

Chapter 4: DCCP with Congestion Control for Wireless Ad Hoc Access Networks

Chapter 4 discusses our proposed congestion control algorithm and unsolved problems.

Chapter 5: Performance Evaluation Criteria

Chapter 5 discusses the performance evaluation criteria and fairness.

Chapter 6: Simulation Results and Analysis

Chapter 6 shows the simulation results using Network Simulator (NS2) and provides an analysis of the results.

Chapter 7: Implementation and Simulations of Reliability on DCCP

Chapter 7 describes the implementation of reliable transmission within our scheme and shows the simulation results.

Chapter 8: Conclusions and Future Works

Chapter 8 presents the conclusion of the thesis and suggestions for future work.

Appendix

Appendix A.1 and A.2 survey additional related work on adapting TCP to Ad Hoc networks, including the investigations and proposals to improve throughput and fairness.

Chapter 2

TCP and TCP in Wireless Ad Hoc Access Networks

In Chapter 1, we stated that TCP has throughput and fairness problems when used over Wireless Ad Hoc networks. Chapter 2 gives a brief introduction of how TCP's congestion control works and studies the causes for the throughput and fairness problems of TCP in Wireless Ad Hoc networks.

2.1 TCP Overview

Transmission Control Protocol (TCP) provides a connection-oriented, reliable data transmission between the source and the destination. TCP includes a flow control mechanism, which allows the receiver to limit the transmission speed, and a congestion control mechanism. The basic idea of TCP congestion control is that TCP senders probe the network for available resources, and increase the transmission rates until packet losses are detected. TCP takes packet loss as the indication of network congestion, and it triggers a series of congestion control schemes upon the detection of congestion. There are several versions of TCP developed, and TCP Reno is the most widely used version on the Internet.

In RFC 2581 TCP Congestion Control, four congestion control algorithms are defined: slow start, congestion avoidance, fast retransmit and fast recovery. The slow start and congestion avoidance algorithms must be used by TCP senders to control the amount of outstanding data being injected into the network [1]. In Figure 2-1, during the slow start, for each received acknowledgment (ACK), the TCP sender increases the congestion window (*cwnd*) by one segment until it reaches the slow start threshold (*ssthresh*). Congestion window size is the amount of outstanding data a TCP sender can send on a connection before it gets an acknowledgement back from the receiver. During the slow start phase, the congestion window grows exponentially per round trip time (RTT). The TCP sender then enters the congestion avoidance phase during which the congestion window size increases linearly (maximum one segment size per RTT).

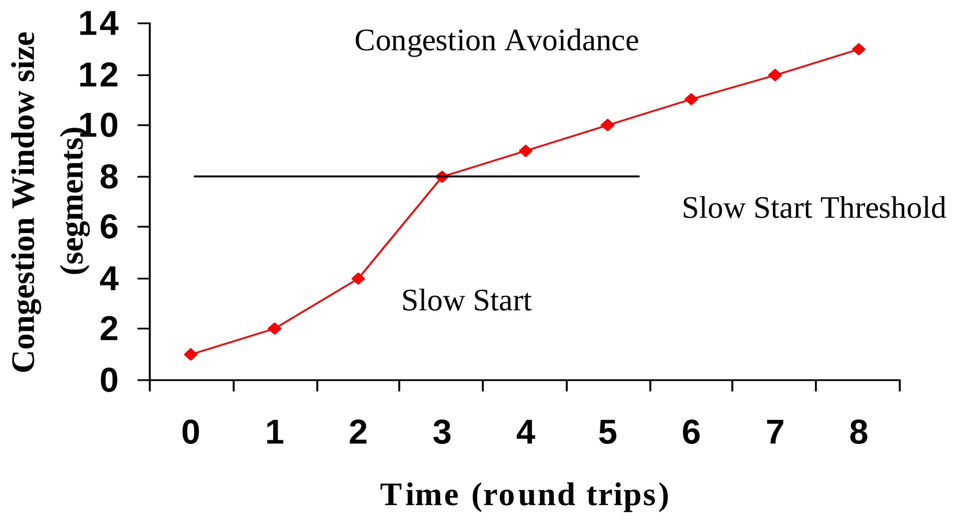


Figure 2-1 Slow Start and Congestion Avoidance

There are two ways for detecting packet loss: by duplicate ACK and by retransmission timeout (RTO). If the sender receives duplicate ACKs, it starts the fast retransmit for that packet without waiting for the retransmission timer to expire. The fast recovery algorithm then governs the transmission of new data until non-duplicate ACKs arrive [1]. Basically, fast retransmit and fast recovery work together to retransmit the lost packet and cut the slow start threshold (*ssthresh*). When a non-duplicate ACK is received, the sender enters the congestion avoidance phase again where the congestion window (*cwnd*) is set to be the *ssthresh*. The TCP sender maintains an average round trip time (RTT) delay, which is used for setting the RTO. If the sender does not receive an ACK after the timeout timer expires, the sender cuts *ssthresh* to half (or 2) of the congestion window size, and sets *cwnd* to 1. Setting of RTO follows the exponential backoff strategy, i.e. in each successive timeout, the RTO doubles.

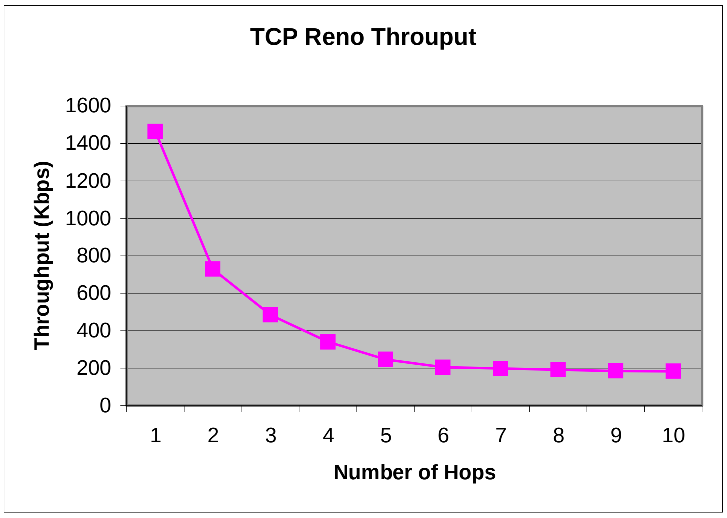
2.2 TCP in Wireless Ad Hoc Networks

The performance of TCP used in Wireless Ad Hoc Networks is discussed in this section.

2.2.1 Throughput Degradation and Reasons

TCP provides end-to-end congestion control and reliable, in-order data delivery, and is proved to be very robust over wired network, but using TCP without any modification in Wireless Ad Hoc Networks results in a drastic throughput drop.

In [11], it is shown that, when mobile nodes are fixed, the measured TCP Reno throughput over IEEE 802.11 links decreases rapidly when the number of hops increases (only a single TCP flow exists in the tests). The measured throughput gets worse when nodes are moving, see Figure 2-2 and Figure 2-3. In Figure 2-2, when the receiver is one hop away from the sender, the throughput can reach up to 1500 kbps. When the number of hops between the sender and the receiver is more than 5, the throughput drops to below 200 kbps. In Figure 2-3, the expected throughput is defined as a function of the mobility patterns and serves as a reasonable upper bound with which the measured performance may be compared [11]. Different mobility patterns yield different throughput results, and Figure 2-3 shows the average result of 50 patterns, in which the throughputs decrease as the nodes’ moving speeds increase.



Hops	Throughput (Kbps)
1	1463
2	729
3	484
4	339.9
5	246.4
6	205.2
7	198.1
8	191.8
9	185.3
10	182.4

Figure 2-2 TCP Reno Throughput over an IEEE 802.11 fixed, linear, multi-hop network of varying length (in hops) [11]

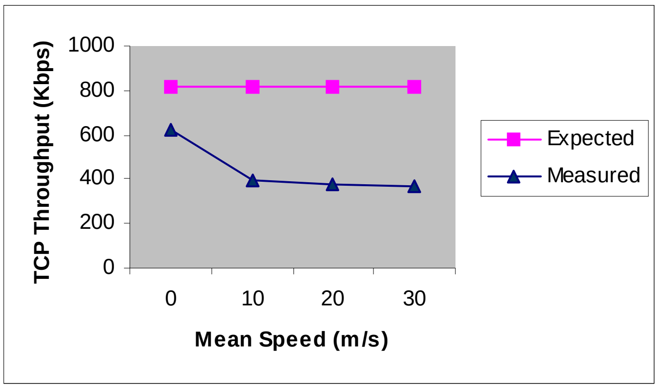


Figure 2-3 Measured and expected throughput, averaged over 50 simulated mobility patterns [11]

The above problems presented by TCP over Wireless Ad Hoc Networks are caused by the special characteristics of the Wireless Ad Hoc Networks.

- High Bit Error Rate

Wired networks have relatively low bit error rate, and TCP treats packet errors as indication of network congestion. In wireless transmission, bit error rates (BER) are high because of the fading and interferences in wireless channels. Assuming that each error indicates network congestion and triggering the congestion control mechanism affects the throughput and link utilization.

- Mobility

All nodes forming the Ad Hoc networks can move freely. When nodes are moving, existing links may break, so the route between two nodes becomes obsolete and a new route has to be selected by the routing protocol during the data transmission. If the time of establishing a new route is longer than RTO, the TCP sender invokes the congestion control and reenters the slow start phase. This degrades the throughput.

Mobility also causes network partitions. A network partition occurs when a given mobile node moves away, or is interrupted by the medium, thereby splitting its adjacent nodes into two isolated parts of the network that are called partitions [19]. If the TCP sender and receiver lie in two different partitions, the network drops all the data segments between the endpoints. The TCP sender invokes congestion control and exponentially backs off the retransmission. If the partitions last longer than several RTOs, network inactivity could happen, that means the route has been reestablished after the exponential backoff timer starts, but the sender still needs to wait until it expires. This wastes the available resources and affects TCP performance.

- ACK Bunching

In Wireless Ad Hoc Networks, the connections often consist of multiple hops. Considering the medium access control protocol used in Wireless Ad Hoc Networks, when CSMA/CA (Carrier Sense Multiple Access/Collision Avoidance) is used, all TCP flows are sharing the same channel. Competition for the bandwidth is among flows and even between data packets and their acknowledgements. TCP is based on ACK for correct congestion control, so timely ACK reception is necessary for packet transmission and correct calculation of congestion window size and RTO.

The well-known exposed node problem and hidden node problem in Ad Hoc networks can cause bunching of ACKs. Bunched ACKs cause bursty traffic and highly variable round trip times (RTTs). It may even cause the TCP sender triggering the congestion control due to starvation of ACKs. A study [11] showed that, to alleviate these problems, smaller packet size and smaller maximum congestion window size have to be used.

2.2.2 Fairness Issues

With the easy network deployment and wide hardware availability, Wireless Ad Hoc Networks are seeing increasing demands. It is very important to ensure that access to the network by each user remains fair. Fairness can be intuitively defined as the obtained throughput to its fair share of the bandwidth, and a more detailed discussion is given in Chapter 5.

In addition to the throughput problem, TCP flows present a severe unfairness in the Ad Hoc Networks, which is the result of the joint interactions of TCP, MAC layer protocol and queuing type at the router. The unfairness is shown in the following aspects:

- TCP’s window-based congestion control adjusts the congestion window size every RTT. The congestion window size doubles every RTT in the slow start phase and increases linearly in the congestion avoidance phase. Therefore, flows with longer RTT tend to increase the congestion window size slower than flows with shorter RTT. This presents per-flow unfairness.

- At the network routers, an unfair packet-dropping scheme, such as FIFO drop tail scheme, causes some flows experiencing more losses than others, which increases the unfairness.
- In an infrastructured wireless local network, where a gateway (access point) is used to forward the traffic from the sender to the receiver (both are one hop away from the gateway), the upstream flows (from the sender to the gateway) tend to occupy the whole media and the downstream flows (from the gateway to the receiver) almost stop transmission when multiple upstream and downstream flows co-exist. With IEEE 802.11, when one sender and one receiver exist, the sender and the gateway both have equal access to the media. If there is one sender and multiple receivers in the network, the sender gets half of the bandwidth, where all the receivers share the other half of the bandwidth. When multiple senders and receivers exist, unfairness between the upstream and downstream flows is extremely serious, with a ratio of the sender to the receiver rate up to 800. The research in [22] shows that the queue size at the access point plays an important role in deciding the bandwidth sharing. When multiple senders exist, their ACKs are also in contention for media access with downstream packets, which experience many timeouts due to packet drops in the gateway buffer.

- In a Wireless Ad Access Hoc network, IN TCP flows (from the wired part to the wireless part) get more bandwidth than the coexisting OUT TCP flows (from the wireless part to the wired part) [29]. [29] studied two scenarios (see Figure 2-4) and scenario B is a combined Wireless Ad Hoc and wired network, where node 3 is working as the gateway. The test shows that scenario B present a different unfairness condition from scenario A, which is a typical WLAN with one hop wireless connections. In Figure 2-5, the table shows the measured throughputs in scenario B, where the IN flow have a much higher share of the bandwidth when mixed flows exist.

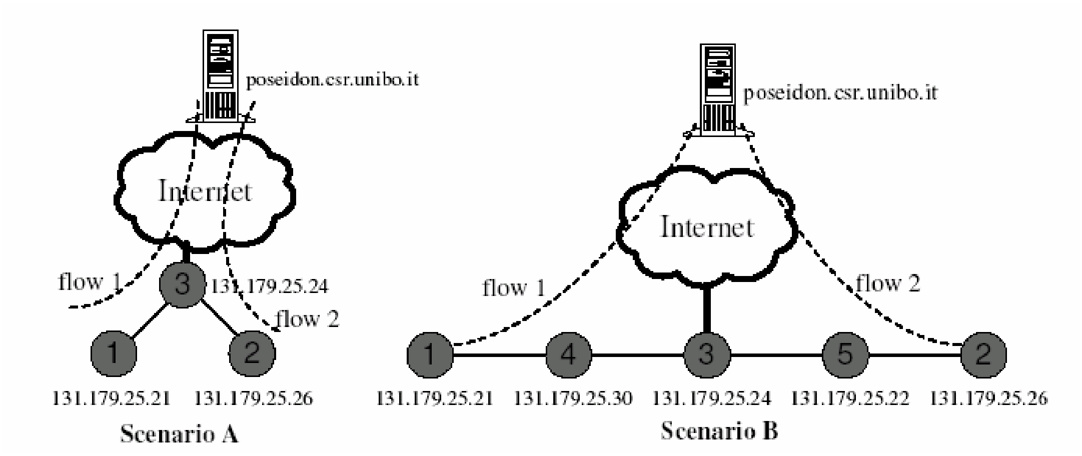
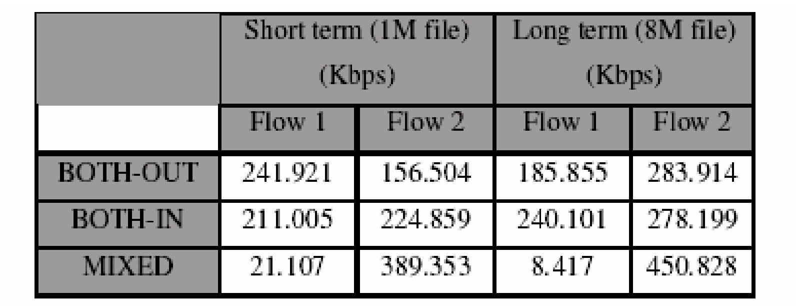


Figure 2-4 Scenarios for Testbed Measurement in [29]



	Short term (1M file) (Kbps)		Long term (8M file) (Kbps)	
	Flow 1	Flow 2	Flow 1	Flow 2
BOTH-OUT	241.921	156.504	185.855	283.914
BOTH-IN	211.005	224.859	240.101	278.199
MIXED	21.107	389.353	8.417	450.828

Figure 2-5 Measured Throughputs for Scenario B in [29]

IEEE 802.11 has the exposed node and hidden node problem. With 802.11 DCF (Distributed Coordination Function) mode, nodes within the transmission range of other nodes are unable to receive a correct RTS (Request To Send) or respond with a CTS (Clear To Send), which causes these nodes to be penalized by other nodes' transmission. See Figure 2-6, when node G is transmitting packets to node 2, node 3 can hear the transmission, so the RTS from node 4 is corrupted (node G is the hidden node in relation to node 4). If node 3 needs to transmit, it must wait until node G stops and then contend for the medium (Node G is the exposed node in relation to node 3). Also, the Binary Exponential Backoff prefers the latest successful nodes, which occupy the media persistently and leave other flows starving of the resource or even stop transmission completely. TCP's own timeout and backoff schemes further worsen the unfairness.

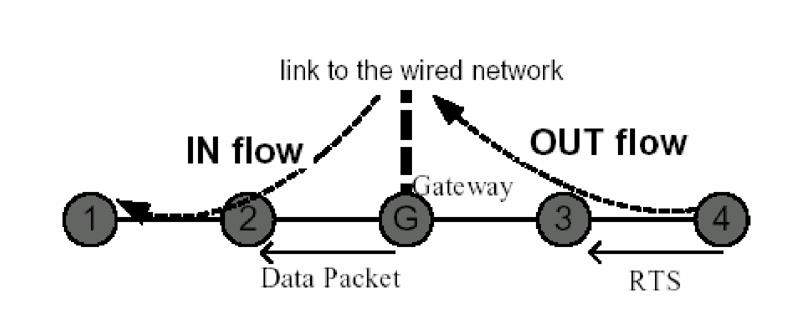


Figure 2-6 Exposed Nodes and Hidden Nodes Problems in [29]

2.3 Summary

TCP works poorly over Wireless Ad Hoc Networks, this is caused by the high bit error rate over wireless links, arbitrary mode mobility, as well as TCP's built in congestion control algorithm working with the contention based media access of IEEE 802.11. A vast amount of research has focused on improving the throughput and fairness issues discussed above. In Appendix A, various specific proposals for modifying and adopting TCP to Ad Hoc networks are discussed.

Chapter 3

Recent Transport Layer Protocols and Bandwidth Estimation Algorithms

In this chapter, several recent transport layer protocols are introduced, including protocols specifically designed for Ad Hoc Networks (ATP), for high-bandwidth and long-delay networks (FAST TCP) and a general purpose transport layer protocol with multiple congestion control mechanism choices (DCCP). They are recent research topics in academia and their features shed some lights on the transport layer issues of Wireless Ad Hoc Access Networks.

The bandwidth estimation is an essential part in the congestion control mechanism to regulate the packet transmission rate at the sender side. In this chapter, existing bandwidth estimation techniques are introduced and ones used in different TCP versions are discussed.

3.1 Recent Transport Layer Protocols

This session discusses transport layer protocols, including ATP, FAST TCP and DCCP.

3.1.1 ATP: A Reliable Transport Protocol for Ad-Hoc Networks

In [25], TCP is taken as a protocol fundamentally inappropriate for the unique characteristics of Ad Hoc networks. Hence a new protocol, Ad Hoc Transport Protocol (ATP), is proposed.

3.1.1.1 The ATP Design

In the design of ATP, some key ideas are fundamentally different from TCP.

- ATP uses lower layer information and explicit feedback from other network nodes to assist in the transport layer mechanisms. This information is used for start-up rate estimation, congestion detection/avoidance/control and path failure notification.
- ATP decouples congestion control and reliability. In ATP, intermediate nodes are used to provide the congestion information in terms of a congestion rate, which is piggybacked on the data packets in the forward path. The ATP receiver sends back the feedback to the sender. To achieve reliable transmission, selective ACKs are used to report back the transmission information.
- Each node maintains two parameters: Q_t (an exponential average of queuing delay for packets traversing the node) and T_t (an exponential average of the transmission time by the head-of-line packet at the node). Every packet will be stamped with $Q_t + T_t$ which is the maximum value through the route, which is used by the receiver as the rate feedback to the sender.
- ATP exhibits good fairness properties. When an intermediate node servicing several flows experiences congestion, it sends feedback of the congestion to all the sources of the flows being serviced by it. All sources respond to this congestion feedback in an identical manner, thereby a higher degree of fairness is achieved. The normalized standard deviation is used as measurement for fairness, and the simulation showed a decrease of 40% compared with TCP and TCP-

ELFN [25]. Explicit Link Failure Notification (ELFN) provides the sender with information about the link and route failure, so the sender will recognize the link failure as not network congestion.

3.1.1.2 The ATP Protocol

The ATP protocol can be described as follows:

- ATP uses quick start during the connection initiation and the route reestablishments by sending out probe packets. All intermediate nodes record $Q_t + T_t$ when the packet traverses the node and check $Q_t + T_t$ in the rate feedback field D. At a certain node, if the D field has a smaller value than what the node records, the D field is reset by the higher value. When the receiver receives the probe packet, the D field is the maximum delay experienced by the probe packet in a certain node on the route from the sender to the receiver. The receiver calculates the average D, adjusts it for the initial transmission, and sends this information back to the sender. The sender then calculates the rate, which is proportional to the inverse of the D value, and uses it as the sending rate for the forward path.
- The receiver sends back the feedback information periodically. The period should be larger than the round trip time of the connection and small enough to track the changes of the path. The authors recommend one second as a reasonable default value.
- ATP has a three-phase congestion control method consisting of increase, decrease and maintenance phases. When the feedback rate from the receiver is greater than

the current rate by a certain threshold, the protocol enters the increase phase, where the flow increases its rate by a fraction of the potential increase amount. On the contrary, if the feedback rate from the receiver is smaller than the current rate, the protocol enters the decrease phase and simply reduces the rate to the feedback rate. If the feedback rate is within a certain range from the current rate, the protocol is in the maintenance phase and the sending rate remains unchanged.

- ELFNs allow ATP to multiplicatively decrease the congestion window size. After receiving ELFNs, ATP enters the connection initiation phase, where the probe packet piggybacked on the next in-sequence data packets is sent out periodically.
- Selective Acknowledgement (SACK) is chosen to provide lost packet information.

3.1.1.3 Conclusions

ATP presents a new rate-based transport protocol based on feedback information from intermediate nodes. It also uses explicit link failure notification, combined with SACK and rate feedback, to indicate the network states. This hybrid information is more reliable than just relying on the feedback from the network or end nodes. But ATP is designed to work on stand-alone Ad Hoc networks, which does not include a wired network extension.

3.1.2 FAST TCP

FAST TCP [16] is a modification to the TCP congestion control algorithm for high-speed long-distance networks. The current TCP is not stable when used in a network with a high bandwidth-delay product. First, in order to sustain a large window size, the current

TCP requires that an end-to-end path should maintain a small loss probability, which may be difficult to achieve even in optical networks. Second, the AIMD (Additive Increase and Multiplicative Decrease) algorithm can lead to oscillation and underutilization of the bottleneck links because of the conservative increase (AI) at long delay links and drastic decrease (MD) at large window sizes.

FAST TCP aims to utilize the link resources fairly among all TCP connections and to avoid the congestion with maximum link utilization. FAST TCP uses queuing delay as well as packet loss to indicate congestion and adjusts the window size. When the congestion is mild, i.e. no packet losses happened, queuing delay is the dominant congestion signal; when congestion becomes severe, packet loss is then the dominant congestion signal. FAST TCP only modifies the TCP sender to adjust the window size based on the congestion signal. FAST TCP also supports ECN (Explicit Congestion Notification) [24]. If the ECN bits in IP header are marked by routers along the route to indicate a network congestion condition, the receiver can send ECN Echo back to the sender and notify the sender to reduce the packet transmission rate. When ECN is available, FAST TCP can be extended to use ECN to supplement or replace queuing delay as the congestion signal.

