STRATEGIES TO IMPROVE THE PERFORMANCE OF TCP IN MOBILE AD HOC NETWORKS

LI JIANFENG

(B.Eng, Beijing Information Technology Institute, China)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2005

Acknowledgement

I greatly appreciate the help of my supervisor, Dr. Yin Qinghe. I want to express many thanks to him for his instruction and valuable suggestions. When I met difficulties or frustrations, he always encouraged and supported me.

I will also express my appreciation to the coworkers in Institute for Infocomm Research (I2R). They give me many good tips to solve the problems met in the course of research work.

To work and live in the I2R building is great. Please permit me to present my appreciations to I2R and NUS for the excellent equipment and facilities offered to fulfill the research work.

Contents

List of Figures

List of Tables

Abbreviations

MANET: **Mobile Ad Hoc Networks**

TCP: **Transmission Control Protocol**

CWND: **Congestion Window**

RTT: **Round Trip Time**

RTO: **Retransmission TimeOut**

DUPACK: **Duplicated ACK**

VRIS: **Vegas Routing Interaction Strategy**

CWL: **Congestion Window Limits**

ASCWL: **Adaptively Setting Congestion Window Limit**

MAC: **Media Access Control**

RTS: **Ready to Send**

CTS: **Clear to send**

DSR: **Dynamic Source Routing**

AODV: **Ad-hoc On-Demand Distance Vector**

RREQ: **Route Request**

RREP: **Route Reply**

RERR: **Route Error**

RED: **Random Early Detection**

Summary

Applying TCP in MANETs is an active research area due to TCP's wide acceptance by the Internets and MANET's easy deployment. However, applying TCP directly in MANETs would face a lot of problems such as throughput degradation and unfairness.

The fundamental reason for the throughput degradation is that TCP is designed for wired networks and it assumes all the packet losses are due to congestion. Its aggressive congestion window increasing strategy is another important reason. Former research works are mostly focused on how to improve the TCP throughput. The TCP source distinguishes the packet loss reasons and performs different implementations according to corresponding reason. We take both increasing the throughput and decreasing the average packet delay into consideration and propose to apply another conservative TCP version, TCP-Vegas in MANETs.

Applying Vegas in MANETs would face the re-routing problem which will cause very low throughput in the mobile scenarios. Thus, we propose the Vegas Routing Interaction Strategy (VRIS) to solve the problem to effectively improve the Vegas throughput.

The unfairness problem is also widely studied. Few resolving strategies are proposed. These strategies are usually MAC layer or network layer based which are hard to implement. We propose the Adaptively Setting Congestion Window Limit (ASCWL) strategy, which is TCP-layer based and thus easy to implement, to solve the severe unfairness problem. We only consider the static scenarios because it is hard to define the fairness in mobile scenarios. As the strategies to solve the unfairness problem are few, we implement ASCWL with both TCP-Reno and TCP-Vegas.

With re-routing problem resolved and appropriate congestion window limit setting, Vegas performance in MANETs can achieve visible improvement, which makes it more suitable for MANETs environment.

Chapter 1

Introduction

1.1 Mobile Ad Hoc Network

Mobile Ad Hoc Network (MANET) [1] is a complex distributed system. It is composed of a group of mobile nodes without fixed infrastructure or central control entities. The mobile nodes can self-organize freely and dynamically. Thus, multi-hop communication is the basis of MANET. The nodes play double roles: the communication endpoints and the routers between the communication nodes multi-hop apart.

MANET can be easily deployed and thus can be used in a lot of scenarios such as military applications and disaster rescues. Thus it attracts many researchers. And we can summarize the characteristics of MANET as follows: multi-hop wireless transmission, frequent topology changes, and limited bandwidth competed among neighboring nodes.

1.2 Transmission Control Protocol

Transmission Control Protocol (TCP) is the most famous transportation layer protocol. It is connection-oriented and designed to provide reliable end-to-end packet transmission over unreliable networks. TCP uses the cumulative ACK and corresponding retransmission mechanisms to ensure the correct delivery of data packets.

TCP uses two strategies for detecting packet loss. The first one is based on the sender's retransmission timeout (RTO) expiration. The other mechanism is based on the receiver's packet sequence number. Because TCP mainly uses cumulative acknowledgments, the receiver will generates a duplicate acknowledgment (DUPACK), which has the same sequence number as the last received packet, until the lost packet is received.

TCP has well designed congestion control mechanisms to adjust the sending window to effectively utilize the network bandwidth. The congestion control mechanisms implementation is closely related with the packet loss detection.

1.3 TCP-Reno over MANET

TCP achieves a nice performance over the wired networks. Because of TCP's success over wired networks and taking the interoperability of MANETs and the world wide Internet into consideration, applying TCP in MANETs is necessary.

TCP is designed for wired networks. The design of TCP's congestion control mechanism is based on the premise that the packet loss is an indication of network congestion. The most popular version is TCP-Reno. When a packet is detected to be

lost, either by timer timeout or triple duplicate ACKs, TCP slows down the sending rate by adjusting its congestion window. In the case of timeout, TCP enters the slow start phase and set its congestion window size to 1. In the case of triple duplicate ACKs, TCP enters congestion avoidance by halving its congestion window. Since bit error rate is very low and route failures are rare in wired networks, the assumption that the packets losses are due to congestion works well. The corresponding congestion window adjusting makes TCP utilize the bandwidth effectively.

However, the premise that packet loss means network congestion does not work any more for MANETs. In MANETs, frequent route change, channel and link errors decide that most of the packet losses are NOT due to network congestion. This is contradictory to the TCP premise. In this case, original TCP-Reno cannot adapt to the network dynamics well. TCP-Reno is quite aggressive in increasing its congestion window large. Thus, the wrongly undifferentiated window reducing action due to non-congestion packet losses when applied in MANETs is probably in a large extent. This reduction will lead TCP-Reno to achieve very low throughput. What is more, the constant route failures may cause many packets lost and long time may need to retransmit the lost packets. Thus, the aggressive window increasing of TCP-Reno will make many packets queued up. This will lead to large average packet delay and delay jitter, which is disadvantageous to real-time applications implementation.

We can summarize that the TCP-Reno performance in MANETs is sensitive to the network dynamics and thus badly degraded. The basic reason is that TCP-Reno cannot differentiate the packet loss due to network congestion with the loss due to other reasons. The aggressive congestion window increasing is another important reason, especially for the large average packet delay and delay jitter.

1.4 Research Motivation

1.4.1 Related Works to Improve TCP Performance

From section 1.3, we know that applying TCP-Reno directly in MANETs will achieve an unsatisfactory performance. Thus, many schemes (TCP-F, TCP-ELFN, TCP-Bus, etc, for details please refer to Chapter 2) are proposed to improve the performance. These schemes base on TCP-Reno and aim to improve the TCP-Reno throughput. As illustrated in section 1.3, the basic reason for TCP-Reno degraded performance is that TCP-Reno incorrectly perform congestion control in case of losses that are not induced by network congestion. The window reducing, halving or decreasing to 1 according to the congestion control algorithm, would lead to low throughput. Consequently, the basic idea for most of the above schemes to improve TCP-Reno throughput is to "give" the TCP source the ability to distinguish the congestion loss and the losses due to bit error or route failures. There are also some schemes that are aiming to improve the MANETs environment such as reducing route failures or improve MAC layer channel spatial reuse so as to improve the TCP throughput. The details will also be presented in Chapter 2.

1.4.2 Reasons to Propose TCP-Vegas: Vegas vs. Reno

The schemes in section 1.4.1 are all Reno-based. The aggressive window increasing of TCP-Reno will lead many packets queue up. The average packet delay cannot be reduced effectively. Moreover, as the MANETs environment is dynamic, the distinguishing information may be lost. The window size then may be reduced despite the strategy. The large window size of TCP-Reno will probably make this threat more detrimental because the window size reducing will be rather large. Thus,

for the dynamic MAMETs, a small window size based TCP version is preferred. The research works on congestion window limit settings (reference details in Chapter 2) also show that small window size can achieve better performance for TCP applied in MANETs. Consequently, we propose to apply TCP-Vegas in MANETs.

TCP-Vegas [2] achieves better performance than Reno in wired networks [3] [4]: higher throughput, less retransmission, less packet loss and more stable congestion windows. Vegas adopts a different mechanism to control the congestion window size. It uses the fine-grained measurement of RTT to estimate the difference between the expected and actual throughput to adjust the congestion window size accordingly. Therefore, Vegas can detect the incipient congestion and it is quite conservative when increasing its congestion window. This is contrary to Reno, which always aggressively increases its window. The small window size of Vegas, together with its lesser fluctuation, makes it less sensitive to the network dynamics. Moreover, smaller queues will build up for Vegas, which leads to less average packet delay and delay jitter.

We aim to reduce the average packet delay and delay jitter as well as improve the TCP throughput. Thus, we propose to apply TCP-Vegas in MANETs in this thesis. The details of the performance comparison of TCP-Reno and TCP-Vegas are presented in Chapter 3.

1.4.3 Re-Routing Problem and Vegas Routing Interaction Strategy (VRIS)

However, applying TCP-Vegas directly to MANETs would face the re-routing problem [5]. As shown in Figure 1.1 and Figure 1.2, before the route change, the TCP connection use the 3-hop route, *Base_RTT* = T1. After the route change, the TCP

connection use the 5-hop route, *Base_RTT* should be T2, but still as T1 according to Vegas implementation. That is, Vegas uses the up-to-date minimum RTT as an estimation of the *Base_RTT*, when the route changes to a longer one, Vegas is not able to increase the value of *Base_RTT*.

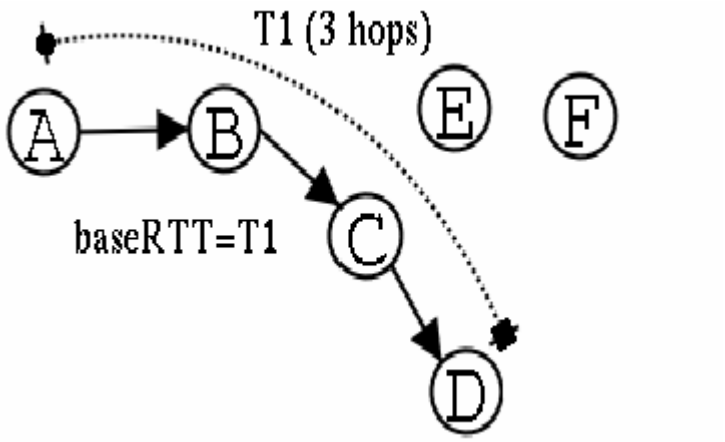

Figure 1.1 Illustration of Re-Routing problem triggered by node movement (Before movement)

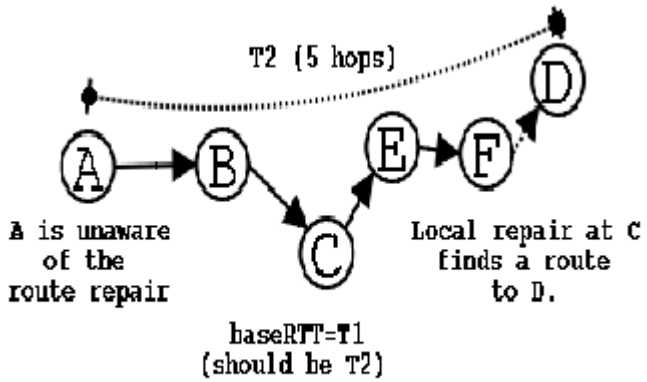

Figure 1.2 Illustration of Re-Routing problem triggered by node movement (After movement)

The wrong estimation of *Base_RTT* would lead Vegas to incorrectly reduce its sending window and cause low throughput. Thus, the re-routing problem should be solved to further improve Vegas throughput. One strategy to solve the re-routing problem is that La et al. [6] used any lasting increase in RTT as a sign of re-routing to update *Base_RTT*. However this strategy is mainly used for wired networks and could not adapt to MANETs very well. Correspondingly, we propose the Vegas-Routing Interaction Strategy (VRIS) based on the MANETs characters. We try to trace the route hop length information from the underlying routing agent and inform the corresponding Vegas agent immediately to direct the Vegas agent to update its sending window correctly and timely. VRIS can work well with both DSR and AODV. We mainly choose DSR for illustration and also give the results with AODV. The simulation results show that using the VRIS, Vegas can achieve throughput improvement for about 10% on average and up to 223.01%. The main content of the VRIS is presented in [29].

1.4.4 Unfairness Problem and ASCWL Strategy

Although our prime research goal is to improve the TCP throughput as well as reduce the average packet delay in the general mobile scenarios, we should also emphasize the unfairness problem in the static scenarios. Here we emphasize the static scenario because it is hard to define the unfairness problem in mobile scenarios. The severe unfairness problem exists for both TCP-Reno and TCP-Vegas. One connection may achieve poor throughput even flow starvation. What is more, most of the related research works on the unfairness problem are focused on identifying the factors triggering the problem and the solving methods are rare. The proposed strategies can alleviate the problem, but their implementations are complicated. The details can be

found in the Chapter 2. Thus, we will try to propose a TCP-based strategy to improve the TCP fairness and make the strategy easy to implement and work for both Reno and Vegas.

The MAC layer factors such as hidden and exposed terminal problems, binary exponential backoff strategy and channel capturing basically trigger the unfairness problem. However, TCP's inadequate interaction with the MAC layer further exacerbates the problem. The above MAC factors are intrinsic for MANETs. To get rid of them is difficult. For example, although IEEE 802.11 uses RTS/CTS mechanism, the hidden and exposed terminal problems still exist. Thus, we aim to modify the TCP implementation based on the MANETs characters so as to adapt TCP to the MANETs environments to improve the TCP fairness. We find that the negative impact of TCP over MAC can be effectively alleviated by dynamically setting the Congestion Window Limits (CWLs) based on the optimal CWL. Thus, we propose the Adaptively Setting Congestion Window Limit (ASCWL) strategy which uses the hop number to set CWLs for the contending TCP connections as a means of solving the unfairness problem. By simulations, we show that our strategy can improve the fairness index by up to 89.21% (see result tables in Chapter 5 for reference). Further more, the overall throughput usually increases as well. The details of the ASCWL strategy are in [30].

1.4.5 Research Tasks

Now, we can summarize our research tasks as follows:

1) Apply TCP-Vegas in MANETS and prove that Vegas can achieve better performance than Reno;

2) Propose a strategy (VRIS) to solve the re-routing problem in the general mobile scenarios to further improve Vegas throughput;

3) Propose a strategy (ASCWL) to improve the fairness index of the contenting TCP connections in the static scenarios.

1.5 Thesis Organization

The rest of the thesis is organized as follows:

In Chapter 2, some related research works on improving TCP throughput in MANATs are reviewed. Some related works on TCP unfairness problem are also presented.

In Chapter 3, we will compare the performance of TCP-Vegas and TCP-Reno in MANETs and prove that Vegas achieves a better performance: higher throughput, less average packet delay and delay jitter.

In Chapter 4, we propose the VRIS to solve the re-routing problem in the mobile scenarios. We implement VRIS with both DSR and AODV. We mainly use DSR to illustration and also present the result with AODV.

In Chapter 5, we propose the ASCWL to solve the unfairness problem in the static scenarios.

Chapter 2

Literature Review

2.1 Introduction

In this chapter, we will present some related research works on improving TCP performance in MANETs. The first part is the related schemes to improve the TCP throughput. The second part is focused on the unfairness problem.

2.2 Schemes to Improve the TCP Throughput

As stated in Chapter 1, the main approaches are based on the basic idea to "give" the TCP source the ability to distinguish the packet loss due to network congestion and the loss due to other reasons so as to avoid the unnecessary window reducing. These approaches are based on TCP-Reno. According to the different loss conditions, some implementations are modified to adapt to the MANETs dynamic environment. The difference between these approaches lies in how notifications are done and how the corresponding reactions are performed. Other approaches are aiming to improve the MANETs environments, such as the MAC layer spatial reuse and network work layer route availability. The increase of the lower layer reliability will also bring the TCP

throughput improvement. One famous example is the backup routing strategy (see 2.2.1.6) to offer the backup path in addition to the primary communication route.

The strategies can be classified into 2 classes according to the implementing method: cross layer interaction proposals and one layer implementing proposals. In the cross layer proposals, TCP and its underlying protocol work jointly. In the layered proposals, the problem of TCP is attacked at one layer. Cross layer proposals usually report higher throughput improvement. Mobility, link breakages and routing failures are taken into consideration in most of these proposals.

2.2.1 Cross Layer Strategies

The Cross layer strategies can be classified into 3 types:

- TCP and network cross layer,
- TCP and physical cross layer,
- Network and physical cross layer.

The following 4 schemes, TCP-F, TCP-ELFN, TCP-Bus and ATCP belong to the TCP and network cross layer strategies. The next two routing schemes belong to the network and physical cross layer strategies.

2.2.1.1 TCP-F

TCP Feedback [7] is the feedback-based approach to allow the TCP sender to distinguish between losses due to route failure and those due to network congestion. The details are as follows. When routing agent detects the route failure, it explicitly sends a Route Failure Notification (RFN) packet to the source. On receiving the RFN, the source goes into the “snooze” state in which the TCP sender will stop sending packets and freeze the timers and the congestion window. When the route is re-

established, a Route Re-establishment Notification (RRN) packet is sent to the source. On receiving the RRN, the sender will continue the transmission with the previous timer and congestion window values resumed.

TCP-F is proved to work better than standard TCP. However, the simulation scenarios are quite special and not the general ones in ad hoc networks.

2.2.1.2 Explicit Link Failure Notification (ELFN)-based Technique

TCP-ELFN [8] is quite similar to TCP-F. The reasoning behind ELFN is to inform the TCP agent about route failures when they occur. On receiving the ELFN message, the source responds by disabling its retransmission timers and enters a "standby" state. During standby state, the sender probes the network to check if the route is restored. If the acknowledgement of the probe packet is received, TCP sender leaves the "standby" mode, resumes its retransmission timers, and continues to the normal operation. The values of the parameter RTO and Congestion Window (CW) are varied to optimize the ELFN performance.

TCP-ELFN can effectively improve the TCP throughput. However, the parameter values need to be selected accordingly.

A further ELFN-Based TCP-Freeze scheme [9] is proposed to decouple the link failure notification into two phases: the underlying routing protocol route failure detection and sender side routing protocol to TCP notification, which makes it an end-to-end scheme.

2.2.1.3 TCP-Bus

This strategy [10] also uses the network feedback to detect route failure events and then to take the corresponding reaction to these events. In this strategy, the buffering capacity of the mobile nodes is proposed and the source-initiated on-demand

Associativity-Based Routing (ABR) routing protocol is selected. The following are the detailed implementations.

