# Design of TCP Congestion Control Techniques by Router-Assisted Approach

Yao-Nan Lien
Department of Computer Science
National Cheng Chi University
lien@cs.nccu.edu.tw

Yung-Bing Chung
Department of Computer Science
National Cheng Chi University
g9207@cs.nccu.edu.tw

***Abstract*—With tremendous growth of Internet traffic, to utilize network resources efficiently is essential to a successful congestion control protocol. Transmission Control Protocol (TCP) is a widely used end-to-end transport protocol across the Internet. It has several enhancing versions such as TCP Reno and TCP Vegas, which intend to improve the drawbacks of the initial version of TCP. Most congestion control techniques use trial-and-error based flow control to handle network congestion. In this paper, we propose a new method named TCP Muzha that requires routers to feedback their status to sender. Based on this information, sender is able to adjust the data rate dynamically. We also propose a multi-level data rate adjustment method to help dissolving congestion. Finally we use NS2 simulator to evaluate the performance of our approach.**

***Keyword**: TCP, congestion control, router.*

## I. INTRODUCTION

With the fast expansion of Internet and growth of various network traffic, resources have become more and more valuable to end users and operators. Some important issues such as network congestion have been raised under the circumstance of traffic burstness. The root cause of congestion is usually the amount of packets generated by end users exceeds the capacity of the network. Network congestion will result in long delay time and high packet loss rate as well as many negative effects in performance.

With respect to a path that connects two end points, congestion usually occurs in a bottleneck node. Network elements including router and end terminals need to work hard to prevent the network from being crashed by congestion. In the present, TCP/IP is the de facto and most famous standard to Internet society for data transmission. The congestion control within the TCP (Transmission Control Protocol) plays a critical role in adjusting data rate to avoid congestion from happening.

TCP offers reliable data transfer as well as flow and congestion control. It is the most widely adopted transport control protocol in the current Internet so that its behavior is tightly coupled with the overall Internet performance. Based on the window-adjustment algorithm, sender not only guarantees the successful packet delivery, but also maintains the correct sequence of packet by receiving the frequent acknowledgement from the receiver. The strength of TCP also relies on the nature of its congestion avoidance and control algorithm and its retransmission mechanism.

TCP protocol is executed at the terminal nodes of a network. It doesn't have real time inside information about the network. The only indicators of network status to the TCP protocol is packet traveling time as well as success or failure of package delivery. Therefore, most current TCP versions count on these indicators to "guess" (estimate) the available bandwidth over the path connecting the sender to the receiver and to adjust data rate accordingly. The accuracy and the promptness of bandwidth estimation are dependent on many factors such as the stability of network traffic and the length of the path. Not surprisingly, most TCP versions are suffering some performance shortcomings such as congesting network by sending data too fast as well as decreasing data rate unnecessarily due to what so called "slow start". Unfortunately, MANET (Mobile Ad Hoc Network) is a very unstable network with slow network elements. Each node may perform both roles of terminal and router. Running TCP over MANET will suffer from even severe performance problems.

A new approach to enhance TCP performance is to have network elements to provide assistance such as more accurate available bandwidth, per-hop latency, or simply per-hop saturation level[17]. By utilizing this information, TCP at the terminal nodes will be able to adjust its data rate closer to the network capacity and to improve the performance of both TCP and network accordingly.

This approach may not be easy to implement on WAN (Wide Area Network) because upgrading a large number of routers in a WAN is almost a business impossible. However, MANET has no such concern so that it is easy for a MANET to embrace this new approach.

In this paper, we propose a new TCP protocol, called TCP Muzha, that uses information provided by routers to achieve better congestion control. To use TCP Muzha, routers are required to provide some information allowing the sender of TCP to estimate the available bandwidth more accurately over the bottleneck node with respect to the path from the sender to the receiver. With this information, TCP Muzha will be able to

enhance the performance of both TCP and network.

The rest of this dissertation is organized as follow. In Chapter 2, we review the relative background and research regarding to TCP. In Chapter 3, we introduce our congestion control mechanism – TCP Muzha and we evaluate our algorithm with others by simulation in Chapter 4. Finally, we conclude our main contribution of this dissertation and highlight some future work in Chapter 5.

## II. Related Work

TCP is now the most well-known and popular transport control protocol. It provides the reliable data transmission between two end hosts and many of the current Internet applications rely on TCP as a fundamental protocol. The sender and the receiver of a TCP session work together to guarantee reliable data delivery and to control data flow. Receiver has to acknowledge the reception of received data segment. The sender retransmits each data segment if the corresponding acknowledgement is not received in time.

In order to ensure the network stability, each terminal node has to response to congestion by implementing different congestion control mechanisms. Otherwise the network would suffer from congestive collapse. The transmitting data rate is governed by two sliding windows: Receiver Window and Send Window. They are used for preventing the transmitted data running over receiver's buffer and network capacity, respectively.

However, the original TCP doesn't have the ability to deal with congestion [1]. By observing serious of congestive collapses, Jacabson proposed a new TCP congestion control mechanism [3] named TCP Tahoe which included Slow Start, additive increase and multiplicative decrease (AIMD) and Fast Retransmit algorithm. Two years later, after Fast Recovery algorithm [4] was proposed and entered as one part of the previous protocol, TCP Reno has formed to become the most widely used TCP version since then.