FAST TCP is based on the prime-dual model of the TCP protocol where TCP is modeled by a nonlinear closed-feedback and time-delayed control system. With queuing delay as the congestion measure, this multi-bit information allows an equation-based implementation of the source rate to stabilize in an equilibrium state with a target fairness and high link utilization. This is achieved by correctly responding to the queuing delay to maintain a stable queue in the bottleneck router, which avoids current TCP window control induced queue overflow and the oscillation caused by queue overflow.

3.1.2.1 Implementation

In [15], the congestion control mechanism of TCP is separated into four independent functional components (Figure 3-1). This independence allows each component to be designed and implemented independently.

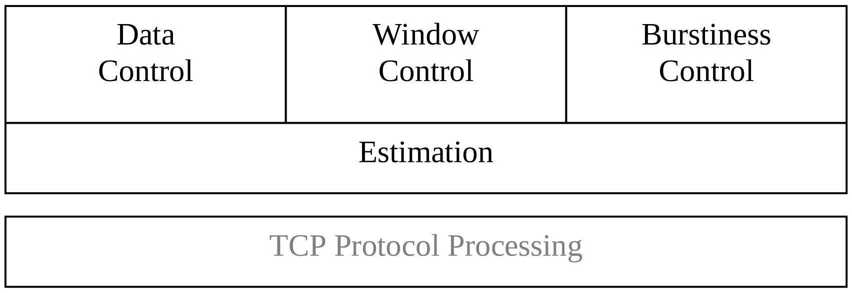


Figure 3-1 FAST TCP Architecture [15]

The *data control* component determines which packets to transmit, *window control* determines how many packets to transmit, and *burstiness control* determines when to transmit these packets. The *estimation* component provides information to the other three components to make decisions. In [15], the functional description for estimation and window control components are as summarized below.

The estimation component studies two types of feedback information from the network, positive acknowledgment and negative acknowledgment (timeout or duplicate ACKs). Positive acknowledgments are used to calculate RTT and update corresponding queuing delay and minimum RTT. Negative acknowledgments are used to provide loss event indication.

FAST TCP's window control determines the congestion window based on queuing delay and packet loss. FAST TCP uses equation-based control with queuing delay and multiplicative decrease with packet loss. Under normal network condition, the equation used by FAST TCP [15] is:

$$w \leftarrow \min\{2w, (1-\gamma)w + \gamma\,[(\text{baseRTT}/\text{avgRTT})w + \alpha(w, \text{qdelay})]\}$$

where $\gamma \in (0, 1]$. The variables in the above equation are:

- w is the congestion window size.
- avgRTT is the average RTT computed using a moving average with a suggested weight of minimum of 3/w and ¼ (min(3/w, ¼) [15].
- baseRTT is the minimum of the observed RTT samples since the start of the connection.
- qdelay= avgRTT – baseRTT is the estimate of the current queuing delay.
- Function $\alpha(w, \text{qdelay})$ is chosen to be constant. The constant, which may be different for different TCP flows, specifies the total number of packets that a single FAST connection tries to maintain queued along the path. For n flows sharing the same bottleneck link and with the same constant, each flow will get 1/n of the bandwidth at a stable state (given router queue size is n times larger than the constant).

When packet loss occurs, the source will follow RFC 2582 (New Reno) to enter Fast Retransmission/Fast Recovery. The source starts reacting to queuing delay again when the acknowledgments of newly transmitted packets are received and enough samples (30% of w at the beginning of the current loss event) are collected.

3.1.2.2 Performance Evaluation

In [15], the following performance evaluation metrics are given to evaluate the experiments:

- Throughput: average aggregate throughput for time interval [1, m] of n flows.
- Intra-protocol fairness: Jain's fairness index is used to evaluate the fairness among n flows.
- Stability: the stability index of flow *i* is defined as the normalized sample throughput standard deviation. The smaller the stability index, the less oscillation a source experiences. For *n* flows, the stability index is the average of *n* individual stability indices.
- Responsiveness: the responsiveness index measures the speed of convergence when network equilibrium changes.

[15] listed experimental results comparing up to 8 flows of FAST TCP, TCP Reno, STCP (Scalable TCP) and HSTCP (High Speed TCP), FAST TCP outperforms the other three protocols for all above evaluation criteria. Figures 3-2 and 3-3 are the summary results for overall throughput and fairness across several experiments with different delays, number of flows and their arrival and departure patterns. Figures 3-2 and 3-3 show that FAST TCP can achieve higher throughput and better fairness.

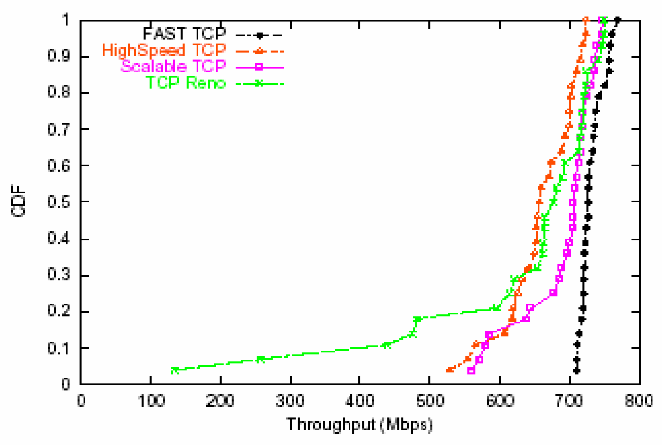


Figure 3-2 Evaluation: Throughput [15]

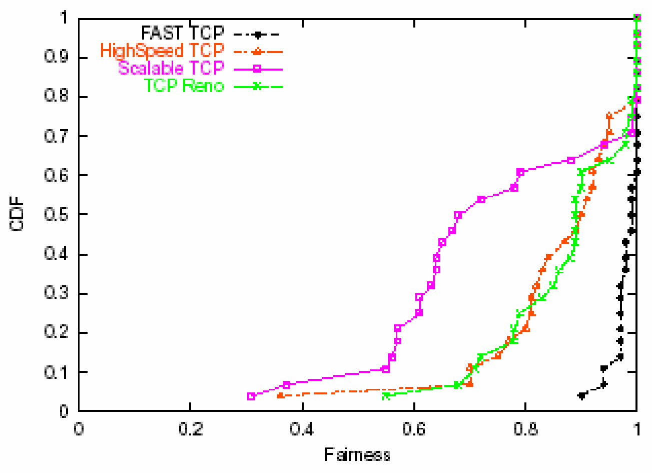


Figure 3-3 Evaluation: Fairness [15]

3.2 Bandwidth Estimation

Bandwidth or throughput in a packet network usually means the amount of data that the network can transfer per unit of time. The estimation of the maximum available bandwidth for an end-to-end connection is crucial to the congestion control mechanisms used in the transport protocols and directly impacts the application performance.

The bandwidth of a link is different from the bandwidth of an end-to-end path, which consists of a sequence of successive links along the path. In data networks, links usually correspond to point-to-point links at the data link layer, called segments. At the network layer, links may consist of one or more segments connected by data link layer devices, such as switches or bridges, which are called hops. An end-to-end connection from a source to a destination may consist of a set of hops. A data link layer normally provides a constant transmission bit rate. For instance, the rate is 1Mbps, 2Mbps, 5.5Mbps or 11Mbps for IEEE 802.11b links, depending on the error rate of the wireless medium, and 10Mbps on a 10BaseT Ethernet segment.

At the network layer, a hop has a lower rate than this data link layer transmission rate because of the overhead of data link layer encapsulation and framing. Suppose that the data link layer transmission rate is C_{L2}, and the total header size at the data link layer is H_{L2}, then for a network layer data packet of length L_{L3}, the achieved network layer transmission rate is [23]:

$$C_{L3} = C_{L2} \frac{1}{1 + \frac{H_{L2}}{L_{L3}}}$$

This is the same for a transport layer data segment of length L_{L4}, the transport layer transmission rate is:

$$C_{L4} = C_{L3} \frac{1}{1 + \frac{H_{L3}}{L_{L4}}}$$

The above equations specify the upper bound of the per-link transmission rate at each layer. Whereas the maximum transmission rate per hop is given above, the maximum transmission rate or bandwidth of an end-to-end connection consisting of n hops is defined by the minimum link capacity, i.e.

$$C = \min_{i=1,\dots,n} C_i$$

where C_i is the capacity of the *i*-th hop. This gives the upper bound for the bandwidth of an end-to-end connection.

Links are shared among many connections, and the available bandwidth over a link is the unused capacity over a certain time period, which depends on the traffic load and is a time-varying metric. At any specific time instant, a link is either transmitting at full capacity or it is idle. The available bandwidth over a link in a certain time interval is the time average of the instantaneous utilizations over the time interval. If C_i is the capacity of hop i and u_i is the average utilization of that hop in the given time interval, the average available bandwidth A_i of hop i is [21]:

$$A_i = (1 - u_i)C_i$$

And for an end-to-end connection including *n* hops, the available bandwidth is determined by the link with minimum available bandwidth along the path:

$$A = \min_{i=1,\dots,n} A_i$$

It should be noted that the narrow link (with minimum capacity) along the path is not always the tight link (with minimum available bandwidth). It is the tight link that specifies the end-to-end available bandwidth.

There are many bandwidth estimation techniques described in the literature. In this section, existing techniques and various TCP bandwidth estimation methods are discussed.

3.2.1 Bandwidth Estimation Techniques

In this section, existing bandwidth estimation techniques are introduced and ones used in different TCP versions, including Variable Packet Size (VPS) probing, Packet Pair probing, Self Loading Periodic Stream (SLoPS) probing, Trains of Packet Paris (TOPP) probing, and TCP with Adaptive Pacing (TCP-AP), are discussed.

3.2.1.1 Variable Packet Size (VPS) Probing

Variable Packet Size (VPS) probing [12] measures the bandwidth at each hop along the path. It uses ICMP (Internet Control Message Protocol) error messages to measure the Round Trip Time (RTT) from the source to each hop. The Time-To-Live (TTL) field of the IP header of the probing packets ranges from 1 to i consecutively, in order to force probing packets to expire at the particular hop. By the returning ICMP "Time-exceeded" error message sent back by the router, which deleted the probing packets after TTL expires, the source can measure the RTT to that hop. This RTT consists of three parts on both the forward and reverse path: transmission delays, which are the time used to transmit a packet onto a link, propagation delays, which are the time for a packet to traverse a series of links, and queuing delays, which are the time a packet is waiting at the buffers of each hop.

VPS assumes that the minimum RTT observed happened when there is no queuing delay. Therefore the minimum RTT only consists of two parts: transmission delay and propagation delay. The transmission delay is proportional to the packet size, and the propagation delay is independent of the packet size. Because the returning ICMP error packets have a constant size, the minimum RTT is the sum of two parts: one is unrelated to the probing packet size and one is related to the probing packet size. When the source sends out probing packets of a given packet size L to a sequence of hops from 1 to i, at the k-th hop, the minimum RTT $T_i(L)$ is:

$$T_i(L) = \alpha + \sum_{k=1}^{i} \frac{L}{C_k} = \alpha + \beta_i L$$

where α is the delay, which is independent of packet size, C_k is the bandwidth at the k-th hop. By observing β_i for each hop, the bandwidth at hop i is:

$$C_i = \frac{1}{\beta_i - \beta_{i-1}}$$

VPS probing can underestimate the bandwidth when used in a path involving layer-2 switches store-and-forward, in which the delay is related to probing packet sizes but does not generate ICMP error messages by the data link layer devices.

3.2.1.2 Packet Pair Probing

Packet pair probing [13] is used to measure the end-to-end available bandwidth for a path. The source sends out multiple pairs of packets with the same size back to back to the receiver.

The dispersion of the two packets in a pair is the time distance between the last bits of each packet. The packet pair probing is assuming that the dispersion of a packet pair at the receiver reflects the bandwidth of the bottleneck link on the path. If two acknowledgments are sent back by the receiver after receiving the probing packet pair, the dispersion of the two ACKs should have the same spacing if the reverse path is uncontested. The sender can thus estimate the available bandwidth along the path by the dispersion between the corresponding ACKs.

This probing algorithm injects extra packets into the networks and the reverse path condition can significantly affect the correct estimation.

3.2.1.3 Self Loading Periodic Stream (SLoPS)

Self Loading Periodic Stream (SLoPS) [13] is another mechanism for end-to-end available bandwidth estimation. In SLoPS, a packet stream consisting of K packets of size L is sent at a constant rate R. The one-way delays of probing packets are measured. If the one-way delays show an increasing trend, then the probe stream transmission rate R is greater than the end-to-end available bandwidth of the path, because probing packets are queued at the buffer of the bottleneck link. On the other hand, if the stream transmission rate R is less than the end-to-end available bandwidth, probe packets are not delayed along the route, so the observed one way delays do not increase constantly, but tend to have the same value.

The receiver analyses the relation of stream transmission rate R and available bandwidth A, and notifies the sender of the relation. If probe stream n has a rate of $R_{(n)}$, and $R_{(n)} > A$, then the sender sends the next probe stream n+1 with rate $R_{(n+1)} < R_{(n)}$. Otherwise, the rate is set to $R_{(n+1)} > R_{(n)}$. The computation of $R_{(n+1)}$ can be as follows:

$$\text{If } R_{(n)} > A,\ R^{\max} = R_{(n)};$$

$$\text{If } R_{(n)} < A,\ R^{\min} = R_{(n)};$$

$$R_{(n+1)} = (R^{\max} + R^{\min})/2;$$

$R^{\max}$ and $R^{\min}$ are the upper and lower bound for the available bandwidth of stream n. If the available bandwidth does not change, the stream rate will converge to a range [$R^{\min}$, $R^{\max}$].

When the available bandwidth *A* changes during the measurement, the one-way delays will not show a clear increasing trend or be constant. SloPS refers to this as the grey region, which is related to the variation range of *A* during the measurements.

3.2.1.4 Trains of Packet Pairs (TOPP)

The Trains of Packet Pairs (TOPP) algorithm [20] follows a similar idea as SLoPS in estimating the end-to-end available bandwidth. In TOPP, the sender sends out *n* probe packet pairs with increasing rates. If the probe packet pair sending rate R_o is more than the available bandwidth *A*, the second packet in a packet pair will be queued and causes a larger dispersion at the receiver side. Hence the receiver will observe the receiving rate $R_m < R_o$. With increasing R_o , the R_m will show a decreasing trend. If the probe packet rate $R_o < A$, the packet pair will arrive at the receiver with the rate they were sent by the sender, i.e. $R_m = R_o$.

TOPP linearly increases the probe packet pair rate, which may provide more information than SLoPS does. How the TOPP sender adjusts its rates is also different from the methodology used in SLoPS. TOPP estimates the available bandwidth *A* to be the maximum probe packet pair rate R_o when $R_o = R_m$.

3.2.1.5 TCP-AP: TCP with Adaptive Pacing

TCP-AP (TCP with adaptive pacing) [10] measures the fluctuation of RTTs using the coefficient of variation. TCP-AP uses the estimation of 4-hop propagation delays (FHD) and the coefficient of RTT variation to calculate the rate for pacing the transmission by TCP senders. Using FHD is because the sender's transmission rate will not be affected by hidden node problems after the data packet is forwarded 4 times in Wireless Ad Hoc Network. So the max transmission rate: $R_{\max} = 1/FHD$ and the actual rate is changed according to the variations of RTT.

TCP-AP uses this rate-based congestion control algorithm to spread the packet transmissions, which prevents burstiness caused by TCP's window based congestion control algorithm. TCP-AP also proactively identifies the network congestion through the measuring of RTT and adjusts its transmission rate accordingly, which reduces the contention and actual congestion state in the network.

3.2.2 Bandwidth Estimation in TCP

TCP has shown to be very efficient over the Internet. In Section 2.1, TCP Overview, TCP's congestion control algorithms are introduced. The bandwidth estimation algorithms in various versions of TCP are studied in this section.

The basic idea of TCP's available bandwidth estimation is based on two state variables for each connection: congestion window size (*cwnd*) and slow start threshold (*ssthresh*). Each connection is in the slow start phase when the connection is set up. During this period, the *cwnd* is increased exponentially per RTT until it reaches the *ssthresh.* After reaching the *ssthresh*, the connection is in the congestion avoidance phase and the *cwnd* is increased linearly. In TCP Reno/NewReno, after a congestion event is detected, the *ssthresh* is set to be half the current *cwnd*. It is assumed that *ssthresh* as an estimation of available bandwidth for the connection, and the linear increase of *cwnd* in the congestion avoidance phase is used to probe extra bandwidth available to the flow. This probe continues until network congestion is detected, i.e. packet loss happens.

To improve TCP performance, several other bandwidth estimation techniques are proposed, which focus on TCP's oscillatory behavior and are more robust when used in high-speed networks and wireless networks.

3.2.2.1 TCP Vegas

TCP Vegas was introduced in 1994 as an alternative to TCP Reno and its implementation and tests showed that it achieves better throughput than TCP Reno. TCP Vegas' bandwidth estimation differs from that in TCP Reno. Unlike TCP Reno, which uses packet loss as the indication of network congestion, TCP Vegas uses the difference between the estimated throughput and the measured throughput as the measure of congestion [4].

TCP Vegas records the smallest measured round trip time as BaseRTT and computes the available bandwidth as:

$$ExpectedBandwidth = \frac{WindowSize}{BaseRTT}$$

where the *WindowSize* is the current window size. During the packet transmission, the round trip time (RTT) of packets are recorded. The actual throughput is calculated as:

$$ActualBandwidth = \frac{WindowSize}{RTT}$$

The difference between the *ExpectedBandwidth* and *ActualBandwidth* is used to adjust the *WindowSize*:

$$Diff = ExpectedBandwidth \text{ - } Actual\ Bandwidth$$

Two values α and β ($0 \le \alpha < \beta$) are defined as the thresholds. If *Diff*< α, the window size is increased during the next RTT; if *Diff*> β, then the window size is decreased during the next RTT. Otherwise, the window size is unchanged. The goal of TCP Vegas is to keep a certain number of packets or bytes in the queues of the network. If the actual throughput is smaller than the expected throughput, TCP Vegas takes this as indication of network congestion, and if the actual throughput is very close to the expected throughput, it is suggested that the available bandwidth is not fully utilized, so TCP Vegas increases the window size.

This mechanism used in TCP Vegas to estimate the available bandwidth does not purposely cause any packet loss. Hence the oscillatory behavior is removed and a better throughput is achieved. But it has problems when packets do not follow the same route and when large delays are present. When routes change for a certain TCP Vegas flow, the *BaseRTT* recorded from the previous route is no longer accurate; this affects the accuracy of *ActualBandwidth* and subsequently influences the performance of TCP Vegas. It can also be shown that TCP Vegas could become unstable when there is large network delay for a flow; later established connections cannot get a fair share of the bandwidth, and when they coexist with TCP Reno connections, TCP Reno connections use most of the bandwidth [21]. Recent studies are focused on solving these problems and enhancing the

TCP Vegas performance [7].

3.2.2.2 TCP Westwood

TCP Westwood proposes an end-to-end bandwidth estimation algorithm based on TCP Reno. TCP Westwood implements slow start and congestion avoidance phases as TCP Reno, but instead of halving the congestion window size as in TCP Reno when congestion happens, TCP Westwood adaptively estimates the available bandwidth and sets the congestion window size and slow start threshold accordingly to improve the link utilization.

In TCP Westwood, packet loss is indicated by the reception of 3 duplicated acknowledgements (DUPACKs) or timeout expiration. When 3 DUPACKs are received, TCP Westwood sets *ssthresh* and *cwnd* as follows [27]:

```
if (3 DUPACKs are received)
        ssthresh = (BE * RTTmin)/seg_size;
        if (cwnd > ssthresh) /* in congestion avoidance phase*/
                cwnd = ssthresh;
        endif
endif
```

where the *seg_size* is the length of the TCP segments and *RTTmin* is the minimum RTT experienced. *BE* is the estimated available bandwidth. It is assumed in TCP Westwood that when 3 DUPACKs are received in the congestion avoidance phase, the available bandwidth is fully utilized. So the values *ssthresh* and *cwnd* should reflect the estimated bandwidth (BE).

If a packet loss is indicated by a timeout expiration, TCP Westwood sets *ssthresh* and *cwnd* as follows:

```
if (timeout expires)
    cwnd = 1;
    ssthresh = (BE * RTTmin)/seg_size;
    if (ssthresh < 2)
        ssthresh = 2;
    endif
endif
```

This sets the *cwnd* to 1 and *ssthresh* to BE after the timeout event and then the TCP Reno behavior continues.

In TCP Westwood, the setting of *ssthresh* and *cwnd* is based on the bandwidth estimation, which is obtained by measuring the rate of the acknowledgments and collecting the information of the amount of packets delivered to the receiver in the ACK. Samples of bandwidth are computed as the amount of packet delivered divided by the inter-arrival time between two ACKs. Those sample bandwidth estimates are then filtered to achieve an accurate and fair estimation.

TCP Westwood modifies the Additive Increase and Multiplicative Decrease (AIMD) in TCP Reno and adaptively sets the transmission rates to remove the oscillatory behavior of TCP Reno and to maximize the link utilizations. But this also causes TCP Westwood to degrade the performance of TCP Reno connections when they coexist in the network. [28] discusses ways to achieve fairness and to maintain efficiency.

3.2.2.3 TIBET: Time Interval based Bandwidth Estimation Technique

The Time Interval based Bandwidth Estimation Technique (TIBET) is newly proposed to improve the TCP performance over links with random loss. TIBET estimates the available bandwidth used by a TCP source and also enables the TCP connections to track changes in the available bandwidth quickly [5].

The bandwidth estimation is obtained by performing a runtime sender-side estimation of the average packet length and the average time interval separately. The algorithm can be applied either to the stream of transmitted packets or the stream of received ACKs. The algorithm applied to the stream of transmitted packets is as follows:

```
if (Packet is sent)
    sample_length[k] = (packet_size * 8);
    sample_interval[k] = now - last_sending_time;
    Average_packet_length[k] =
                    alpha * Average_packet_length[k-1] +
                    (1-alpha) * sample_length[k];
    Average_interval [k] =
                    alpha * Average_interval[k-1] +
                    (1-alpha ) * sample_interval[k];
    Bwe[k] = Average_packet_length[k] / Average_interval[k]
endif
```

Where *packet_size* is the TCP segment size in bytes, *now* and *last_sending_time* are current time and the time for previous packet transmission respectively.
Average_packet_length and *Average_interval* are the smoothed values for the packet length and inter-departure time filtered by alpha (0<=*alpha*<=1). Different values of alpha have a large impact on the performance of TIBET. With a lower value of alpha, the bandwidth estimation is more responsive to bandwidth change, but oscillates widely. A lower alpha makes the bandwidth estimation less responsive to bandwidth change, but very stable. The test results showed that alpha equal to 0.99 provides a good tradeoff between responsiveness and stability.
If the algorithm is applied to the stream of received ACKs, the calculation of *sample_length* and *sample_interval* are:

```
    sample_length[k] = (acked * packet_size * 8);
    sample_interval[k] = now – last_ack_time;
```

Where acked is the number of segments acknowledged by the ACK and last_ack_time is the time when the last ACK was received.
After congestion events are detected, TIBET sets *ssthresh* and *cwnd* based on the bandwidth estimation as follows:

```
    if (3 duplicate ACKs are received)
        ssthresh = BE * RTTmin
        if (cwnd > ssthresh )
            cwnd = ssthresh
        endif
    endif
```

```
if (retransmission timeout expires)
        *ssthresh* = bandwidth * RTTmin
        *cwnd* = 1
endif
```

The rationale behind this is the same as in TCP Westwood.

