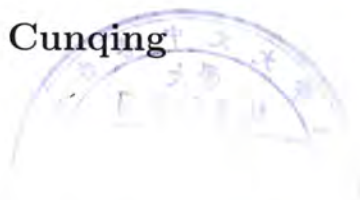


# Congestion Control and QoS Provisioning in IP Networks

HUA Cunqing



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

in

Information Engineering

©The Chinese University of Hong Kong

JUNE, 2002

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



## Acknowledgment

I am extremely grateful to Professor Tak-Shing Peter Yum, my thesis supervisor, for leading me into this research world and giving me the right amount of freedom and guidance during my graduate studies. Without his intelligent and patient guidance, it is impossible for me to finish this thesis in the end. His valuable discussion and comments have contributed greatly to this thesis.

I would like to thank other members of our research group-Yang Yang, Shen Fang Zhong, Chan King Tung, Li Cheng, Zhang Li. Their friendship and encouragements have accompanied with me in the past two years. The discussions with them have helped to deepen my understanding in the research topic and improve my research skills.

Last but not least, I would like to express my earnest gratitude to my parents, my brother and sister for their love and support. They are always behind me and gave their unconditional supports throughout my life.

## Abstract

More and more people have realized that the Internet has become an indispensable part of their lives. They rely more and more on the Internet for searching information, listening to music, watching videos, sending and receiving email, locating friends and families, playing interactive games, as well as doing business etc. The diverse requirements from the customers promote the proliferation of applications on the Internet, such as WWW, IP telephony, VOD and e-commerce etc.. The Internet has evolved gradually into a global commercial infrastructure to support these versatile applications. Most of the technologies in current IP network, designed at a the early stage of Internet mainly for the academical research usage, are not sufficient to face the challenge of emerging applications and customers.

The first challenge comes from the congestion, a state in which performance degrades due to the saturation of network resource such as link bandwidth, buffer and processor cycles. Congestion is adverse to the network performance in that it may lead to packet loss, increase the end-to-end delay, and in the worst case results in congestion collapse. Over the past decade, there has been intense research towards controlling the congestion on the network. Two classes of solution have been proposed: **host-based** mechanisms that make control decision at the end host with the end-to-end measurement, and **router-based** mechanisms that monitor network state at network core and provide congestion notification to end hosts. In this thesis, we first investigate the fairness issue of TCP Vegas, a TCP protocol employs a host-based congestion control mechanism, in the network with multiple congested gateway. We find that TCP Vegas fails to achieve the fair allocation of network resource to flows passing through different number of congested gateway. This unfairness is rooted in the cumulative nature of round trip time (RTT), the metric that TCP Vegas relies on. With the understanding that host-based mechanism cannot provide reliabel congestion control function, we then propose a scheme called *Joint Congestion Control(JCC)* for TCP/IP networks. JCC uses probing packet to collect the up-to-date congestion information from routers along the path and with which to adjust the sending rate from the end host in a proactive way. JCC is proved to be able to achieve higher utilization of network resource, lower fluctuation of flow throughput and packet loss, and fair allocation of network resource among different users.

The IP network also faces the challenge to provide quality of service guarantee to satisfy the diverse requirements of applications and users. Today's Internet provides only best-effort service, which treats all the packets as the same and forwards them with the same service level. The emerging applications such as IP telephony and e-commerce require a powerful services guarantees. The existing solutions include *integrated services*, *differentiated services*, etc., which aims to provide flow isolation and protection according

to the requirement of applications and customers. In this thesis, we propose *Shifted Waiting Time Priority (SWTP)* as a scheduling algorithm for *delay differentiated services*. We focus on the computational complexity and effectiveness of this algorithm in high speeds network environment. Using simulation experiments, we show that *SWTP* achieves a comparable functionalities with lower computational costs than existing algorithms.

## 摘要

越來越多的人已經意識到互聯網已逐漸成爲他們生活中不可或缺的一部分。他們越來越多的依賴于互聯網來搜索信息、欣賞音樂、觀看電影、收發電子郵件、聯絡朋友和家人、玩網絡交互遊戲以及進行商業活動等等。用戶多樣化的要求推動了互聯網上應用程序的興旺發展，例如萬維網、IP 電話、視頻點播也及電子商務等等。互聯網已經逐步演變爲一個全球性的商業架構，以滿足這些日益多樣的應用程序的要求。現有的 IP 技術大多設計于二十年前且主要用于于科研用途，現在已經難以應付日益涌現的應用程序及用戶群的挑戰。

首先的挑戰來自于網絡擁塞。這種狀態下，網絡性能隨著網絡資源例如帶寬、緩衝區、處理器的耗盡而顯著下降。網絡擁塞對網絡性能危害頗深，它會導致大量的數據包丟失、增加端到端的時延，最差的情況下會導致擁塞崩潰。在過去的幾年裏，已經有大量的研究致力于解決網絡的擁塞問題。目前的解決方案主要有兩類。一種是基于終端主機的措施，即所有的控制決策都是在終端基于端到端的測量數據做出。另一種是基于網絡路由器的，即路由器本身通過對網絡內部的狀態狀態的監視，提供相應的擁塞預警給終端用戶進行擁塞控制。本論文中，我們首先研究 TCP Vegas（一種基于主機擁塞控制的 TCP 協議）在有多網關出現擁塞的網絡中的不同數據流之間的公平問題。我們發現 TCP Vegas 無法公平的分配資源給經過不同數目的擁塞鏈路的數據流。這種不公平源自于 TCP Vegas 所采用的往返行程延遲（RTT）的累積特性。理論分析和計算機仿真的結果都證實了這一問題。基于這樣的認識即基于主機的機制難以提供有效和準確的擁塞控制功能，我們提出了一個新的解決方案即 Joint Congestion Control (JCC)。JCC 利用探測數據包去搜集當前網絡的擁塞信息，并反饋到終端節點進行早期速率調整于應對網絡中可能出現的擁塞狀況。JCC 能夠有效地利用網絡資源、降低流量抖動和數據包丟失，并能夠公平地分配資源給不同地用戶。

IP 網絡同時也面臨著這樣地挑戰即根據應用程序和用戶的不同要求提供不同地服務質量保障。目前的 IP 網絡只提供盡力型服務，這種模型下，所有的數據包都是在同樣的服務水平下進行轉發。一些新興的應用程序比如 IP 電話和電子商務要求有較高的服務質量保障。現有的解決方案包括 Integrated Services、Differentiated Services 等等，它們都是致力于根據應用程序和用戶的要求提供數據流隔離和保護。本論文中，我們提出一種面向 delay differentiated service 的調度算法:Shifted Waiting Time Priority (SWTP)。我們特別關注算法在高速網絡下的計算複雜度和有效性。計算機仿真結果顯示，相對於現有的算法，SWTP 算法能夠以較低的計算代價提供相當的功能。

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Congestion Control in the IP Network . . . . .	1
1.2	Quality of Service in the IP network . . . . .	2
1.3	Structure of Thesis . . . . .	3
<b>2</b>	<b>Background</b>	<b>4</b>
2.1	TCP and Congestion Control . . . . .	4
2.1.1	Slow Start . . . . .	4
2.1.2	Congestion Avoidance . . . . .	5
2.1.3	Fast Retransmit, Fast Recovery and Timeout . . . . .	5
2.2	Active Queue Management . . . . .	7
2.3	Integrated Services and Differentiated Services . . . . .	8
<b>3</b>	<b>The Fairness of TCP Vegas in Networks with Multiple Congested Gateways</b>	<b>10</b>
3.1	Introduction . . . . .	10
3.2	TCP Vegas and related works . . . . .	11
3.3	Analysis . . . . .	13
3.4	Simulation Results . . . . .	15
3.4.1	Throughput for different number of active cross connections . . . . .	16
3.4.2	Throughput for different number of flows in each connection . . . . .	17
3.4.3	Multiple congestion vs Single congestion . . . . .	17
3.5	Summary . . . . .	19
<b>4</b>	<b>The Joint Congestion Control for TCP/IP Networks</b>	<b>21</b>
4.1	Background . . . . .	21
4.2	The Joint Congestion Control . . . . .	23
4.2.1	Path Load Reduction Factor . . . . .	23
4.2.2	The Congestion Control Algorithm . . . . .	24
4.2.3	Probing Interval . . . . .	26

4.2.4	Parameter Setting . . . . .	26
4.2.5	Encoding of R . . . . .	27
4.3	Simulation Results . . . . .	28
4.3.1	Congestion Window Behavior . . . . .	28
4.3.2	Throughput Stability . . . . .	31
4.3.3	Packet Loss Ratio . . . . .	31
4.3.4	Fairness Index . . . . .	32
4.3.5	Fairness in Multiple-hop Network . . . . .	32
4.3.6	Parameter Sensitivity . . . . .	33
4.3.7	Interaction between JCC and Reno flows . . . . .	35
4.4	Summary . . . . .	35
<b>5</b>	<b><i>S</i>-WTP : Shifted Waiting Time Priority Scheduling for Delay Differentiated Services</b>	<b>37</b>
5.1	Introduction . . . . .	37
5.2	Scheduling Algorithms for Delay Differentiated Services . . . . .	38
5.3	Shifted Waiting Time Priority Scheduling . . . . .	41
5.3.1	Local Update . . . . .	42
5.3.2	Global Update . . . . .	42
5.3.3	Computational overhead . . . . .	42
5.4	Simulation Results . . . . .	43
5.4.1	Microscopic View of Individual Packet Delay of <i>S</i> -WTP and WTP	43
5.4.2	Delay Ratios in Different Timescales . . . . .	44
5.4.3	Effects of aggregate traffic and class load distribution on delay ratio	44
5.4.4	Delay Ratios with More Classes . . . . .	48
5.5	Summary . . . . .	48
<b>6</b>	<b>Conclusions</b>	<b>50</b>
6.1	Congestion Control . . . . .	50
6.2	Quality of Service Provision . . . . .	51
6.3	Final Remarks . . . . .	51



# List of Figures

2.1	TCP congestion control behavior . . . . .	6
2.2	RED algorithm . . . . .	7
2.3	DiffServ Architecture . . . . .	9
3.1	Network model . . . . .	15
3.2	Throughput with different number of active cross connections . . . . .	16
3.3	Throughput with different number of flows . . . . .	17
3.4	Throughput of connection 0 in two cases . . . . .	18
3.5	Average queue length of three gateways . . . . .	20
4.1	An example of a path with five links . . . . .	24
4.2	Reduction ratio of AIED algorithm . . . . .	25
4.3	Congestion window behavior of JCC and Reno . . . . .	26
4.4	Modified RED algorithm . . . . .	27
4.5	Simulation network . . . . .	28
4.6	Congestion window behavior of JCC and Reno . . . . .	29
4.7	Microscopic view of congestion window of JCC and Reno . . . . .	29
4.8	Throughput of four flows starting at different time . . . . .	30
4.9	Throughput of four flows with different round trip time . . . . .	30
4.10	Packet loss with different number of flows . . . . .	31
4.11	Fairness index of competing flows . . . . .	32
4.12	Throughput for different number of active cross connections . . . . .	33
4.13	Throughput and Path Load Reduction Factor ( $\alpha = 0$ ) . . . . .	34
4.14	Throughput and Path Load Reduction Factor ( $\alpha = 0.02$ ) . . . . .	34
4.15	Throughput of mixed JCC and Reno flows . . . . .	35
5.1	Priority function of WTP. . . . .	40
5.2	Simulation network . . . . .	43
5.3	Individual packet queueing delay with S-WTP and WTP) . . . . .	45

5.4	Distribution of delay ratios in different timescales	
	( $U = 95\%$ ) . . . . .	46
5.5	Delay ratios with different traffic load and distributions	
	( $\frac{b_{i+1}}{b_i} = 2.0$ ) . . . . .	47

# List of Tables

4.1	Average R and total packet loss for different $\alpha$ . . . . .	33
5.1	Average delay ratios between classes ( $\frac{b_{i+1}}{b_i} = 2.0, U = 95\%$ ) . . . . .	48
5.2	Average delay ratios between classes ( $\frac{b_{i+1}}{b_i} = 1.5, U = 95\%$ ) . . . . .	48

## Chapter 1

# Introduction

Congestion has long been recognized as an important problem in IP networks. Congestion occurring in the core network may lead to packets loss, increase queue delay, and in the worst case result in congestion collapse, a state that an increase of the offered load leads to a decrease of throughput in the network. Another problem having accompanied the growth of the IP network is to provide quality of service guarantee to satisfy the diverse requirements of applications and customers in IP networks. This thesis addresses some aspects of these challenging problems and tries to find solutions to improve the performance IP networks.