Two control Explicit Notification messages are used to notify the source about the route failure and the route re-establishment. On receiving the Explicit Route Disconnection Notification (ERDN), the source stops sending. Similarly, on receiving the Explicit Route Successful Notification (ERSN), the source resumes data transmission.

During the route reconstruction phase, packets along the path from the source to the Pivoting Node (PN) used to detect route failures are buffered. And the retransmission timer is doubled to avoid the timeout events. To overcome the late retransmission due to the timer doubling, an indication is made to make the source selectively retransmit the lost packets in advance.

In TCP-Bus, reliable retransmission of the control message is implemented to ensure the reliability of the ERDN and ERSN.

TCP-Bus outperforms standard TCP and TCP-F. However, the evaluation is based only on the ABR routing protocol.

2.2.1.4 ATCP

Ad hoc TCP [11] inserts a layer called ATCP between the TCP and IP layers of the TCP source nodes. ATCP listens to the network layer feedback via the information provided by ECN or ICMP message. Then according to the information, ATCP puts TCP in one of the following states: persist, congestion control, retransmit, and normal. In each state, corresponding TCP behaviors are implemented.

ATCP can effectively improve the throughput. But its implementation is based on some special scenarios and some assumptions that might be somehow hard to meet such as ECN-capable nodes as well as sender node being always reachable.

2.2.1.5 Preemptive Routing in Ad hoc Networks

This scheme [12] is designed to reduce the number of routing failures. It is achieved by switching to a new route when a link of the current route is expected to fail in the future so as to reduce the number of routing failures. Specifically, when the signal power drops below a given preemptive threshold, the source is notified and proactively looks up for a new route. When the new route is available, the routing agent switches to this new route.

By simulations, this scheme can yield a considerable reduction of the number of route failures. This reduction of route failures can make TCP achieve higher throughput.

2.2.1.6 Backup Path Routing

Backup path routing scheme [13] uses only one path at a time but it maintains some backup paths and can switch from current path to another alternative path rapidly if current path fails. It is found that maintaining 2 paths is usually optimal for the best TCP performance. Shortest-hop path, shortest-delay path, and maximally disjoint path are the paths usually found and cached for the choice of primary path or the backup path. Two schemes of choosing the primary path and the backup path are proposed. Scheme 1 is to choose the shortest-hop path as the primary and the shortest-delay path as backup. Correspondingly, scheme 2 chooses shortest-delay path as primary and maximally disjoint path as backup.

As Scheme 2 does not use the shortest-hop path, it achieves worse performance than the original TCP. But Scheme 1 is shown to be able to improve the TCP throughput by 23% to 30%.

2.2.2 Layered Proposals

The layered proposals can be classified into the following two categories according to in which layer the proposal is implemented.

- TCP layer proposal,
- Link layer proposal.

The following Fixed RTO, CWL setting belong to the TCP layer proposals. The Link RED and Adaptive Pacing belong to the Link Layer proposals.

2.2.2.1 Fixed RTO

In this scheme [14], a heuristic is introduced to distinguish between route failures and congestion. When two timeouts expire in sequence, the sender concludes that a route failure has occurred. The unacknowledged packet is retransmitted but the Retransmission Timeout (RTO) is not doubled a second time. The RTO remains fixed until the route is re-established and the retransmitted packet is acknowledged.

The throughput can achieve effective enhancement. However, in case of congestion, the supposition that two consecutive timeouts are exclusively the indication of route failure needs more consideration.

2.2.2.2 CWL Setting

This scheme [15] is based on the study of the relationship between the hop number and the optimal Congestion Window Limit (CWL) achieving the best throughput. Given a specific network topology and flow patterns, there exists an optimal TCP window size limit, on which TCP achieves the best throughput via improving the spatial reuse. And usually, this window size is not too large. It was found that the bandwidth delay product (BDP) of multi-hop routes cannot exceed the Round-Trip Hop-Count (RTHC)

of the path. In the case of a multi-hop chain that implements IEEE 802.11 MAC, the BDP is bounded by 1/5 of the RTHC. Based on this bound, they propose to use the CWL setting algorithm to adjust TCP's maximum window size limit according to the path hop numbers. This scheme can improve the TCP throughput by up to 16%.

2.2.2.3 Link RED

This scheme [16] is to monitor the average number of retries in the packet transmission at the link layer. Then the scheme compares this number with a given threshold and drops/marks the packets by the probability calculated according to the RED algorithm. When the marked packets arrive at the destination, the sender will be informed to increase the backoff time at the MAC layer. This scheme can effectively reduce the contention on the wireless channel. The MAC layer optimization can also improve the TCP throughput.

2.2.2.4 Adaptive Pacing

This scheme [16] is to improve channel spatial reuse. In IEEE 802.11, a node is constrained from contending for the channel by a random backoff period, plus a single packet transmission time. However, the implementing of the above action can not solve the exposed terminal problem. Thus, the Adaptive Pacing scheme is proposed. This scheme can alleviate the problem by increasing the backoff period by an additional packet transmission time. It works together with LRED. When the average number of retries is less than a threshold, it calculates its backoff time as usual. When the average number is beyond this threshold, adaptive pacing is enabled and the backoff period is increased by duration equal to the previous packet transmission time.

As the Link RED, the MAC layer optimization can improve the throughput in the above TCP layer.

2.3 Related Works on TCP Unfairness Problem

It is known that substantial unfairness and even flow starvation exist in MANETs. As fairness is an important metric to evaluate the network performance, the severe unfairness problem should also be widely studied.

There are some related research works on this topic. However, most research on TCP unfairness problem is focused on identifying the factors which cause the unfairness problem. Some scheduling strategies can alleviate the unfairness problem, but their implementations need the MAC protocols modification. One effective method to solve the problem is the Neighborhood RED (NRED) strategy proposed by K. Xu and Gerla et al. [17], which is implemented on the network layer.

2.3.1 Related Works on Identifying the Triggering Factors

K. Tang and M.Gerla [18] used the static topologies such as chain and grid topology to compare the impact of different MAC protocols, namely CSMA, FAMA, MACAW and IEEE 802.11. IEEE 802.11 was found work best. S. Xu et al. [19] investigated the factors for IEEE 802.11 that cause the unfairness problem. K. Xu et al. found in [20] the greedy behavior of TCP and its poor interaction can exacerbate the unfairness situation.

2.3.2 Neighborhood RED

In NRED, each node in the network maintains its own distributed queue, called neighborhood queue. The neighborhood queue aggregates the node's local queue and the upstream and downstream queues of its 1-hop neighbors. Then, the RED algorithm is used based on this neighborhood queue to give the disadvantage connections less dropping probability and thus more chance to transmit packets.

NRED can effectively improve the TCP fairness. However, its implementation is somewhat complicated. Moreover, the fairness evaluation in the mobile scenarios needs more consideration.

2.4 Summary

Now we summarize the research works on improving TCP performance in MANETs. Most of the research works on this topic are focused on how to improve the TCP throughput. There are also quite a few works on the severe unfairness problem.

In the research works to improve the TCP throughput, most of them are focused on how to make the TCP source have the ability to distinguish the packet loss due to network congestion and the loss due to other reasons. According to how the distinguishing information is gotten, the strategies can be classified into 2 types: cross layer strategies and layered strategies. There are also some related works that are aiming to improve the MANETs environment such as reducing route failures or improve MAC layer channel spatial reuse so as to improve the TCP throughput. The implementations are focused on the TCP-Reno version.

In the research works on the unfairness problem, most of the works are focused on identifying the intrinsic MAC factors that trigger the unfairness problem. The

proposals to solve this problem are rare. One effective method is the Neighborhood RED (NRED) strategy proposed by K. Xu and Gerla et al. [17], which is implemented on the network layer.

Chapter 3

TCP-Vegas vs. TCP-Reno in MANETs

3.1 Introduction

In this chapter, we apply TCP-Vegas in MANETs. We will prove that Vegas can achieve a better performance than Reno via NS-2 simulation. First, we will present some background information on TCP-Reno and TCP-Vegas implementation. Then, we will compare their performance in MANETs.

3.2. Background Information on TCP-Reno and TCP-Vegas

3.2.1 TCP-Reno Congestion Window Control

TCP-Reno induces packet losses to estimate the available bandwidth in the network. The source assumes the packet losses are due to network congestion and the corresponding congestion mechanisms are implemented to control the sending window size. The TCP-Reno congestion control mechanisms include two phases: Slow Star and Congestion Avoidance.

Slow Start begins with congestion window size as 1. The window size will double once the sender received the ACK for the former sending packets until the window

size reaches the slow start threshold (ssthresh). Then the congestion control enters the Congestion Avoidance phase, during which the congestion window size is increased linearly by 1/*cwnd* each time an ACK is received. When TCP-Reno experiences a packet loss, it will reduce its window size. If the packet loss is due to triple duplicated ACKs, it will halves its congestion window size and enter the Congestion Avoidance phase. If the packet loss is due to the sender timer timeout, it will reduce the window size to 1 and enters the Slow Star phase.

3.2.2 TCP-Vegas Congestion Window Control

Vegas adopts a timestamp-based mechanism to control the congestion window size. It uses the fine-grained measurement of *RTT* to estimate the difference between the expected and actual throughput to adjust the congestion window size accordingly. The details are as follows:

$$Expected = cwnd / Base\_RTT \tag{3.1}$$

where *cwnd* is the size of the current sending congestion window and *Base_RTT* is the minimum *RTT* of that connection.

$$Actual = PktsTrans / rtt \tag{3.2}$$

where *PktsTrans* is the number of packets transmitted during the last RTT, rtt is the average *RTT* of the segments acknowledged during the last *RTT*. In the steady state, we can take *cwnd* as *PktsTrans*

$$\begin{aligned} Diff &= (Expected - Actual) * Base\_RTT \\ &= \left(\frac{cwnd}{Base\_RTT} - \frac{cwnd}{rtt}\right) * Base\_RTT \\ &= cwnd * \left(1 - \frac{Base\_RTT}{rtt}\right) \end{aligned} \tag{3.3}$$

$$cwnd = \begin{cases} cwnd + 1, & \text{if } Diff < \alpha \\ cwnd, & \text{if } \alpha \le Diff \le \beta \\ cwnd - 1, & \text{if } Diff > \beta \end{cases} \quad (3.4)$$

We can take Diff as the backlog in the router queue. α and β are constants which can be respectively taken as the lower and upper bound of the backlog. They usually took the value of 1 and 3 respectively.

3.2.3 Difference between Vegas and Reno

We can summarize the difference between Vegas and Reno as follows:

- Congestion control implementation

  TCP-Vegas has a fundamentally different congestion avoidance scheme from that of TCP-Reno as we can infer from the implementation rules above.

  Reno uses the loss of packets as a signal that there is congestion in the network and thus has no way of detecting any incipient congestion before packets losses occur. Thus, Reno reacts to congestion rather than attempts to prevent the congestion. Correspondingly, what Vegas tries to do is as follows. As we can see from the implementation, if the actual throughput is much smaller than the expected throughput, then it suggests that it is likely that the network is congested. Thus the source should reduce the flow rate. On the other hand, if the actual throughput it too close to the expected throughput, then the connection may not be utilizing the available flow rate, and hence should increase the flow rate. Thus, Vegas is conservative when increasing its congestion window size and so it can detect incipient congestion.

- Packet delay and jitter

As Vegas is quite conservative when increasing its congestion window size, only a few packets will queue up in the buffers. On the other hand, Reno is very aggressive when increasing the window size. Thus, the average packet delay and delay jitter of Vegas tend to be much smaller than those of Reno.

- Retransmission mechanism

  Vegas has better retransmission mechanisms. For TCP-Reno, a rather coarse grained timer is used to estimate the *RTT* and the variance, which results in a poor estimation. Vegas, on the other hand, uses a fine-grained timer. Vegas records the system clock each time a packet is sent. When an ACK is received, Vegas calculates the *RTT* and uses this more accurate estimation to decide to retransmit the packet in the following two situations: (1) When it receives a duplicate ACK, Vegas checks to see if the *RTT* is larger than timeout. If it is, then without waiting for the third duplicate ACK, it immediately retransmits the packet. (2) When a non-duplicate ACK is received, if it is the first or second ACK after a retransmission, Vegas again checks to see if the *RTT* is greater than timeout. If it is, then Vegas retransmits the packet.

3.3. TCP-Vegas vs. TCP-Reno in MANETs

Vegas achieves a much better performance than Reno in wired networks: higher throughput, less retransmission, less packet loss and more stable congestion windows. Now we consider their performance in MANETs. The small window size of Vegas, together with its lesser fluctuation, will make it less sensitive to the network dynamics. Moreover, smaller queues will build up for Vegas, which will lead to less average

packet delay and delay jitter. Thus we expect that Vegas will also achieve better performance in MANETs than Reno. We will prove this by NS-2 [21] simulations.

3.3.1 Static Scenario

First, we use a chain topology as shown in Figure 3.1 to study the multi-hop wireless transmission in MANETs.

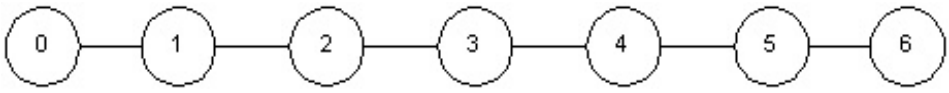

Figure 3.1. Chain Topology

Table 3.1 Parameter Settings

Wireless Channel Raw Capacity	2M bps
Radio Transmission Range	250 meters
Carrier Sense Range	550 meters
Adjacent Node Distance	200 meters
MAC	IEEE 802.11
Routing Protocol	DSR
Default TCP Packet Size	1000 bytes
interface queue limit	50 packets

The parameter settings in Table 3.1 decide that only the adjacent nodes can transmit packets directly. We set up TCP connections with different route hop numbers. That is, we set node0 the source and node1, node2 … node6 the destination respectively. For each TCP connection, the simulation is run for 500s.

The simulation results are shown in Figure 3.2. For the 1 and 2 hop(s) situations, Reno gets almost the same throughput as Vegas. But for larger hop numbers, Vegas achieves better throughput performance. As the hop number increases, the throughput of both Vegas and Reno decreases exponentially. Here, by throughput we refer to the

goodput, which means the throughput without the header and excluding the retransmission. We follow the same good definition in the following chapters.

After MAC and TCP layer tracing, we find link failure and TCP sending window control strategy are two main factors that contribute to the simulation result in Figure 3.2. Since IEEE 802.11 can not totally solve the "hidden and exposed terminal" problems, DATA and RTS packets in MAC layer may conflict with ACK or CTS and be lost. When MAC layer retransmits for 7 times and still fails, it will send link failure message to the upper layer, which will usually result in TCP retransmission and even timer timeouts. It is well known that Vegas and Reno have different window control strategy. Reno always tries to increase its sending window until it gets some sending packet lost. Vegas, on the other hand, is more conservative. It uses the fine-grained measurement of *RTTs* to control its window size. It always avoids packet loss and makes its sending window stable. Hence, Reno puts more packets into the network than Vegas and has more packet loss than Vegas. The following window halving or even reducing to 1 will bring larger performance degradation for Reno.

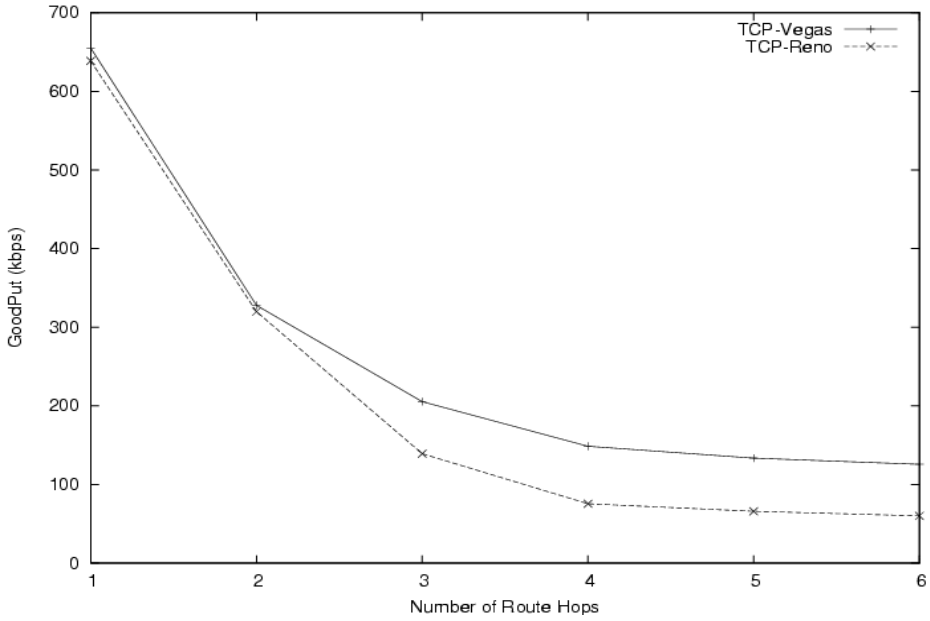

Figure 3.2 Goodput vs. Hop Number for Reno and Vegas

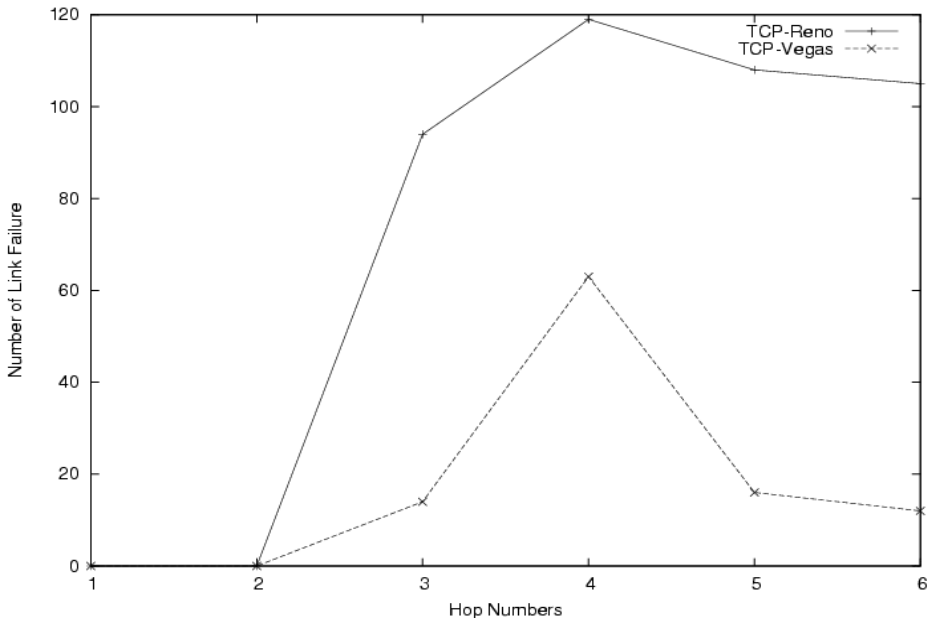

Figure 3.3 Seven Times Retransmission Failure Numbers vs. Hop Number For Reno and Vegas

As shown in Figure 3.3: for the 1 or 2 hop cases, as the source node can "hear" the destination node, both Vegas and Reno have no link failures; for the larger hop cases, Reno gets much more "link failures" than Vegas on average. Tracing the IFQ length, we find that Reno uses more queue space than Vegas. Consequently, Vegas achieves higher throughput. Moreover, Vegas achieves shorter average packet delay and delay jitter than Reno. This is important for applications such as interactive multimedia and network games. We put the 4-hop cases results here. The average packet delay of the 4-hop (node0 to node4) connection for TCP-Reno is shown in Figure 3.4, for TCP-Vegas is shown in Figure 3.5.