Comparing TIBET with TCP Westwood, simulations in [5] show that TIBET can achieve an accurate estimate of the available bandwidth. It also shows the TIBET is fair towards TCP Reno connections in the simulations. In TIBET, the bandwidth estimation algorithm considers the past history of the connection when congestion events are detected, so it is able to achieve high throughput in the presence of random errors over the links. In a combined scenario with multiple wired links and one wireless link, simulations show TIBET works better than TCP Reno. In summary, TIBET enables the improvement of throughput over wireless links and fairness over wired links.

3.3 DCCP: Datagram Congestion Control Protocol

Datagram Congestion Control Protocol (DCCP) is a new protocol designed for applications that require the flow-based semantics of TCP, but prefer timely delivery to in-order delivery, or a congestion control mechanism different from what TCP provides.

3.3.1 Introduction of DCCP

DCCP is discussed in the IETF DCCP charter, which provides a congestion-controlled, unreliable packet stream, without TCP’s reliability or in-order delivery semantics. DCCP aims to be a minimal overhead and general-purpose transport-layer protocol providing only two core functions:

- The establishment, maintenance and teardown of an unreliable packet flow.
- Congestion control of that packet flow.

The purpose of DCCP is to provide a standard way to implement congestion control and congestion control negotiations for special applications. DCCP provides the following features:

- A choice of congestion control mechanisms.

  Applications should be allowed to choose the most desirable congestion control schemes based on their special requirements. Currently two congestion control mechanisms are undergoing discussion as Internet drafts: TCP-like congestion control and TCP-Friendly Rate Control. DCCP provides a standard way to implement more congestion control mechanisms and to negotiate the control options. Each congestion control scheme is uniquely identified by a CCID (congestion control identifier).
- A reliable handshake for connection setup and teardown.

  Unlike UDP, DCCP is a stateful protocol, which has connection setup and teardown functionality to be friendly to network address translators and firewalls. To avoid Denial-of-Service attacks, a cookie mechanism can be used during the connection setup, so a server is able to avoid holding states of unacknowledged connection attempts.
- An unreliable delivery of data flows, with acknowledgements.

DCCP provides options that can tell the sender which packets the receiver receives and whether those packets are Explicit Congestion Notification (ECN) marked, corrupted, or dropped at the receiver side.

- Negotiation of features.

  DCCP uses a standard mechanism to negotiate properties for connections (between two DCCP endpoints). These properties are called features, such as the CCID. During the feature negotiation, the Change, Prefer and Confirm options are used for both endpoints to agree on a certain value of a feature for a half connection (a connection includes two half connections, each is made up by the data flow in one direction and its acknowledgements between the two endpoints). The negotiation procedure is reliable.

- Low per-packet overhead.

  DCCP's packet header has a condensed form, with only a 12-byte generic header (Figure 3-4). It offers an option mechanism similar to TCP, which is appended to the end of DCCP header and used for acknowledgement reporting and parameter negotiation.

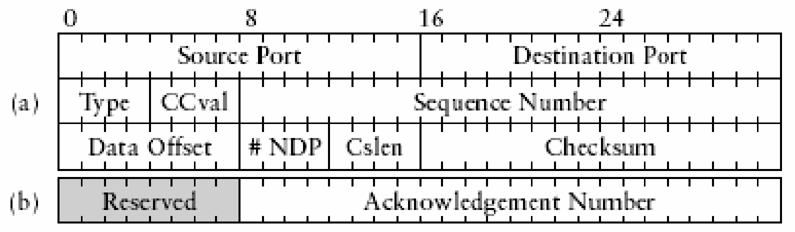


Figure 3-4 DCCP Generic Header Format [17]

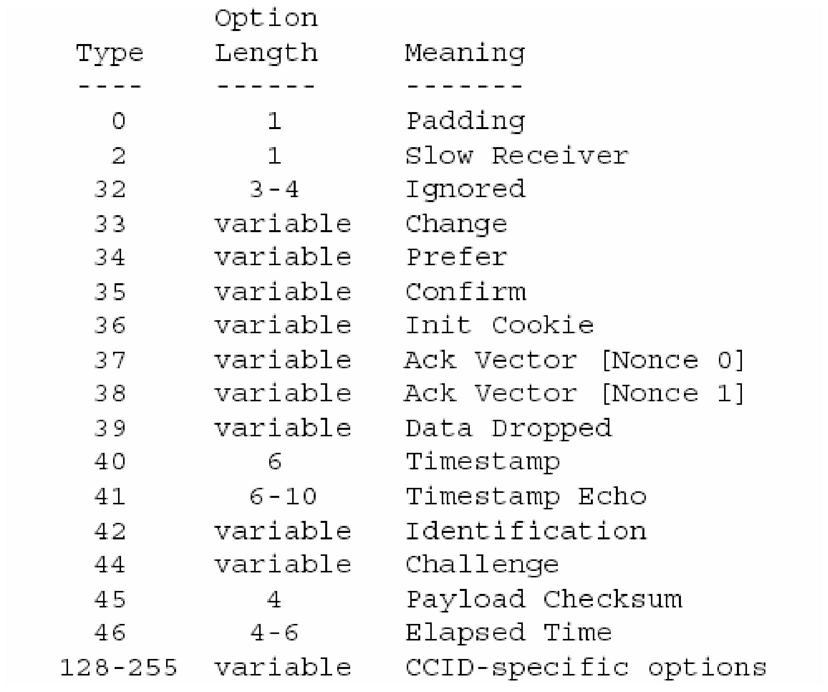
3.3.2 Main Differences of DCCP from TCP

As stated above, DCCP provides unreliable delivery, generic feature negotiation mechanism and choices for congestion control, which are different from TCP. Besides these three features, there are other differences between DCCP and TCP.

- DCCP is a packet stream protocol, and the application is responsible for packet framing. The sequence number hence refers to packets, not bytes.
- DCCP has flexible acknowledgement formats. Options used in acknowledgments are specified in a CCID and negotiated during the connection setup.
- DCCP can potentially distinguish between different reasons for a packet loss. DCCP supports ECN and a Data Dropped option, which help the sender to distinguish non-network-congestion caused packet losses.
- DCCP provides support for mobility. When the moving endpoint gets a new address, it sends a DCCP-Move packet to the other endpoint notifying it of its new address. The connection is changed to the new address accordingly. This is intended to solve only the simplest mobility problems.

3.3.3 Introduction of Options and Features

Options can be contained in all DCCP packets. Options occupy space at the end of the DCCP header and are a multiple of 8 bits, totally up to 1020 bytes in length. Options, which are currently defined in the DCCP Internet Draft, are listed in Figure 3-5.



Type	Option Length	Meaning
0	1	Padding
2	1	Slow Receiver
32	3-4	Ignored
33	variable	Change
34	variable	Prefer
35	variable	Confirm
36	variable	Init Cookie
37	variable	Ack Vector [Nonce 0]
38	variable	Ack Vector [Nonce 1]
39	variable	Data Dropped
40	6	Timestamp
41	6-10	Timestamp Echo
42	variable	Identification
44	variable	Challenge
45	4	Payload Checksum
46	4-6	Elapsed Time
128-255	variable	CCID-specific options

Figure 3-5 DCCP Options [17]

DCCP contains a generic mechanism for reliable feature negotiation. Three options, Change, Prefer and Confirm, are used to implement feature negotiation. Change is used to ask a DCCP endpoint to change a feature's value, and the DCCP endpoint can reply with a Prefer, which asks the other endpoint to change to another value for a feature, or with a Confirm to acknowledge the request. Negotiation of multiple features may take place simultaneously, i.e. a packet can contain multiple Change options for different features.

Features are identified by numbers, which are the first data byte in every Change, Prefer and Confirm options. The currently defined feature numbers are listed in Figure 3-6.

		Value	Initial
Number	Meaning	Type	Value
------	-------	-----	-----
1	Congestion Control ID (CCID)	SP	2
2	ECN Capable	SP	1
3	Ack Ratio	NN	2
4	Use Ack Vector	SP	0
5	Mobility Capable	SP	0
6	Loss Window	NN	1000
7	Connection Nonce	NN	random
8	Identification Regime	SP	1
9	Mobility ID	NN	0
128-255	CCID-specific features	?	?

Figure 3-6 DCCP Feature [17]

The negotiation procedure is the same for all features. Features have different value types and currently all features fit one of the two value types: non-negotiable ("NN") or server-priority ("SP"). The NN features are set by Change options containing exactly one feature value from the remote side, and are confirmed with a Confirm option by the feature location. The SP features are set by the Change, Prefer and Confirm negotiation options, which contain a prioritized list of values. The server priority rule means that the first entry in the server's list, which is also in the client's list, will be selected.

DCCP has special capabilities provided by options and feature, which may be used in many applications. Characteristics of congestion control, RTT estimation, acknowledgement and flow control are introduced below.

Congestion Control:

- Applications have the choice of using a specific congestion control scheme.
- Each congestion control mechanism has an identifier (CCID) from 0-255.
- In each CCID, the following procedures are described: how a half connection (HC) sender limits data packet rates; how to maintain connection parameters, such as *cwnd*; how a HC receiver sends back ACKs as the congestion information and how to manage the ACK rate.
- CCID allows CCID-specific options and features in addition to the global ones.

RTT Estimation:

Three options can be used for the RTT estimation.

- Timestamp option. This option has four bytes of option data carrying the timestamp of this packet. A Timestamp Echo option should be returned upon receiving a Timestamp option.
- Timestamp Echo option. Generally, a DCCP endpoint should send one Timestamp Echo option for each Timestamp option it receives. This option can carry four to ten bytes of option data, in which four bytes are the Timestamp taken from the Timestamp option, and another four to six bytes are the Elapsed time, indicating the amount of time elapsed since receiving the packet whose timestamp is being echoed. This is used to separate the RTT from processing time.
- Elapsed Time option. This option is permitted in any DCCP packet with an Acknowledgement Number. It indicates how much time has elapsed since the packet being acknowledged was received (packet is the one shown in the

Acknowledgement number). This may help correct RTT estimates when the gap between receiving a packet and acknowledging that packet is long (such as when acknowledgements are sent not frequently).

Acknowledgements:

Acknowledgements are used to report to the sender the network congestion situation. The CCID specifies which options are needed in an ACK. To assure ACK reliability, ACK of ACK is used to free the states at the ACK sender after confirming the reception.

- ACK-Radio Feature (3): Send one ACK per R data packets with default of 2.
- ACK Vector Feature (4): lets DCCP negotiate whether they should use ACK Vector options. ACK Vector provides detailed loss information for each packet.
- ACK Vector option: ACK vector gives a run-length encoded history of data packets received at the client. It can report whether a packet is received, ECN marked or not yet received.

Flow Control:

DCCP has two flow control mechanisms:

- Slow Receiver Option: tells the sender to not increase sending rate for 1 RTT after seeing this.
- Data Dropped Option: provides precise feedback about which packets were dropped and why.

3.3.4 Conclusion

DCCP is a connection oriented transport layer protocol, which provides the basic functionalities for connection setup, maintenance, and teardown. It also provides choices of congestion control mechanisms to applications. DCCP's feature negotiation and variety of options make it possible to be used over widely different networks.

3.4 Summary

In this chapter, recent transport layer protocols are introduced, including ATP, FAST TCP and DCCP. As a general-purpose transport layer protocol, DCCP provides a standard way to implement and choose congestion control mechanisms based on application's needs, a generic feature negotiation mechanism for reliable feature value selection, and support of ECN and other options to identify non-network-congestion losses. DCCP provides unreliable delivery but with detailed and reliable acknowledgements, which is achieved by sending ACK of ACK to the receiver by the sender. These special characteristics of DCCP present a potential of using DCCP in Wireless Ad Hoc Access Networks with a specially designed congestion control algorithm.

Bandwidth estimation or rate control is the essential part in congestion control. Several bandwidth estimation techniques are studied in this chapter. Delay-based rate estimation approaches are used in TCP Vegas, FAST TCP, TCP Westwood and TIBET. Studies of these algorithms showed improvements in throughputs as well as fairness compared with TCP Reno/NewReno under several conditions. In this thesis, this delay-based bandwidth estimation is used in the congestion control for a Wireless Ad Hoc Access Network.

Chapter 4

DCCP with Congestion Control for Combined Wireless Ad Hoc Access Networks

The goal of the research is to propose a transport layer protocol with suitable congestion control mechanism, which addresses the throughput and fairness issues when used in a combined wireless and wired scenario shown in the scenario B of Figure 2-4. To design a robust and effective approach, there are several problems that need to be considered.

- **Does the congestion control mechanism maintain end-to-end semantics?**
  The wireless and wired networks are very different with respect to packet transmission delay and error rates. Some schemes add special functions at the gateway, which works as a proxy between the wired part and wireless part. In I-TCP (Indirect TCP) protocol [3] (discussed in the Appendix), the fixed host from the wired part only sees an image of the mobile host on the gateway, which sends the acknowledgements to the sender on behalf of the receiver and maintains a packet buffer to retransmit lost packets according to the receiver's requests. In this approach, the transport protocol's end-to-end semantics is violated, thus acknowledgements do not reflect the whole route condition and cannot be used for end-to-end congestion control. It is desirable to maintain end-to-end semantics in the congestion control algorithm.

- **How to recognize the exact reason of packet loss?**

  The main problem of TCP is that a TCP sender takes every packet loss as an indication of network congestion and triggers the exponential backoff procedure unnecessarily. The ideal behavior of the protocol is to recognize the true reason of packet losses due to congestion, network partition or extensive error condition, and respond accordingly.

- **Who is responsible for network failure detection?**

  In order to respond to the network condition accurately, the sender needs a network failure detection scheme. If network detection is used, like in TCP-ELFN [11], TCP-Feedback [6] and ATCP [9], intermediate nodes can provide fast and accurate network information, but the sender may fail to receive this information because of the route disconnection to the next hop. Fixed RTO [2] and TCP DOOR [26] use end nodes to detect network problems, in which there is no overhead to send information back to the sender, but the inferred detection may not be accurate. Therefore, a hybrid approach is needed.

- **Should probe packets be used upon detection of route reestablishment?**

  Probe packets induce network congestion when multiple flows are probing the network at the same time [2], but the sender can resume faster after the route is reestablished with the utilization of probe packets. If the sending rate is carefully adjusted, the use of probe packets after a route failure is beneficial. Also, probe packets can report extra information about the newly established route, which could be different from the obsolete route. Probe packets are useful in Wireless

Ad Hoc networks, and the sending rate of probe packets should be carefully studied to avoid causing network congestion.

- **Should cumulative acknowledgements be used?**

  Using cumulative ACKs, both the lost and the following in-sequence packets need to be retransmitted; hence the goodput, which is the actual throughput that receivers received without the retransmissions, is affected in the bandwidth-constrained wireless network. Also cumulative ACKs cause burstiness. Acknowledgements are important for reliable packet transmission, so non-cumulative acknowledgements, without a close relationship with timely packet transmission, are desirable, such as SACK.

- **What rate should be used after a new route is established?**

  Route changes during packet transmission happen often due to the mobility of the mobile nodes. After a new route is established, if the old transmission rate prior to the invoked congestion control is selected, congestion is likely to happen again when the new route has a smaller bottleneck. The other choice is to probe the route from the slow start phase, which wastes the bandwidth significantly if performed every time a new route is selected, as show in [2].

  To solve this problem, equation-based congestion control is considered. With RTT or maximum delay in the route (used by ATP) as the feedback information, the sending rate calculated based on the above information is used as the initial sending rate for the new route to avoid slow start. The primary goal of the equation-based congestion control is not to aggressively find and use available bandwidth, but to maintain a relatively steady sending rate while still being responsive to congestion. So the tradeoff of this equation-based congestion control is that the sending rate change is slower than TCP-like congestion control, therefore its response to a sudden increase in available bandwidth is slower. In an environment where route changes happen often, using this equation-based congestion control to avoid frequent slow start phases should be able to achieve a better performance.

- **Fairness among flows and friendliness towards standard TCP should be studied.**

  The congestion control algorithm should allow flows of the same type to have a fair share of the bandwidth. Because of the large amount of TCP flows in the internet, it should also be friendly towards TCP flows.

4.1 Congestion Control Protocol Design

TCP is a widely used transport layer protocol. To make TCP work well in Wireless Ad Hoc Networks, most approaches suggest modifications to TCP, which could be a large amount of work considering the wide distribution of TCP.

Based on the discussion above, the proposed transport layer protocol with congestion control will maintain end-to-end congestion control semantics, which do not utilize a proxy between the wired and wireless connections to act as the receiver for a connection. The proposed solution utilizes DCCP with the congestion control mechanism specified in a new CCID. The new CCID profile should define when acknowledgments are sent and which options must be used. Acknowledgements with ACK Vector Option and Data

Dropped Option are used to identify the true reasons of packet loss. Additional ECN support and ELFN support is used to provide network-detected information to the sender. To estimate the available end-to-end bandwidth, the equation used in FAST TCP is chosen for the thesis. FAST TCP uses a delay-based rate estimation mechanism based on stabilized Vegas to eliminate the unstable condition when large delays are present. TCP Vegas and FAST TCP are designed for high bandwidth networks, and their bandwidth estimation is calculated based on the queuing delay. In the thesis, a similar bandwidth estimation mechanism is chosen to study the throughput and fairness issues in a Wireless Ad Hoc Access Network.

DCCP does not support reliable transmission, but it has an ACK Vector Option to convey packet-reception information back to the sender, which can be used by applications for packet retransmission.

The ACK Vector Option gives a run-length encoded history of data packets received at the client. Each byte of the vector gives the state of that data packet in the loss history, and the number of the preceding packets with the same state [17]. The ACK Vector Option's data format is shown in Figure 4-1:

```
+--------+--------+--------+--------+--------+--------
|001001??| Length |SSLLLLLL|SSLLLLLL|SSLLLLLL|  ...
+--------+--------+--------+--------+--------+--------
Type=37/38         \___________ Vector ___________...
```

Figure 4-1 DCCP ACK Vector Option [17]

In Figure 4-1, the SS in each byte of the vector part indicates the state, which can be:

0 Packet received (and not ECN marked).

1 Packet received ECN marked.

2 Reserved.

3 Packet not yet received.

The first byte in the first ACK vector option refers to the packet indicated in the acknowledgment number.

With ECN support, ACK Vectors return the network-detected congestion information to the source node. In addition to the ACK vector, DCCP also has a Data Dropped Option, which indicates that some packets reported as received actually had their data dropped before reaching the application. With the help of the Data Dropped option, the sender is able to further differentiate between network loss and endpoint loss, so the sender's congestion control mechanism might be different. How the sender responds to the Data Dropped Option can be defined in the CCID. If the dropped packet is ECN marked, the sender must respond to the ECN mark. The format of this option is shown in Figure 4-2:

```
+--------+--------+--------+--------+--------+--------
|00100111| Length | Block  | Block  | Block  |  ...
+--------+--------+--------+--------+--------+--------
 Type=39          \___________ Vector ___________ ...
```

Figure 4-2 Data Dropped Option 1 [17]

The vector consists of a series of blocks, which correspond to one of the choices shown in Figure 4-3:

```
 0 1 2 3 4 5 6 7                  0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+                +-+-+-+-+-+-+-+-+
|0| Run Length  |       or       |1|Dr St|Run Len|
+-+-+-+-+-+-+-+-+                +-+-+-+-+-+-+-+-+
  Normal Block                      Drop Block
```

Figure 4-3 Data Dropped Option 2 [17]

Normal blocks, with first bit 0, indicate that any received packets in the Run Length have been delivered to the application. Dropped Blocks, with first bit 1, indicate that received packets in the Run Length were not delivered as usual. The reasons are indicated by the three Dr[op] St[ate] bits:

0 Packet data dropped due to protocol constraints

1 Packet data dropped in the receiver buffer

2 Packet data dropped due to corruption.

3 Packet data corrupted, but delivered to the application.

4 Packet data dropped because the application is no longer listening.

5-7 Reserved.

With the detailed packet drop information, a sender can treat random errors differently by not decreasing *cwnd* as dramatically as with a drop due to congestion.

4.2 State Diagram and Flow Chart

In our proposal, the sender has four states: Normal State, Congestion State, Failure State (route change or link failure) and Error State (transmission error). After the connection setup, the sender is in the Normal State. Upon receiving an ACK, the sender will check the ACK Vector Option, data dropped option and ELFN (may not be supported by certain routing protocols). Assuming that ECN is supported and ELFN is not supported, the processing flow is shown in Figure 4-4.

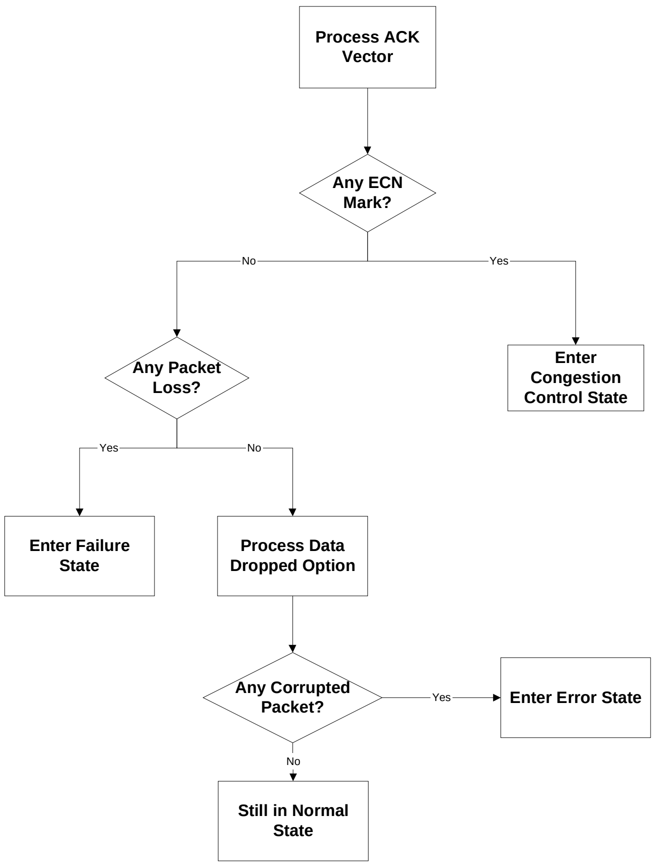


Figure 4-4 Flow Chart of Processing ACK

The state transition diagram is as follow (with or without ELFN):

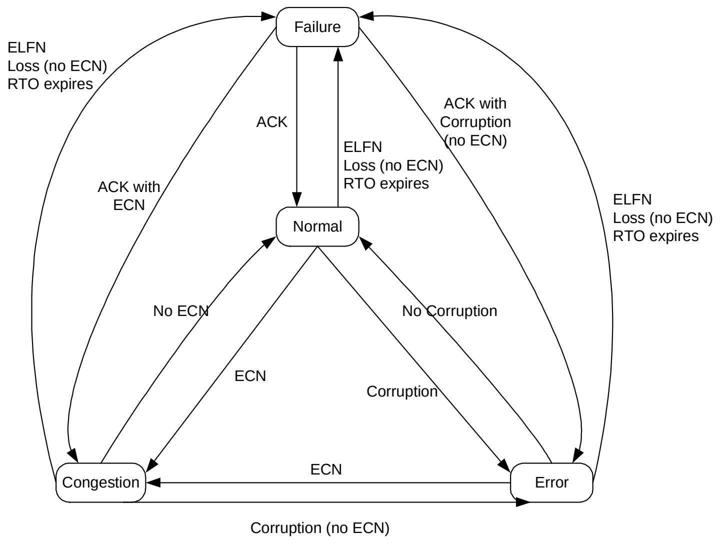


Figure 4-5 State Diagram of the Congestion Control

As described before, rate-based congestion control is used to avoid the frequent slow starts. The most important task is to design the rate equation for each state, which is the key for throughput and fairness.