In order to achieve maximum data throughput while avoiding congestion and subsequent packet losses in the network, the flow control in TCP needs to perform three tasks: (1) to estimate the available bandwidth along the path connection the sender to the receiver; (2) to detect the occurrence of congestion; and (3) to reduce the data rate to an appropriate level to resolve the congestion when it occurs.

Most versions of TCP use packet loss as the indicator of network congestion although it may not be a reliable signal under some special circumstances. TCP Reno is the most typical protocol of this category.

### A. TCP Reno

In TCP Reno, two different types of packet loss indicators represent two different levels of network congestions. When the receiver receives mis-ordered packets, it transmits a duplicate ACK to the sender. If three duplicated ACKs are received by the sender before timeout, it assumes only a small portion of packets are lost so that the Send Window is cut into about one half and Fast Recovery is activated to get into Congestion Avoidance phase. The size of Send Window is increased by one every time an ACK is received until a congestion is sensed.

On the other hand, when transmission timer is expired, the network is assumed in serious congestion and the Send Window is cut into one to reduce data rate dramatically. Furthermore, Slow Start phase is activated to double the size of Send Window every time an ACK is received until a congestion is sensed. Slow Start is also used when TCP initiates a connection to discover the available bandwidth.

### B. TCP Vegas

In 1994, L.S.Brakmo proposed TCP Vegas [10][11][12] which uses the fluctuation of round trip packet traveling time (RTT) to measures the congestion level and adjust data rate accordingly. While RTT increases, TCP Vegas diagnoses congestion level is increasing and then reduces the size of congestion window. Otherwise, if RTT decreases, TCP Vegas treats it as the relief of congestion and increases the congestion window. Vegas can react to network congestion in such a fine grain manner so that it can avoids the periodic packet loss that usually occurs in TCP Reno and therefore the change of congestion window size is much more stable. However, Vegas is too conservative when other types of protocols are coexisting, particularly TCP Reno. The behavior of competing resources will lead to great degradation of Vegas' throughput. [13][14][15][16].

### C. Router Assistance

As mentioned in Section 1, TCP doesn't have real time inside information about the network so that it has to estimate the available bandwidth. However, accuracy and the promptness of bandwidth estimation depend on many factors such as the stability of network traffic and the length of the path. Therefore, most TCP versions are suffering some performance shortcomings. It is not easy to enhance their performance unless routers can provide some assistance.

ECN[19] was proposed by Sally Floyd. ECN employs the support from router which is responsible for marking the ECN bits in the IP header to indicate the tendency of congestion. If the sender receive the packets with ECN marking, it can activate the congestion control mechanism to alleviate the network congestion.

TCP RoVegas [ ] is an enhanced version of TCP Vegas by router-assisted approach. The main target of this protocol is to solve unreliable reception of ACK under asymmetric network. In a regular TCP, if an ACK is blocked due to the backward link congestion, TCP sender would determine the forward link congestion occurred and trigger congestion control mechanism to result unnecessary throughput degradation. In RoVegas, the latency that a packet passing through routers are accumulated and marked in IP header so that the sender could use this information to determine whether the lost of ACK is caused by

forward or backward path. Thus, it can ignore the congestion on the backward path and keep the data rate on the forward path unchanged.

Recently Intel and HP has advanced a project named PlanetLab [20]. This project proposes a "New NET" concept that requires routers to provide some assistance to the upper layer programs in order to achieve more realistic and efficient congestion control. The original intention of New Net has the same idea by coincidence with what we are going to propose in rest of the paper.

## III. TCP MUZHA

Most of the TCP are not aware of network condition such that they may not be able to control congestion precisely and promptly resulting in unstable bandwidth utilization. If two end hosts can obtain network information from routers, they will be able to execute congestion control more efficiently.

With respective to a path, congestion usually occurs in the bottleneck point which has the minimum available bandwidth. If the sender can adjust the data rate dynamically according to the status of the bottleneck, the congestion should be avoid or dissolved efficiently. Therefore, we propose a new congestion control mechanism – TCP Muzha, with router-assisted approach to detect the bottleneck along the path and estimate the available bandwidth of the bottleneck.

### *A. Design Objectives*

The objectives of TCP Muzha are as follows:

1. Reducing the occurrence of congestion
2. Allowing sender reach the maximum sending rate more quickly
   While every new TCP connection initiates, transmitting data rate increases according to the congestion window, such as slow start. TCP Muzha allows sending rate react a reasonable value without causing congestion. If the congestion can be dissolved efficiently, the utilization degradation due to packet loss or timeout is able to be alleviated. By doing this, the performance will be improved significantly.
3. Maintaining stable throughput while coexisting with TCP Reno.
   TCP Reno has a very aggressive behavior regarding to bandwidth competition with other protocols such as TCP Vegas. Therefore our proposed mechanism wish to overcome this shortcoming while coexisting with TCP Reno.

### *B. Design Issues*

The design issues are how to calculate the available bandwidth; how to locate the bottleneck; and how to use the residual bandwidth in the bottleneck.

### *C. Calculation of Available Bandwidth*

The available residual bandwidth in each router depends on many factors such as the length of the queue, queueing time, buffer size and length of the spare queue. We assume each router is able to calculate by itself the available bandwidth and convert it into an index called Data Rate Adjustment Index (DRAI), which will be explained later.

### *D. Locating the Bottleneck*

TCP Muzha defines a new IP option named AVBW-S (Available Bandwidth Status) in IP packet header. The sender of a TCP Muzha connection sets the AVBW-S to a maximum value for every packet it sends. Each router compare its own DARI with this value and replace it if its value is smaller. The receiver end notices the minimum value of DRAI and sends this information back to the sender. The sender can then use this information to adjust its data rate, i.e. the size of its Send Window.