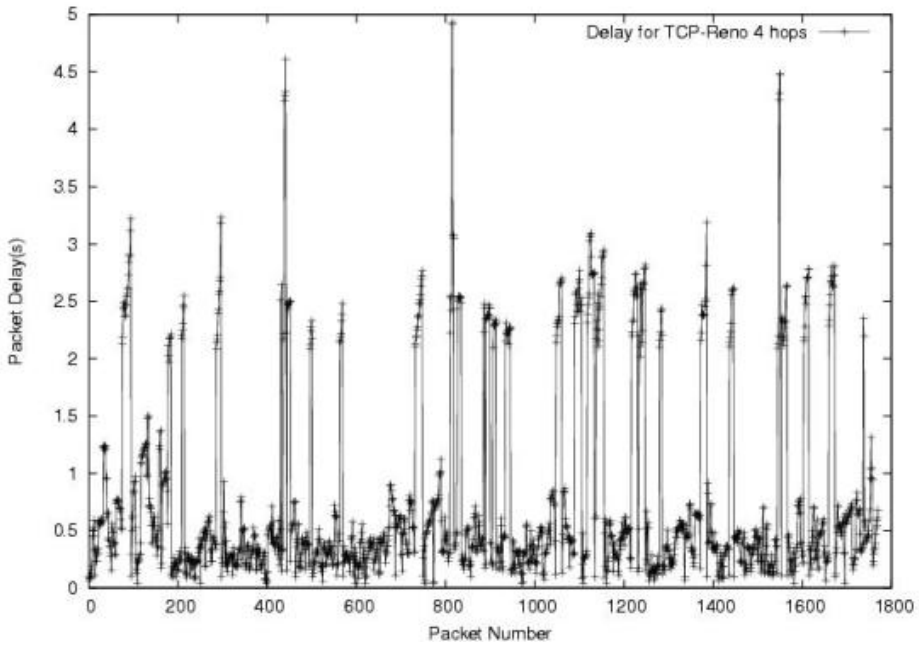

Figure 3.4 Average Packet Delay of the node0 to node4 connection for TCP-Reno

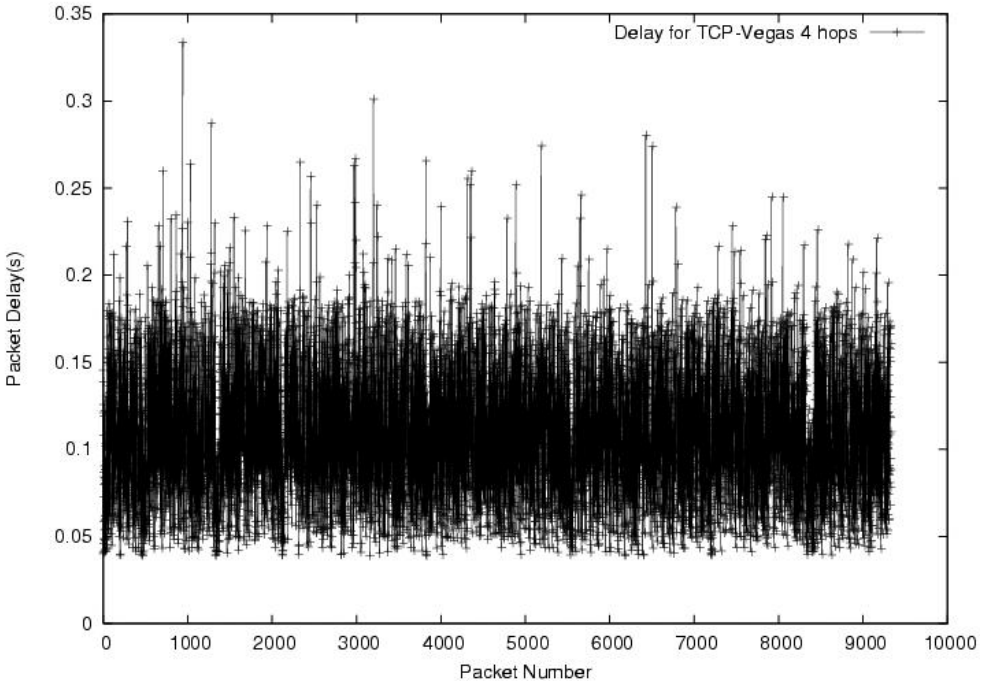

Figure 3.5 Average Packet Delay of the node0 to node4 connection for TCP-Vegas

3.3.2 Mobile Scenario

This part will focus on the "topology change" of MANETs. We will use the random way-point model [22]. In this model, each node is placed randomly in the simulated area and remains stationary for a specified pause time. After the pause time, a node randomly selects a destination (x y-coordinates) and moves in the direction of the destination point at a speed uniformly chosen between a minimum (usually 0) and maximum value. In our simulation, each of the 7 nodes randomly selects a destination point in a 1500*500m rectangle area. Other parameters are the same in Table 1 expect the adjacent node distance. The max node speed is varied as 10, 20 and 30m/s and the pause time is chosen as 2s. The simulation is run for 150s. Still we try to setup a TCP connection between node0 and node 1, node2… node 6 respectively. In order to be consistent with the static scenario and due to the non-correlation of the random seed for the different average node speed, we put the one-seed simulation results as follows in figure 3.6, figure 3.7 and figure 3.8 for average speed of 10, 20 and 30 m/s respectively. We vary the number of nodes as well as simulation scenarios with different seeds and get similar result trend.

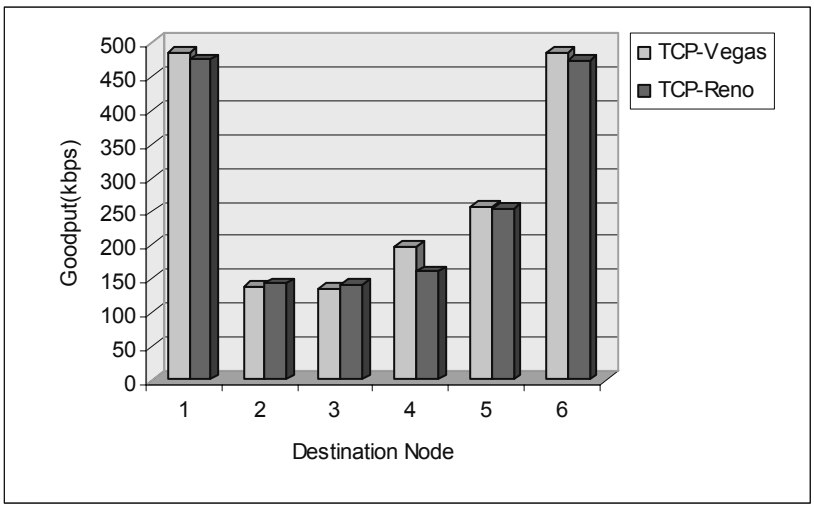

Figure3.6 Goodput of TCP from Node0 to Different Destination Node (max speed=10m/s)

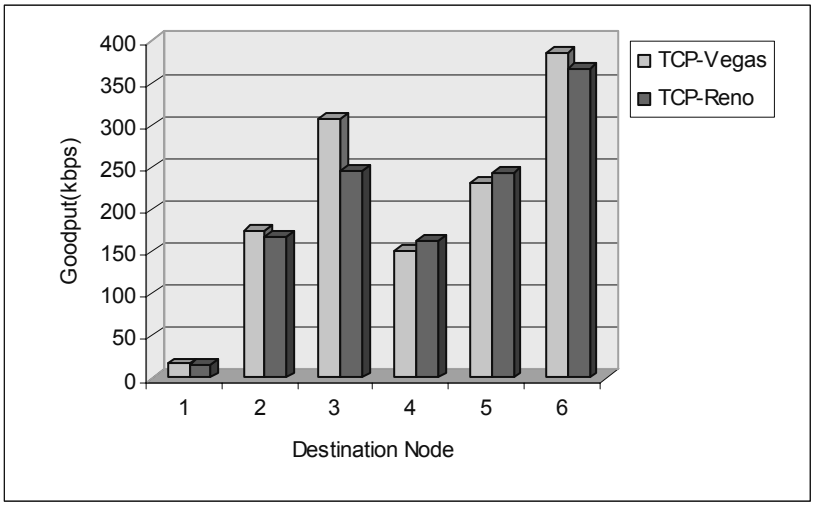

Figure3.7 Goodput of TCP from Node0 to Different Destination Node (max speed=20m/s)

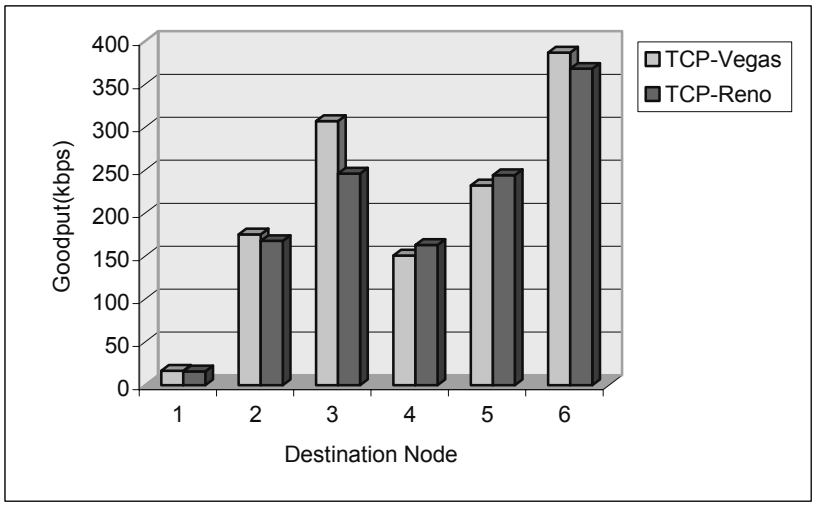

Figure3.8 Goodput of TCP from Node0 to Different Destination Node (max speed=30m/s)

Despite the re-routing problem, Vegas outperforms Reno in most cases. In mobile scenarios, the mobile nodes are always moving and so the route to the destination is always changing. Consequently, there are two factors causing route failure: MAC collision and node movement. Reno is more aggressive than Vegas. It always

increases its sending window until packet loss happens. Consequently, Reno tends to lose more packets and the following window reducing will yield worse result than Vegas. Moreover, routing traces show that nodes movement may bring shorter new routes and thus “re-routing problem” doesn’t exist.

In mobile scenarios, Vegas also achieves less average packet delay and delay jitter. Figure 3.9 and Figure 3.10 are one of the node 0 to node 3 connection scenarios for Reno and Vegas respectively.

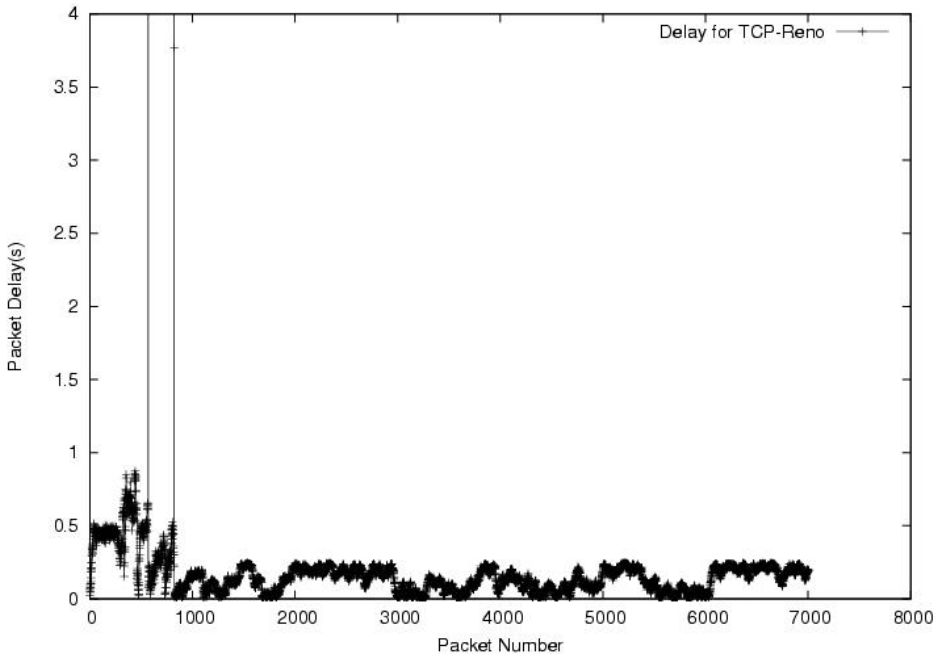

Figure 3.9 Average Packet Delay of the node0 to node3 connection for TCP-Reno

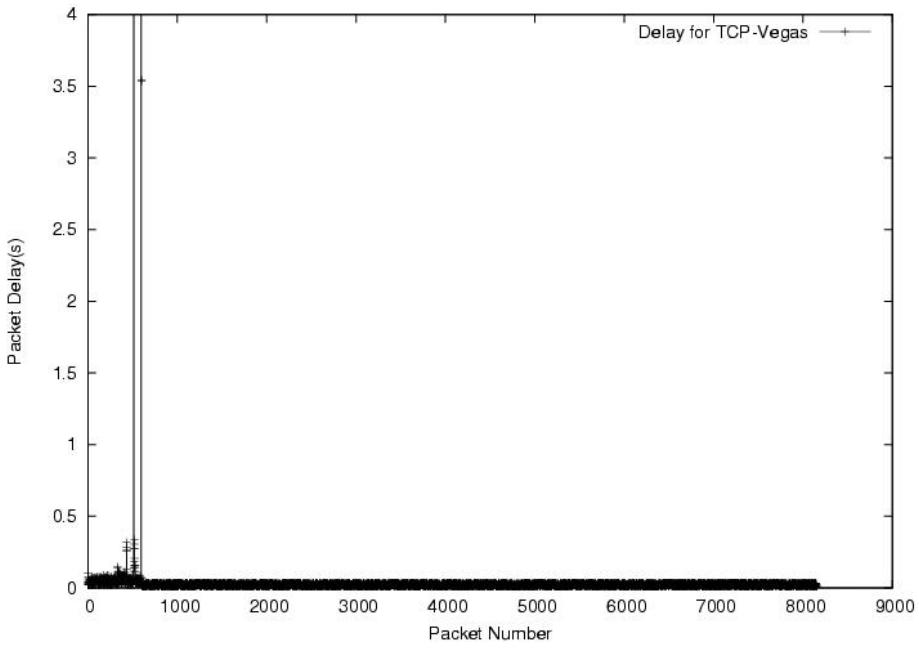

Figure 3.10 Average Packet Delay of the node0 to node3 connection for TCP-Vegas

The vertical lines in Figure 3.10 show that occasionally in the mobile scenario, due to the node mobility, some packet may also experience a long delay.

3.3.3 Summary

In summary, Vegas achieves a better performance when applied in MANET than Reno. In static and most mobile scenarios, Vegas achieves higher throughput than that of Reno. Moreover, Vegas achieves less average packet delay and delay jitter than Reno. This is important for the future applications such as interactive multimedia and network games. However, the re-routing problem is intrinsic for Vegas applying in MANETs. So we need to solve it so as to achieve further throughput improvement for Vegas.

Chapter 4

Vegas Routing Interaction Strategy (VRIS)

4.1 Introduction

In this chapter, we propose the Vegas Routing Interaction Strategy (VRIS) to solve the intrinsic re-routing problem to improve the TCP-Vegas throughput.

The re-routing problem [5] is rooted in TCP-Vegas implementation. Vegas uses the propagation delay as *Base_RTT* to adjust its congestion window size. When implementing, the up-to-date minimum round trip time (*RTT*) is used for the estimation of *Base_RTT*. Vegas is not able to increase the value of *Base_RTT* when the route changes to a longer one and the actual propagation delay have also increased, because it uses the **up-to-date minimum** *RTT* as the estimation. The wrong estimation of *Base_RTT* would lead Vegas to incorrectly reduce its sending window and thus cause very low throughput.

In MANETs mobile scenarios, as the mobile nodes are always moving and the topology is dynamic, there is a high probability of the re-routing problem occurrence. The mostly used strategy to solve the re-routing problem is that La et al. [6] use any

lasting increase in *RTT* as a sign of re-routing problem occurrence and thus update *Base_RTT* to a larger value. For simplicity, we call this strategy Lasting Increase of *RTT* (LIR).

The details of LIR implementation are as follows:

(1) LIR uses same mechanisms as the original TCP-Vegas for the first *K* packets. When it receives the ACK for the *Kth* packet, it computes the difference *diff_estimate* between *Base_RTT* and RTT_K, where RTT_K is the round trip delay of the *Kth* packet. This difference provided a rough estimate for the increase in the round trip delay due to its own packets queued at the buffers. An average of several differences could be used instead of one value. For simplicity, only one value is used for *diff_estimate* in LIR.

(2) After the ACK for the *Kth* packet is received, the source checks the smallest round trip delay of every N packets, which is denoted by *BaseRTTestimate*. If the difference between the minimum round trip time of N packets and *Base_RTT* is larger than $diff\_estimate + \min\{\delta * baseRTT, \gamma\}$, where $0 < \delta < 1$ and $\gamma > 0$ are prefixed parameters, for L consecutive times, then the source interprets this as a change in propagation delay, i.e., change in route, and (a) sets the *Base_RTT* equal to the minimal round trip time of the last N packets, and (b) sets the congestion window size *cwnd* to $cwnd = cwnd * \frac{baseRTTestimate}{baseRTT} + 1$

The basic idea behind this mechanism is as follows. If the minimum round trip time computed for N packets is consistently much higher than the sum of *Base_RTT* and *diff_estimate*, then it is likely that actual propagation delay is larger than the measured *Base_RTT*, and it makes sense to increase *Base_RTT*. Since the increase in delay

forces the source to decrease its window size, the round trip delay comes mostly from the propagation delay of a new route. Thus, the minimum round trip delay of the previous N packets is a good estimate of the new propagation delay, as is *Base_RTT* for the current route.