In the research of ATP [25], the packet queuing and sending delay at each node is calculated and the maximum delay is recorded in each packet. The receiver then calculates the rate based on the delay information and feeds it back to the sender. This approach was studied on standalone Wireless Ad Hoc networks without an access point or gateway connecting to the wired networks. When cooperating with a wired network, the relationships between the delay and rate are different in the wired and wireless parts, so the receiver cannot make decisions without knowing where the maximum delay happened. Also, intermediate nodes are working as routers, which process packets up to the network layer. To record the delay information at each node through the path and later to be used at the receiver for transport layer, additional effort is needed to make changes at the intermediate nodes. So, the ATP approach is excluded from our solution.

Similar to FAST TCP, in the thesis, the sender maintains two RTT values, one is base RTT (baseRTT), which is the minimum recorded RTT, and the other is exponentially averaged RTT (avgRTT). Each time the sender goes into the failure state, the baseRTT will be reset by the round trip time of the probe packet and its corresponding acknowledgment, after being temporarily saved as old baseRTT. The sending rate after the route establishment is proportional to baseRTT/oldbaseRTT.

In the Normal State, the sender adjusts the rate proportional to baseRTT/avgRTT. In the congestion state, when ECN mark without packet loss happened, the rate adjustment is the same as in normal state. But when packet loss happened, the sending rate will halve. This idea is based on FAST TCP for High-Speed Long-Distance Networks, which showed proportional fairness under no congestion or mild congested situations when packet loss occurs infrequently.

In the Error State, the rate can be β*rate, calculated using the above equation, where β is from ½ to 1 according to the error rate.

In the Failure State, probe packets are send out to monitor the network situation. The rate of sending probe packets can be set to one packet per RTO like in Fixed RTO, but it should be studied further by experiments.

A simplified DCCP with rate-based congestion control is implemented based on the TCP implementation in NS2. Because wireless nodes do not support ECN and the limitation of getting network-detected link failure in NS2, the implementation has only two states: Congestion State and Normal State.

In the implementation, ACKs are sent back to the sender whenever the receiver receives a packet. ACKs have the ACK Vector option as specified in the DCCP specification. ACK vectors contain the packet information (whether they are received, not received or ECN marked). Also, ACK Vector can be used to bring back information of several packets to make sure the sender receives the acknowledgments though some ACKs may be lost. A weighted average RTT (3/4* RTT +1/4*current RTT) is calculated using the timestamp echo contained in the ACKs. The congestion window size (*cwnd*) is adjusted accordingly using the control equation. The function used in the simulation is:

$$cwnd = \{cwnd + (1 + (\text{int})(1 - cwnd * \frac{qdelay}{RTT}))\}$$

In the equation, *qdelay* is the difference between newly calculated weighted average RTT and baseRTT, which is calculated by weighted average RTT minus baseRTT. When *cwnd* is 1, the equation will increase *cwnd* by 1each RTT; when *qdelay* is zero, the *cwnd* is higher than 1, the equation will increase by 2 packets per RTT.

A timeout timer (RTO) is set for the transmitted packets. Since the test scenario is static, and no movement-caused packet drops are involved, the sender enters Congestion State whenever the timeout timer expires. In this Congestion State, probe packets with only headers are sent by the sender every RTO until an ACK is received. Upon successfully receiving an ACK, the sender resets the RTT and baseRTT, sets the congestion window size to *cwnd**oldBaseRTT/baseRTT, and enters the Normal State again.

An alternative design is to reset the *cwnd* every RTT based on the average RTTs collected and keep the same *cwnd* for this RTT as in FAST TCP. This way of changing *cwnd* is similar to the idea of TCP, so it may provide a fairer sharing between DCCP and TCP flows in some cases. The advantages of the first design are that the unfairness caused by different RTT between flows is removed, and it may be more suitable for the wireless situations where mobility is involved.

4.3 Additional Considerations

Additional features, which the CCID may consider, are discussed in this section.

4.3.1 ELFN

In the RFC of DCCP [17], there is no mention of ELFN support in DCCP. If ELFN is supported, it should be considered in the CCID. Whether or not a change to DCCP is needed should be further studied.

4.3.2 Retransmission and In-order-delivery

Because DCCP does not provide packet retransmission and in-order-delivery, a sub-layer implemented above the transport layer may be needed to provide such services, or it can be left to the applications.

4.3.3 TCP Friendly

TCP is the most widely used protocol on the Internet. Any new congestion control mechanism should not cause the TCP flows to suffer from bandwidth starvation. The rate equation used in the thesis should be carefully studied to fulfill this requirement.

4.4 Summary

Our proposal of the congestion control algorithm used for Wireless Access Ad Hoc Networks are presented in this chapter. In the thesis, the detected network condition will set the sender to different states, and the sender will adjust the sending rate accordingly. Network conditions, i.e. congested and failed, can be detected through the use of ECN, lower layer assistance, such as ICMP, and carefully adjusted probe packets. Those probe packets can also be used to detect the RTT on a newly established route after the failure state and set the sending rate based on the new RTT and previous RTT. This will improve the utilization of the link and decrease the possibility to overflow the bottleneck link of the new route. The congestion control algorithm maintains an end-to-end congestion control semantics, and it should meet the following goals:

- Provide fair bandwidth share to flows of the same type regardless the directions of flows.
- The fairness among flows using newly designed congestion control algorithm and TCP flows should not worse than the fairness among TCP flows.
- The newly designed congestion control algorithm should also not decrease the bandwidth utilization.

Our proposal uses DCCP as the base protocol and implements the congestion control algorithm as a new CCID within the DCCP framework. Simulation results of our proposal will be presented in later chapters for evaluations.

Chapter 5

Performance Evaluation Criteria

5.1 Evaluation Criteria

To evaluate the performance of the designed congestion control mechanism with DCCP in a combined Wireless Ad Hoc Access Network, the following criteria can be used:

- **Throughput**

  Throughput is defined as the average data rate of a source sending packets and received by the receiver.

- **Goodput**

  Goodput is the effective data rate as observed by the user. It is the actual throughput that receivers received without the retransmissions.

- **Fairness**

  Bandwidth sharing between flows of the same type.

5.2 Fairness

The Internet can be considered as a set of links with finite capacities as resources and a finite number of sources as network users. To avoid congestion and packet loss in the network, the rate allocated to each flow from sender to receiver should be regulated. The objective of bandwidth sharing or rate allocation is generally to use all available bandwidth to the fullest while maintaining “fairness” among flows without causing starvation of any flow. Fairness has many definitions; each fairness criterion regulates the bandwidth on a different basis. The most simplified is to allocate the same share of bandwidth to each connection.

As a mathematical notion, fairness is presented as an optimization problem, with the goal of finding how to allocate the bandwidth that maximizes the sum of each utilities specified by a certain fairness criterion. This optimization approach generalizes the concept of fairness.

5.2.1 Fairness Criteria

A fairness criterion describes how to allocate bandwidth fairly and how to measure it. The basic fairness criteria are throughput maximization, max-min fairness, proportional fairness and potential delay minimization.

- Maximum throughput

  The objective of the bandwidth allocation in the maximum throughput fairness criterion is to find the feasible rate allocation that maximizes the total throughput and uses the network resources efficiently.

- Max-min fairness

  Max-min fairness is the most commonly used definition for the concept of fairness. The objective is to maximize the minimum of the given bandwidths. Each of the users in the network has different demands but equal right to the resource. The Max-min fairness scheme first allocates users with smaller demands, and then evenly distributes unused resources to the users with higher demands.

- Proportional Fairness

In proportional fairness, each session has a utility function, which is increasing concave and continuous. The proportional fair allocation is the set of rates that maximizes the aggregate without links used beyond capacity. It maximizes the aggregate utility (optimum).

- Potential Delay Minimization

The objective of potential delay minimization is to minimize the delay time used to complete the full data transmission. This overall delay is inversely proportional to the sending rate of the source, so the nature of potential delay minimization is the maximization of throughput.

5.2.2 Fairness Index

A fairness index is defined as a function of variability of throughput across users. Jain's fairness index and the Max-min fairness index are the most widely used fairness measures.

- Max-min Fairness Index

The Max-min fairness index is defined as follows:

$$U = \max_i \left| \frac{A_i - F_i}{F_i} \right|$$

where U is the maximum unfairness, A_i is user i's actual resource usage and F_i is user i's max-min fair share allocation of the resource. A value of 0 indicates a completely fair system.

- Jain's Fairness Index [14]

Jain’s fairness index, ranging from 0 to 1, is defined as follows:

$$U = \frac{(\sum_{i=1}^{n} x_i)^2}{n\sum_{i=1}^{n} x_i^2}$$

where x_i is the normalized share allocation, which equals the actual allocation divided by the optimal allocation (use any fairness criterion). An index value of 1 indicates a completely fair condition.

Fairness is most frequently considered in a static regime where a fixed set of source-destination pairs share network resources for the transfer of infinite sized documents. In reality, the number of flows in progress is highly dynamic. There is a very strong interaction between the stochastic process describing the numbers of flows in progress on different network routes and the way in which these flows share bandwidth. Very little is known about the way fairness affects the performance perceived by users.

Jain’s fairness index is used in this thesis to evaluate the bandwidth sharing fairness among flows, because in the simulations, all flows should have equal share of the available bandwidth, which can be easily reflected by Jain’s fairness index. Equally sharing of bandwidth also simplified the normalization in Jain’s fairness index calculation, because the optimal allocation for each flow is the same, so the x_i in above equation becomes the actual bandwidth obtained by each flow.

Chapter 6

Simulation Results and Analysis

The Network Simulator 2 (NS2) is used to conduct preliminary simulations and to demonstrate the performance of our proposal. NS2 is a discrete event simulator for networking research and provides simulation support for transport protocols, routing and multicast protocols over wired and wireless networks.

The test scenario used in the simulation is shown in Figure 6-1. In the simulation, data are sent from the wired node to the wireless nodes (1 and 4) that are two hops away from the access point; or from those wireless nodes to the wired node via the access point. All wireless nodes are stationary in the simulation. All data flows are 10MB FTP flows.

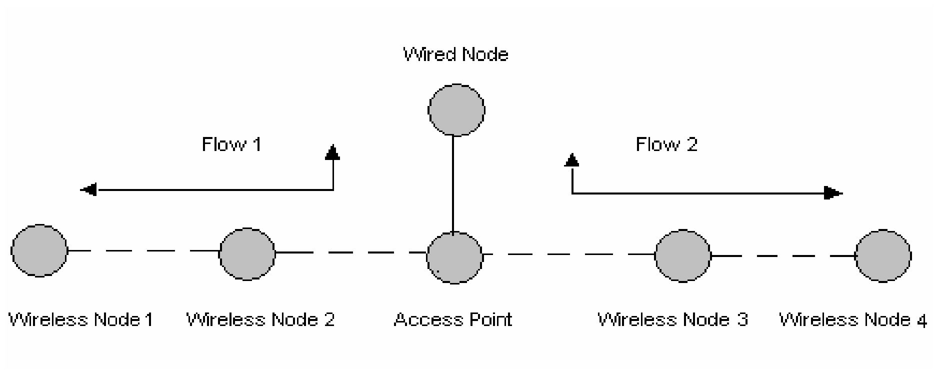


Figure 6-1 Wireless Ad Hoc Access Network

All simulations in the thesis are conducted multiple times to verify the coding. The results provided in the thesis are from single simulation run results. Capturing and analyzing simulation results of multiple runs can be conducted in future works.

The goal of simulations in this Chapter is to compare performance between the proposal congestion control algorithm and TCP's congestion control algorithm. Fairness index is the main evaluation criteria in the comparison of Unreliable DCCP flow with TCP flow. Throughputs are considered only as an indication of the possible bandwidth utilization in the simulations. Throughputs of reliable DCCP flows and TCP flows are compared in Chapter 7.

6.1 Throughput Limitation

In IEEE 802.11, the cycle of packet transmission is shown in Figure 6-2 when RTS and CTS are used.

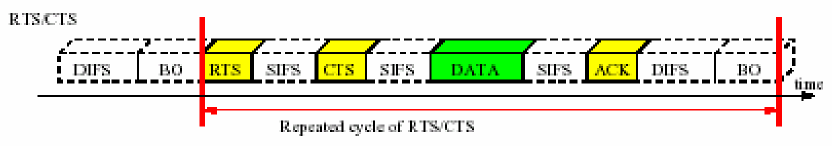


Figure 6-2 Transmission for CSMS/CA with RTS/CTS

Control frames (RTS, CTS and ACK) and inter-frame timing (SIFS, DIFS and Backoff) affect the maximum throughput offered by the IEEE 802.11 networks. The maximum throughput can be calculated as:

$$Throughput = \frac{MSDU\ \ Size}{Delay\ \ per\ MSDU}$$

MSDU is the MAC layer Service Data Unit; the delay per MSDU is the sum of transmission of data, control bits, and all other delay components.

$$\text{Delay per MSDU} = T_{DIFS} + 3 * T_{SIFS} + T_{BO} + T_{RTS} + T_{CTS} + T_{ACK} + T_{DATA}$$

In NS2, the transmission rate for data is set to 11 Mbps and control information is send at 2Mbps; DIFS is 50 µs; SIFS is 10 µs; average backoff time is 310 µs (minimum contention window size 31, time slot 20µs); data packet length is 1540 bytes (including IP header and TCP/DCCP headers). The MAC header is 44 bytes, an RTS or CTS is 40 bytes long, and a MAC ACK is 38 bytes long. So the throughput per link is (assuming a propagation delay of 0 due to the close distance of nodes, relative to the speed of light):

$$\text{Throughput} = \frac{1540*8}{\frac{30+50+310}{10^6}+\frac{(2*40+38)*8}{2*10^6}+\frac{(1540+44)*8}{11*10^6}} = 6.12\text{Mbps}$$

When higher layer acknowledgment is needed, such as TCP, the throughput is lower because TCP ACKs are added into the transmission and have the same transmission cycle. So the throughput for a data flow which has an ACK of 40 bytes for each packet is:

Throughput =

$$\frac{1540*8}{\frac{2*(30+50+310)}{10^6}+\frac{[(2*40+38)*2)]*8}{2*10^6}+\frac{(1540+40+44*2)*8}{11*10^6}} = 4.19\text{Mbps}$$

In Ad Hoc networks, when packets traverse a chain of nodes, successive packets of the same connection contend with each other along the chain. In Figure 6-1, when packets are sent from wireless node 1 to the access point, wireless nodes 1 and 2 cannot transmit at the same time, so the throughput is only, at best, ½ of the maximum throughput. The throughput limit also decreases when the radios can interfere with each other beyond the range at which they can receive correctly. In the simulation, the correct receiving range is set to 200m with interference range of 350m, and the distance between two nodes is 150m. This means the transmission of one node can be received successfully by the nodes adjacent to it, but can prohibit the transmission of nodes two hops away from it. In the network shown in Figure 6-1, within the 5 nodes with radios (wireless nodes 1-4 and access point), if wireless node 1 and wireless node 2 are transmitting and receiving packets, the access point cannot transmit after it receives CTS from node 2, also wireless nodes 3 and 4 cannot transmit, because the RTS from node 4 will collide at node 3 with the signal from node 2. This situation indicates that at most only 1/4 of the single hop throughput is reached when the connection is from wireless node 1 to node 4.

Because of the backoff mechanism used in the IEEE 802.11 MAC, it is possible that when a node stops transmitting and leaves the media free, another node is still in the backed off condition and does not start the transmission until the backoff finishes. The throughput decreases further due to the wasted idle time. Also, nodes in the middle of the chain experience more contention than nodes at the ends of the chain, so the end nodes can transmit more packets than the middle nodes can forward. This uneven bandwidth allocation can cause packet drops by the middles nodes and affect the bandwidth. In [18], when the radio transmission rate is 2Mbps, the measured maximum throughput in a chain of nodes is only 1/7 of the single hop throughput where a maximum ¼ of the single hop throughput should be achieved theoretically.

To eliminate the affects of different MSDU header lengths, the payload size at the transport layer is used in all throughput calculations in the thesis from now. Using the same formulas and payload size, the maximum throughput for a flow without ACK is:

$$\text{Throughput} = \frac{1500*8}{\frac{30+50+310}{10^6}+\frac{(2*40+38)*8}{2*10^6}+\frac{(1540+44)*8}{11*10^6}} = 5.96\text{Mbps}$$

And the maximum throughput for a flow with ACK for each packet is:

Throughput =

$$\frac{1500*8}{\frac{2*(30+50+310)}{10^6}+\frac{[(2*40+38)*2)]*8}{2*10^6}+\frac{(1540+40+44*2)*8}{11*10^6}} = 4.09\text{Mbps}$$

In the simulation, different CBR flows using UDP are used first to test the throughput limit for the test setup. Each CBR flow sends data packets (1500 bytes) from wireless node 1 to node 4 using UDP. The tests showed that a maximum rate of 1.21 Mbps can be achieved (Table 6-1). The maximum throughput from these simulations is smaller than ¼ of the calculated throughput, this further indicates that the wireless media could not be fully utilized mainly because of sender's idle state due to backoffs.

CBR (Mbps)	1	1.5	2	5	6	8
Throughput (Mbps)	0.996	1.16	1.17	1.21	1.21	1.20

Table 6-1 UDP Flow Performance

10 MB FTP flows are used in the tests for TCP and DCCP with a packet size of 1500 Bytes. Table 6-2 shows the throughputs using TCP and DCCP with congestion control. Throughputs are calculated as the total payload size of all received packets within 1 second (including retransmitted packets for TCP).

	Total Pkts	**Received Pkts**	**Throughput**
TCP	6667	6682	0.412Mbps
DCCP	6667	5871	0.829Mbps

Table 6-2 TCP/DCCP One Flow Performance

The diagrams below show the changes of throughputs over the transmission time for both the single TCP flow and the single DCCP flow.

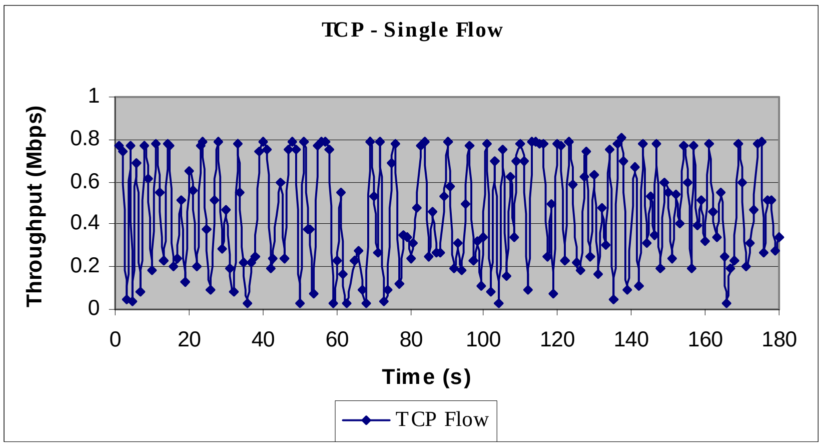


Figure 6-3 TCP Performance: Single TCP Flow

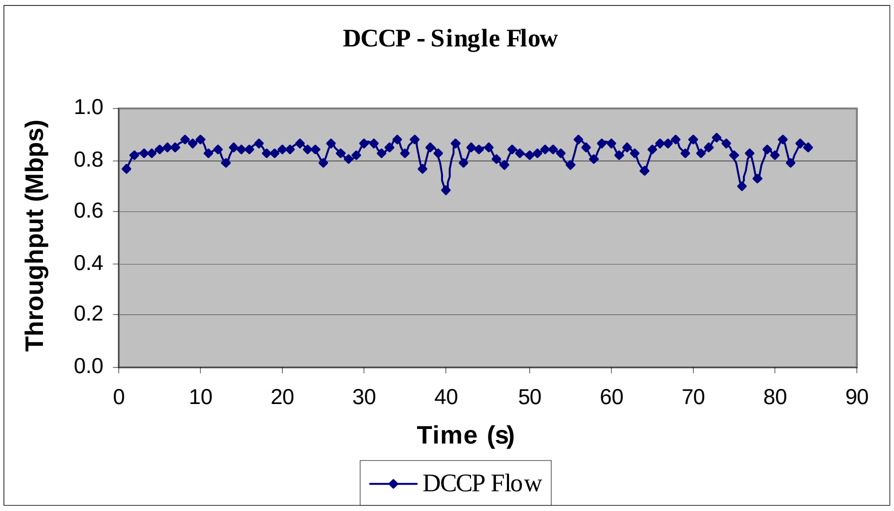


Figure 6-4 DCCP Performance: Single DCCP Flow

The NS2 tracing functions generated traces for packet handling events from the MAC layer to the transport layer on each node. Perl scripts were used to process traces, which selected packets at different layers for detailed study. By analyzing the simulation trace, it is found that the contention of the wireless medium causes TCP flows to backoff when the sender fails to receive CTS or ACKs in a timely manner. The TCP sender decreases *cwnd* and stops transmission in the backoff period, therefore the throughput over time fluctuates, and the overall throughput is decreased. On the contrary, the DCCP flow removes the Binary Exponential Backoff. It uses a constant backoff value and probing packets to detect the medium availability, and achieves a better throughput. DCCP throughput from the simulation is again less than ¼ of the calculated maximum throughput, this is also mainly due to the idleness on the wireless media because senders remain in backoff state when other nodes have stopped transmission.

6.2 TCP Performance

In the simulations, an IN flow is a connection from the wired node to the wireless nodes (1 or 4), and an OUT flow is a connection from the wireless nodes (1 or 4) to the wired node.

TCP performance is studied first to confirm the results shown in [29]. In the simulation, the radio data rate is set to 11 Mbps and throughputs are calculated as the total payload size of all received packets within 1 second (including retransmitted packets for TCP). The simulations show the same result as those reported in [29]. When the two flows are both IN (Figure 6-5), they share the medium almost fairly: one IN flow finished the transmission shortly after the other IN flow.

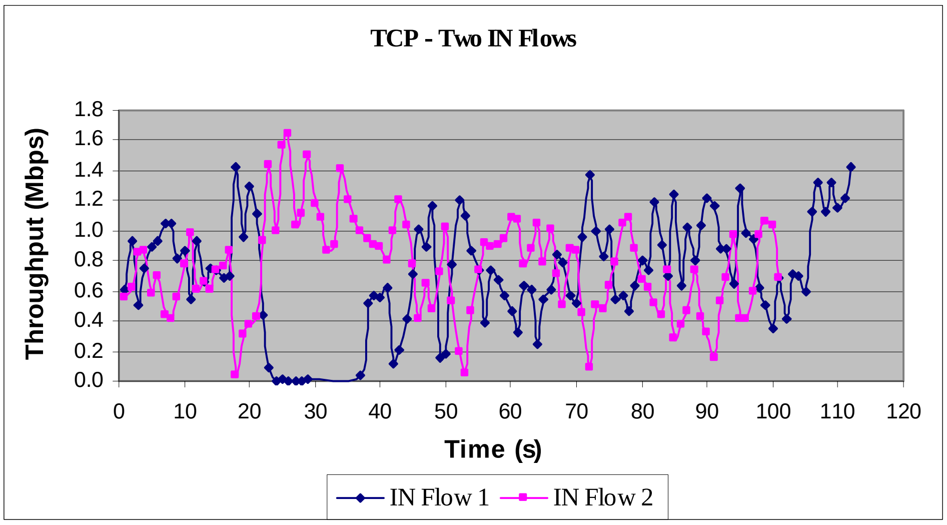


Figure 6-5 TCP Performance: Two IN Flows (RTT ≅ 110 ms)

When there are two OUT flows, the wireless channel is not shared fairly, as shown in Figure 6- 6, one OUT flow occupied the whole bandwidth after the other flow experienced losses and finished the transmission long before the other flow. The trace from the simulation shows that, after failing to receive an ACK in time for one OUT flow (here OUT flow 2), the sender of OUT flow 2 went into Exponential Backoff mode, and OUT flow1 obtained more bandwidth. With the re-transmission timer increasing exponentially, OUT flow 2 failed to contend with OUT flow 1 and remained quiet for an extended period of time. OUT flow 2 re-started transmissions after the backoff timer expires.