### 1.1 Congestion Control in the IP Network

Congestion is a state in which performance degrades due to the saturation of network resource such as link bandwidth, buffers, and processor cycles. The IP network has suffered from the congestion problem for a long history as for the uncoordinated resource share nature of the packet switching network. The packet switching network employs the statistical multiplexing mechanism to schedule the packet forwarding at the routers. It is possible for several IP packets to arrive at the router simultaneously, waiting for forwarding on the same output link. Obviously, not all of them can be forwarded at the same time. In this case, the buffer space in the router offers the first level of protection against the burst of traffic. However, the buffer space can be exhausted if the burst size is larger than the capacity. In this case, the router has no choice but dropping the packets. Adverse effects resulting from congestion include the packet loss, longer delay of packet delivery, wasting of network resource, and possible *network collapse* (or "network meltdown");

The studying on congestion control has long been a hot topic in IP networks. A large number of congestion control algorithms and strategies have been reported in literature. Earlier works mainly followed the end-to-end approach[8] where the congestion control

mechanisms were mainly designed at the end hosts. For example, the slide-window mechanism in TCP protocol adjust the sending rate by detecting the packet loss event at the end host. In 1984, Nagle [27] discussed the congestion collapse problem in the IP network. This promoted the initial QoS provision for end host system which addressed the problem of TCP protocol on wide-area network (WAN). Nagle algorithm is now supported by all IP host implementations. Two years later, Van Jacobson proposed a set of mechanisms to improve the congestion control functionalities for end system, which are well known as *slow start* and *congestion avoidance*. These mechanisms, together with other two additional mechanisms—*fast retransmit* and *fast recovery*, form the basis of current Internet congestion control practice.

However, due to the limited information collected at the end hosts, this kind of host-based end-to-end congestion control may be not effective in preventing congestion when the traffic concentrates at some network hot spots. In recent years, the understanding that network core should participate to control the congestion in the network has led to an intensive studying on a router-based approach called *active queue management(AQM)*. *AQM* monitors the network state and allows routers to control which packets to drop and when this should be done in a proactive way. By dropping packets(the traditional method of congestion notification) or marking the packet, the router notifies the source about the congestion and reduce the rate accordingly. RED[15] and ECN[13] are two of the most prominent and widely studied active queue management mechanisms. The goal of active queue management is to detect congestion at a earlier time and to convey congestion notification to source before queue overflow and packet loss occur. By decoupling congestion notification from packet and using active queue management mechanism, it is expected the packet loss rates in the IP network can be reduced.

## 1.2 Quality of Service in the IP network

From the birth day of the Internet, providing QoS has been envisioned and a Type of Service (ToS) field was allocated in the IP header. The purpose of ToS byte is to provide an indication of the abstract parameters for the desired quality of service. These parameters are mapped to the actual service parameters of the particular networks and used to schedule the packet forwarding as datagram traverses through the network [30]. But at the early day of Internet, ToS support was of little use due to the limited applications and traffics in the network. Almost all IP implementations ignored this field. This situation remained unchanged until the late 1980s. The interest of providing QoS in IP network grows gradually with the evolution of the Internet from its academic roots to commercial usage.

From the early 1990s, in order to better meet the needs of emerging real-time applications, a number of architectures were proposed and developed to enhance the IP

network infrastructure. *Integrated Service (IntServ)*[32, 9] aimed to provide the means for the application to express end-to-end resource requirement with support mechanisms in network core. In this model, the end host initiates a request prior to packet transmission. This request is carried by *resource reservation protocol*(RSVP). Once this request is accepted, the network will maintain per-flow state to guarantee such QoS request in the network during the transmission. The IntServ faces the scaling problem as it is required to keep per-flow states in routers along the path. *Differentiated Service (DiffServ)*[4, 28] was proposed to address this problem. DiffServ provides traffic differentiation by classifying traffic into a few classes, with relative service priority among the traffic classes. It uses a newly standardized DSCP [28] field in the IP header to mark the QoS required by the packet, and a DiffServ-enabled network delivers the packet with a PHB indicated by the DSCP field. Traffic is policed and marked appropriately at the edge of the DiffServ-enabled network. The core network is responsible for the forwarding of packets according to the per-hop behavior.

### 1.3 Structure of Thesis

This rest of this thesis is organized as follows. In Chapter 2, we first review the existing TCP protocols and their congestion control mechanisms. We then describe some works on active queue management. Lastly, We describe the IntServ and DiffServ architecture and discussed the latter one in detailed.

In Chapter 3, we investigate the fairness problem of TCP Vegas in networks with multiple congested gateways. Our analysis shows that the congestion control mechanism used in TCP Vegas may lead to unfair throughput for flows that traverse through multiple congested gateways. We then verify the analytical results through a series of simulation experiments.

With the understanding that network core can play a more active role in controlling the congestion in the network, in Chapter 4 we propose a mechanism to incorporate the efforts of the network routers and the end hosts to take effective and accurate action to avoid congestion. Simulation results show that this approach is very effective.

In Chapter 5, we study the delay differentiated services and scheduling algorithm for this service model. We propose a scheduling algorithm that reduce the complexity of delay differentiated services.

Finally, In Chapter 6 we summarize the work of this thesis and discuss the future directions of this research.

## Chapter 2

# Background

This chapter reviews some important technologies on congestion and as well as current progress on quality of service provision in the IP networks. In Section 2.1 we first discuss TCP protocols and their congestion control mechanisms. In section 2.2 we then describe the *active queue management* mechanisms. In section 2.3 we discuss the *integrated services* and *differentiated services*.

### 2.1 TCP and Congestion Control

Transmission Control Protocol(TCP) is currently the dominant transport protocol used in the IP networks. TCP is a connection-oriented protocol and uses a *congestion window (cwnd)* to control the flow rate. Currently, most TCP implementations employ a series of congestion control mechanisms, including *slow start*, *congestion avoidance*, *fast retransmit* and *fast recovery*[16, 17, 36].

#### 2.1.1 Slow Start

When a TCP connection starts up, it first enters *slow start* stage which help it to increase the congestion window rapidly from the cold start. Each time an ACK is received, the congestion window is increased by one segment. The sender can transmit up to the minimum of its congestion window and the advertised window of the receiver. The congestion window is controlled by the sender based on the sender's assessment of perceived network state; The advertised window is controlled by the receiver depending on the amount of available buffer space at the receiver for this connection.

In *slow start*, the TCP source starts by initializing the congestion window to one segment, it then transmits one segment and waits for the ACK. When the packet is acknowledged, the congestion window is incremented from one to two, and two segments can be sent. When those two segments are acknowledged, the congestion window is

doubled to four, then four segments can be sent. In this manner, the congestion window size grows exponentially. As a result, the sending rate of a TCP source can quickly approach the available network bandwidth.

### 2.1.2 Congestion Avoidance

The endless exponential increment of congestion window may lead to excessive packet losses when the sending rate is beyond the available network bandwidth. TCP employs a *congestion avoidance* mechanism to stop this exponential increase at a proper time. Each connection is required to maintain a parameter called *slow start threshold (ssthresh)*. *ssthresh* is used as an estimation of the bandwidth-delay product for the TCP connection. When the congestion window size exceeds *ssthresh*, TCP enters the congestion avoidance stage and the congestion window increases linearly (by  $1/cwnd$ ) per acknowledgement received. This provides a slow increment rate at one packet per round trip time instead of the exponential increase as in *slow start*.

### 2.1.3 Fast Retransmit, Fast Recovery and Timeout

Either in *slow start* or in *congestion avoidance* stage, the congestion window size may increase to a level beyond the network capacity, eventually leading to packet loss. When a packet is dropped in the network. TCP should be able to detect this event and retransmit the dropped packet. This is done with duplicate ACK. Specifically, when a packet is lost, the following packets arriving at the receiver are treated as out-of-order segments and duplicate ACKs (ACKs with the same sequence number as the lost packet) are sent back to the source immediately. When the duplicate ACKs are received by the TCP source, it responds with *fast retransmit* and *fast recovery* mechanisms to retransmit the lost packet. They work as follows:

1. When the third duplicate ACK is received, it sets *ssthresh* to one-half of the current congestion window size, *cwnd*. It retransmits the lost segment and sets *cwnd* to *ssthresh* plus 3. This inflates the congestion window by the number of segments that have left the network and the other end has received as out-of-order segments.
2. Each time another duplicate ACK arrives, it increases *cwnd* by one. This inflates the congestion window for the additional segment that has left the network. It may transmit a packet if allowed by the new value of *cwnd*.
3. When the next ACK arrives and acknowledges the segment, set *cwnd* to *ssthresh* (the value set in step 1). This ACK should be the acknowledgment of the retransmitted packet in step 1, one round-trip time after the retransmission. In addition, this ACK should acknowledge all the intermediate segments sent between the lost



packet and the receipt of the first duplicate ACK. This step is congestion avoidance, since TCP is down to one-half the rate it was at when the packet was lost.

If the network is so congested that the packet loss cannot be detected with duplicated ACKs, TCP relies on a separate *retransmission timeout* mechanism to trigger the retransmission of lost packets. The TCP source starts a retransmission timer when it sends a packet. When the retransmission timer is expired but the corresponding ACK is still not received, the TCP source assumes that this packet has lost in the network and reduces its window size to one segment, and retransmits the lost packet. To prevent continual retransmissions in times of severe congestion, TCP uses an exponential back-off algorithm to extend the retransmission time interval. Specifically, if the sender continually sends the same packet but never receives the corresponding ACK, it doubles the retransmission timeout interval. When the ACK for this packet is eventually received, the timeout interval is reset to the default value.

Figure 2.1 illustrates the congestion control behavior of TCP with the collaboration of these algorithms. As this figure show, TCP continually increases the congestion window until a packet loss is detected, the congestion window size is then cut down and increased again until another packet loss occurs. This process leads to a saw-tooth like behavior to TCP.

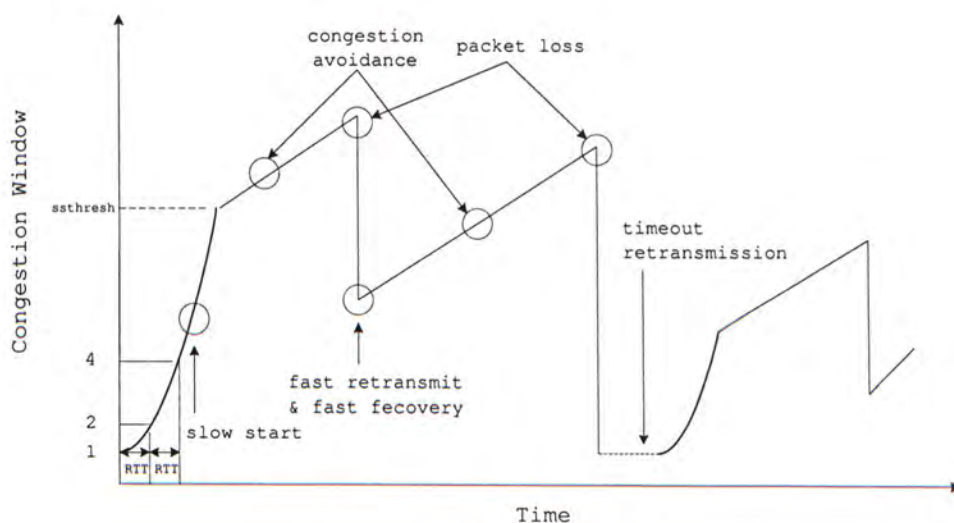


Figure 2.1: TCP congestion control behavior

Although these mechanisms help TCP to utilize the network bandwidth, it also leads to extensive packet loss. Tri-S [40] and Vegas [23] tried to improve the performance of TCP by monitoring the variation of round trip time. They take the increase of round trip time as a signal of congestion in the network. If the round trip time is detected to be

larger than a specific threshold, TCP source will decrease the congestion window size to slow down the sending rate. Otherwise it increases the window size to seize the available network bandwidth. The details of TCP Vegas will be discussed in the Chapter 3.

## 2.2 Active Queue Management

The congestion control mechanisms employed in the end hosts is activated only when a packet is lost or the round trip time has increase beyond a limit. This will cause some problem since it needs considerate amount of time to take effects. While during this time, there may already have considerable packets lost in the network.

*Active queue management* [5] algorithms address this problem by detecting the incipient congestion at an early stage and notifying the sources to reduce their sending rates before a large number of packets are piled up at the router. One of the prominent *active queue management* algorithm is *Random Early Detection (RED)*[15]. RED maintains three parameters  $min_{th}$ ,  $max_{th}$  and  $max_p$ . Each time a packet arrives, the router computes a dropping probability according to a function as shown in Figure 2.2. In this figure, the x-axis is the average queue size, the y-axis is the dropping probability. The probability of dropping an arriving packet is determined by the current average queue length. When the average queue length is less than  $min_{th}$ , no packet is dropped. When average queue length is between  $min_{th}$  and  $max_{th}$ , the arriving packet is dropped with a probability proportional to the average queue length with a maximum value of  $max_p$ . While if the average queue length is larger than  $max_{th}$ , all arriving packet are dropped.