The parameter values used in their scheme implementation are K=100, N=20, δ=0.2, L=4, and γ=100 ms. However, to choose the appropriate values for LIR is still an open problem.

The LIR strategy is mainly design to deal with the re-routing problem in the wired networks. In MANETs, re-routing problems are mainly triggered the node movements and the following communication route changes. Thus LIR cannot adapt to the MANETs environment well since it cannot find the route changes timely and correctly. Based on the MANETs characters, we propose the Vegas Routing Interaction Strategy (VRIS) to resolve the Re-Routing problem.

In the following sections, we will give a detailed illustration of VRIS. Firstly, the VRIS scheme design is presented. Secondly, we will describe the VRIS implementation with the DSR and AODV. Thirdly, the Enhanced VRIS (E-VRIS) is extended. Then we will present the scheme analysis and explanation. The simulation results are as following.

4.2 Vegas Routing Interaction Strategy (VRIS)

Based on the MANETs characters of constant node movement and route change, we propose our Vegas Routing Interaction Strategy (VRIS). The basic reasoning behind VRIS is that we can trace the route change via the routing agent in the IP layer, and inform the corresponding Vegas agent about this routing change timely and correctly.

Then Vegas agent will be able to adjust its congestion window size correspondingly according to this information.

4.2.1 Scheme Design

The basic scheme for VRIS strategy is the CB-Vegas algorithm, i.e. "Change" the Vegas parameter "*Base_RTT*".

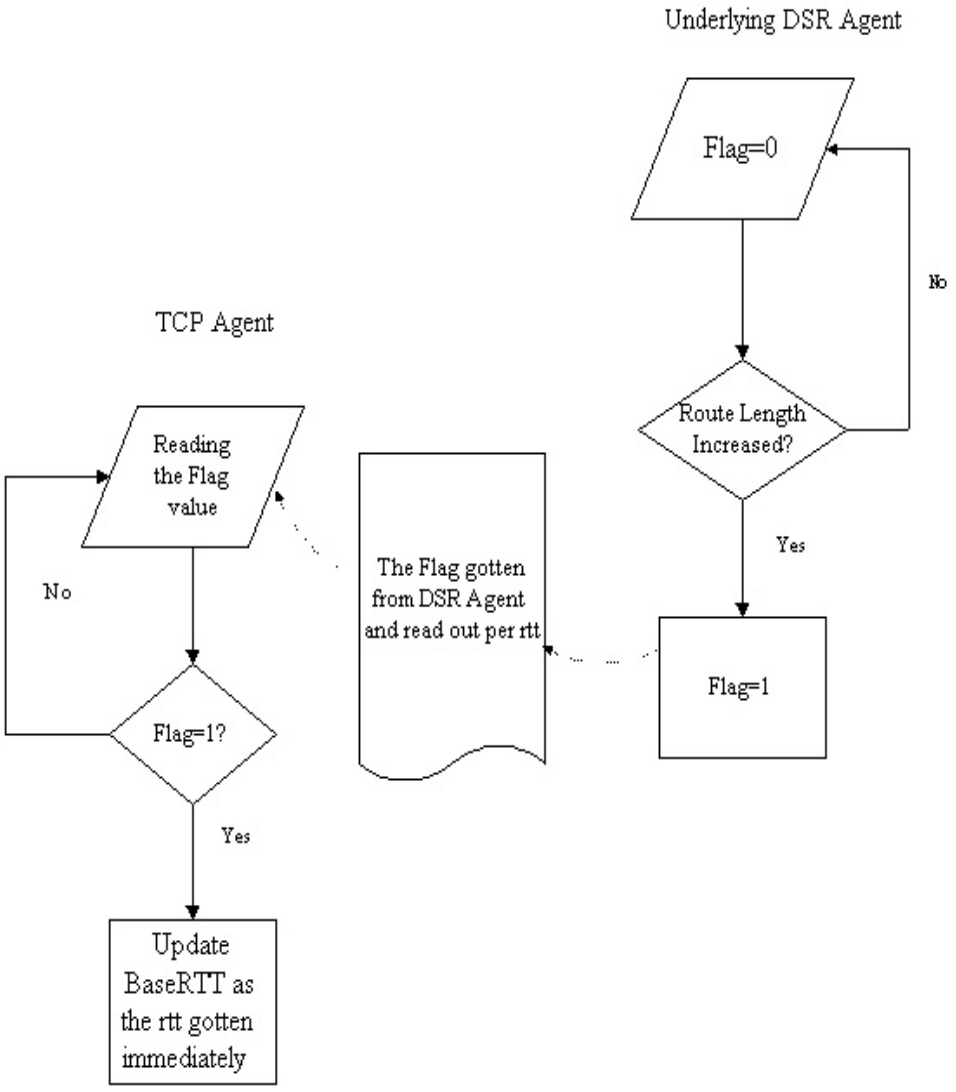

Figure 4.1 CB-Vegas Algorithm Flowchart

CB-Vegas is summarized as in Figure 4.1. We know node mobility is a very important characteristic of MANETs. Thus we can make the underlying routing agent

"monitors" the route changes. We should claim that the "route change" refers to the change of the hop number of the communication route. The reasoning is that hop distance is a key factor deciding the *RTT* and geographic distance in a limited distance area as in MAMETs is much less a factor that can be omitted. If the route hop number is increased, we will set Flag=1 and immediately inform the corresponding Vegas connection. Then the TCP Agent can update its *Base_RTT* as the value of *RTT* gotten right after detecting the route change. The Flag passing from the DSR agent to Vegas agent can be easily fulfilled in NS-2 by two agent-classes communication. When implementing, the Flag passing can be added in the SAP function from the network layer to the transport layer.

DSR and AODV are the two mostly used on-demand routing protocols. Thus we will implement our VRIS strategy with both DSR and AODV as the routing protocol. We will mainly use DSR to illustrate the algorithm and for AODV we give the simulation results.

4.2.2 VRIS with DSR and AODV

VRIS can work well with all the routing protocols. Since DSR and AODV are the mostly used on-demand routing protocols in MANETs, we will illustrate the implementation of VRIS with DSR and AODV respectively.

4.2.2.1 DSR (Dynamic Source Routing) Routing Protocol

DSR is composed of two main mechanisms that work together to allow the discovery and maintenance of source routes in MANETs. The two mechanisms are Route Discovery, undertaken when source needs a route to a destination, and Route Maintenance, used when link breaks render specified route unusable. When

implementing Route Discovery, the source node flooding the network with a RREQ for the specified destination. And the intermediate nodes will reply to the source with a RREP if a path to the destination is stored or it will add itself to the route record and broadcast the message toward the destination. On receiving the RREQ, the destination will unicast the RREP to the source with the complete route built by intermediate nodes. When link breakages occur, RERR is sent back to the source and the intermediate nodes will adjust cached routes. If the intermediate nodes or the source have one cached route available to the destination, the new routes can be used to deliver the packet. Otherwise when the REER reaches the source, the source will initiate another Route Request.

For DSR, the routing information is first found and then put into the header of all the data packets. Thus, to implement VRIS, we can get the route hop number information from the packet header.

4.2.2.2 Ad hoc On Demand Distance Vector (AODV) Protocol

AODV builds routes between nodes only as desired by the source nodes. It uses route request/route reply query cycle. When a source node desires a route to a destination for which it does not already have a route, it broadcasts a route request (RREQ) packet across the network. Nodes receiving this packet update their information for the source node and set up backward pointers to the source node in the route tables. In addition to the source node's IP address, current sequence number, and broadcast ID, the RREQ also contains the most recent sequence number for the destination of which the source node is aware. A node receiving the RREQ may send a route reply (RREP) if it is either the destination or if it has a route to the destination with corresponding sequence number greater or equal to that contained in the RREQ. If this is the truth, it

unicasts a RREP back to the source. Otherwise, it rebroadcasts the RREQ. As the RREP propagates back to the source, nodes set up the forward pointers to the destination. Once the source nodes receive the RREP, it may begin to forward data packets to the destination.

For AODV, the route hop number information is stored in the routing tables. Thus, to implement VRIS, we can trace the updating of the routing table to get this information.

4.2.3 E-VRIS

Based on CB-Vegas, we can further extend the FW-Vegas scheme and incorporate FW-Vegas with CB-Vegas to propose the Enhanced VRIS.

FW-Vegas, i.e. "Fix" the Vegas sending "Window", is quite similar to CB-Vegas. The only difference is that the sending window will be fixed for the present *RTT* apart from updating the *Base_RTT*.

The key issue of Enhanced VRIS is to selectively use CB-Vegas or FW-Vegas according to the real-time network conditions.

We try to compare RTO_{new}, the Retransmission Timeout (RTO) after the route change, and RTO_{old} , the RTO before the route change.

If (*RTO new* <= αRTO old*)**

CB-Vegas

Else

FW-Vegas

The parameter α is a constant between 0 and 1 whose value is chosen to consider the node speeds factor, where a smaller value for α indicates higher node speeds. In our following implementation, we choose α as 1 for the 10m/s case, 0.95 for the 20m/s

case, and 0.925 for the 30m/s case. The above 3 values may not be the optimal values. They are randomly chosen for taking the node speeds factor into consideration.

E-VRIS is an enhancement over VRIS in the general mobile scenarios. The core idea is still the VRIS. Thus, we only give the E-VRIS results for the general mobile scenarios with DSR routing protocol.

4.2.4 Scheme Analysis and Explanation

A detailed explanation about the principles behind our scheme design is presented in this part.

As we know, TCP-Vegas updates its sending window according the *RTTs* it observed. The details are as shown in section 3.2.2.

As shown above, *Base_RTT* is a very important parameter to update *cwnd*. As discussed in the former chapters, re-routing can cause inaccuracy of *Base_RTT*. When the route for the TCP-Vegas connection has changed, *Base_RTT* may also change. Vegas is usually not affected if the new route has smaller *RTTs*, as *Base_RTT* will immediately be updated. But when the new route has larger *RTTs*, it will cause problem to Vegas. Vegas cannot tell whether the *RTT* becoming larger was caused by network congestion or route changes. Therefore it always assumes that the increase of *RTT* was due to the network congestion and consequently it will decrease *cwnd*.

However, we can infer from (3.1) (3.2) (3.3) (3.4) that, as *Base_RTT* becomes larger in fact, we should also increase *cwnd* to make *Diff* balanced between α and β. We can consider this from another point. When *RTTs* become larger, as the bandwidth-delay product increases, the number of packets that can be held in the network increases. Thus *cwnd* should also be increased so that to satisfy (3.4). The untimely *Base_RTT*

updating can limit the *cwnd* increase. Based on the above arguments, we propose the CB-Vegas scheme.

CB-Vegas can effectively solve the re-routing problem to improve the Vegas throughput. However, it may have some limitations for the following reasons.

The first is due to the route-cache use after route failures. In DSR or AODV, when a route is broken, some mechanisms are performed to use the salvage route to transfer the data packets in flight. For DSR, if the intermediate nodes or the source has one cached route available to the destination, the new routes can be used to deliver the packets. For AODV, the local repair mechanism is used to update the routing table to find the salvage route. The salvage route may be probably much longer and thus we update *Base_RTT*, then Vegas will increase *cwnd*. Because the route caches may be stale, especially for the aggressive cache use of DSR, the salvage route may probably be unable to succeed. Thus, increasing *cwnd* may cause more packets lost.

Another reason is because of the instability of *cwnd*. As mentioned above, the salvage route may be stale and cannot succeed, if we increase *cwnd* for the present *RTT*, we may need to decrease *cwnd* the following *RTT*, which will make *cwnd* fluctuate. However, the main advantage of TCP-Vegas is to maintain *cwnd* stable. This advantage will be lost in this case.

The third reason is that when there are route changes the MANETs is usually unstable at the moment and the packets are easily lost. If we increase cwnd, more packet losses may cause numerous retransmissions and even timeouts. FW-Vegas can effectively alleviate the above negative effects. Using this scheme, on one hand, we can prevent Vegas from oppositely changing *cwnd*, that is, when the new route is longer, it does not decrease *cwnd* any more. This will solve the re-routing problem. On the other

hand, we can make *cwnd* more stable and thus inherit the advantages of TCP-Vegas. However, FW-Vegas always keeps Vegas from increasing *cwnd*, which may prevent the full use of the bandwidth. Thus, we propose the RTO comparison mechanism to decide when to use CB-Vegas or FW-Vegas.

Vegas adapts its behavior to the network by considering the *RTT* of the packets. It computes a smoothed *RTT* (*SRTT*), which is a weighted average of all the past *RTT* values and the current *RTT*, with more weights allocated to the former. Thus, we can claim that *SRTT* can be used to indicate the network path conditions. When the *SRTT* updated with the current *RTT* is larger than the before-updating value, it shows that the new path has rather large *RTTs* and the path quality doesn't improve. Thus, it is better to use FW-Vegas. On the contrary, the new path has better quality, we will use CB-Vegas to better utilize the bandwidth. In TCP implementation, *SRTT* is then used to determine *RTO*. For simplicity, we will compare the *RTO* calculated with and without the current *RTT* to decide whether CB-Vegas or FW-Vegas will be used.

4.3 VRIS Simulation Implementation

We will simulate VRIS with both DSR and AODV. We will mainly use DSR for illustration. The simulations are performed in steps. For the general mobile scenarios, we will compare the simulation result with VRIS, E-VRIS, LIR and standard TCP-Vegas. For AODV, we will give the simulation results with VRIS in the general mobile scenarios.

First, we will discuss the issues of the simulation results statistics in the general mobile scenarios. Then, simulation result and performance evaluations are presented.

4.3.1 Simulation Result Statistic: 30 Scenarios Average

By simulation, we find that TCP throughput is greatly related to the mobile scenarios. TCP performance may vary a lot with the same simulation background settings, such as the same number of nodes, same max node speed and the same node pause time between movements, but different moving steps. A scenario is corresponding to a detailed set of node movements. The following are some simulation examples.

The simulation settings are as follows. We use the random-way point model. Each of the 20 nodes randomly selects a destination point in a 1500*500m rectangle area. The maximum node speed is set as 10m/s and the pause time is chosen as 2s. We randomly create two simulation scenarios with different seeds. We randomly choose a node pair (node 5 as the source, node 15 as the destination) for the TCP-Vegas connection. The simulation is run for 150s for each scenario. The simulation results are as in Figure 4.1. We can see, for the first scenario, TCP achieves almost no throughput. But for the second one, the throughout is much higher.

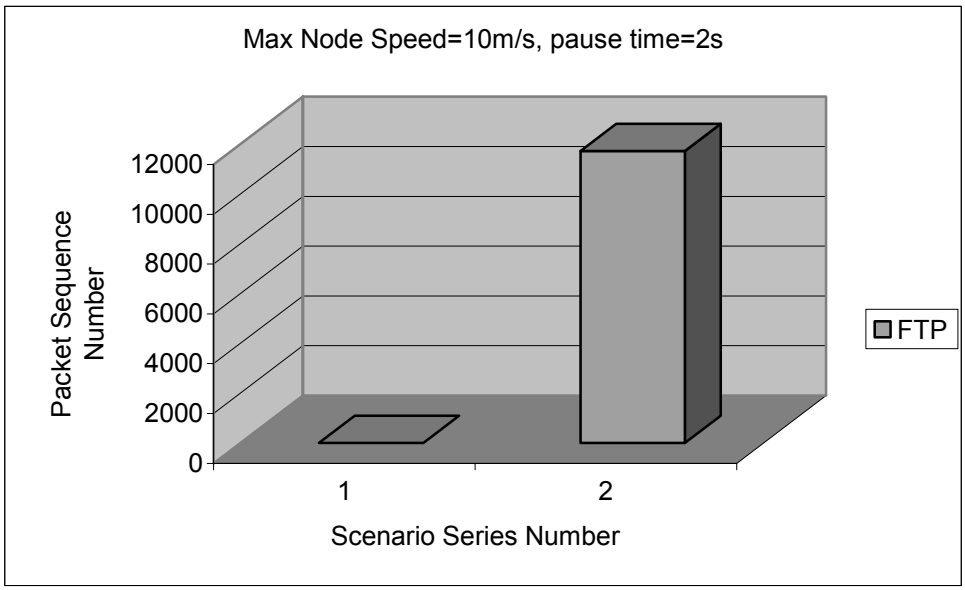

Figure 4.2 TCP-Vegas Throughput for Different Scenarios

We also find that higher max node speed does not mean that the TCP connection will achieve worse throughput. In some scenarios, although the nodes have higher speeds, the source destination pair may have better communication routes with less hop numbers. Moreover, with higher speeds, the network may recover quickly from an early route failure and a stable route is quickly established. Thus, we can state that there is no direct correlation between throughput and node speed.

The above two findings adapt to both TCP-Reno and TCP-Vegas.

Thus, in order to minimize any arbitrary randomness, we use 30 different seeds to simulate 30 different random node movement scenarios for one TCP-Vegas connection. The final result is the average of the 30 runs. Taking 30 scenarios on average into consideration, we find that lower speed really means higher throughput.
For Vegas, Not in every scenario does the re-routing problem exist. When it does exist, VRIS can effectively improve the throughput of the TCP connection. Thus, it should be noted that for the scenario where the re-routing problem is absent, VRIS may not be able to show improvements. The 30 scenario average is based on 30 general scenarios.

4.3.2 Simulation Result and Performance Evaluation

4.3.2.1 VRIS with DSR

We try to simulate our strategy with NS-2 in the following steps.

The first step is quite simple. We just use a designed topology to verify the feasibility of the VRIS strategy. The topology change is shown in Figure 4.3. At first the topology is as the above. At 70s node 2 leaves its position and goes upwards very quickly. The simulation is run for 150s. The parameter settings are as in Table 3.1.

The communication routes are as shown in the figure. The simulation result is shown in Figure 4.4. The throughput has improved from 78.5 kbps to 82 kbps with VRIS.