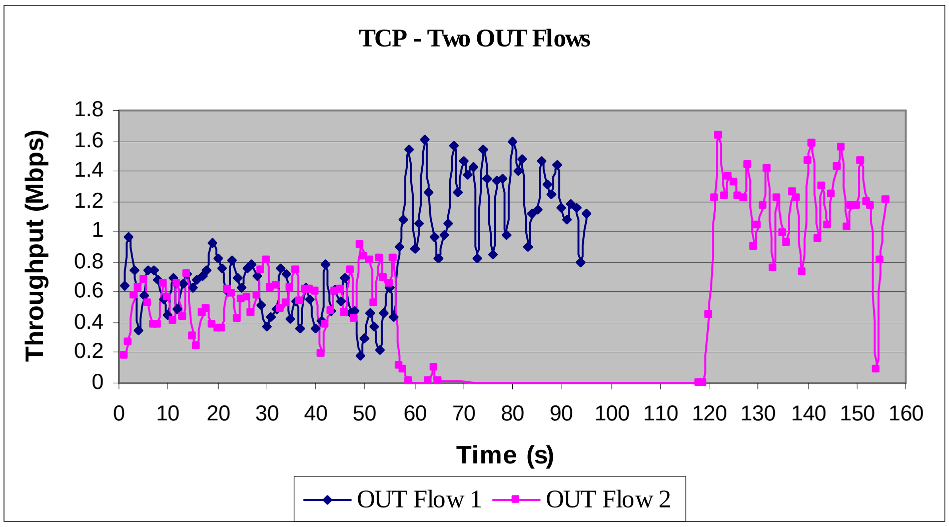


Figure 6-6 TCP Performance: Two OUT Flows (RTT ≅ 110 ms)

While one IN flow and one OUT flow co-exist, the IN flow uses more bandwidth than the OUT flow (Figure 6-7). By analyzing the trace file of the simulation, it is shown that, while the IN flow is transmitting from the wired node to wireless node 1, wireless node 4 is the hidden node and RTSs from node 4 collide at wireless node 3, so TCP packets of the OUT flow which are queued in node 4 experience losses as the result of MAC layer collisions; when the OUT flow is transmitting, nodes 1 and 2 have more ACKs dropped by the MAC layer. Because of the cumulative nature of ACKs in TCP, effects of ACK drops can be corrected by later ACKs. In contrast, drops of TCP packets will cause the TCP sender to backoff the transmission, and become inactive for longer periods, increasing the exponential backoff value. So the OUT flow transmits less than the IN flow as shown in Figure 6-7.

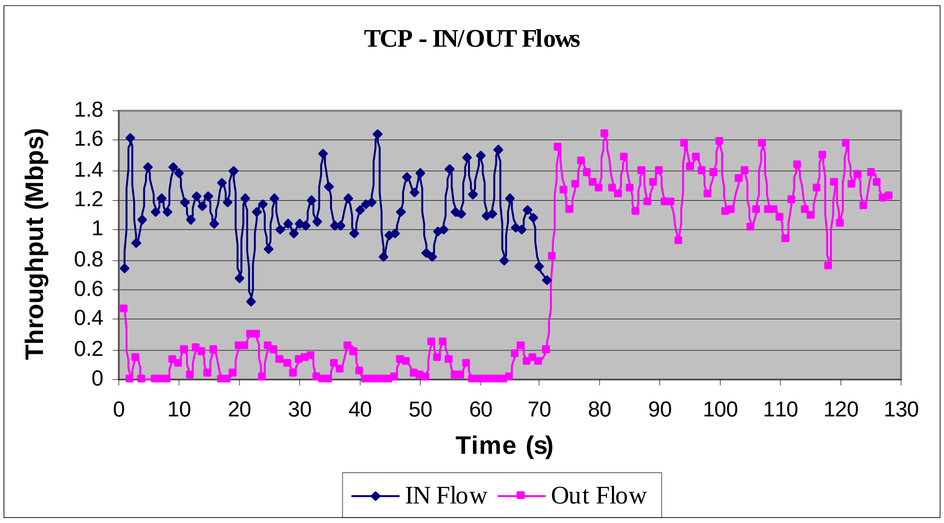


Figure 6-7 TCP Performance: IN Flow/OUT Flow (RTT ≅ 110 ms)

The unfairness becomes more severe when the RTT is shorter for the IN/OUT flow scenario. In Figure 6-8, when the RTT is around 60ms, the IN flow almost completely occupies the whole bandwidth, and the OUT flow only transmits after the IN flow finishes. With a shorter RTT, when the OUT flow experiences a packet loss, the IN flow gains a faster increase in the window size and uses more link capacity, and further causes the OUT flow to backoff.

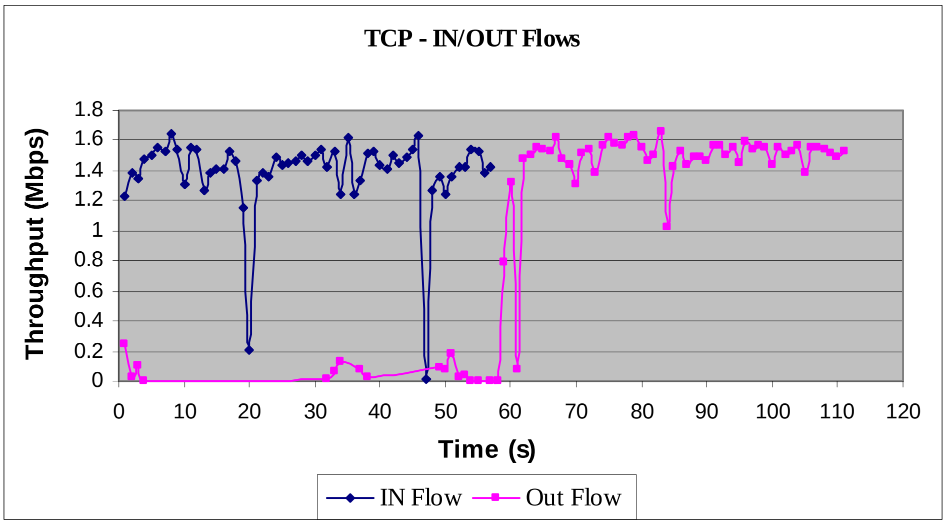


Figure 6-8 TCP Performance: IN Flow and OUT Flow (RTT ≅ 60 ms)

Table 6-3 lists the throughputs in the simulations and the fairness index calculated by Jain's Fairness Index. In the first part of the table, overall throughputs are the average long-term throughput by each flow, calculated based on the time needed by each flow to finish the 10M data transfer. The second part of the table lists the throughput of each flow during the period they are both active, i.e., based on the time it takes to complete the faster of the two flows. The Fairness Index calculations are obviously only meaningful in that latter case.

Overall Throughput (Mbps)	**Both IN (110ms)**	**Both OUT (110ms)**	**IN/OUT (110ms)**	**IN/OUT (60ms)**
Flow 1	0.70	0.84	1.12	1.37
Flow 2	0.76	0.49	0.62	0.71

Throughput (Mbps)	**Both IN (110ms)**	**Both OUT (110ms)**	**IN/OUT (110ms)**	**IN/OUT (60ms)**
Flow 1	0.65	0.84	1.12	1.37
Flow 2	0.76	0.33	0.10	0.02
Sum	1.41	1.17	1.21	1.38
Jain's Fairness Index	0.994	0.840	0.585	0.514

Table 6-3 TCP Performance: Throughputs and Fairness

6.3 Performance of DCCP with Congestion Control

Figures 6-9 to 6-12 show the simulation results of our proposed congestion control algorithm. In all situations (both IN, both OUT, and IN/OUT flows), the two flows present good long-term fairness on bandwidth sharing. There are two reasons contributing to the improvement in fairness. In the implementation, no in-order packet delivery and packet retransmission are required; this means all packets dropped along the route have no effects on later packet transmission. The second reason is, when timeout happens, the sender sends out probe packets every RTO, so there is no large inactive period caused by the exponential backoff timer. The severe unfairness shown by TCP during a shorter RTT is corrected. In Figure 6-12, the IN/OUT flows are sharing the bandwidth fairly when RTT is around 60ms. This shows that the long-term unfairness is not affected by the RTT.

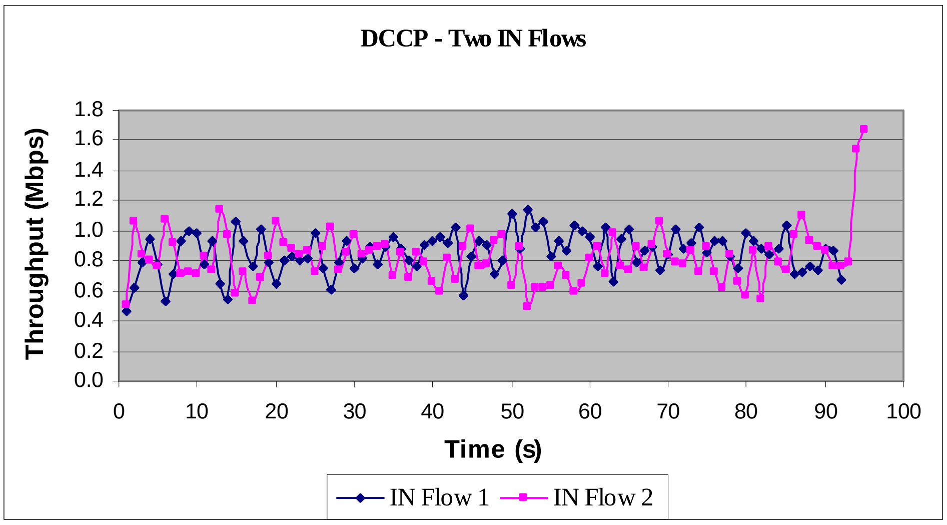


Figure 6-9 DCCP Performance: Two IN Flows (RTT ≅110 ms)

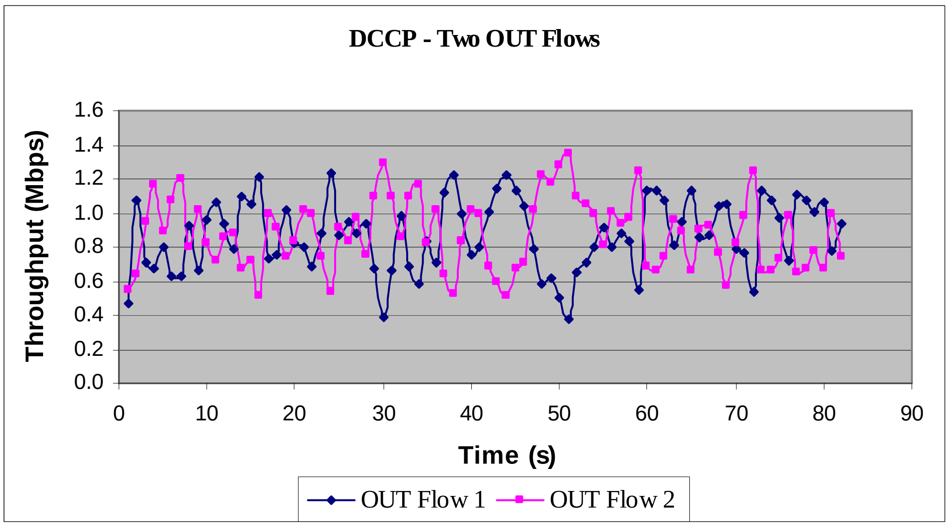


Figure 6-10 DCCP Performance: Two OUT Flows (RTT ≅ 110 ms)

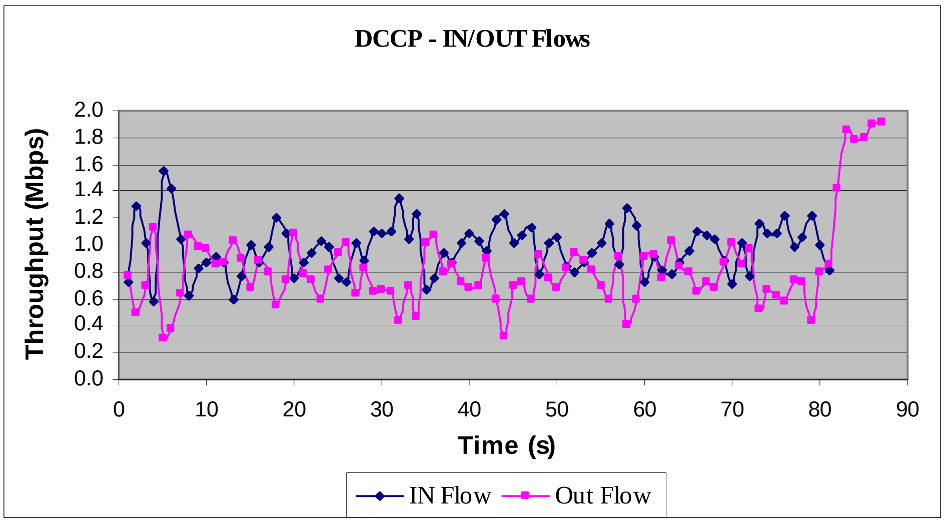


Figure 6-11 DCCP Performance: IN/OUT Flows (RTT ≅ 110 ms)

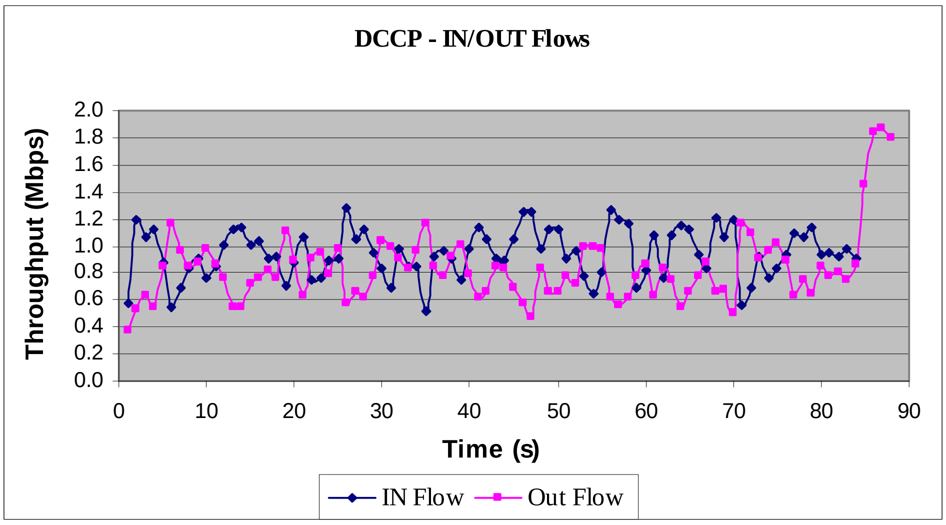


Figure 6-12 DCCP Performance: IN Flow/OUT Flow (RTT ≅ 60 ms)

The throughputs and fairness index for DCCP with congestion control are given in Table 6-4, which indicates the improvement in both the throughputs and fairness compared to TCP (Table 6-3). DCCP with congestion control offers higher throughputs because of less wasted time in backoff and smooth congestion window size adjustment used in the proposed congestion control mechanism. Instead of overflowing the channel and halving the congestion window size, the new congestion control monitors the delay and updates the window size on a per packet basis.

Overall Throughput (Mbps)	Both IN (110ms)	Both OUT (110ms)	IN/OUT (110ms)	IN/OUT (60ms)
Flow 1	0.85	0.87	0.97	0.93
Flow 2	0.83	0.87	0.83	0.83

Throughput (Mbps)	Both IN (110ms)	Both OUT (110ms)	IN/OUT (110ms)	IN/OUT (60ms)
Flow 1	0.85	0.87	0.97	0.93
Flow 2	0.80	0.87	0.77	0.80
Sum	1.66	1.74	1.73	1.73
Jain's Fairness Index	0.999	1.000	0.987	0.994

Table 6-4: Performance of DCCP with Congestion Control: Throughput and Fairness

6.4 Combined TCP and DCCP Flows

Compared with TCP’s congestion control, our proposed DCCP with congestion control algorithm shows improved inter-flow fairness on all combination of IN and OUT flows. In the current Internet, a majority of flows uses TCP and UDP, so it is also important to study the inter-protocol fairness between DCCP flows and TCP flows and between DCCP flows and UDP flows. In this section, the simulation results of combined TCP and DCCP flows are presented and discussed.

6.4.1 Both IN Flows

In the simulation of combined TCP and DCCP flows, when combined TCP IN and DCCP IN exist, the bandwidth sharing is different with different RTTs. Figure 6-13 shows the results of two IN flows when RTT is around 110ms and Figure 6-14 show the result when RTT is around 60ms. Figure 6-13 shows that, when the RTT is around 110 ms, the TCP IN flow and the DCCP IN flow share the medium almost fairly. Figure 6-14 shows that, when RTT was 60 ms, the TCP IN flow took more bandwidth.

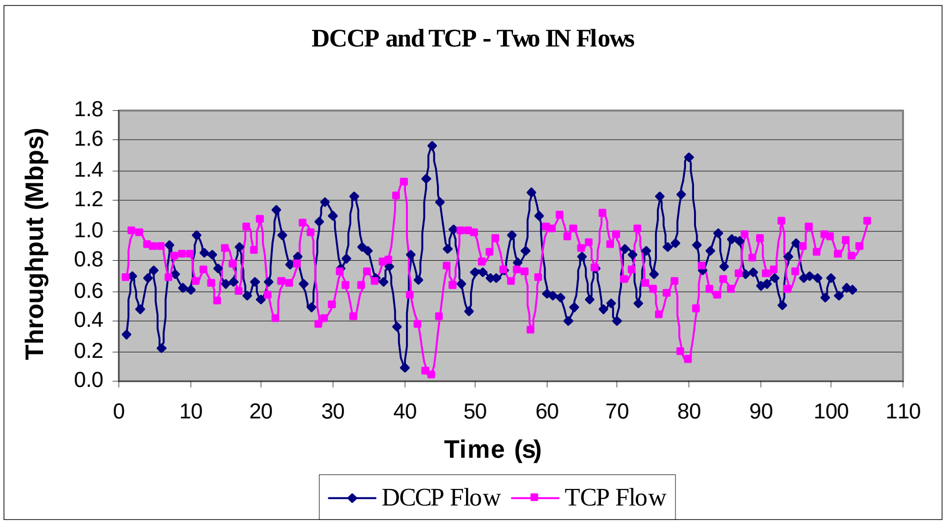


Figure 6-13 TCP/DCCP: Two IN Flows (RTT ≅ 110 ms)

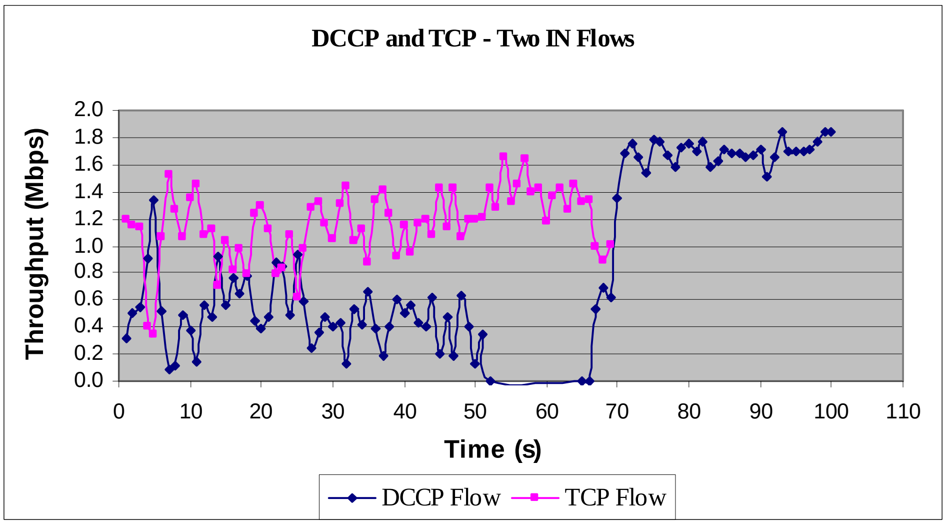


Figure 6-14 TCP/DCCP: Two IN Flows (RTT ≅ 60 ms)

The throughputs of the two flows get closer when the RTT is longer. This is because of the different congestion window size (*cwnd*) adjustment mechanisms used in TCP and DCCP congestion control. The *cwnd* is increased by 1 segment during the congestion avoidance phase in TCP and adjusted according to the delay of every received ACK in DCCP. Figure 6-15 shows the *cwnd* samples for the DCCP flow and Figure 6-16 shows the *cwnd* and slow start threshold samples for the TCP flow when RTT is around 110ms. Figure 6-17 shows the *cwnd* samples for the DCCP flow and Figure 6-18 shows the *cwnd* and slow start threshold samples for the TCP flow when RTT is around 60ms. TCP increases its *cwnd* slower when RTT is longer (Figures 6-15 and 6-16) and DCCP can get more bandwidth by working at a larger *cwnd*. When RTT is shorter (Figures 6-17 and 6-18), TCP’s *cwnd* increases faster and leaves less bandwidth to the DCCP flow, which is working at a smaller *cwnd*.