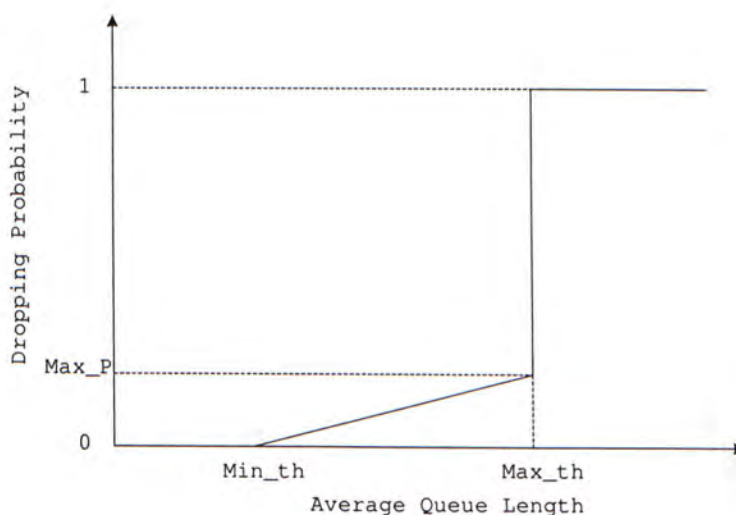


Figure 2.2: RED algorithm

In RED, the router conveys the congestion information to the end host by dropping the packets. This can be improved using the *Explicit Congestion Notification (ECN)*[13]. Instead of dropping the packet, ECN marks the packet by setting a congestion experienced bit (CE) in the packet header and uses it to inform the receiver that congestion has occurred. The receiver then sends this information with the ACK back to the sender to adjust the congestion window.

## 2.3 Integrated Services and Differentiated Services

QoS provision in IP network have long been discussed by researchers. In recent years, two classes of mechanisms have been proposed to enhance the service offered by the Internet. one is known as *integrated services(IntServ)*[32, 9], the other is *differentiated service (DiffServ)*[4, 28].

In the IntServ model, the end host requests network to reserve an amount of network resource( link bandwidth, buffer etc.) using the *Resource Reservation Protocol(RSVP)*. Once the request is admitted, the router provides a per-flow QoS guarantee to the users. IntServ provide two type of services: *guaranteed and controlled load services*. *Guaranteed service* provides deterministic delay guarantees, whereas *controlled load service* provides a network service close to that provided by a best-effort network under lightly loaded condition.

IntServ model needs to maintain per-flow information in the routers and to perform a complicated call admission control before the transmission. With the vast number of flows on the Internet today, the amount of state information required in the routers is enormous. This leads to scaling problem since the state information increases with the number of flows. This make IntServ hard to deploy on the Internet.

*Differentiated Service* uses a different approach. It provides traffic differentiation by classifying the traffic into a number of classes with relative service priorities assigned. A user can choose the performance level on a packet-by-packet basis by simply marking the packet's *Differentiated Services Code Point(DSCP)* field to a specific value. The routers forward the packet with a per-hop behavior (PHB) according to the DSCP of the packet. Typically, each user is associated with a *service level agreement (SLA)*, which is used to negotiate between the user and the Internet Service Provider (ISP) for the forwarding service the user expects. Traffics submitted out of the profile are not provided with any kind of service assurance.

Currently, the DiffServ architecture only specifies the basic mechanisms as how packet should be treated. The service provider can choose to build a variety of services by using these mechanisms as building blocks. A service is defined by some characteristics of packet transmission, such as throughput, delay, jitter, and packet loss rate. After that, a PHB is specified at all the nodes of the network offering the service (DiffServ-

enabled domain), and a DSCP is assigned to the PHB. A PHB is an externally observable forwarding behavior given by a network node to all packets carrying the corresponding DSCP value. The packet tells the network about a specific service level by carrying the associated DSCP field in its packets.

In DiffServ model, the network nodes on the DiffServ domain's boundary are responsible for conditioning the traffic entering the domain. Traffic conditioning involves functions such as packet classification and traffic policing. Traffic conditioning plays are essential to delivering differentiated services in DiffServ architecture. The boundary nodes rely on it to meter all the traffic entering the network against the customer's traffic profile. Inside the network, the core-routers are responsible for forwarding the packet according to the specified PHB. They carry the important function of resource allocation and packet scheduling with such algorithm as RIO[7], WRED[37] etc.. Figure 2.3 illustrates the architecture of the *differentiated services* with a single DiffServ domain.

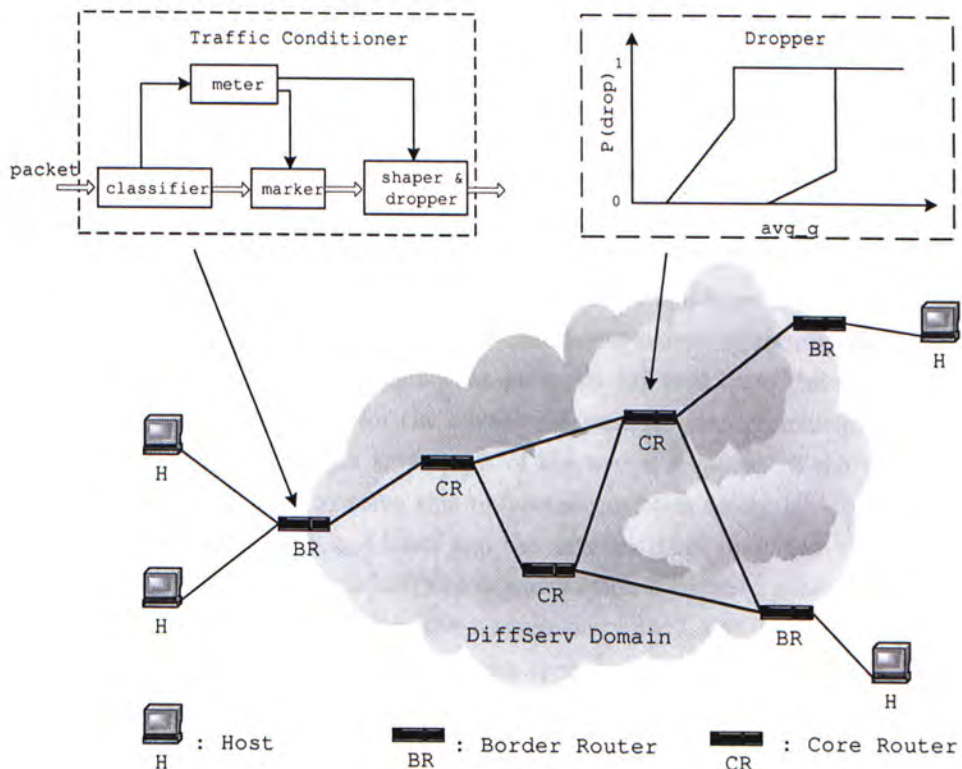


Figure 2.3: DiffServ Architecture

## Chapter 3

# The Fairness of TCP Vegas in Networks with Multiple Congested Gateways

In this chapter we investigate the performance of TCP Vegas in the network with multiple congested gateways. Our work focuses on the fairness between the multiple-hop connections and the cross traffic connections at each gateway. We find that TCP Vegas tends to bias against the multiple-hop connections. By analysis and computer simulations, we show that this bias is rooted in Vegas's RTT-based congestion avoidance mechanism. In other words, it uses the variation of round trip time as the tuning knob to adjust the congestion window. Due to the cumulative nature of round trip time, the congestion status of a connection cannot be correctly inferred from the measurement of round trip time. It is possible for the connection passing through multiple-hop path to receive a degraded throughput as a result of the wrong response to the congestion. We show that it is not easy to solve this unfairness problem solely by the end hosts, collaborative efforts from both end hosts and the network itself should be considered.

The rest of this chapter is organized as follows. Section 3.1 gives a general description of the problem. Section 3.2 describes the congestion control algorithm of TCP Vegas and some related works. In 3.3 we then give an approximate analysis of the TCP Vegas in a network with multiple congested gateways. In section 3.4 simulation results are presented to verify the analytical results. Finally, we summarize this chapter in Section 3.5.

### 3.1 Introduction

In today's Internet, it is common for data flows starting from a source to go over many hops before arriving at the destination. These multiple-hop flows need to compete with

the cross traffic at each gateway and are often treated unfairly due to their higher packet loss rate and longer end-to-end delay.

Several researchers have studied the bias against connections passing through multiple congested gateways in TCP/IP network. In [25], Mankin measured the traffic in a network with several congested gateways. He remarked that the longer connections are more likely to have a higher packets loss rate at the gateway and degraded throughput. In [10], similar result was obtained in a network with three congested gateways: longer connection receives just half of the available throughput at each gateway. In [35], it was found that as a connection passes through a path with  $n$  congested gateway, its throughput is limited by  $M \times p$ , where  $M$  is the maximum possible throughput in each congested gateway,  $p$  is the packet dropping probability given by  $p = \frac{1}{1+\sqrt{n}}$ .

The earliest studies on this multiple congestion problem were conducted on TCP Tahoe/Reno (and their variants) with emphasis on the packets loss statistics and its impact on the throughput of the connections. TCP Tahoe and Reno are well known for their congestion control mechanisms based on packet loss detection: they adjust the congestion window using a AIMD-based (Additive-Increase Multiplicative-Decrease) algorithm triggered by the packet loss. The analytical and simulation results in [35] also revealed that the bias in Reno comes from its loss based congestion control scheme: the overall loss probability of a connection is inversely proportional to the number of the congestion gateways passed.

TCP Vegas [23] employs a more sophisticated end-to-end congestion avoidance algorithm known as *delay-based congestion avoidance (DCA)*. In this paper we investigate the fundamental factors affecting the performance of TCP Vegas in multiple-hop networks. Our work shows that TCP Vegas has a bias against the multiple-hop flows and this bias problem is rooted in TCP Vegas's RTT-based congestion control mechanism: the cumulative nature of RTT makes it difficult for the Vegas source to adjust the window size correctly. In other words, it is reliable to judge whether the multiple-hop connection is undergoing multiple mild congestion or a single severe congestion with the measurement of end-to-end delay variance. As a result, the multiple-hop flows tend to be wrongly penalized even when the network is still under moderate congestion level, resulting in a degraded throughput and poor utilization of network resource.

## 3.2 TCP Vegas and related works

TCP Vegas augments the TCP Reno by employing a more sophisticated bandwidth estimation mechanism. It measures the round trip time and computes the difference between the expected and actual flows rates to estimate the available bandwidth in the network. The underlying idea is that when the network is not congested, the actual flow rate should be close to the expected rate. Under congestion, the actual rate should

be smaller than the expected rate due to the packet loss and increase of end-to-end delay. TCP Vegas uses this difference to estimate the congestion level in the network and adjusts the window size accordingly. The details of the algorithm are as follows:

1. TCP source computes the expected throughput, which is given by:

$$Expected = W/baseRTT \quad (3.1)$$

where  $W$  is the current window size and  $baseRTT$  is the minimum of all measured round trip time.

2. TCP source estimates the actual flow rate using the current round trip time  $RTT$ :

$$Actual = W/RTT \quad (3.2)$$

3. The difference between actual and expected flow rate is

$$\Delta = (Expected - Actual) \times baseRTT \quad (3.3)$$

$\Delta$  can be interpreted as the packet backlog in the path, which is a measure of path congestion level.

4. Based on  $\Delta$ , TCP source adjusts the congestion window size as follows

$$W = \begin{cases} W + 1 & \text{if } \Delta < \alpha \\ W & \text{if } \alpha \leq \Delta \leq \beta \\ W - 1 & \text{if } \Delta > \beta \end{cases} \quad (3.4)$$

where  $\alpha$  and  $\beta$  are two constant parameters. In this way, TCP Vegas tries to keep at least  $\alpha$  packets but no more than  $\beta$  packets in the queue. It tries to detect and utilize the extra bandwidth without congesting the network. This mechanism is fundamentally different from TCP Reno. Reno always increases the congestion window size as high as possible until packet loss. TCP Vegas, on the other hand, tries to avoid packet loss by adjusting the congestion window at a early time by measuring the variance of round trip time. The congestion window of TCP Vegas are much more stable than that of TCP Reno, which help TCP Vegas to utilize the network resources more efficiently.

Many researchers have studied the performance of Vegas from different aspects. Ahn et.al. [2] confirmed the claims in [23] that Vegas can achieve higher throughput than Reno with a lower packet loss rate. They also found that Vegas's higher throughput and lower delay come from its RTT based congestion avoidance mechanism. In [34] the problem of rerouting and persistent congestion and their impacts on the performance TCP Vegas were investigated. In [3] Mo showed that TCP Vegas does not discriminate against connection with long propagation delays. It shares the available bandwidth evenly between connections, regardless of their propagation delay.

In spite of the many results in favor of TCP Vegas, there are also many studies on the effectiveness of Vegas's RTT-based congestion sampling algorithm in predicting future packet loss events. Paxson [38] looked at the correlation between one way packet delay variation and loss. He concluded that loss is weakly correlated to the rise of packet delay and conjectured that the linkage between the two is weakened by routers with large buffer space. In [26] the study is on the delay-based congestion avoidance algorithm. They found similar results and presented a set of conjectures to explain their measured results. They also explored the effect of this algorithm on the throughput of TCP with the use of the little correlation between the increment of delay and packet loss. Their conclusion was that RTT-based congestion avoidance may not be reliably deployed incrementally.