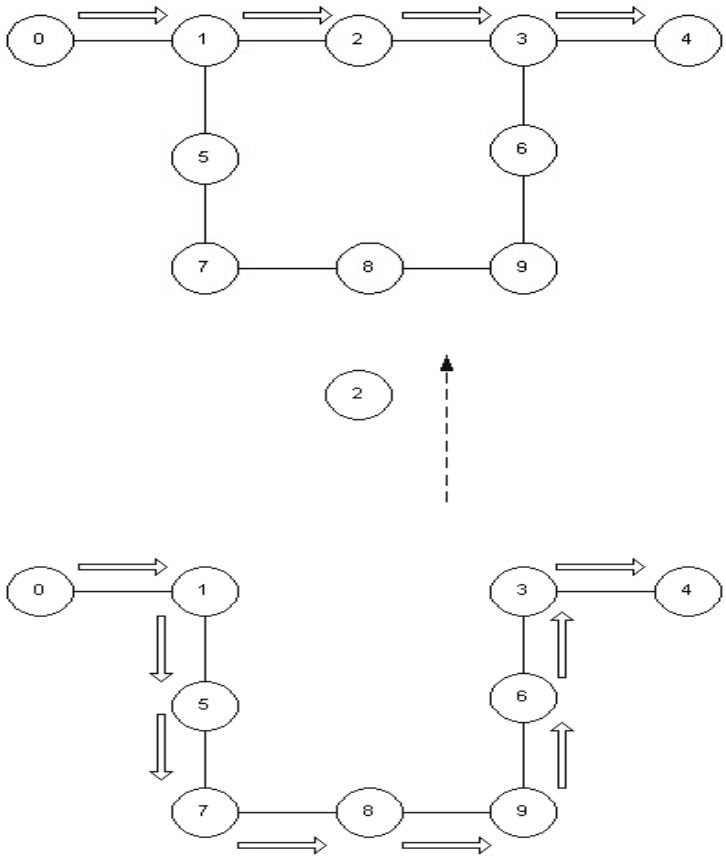

Figure4.3. Simple Topology to Verify the VRIS

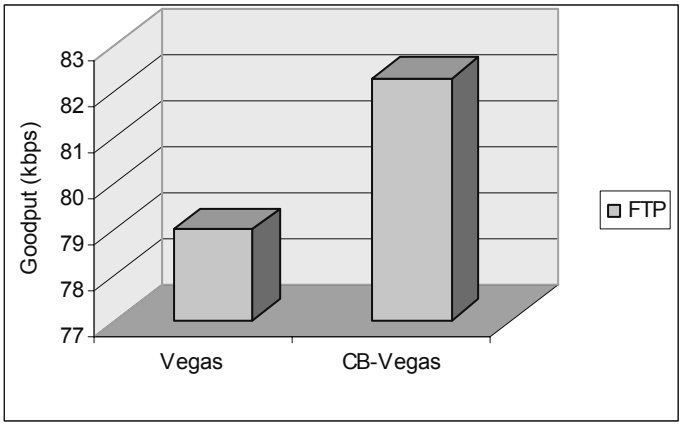

Figure4.4. Goodput Improvement for the Special Topology

In the second step, we apply VRIS in the more general mobile scenarios, which are the major in MANETs. We still use the random way-point model. 20 nodes move in a rectangle area of 1500*500 randomly. The max speed of the nodes is varied as 10, 20 and 30m/s and the node pause time between two adjacent movements is chosen as 0. Other parameters are the same in Table 3.1 expect the adjacent node distance. We randomly choose a source destination node pair to setup a TCP connection. As stated above, in order to minimize any arbitrary randomness, we use 30 different seeds to simulate 30 different random node movement scenarios for the connection. For each scenario, the simulation is run for 150s. The throughput (30 run average) improvement is obvious. We put two of the simulation results below. Figure 4.5 is for node pair (0, 1) and Figure 4.6 is for node pair (5, 15). We randomly choose other node pairs and get similar results. For these scenarios, we also give the results with Enhanced VRIS and LIR. We can see due to the mechanism to decide the bandwidth utilization, Enhanced VRIS synthesize the advantages of both FW-Vegas and CB-Vegas. The throughput is further improved. Contrarily, LIR cannot adapt to the mobile scenarios well.

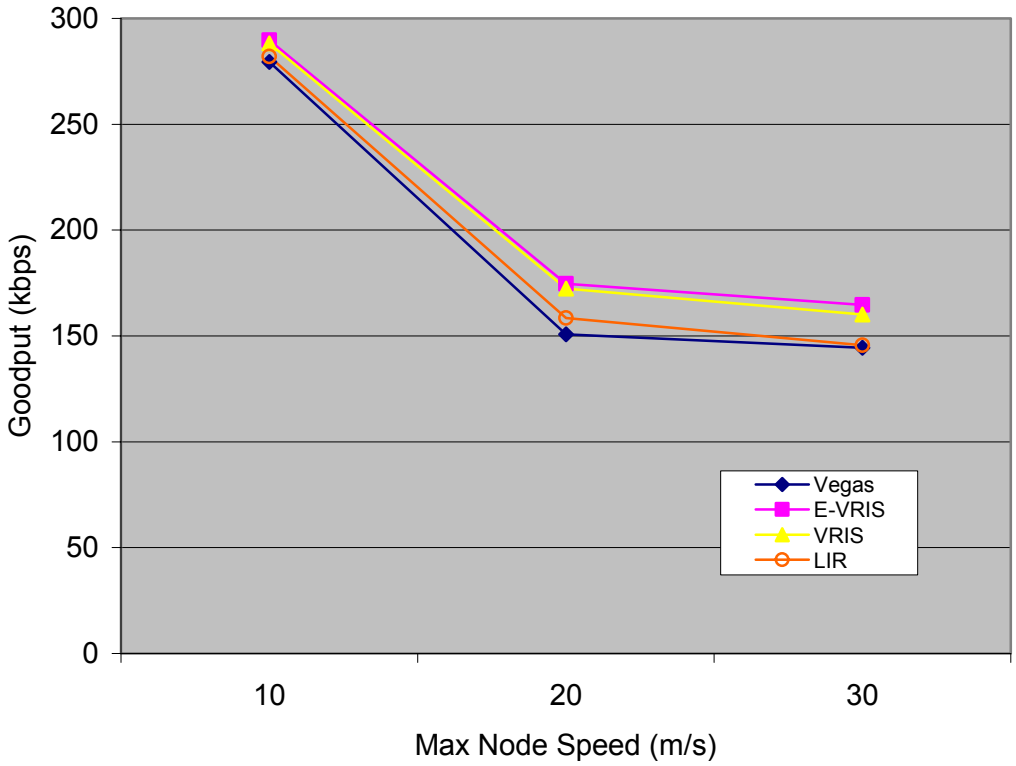

Figure 4.5 Goodput Improvement for Node Pair (0, 1)

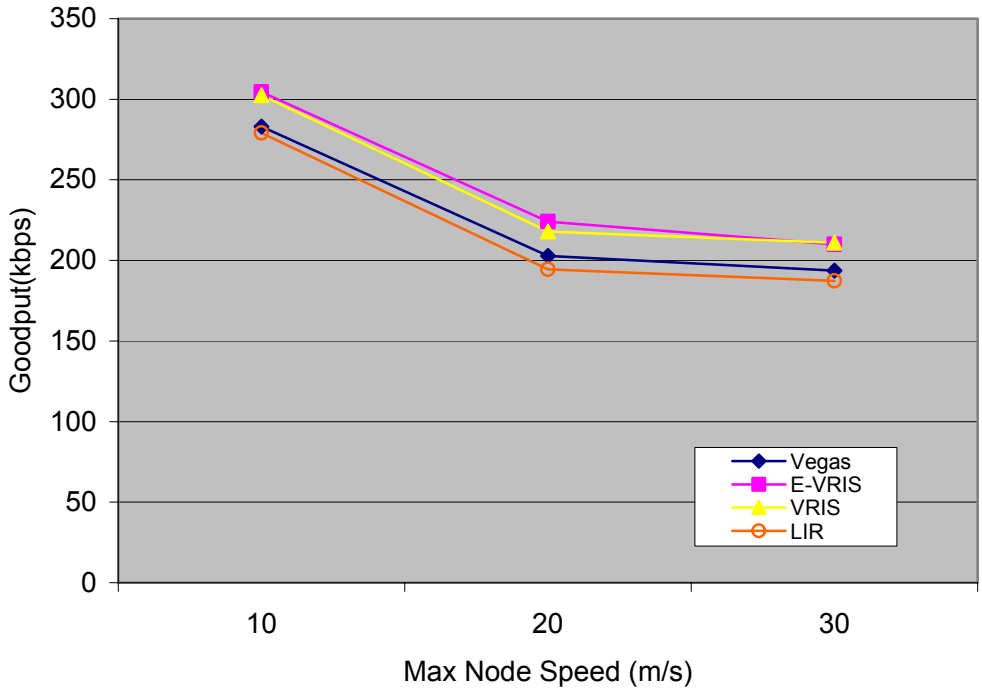

Figure 4.6 Goodput Improvement for Node Pair (5, 15)

In the third step, we try to vary the node pause time between the two adjacent random movements. Other parameters are the same as in step 2. We give the simulation results for pause time as 2s and max node speed as 10s and 20s in Figure 4.7 and Figure 4.8 respectively.

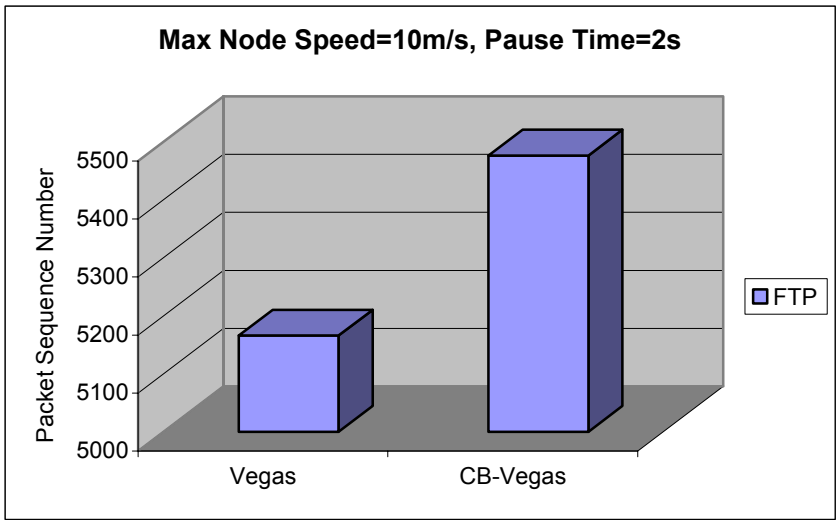

Figure 4.7 Goodput Improvement for the scenario of Max Speed=10m/s, Pause Time=2s, node pair (5, 15)

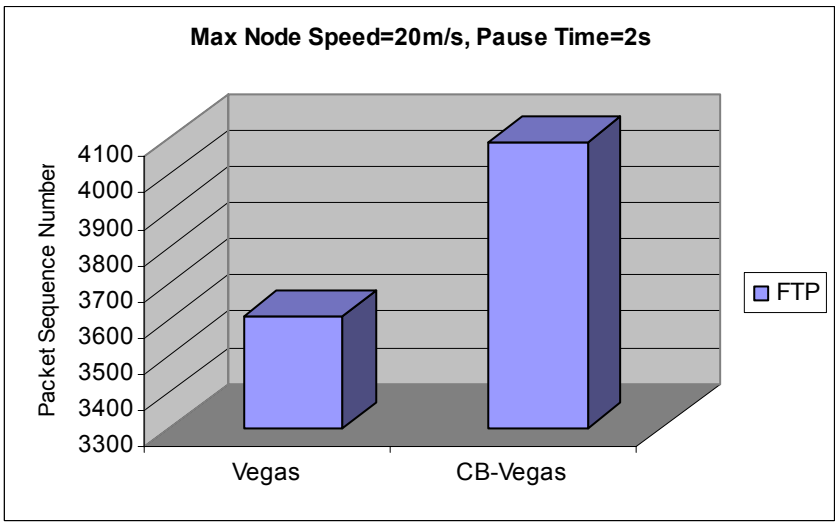

Figure 4.8 Goodput Improvement for the scenario of Max Speed=20m/s, Pause Time=2s, node pair (5, 15)

Summary:

VRIS works well with the DSR routing protocol. It can effectively improve the TCP-Vegas throughput. It catches hold of the most obvious characteristic of MANET, constant route changes, and makes use of the interaction between TCP-Vegas and underlying routing protocol. Vegas throughput achieves improvement by 10% on average and up to 223.01%. We can also find that the node pause time factor doesn't impact the VRIS much, it can also work well. What is more, As VRIS does not change the conservative characteristic of Vegas, the average packet delay and delay jitter are almost unaffected.

4.3.2.2 VRIS with AODV

We try to simulate our VRIS with AODV as the routing protocol for the mobile scenarios. The simulation parameters settings are the same with that of DSR.

First, we use the same simulation settings as in the second step of DSR: 20 nodes move in a rectangle area of 1500*500. We vary the max speed of the nodes as 10, 20 and 30m/s. The pause time is set as 0. We randomly choose the source destination node pairs. Node pair (5, 15) is chosen for illustration. The simulation results are based on the average of 30 runs. Each run is for 150s. The simulation results are as shown in Figure 4.9.

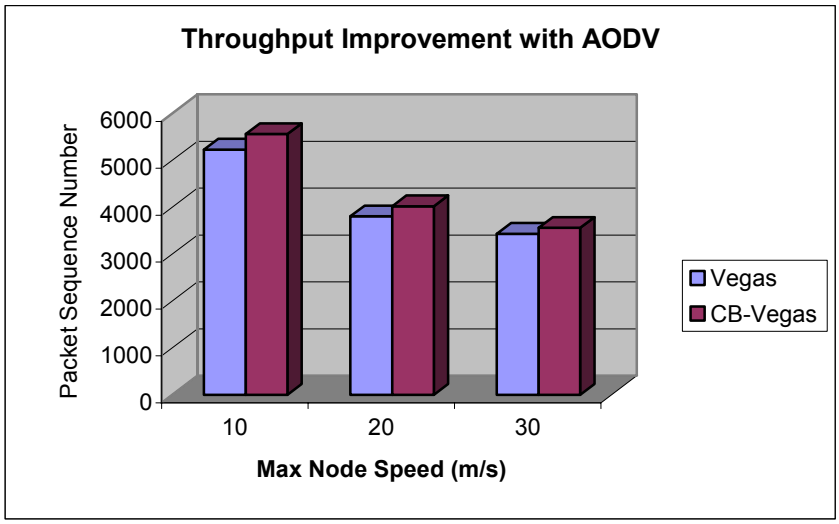

Figure 4.9 Goodput Improvements with AODV

Then, we also try to vary the node pause time between the two adjacent random movements. We set the node pause time between two adjacent movements as 2s. The simulation results as shown in Figure 4.10 and Figure 4.11 are for the scenarios of max node speed as 10m/s and 20m/s respectively. The node pair is still chosen as (5, 15).

For VRIS with AODV, the Vegas throughput can also be effectively improved with the average packet delay and delay jitter almost unaffected.

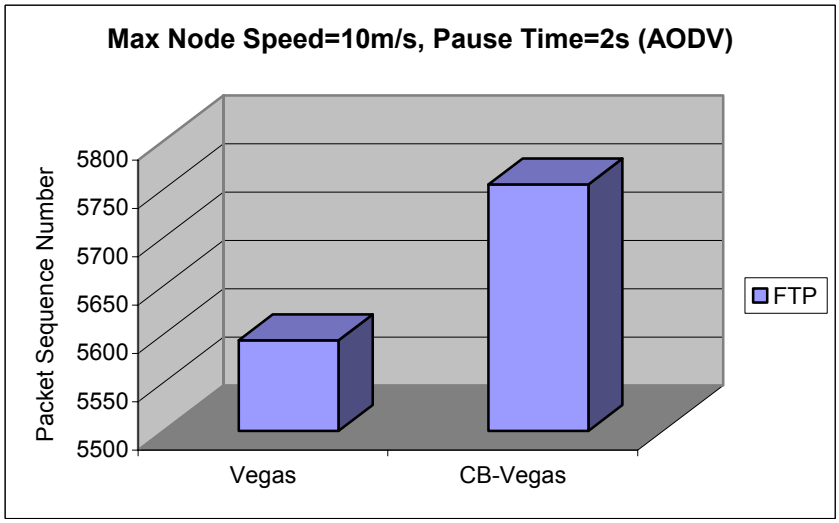

Figure 4.10 Goodput Improvement for the scenario of Max Speed=10m/s, Pause Time=2s, node pair (5, 15), AODV

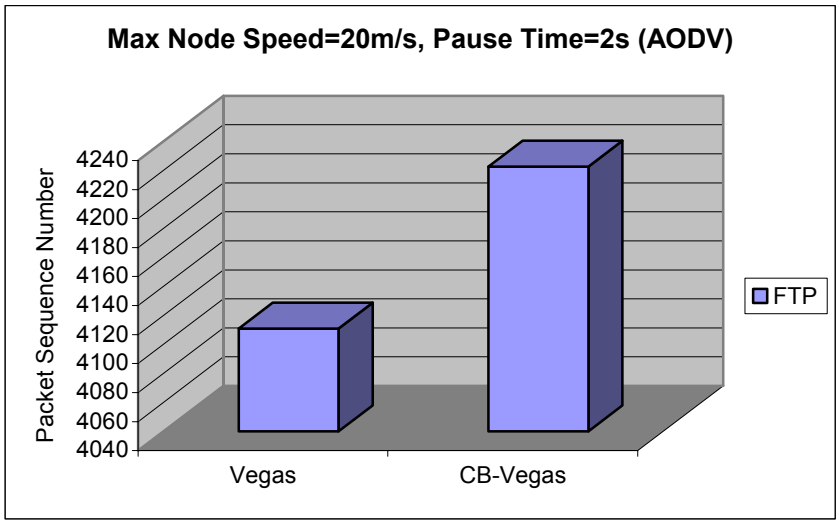

Figure 4.11 Goodput Improvement for the scenario of Max Speed=20m/s, Pause Time=2s, node pair (5, 15), AODV

4.3.2.3 Comparison of DSR and AODV Results

Then, we try to compare the DSR and AODV results.

We found that on average, AODV achieves better performance than DSR with the original standard TCP-Vegas. This is probably because that there is a constant overhead in DSR packets since in DSR implementation the packets contain the full route to the destination in their packet headers which result in a constant byte overhead. Moreover, AODV has better route cache management mechanism. DSR is making aggressive use of caching and it lacks any mechanism to expire the stale routes or to determine the freshness of routes when multiple choices are available. On the contrary, for AODV, route cache management is done through cache entry timeout. It ensures that only routes that are recently used are maintained in the route cache. This prevents the problem of a stale entry in the route cache. The above factors decide the AODV performs better than DSR.

However, the VRIS works better with DSR than with AODV. VRIS can improve the Vegas throughput by 7% on average with AODV compared with the 10% with DSR. The reason lies in that it is more accurate and timely to find the route hop number information from the packet header than from the routing table.