Please note that the location of the bottleneck within a path (TCP connection) may be shifted over time due to the dynamic of traffic.

### *E. Use of Available Bandwidth*

Because of the following reason, each router publishes a DRAI value instead of the original available bandwidth. If there is more than one TCP Muzha connection passing through a router, which is very likely, most of them may try to adjust their transmitting data rates up to the level they are informed. Disseminating the original residual bandwidth directly to all TCP senders will lead to traffic burst immediately. Therefore, a router must smartly share its residual bandwidth to all the TCP connections that pass through the router. Unfortunately, routers are usually not aware of the types of transport protocols that control the packets which the routers are forwarding such that they cannot simply divide their residual bandwidth by the total number of TCP connections. Although a router may be able to peek into the content of packets to determine their controlling transport protocols, we do not take this approach because it may consume a significant part of router's capacity. Furthermore, violating protocol independence principle may induce unexpected reliability problems.

### *F. Design of DRAI*

Because of the difficulties mentioned above, TCP Muzha takes the following approach: instead of publishing available bandwidth, routers make recommendation to the passing traffic flows to increase or to decrease their data rates. Each router determines a DRAI value, which is a quantified data rate adjustment recommendation, according to its own network status and publish this information. With respect to each TCP connection, there is a minimum DRAI value called Minimum data Rate Adjustment Index (MRAI). Senders can refer to this value to increase or decrease its data rate (i.e. window size). By this approach, the decision of data rate adjustment is no longer the sole responsibility of senders. Routers which have knowledge of network status can participate in the decision of data rate adjustment. The sender will be able to adjust its data rate according to the MRAI before actual congestion occurring and doesn't have to rely on the actual occurrence of congestion to trigger congestion control.

### *G. Multi-Level Data Rate Adjustment*

The most critical design issue in TCP Muzha is the determination of DRAI. Currently, there doesn't exist any

theoretical formula for this. We take empirical approach to design the DRAI formula. Due to a lack of mature knowledge, we choose a coarse grain multi-level quantization formula that defines the data rate adjustment recommendation into levels such as aggressive acceleration / deceleration, moderate acceleration / deceleration, and stabilizing. Further empirical research is needed to find a formula for routers to determine their DRAIs based on their bandwidth utilization.

In fact, ECN is the perfect example of router-assisted and it can be viewed as an extreme case of multi-level DRAI. Routers mark the ECN field to 1 within the packets if the queue maintained by RED mechanism exceeds the certain threshold. Then routers publish this information and notice sender to be aware of coming congestion. But this information is too brief for sender to gain further network status. Therefore, only passive approach such as AIMD is used for congestion avoidance under such scenario. That is why the binary approach of ECN still has drawbacks on controlling flow and date rate. If the sufficient information can be provided by routers, the congestion control mechanism is able to work more efficiently and preciously. Therefore we proposed a fuzzy multilevel date adjustment approach as a guideline for routers and end hosts.

### *H. TCP Muzha Congestion Control Mechanism*

Unlike other TCP versions that need to "probe" network bandwidth by increasing their data rates carefully using Slow Start and AIMD, TCP Muzha is able to adjust data rate based on the recommendation given by routers. Thus, TCP Muzha simplifies the three phases of TCP Reno into two phases; CA (Congestion Avoidance) phase and FF (Fast Recovery & Fast Retransmit) phase.

While TCP session initiated, it directly enters the CA phase. After receiving the new ACK, sender adjusts the CWND size according to the MRAI. After a congestion occurs, TCP Muzha inherits most of the congestion control mechanisms from the traditional TCP in order to response to congestion immediately. If three duplicate ACKs are received by the sender, TCP Muzha enters the FF phase and reduces CWND to one half. If transmission timer expires, the sender would reset CWND to 1 and return to CA phase.

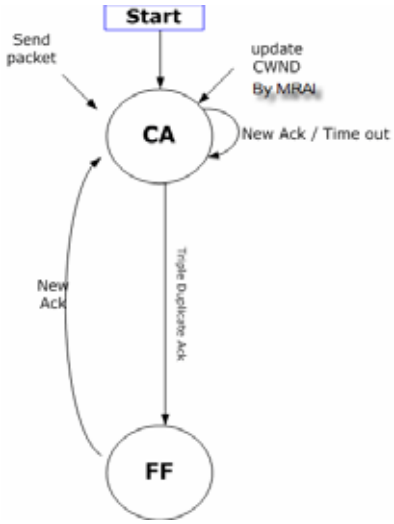



Figure 3.1 : Phase transition diagram of TCP Muzha

| Event | Status | Behavior of TCP Sender | Note |
|---|---|---|---|
| Receive the ACK of the pervious packet | Congestion Avoidance (CA) | Dynamically adjust CWND according to the returning rate adjustment index | Adjust CWND in every RTT |
| Receiving three duplicate ACKs | Congestion Avoidance (CA) | (1)CWND = CWND * (1/2)<br>(2) Enter FF phase | Fast respond and half the CWND |
| Timeout | Congestion Avoidance (CA) | (1)CWND = 1<br>(2)Re-enter CA phase | Re-enter the congestion avoidance phase |

Figure 3.2 : Congestion control mechanism of TCP Muzha


Figure 3.3 : Flowchart of the procedure for TCP Muzha upon receiving an ACK

### I. *Summary*

TCP Muzha provides a feasible mechanism of date rate control without packet loss according to the information published by routers. By doing so, the degradation of throughput due to packet loss can be resolved and the throughput can be controlled to match to the fluctuation of the network bandwidth. The design principle of TCP Muzha is trying to locate the bottleneck along the path and calculate the available bandwidth of the bottleneck. According to the recommendation given by the bottleneck, TCP Muzha is able to dynamically control data rate in order to utilize bandwidth more efficiently and avoid congestion. The target of TCP Muzha is how to determine the information published by routers and employ them to adjust date rate dynamically. We also proposed a fuzzy multi-level data adjustment mechanism for precisely control the date rate without occurrence of congestion. In the next chapter, we evaluate our approach by NS2 network simulator.