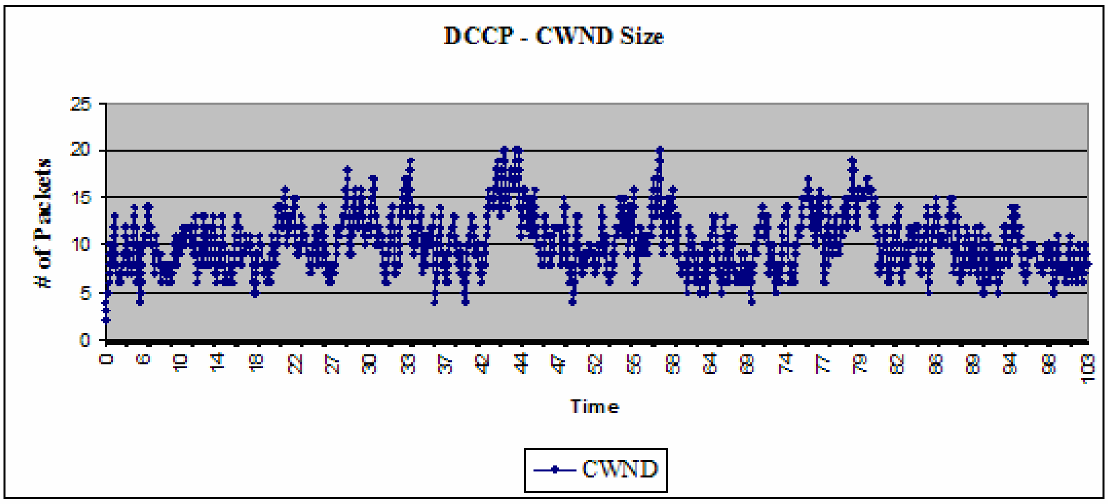


Figure 6-15 DCCP *cwnd* (RTT ≅ 110 ms)

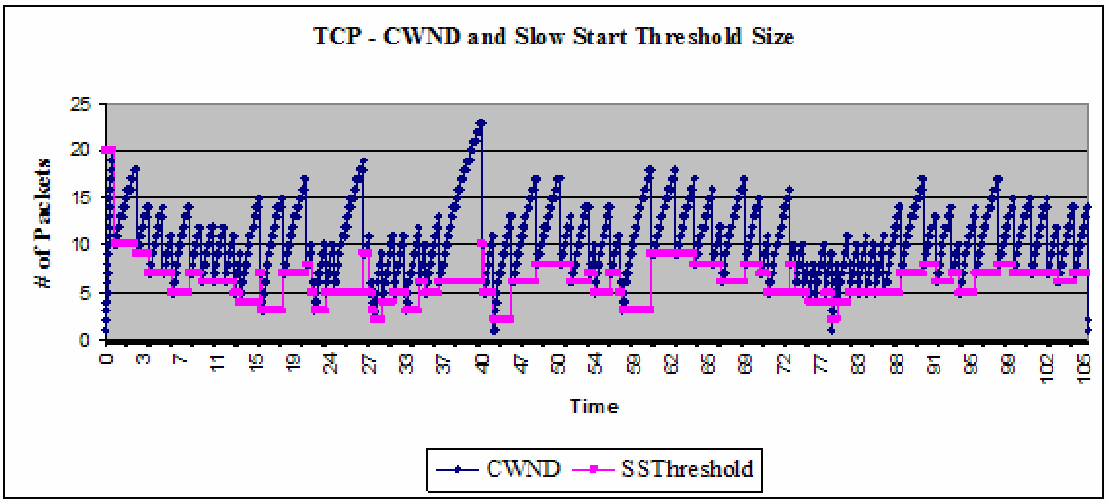


Figure 6-16 TCP *cwnd* and Slow Start Threshold(RTT ≅ 110 ms)

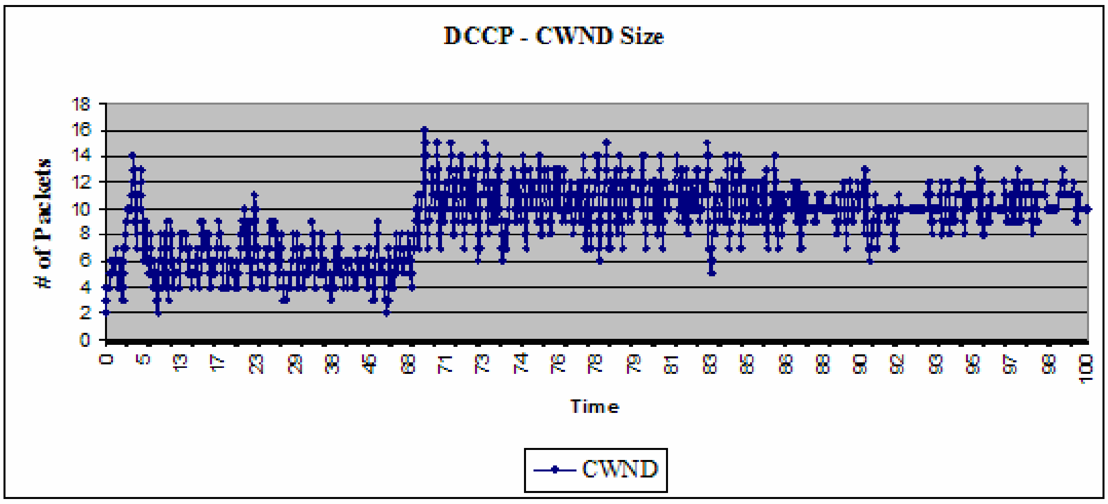


Figure 6-17 DCCP *cwnd* (RTT ≅ 60 ms)

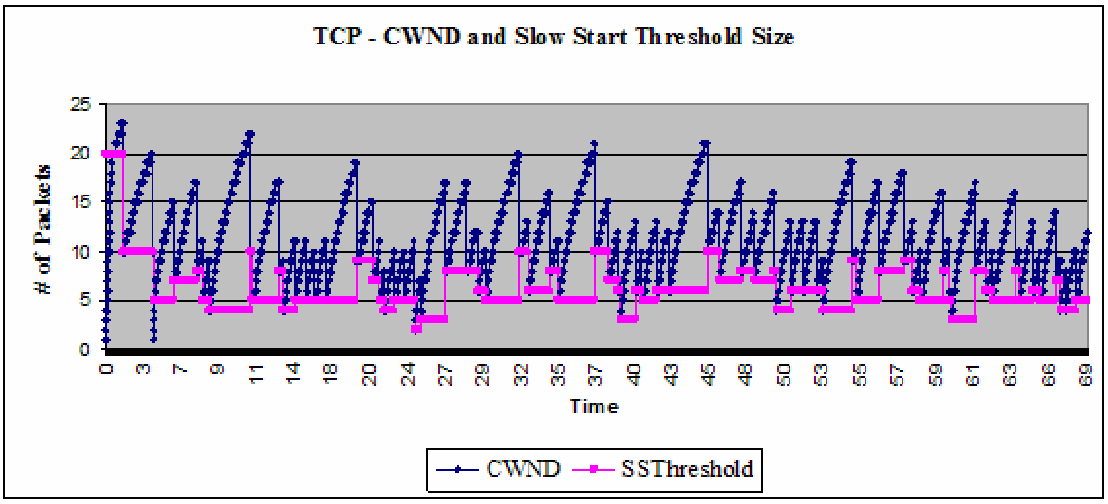


Figure 6-18 TCP *cwnd* and Slow Start Threshold(RTT ≅ 60 ms)

6.4.2 Both OUT Flows

When the two flows are both out flows, the DCCP flow tends to use the bandwidth fully and starves the TCP flow as shown in Figure 6-19. This is because of the removal of in-order delivery and exponential backoff in DCCP's congestion control.

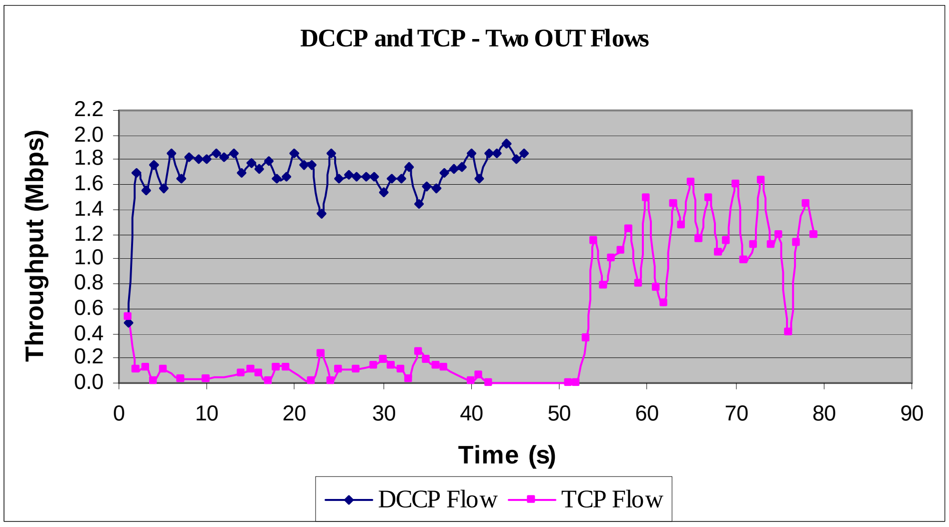


Figure 6-19 TCP/DCCP: Two OUT Flows (RTT ≅ 110 ms)

6.4.3 IN/OUT Flows

When both IN and OUT flows exist, the unfairness between IN and OUT flows still exists. In the DCCP IN flow and TCP OUT flow combination, the DCCP IN flow uses more bandwidth than the TCP OUT flow. Similar to the TCP IN/OUT flows scenario, the TCP IN flow looses more data packets and the DCCP OUT flow looses more ACKs, so the IN flow is affected more. Figure 6-20 shows the throughput difference between the two flows.

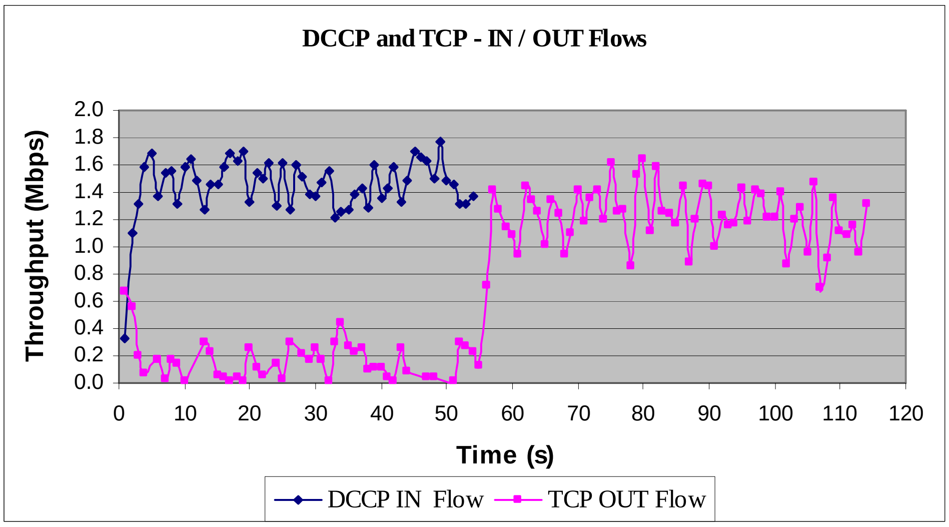


Figure 6-20 TCP/DCCP: DCCP IN Flow/TCP OUT Flow (RTT ≅ 110 ms)

When a TCP IN flow coexists with a DCCP OUT flow, the unfairness is improved compared with TCP IN/OUT flows combination. The improvement in fairness is mainly because DCCP does not provide reliability and retransmission, so the lost packets are simply dropped during the transmission. Figure 6-21 shows the throughput difference between the two flows.

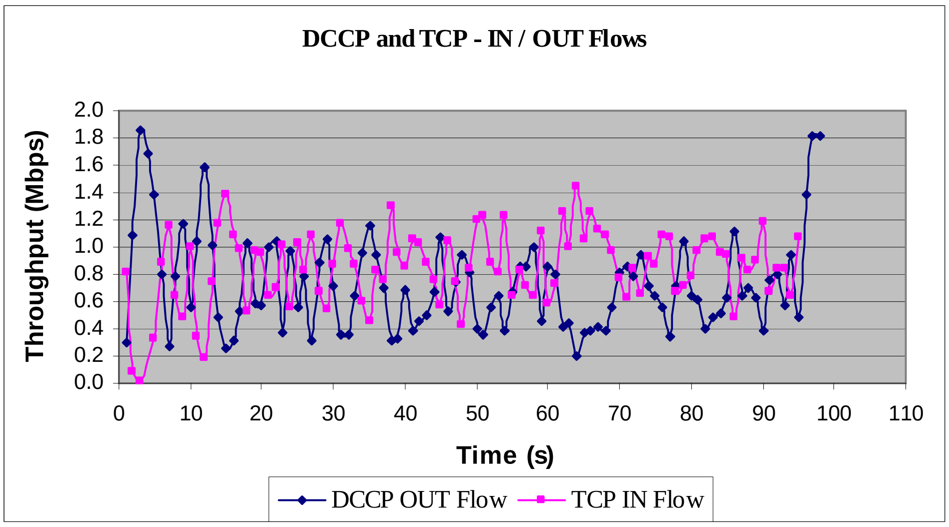


Figure 6-21 TCP/DCCP: DCCP OUT Flow/TCP IN Flow (RTT ≅ 110 ms)

The throughput and fairness of DCCP and TCP flows is given in Table 6-5.

Overall Throughput (Mbps)	**Both IN (110ms)**	**Both OUT (110ms)**	**IN/OUT (DCCP IN)**	**IN/OUT (TCP IN)**
DCCP Flow	0.764	1.675	1.443	0.728
TCP Flow	0.755	0.423	0.702	0.843

Throughput (Mbps)	**Both IN (110ms)**	**Both OUT (110ms)**	**IN/OUT (DCCP IN)**	**IN/OUT (TCP IN)**
DCCP Flow	0.764	1.675	1.443	0.699
TCP Flow	0.747	0.071	0.118	0.843
Sum	1.511	1.747	1.561	1.541
Jain's Fairness Index	1.000	0.542	0.581	0.991

Table 6-5 DCCP/TCP Throughputs and Fairness

6.5 Combined UDP and DCCP Flows

The unfairness is more severe when there are OUT flows in the simulation and the most unfair bandwidth sharing happens when the TCP and DCCP flows are both OUT flows. In the simulations for mixed flows, when the TCP flow is the OUT flow, the unfairness is also rather severe. The reason is that a DCCP flow is more competitive to obtain more bandwidth compared with a TCP flow. This is also shown in the simulations when a TCP or DCCP flow coexists with a UDP flow (1.6MBps CBR flow), as shown in Figures 6-22 and 6-23, in which the two flows are both OUT.

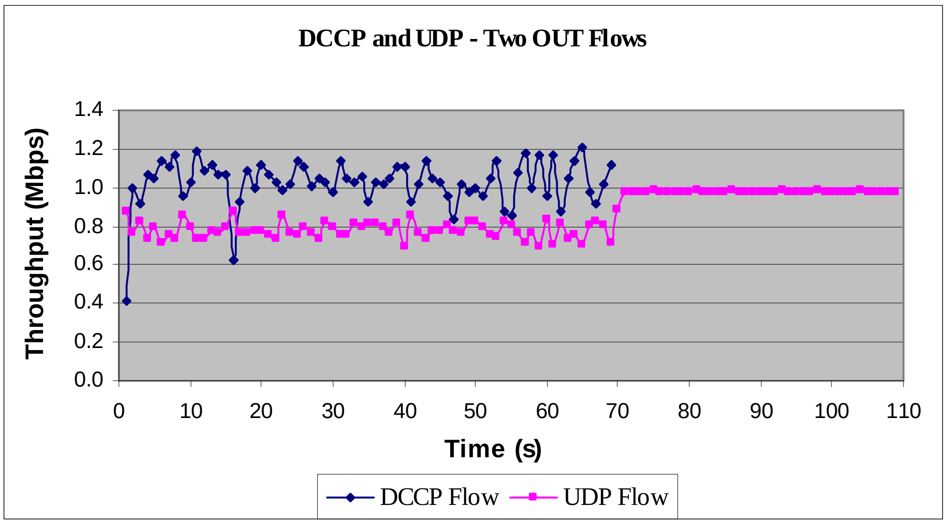


Figure 6-22 UDP/DCCP: Two OUT Flows (RTT ≅ 110 ms)

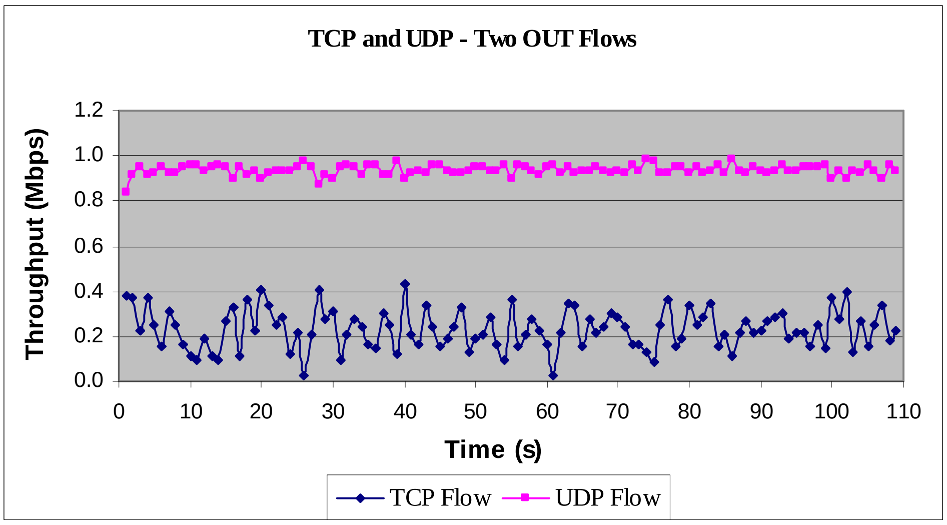


Figure 6-23 UDP/TCP: Two OUT Flows (RTT ≅ 110 ms)

6.6 Conclusion

It is demonstrated in the simulations that DCCP flows have good inter-flow fairness due to the modified congestion control algorithm, which uses a rate based window control algorithm based on the feedback from the acknowledgments, and keeps a constant backoff timer according to the network condition. When DCCP flows coexist with TCP flows, DCCP flows starve the TCP flows only when there is an OUT TCP flow, and DCCP flows have better throughput when co-existing with UDP flows. The results show that the proposed congestion control algorithm can fulfill the requirements specified in Chapter 4.5: flows using the proposed congestion control algorithm share the bandwidth almost fairly regardless of where the senders are. When co-existing with TCP flows, the bandwidth sharing shows similar fairness issues as pure TCP flows, and the unfairness is more severe in these cases, this means that the TCP-friendliness of the proposed congestion control algorithm should be further studied and improved if possible.

These simulation results indicate that our proposed congestion control mechanism used in the Wireless Ad Hoc Access Network can achieve better bandwidth utilization and improved efficiency when used for applications, such as streaming audio or video transmission, compared with using UDP, which has no congestion control and can cause severe congestion. DCCP with the designed congestion control algorithm can potentially replace TCP and UDP for transmissions of real time audio and video traffics. With further study and improvement on providing in-order delivery of packets, our proposal may also have the potential to replace TCP in Wireless Ad Hoc Access Networks.

Chapter 7 discusses our extension to the basic DCCP protocol to provide for reliable, in-order delivery.

Chapter 7

Implementation and Simulations of Reliability on DCCP

Transmission Control Protocol (TCP) provides reliable, end-to-end, in-order delivery of a data stream. TCP's reliable transmission has the following properties:

- TCP provides buffered transfer and allows the application programs to send data in any size. At the sender side, data from application programs are buffered and segmented; at the receiver side, TCP segments are buffered and re-integrated before delivering them to the application programs. Data segments are numbered sequentially according to the number of bytes.
- TCP receivers are able to detect lost and corrupted packets and request retransmissions of those packets by acknowledgements.
- TCP receivers are able to detect and delete duplicated packets.
- Each TCP connection is a bit stream. The receiver delivers exactly the same data to the application programs as what the sender sends.

Although Datagram Congestion Control Protocol (DCCP) is designed for applications which do not need reliability, it has features which can be used to implement reliable transmission based on DCCP:

- The DCCP header has a sequence number for each DCCP Request or DCCP response packet. Senders and receivers use this information to detect whether packet losses have happened.

- The DCCP header includes a checksum field, which uses the same algorithm as TCP's checksum algorithm. Packets with invalid checksum are identified as corrupted.
- DCCP's ACK vector option provides packet loss and corruption information to DCCP senders.

Both DCCP and TCP are end-to-end sliding window protocols. Data packets are transmitted in both directions: packets are sent from the senders to the receivers and acknowledgements are sent from the receivers to the senders. Senders are allowed to send a window of packets before receiving the acknowledgment. This window starts at a constant size and is later controlled by the congestion control algorithms implemented in the protocols. Acknowledgments are valid when sequence numbers of acknowledged packets are within the range of the current window. To implement reliable transmission based on DCCP and provide a comparable level of reliability as TCP does, the following functions should be added to DCCP:

- Buffering of received packets at the receivers
- Retransmission of lost or corrupted packets by the senders
- Detection and deletion of duplicated packets at the receivers
- In-order delivery of received packets to the application program at the receivers.

7.1 Implementation of Reliable Transmission

To provide end-to-end, reliable transmission, buffers are implemented at both the sender's side and the receiver's side for a DCCP flow. The sender buffer keeps track of the packets which are sent by a DCCP sender, and the receiver buffer records information of received packets at the DCCP receiver. A buffer is implemented as a linked list using

C++ in NS2. Information in the list is used for retransmission at the sender and in-order packet delivery at the receiver. Information in the receiver's list is also used to build the acknowledgment and ACK vectors. The section below illustrates how the lists are used to provide reliable transmission.

7.1.1 Initial State and Normal Transmission

The initial state of the buffers is as shown in Figure 7-1, each of the lists has a pointer pointing to the list head.

List Head (Sender)
List Head (Receiver)

Figure 7-1: Initial State of the linked list at the Sender and the Receiver

When the sender sends a packet, a list member is added to the sender list, with the status of the acknowledgement as Not Acknowledged. When a receiver receives a packet, a list number is added to the receiver's list, with the status as received.

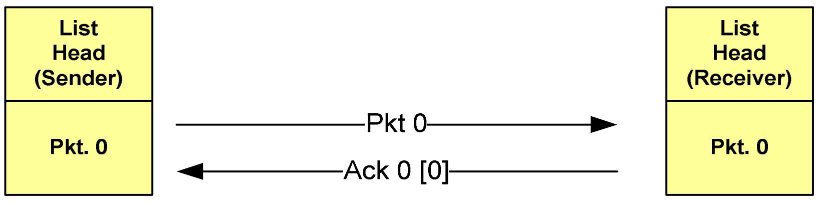


Figure 7-2: Packet No. 0 is added to linked lists at the Sender and the Receiver

After the receiver sends an ACK, the list is checked and all list members received in sequence up to this moment are handed to the application and deleted from the list. Similarly, after the sender received the acknowledgement, the ACK vector is parsed, all packets being acknowledged as received are marked in the list, then the list is checked and all in sequence packets marked as received are deleted from the list. After this, the congestion window is checked. When the number of outstanding packets in the network is within the congestion window, new packets will be transmitted. As an example, the packet with sequence number 1 is sent and the list at the sender's side adds a new member, packet number 1. After the receiver successfully received this packet, the receiver's list adds packet number 1 as well. The updated lists at sender and receiver are show in Figure 7-3.

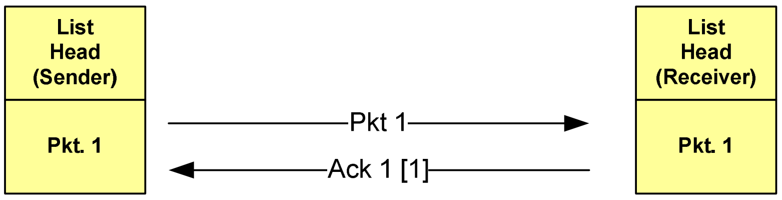


Figure 7-3: Packet No. 1 is added to linked lists at the Sender and the Receiver

7.1.2 Loss Scenario: Data Packet is lost during the Transmission

Figure 7-4 shows a scenario where packet number 3 is lost.

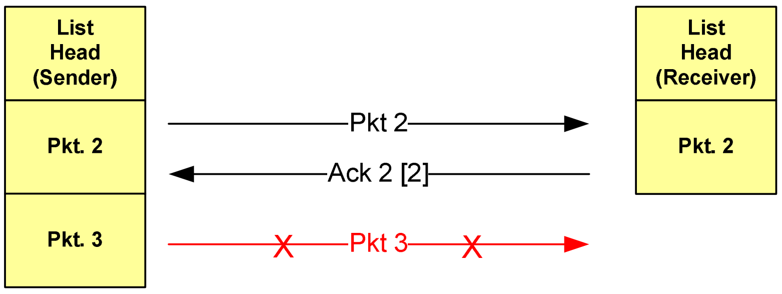


Figure 7-4: Lost Packet: linked lists at the Sender and the Receiver

When the congestion window size is 2, after receiving the acknowledgement of packet 2, the following in-sequence packet (number 4) is sent by the sender. When the receiver receives packet number 4, it checks the last received packet number (recorded by the receiver), and finds that there is a packet which was lost before the currently received packet. The receiver creates a list member for the lost packet with the status Not Received. The receiver also creates a list member for the received packet with status Received. The acknowledgement is sent by the receiver with an ACK vector indicating packet number 3 is lost and packet number 4 is received (Figure 7-5).