4.3.3 Summary

In summary, VRIS is quite simple but efficient. It can effectively improve the Vegas throughput with both DSR and AODV as routing protocols with the average packet delay and delay jitter almost unaffected.

4.4 Conclusion

We have proposed to apply Vegas in MANETs since we find it achieves better performance than Reno in MANETs by simulation. Its conservative window updating mechanism adapts to the MANETs better.

In this chapter we have proposed the Vegas Routing Interaction Strategy to solve the intrinsic re-routing problem of Vegas to further improve the Vegas throughput in MANETs with the small average packet delay and delay jitter almost unaffected.

With the re-routing problem solved by VRIS and together with other enhancement strategies, Vegas will be a better choice for MANETs.

Chapter 5

Adaptively Setting Congestion Window Limit (ASCWL) Strategy

5.1 Introduction

In Chapter 3 and 4, we have proposed to apply TCP-Vegas in MANETs and also worked out the VRIS to solve the re-routing problem in the mobile scenarios. In this chapter, we will study the unfairness problem. We consider the static scenarios because it is hard to define fairness in mobile scenarios.

There is severe unfairness problem or even flow starvation when applying TCP in MANETs. The problem is triggered by the underlying MAC protocol. In addition, the TCP characteristics and its poor interaction with the MAC layer further exacerbate the situation.

Previous works on the unfairness problem have mainly focused on identifying the factors that cause the problem [18] [19] [20]. Some proposed MAC layer scheduling strategies [23] [24] can alleviate the problem, but their implementation needs the MAC

protocol modification. One effective method to solve the problem is the Neighborhood RED (NRED) strategy proposed by K. Xu and Gerla et al. [17], which is implemented on the network layer. However, the implementation of NRED is complicated. Each node in the network maintains its own distributed queue, called neighborhood queue. The neighborhood queue needs to aggregate the node's local queue and the upstream and downstream queues of its 1-hop neighbors. Hence, we try to propose some method that can be implemented more easily. We propose to solve the TCP unfairness problem in the transport layer, without the lower layer modification to ensure its easy implementation. The ATP strategy [25], which is reported to solve many TCP problems in MANETs, is also transport layer based. But it is a new protocol that is not interoperable with TCP.

We need to make our strategy TCP-based. More specifically, we manage to control the TCP congestion window limit (CWL) to reduce the negative impact of TCP over MAC and improve the fairness. We get the relationship between the hop number and the optimal CWL achieving the best throughput via simulation. We also find that dynamically setting CWLs based on the optimal CWLs can effectively improve the fairness. Consequently, we propose the Adaptively Setting Congestion Window Limit (ASCWL) strategy which uses the hop number to set CWLs for the contending TCP connections as a means of solving the unfairness problem. As the solving methods of the unfairness problem are rare, the ASCWL strategy will be implemented with both Vegas and Reno. Simulation results (see section 5.5 for reference) show that our strategy can improve the fairness index by up to 89.21%. Furthermore, the overall throughput usually increases as well.

In the following sections, we will first give some related works on Congestion Window Limit (CWL) setting. Then by simulation, we offer the Optimal CWL vs.

hop number relationship results for both TCP-Reno and TCP-Vegas. Next, based on the optimal CWL values, we will propose the Adaptively Setting Congestion Window Limit (ASCWL) strategy and present the simulation results of ASCWL implementation.

5.2 Related Works on Congestion Window Limit (CWL) Setting

There are many reported works on how the CWL can be set to improve TCP throughput. It has been observed that setting TCP-Reno's CWL large can adversely degrade its throughput performance [26] [16].By considering the spatial reuse property of 802.11 MAC layer protocol in a chain topology, Li et al. [27] and Fu et al. [16] discovered that the optimal value for CWL is 1/4 of the chain length. Chen et al. [15] further proposed the round-trip hop-count (RTHC) and bandwidth-delay product (BDP) concepts. Taking the spatial reuse and the collision of the DATA and ACK into consideration, they proved that BDP of a chain (IEEE 802.11 MAC) is bounded by k times RTHC (1/8<k<1/4). Chen et al. [15] and Fu et al. [16] use a similar topology as Figure 5.1 to illustrate the spatial reuse issue.

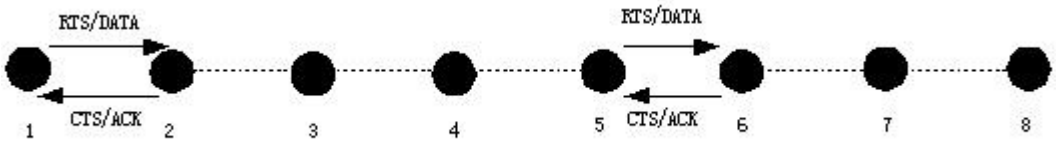

Figure 5.1 Illustration of the maximum spatial reuse and the hop number relationship for IEEE 802.11

As the transmission range is 250m (the adjacent nodes are 200m apart) and the interference range is 550m, we can see only 4 hops away nodes, such as node 1 and node 5, can initiate transmission concurrently without interfering each other. Since IEEE 802.11 exhibits a similar feature to the stop-and-wait protocol, the pipe size over

each hop is 1 packet and so the best channel utilization is obtained when there are h/4 packets in flight.

5.3 Optimal CWL vs. Hop Number Relationship for both TCP-Reno and TCP-Vegas

We use the 25-node chain topology similar as in Figure 5.1 and the parameter settings as in Table 5.1 to find the relationship via ns-2 simulations.

Table 5.1 Parameters Settings

Wireless Channel Raw Capacity	2M bps
Radio Transmission Range	250 meters
Carrier Sense Range	550 meters
Adjacent Node Distance	200 meters
MAC	IEEE 802.11
Routing Protocol	DSR
Default TCP Packet Size	1000 bytes
interface queue limit	50 packets

We have managed to find the optimal CWL vs. hop number relationship for both TCP-Reno and TCP-Vegas (see as in Figure 5.2 and Figure 5.3).

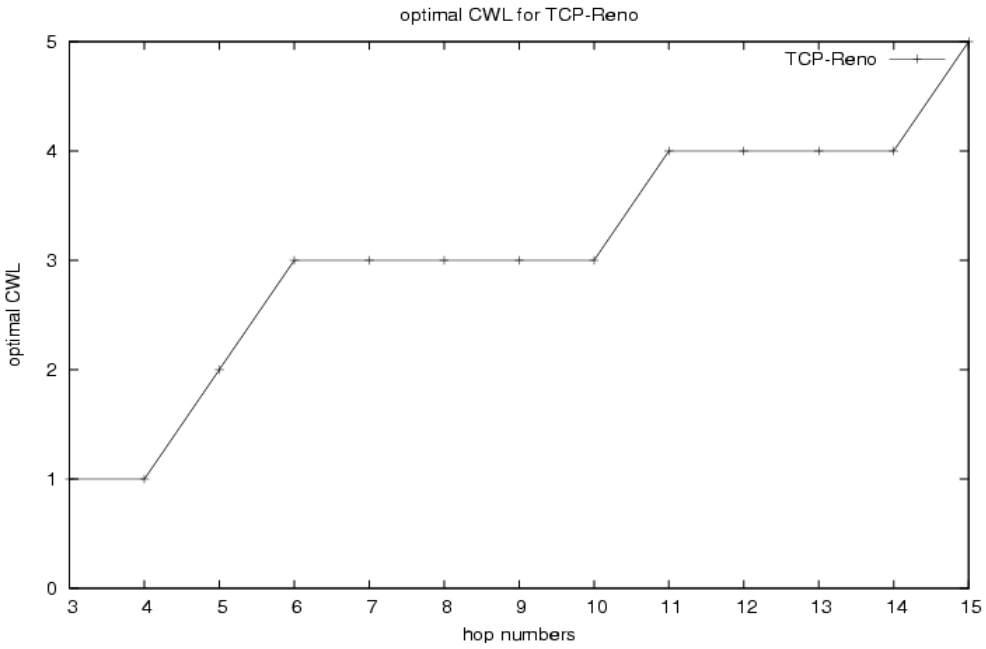

Figure 5.2: optimal CWL vs. hop number (Reno)

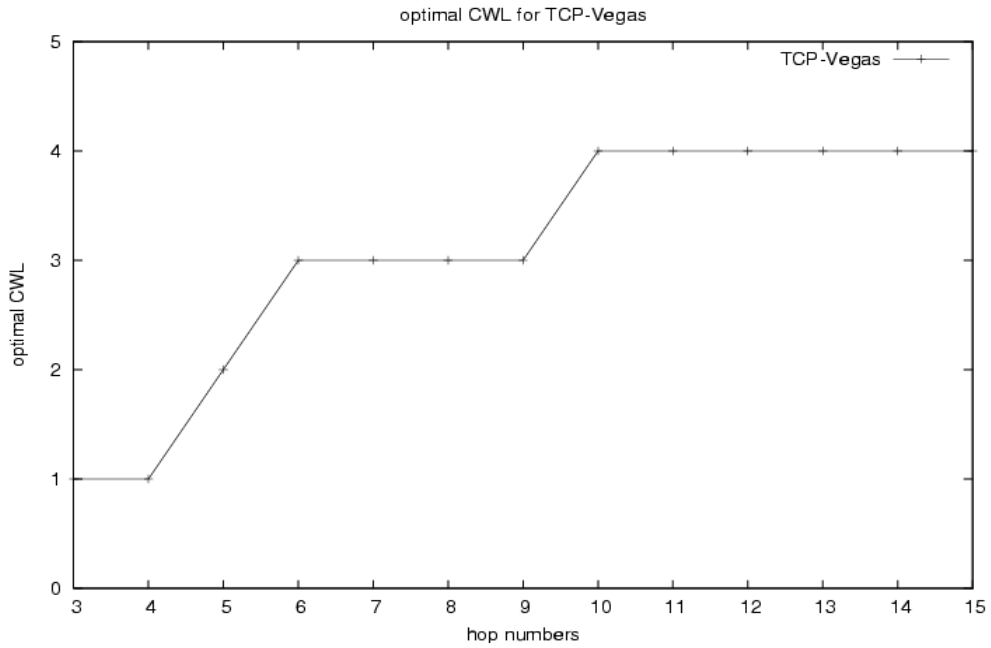

Figure 5.3: optimal CWL vs. hop number (Vegas)

For TCP-Reno, we find similar optimal CWL results as in [15] and [16]. But when we setup 1 or 2 hop TCP connections, as the source can “hear” the destination and the

hidden terminal problem doesn't exist, packet collision is avoided and loss is rare. The throughput will increase if we set larger CWL. Hence, we can not find the "so-called" optimal CWL.

Then we use Vegas to perform the simulation. As the spatial reuse factor is decided by the IEEE 802.11 protocol, Vegas shows similar hop number and optimal CWL relationship with that of Reno. But we know that Vegas is different from Reno with respect to updating sending window *cwnd*. Vegas uses the fine-grained measurement of *RTTs* to control its window size. It always avoids packet loss and very conservative when increasing *cwnd*. So Vegas tends to make its cwnd small to use the bandwidth. Thus usually if we set CWL larger, Vegas will still achieve the same throughput. Consequently, the optimal CWL for Vegas refers to the smallest CWL at which Vegas achieves highest throughput.

We also find that when the hop number is larger, the optimal CWL of Vegas tends to converge to the value of 4. This is because 4 is the least value to fulfill fast retransmission and retransmission in time is an important factor for Vegas' success.

Correspondingly for Reno, if we set CWL above the optimal value, the throughput will first decrease and then flatten out. This is mainly because Reno is very aggressive when increasing its *cwnd*. So if we set larger CWL than the optimal value, Reno will inject more packets to reach out the network's capacity. Consequently, the network congestion will cause packet loss and throughput degradation.

5.4 ASCWL strategy

The negative impact of TCP over MAC can be effectively alleviated by dynamically setting the CWLs based on the optimal CWL. Thus, we propose the Adaptively Setting Congestion Window Limit (ASCWL) strategy. We monitor the underlying routing agent to get the hop number which is then used to decide the optimal CWLs for the respective connections. If the TCP connection is less than 3 hops, as the source can "hear" the destination and the throughput will increase if we set CWL larger, the so-called optimal CWL doesn't exist. We will set the optimal CWL for Reno as 1 since a connection greedily increase its congestion window (*cwnd*) causes unfairness. Correspondingly, Vegas is conservative when increasing its *cwnd*, so the optimal CWL is set as 4, the least value for fast retransmission. Then we try to compare the route hop numbers or use the "beacon signals", which are used to provide connectivity information on the network topology, between the nodes to decide border connections so that we know if some connections are disadvantaged. For example, if two TCP connections of different lengths are competing for network resources, the larger hop number can be used as an indication that the longer connection is disadvantaged. Beacon signals can also indicate which are the more disadvantaged inner connections. If the border connection is also the longer connection, we consider this connection disadvantaged. The fundamental idea of ASCWL is to set larger CWLs for the relatively disadvantaged TCP connections. We illustrate our concept using two connections and extend the same rules for the case of multiple connections.

The algorithm details are as follows.

If (one connection is disadvantaged)

If (Optimal CWL $_{disadvantaged}$ >Optimal CWL $_{advantaged}$)

Set the CWLs to their optimal CWLs respectively

Else

If (Optimal CWL >1)

CWL $_{disadvantaged}$ = Optimal CWL

CWL $_{advantaged}$ = Optimal CWL – 1

If (Optimal CWL ==1)

CWL $_{disadvantaged}$ = Optimal CWL + 1

CWL $_{advantaged}$ = Optimal CWL

Else

Set the CWLs to their optimal CWLs respectively

As it is extremely challenging to define fairness in mobile scenarios, we implement our strategy on static topologies such as the chain topology in [18], the cross topology and the grid topology in [17], where NRED is implemented. We use TCP-Reno and TCP-Vegas to verify our strategy. For the grid topology in [17], we also use TCP-NewReno to compare the results with that of NRED. We use the fairness index defined in [28] for a set of throughputs (x_1, x_2 … x_n),

$$f(x_1, x_2, \ldots, x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} x_i^2} \tag{5.1}$$

In the following, we show some simulation results by NS-2 simulator. Reno+ stands for Reno with ASCWL and Vegas+ stands for Vegas with ASCWL. We run FTP applications on the TCP connections with all FTPs starting and ending at the same time. The parameter settings are the same as shown in Table 5.1 unless otherwise specified.

5.5 Simulation Results for ASCWL

5.5.1 Chain Topology

We use the 25-node chain topology similar as in Figure 5.1. As chain topology is the basis for the complicated topology. We will study it in details. First, we avoid the border effect to take consideration of the factor of hop number. Then we will consider the border effect. That is, we will classify the unfairness problem into 3 types as following. FTP applications are run on the TCP connections and all FTP applications start at 10s and end at 150s.

- Same Hop Length TCP connections without Border Effect

  We set the two TCP connections the same length and make the destination of the first TCP connection the source the second one. We vary the length of the TCP connection as 2, 4, 6, 8 in our simulation. The illustration is as in the Figure 5.4.

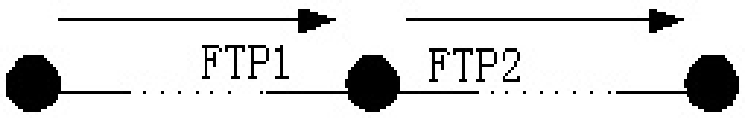

Figure 5.4 Same Hop Number without Border Effect Illustration

The simulation results are listed from Figure5.5 to Figure 5.8. The fairness indexes calculated using (5.1) are summarized in Table 5.2. For TCP-Reno, as the CWL setting can change the aggressive window increasing operation, the average packet delay and delay jitter will be evidently reduced. The packet delay distributions for the 4 hop cases of the Reno and Reno+ are as Figure 5.9 and Figure 5.10. We would not put the average packet delay trace for the other simulation scenarios as they present the similar trends.

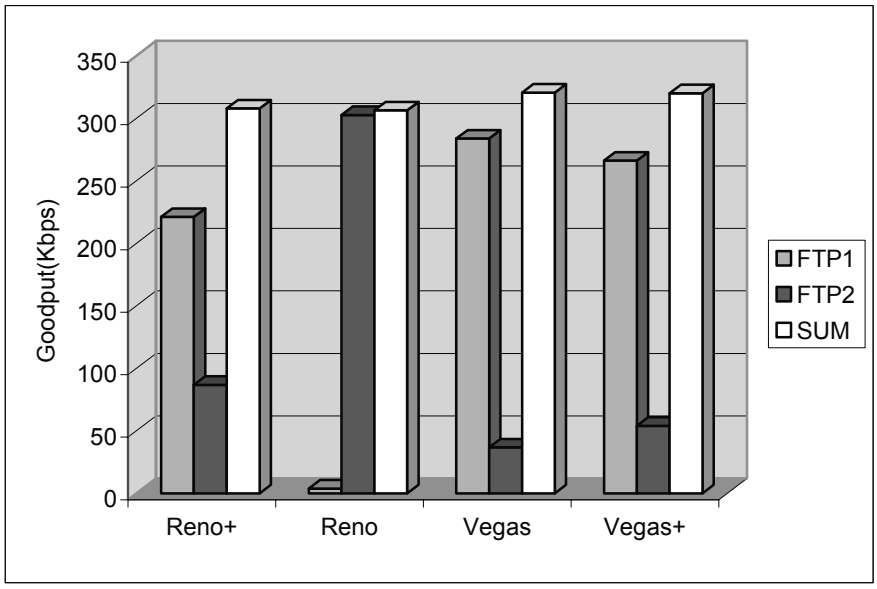

Figure 5.5: TCP Goodputs for connections with same hop length
In the chain topology (hop number = 2)

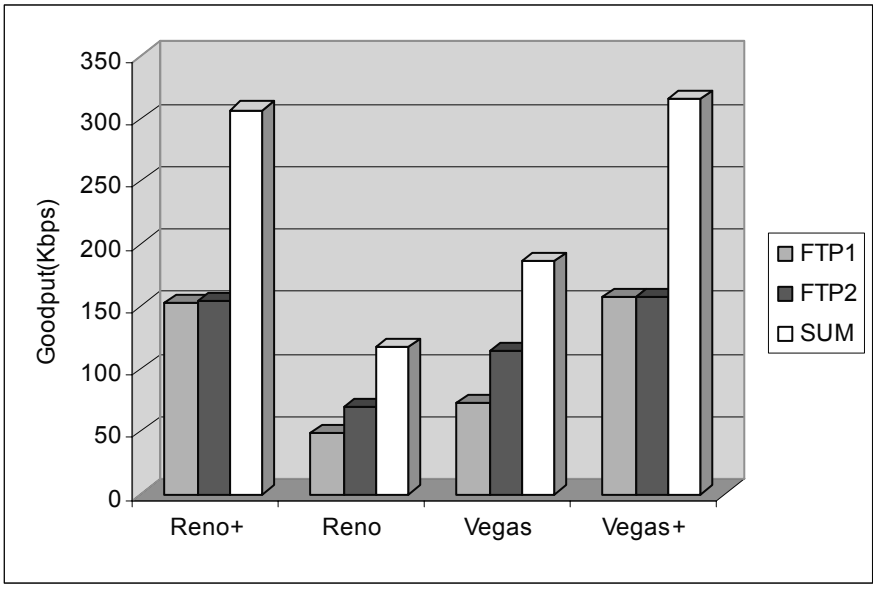

Figure 5.6: TCP Goodputs for connections with same hop length
In the chain topology (hop number = 4)