## IV. PERFORMANCE EVALUATION

In this section, we evaluate and compare the performance of proposed congestion control mechanism, TCP Muzha with TCP Tahoe, TCP Vegas and TCP Reno in different designed network environments by using the network simulator NS2 []. We also observe and show the behavior and performance while TCP Muzha coexists with TCP Reno as well as the issue of TCP global synchronization.

The metrics of performance evaluation are as follows.

1. Change of CWND (congestion window size )
2. Average Throughput
3. Packet Loss Rate
4. Average Delay Time

The general parameters are listed in Table 4.1 and the DRAI formula used by TCP Muzha is shown in Table 4.2.

| Parameter | Range |
|---|---|
| Number of Nodes | 3~16 |
| Link Bandwidth | 1~15Mbps |
| Link Delay | 5~96 ms |
| Buffer Size | 15~100 packets |
| Traffic Load | 300~900Kbps |

Table 4.1: Simulation Parameters

| DRAI | Meaning | Change of CWND |
|---|---|---|
| 6 | Aggressive Acceleration | cwnd = cwnd *3 |
| 5 | Moderate Acceleration | cwnd = cwnd*1.75 |
| 4 | Stabilizing I | cwnd = cwnd +2 |
| 3 | Stabilizing II | cwnd = cwnd +1 |
| 2 | Moderate Deceleration | cwnd = cwnd -1 |
| 1 | Aggressive Deceleration | cwnd =cwnd *3/4 |

Table 4.2 DRAT Formula

Four simulations were executed to evaluate the overall performance of TCP Muzha

1. Observe the change of CWND for TCP Muzha under single TCP session.
2. Observe the performance of TCP Muzha under multiple TCP sessions and various network conditions
3. Observe the behavior regarding to fairness while coexisting with TCP Reno
4. Observe the behavior of TCP Muzha under the scenario of TCP global synchronization.

### A. *Simulation 1: Single TCP Session*

In this subsection, we investigate the change of CWND for TCP Muzha under single TCP session. The first network topology for the simulation is shown in Fig. 4.1. This topology included source, destination and two routers only have a single TCP session. The bandwidth between routers is 1Mb and the link delay time is 20ms. The bandwidth from source to router and destination to router are 10Mb and link delay time is 2ms. The queuing management of routers is DropTail. We set the buffer size of routers to 50 and 15 respectively and observe the behavior and the CWND change of TCP Muzha.

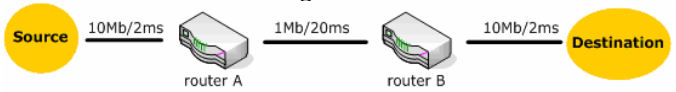



Figure 4.1: Network topology for simulation 1

The results are shown in Fig.4.2. TCP Muzha can adjust its CWND size up to the network bandwidth promptly. TCP Vegas remains its CWND steadily but due to its conservative nature in congestion control, the CWND size is not able to keep up to the network bandwidth. TCP Tahoe and Reno tend to trigger its congestion control mechanism due to periodic packet loss.

However when the buffer size of routers is reduced, the packet loss rate raises tremendously. From Fig. 4.3, we found that TCP Muzha can't avoid first congestion, but the dynamic adjustment mechanism can stabilizes the change of the CWND Immediately. On the other hand, the CWND of TCP Tahoe and Reno fluctuate extremely and frequently due to the frequent packet loss. TCP Vegas still controls its CWND conservatively.