### 3.3 Analysis

For a multiple-hop network, the round trip time of a connection is the accumulation of the propagation and queueing delays of all gateways along the path. Consider a connection with  $N$  gateways in its path (forward and reverse path). Assume that  $d$  is the round-trip propagation delay between the source and the destination,  $t_i$  is the average queueing delay in the  $i$ th gateway, then the round trip time  $RTT$  of this connection is

$$RTT = d + \sum_{i=1}^N t_i \quad (3.5)$$

To quantify the relation between the round trip time and backlog in each gateway, let  $n_i$  be the average number of packets backlogged in the  $i$ th gateway, and  $\tau_i$  be the average service time for a packet in this gateway. Then we have

$$RTT = d + \sum_{i=1}^N (n_i + 1)\tau_i \quad (3.6)$$

The second term above is the total time spent in the gateways by a packet, which is the sum of its waiting time and service time. Since the service time  $\tau_i$  is not relevant to the congestion status, we introduce a new term  $\tilde{d}$  that combines the propagation delay  $d$  and the total service time, as follows

$$\tilde{d} = d + \sum_{i=1}^N \tau_i \quad (3.7)$$

Substitute it into 3.6, we have

$$RTT = \tilde{d} + \sum_{i=1}^N n_i \tau_i \quad (3.8)$$



Suppose that at some time, the source obtains a backlog measurement  $\Delta$  as defined in 3.3 where  $baseRTT = \tilde{d}$  (For simplicity, we do not consider the overestimation of the propagation delay). Then from (3.1),(3.2),(3.3) and (3.8), we obtain

$$\left[ \frac{W}{\tilde{d}} - \frac{W}{\tilde{d} + \sum_{i=1}^N n_i \tau_i} \right] \times \tilde{d} = \Delta$$

upon simplification, we obtain

$$\sum_{i=1}^N n_i \tau_i = \frac{\Delta}{W - \Delta} \tilde{d} \quad (3.9)$$

If  $\tau_1 = \tau_2 \dots = \tau_n = \tau$ , the total backlog  $M$  along the path IS JUST

$$M = \sum_{i=1}^N n_i = \frac{\Delta}{W - \Delta} \frac{\tilde{d}}{\tau} \quad (3.10)$$

We now consider two cases:

- Case 1: single gateway congestion. Without lose of generality, assume that packets are mainly backlogged at the first gateway, so  $n_2 \approx n_3 \dots \approx n_N \approx 0$ . In this case

$$n_1 \approx \frac{\Delta}{W - \Delta} \frac{\tilde{d}}{\tau} \quad (3.11)$$

- Case 2: multiple gateway congestion. If gateways have different level of congestion, there is no uniform expressions for  $n_i$ s. All we can be sure is the backlog that is bounded by

$$\frac{\Delta}{W - \Delta} \frac{\tilde{d}}{N\tau} \leq n \leq \frac{\Delta}{W - \Delta} \frac{\tilde{d}}{\tau} \quad (3.12)$$

For the case where the backlogs are distributed uniformly in each gateway, we have

$$n_i \approx \frac{\Delta}{W - \Delta} \frac{\tilde{d}}{N\tau}, i = 1, 2 \dots N \quad (3.13)$$

From the perspective of individual gateway, the situation of case 1 is more serious than that of case 2. If the congestion is concentrated in a single gateway, a large backlog will be piled up in this gateway. If the burst of traffic is beyond the capacity of the buffer, it will lead to packet loss. It is therefore necessary for the senders to slow down their sending rates to relieve the congestion. While for the second case, the backlogs are distributed to all gateways. Even if  $M$  is very large, the backlog in individual gateways may not. Therefore, it may not be necessary to reduce the sending rates as it may leave the network under-utilized. In other words, the reduction of traffic is necessary in single congestion case but not in multiple congestion case.

If congestion status, i.e. case 1 or case 2, in the network can be determined, the source can response properly. Unfortunately, as shown by (3.10), the TCP Vegas source can only measure the cumulative backlog  $M$ . If the source finds that  $\Delta$  is larger than the upper threshold  $\beta$ , it will reduce the sending rate regardless whether the connection passes through "multiple mildly congested" path or "single severely congested" path. As a result, these multiple-hop connections are more likely to be wrongly treated.

### 3.4 Simulation Results

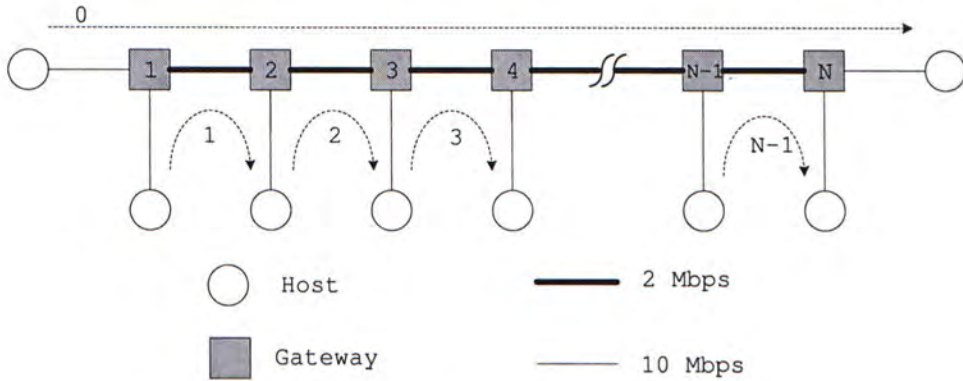


Figure 3.1: Network model

Figure 3.1 shows a network with  $N$  gateways and  $N$  connections. The link bandwidth is 10 Mbps between the host and the gateway and 2 Mbps between the gateway pairs. The dotted lines show the routing of these connections:

- connection 0 is a multiple-hop connection passing through all links
- connection 1 through  $N - 1$  are cross traffic connections passing through only one link

We assume that the propagation delay and transmission time are the same for all connections. This assumption is necessary to isolate the effects of queueing delay from the traffic phase effects as discussed in [14]. All the simulation experiments are run on NS-2 [1]. In the simulation each flow starts a FTP type traffic with unlimited data. The number of flows in each connection are varied in different simulation sessions to produce different congestion level at the gateways.

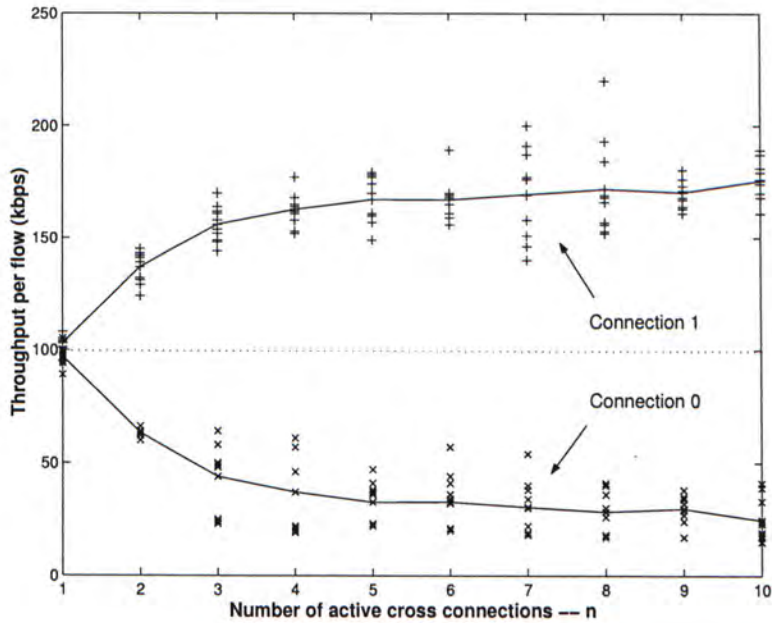


Figure 3.2: Throughput with different number of active cross connections

### 3.4.1 Throughput for different number of active cross connections

The first simulation is done using the network in Figure 3.1 with  $N = 11$ . The connections are either idle (no flows) or active (with ten flows). A series of simulation experiments are performed, each with  $n$  active cross connections ( $n$  varies from 1 to 10). Connection 0 is always active in all of these simulation experiments. All cross links have exactly 20 flows passing through, so the fair share of bandwidth for each flows is  $2/20$  Mbps (100 kbps). Figure 3.2 shows the measured throughput of all the flows in connection 0 (marked by "x") and connection 1 (marked by "+") for different number of active cross connections. The throughput of other cross connections (2 to 10) are similar to that of connection 1 and are therefore not shown. The solid line is the average throughput of these flows in each connection. It can be seen that when  $n=1$ , both connection 0 and connection 1 can achieve their fair share of bandwidth. But when  $n$  is larger, connection 0 needs to compete with the cross connection traffic at every gateway, so its throughput drops rapidly. Meanwhile, connection 1, being a FTP type traffic, can easily takes up the rest of the cross link bandwidth.

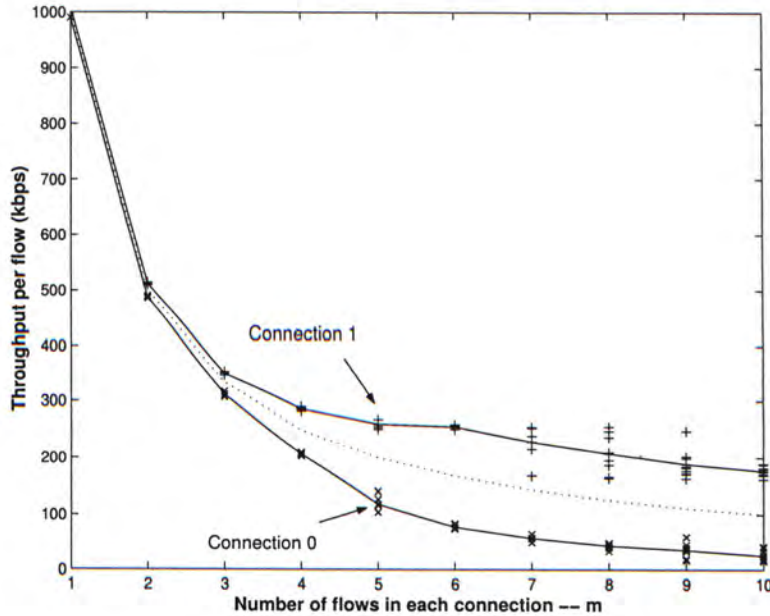


Figure 3.3: Throughput with different number of flows

### 3.4.2 Throughput for different number of flows in each connection

For the same network, we next consider the case where all connections are active and have the same number of flows  $m$ . We run the simulation ten times with  $m = 1, 2, 3, \dots, 10$  and show the throughput of flows in connection 0 and connection 1 in Figure 3.3. The dotted lines show the fair share of bandwidth for each flow, which decreases with the increase of number of flows  $m$ . It can be seen from the figure that when the network is lightly loaded (small  $m$ ), connection 0 can achieve a comparable throughput as connection 1 even though it has to pass through more gateways than connection 1. This is because the queuing delay in each gateway is small and so the round trip time is dominated by propagation and transmission delay. Since this is the same for all connections, connection 0 can achieve similar throughput as the cross connections. But when the network load increases (increasing  $m$ ), the throughput of connection 0 quickly drops below its fair share value.

### 3.4.3 Multiple congestion vs Single congestion

In the previous simulation experiments, we demonstrate the bias problem of TCP Vegas in the network with multiple congested gateways. The simulation results show clearly that the number of congested gateways and the traffic load in the gateway have a strong effect on the throughput of the multiple-hop connections. The analysis in section 3.3

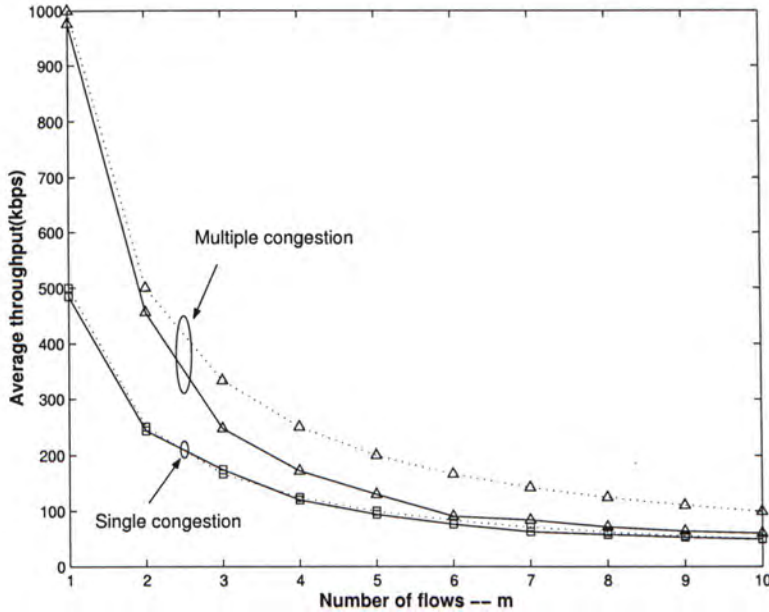


Figure 3.4: Throughput of connection 0 in two cases

shows that the sender of the multiple-hop connection cannot distinguish the multiple congestions from single congestion. As a result, the throughput of multiple-hop connection is reduced significantly.

We use the network with  $N=4$  to illustrate this phenomenon. We consider the following two cases as discussed in section 3.3.

- **Case 1:** Only connection 0 and connection 1 are active. Connection 0 and connection 1 consists of  $m$  and  $3m$  flows respectively,  $m$  varies from 1 to 10 in the simulation. This is the case of single gateway congestion.
- **Case 2:** All connections (0,1,2 and 3) are active and all have the same number of flows  $m$ . This is the case of multiple gateway congestion.