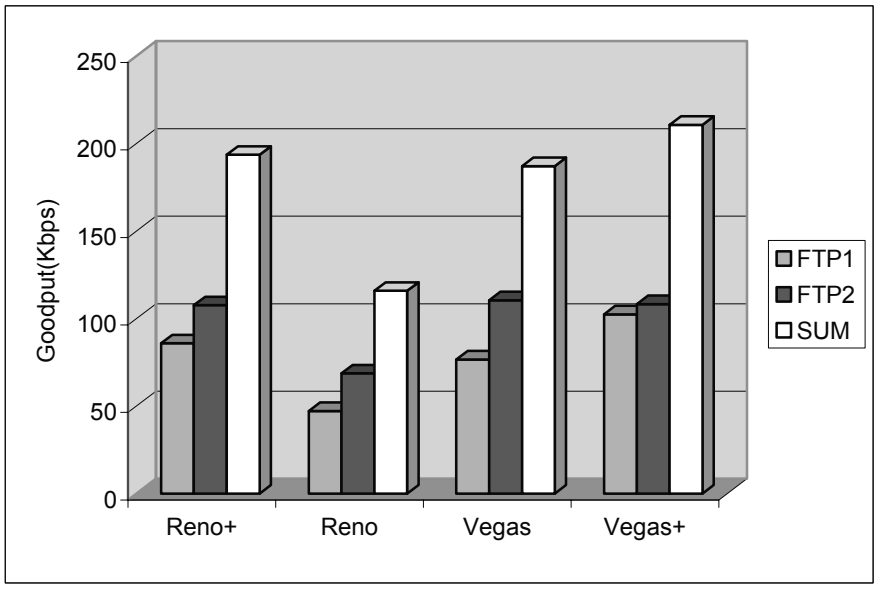

Figure 5.7: TCP Goodputs for connections with same hop length
In the chain topology (hop number = 6)

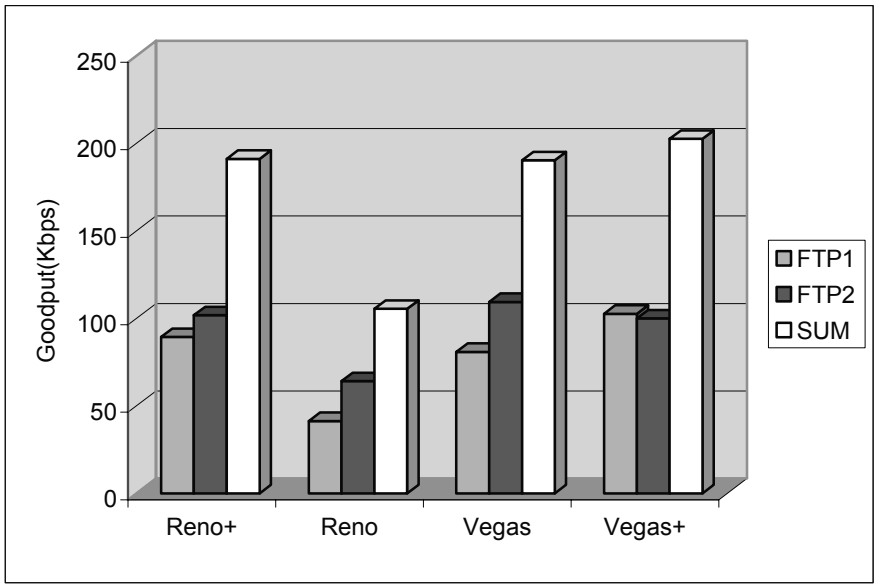

Figure 5.8: TCP Goodputs for connections with same hop length
In the chain topology (hop number = 8)

Table 5.2: Fairness indexes for TCP connections with same hop length
In the chain topology

	hop number 2	hop number 4	hop number 6	hop number 8
Reno	0.5125	0.9690	0.9669	0.9552
Reno+	0.8401	0.9999	0.9875	0.9957
Vegas+	0.6949	1	0.9992	0.9998
Vegas	0.6273	0.9516	0.9684	0.9780

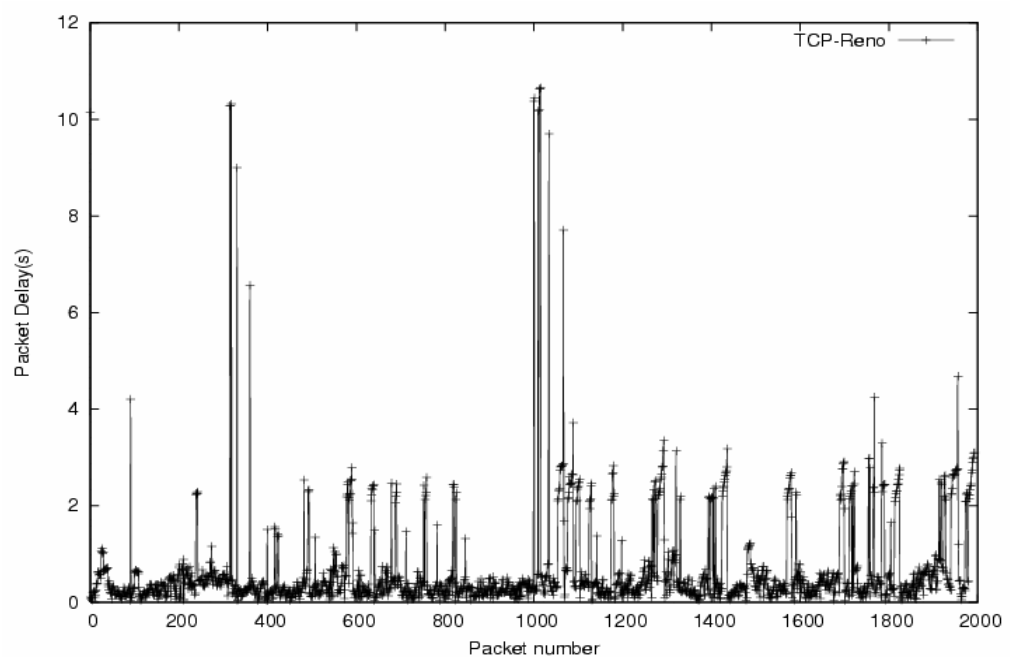

Figure 5.9 Reno packet delay distribution (4 hops) for connections
With same hop length in the chain topology

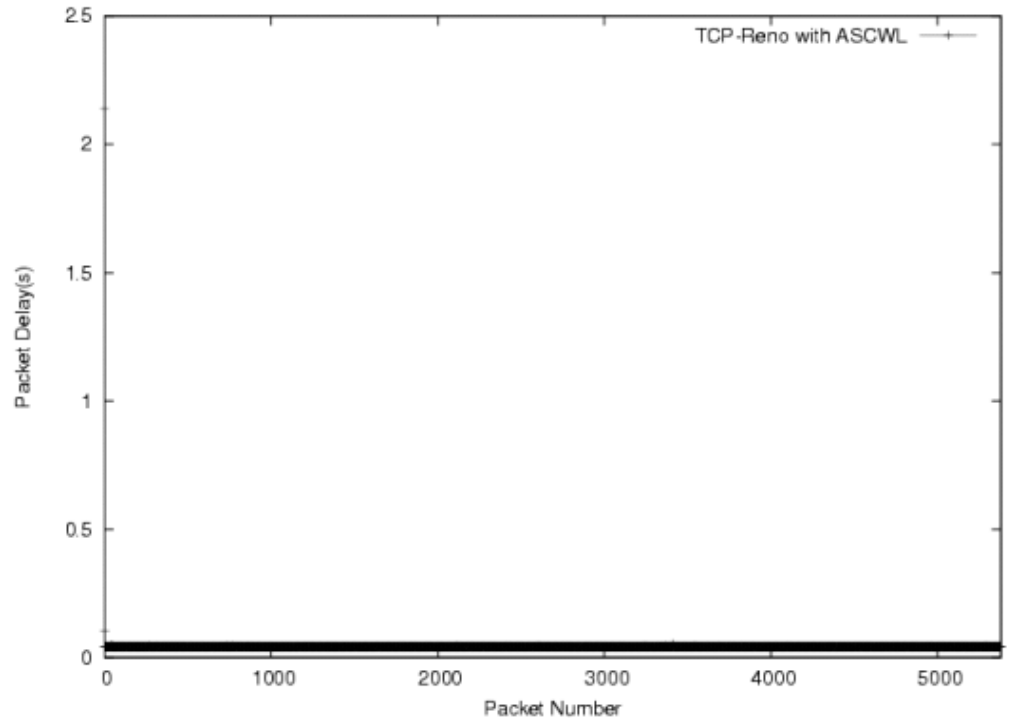

Figure 5.10 Reno+ packet delay distribution (4 hops) for connections
With same hop length in the chain topology

- Different Hop Lengths TCP connections without Border Effect

We setup connections that have resource contenting with different hop length in the chain topology. The illustration is as in Figure 5.11.

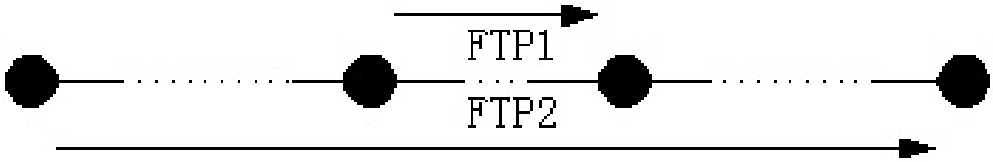

Figure 5.11 Different Hop Numbers without Border Effect illustration

Case A: two connections

We setup two connections with different hop length in the chain topology. We list the simulation results of 3 scenarios. In scenario 1, we set a TCP connection from node3 to node10 and another from node1 to node16. The simulation result is in Figure 5.12. Similarly, Figure 5.13 is for the scenario one TCP from node 3 to node7 the other from node1 to node13 and Figure 5.14 is for the scenario that one TCP from node1 to node9 the other from node3 to node7.

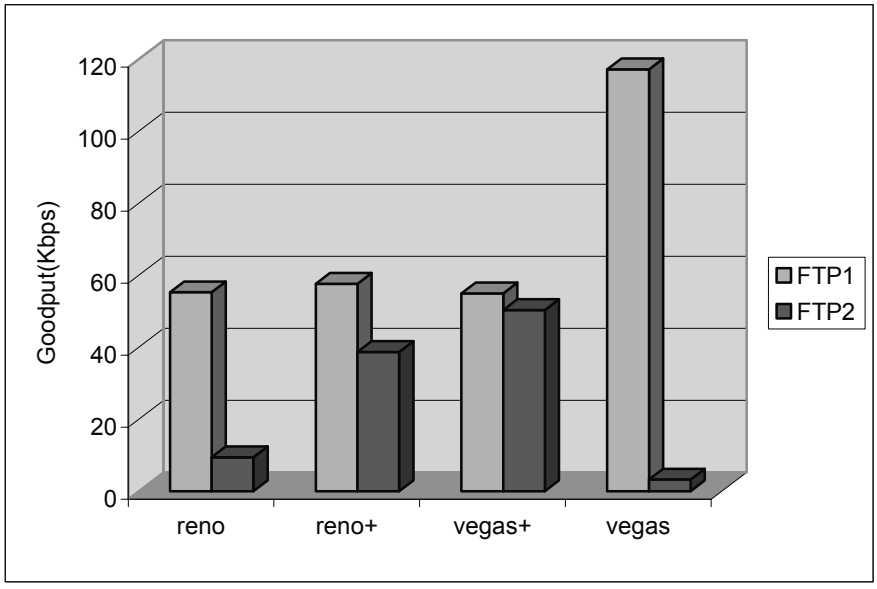

Figure 5.12: TCP Goodputs for connections with TCP1 7 hops, TCP2 15 hops in the chain topology

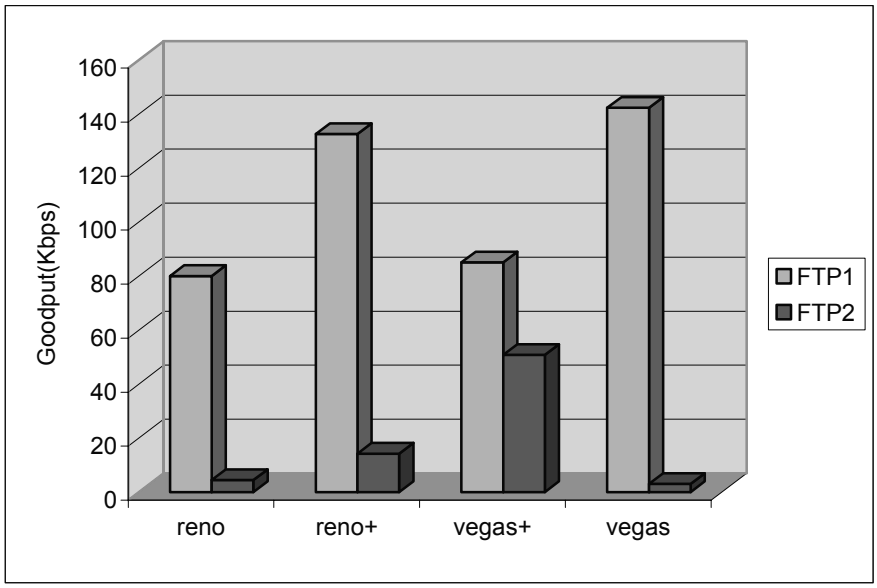

Figure 5.13: TCP Goodputs for connections with TCP1 4 hops, TCP2 12 hops in the chain topology

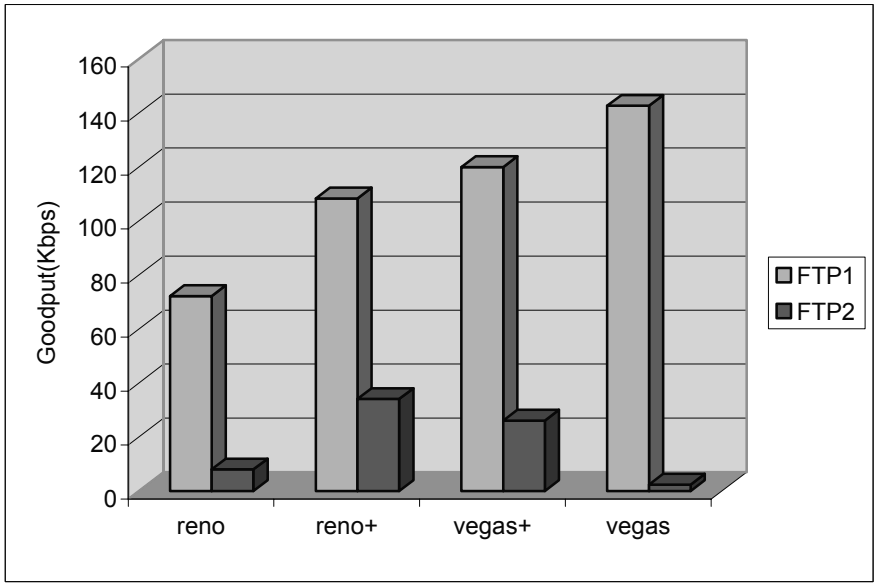

Figure 5.14 TCP Goodputs for connections with TCP1 7 hops, TCP2 15 hops in the chain topology

Case B: multiple connections

We try to study the situation of multiple connections. A scenario of 3 connections is listed here: one TCP is from node1 to node16, another from node2 to node9 and the third from node3 to node7. The simulation result is shown in Figure 5.15.

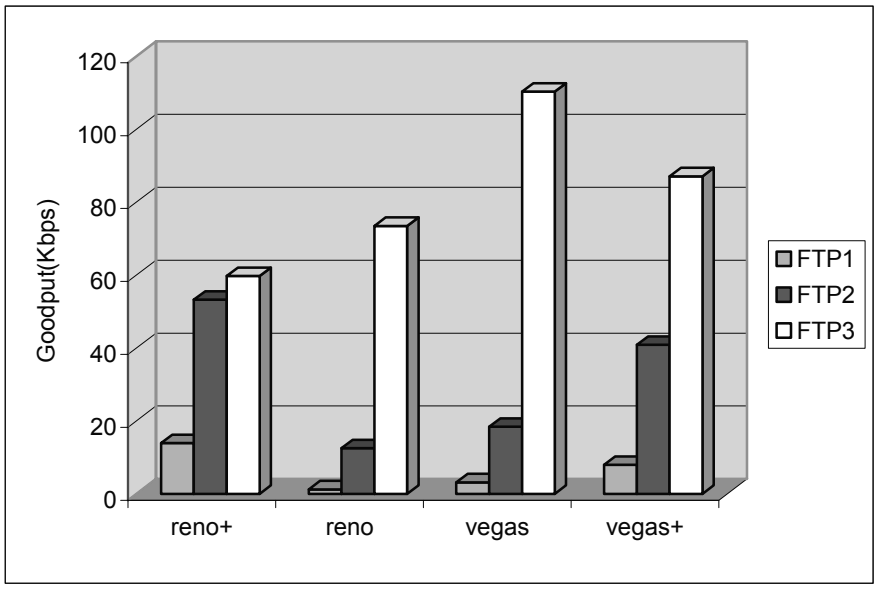

Figure 5.15 TCP Goodputs for connections with TCP1 15 hops, TCP2 7 hops, TCP3 4 hops in the chain topology

The fairness indexes for the above four scenarios are shown as in Table 5.3. The throughput improvement for the disadvantaged connections is in Table 5.4. Specifically, the ratio means the throughput improvement ratio for the TCP connection(s) that has a disadvantaged position when contending for the bandwidth.