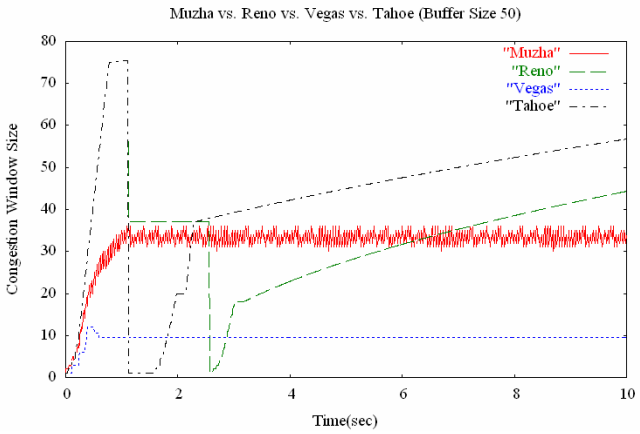



Figure 4.2: The change of CWND when buffer size = 50 under single TCP session

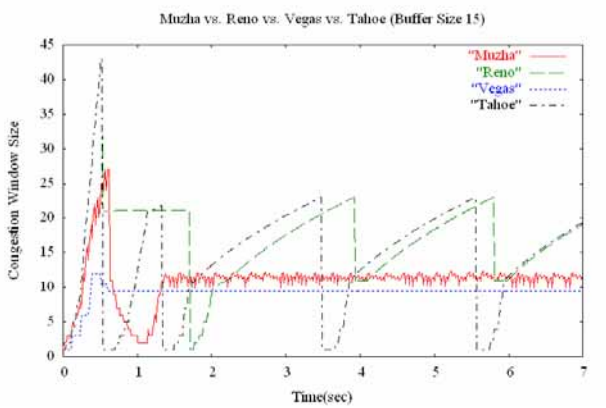



Figure 4.3: The change of CWND when buffer size = 15 under single TCP session

*B. Simulation 2: Multiple TCP Session*

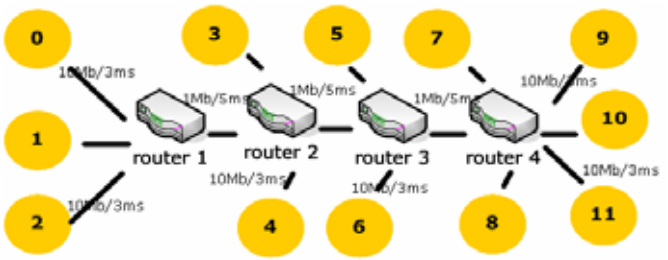


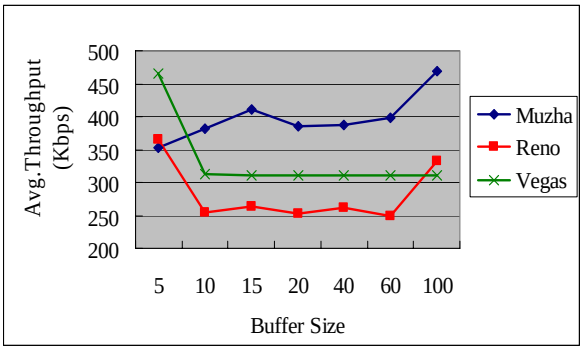

Figure 4.4: Topology for simulation 2A

The network topology used in the simulation 2A is more complex as depicted in Fig. 4.4. Some TCP connections pass through intermediate routers and burst traffic with 100kbps is emitted into the topology in order to increase the traffic variety. The overall simulation time is 30 sec. The bandwidth between routers is 1Mb and delay time is 5ms. The bandwidth from sender to router and receiver to router is 10Mb and delay time is 3ms. The queuing management of router is DropTail. Table 4.3 shows the parameters of this simulation.

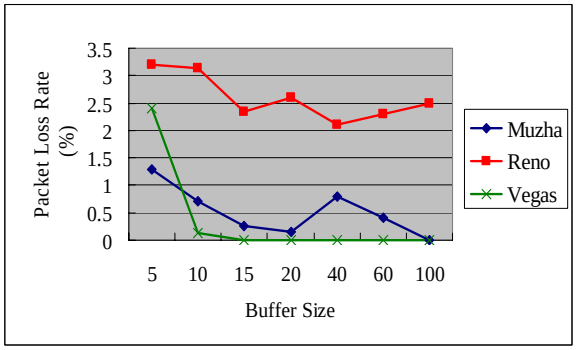
| Simulation 2A | |
|---|---|
| Buffer Size | 5~100 packets |
| Traffic Load | 100K |
| Link Bandwidth | 1~10Mb |
| Delay Time | 11~21ms |

Table 4.3: Parameters of Simulation 2A

From the simulation results presented in the Fig. 4.5, with the small buffer size, the performance of our proposed TCP Muzha suffers from buffer overflow while the CWND increases rapidly. TCP Vegas performs better than TCP Muzha because TCP Vegas tends to be conservative for CWND adjustment and maintains a reasonable queue length with lower delay time. On the other hand, when buffer size is large, TCP Muzha has greater performance than TCP Vegas. Finally TCP Reno only has a good throughput while buffer size is large due to its aggressive congestion control.




(a)




(b)

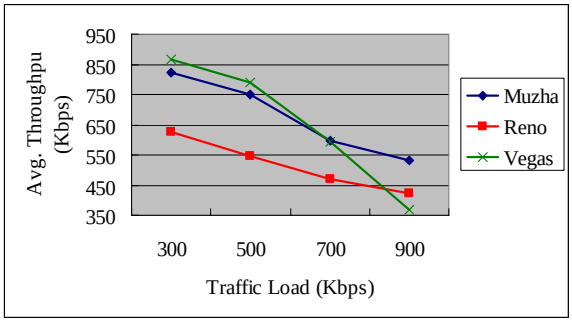
Figure 4.5: Results of Simulation 2A
(a) Avg. Throughput (b) Packet Loss Rate

The topology used in simulation 2B is the same as that in simulation 2A and parameters are listed in table 4.4. The simulation time is 30 sec and the bandwidth between routers is 2Mb with 4ms delay time. The management of queuing is DropTail. Traffic load is from 300K to 900K.