Figure 3.4 compares the average throughput of connection 0 in these two cases. The solid lines are the measured throughput in the simulation and the dotted lines are the fair share of bandwidth of each flow. The figure shows that in single gateway congestion, connection 0 can always receive its fair share of bandwidth under different traffic load conditions. But in multiple gateway congestion, the throughput of connection 0 drops below its expected value with the increase of flows in the network. In fact the throughput of connection 0 in multiple congestion case falls to the same level as that in single congestion case.

To show this more clearly, Figure 3.5 shows the average queue length of three gateways

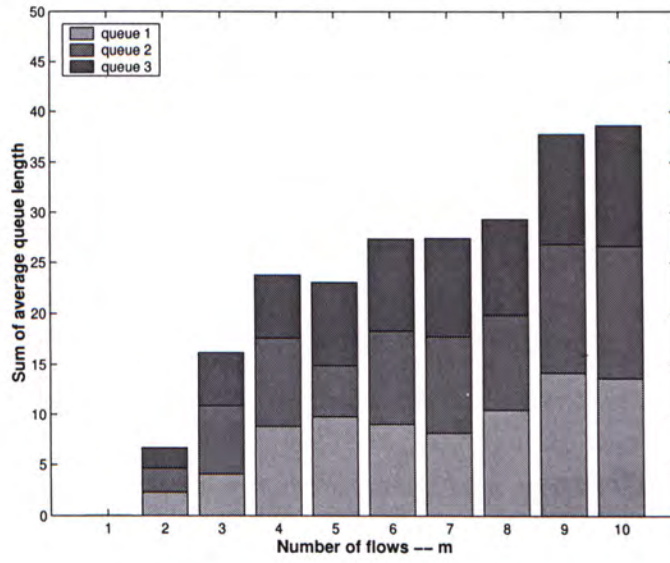
in these two cases. In multiple congestion case, three gateways have the similar queue length, while in single congestion case, only the first gateways has a large queue. An interesting result is, although the individual queue length under these two situations is different, the sum of these three queues are quite close in these two cases. In other words, the sender sees a similar accumulation of packet backlog along the path under two situations, so it takes similar actions to respond to the congestion. This explains why connection 0 achieves a similar throughput under two situations.

The results of Figures 3.4 and 3.5 confirm the analytic results in section 3.3: TCP Vegas is not able to distinguish the single gateway congestion from multiple gateway congestion. As multiple mild congestion is more likely to occurs than single severe congestion, the multiple-hop connections tend to be treated unfairly.

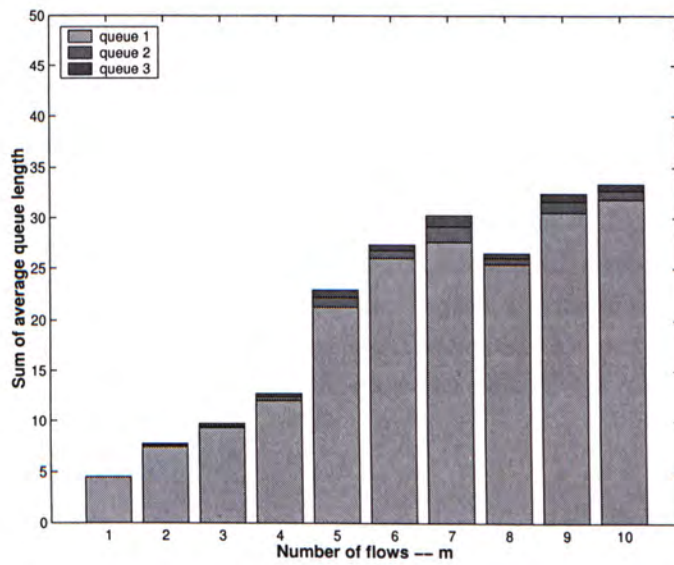
### 3.5 Summary

In this chapter we investigate the fairness problem of TCP Vegas in the network with multiple congested gateways. Using both analysis and simulation, we have shown that when TCP Vegas uses the round trip time to estimate the backlog in the path, it fails to distinguish the multiple mild congestion case from the single severe congestion case. As a result, it tends to degrade the throughput of multiple-hop connections by guiding the source to reduce the sending rates unnecessarily.

Due to the cumulative nature of end-to-end delay, the discrimination against multiple-hop connections is a common problem in all delay-based congestion avoidance schemes for which TCP Vegas is just one of the examples. We believe that it is not easy to solve this problem solely by the end hosts. The intermediate nodes in the network should also participate for a collaborative congestion control.



(a) Multiple congestion case



(b) Single Congestion case

Figure 3.5: Average queue length of three gateways

## Chapter 4

# The Joint Congestion Control for TCP/IP Networks

In the previous chapter we have studied the fairness issue of TCP Vegas in the network with multiple congested gateways. TCP Vegas employs the host-based congestion control mechanism and responds to congestion with the measurement of end-to-end delay variance. It has long been discussed as how much control this kind of host-based congestion control mechanisms can achieve with the limited information collected from the end host. In recent years, it is realized that the participation of routers will help to achieve a more effective control on the congestion. This understanding leads to much interesting on the research of router-based congestion control mechanisms. In this chapter, we propose a *Joint Congestion Control (JCC)* scheme for TCP/IP networks. JCC seamlessly unifies the efforts from routers (IP layer) and end hosts (TCP layer). It can provide more effective and accurate management in case of congestion in the network.

The rest of this chapter is organized as follows. Section 4.1 presents the motivation of JCC and describes the existing mechanisms. Section 4.2 describes the algorithm of JCC in details. Section 4.3 presents the simulation results and compares the performance of JCC with other TCPs. Section 4.4 summarizes the work of this chapter.

### 4.1 Background

The Internet was originally designed following the concept that all flow-related states should be kept at the end host[8]. As a result, most of the earlier congestion control mechanisms were implemented at the end hosts. In the absence of explicit information from the network, the end hosts treat the network as a black box and infer the network states from the flow's throughput, end-to-end delay, and packet loss statistics. Depending on the metrics used, the congestion control mechanism can be classed into two categories: "*Loss-based Congestion Avoidance (LCA)*" and "*Delay-based Congestion*



*Avoidance (DCA)*". TCP Tahoe, Reno and NewReno are LCA based mechanisms which take the packet loss as indication of network congestion. In case of packet loss, the TCP source backoff the congestion window to reduce the sending rate. It then increases the window until the next packet loss. DCA mechanism was firstly introduced by Jain [18] and adopted in TCP Vegas. It measures the variation of round trip time (RTT) at the source and adjusts the congestion window accordingly.

Host-based congestion control mechanisms are easy to implement as they do not need the support from routers. All the work are done at the end hosts with some simple measurements. This facilitates the decentralized resource allocation among different hosts and users. However, the accuracy and effectiveness of these mechanisms are limited by the information obtained from the end hosts. In recent years, with Internet evolving into the multi-purpose commercial infrastructure, it was found that host-based congestion control mechanisms were not sufficient to prevent the packet loss as a huge volume of traffics are injected into the network. It was recognized that the proper place to tackle the congestion problem is at routers: they know exactly how congested they are and can therefore perform more drastic resource management. This understanding leads to several router-based congestion control approaches, known collectively as *Active Queue Management*. *Random Early Detection (RED)*[15] is one of this kind of mechanisms. RED works by dropping packets before buffer overflow and allows router to decide when and how many packets to drop. Similar schemes such as BLUE [39] depends on packet loss and link idle events to manage congestion. *Explicit Congestion Notification (ECN)*[13, 31] promoted the interaction between hosts and intermediate nodes in a more explicit way. Here, instead of dropping packets, the router marks the packet with an ECN bit in the header and in this way conveys the congestion information to the end host for congestion control.

*Active queue management* mechanisms allow the routers to play a more active role in controlling the congestion in the network. In RED,ECN or other AQM mechanisms, the router measures the congestion level and decides whether drop(or marks) a packet to notify the source of the congestion. All of these mechanisms follow the same way in which the routers just tell the source that the network is "congested" or "not congested". With this coarse information, the TCP source simply back-off the sending window when it receives the "congested" notification. The congestion window of the TCP still exhibits the sawtooth behavior, which leads to a fluctuation in throughput. In this chapter, we propose the Joint Congestion Control (JCC) scheme for TCP/IP networks. JCC is designed to unify the efforts of both end hosts and network routers. With the close cooperation from network and end hosts, JCC can provide effective and accurate congestion control in a proactive way, eventually improve the utilization of network bandwidth with a lower packet loss and fair share of resource among different users.

## 4.2 The Joint Congestion Control

As for the burst nature of Internet traffic, the offered load on a link fluctuates from time to time. The congestion state of the link, therefore, varies in a wide range depending on the traffic load and capacity of the link. JCC aims to collect and use the quantitative information of the congestion level of the network. In JCC, special probing packets are sent out periodically from the source to the destination. The probing packet carries a congestion state field in the header. When the routers along the path detect these probing packets, they update the congestion state field of these packets with the local congestion information. As the probing packet arrives at the destination, this information is sent back to the source with the ACK. The source then uses this information to adjust the congestion window accordingly.

### 4.2.1 Path Load Reduction Factor

The congestion state information collected by the probing packet is the congestion level of the link. In JCC, the congestion level is defined as the packet dropping probability of the link, which is widely used in many active queue management mechanisms. Consider a path consisting of a set of links  $P$ , let  $r_i$  be the dropping probability of the  $i$ th link of this path. The congestion measure of this entire path should be a function of the congestion level of individual link along the path. We choose *Path Load Reduction Factor*  $R$  as a measure of the path congestion level, which is defined as

$$R \triangleq \max_{i \in P} \{r_i\} \quad (4.1)$$

This definition comes from the understanding that when a flow adjusts the sending rate, it will affect the state of all the links it traverses. In case of congestion, the flows should reduce their rates according to the requirement of the most congested link in order to reduce the traffic load of other links of the path as a side effect. Take the path consisting of five links as shown in Figure 4.1 as an example. The congestion level of each link is shown beside the link. It can be seen that currently the third link is the bottleneck link ( $r_3=0.1$ ) of this path, so  $R = r_3 = 0.1$  according to 4.1. Therefore if all flows at the source node reduce their rates by 10%, the congestion of the entire path can be relieved. While if the source reduces the rates according to other link's state, the reduction ratio is not enough to relieve the congestion problem on the third link.

$R$  is collected by the probing packet in a distributed way. At the source,  $R$  is initialized to zero. As the probing packet traverses through the network, the router updates  $R$  as follows

$$R^{new} = \max\{R^{old}, r\} \quad (4.2)$$

As the probing packet arrives at the destination,  $R$  is copied into the next outgoing ACK (denoted as probing ACK) and sent back to the source. The source then adjusts

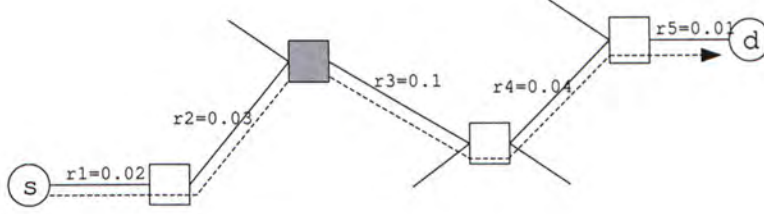


Figure 4.1: An example of a path with five links

the congestion window according to current state with the congestion control algorithm as discussed in the follows section.

## 4.2.2 The Congestion Control Algorithm

JCC still uses *slows start* and *congestion avoidance* as the congestion control mechanism and incorporates the *Path Load Reduction Factor R* into these algorithms. Besides *cwnd* and *ssthresh*, JCC maintains two other parameters  $\alpha$  and  $\beta$  for each connection as the lower and upper thresholds of  $R$ . The congestion control algorithm operates according to the stage that JCC is undergoing.

### Slow Start Stage

If the current window size *cwnd* is below the slow start threshold *ssthresh*, JCC is in the slow start stage. In this case, the congestion window increases exponentially as the other TCP protocol. This process continues until the source receives an  $R$  value (from the probing ACK) that is larger than the upper threshold  $\beta$ . As this happens, the source set *ssthresh* to the current window size and JCC enters the congestion avoidance stage.

### Congestion Avoidance Stage

In the congestion avoidance stage, JCC adjusts the congestion window with a **Additive-Increase Exponential-Decrease(AIED)** algorithm, which is defined as

$$cwnd = \begin{cases} cwnd + 1 & \text{if } R \leq \alpha \\ cwnd \times \exp[-\ln 2 \times \frac{R-\alpha}{\beta-\alpha}] & \text{if } \alpha < R < \beta \\ cwnd/2 & \text{if } R \geq \beta \end{cases} \quad (4.3)$$

The adjustment ratio as a function of  $R$  is shown in Figure 4.2. This algorithm operates at three different phases according to the value of  $R$

- congestion free phase: If  $R \leq \alpha$ , the path is in good condition. The congestion window is increased by one in each round trip time. This additive increment allows the flow to obtain the available bandwidth as much as possible.