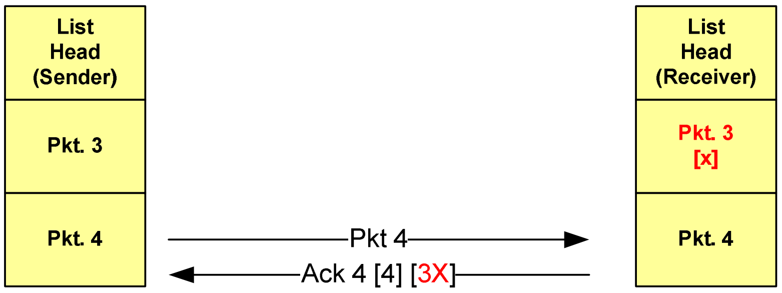


Figure 7-5: ACK Vector for Lost Packet: linked lists at the Sender and the Receiver

When the sender receives the acknowledgement, it parses the ACK vector and updates the packet number 3 as Not Received and packet number 4 as Received. When the sender is ready to send out packets, it first checks if there are packets marked as Not Received in the buffer. If so, the sender will re-transmit those packets first if they have not previously been retransmitted. If a lost packet has been retransmitted, the sender will check the retransmission timer associated with the retransmitted packet, if the timer expires, the sender will send the lost packet immediately. The retransmission timer value is set as the sum of average RTT and 8 * average RTT variance, which is calculated by:

RTT_variance = ¾ *old_RTT_variance + ¼ * new_RTT_variance

where new_RTT_variance is the delta between new RTT and average RTT. The initial value of the retransmission timer is set to 6 seconds.

Figure 7-6 depicts the scenario that the sender re-sends packet number 3 and sends packet number 5 after the retransmission of packet number 3. Acknowledgements are sent to the sender after the receiver receives the packets.

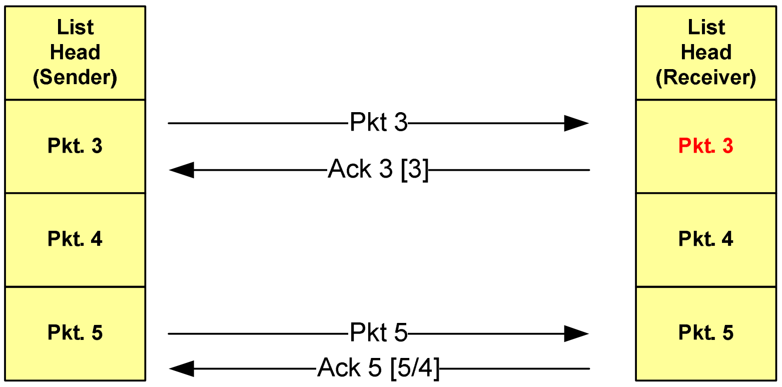


Figure 7-6: Retransmission of Lost Packet: linked lists at the Sender and the Receiver

7.1.3 Loss Scenario: ACK is lost during the Transmission

Acknowledgements can also be lost during the transmission. Figure 7-7 shows that packets number 6 and 7 are both received by the receiver, but ACK loss occurred.

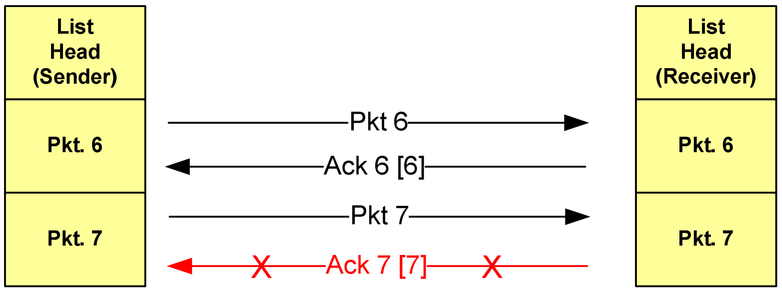


Figure 7-7: Lost ACK: linked lists at the Sender and the Receiver

When the ACK of packet number 7 is lost before reaching the sender, the sender assumes there is one outstanding packet (number 7) in the network. If the congestion window is 1, the sender will retransmit this packet after the timer expires. If the congestion window is greater than 1, the sender will send out new packets as show in Figure 7-8.

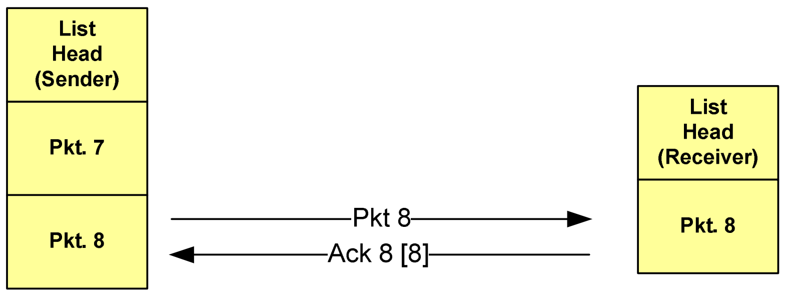


Figure 7-8 Transmission of new packet: linked lists at the Sender and the Receiver

After the sender receives the acknowledgement for packet number 8, which has the ACK vector containing information only for packet number 8, it will mark packet number 7 as received. All in sequence packets marked as received are handed to the application in the receiver and deleted from the lists at both the sender and the receiver.

7.2 Simulations and Analysis

Simulations are conducted to verify the performance of the Reliable DCCP with rate based congestion control. The test setup is the same as one used in simulations for TCP and DCCP in Chapter 6.

First, a single flow traversing a chain of nodes (4 hops) was tested. The throughputs including retransmitted packets are provided in Table 7-1, which also includes previous test results for a single TCP flow and a single DCCP flow. The DCCP flow with reliability implementation has lower throughput than the unreliable DCCP flow because the sender of the reliable DCCP implementation needs to detect lost packets or acknowledgements and retransmits lost packets or packets not acknowledged positively.

	Total Pkts	Received Pkts	Throughput	Goodput
TCP	6667	6912	0.447Mbps	0.444Mbps
DCCP	6667	5871	0.829Mbps	0.829Mbps
Reliable DCCP	6667	8801	0.765Mbps	0.629Mbps

Table 7-1 Single Flow Performances

The diagram below shows the changes of throughput over the transmission time of the single reliable DCCP flow.

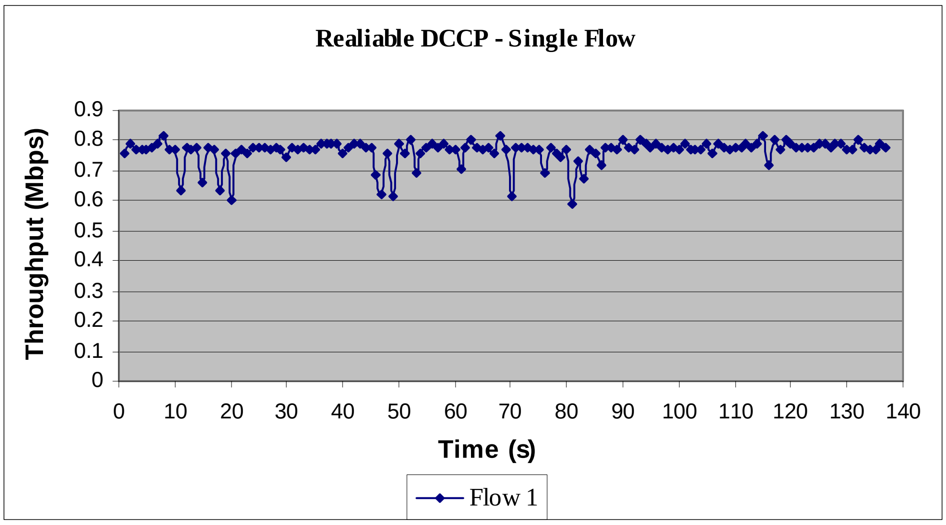


Figure 7-9 Reliable DCCP Performance: Single Reliable DCCP Flow

7.2.1 Reliable DCCP Flows

Similar to previous simulations, an IN flow is a connection from the wired node to the wireless nodes, and an OUT flow is a connection from the wireless nodes to the wired node. Figures 7-10 to 7-12 are the throughput diagrams for tests of two IN flows, two OUT flows and one IN and one OUT flow. The simulations for reliable DCCP flows show similar bandwidth sharing patterns as the simulations for DCCP flows, i.e. the two flows can share the bandwidth almost fairly in all test cases. The average throughputs for Reliable DCCP are lower than the throughputs for DCCP flows, as already shown above.

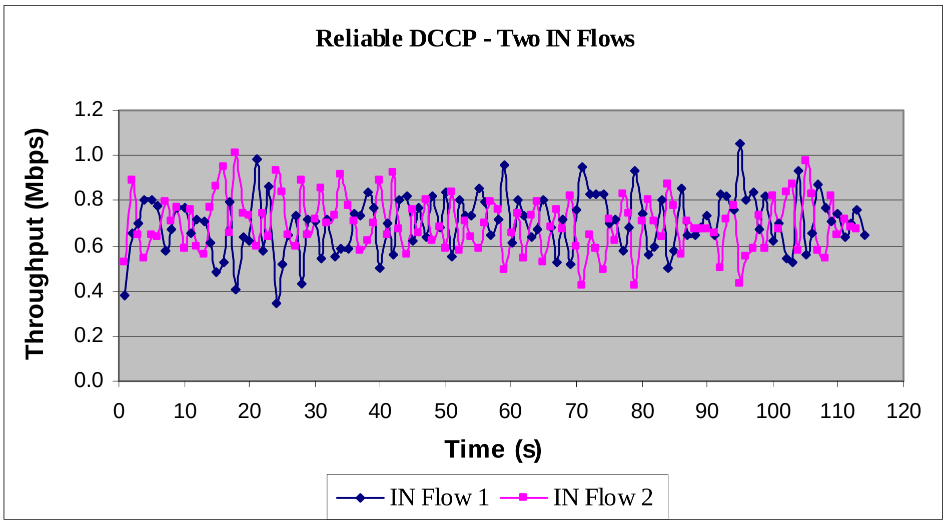


Figure 7-10 Reliable DCCP Performance: Two IN Flows (RTT ≅ 110 ms)

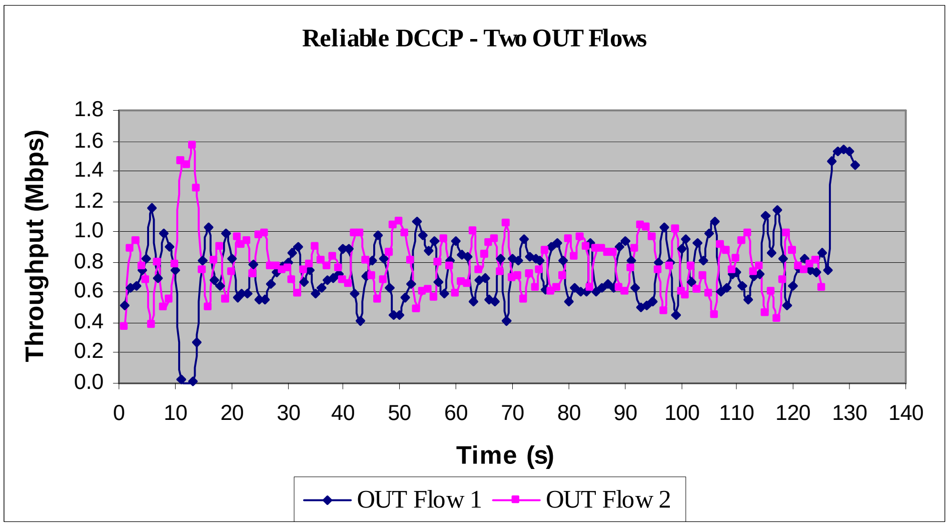


Figure 7-11 Reliable DCCP Performance: Two OUT Flows (RTT ≅ 110 ms)

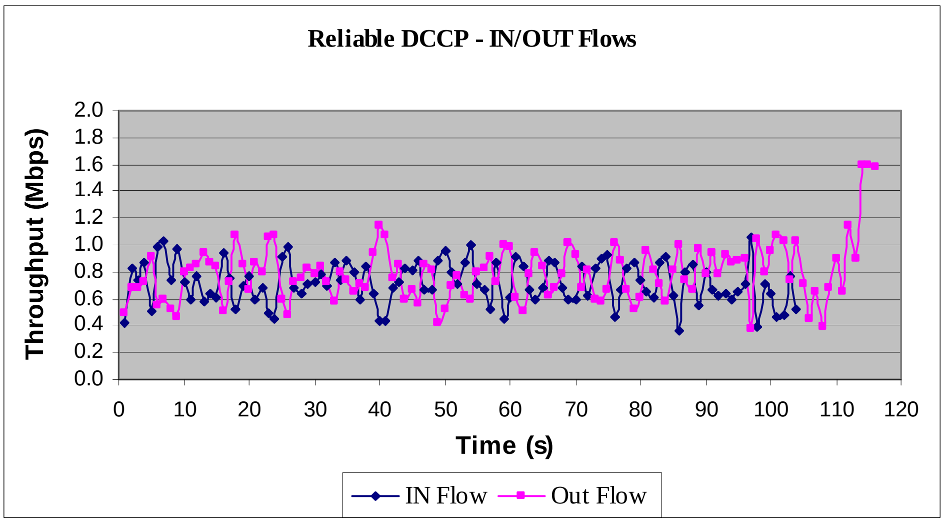


Figure 7-12 Reliable DCCP Performance: IN/OUT Flows (RTT ≅ 110 ms)

Table 7-2 lists the throughputs in the simulations for both IN flows, both OUT flow and IN/OUT flows, and the fairness index in those tests calculated by Jain's Fairness Index.

Overall Throughput (Mbps)	Both IN (110ms)	Both OUT (110ms)	IN/OUT (110ms)	IN/OUT (60ms)
Flow 1	0.71	0.75	0.72	0.84
Flow 2	0.71	0.78	0.81	0.83

Throughput (Mbps)	Both IN (110ms)	Both OUT (110ms)	IN/OUT (110ms)	IN/OUT (60ms)
Flow 1	0.71	0.75	0.72	0.84
Flow 2	0.71	0.79	0.77	0.69
Sum	1.42	1.53	1.49	1.53
Jain's Fairness Index	1.000	0.999	0.999	0.990

Table 7-2 Reliable DCCP Performance: Throughputs and Fairness

The comparison of Table 7-2 and Table 6-3 TCP performance: throughputs and fairness shows that Reliable DCCP flows achieve better throughputs and fairness among flows. The aggregate throughputs given by row Sum for the tests of reliable DCCP flows are higher than the aggregate throughputs in TCP's tests. Jain's fairness indexes in Table 7-2 are all close to 1, which indicates a fair sharing of network bandwidth by the two flows.

7.2.2 Combined TCP and Reliable DCCP Flows

In this section, the simulation results of combined TCP and reliable DCCP flows are provided. Figures 7-13 to 7-16 are the throughput diagrams for tests of two IN flows, two OUT flows and one IN and one OUT flow of combined TCP and reliable DCCP flows. The simulation results show that, when both flows are IN flows, they share the media almost fairly: one IN flow finished the transmission shortly after the other flow; when both are OUT flows, the DCCP flow uses more bandwidth than the TCP flow: the DCCP flow always finished the transmission first and the TCP flow was almost stopped during the transmission of the DCCP flow; and when IN/OUT flows co-exist, the IN flow always uses more bandwidth than the OUT flow. The reasons are the same as we discussed in Chapter 6.

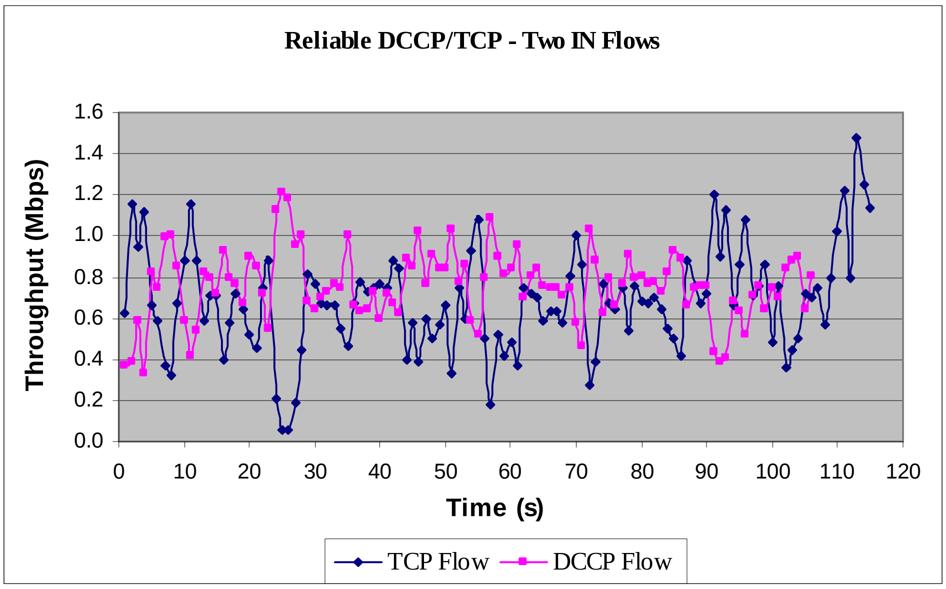


Figure 7-13 TCP/Reliable DCCP: Two IN Flows (RTT ≅ 110 ms)

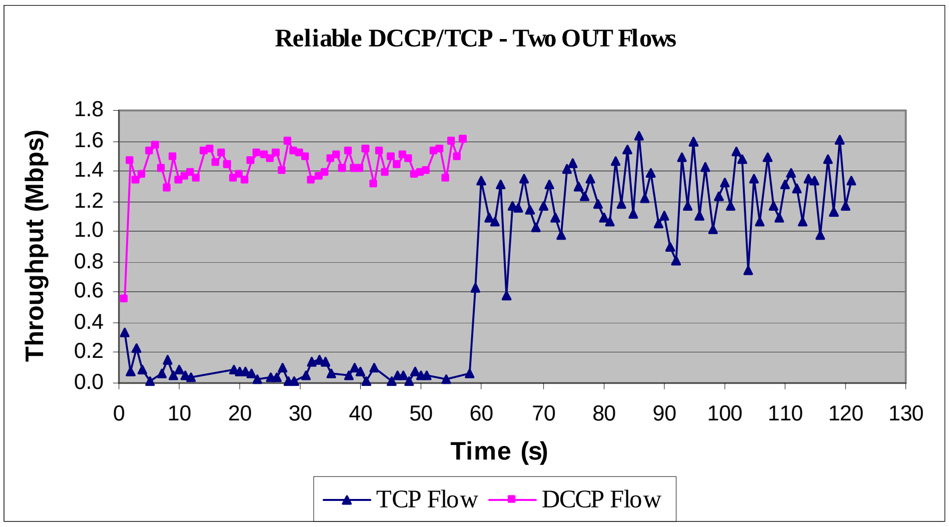


Figure 7-14 TCP/Reliable DCCP: Two OUT Flows (RTT ≅ 110 ms)

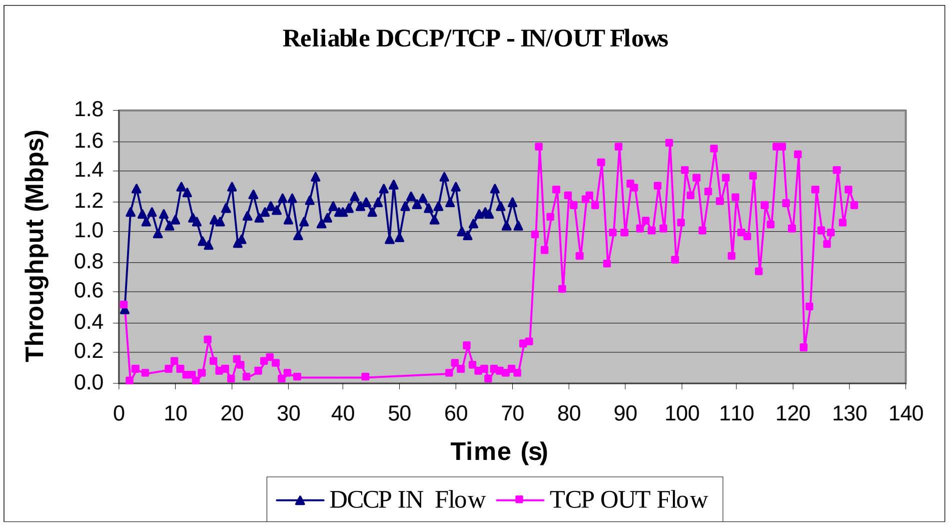


Figure 7-15 TCP/Reliable DCCP: DCCP IN Flow/TCP OUT Flow (RTT ≅ 110 ms)

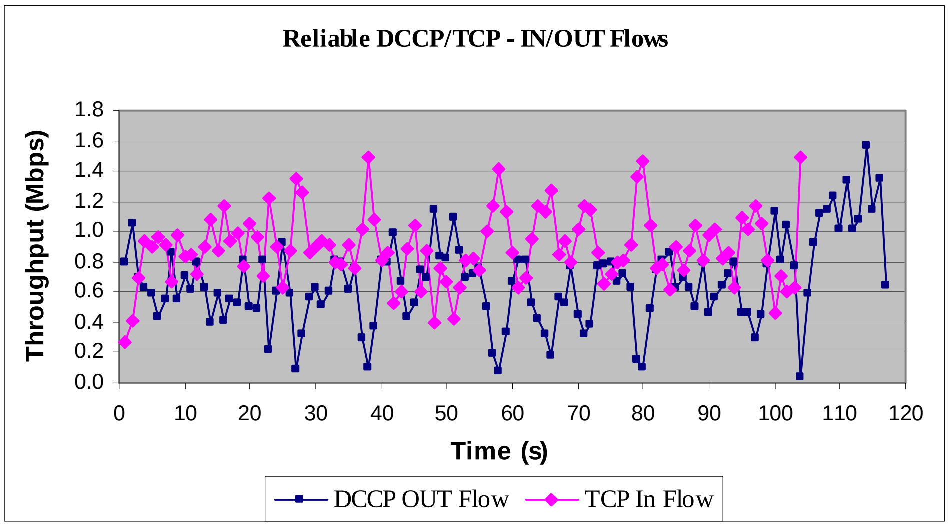


Figure 7-16 TCP/Reliable DCCP: DCCP OUT Flow/TCP IN Flow (RTT ≅ 110 ms)

The throughputs of combined Reliable DCCP and TCP flows are given in Table 7-3. Comparing the simulation results of combined DCCP/TCP flows in Chapter 6 (Table 6-5) and combined Reliable DCCP/TCP flows, it is found that the aggregate throughputs in the simulations of reliable DCCP are less than the aggregate throughputs in the simulations of unreliable DCCP. Similar to unreliable DCCP, reliable DCCP also has fairness issues when the flows include an OUT TCP flow.