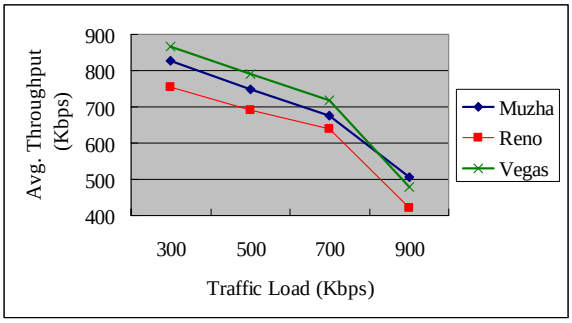
| Simulation 2B | |
|---|---|
| Buffer Size | 20~100 packets |
| Traffic load | 300、500、700 、900K |
| Link Bandwidth | 2~10Mb |
| Delay Time | 6~15ms |

Table 4.4: Parameters for simulation 2B

The results of simulation 2B are shown in Fig. 4.6. When traffic load is lower than 500K, TCP Vegas slightly perform better than TCP Muzha. But when traffic load is higher than 500K, TCP Muzha has better performance than TCP Vegas. TCP Reno suffers from the congestion seriously due to heavy traffic load. The packet loss rates are shown in Fig. 4.7



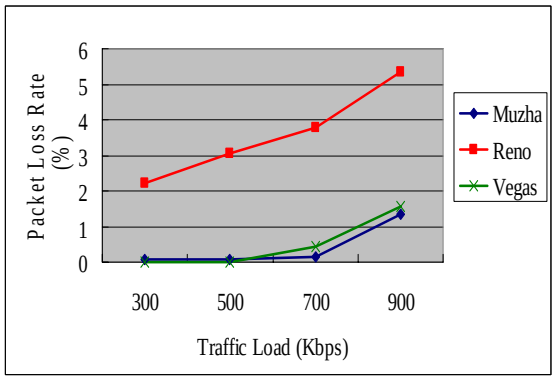

(a)



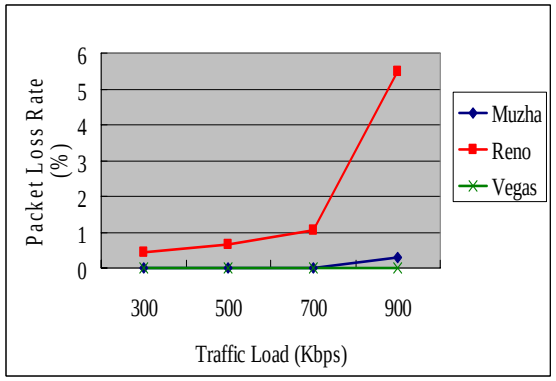

(b)

Figure 4.6: Results of Simulation 2B, Average Throughput
(a) buffer size =20 (b) buffer size =60

When traffic load is increased, TCP Muzha has much lower packet loss rate than TCP Reno.




(a)




(b)

Figure 4.7: Results of Simulation 2B, Packet Loss Rate
(a) buffer size =20 (b) buffer size = 60

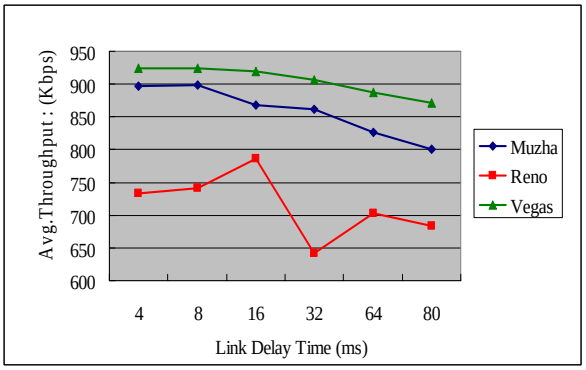
Because TCP Muzha uses returning ACKs as a guide to adjust data rate, in this subsection we describe the impact on the performance with increasing link delay time.

The network topology of simulation 2C is the same as the last simulation. The parameters are shown in Table 4.5.

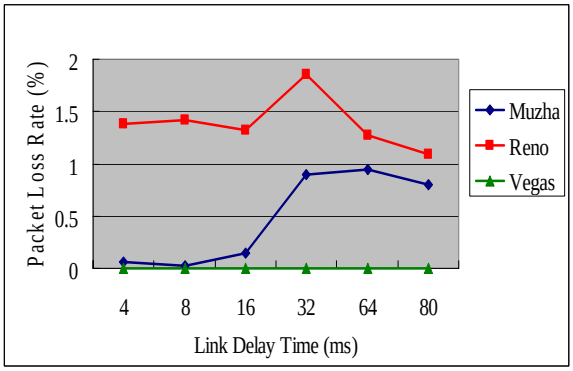
| Simulation 2C | |
|---|---|
| Buffer Size | 25-75 packets |
| Traffic Load | 100K |
| Varied Link Delay Time | 4、8、16、32、64、80 ms |

Table 4.5: Parameters for simulation 2C

The simulation results are shown in Fig. 4.8. Because TCP Muzha depends on the latest network status provided by routers, the large link delay time may hurt the response time of TCP Muzha. When link delay time is greater than 32ms, the packet loss rate increases significantly. On the other hand, TCP Vegas also has throughput degradation with increase of link delay time but the packet loss rate is much less than TCP Reno and TCP Muzha.