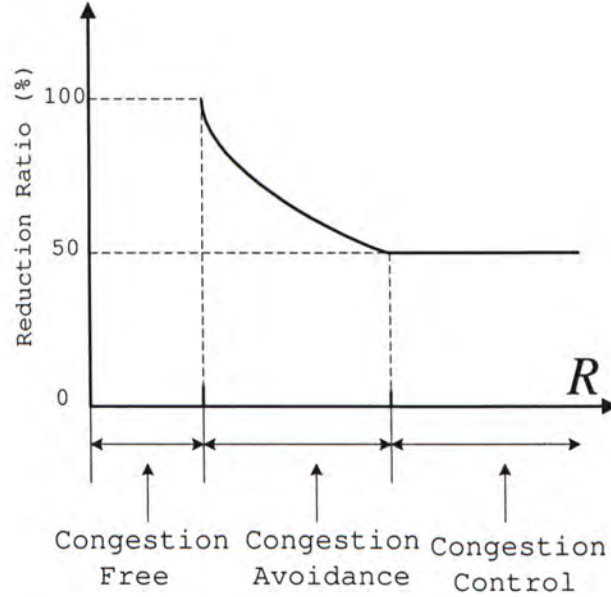


Figure 4.2: Reduction ratio of AIED algorithm

- congestion avoidance phase: If  $\alpha < R < \beta$ , the path is suffering moderate congestion and may have packet loss. In this case, the window size undergoes a mild exponential decrease as shown in Figure 4.2
- congestion control phase: If  $R \geq \beta$ , it strongly indicates that the path is in serious congestion and there is high probability of packet loss. In this case, the window size is simply cut to half to reduce the sending rate.

Figure 4.3 depicts the expected behavior of congestion window of JCC and TCP Reno. As the dashed line shows, TCP Reno increases the congestion window until the packet loss, then reduces the congestion window to half and increases again until another packet loss. This saw-tooth behavior leads to a large variation of the congestion window, as a result the throughput of TCP Reno fluctuates drastically. JCC, on the other hand, decreases the congestion window exponentially at an earlier time before the network is overwhelmed, so it can avoid the packet loss. When the network becomes available, the congestion window is increased linearly. In this way, JCC should be able to stabilize around the expect rate with less fluctuation than TCP Reno. This potentially reduce the total packet loss and help to improve the network utilization.

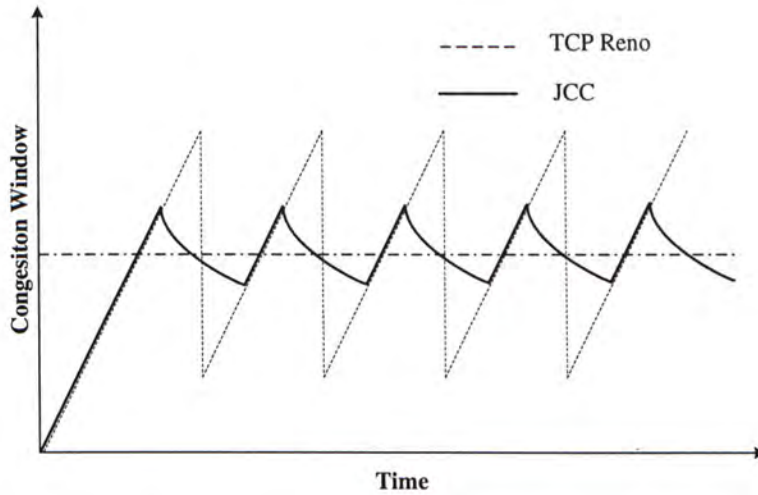


Figure 4.3: Congestion window behavior of JCC and Reno

### 4.2.3 Probing Interval

The probing interval is the time between two subsequent probing packets. It determines the frequency of window adjustment. The optimal probing interval should consider both the stability of JCC flow and the effectiveness of the congestion control. If adjusted too frequently, it may lead to oscillation of the congestion window and eventually adverse the performance. On the other hand, infrequent probing may lead to unresponsive behavior in case of congestion and unnecessary packet loss. It is especially harmful to short life flow such as HTTP traffic. As a trade-off, it was suggested that the adjustment should not be more than once per round trip time [24]. JCC follows this recommendation and set the probing interval to one round trip time.

In JCC, it is recommended that routers should not drop the probing packets due to their special function in JCC's congestion control mechanism. But in case of severe congestion, this may occur occasionally. Whenever this happens, the source simply cut the congestion window to half. The important thing here is to differentiate the probing packet loss from the normal packet loss. This can be done by recording the sequence number of the probing packets when they are sent out. In case of packet loss, the source can check whether sequence number of the lost packet coincides with the probing packet.

### 4.2.4 Parameter Setting

$\alpha$  and  $\beta$  are two important parameters that control the performance of JCC. The setting of these two parameters depends on the dropping probability function used in the routers and also on the desired aggressiveness of JCC. In our simulation experiments, we use

a modified RED algorithm where  $\alpha$  is set to zero, and  $\beta$  is initialized to  $max_P$ . These correspond to the lower and upper thresholds of the dropping probability in RED.  $\alpha$  is a fixed parameter.  $\beta$  is dynamically updated in run time. i.e. each time a packet is detected lost,  $\beta$  is set to the latest value of  $R$  from the probing ACK. In this way,  $\beta$  can always reflect the network congestion level corresponding to recent packet loss event. It is then used as a reference to control the upper bound of  $R$  until the next packet loss occurs.

#### 4.2.5 Encoding of R

$R$  is computed from the dropping probability  $r_i$ s, which is a real numbers between 0 and 1. In practice, it is undesirable to carry such a real number with the packet as for the scarce space in the IP packet header. JCC solves this problem by quantizing  $R$  to a number of scales and encoding it with some bits. Assume that  $m$  bits can be allocated for  $R$  in the header, with the binary coding,  $m$  bits can represent the number from 0 to  $2^m - 1$ .

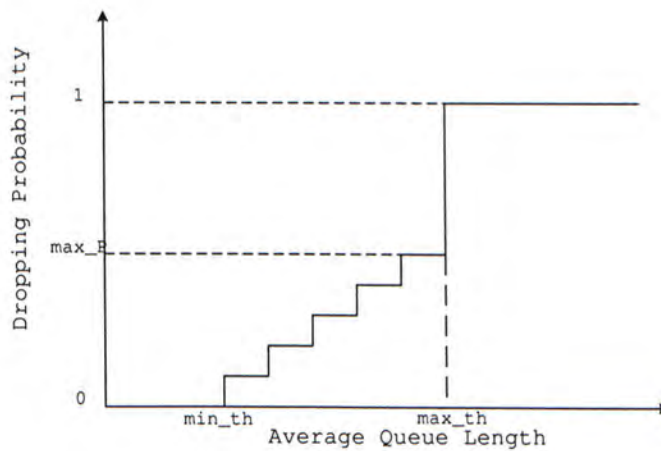


Figure 4.4: Modified RED algorithm

To work with this encoding scheme, we modify RED algorithm to provide scaled dropping probability. Figure 4.4 shows the modified algorithm. The original dropping probability in RED ranges from 0 to  $max_P$ , and then 1, In the modified algorithm, the probability of 1 is encoded as  $\overbrace{1111\dots11}^m$ . The probability from 0 to  $max_P$  is quantized to  $2^m - 1$  scales and encoded from  $\overbrace{0000\dots00}^m$  to  $\overbrace{1111\dots10}^m$ . The computation of dropping

probability is as follows

$$r = \begin{cases} 2^m - 1 & \text{if } avg_q > max_{th} \\ \lfloor \frac{avg - min_{th}}{max_{th} - min_{th}} \times (2^m - 1) \rfloor & \text{if } min_{th} \leq avg_q \leq max_{th} \\ 0 & \text{if } avg_q < min_{th} \end{cases} \quad (4.4)$$

where  $avg_q$ ,  $min_{th}$ ,  $max_{th}$  are the same variables as in RED which represent the average queue size, minimum threshold and maximum threshold for  $avg_q$  respectively.  $\lfloor \cdot \rfloor$  is the function rounding a real number to the next integer. Accordingly,  $\alpha$  and  $\beta$  are also set with this quantized value in the range of  $(0, 2^m - 1)$ .

### 4.3 Simulation Results

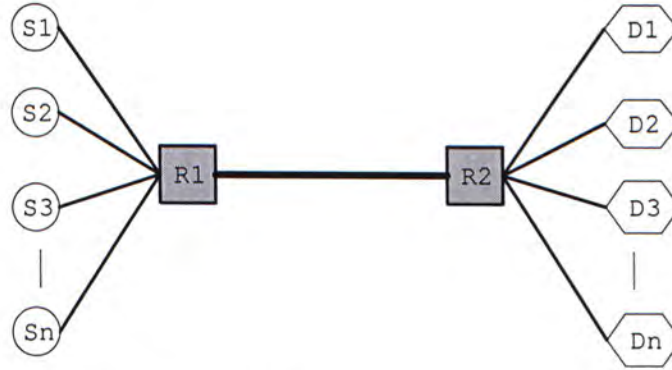


Figure 4.5: Simulation network

In this section we present simulation results to illustrate the performance of JCC from different aspects. As a comparison, we also provide the results of TCP Reno under the same conditions. All the simulation experiments are conducted on the network as shown in Figure 4.5 using NS-2[1]. All the flows in simulation use FTP type traffic with unlimited data.

#### 4.3.1 Congestion Window Behavior

Figure 4.6 shows the congestion window of JCC and TCP Reno. Unlike the congestion window of Reno that fluctuates drastically in a wide range (from 0 to 20), JCC stabilizes its congestion window in a narrow range (between 5 and 10). To show the behavior of congestion window more clearly, Figure 4.7 shows the microscopic view of Figure 4.6 in the time interval between 150s and 200s. It can be seen that the congestion window of TCP Reno clearly exhibits the saw-tooth behavior. It aggressively increases until the packet loss, then drop down and increase again up to the next packet loss. While JCC,

as expected, adjusts the congestion window in a more conservative way according to the network state, the variance of its congestion window is also smaller than that of Reno.