Overall Throughput (Mbps)	Both IN (110ms)	Both OUT (110ms)	IN/OUT (DCCP IN)	IN/OUT (TCP IN)
DCCP Flow	0.76	1.44	1.12	0.67
TCP Flow	0.68	0.66	0.57	0.89

Throughput (Mbps)	Both IN (110ms)	Both OUT (110ms)	IN/OUT (DCCP IN)	IN/OUT (TCP IN)
DCCP Flow	0.76	1.44	1.12	0.61
TCP Flow	0.69	0.05	0.05	0.89
Sum	1.45	1.49	1.18	1.50
Jain's Fairness Index	0.998	0.535	0.545	0.966

Table 7-3 Reliable DCCP/TCP Throughputs and Fairness

7.3 Conclusions

Comparing the simulation results of TCP flows and unreliable DCCP flows in Chapter 6 and reliable DCCP flows in this chapter, the simulations show the following characteristics of reliable DCCP implementation:

When both flows were either both TCP, both DCCP, or both reliable DCCP flows:

- Reliable DCCP flows have higher throughputs than TCP flows.
- Reliable DCCP flows have lower throughputs than DCCP flows.
- Reliable DCCP as well as DCCP provides improved inter-flow fairness, independent of the flow directions.

When tested jointly with TCP flows,

- Reliable DCCP and TCP flows have similar overall throughputs as only TCP flows
- Reliable DCCP and TCP flows have similar fairness issues as only TCP flows

The above conclusions show that the proposed congestion control algorithm fulfills the following requirements specified in Chapter 4.5:

- Flows using the proposed congestion control algorithm share the bandwidth almost fairly regardless of where the senders are
- The bandwidth utilization is improved using the proposed congestion control algorithm.

Similar to the unreliable DCCP flows, when co-existing with TCP flows, the bandwidth sharing of reliable DCCP flows shows similar fairness issues as pure TCP flows, and the unfairness is more severe in these cases, this means that the TCP-friendliness of the proposed congestion control algorithm should be further studied and improved if possible.

Chapter 8

Conclusions and Future Work

In this thesis, a congestion control algorithm using DCCP for a Wireless Ad Hoc Access Network is developed and tested using the NS2 simulator. In the thesis, the reasons of throughput degradation and unfairness among flows when using TCP in a Wireless Ad Hoc Access Network were studied and simulations were conducted to verify the performance of TCP. The thesis introduces several recent transport layer protocols, and current bandwidth estimation algorithms, which are the basis for designing the congestion control algorithm in this thesis. The proposed congestion control algorithm is based on DCCP, because DCCP allows the selection of a congestion control algorithm according to different applications and has many features to be used in the congestion control algorithm, such as ACK Vector, which carries specific packet information in the acknowledgement, options for RTT calculation, which includes Time Stamp and Time Stamp Echo options, and options to identify packet losses not because of network congestions. The proposed congestion control algorithm is based on state transitions due to different types of failure scenarios: network congestion, packet loss, or transmission errors, which are differentiated with the help of feedback from the network where appropriate. The algorithm controls the transmission rate by maintaining two RTT variables: base RTT and average RTT. Because of the limitations of NS2, the implementation of the congestion control algorithm on NS2 only detects packet losses due to collisions on the wireless media. All nodes in the simulations are static to eliminate route changes.

The thesis presents an implementation for DCCP with congestion control and reliability to provide lost packet detection and retransmission, which is not provided by the basic DCCP framework. Comparing the proposed congestion control algorithm in unreliable DCCP implementation and reliable DCCP implementation, the thesis has the following conclusions:

- Reliable DCCP provides better inter-flow fairness among Reliable DCCP flows.
- Reliable DCCP flows can achieve better throughputs when they are only DCCP flows.
- When mixed with TCPs flows, reliable DCCP shows similar fairness issues as scenarios with TCP flows only and the unfairness is more severe compared with pure TCP flows when unfairness exist.

Flows with the new congestion control algorithm tested in the simulations had Jain's fairness index greater than 0.95 in all combinations, where TCP flows may have a fairness index of less than 0.6. The unreliable DCCP implementation achieved better bandwidth utilization and similar fairness as the proposed reliable DCCP implementation. Those improvements in fairness and bandwidth utilization showed that DCCP with the designed congestion control algorithm can potentially replace TCP and UDP for transmitting audio and video applications, and with the implementation of reliability similar to what TCP provides, the reliable DCCP with the designed congestion control algorithm have the potential to replace TCP at Wireless Access Ad Hoc Networks.

The thesis can be further extended in the following areas to verify and improve the design:

- Perform more simulation runs to verify the test results

- Add in node mobility to the simulations and study the impact of additional loss scenarios caused by broken links during the transmission on throughput and fairness achieved by the proposed algorithm.
- Perform simulations with different network setups, such as infrastructureless Ad Hoc Network with more hops between the sender and receivers.
- Further study and improve throughput and fairness when mixed Reliable DCCP and TCP flow are co-existing.
- Further adjust the rate control formula and retransmission timer to optimize the packet sending rate. Study its impact on the inter protocol fairness between DCCP flows and TCP flows.
- Adding new features to the implementation in the simulation: such as support of ECN, to provide additional information for the sender to identify network condition and to adjust the sending rate accordingly.
- Implement the design on a test bed to verify the simulation results.

References

[1] M. Allman, V. Paxson and W.Stevens, “TCP Congestion Control”, Request for Comments 2581, Internet Engineering Task Force, April 1999.

[2] V. Anantharaman and R. Sivakumar, “TCP Performance over Mobile Ad Hoc Networks – a Quantitative Study”, Wireless Communication and Wireless Networks (WCMC), Special Issue on Wireless Evaluation of Wireless Networks, pages 203 – 222, 2003.

[3] A. Bakre and B. R. Badrinath, “I-TCP: Indirect TCP for Mobile Hosts International Conference on Distributed Computing Systems”, In Proceedings of the 15th International Conference on Distributed Computing Systems, pages 136-143, Vancouver, Canada, June 1995,

[4] L. S. Brakmo, S. O’Malley, and L.L. Peterson, “TCP Vegas: new techniques for congestion detection and avoidance”, In Proceedings of the Conference on Communication Architectures, Protocols and Applications, SIGCOMM ’94, volume 24, issue 4, pages 24 – 35, London, United Kingdom, Oct. 1994.

[5] A. Capone, L. Fratta, F. Martignon, “Bandwidth Estimation Schemes for TCP over Wireless Networks”, IEEE Transactions on Mobile Computing, volume 3, issue 2, pages 129-143, April-June 2004.

[6] K. Chandran, S. Raghunathan, S. Venkatesan, R. Prakash, “A Feedback Based Scheme For Improving TCP Performance In Ad-Hoc Wireless Networks”, In Proceedings of International Conference on Distributed Computing Systems‘98. pages 472-479, Amsterdam, Netherland, 1998.

[7] H. Choe and S. H. Low, “Stabilized Vegas”, In Proceedings of the 22nd Annual Joint Conference of IEEE Computer and Communications Societies, volume: 3, pages 2290 – 2300, San Francisco, USA, April 2003.

[8] S. Corson and J. Macker, “Mobile Ad Hoc networking [MANET]: routing protocol performance issues and evaluation considerations”, Request for Comments 2501, Internet Engineering Task Force, January 1999.

[9] T. D. Dyer, R. V. Boppana, “A Comparison of TCP Performance over Three Routing Protocols for Mobile Ad Hoc Networks”, In Proceedings of the International Symposium on Mobile Ad Hoc Networking & Computing, pages 56 – 66, Long Beach, USA, 2001.

[10] S. ElRakabawy, A. Klemm and C. Lindemann. “TCP with Adaptive Pacing for Multihop Wireless Networks”, In Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking & Computing, pages 288 -299, Urbana-Champaign, USA, May 2005.

[11] G. Holland and N. Vaidya, “Analysis of TCP performance over Mobile Ad Hoc Networks”, In Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking, pages 219 – 230, Seattle, USA, 1999.

[12] V. Jacobson, “Using pathchar to estimate Internet link characteristics”, In Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, pages 241 – 250, Cambridge, 1999.

[13] M. Jain, C. Dovrolis, “End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput”, IEEE Transaction on Networking, volume 11, issue 4, pages 537 – 547, August 2003.

[14] R. Jain, D. Chiu, and W. Hawe, “A quantitative measure of fairness and discrimination for resource allocation in shared computer systems”, Technical Report, DEC Research Report TR-301, Digital Equipment Corporation, Hudson MA, USA, Sept. 1984.

[15] C. Jin, D. Wei, S. H. Low, “FAST TCP: Motivation, Architecture, Algorithms, Performance”, In Proceedings of the 23rd Conference of the IEEE Communication Society, pages 81-94, Hong Kong, China, March 2004.

[16] C. Jin, D. Wei, S. H. Low, G. Buhrmaster, J. Bunn, D. H. Choe, R. L. A. Cottrell, J. C. Doyle, W. Feng, O. Martin, H. Newman, F. Paganini, S. Ravot, S. Singh, “FAST TCP: From theory to experiments”, IEEE Network, volume 19, issue 1, pages 4-11, January-February 2005.

[17] E. Kohler, M. Handley and S. Floyd, “Datagram Congestion Control Protocol (DCCP)”, Request for Comments 4340, Internet Engineering Task Force, March 2006.

[18] J. Li, C. Blake, D. S. J. De Couto, H. I. Lee, R. Morris, “Capacity of Ad Hoc Wireless Networks”, In Proceedings of 7th Annual ACM/IEEE International Conference on Mobile Computing and Networking, pages 61 – 69, Rome, Italy, 2001.

[19] J. Liu, S, Singh, “ATCP: TCP for Mobile Ad Hoc Networks”, IEEE Journal on Selected Areas in Communication, volume 19, issue 7, pages 1300 – 1315, July 2001.

[20] B. Melander, M. Bjorkman, and P. Gunningberg, “A New End-to-End Probing and Analysis Method for Estimating Bandwidth Bottlenecks”, In Proceedings of Global Telecommunications Conference, GLOBECOM ’00, IEEE, volume 1, issue 2000, pages 415 – 420, San Francisco, USA, November 2000.

[21] J. Mo, V. Anantharam, R. J. La and J. Walrand, "Analysis and Comparison of TCP Reno and Vegas", In Proceedings of the 18th Conference of the IEEE Communication Society, pages 1556 – 1563, New York, USA, March 1999.

[22] S. Pilisof, R. Ramjee, D. Raz, "Understanding TCP fairness Over Wireless LAN", INFOCOM 2003, In Proceedings of the 22nd Annual Joint Conference of IEEE Computer and Communications Societies, IEEE, volume 2, pages 863 – 872, San Francisco, USA, April 2003.

[23] R. S. Prasad, M. Murray, C. Dovrolis and K. Claffy, "Bandwidth estimation: metrics, measurement techniques and tools", IEEE Network, volume 17, issue 6, pages 27 – 35, Nov/Dec. 2003.

[24] K. Ramakrishnan, S. Floyd and D. Black, "The addition of explicit congestion notification (ECN) to IP", Request for Comments 3168, Internet Engineering Task Force, Sep. 2001.

[25] K. Sundaresan and V. Anantharaman, "ATP: A reliable Transport Protocol for Ad-Hoc Networks", IEEE Transactions on Mobile Computing, volume 4, issue 6, pages 588 – 603, Nov/Dec, 2005.

[26] F. Wang and Y. Zhang, "Improving TCP Performance over Mobile Ad-Hoc Networks with Out-of-Order Detection and Response" In Proceedings of 3rd ACM International Symposium on Mobile Ad Hoc Networking & Computing, pages 217 - 225, Lausanne, Switzerland, June 2002.

[27] R. Wang, M. Valla, M.Y. Sanadidi and M. Gerla, "Adaptive Bandwidth Share Estimation in TCP Westwood", In Proceedings of the Global Telecommunications

Conference, GLOBECOM'02, IEEE, volume 3, pages 2604 – 2608, Taipei, Taiwan, October 2002.

[28] R. Wang, M. Valla, M.Y. Sanadidi, B. Ng and M. Gerla, "Efficiency/Friendliness Tradeoffs in TCP Westwood", In Proceedings of the 7th annual International Conference on Mobile Computing and Networking, pages 287 – 297, Rome, Italy, July 2001.

[29] K. Xu, S. Bae, S. Lee and M. Gerla, "TCP Behavior across Multihop Wireless Networks and the Wired Internet", In Proceedings of the 5th International Workshop on Wireless Mobile Multimedia, WoWMoM'02, pages 207 – 218, Seattle, USA, September 2002.

[30] L. Yang, W. K. G. Seah, Q. Yin, "Improving Fairness among TCP Flows crossing Wireless Ad Hoc and Wired Networks", In Proceedings of the 4th ACM International Symposium of Mobile Ad Hoc Networking & Computing, pages 57 – 63, Annapolis, USA, 2003.

Appendix A

In Appendix A, the existing proposals to improve the throughput and fairness problems of TCP over Mobile Ad Hoc Network are introduced.

A.1 Existing TCP Proposals in Ad Hoc Networks

A.1.1 TCP with ELFN (Explicit Link Failure Notification)

[11] presents the analysis of the use of explicit link failure notification on the performance of TCP over Mobile Ad Hoc networks. The objective of ELFN is to provide the TCP sender with information about link and route failures so it can respond properly. There are several ways in which ELFN messages can be implemented. The first one is to use the “host unreachable” ICMP message as a notice to the TCP sender. If the routing protocols send route failure messages, then the notice can be piggy-backed on these messages. In [11], DSR’s route failure message is modified to carry a notice similar to the “host unreachable” ICMP message.

Upon receiving a route failure notice, the TCP sender enters a “stand-by” state and freezes all timers. A probe packet is sent periodically to probe the network to see if the route has been reestablished. If an ACK is received, the TCP sender leaves the “stand-by” state, restarts the data transmission and resumes timers. The study shows significant throughput increase with the use of ELFN. But the simulation is conducted with DSR and one TCP flow only.

When multiple flows exist, [2] shows that this approach cannot achieve throughput improvements, and it even degrades the performance as the mobility rate increases. It shows that, when the probing is conducted by several connections, the flooding of probe packets increases the congestion of the network. Also when a new route is calculated, the TCP sender restarts to send at the old rate. The congestion window is estimated inaccurately. When the congestion window is overestimated, network congestion is likely to happen.

A.1.2 TCP-Feedback

TCP-Feedback is a feedback scheme in which the TCP sender utilizes the network layer feedback (Route Failure Notification) from intermediate nodes to distinguish route failure and network congestion.

After receiving a Route Failure Notification (RFN), TCP enters into the “snooze state”. In this state, TCP stops sending packets and freezes all its variables such as timers and *cwnd* size. Upon receiving a Route Re-establishment Notification (RRN), via the routing protocol, TCP knows the route is reestablished and leaves the frozen state and resumes transmission using the same variable states before the “snooze state”. In addition, a route failure timer is used to prevent infinite wait for RRN messages. When a route failure timer expires, the TCP normal congestion control is invoked.

[6] shows TCP-Feedback perform significantly better than standard TCP when route reestablishment delay grows. This is mainly due to the reduction of the number of unnecessary packet retransmission/timer backoffs during the route failure interval. But the simulation scenario was simplified and the costs of arouting protocol carrying the RRN and RFN messages were not considered.

A.1.3 Fixed RTO (Fixed Retransmission Timeout)

The Fixed Retransmission Timeout scheme studied the TCP performance over three routing protocols (DSR, DSDV and AODV) and pointed out that the regular exponential backoff mechanism is unnecessary, because route disconnection should be treated as a transitory period. Fixed RTO disables the exponential backoff after two successive retransmissions due to expired RTO, assuming it is caused by route failures. TCP retransmits a data packet more frequently because the retransmit timer is fixed; this reduces the inactive period after a route is reestablished.

In [2], significant improvement of throughput was achieved by the use of Fixed RTO. The article also studied the TCP selective and delayed acknowledgments options, which could only achieve marginal gains. But they stated that this approach is limited to wireless only, which is not good for an environment of combined wireless and wired networks.

A.1.4 ATCP: TCP for Mobile Ad Hoc Networks

In ATCP, to maintain compatibility with the standard TCP/IP protocol suite, a thin layer called Ad Hoc TCP is inserted between TCP and IP. This scheme is different from the above three approaches where standard TCP is modified.

ATCP [19] utilizes the ICMP protocol and the ECN (Explicit Congestion Notification) scheme to detect network partition and congestion respectively. The intermediate layer ATCP keeps track of the packets to and from the transport layer. The feedback from intermediate nodes are used to put the TCP sender into either a persist state, congestion control state, or retransmit state. When a “Destination Unreachable” ICMP message is received, indicating route failure happened, the TCP sender enters a “persist state” and ends when the connection is reestablished. When three duplicate acknowledgements are received, indicating random errors, ATCP puts the TCP sender into “retransmit state” and quickly retransmits the lost packets from the TCP buffer. When an ECN message is received, which indicates real network congestion, ATCP puts the TCP sender into “congestion control state” and the TCP sender invokes the normal congestion control procedure.

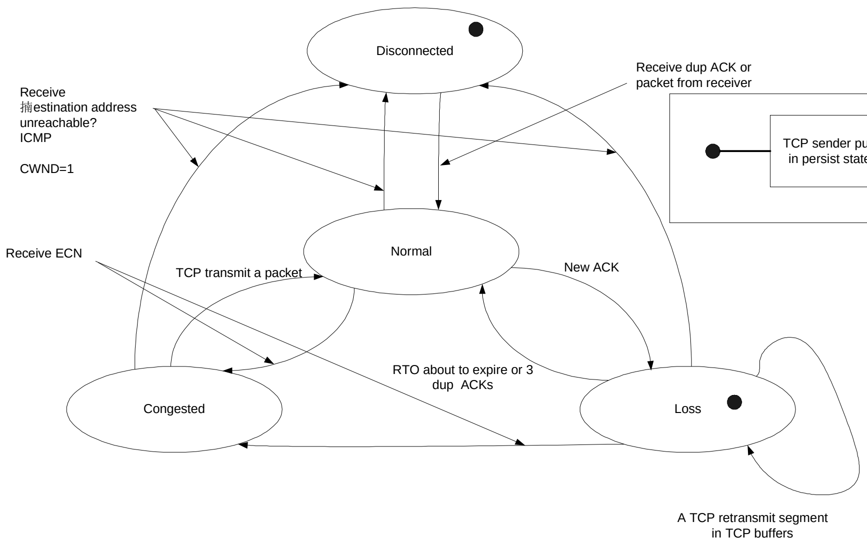

Disconnected
Receive dup ACK or packet from receiver
Receive 揃estination address unreachable? ICMP
CWND=1
TCP sender put in persist state
Normal
Receive ECN
TCP transmit a packet
New ACK
RTO about to expire or 3 dup ACKs
Congested
Loss
A TCP retransmit segment in TCP buffers

Figure A-1 State Transition Diagram for ATCP in the Sender

ATCP maintains end-to-end TCP semantics and is transparent to all nodes. Studies in [19] show great improvement of throughput under congestion, packet loss, and network partitions.

A.1.5 TCP-DOOR

Detection of Out-Of-Order and Response (DOOR) relies on the idea that out-of-order packets can happen frequently in Mobile Ad Hoc networks as a result of node mobility, and it can be an indication of link failure.

Out-Of-Order (OOO) delivery could happen to data packets as well as ACK. To precisely detect OOO delivery, TCP DOOR adds additional ordering information to ACKs (one byte) and data packets (two bytes) as options. Upon detecting OOO events, the TCP sender can respond with two mechanisms: temporarily disabling congestion control, and instant recovery during congestion avoidance. During the temporarily disabled congestion control, the TCP sender keeps its state variables constant for a while after the OOO event detection. In the instant recovery during congestion avoidance, the TCP sender checks if the congestion control mechanism is invoked in the recent past. If so, the connection state prior to the congestion control invocation is restored.

In [26], the simulations show that TCP-DOOR can improve the TCP throughput significantly, by 50 percent on average. But taking OOO events solely as the result of network route changes is too general, because some routing protocols can also induce OOO packets that are not related to route changes.

The above five schemes address problems related to mobility, which trigger network congestion control mechanism unnecessarily. The common idea is to stop, delay or change the exponential backoff strategy when non-congestion caused packet loss is detected.

A.2 Proposals to Improve Fairness

Several researchers have studied TCP fairness in Mobile Ad Hoc networks under the IEEE 802.11 protocol. Some of them focused on the MAC layer. Three recently published proposals focusing on TCP fairness issues are discussed below.

A.2.1 Available Gateway Buffer Size Advertisement in Wireless LAN

In [22], TCP unfairness among upstream and downstream flows is presented and investigated. In an infrastructure wireless LAN, the gateway is used to forward traffic, and the buffer size in the gateway plays a key role in obtaining fair sharing of the medium among upstream and downstream flows. [22] identifies four regions of unfairness that depend on the buffer availability and it shows via simulation that, when equal number of downstream and upstream flows exist, the average throughput ratio between the upstream and downstream can go up to 800. The reason is that upstream flows' ACKs clutter the gateway buffer and cause the buffer to overflow. Downstream flows experience timeouts and transmit only with a window of 0-2 packets because of the packet drops at the gateway buffer. Upstream flows normally can reach their maximum window size. Because of the cumulative nature of TCP ACKs, small losses of ACKs do not affect the window size.

The proposed solution is to advertise the available buffer size to the sender. The gateway keeps the number of current TCP flows in the system. If the buffer size at the gateway is

B and the number of flows is N, then the receiver window of all the TCP flows are set to the minimum of advertised receiver window or [B/N] by modifying the receiver window field of ACKs traversing the gateway.

Through simulation and test bed implementation, this proposal shows a very good fairness, with the throughput ratio of upstream and downstream flows being 1 in the simulation and 1.007 in the test bed. The study is based on the assumption that all the losses happen in the gateway due to buffer overflow and all RTTs are the same among flows in the wireless LAN.

A.2.2 Optimal TCP Congestion Window Size in Combined Wired and Mobile Ad Hoc Networks

In [29], the TCP fairness problem in a combined wireless and wired network is investigated. The study shows that IN flows get significant more bandwidth than OUT flows. This unfairness is the joint result of MAC layer's exposed nodes and hidden nodes problem and TCP's timeout and backoff schemes (see Section 2.2).

By the study performed on the test bed, it is found that when the maximum congestion window size is smaller than a certain value (8 in the test), the two flows share the bandwidth fairly and the aggregate throughput reaches the upper limit. The problem is that this window size could not be preconfigured. A similar study is conducted in a pure Ad Hoc network, and the optimal congestion window size is found to be 1-2 packets. With a long propagation delay in the wired part, such a small window size will affect the efficiency.

A.2.3 Timer-Controlled Data Packet Sending to Improve Fairness among TCP Flows crossing Mobile Ad Hoc and Wired Networks

To improve fairness over a combined wired and Ad Hoc network, a non-work-conserving scheduling algorithm working with IEEE 802.11 MAC is proposed in [30]. In the proposal, the normal FIFO work-conserving scheduling scheme is replaced, which treats routing packets (generated by routing protocols) as high priority packets over data packets (generated by applications), and puts the high priority packets in the queue before all data packets upon arrival. The head of the queue is send to the MAC after knowing that the MAC is ready to send another packet.

In [30], a timer is set after a data packet is sent to the MAC. Only after the timer expires can the queue send another data packet. The routing packets still have high priority and dequeue immediately after knowing that the MAC is ready. No timer is set after a routing packet is sent. The duration of the timer is based on the queue output rate and is the sum of three parts: transmission delay without contention; transmission delay based on recent queue output (choosing from four predefined values based on the queue output rate); and a random value uniformly distributed from zero to the value of the second part. The timer adds extra adaptive delay in the scheduling, so the more aggressively a node is sending packet, the more severely it is penalized, thereby nodes failing to grab the medium can compete with the fast sending nodes now.

By simulation, [30] shows that the severe unfairness among flows can be eliminated while the aggregate throughput experiences small degradation. Also, the maximum congestion window size is not a threat to the fairness by use of this scheme, so it does not need to pre-configure the maximum congestion window size to achieve optimal throughput.