(a)




(b)

Figure 4.8 : Results of Simulation 2C, Buffer Size =25
(a) Avg. Throughput (b) Packet Loss Rate

### *C. Simulation 3: Coexistence with TCP Reno*

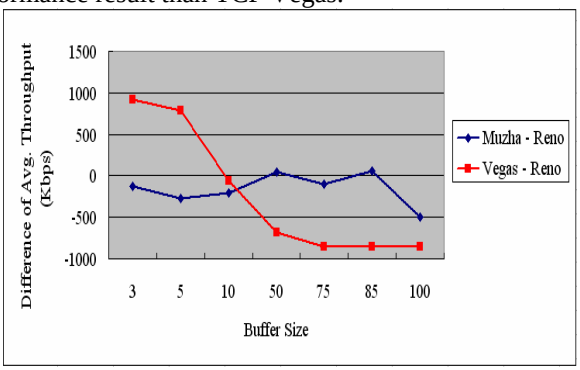
In this section, TCP Muzha is evaluated while coexisting with TCP Reno. The topology is shown in Fig. 4.9. There are two connections from node 1 to node 3 and from node 2 to node 4 respectively. One of the connections is TCP Reno and the other is either TCP Vegas or TCP Muzha. The simulation time is 30 sec and the bandwidth between routers is 3Mb. The link delay time is 3 ms and the queuing management is DropTail. The parameters of this simulation is shown in Table 4.6


Figure 4.9: Network Topology for simulation 3

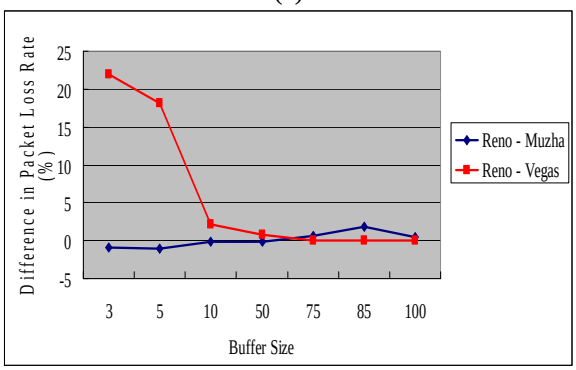
| Simulation 3 | |
|---|---|
| Buffer Size | 3、5、10、50、75、85、100 |
| Link Bandwidth | 3~10Mb |
| Delay Time | 21ms |

Table 4.6: Parameters for simulation 3

From simulation 1 and 2, we can conclude that both TCP Muzha and TCP Vegas have a good performance under single-protocol environment. However, while multiple protocol coexist, the outcomes of resource competing between TCP Reno and Vegas, as shown in the Fig. 4.10, drive TCP Vegas to a serious performance degradation. Because TCP Muzha is not as aggressive as TCP Reno in resource competing, the performance of TCP Muzha does degrade while coexisting with TCP Reno. However TCP Muzha has a more stable performance result than TCP Vegas.




(a)




(b)

Figure 4.10: Results of Simulation 3
(a) Difference in Avg. Throughput
(b) Difference in Packet Loss Rate

### *D. Simulation 4: TCP global synchronization*

There are numerous TCP sessions simultaneously in real world network. Different TCP sessions activate and enter the network at different time point. While a new TCP session initiated, its CWND size is different than other sessions. Ideally while several TCP sessions share the same link, the RTTs (round trip time) of these TCP sessions from sender to receiver should be the same if the bandwidth sharing is fair.

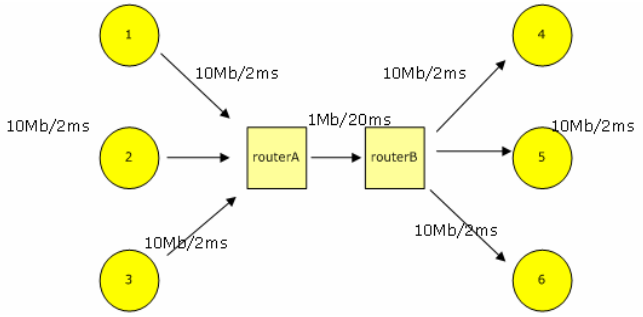



Figure 4.11: Topology for Simulation 4

The topology of simulation 4 is shown in Fig. 4.11. Three TCP sessions are injected. The simulation time is 50 sec and the bandwidth between routers is 3 Mb. The link delay time is 15ms. The queuing management is DropTail.

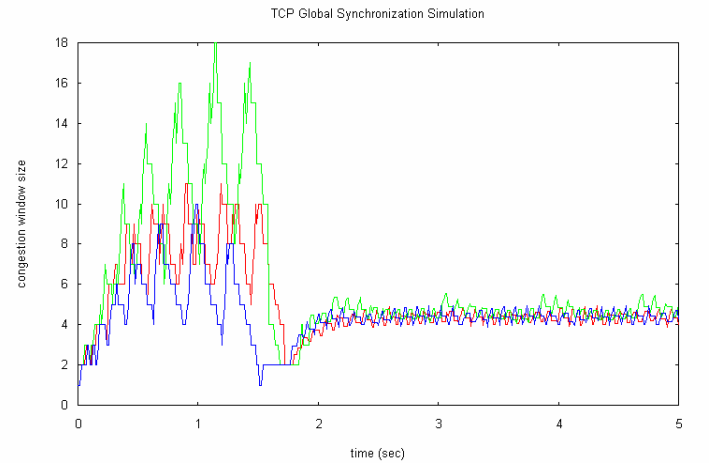



Figure 4.12: Observation of TCP Muzha under TCP global synchronization

The results of simulation 4 are shown in Fig. 4.12. When all three sessions are initiated simultaneously, the final data rate of TCP Muzha tends to be the same. If the sessions enter the network at different time, newer sessions may have lower throughput because they tend to use the residual bandwidth.

## V. CONCLUSION

From the results shown from a serious of simulations, we conclude that TCP Muzha can resolve congestion efficiently and has higher average throughput than TCP Reno about 25%.While comparing with TCP Vegas, the performance of TCP Muzha is slightly lower but while network has sufficient resource, TCP Muzha performs better that TCP Vegas due to its dynamic data rate adjustment mechanism. While coexisting with TCP Reno, TCP Muzha remains a stable performance output compared with TCP Vegas.