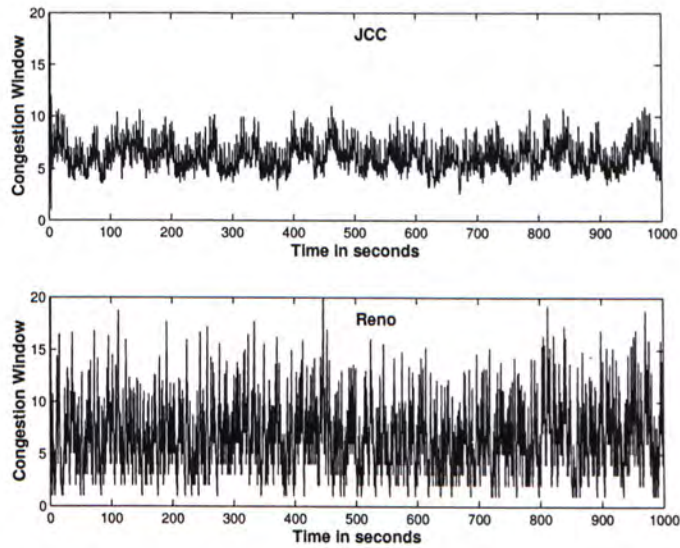


Figure 4.6: Congestion window behavior of JCC and Reno

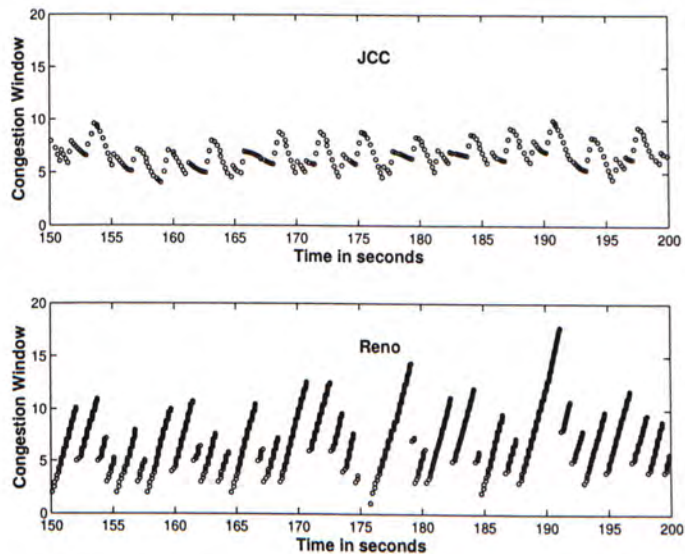


Figure 4.7: Microscopic view of congestion window of JCC and Reno



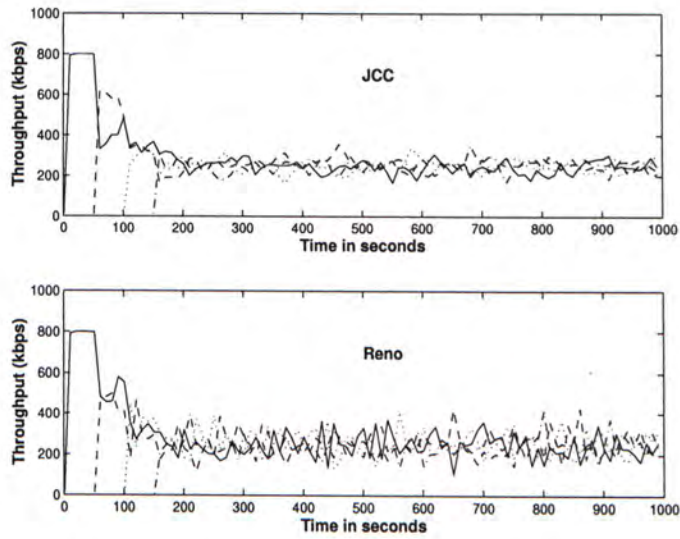


Figure 4.8: Throughput of four flows starting at different time

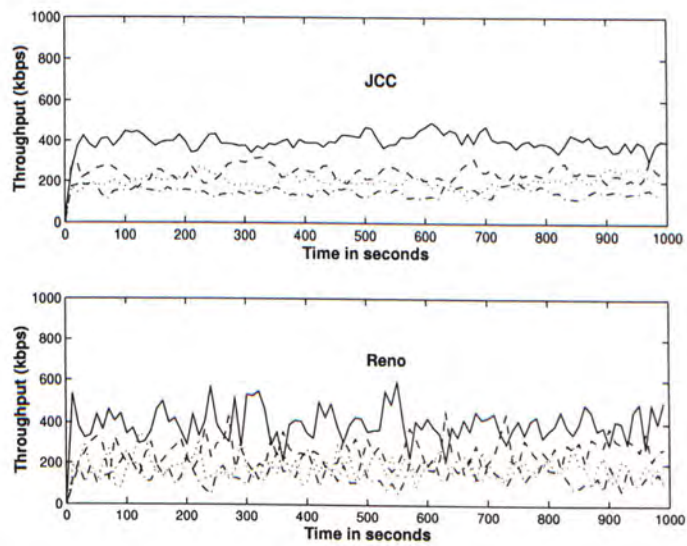


Figure 4.9: Throughput of four flows with different round trip time

### 4.3.2 Throughput Stability

Figure 4.8 shows the throughput of four flows competing at an 1Mbps link. These flows have the the same round trip time and start at  $t = 0, 50, 100$  and  $150$  seconds respectively. As a new flow joins in the network, the existing ones cut down their sending rates and obtain roughly the fair share of the bandwidth. Eventually, all four flows stabilize around 250 kbps, exploiting the full bandwidth of the link. As we have expected, with the fine-grained window adjustment, JCC flows interact better than the Reno flows and suffer from little variation of throughput after the stabilization.

Figure 4.9 shows the throughput of four flows with a round trip time of 50ms, 100ms, 150ms and 200ms respectively and starts at the same time  $t = 0$ . As this figure shows, JCC is still "unfair" to the flows with longer round trip time in the same way as TCP Reno, a problem reported in [24]. The flow with shorter RTT are more aggressive and obtains more bandwidth than its fair share, while the long-RTT flows are slow to respond to the congestion and fail to seize the available network bandwidth as the shorter ones. It has been suggested that this is an unavoidable problem in all "RTT self-clocked" congestion control mechanisms based on the feedback of ACK. We leave this for future discussion for improvement of JCC performance.

### 4.3.3 Packet Loss Ratio

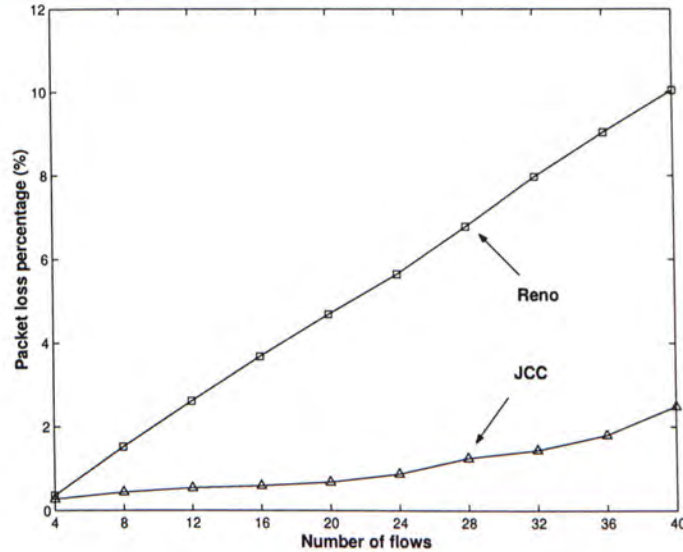


Figure 4.10: Packet loss with different number of flows

Figure 4.10 shows the packet loss ratio for different number of flows in the network. Although the packet loss ratio of both JCC and Reno increases gradually with the number

of flows in the network, JCC sees a much lower packet loss rate than Reno. For example, when there are 40 flows in the network, the high competence can result in as high as 10 percent packet loss in Reno, while this is just about 2 in JCC. This shows the main advantage of JCC over Reno in controlling the packet loss rate.

#### 4.3.4 Fairness Index

One observation in previous simulation experiment is that the JCC flows can fairly share the bandwidth. In this example we illustrate this property using the fairness index defined by Chiu and Jain [6]. Specifically, if  $x_i$  is the resource allocated to flow  $i$ , the fairness index  $f$  is defined as:

$$f \triangleq \frac{(\sum_{i=1}^n x_i)^2}{n \sum_{i=1}^n x_i^2} \quad (4.5)$$

$f$  ranges from 0 to 1 and is maximum when all users receive the same allocation of resource.

Figure 4.11 shows the fairness index of four flows starting at time  $t=0$ . As seen from the plot, JCC maintains fairness much better than Reno.

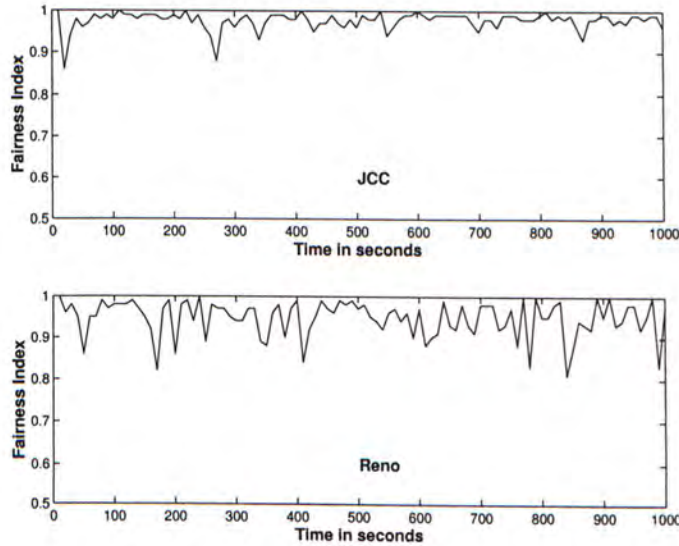


Figure 4.11: Fairness index of competing flows

#### 4.3.5 Fairness in Multiple-hop Network

The *path load reduction factor*  $R$  reflects the congestion level of the most congested node along the path instead of the cumulative statistics of the entire path as TCP Vegas or TCP Reno. Using it as the tuning knob for the congestion control, JCC is expected to improve the fairness for flows passing through multiple-hop network. To show this

property, we use the same multiple-hop network as shown in Figure 3.1 and the same simulation experiments with JCC. Figure 4.12 shows the throughput of connection 0 and connection 1. Compared with the Vegas performance as shown Figure 3.3, JCC can achieve a fairer throughput between multiple-hop flows and single-hop flows. In addition, the throughput of the multiple-hop flows is insensitive to the number of hops it passes.

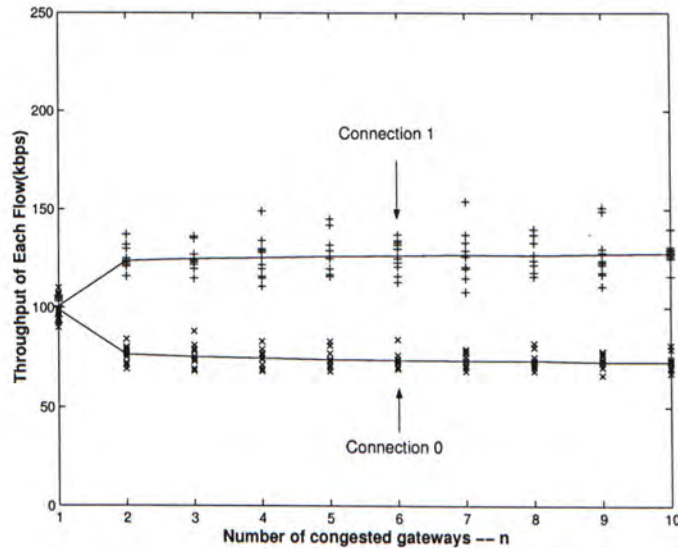


Figure 4.12: Throughput for different number of active cross connections

#### 4.3.6 Parameter Sensitivity

$\alpha$	R(mean)	Packet Loss Ratio (%)
0	0.0057	0.2574
0.02	0.0235	1.4004

Table 4.1: Average R and total packet loss for different  $\alpha$

$\alpha$  and  $\beta$  control the performance of JCC. As we have explained before,  $\beta$  can be updated in real time, so in this sub-section we focus on JCC'S sensitivity on  $\alpha$ . According to (4.4), when  $R$  is smaller than  $\alpha$ , the congestion window increases linearly. When  $R$  is larger than  $\alpha$ , the congestion window decreases exponentially. So in the steady state, JCC should control  $R$  around  $\alpha$ . In this sense,  $\alpha$  determines JCC's aggressiveness and packet loss tolerance degree.

Figure 4.13 and 4.14 illustrate the behavior of JCC for two cases where  $\alpha$  is set to 0 and 0.02 respectively. In these two figures, the top curve shows the throughput of

four flows, the bottom one shows the value  $R$  collected by the probing packet. Table 4.1 shows the average value of  $R$  and total packet loss ratio in these two cases. It can be seen that the throughput and the packet loss ratio of JCC flow is largely affected by the value of  $\alpha$ . With a larger  $\alpha$ , JCC becomes more aggressive, leading to more bursty throughput and higher packet loss rate.

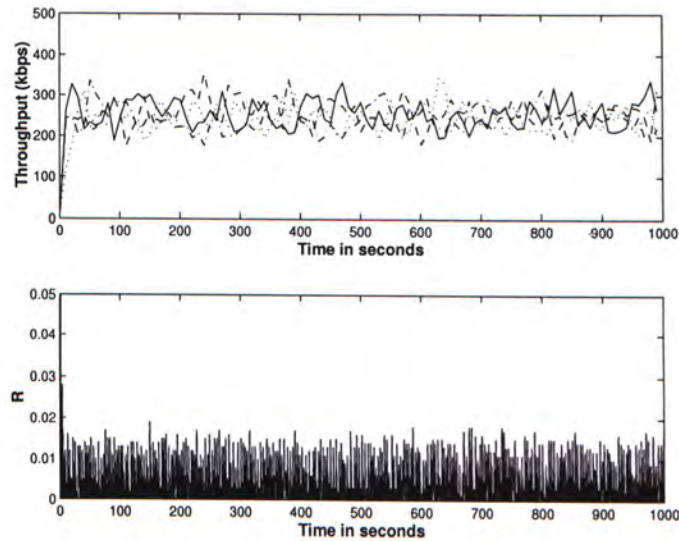


Figure 4.13: Throughput and Path Load Reduction Factor ( $\alpha = 0$ )

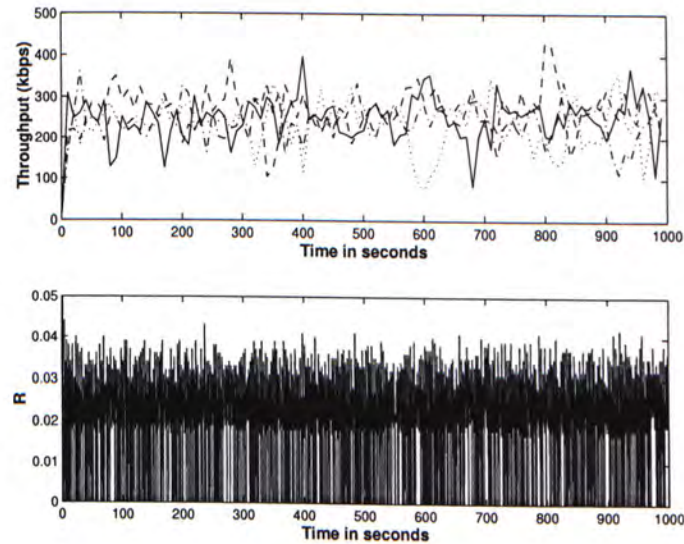


Figure 4.14: Throughput and Path Load Reduction Factor ( $\alpha = 0.02$ )

### 4.3.7 Interaction between JCC and Reno flows

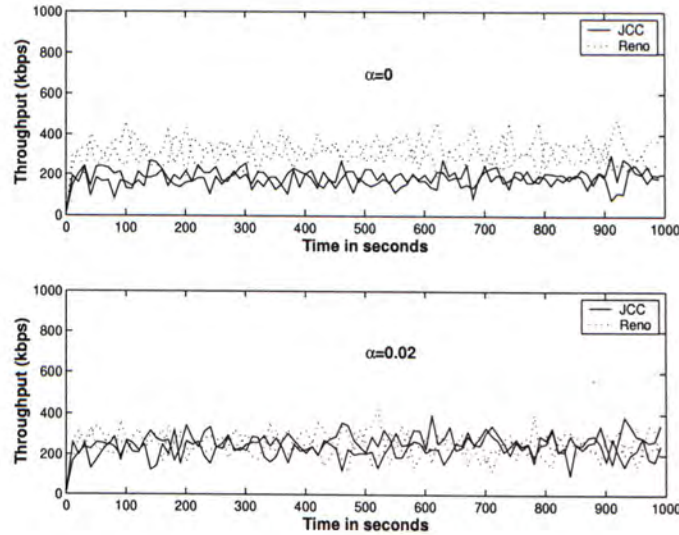


Figure 4.15: Throughput of mixed JCC and Reno flows

In a homogeneous environment where there are just JCC flows in the network, it is better to set a smaller  $\alpha$  value to avoid burst throughput and high packet loss rate. But in a heterogeneous environment where JCC and Reno flows co-exist in the network, as JCC's congestion control scheme is more conservative than that of Reno, it is necessary to tune the value of  $\alpha$  of JCC to avoid the bias problem between JCC and Reno flows. Figure 4.15 shows the case that two JCC and two Reno flows compete for the same bottleneck link. The top figure is the case that  $\alpha = 0$  and the bottom one with  $\alpha = 0.02$ . It can be seen that as  $\alpha$  is set to 0, JCC flows fail to get their share of bandwidth as this setting is too strict. But if  $\alpha$  is set to 0.02, JCC achieves a similar aggressiveness as Reno and can compete with Reno in a fair way.