Table 5.3: Fairness indexes for TCP connections with different hop lengths In the chain topology

	hop number 15, 7	hop number 12, 4	hop number 8, 4	hop number 15, 7, 4
Reno	0.6651	0.5548	0.6090	0.4562
Reno+	0.9629	0.6061	0.7861	0.8136
Vegas+	0.9980	0.9401	0.7072	0.6608
Vegas	0.5275	0.5209	0.5162	0.4630

Table 5.4: Goodput improvement for the disadvantaged connections for TCP connections with different hop lengths in the chain topology

	hop number 15, 7	hop number 12, 4	hop number 8, 4	hop number 15, 7, 4
Reno+/				11.3478 (most-disadvantaged)
Reno	4.1221	3.2363	4.2709	4.2558 (disadvantaged)
Vegas+/				2.5042 (most-disadvantaged)
Vegas	15.5867	17	11.2183	2.2249 (disadvantaged)

- Border Effect

In the chain topology, the connections from or to the border nodes have about half of the collision chances of the inner connections. So here we can make use of our ASCWL strategy to improve the throughput of the inner connections so as to improve the TCP fairness.

In our simulation, we try to set up 5 consecutive 1 hop connections as in Figure 5.16. The destination of the former connection is the source of the latter.

Figure 5.16 A 6-node chain topology to illustrate the Border Effect

The simulation result is shown in Figure 5.17 and the fairness indexes are shown in Table 5.5.

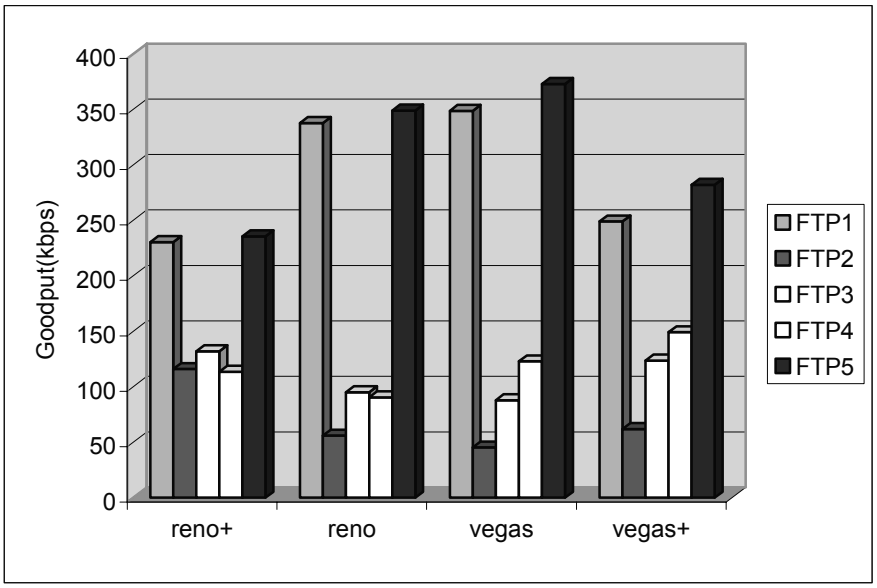

Figure 5.17 TCP Goodputs for connections with border effect
In the chain topology

Table 5.5: Fairness indexes for the TCP connections with border effect

Reno	Reno+	Vegas+	Vegas
0.6727	0.8988	0.8196	0.6696

5.5.2 Cross Topology

We setup the two TCP connections as shown in Figure 5.18 and run FTP applications on them. The two FTPs start at 10s and end at 150s. We use the ASCWL to set the CWLs for both connections their optimal values.

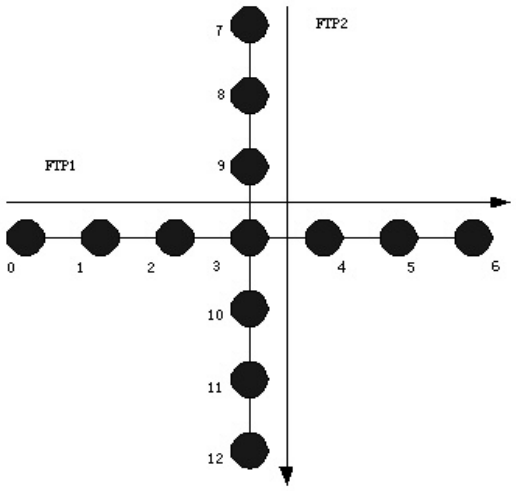

Figure 5.18 Cross Topology

The simulation result is shown in Figure 5.19 and fairness indexes are shown in Table 5.6.

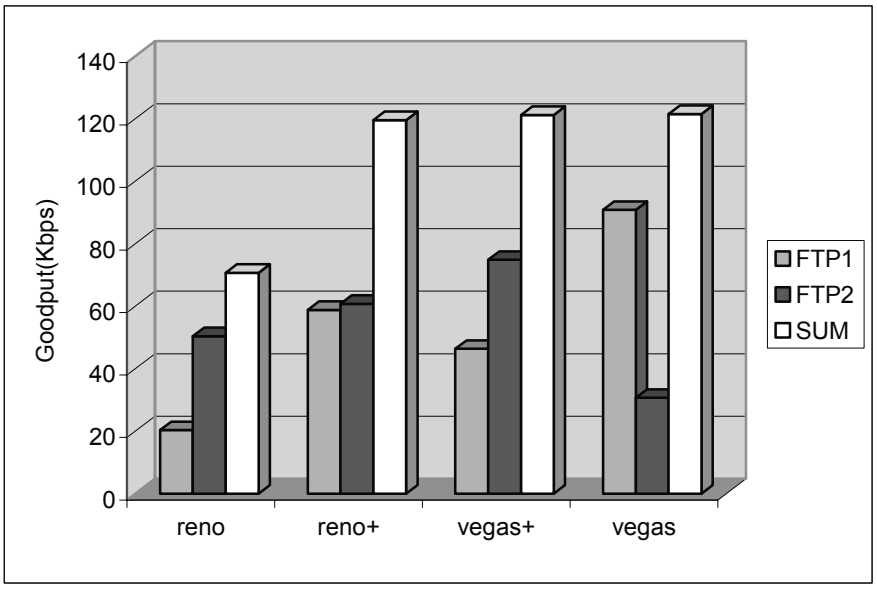

Figure 5.19 TCP Goodputs for connections in the cross topology

Table 5.6: Fairness indexes for the TCP connections in the cross topology

Reno	Reno+	Vegas+	Vegas
0.8469	0.9997	0.9477	0.8031

5.5.3 Grid Topology

As in Figure 5.20, we setup 6 TCP connections and run FTP applications on them from 10s to 150s. As we can see that the inner connection need to contend with both side connections and so disadvantaged. By ASCWL, the two inner connections will have larger CWLs.

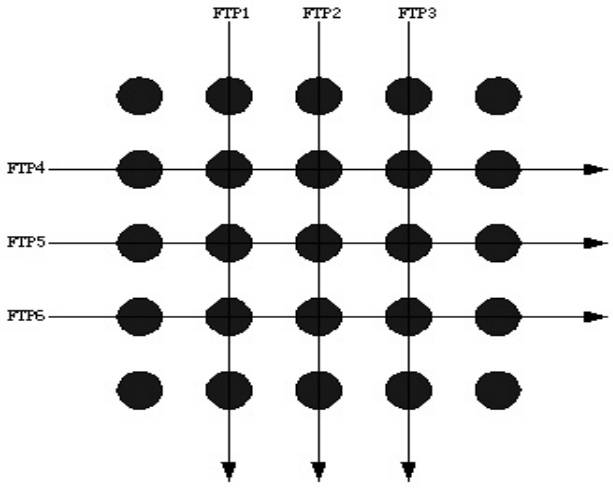

Figure 5.20 Grid Topology

The simulation result is in Figure 5.21 and fairness indexes are in Table 5.7.

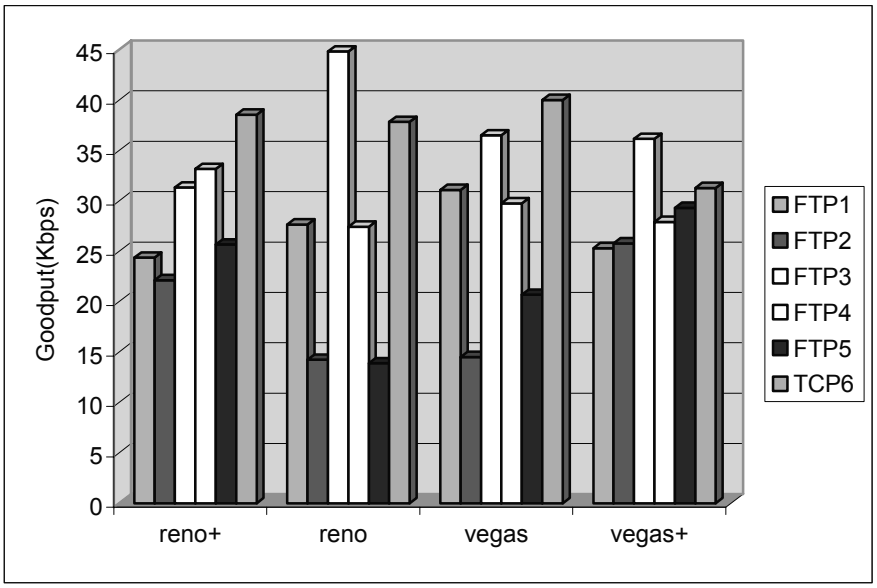

Figure 5.21 TCP Goodputs for connections in the grid topology

Table 5.7: Fairness indexes for the TCP connections in the grid topology

Reno	Reno+	Vegas+	Vegas
0.8565	0.9635	0.9844	0.9149

For this grid topology, we also simulate our strategy with TCP-NewReno and try to compare the results with the NRED strategy in [17]. The simulation parameters are adjusted according to the setting in [17]: The distance between the adjacent nodes is set to 250m and packet size is set to 512 Bytes. The interface queue size at each node is set to 66 packets. The TCP connections start at 10s and end at 130s. The results are listed in Table 5.8.

Table 5.8 ASCWL vs. NRED

	Fairness index	Overall throughput increase
ASCWL	0.9494	+23.6%
NRED	0.9367	-35.5%

5.5.4 Summary

With ASCWL implemented, the fairness indexes achieve obvious improvement. Moreover, the overall throughput is usually also improved. And the average packet delay is decreased. The main reason for the performance enhancement is that after CWL setting with ASCWL strategy, the TCP source would shoot much less packets into the network. Thus the disadvantaged TCP connections have more chance to catch up. Moreover, the packets of all the connections can be transmitted more effectively. Consequently, the overall throughput will increase and the average packet delay will decrease. As Vegas itself is conservative in increasing congestion window, the improvement extent is usually less compared with Reno.

5.6 Conclusion

Simulation results show that our strategy can effectively improve the fairness index of the contending TCP connections (by up to 89.21% and an average of 28.07%). In addition, the disadvantaged connections show dramatic throughput improvement (up to 17 times), and the average packet delay has also decreased, especially for Reno.

Chapter 6

Conclusions and Future Works

In this thesis, we have worked on the topic of improving TCP performance in MANETs. We propose to apply TCP-Vegas in MANETs. Via simulation, we have shown that TCP-Vegas can achieve better performance than TCP-Reno: higher throughput, less average packet delay and delay jitter. To solve the intrinsic re-routing problem of applying Vegas in the mobile MANETs environment, we propose the Vegas Routing Interaction Strategy (VRIS). VRIS can effectively improve the Vegas throughput with the average packet delay and delay jitter almost unaffected. We also study the unfairness problem in the static MANETs environment and propose the Adaptively Setting Congestion Window Limit (ASCWL) strategy. ASCWL can be implemented with both TCP-Vegas and TCP-Reno and with both it can effectively improve the fairness index.

6.1 Contributions

In this thesis, aiming at both improving the TCP throughput and reducing the average packet delay and delay jitter, we propose to apply TCP-Vegas in MANETs. This is unique compared with the former related research works.

The VRIS is based on the mobile MANETs characters and thus can effectively solve the re-routing problem to improve the Vegas throughput. Together with other strategies to differentiate the different packet loss and thus implement different operations, we can make Vegas a better choice for the MANETs.

The ASCWL strategy can effectively improve the fairness index in the static MANETs scenarios. It is TCP-based and easy to implement. The strategy is executed with both Vegas and Reno as the solving methods of the unfairness problem are rare. Moreover, the overall throughput is usually increased.

6.2 Future Work

VRIS only deals with the re-routing problem. To further improve the Vegas throughput, we may also need to differentiate the packet losses due to congestion and the losses due to other reasons so as to implement appropriate operations.

In our future work, we should also consider the more accurate *Base_RTT* updating condition to avoid the immature update that may happen occasionally if deciding only by the route hop length changing.

The disadvantaged MANETs environment can be improved by new routing algorithm or MAC layer optimization as well.

We will also try to find a more reasonable fairness definition in the mobile scenarios so as to implement ASCWL to improve fairness in mobile cases.

Bibliography

[1] M. S. Corson, J. Macker and G. H. Cirincione, “Internet-based mobile ad hoc networking,” IEEE Internet Computing 3 (4) (1999), pp 63-70

[2] L. S. Brakmo, S. W. O’Malley and L. L .Peterson, “TCP Vegas: New techniques for congestion detection and avoidance,” Proceedings. of ACM SIGCOMM’94, pp 24-35, October 1994

[3] U. Hengartner, J. Bolliger and Th. Gross, “TCP Vegas Revisited,” Proceedings of IEEE INFOCOM’00, pp.1546-1555, March 2000

[4] J. Mo, R. J. La, V. Anantharam and J. Walrand, “Analysis and Comparison of TCP Reno and Vegas,” Proceedings of IEEE INFOCOM’ 99, pp 1556-1563, March 1999

[5] K. N. Srijith, L. Jacob and A.L. Ananda, “TCP Vegas-A: Solving the Fairness and Rerouting Issues of TCP Vegas,” IEEE IPCCC 2003, pp 309-316, April 2003

[6] R. J. La, J. Walrand and V. Anantharam, “Issues in TCP Vegas,” available at http://www.eecs.berkeley.edu/~ananth/1999-2001/Richard/IssuesInTCPVegas.pdf, July 1998

[7] K.Chandran, S.Raghunathan, S.Venkatesan, and R.Prakash, “A feedback based scheme for improving TCP performance in Ad-Hoc wireless networks,” in Proc. of the

International Conference on Distributed Computing Systems (ICDCS'98), Amsterdam, Netherlands, May 1998

[8] G. Holland and N. Vaidya, "Analysis of TCP performance over mobile ad hoc networks," ACM Wireless Networks, vol. 8, no. 2, pp. 275-288, Mar. 2002

[9] B. Zhang, M.N. Shirazi and B. Komiyama, "An ELFN-Based TCP-Freeze Scheme Using the Route Information of Sender Node for Ad Hoc Networks," IEEE 10th Asia-Pacific Conference on Communication and 5th International Symposium on Muliti-Dimensional Mobile Communications, vol. 1, pp. 457-461, Aug. 2004

[10] D.Kim, C.Toh, and Y. Choi, "TCP-Bus: Improving TCP performance in wireless ad hoc networks," Journal of Communications and Networks, vol.3, no.2, pp. 175-186, Jun. 2001

[11] J. Liu and S. Singh, "ATCP: TCP for mobile ad hoc networks," IEEE journal on Selected Areas in Communications, vol.19, no.7, pp.1300-1315, Jul. 2001.

[12] T. Goff, N. Abu-Ghazaleh, D.Phatak, and R.Kahvecioglu, "Preemptive routing in ad hoc networks," in proc. of ACM Mobicom, Rome, Italy, 2001, pp.43-52

[13] Haejung Lim, Kaixin Xu, and Mario Gerla, "TCP Performance over Multipath Routing in Mobile Ad Hoc Networks," in IEEE ICC 2003, USA, May

[14] T.Dyer and R. Boppana, "A comparison of TCP performance over three routing protocols for mobile ad hoc netoworks," in Proc of ACM Mobihoc, USA, 2001, pp.56-66

[15] K.Chen, Y.Xue, and K.Nahrstedt, "On setting TCP's congestion window limit in mobile Ad Hoc networks," in proc. of IEEE ICC, USA, May 2003

[16] Z. Fu, P. Zerfos, H. Luo, S. Lu, L.Zhang, and M. Gerla, " The impact of multihop wireless channel on TCP throughput and loss," in Proc. of IEEE INFOCOM, USA, April 2003

[17] K. Xu, M. Gerla, L. Qi and Y. Shu, “Enhancing TCP Fairness in Ad Hoc Wireless Networks Using Neighborhood RED,” IEEE Mobicom’03, September 2003

[18] K. Tang and M. Gerla, “Fair Sharing of MAC under TCP in Wireless Ad Hoc Networks,” Proceeding of IEEE MMT’99, October 1999

[19] S. Xu and T. Saadawi, “Dose the IEEE 802.11 MAC protocol work well in multihop wireless ad hoc networks?” IEEE Communications Magazine, 39(6), Jun. 2001

[20] K. Xu, S. Bae, S. Lee and M.Gerla, “TCP behavior across multihop wireless networks and the wired internet,” Proceedings of ACM WoWMoM’02, Sep. 2002

[21] Network Simulator. http://www.isi.edu/nsnam/ns

[22] D. Johnson and D. Maltz, “Dynamic source routing in ad hoc wireless networks,” Mobile Computing, T. Imelinsky and H. Korth, eds, Kluwer Academic Publishers, pp 153-181, 1996

[23] V. Kanodia, A. Sabharwal, B. Sadeghi and E. Knightly, “Ordered packet scheduling in wireless ad hoc networks: mechanisms and performance analysis,” Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing, June 2002

[24] Zuyuan Fang, Brahim Bensaou, Yu Wang, “Performance evaluation of a fair backoff algorithm for IEEE 802.11 DFWMAC,” Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing, June 2002

[25] Karthikeyan Sundaresan, Vaidyanathan Anantharaman, Hung-Yun Hsieh, Raghupathy Sivakumar, “ATP: a reliable transport protocol for ad-hoc networks,” Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing, June 2003

[26] M. Gerla, K. Tang, and R.Bagrodia, “TCP performance in wireless multihop networks,” Proceeding IEEE International Workshop on Mobile Computing System and Applications (WMCSA’99), February 1999

[27] J. Li, C. Blake, D.S.J. DeCouto, H.Lee, and R. Morris, “Capacity of ad hoc network wireless networks,” Proceeding of ACM/IEEE MobiCom, July 2001

[28] R. Jain, “The Art of Computer System Performance Analysis,” New York, April 1991

[29] J. Li and Q. Yin, “Applying TCP-Vegas in MANETs and Using Cross-Layer Interaction Strategy to Solve the Re-Routing Problem to Improve Vegas Throughput,” Submitted for publication

[30] J. Li and Q. Yin, “Improving TCP Fairness in MANET by Adaptively Setting Congestion Window Limit,” Submitted for publication