The concept of multi-level data rate adjustment and the details of how to control the size of CWND are still needed to be investigated. For instance, improve the vibrating behavior of CWND of TCP Muzha, global synchronization enhancement, consideration of queue size and RTT.

The mechanism of using ACK as indicator still has drawbacks. For example, while sessions share and pass through the same link, the sender with large link delay would have weakness while competing bandwidth due to the delay ACK. There are several other issues under asymmetric network as well.

Fairness is still a important issue which needs to pay extra attention on. TCP Vegas suffers seriously in multi-protocols environment because it can't share bandwidth fairly and this drives to difficulties of real-world implementation of TCP Vegas. Therefore how to enhance TCP Muzha to meet such demands will be our future task.

## REFERENCES


[1] J. Postel, "Transmission Control Protocol," *IETF RFC 793*, 1981.

[2] D. Clark, "Window and Acknowledgement Strategy in TCP," *IETF RFC 813*, 1982.

[3] V. Jacobson, "Congestion Avoidance and Control," *Proc. of ACM SIGCOMM*, pp. 314-329, Aug. 1988.

[4] V. Jacobson, "Modified TCP Congestion Avoidance Algorithm," Technical report, Apr. 1990.

[5] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," *IETF RFC 2001*, 1997.

[6] D. Chiu and R. Jain, "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks," *Computer Networks and ISDN Systems*, vol.1, pp. 1-14, 1989.

[7] Sally Floyd, T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm, " *IETF RFC 2582*, 1999.

[8] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgement Options," *IETF RFC 2018*, 1996.

[9] K. Fall and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP," *ACM Computer Communication Review*, vol. 26, no.3, pp. 5-21, 1996.

[10] L. S. Brakmo, S. W. O'Malley, and Larry L. Peterson. "TCP Vegas: New Techniques for Congestion Detection and Avoidance, " *Proc. Of ACM SIGCOMM*, pp. 24-35, Aug. 1994.

[11] L. S. Brakmo and L. L. Peterson. "TCP Vegas: End to End Congestion Avoidance on a Global Internet," *IEEE Journal on Selected Areas in Communication*, vol.13, no.8, pp. 1465-1480, Oct. 1995.

[12] J. S. Ahn, P. B. Danzig, Z. Liu, and L. Yan, "Evaluation of TCP Vegas : Emulation and Experiment," *Proc. of ACM SIGCOMM*, pp. 185-195, Aug. 1995.

[13] C. Barakat, E. Altman, and W. Dabbous, "On TCP Performance in a Heterogeneous Network : A Survey," *IEEE Communications Magazine*, vol.38, no.1, pp. 44-46, Jan. 2000.

[14] E. Altman, C. Barakat, E. Laborde, "Fairness Analysis of TCP/IP," *IEEE Conference on Decision and Control*, Dec. 2000.

[15] G. Hasegawa and M. Murata, "Survey on Fairness Issues in TCP Congestion Control Mechanisms," *IEICE Transactions on Communications*, vol. E84-B, no.6, pp. 1461-1472, Jun. 2001.

[16] R. Denda, A. Banchs and W. Effelsberg, "The Fairness Challenge in Computer Networks," *Proc. of Quality of future Internet Service*, vol. 1922, pp. 208-220, Springer-Verlag Heidelberg, Jun. 2000.

[17] Yi-Cheng Chan, Chia-Tai Chan, and Yaw-Chung Chen, "RoVegas : A Router-based Congestion Avoidance Mechanism for TCP Vegas," *Computer Communications*, vol.27, Issue 16, pp. 1624-1636, Oct. 2004.

[18] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transaction on Networking*, vol.1, no.4, pp. 397-413, 1993.

[19] Sally Floyd, "TCP and Explicit Congestion Notification," *ACM Computer Communication Review*, 1994.

[20] Planet Lab, http://www.planet-lab.org/.

[21] L. Peterson, T. Anderson, D. Culler, T. Roscoe, "A Blueprint for Introducing Disruptive Technology into the Internet," *Proc. of the 1st ACM Workshop on Hot Topics in Networks (HotNets)*, Princeton, Oct. 2002.

[22] V. Paxson, "End-to-end Internet Packet Dynamics," *IEEE/ACM Transactions on Networking*, pp. 277-292, 1999.

[23] R. Carter and M. Crovella, "Measuring Bottleneck Link Speed in Packet-Switched Networks," *International Journal on Performance Evaluation*, pp. 27-28, 1996.

[24] A. S. Tanenbaum, "Computer Networks," 4th edition, Prentice Hall, 2002.

[25] L. L. Peterson, B. S. Davie "Computer Network : A Systems Approach," 3rd edition, Morgan Kaufmann, 2003.

[26] "The Network Simulator - ns-2", http://www.isi.edu/nsnam/ns/.

[27] S. Ryu, C. Rump, C. Qiao, "Advances in Internet Congestion Control," *IEEE Communications Surveys and Tutorials*, vol 5. no.2, 2003.

[28] J. Postel, "Internet Protocol," *IETF RFC 791*, Sep. 1981.

[29] W. Stevens, "TCP/IP Illustrated, Volume 1: The Protocols," Addison-Wesley, 1994.

[30] D. Bertsekas and Robert Gallager, "Data Networks," Prentice-Hall, 1992.