The simulation results in this section show that we can tune the parameter to make JCC and Reno interact well in a heterogeneous environment. But in practice, the proper choice of  $\alpha$  is different depending on the environments and needs further investigation.

## 4.4 Summary

In recent years, many people have realized that the participation of network routers will help to increase the accuracy and effectiveness of congestion control mechanism. In this chapter we propose the *Joint Congestion Control* scheme to unify the effort of end hosts and intermediate nodes to provide cooperative congestion control. JCC uses the probing packet to collect the congestion state from the routers along the path. The source then

uses this information to adjust the congestion window with fine-grained scale. With proactive tuning, JCC can make efficient bandwidth usage with a lower level of packet loss rate. Our simulation results show that JCC achieves better performance than Reno in terms of stability, loss rate and fairness.

## Chapter 5

# *S*-WTP : Shifted Waiting Time Priority Scheduling for Delay Differentiated Services

The *Delay Differentiated Service* has been proposed as a *DiffServ* model to provide quality of service (QoS) on the Internet. In this model, the scheduler schedules the packet transmissions according to some specific delay metrics. *Waiting Time Priority (WTP)* is one of this kind of scheduling algorithms that uses the waiting time of the packet as the priority value to schedule the packet transmission. But due to the computational complexity, it is impractical to implement the WTP algorithm in high speeds network. In this chapter, we propose a modification algorithm based on WTP called *Shifted Waiting Time Priority (S-WTP)*. SWTP preserves the fundamental properties of WTP and reduces the computational complexity from  $O(n)$  to  $O(\log(n))$ . Simulation results illustrate that S-WTP preserves most of the functionalities of WTP.

The rest of chapter is organized as follows. The next section gives a brief description of *Delay Differentiated Services* model. Section 5.2 introduces some previous works on scheduling algorithms that aim to provide *Delay Differentiated Service*. In section 5.3, we describes the details of S-WTP and discuss its computational complexity. In section 5.4, the performance of S-WTP is reported with computer simulations from different aspects. Section 5.5 summarizes this chapter.

### 5.1 Introduction

To support quality of service over the Internet, the *Differentiated Services (DiffServ)*[4, 28] model has been proposed by IETF. In this model, flows with similar qualities of service (QoS) requirements are aggregated into classes. Rather than providing end-to-end performance guarantee for individual flows as in InterServ model, *DiffServ* aims at



differentiating different classes of traffic using per-hop packet forwarding mechanisms. Two popular approaches for realizing DiffServ are *absolute differentiated services* and *relative differentiated services*.

The goal of *absolute differentiated service* is to achieve a performance similar to those used in the Integrated Service model without keeping per-flow state in the routers. For example, *Expedited Forwarding Service* [29] aims to offer users a performance level that is similar to a leased line, as long as the user's traffic is within the scheduled profiles. In *Assured Forwarding Services* [7], packets are separated into four classes (AF1x–AF4x), each class has three drop precedence levels (e.g. AF11–AF13). As congestion occurs, packets of higher drop precedence level are dropped with a higher probability than the lower precedence one. This is a way to provide some level of absolute service to different classes.

*Relative differentiated service*, on the other hand, attempts to guarantee that the performance of the higher classes will be better or at least no worse than lower classes with fixed quality spacing. A prominent scheme for relative differentiated service is proposed in [11] called *Proportional Differentiated Service*, which aims to provide proportional performance spacing between classes rather than absolute spacing. Two principles were also proposed for both users and network operators to design and evaluate the service model. First, a model must be *predictable*. i.e. the differentiation should be consistent (the service received by higher classes should be better or at least no worse than that of lower classes) and the differentiation should be independent of class loads. Second, the mode must be *controllable*. i.e. the network operator should be able to control the appropriate level of spacing between classes.

Depending on the quality metrics used, the differentiated service model also can be classified as *Delay Differentiated Service(DDS)* or *Loss Differentiated Service(LDS)*. The *Delay Differentiated Services* is suitable for some delay-sensitive applications such as IP-telephony and video-conferencing. In this model, the router schedules the forwarding of the packet according to the specific delay metrics of different classes. *Loss Differentiated Service*, on the other hand, focuses on the packet loss statistics and aims to provide different level of packet loss guarantee to different classes. The work in this chapter focus on the *Delay Differentiated Service*.

## 5.2 Scheduling Algorithms for Delay Differentiated Services

Scheduling algorithms that aim to provide relative delay differentiated service have been studied extensively in literature. In [22] the authors proposed a dynamic packet transmission priority discipline called *head-of-line with priority jumps(HOL-PJ)*, the funda-

mental principle of HOL-PJ is to give transmission priority to the packet having the largest queuing delay in excess of its delay requirement. An explicit priority is assigned to each traffic class. The server always chooses the packet for service at the head of the highest priority nonempty queue. Unlike HOL, however, the priorities of the packet is increased using *priority jumping(PJ)* mechanism as their queuing delay increase relative to their delay requirement. The limitation of this scheme is that delay differentiation function may become invalid under heavy traffic load condition, as most of the packet from the low priority may jump to the higher class when they have been delayed for a sufficient longer time than it expect. As a result, the differentiation between classes is weaken when most of the packets have jumped to the same queue.

Weighted fair queueing(WFQ) is a generic service discipline that widely used to distribute the link bandwidth between classes. In [21] a dynamic WFQ was proposed which adjusts the weight of each class dynamically according to the arrival rate and experienced queue delay of each class so that the delay difference between classes can be well controlled. However, the main difficult of this scheme is to determine the optimal interval to measure the arrival rate and queue delay, so it may fail to provide consistent differentiation in relatively short timescales, because the forwarding resource allocated to each classes may not be able to catch up with the class load variations.

Backlog-Proportional Rate(BPR) scheduler is proposed in [11]. The basic idea of BPR is that the traffic load and service rate can be deduced from the backlogs of each class. In other words, if a certain class has received a small amount of service relative to the amount of arrivals in the recent time interval, then that class tends to have a relatively larger backlog. So the scheduler can adjust the scheduling parameters according to the backlog. It is observed that BPR exhibits a sawtooth-type of variations in the queuing delay which make it deviate quite often from the delay differentiation ratio in short timescale.

Another algorithm proposed in [11] is *Waiting Time Priority (WTP)*, which was originally studied by L. Kleinrock [19] under the name of Time Dependent Priorities. WTP is a non-preemptive scheduling algorithm with a set of parameters  $\{b_1, b_2, \dots, b_N\}$ , where  $N$  is the total number of class,  $b_r$  is the priority weight of class  $r$ . For a tagged packet of class  $r$  arriving at time  $\tau$ , its priority value at time  $t(t \geq \tau)$  is its *weighted waiting time*  $q_r(t)$ , which is calculated from

$$q_r(t) = b_r(t - \tau) \quad (5.1)$$

Figure 5.1 shows the priority function of WTP in the case of two classes. Whenever the server is ready to transmit a packet, the scheduler computes the priority value the HOL (head-of-the-line) packet of all classes and chooses the packet with the highest priority value for transmission

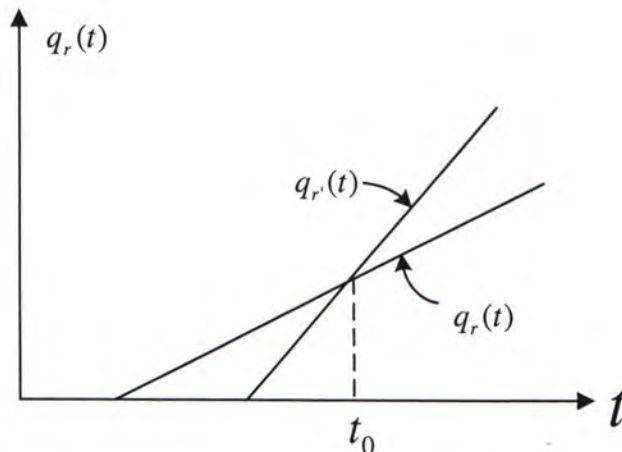


Figure 5.1: Priority function of WTP.

One of the desirable properties of WTP is that it is able to provide effective delay differentiation between classes in short timescales. But it is also found that this delay spacing depends not only on the parameter sets  $\{b_1, b_2, \dots, b_N\}$ , but also on the offered traffic load. Analytical and simulation results have suggested that only in heavy load condition can WTP scheduler approximate the proportional delay differentiation model according to the configured parameters. How to tune the  $b_r$ s to cope with the dynamics of traffic load is a complicated problem. In [20], an algorithm was presented for finding WTP control parameters to realize a set of specified queuing delay spacing. In [12], an iterative procedure to optimize the scheduling parameters was proposed. Although both algorithms can achieve the desired performance, they are computational intensive and both need to track the frequent change of traffic load and adjust the parameters accordingly. As for the burst nature of Internet traffic, these schemes are often not feasible.

In our work, we focus on the computation cost of WTP. We find that in WTP, the priority value of the HOL packet in each class is computed at the time of each transmission. So for a scheduler with  $N$  priority classes,  $N$  subtractions and  $N$  multiplications are needed to compute the priority of the HOL packet of all classes, then another  $N - 1$  comparisons are needed to find the highest priority class. So totally it needs  $3N$  computations to determine the candidate for each transmission. Even though the number of class may not be too large (4 to 8 in practice), so many computations are still a burden to the router especially in high speeds network where efficient forwarding is important. Our work aims to address this problem. We propose a modified algorithm based on WTP which reduces the computational complexity of WTP without losing the basic functionalities of WTP.

### 5.3 Shifted Waiting Time Priority Scheduling

*S-WTP* still uses the waiting time of the packet as the priority value. It differs from *WTP* in that the priority of the HOL packet can be determined in advance instead of at the time of transmission.

Before explaining the details of *S-WTP*, let's take a look at the packet transmission sequences of a class in *WTP*. Let  $\tau_r^{(i)}$  be the arrival time of the  $i$ th packet  $C_r^{(i)}$  in class  $r$ . Then at time  $t$ , according to (5.1), its priority value is  $q_r^{(i)} = b_r(t - \tau_r^{(i)})$ . The priority value of the HOL packet is computed in this way upon the time of transmission, and the class with the largest value is chosen to transmit the packet.

It can be seen that *WTP* has two characters in its scheduling procedure. First, the transmission candidate can not be determined in advance, as the priority value of the HOL packet depends on its actual waiting time, which can just be determined at the time of transmission. Second, for a scheduler with  $N$  class, the computation efficiency is just  $\frac{1}{N}$ , as each time just one class is chosen for transmission. For other  $N - 1$  failed classes, their computation results are obsolete at the next transmission and need to be computed again. These characters are not desirable in high speeds networks where efficient scheduling is needed.

*S-WTP* solves this problem using a simple modification to *WTP*. Instead of computing the priority value of all the classes on each transmission, *S-WTP* uses the departed packet's actual waiting time as the priority value of the next HOL packet of its class. In other words, as packet  $C_r^{(i)}$  is chosen for transmission, its actual weighted waiting time  $q_r^{(i)}$  is computed. This value is then "shifted" as the priority value of the next packet  $C_r^{(i+1)}$  of the same class. Similarly, when  $C_r^{(i+1)}$  leaves,  $q_r^{(i+1)}$  is used as the priority of  $C_r^{(i+2)}$ , etc. The underlying idea of this scheme is, if a packet from a class has waited longer than it should, that class is compensated by awarded a higher priority value to the next packet in queue. On the other hand, if a packet from has a shorter than expected waiting time, in order to preserve the average waiting time ratio, the next packet in the same class is given a smaller priority value(hence a longer delay). Another desirable feature of this scheme is that a departed packet's actual waiting time is a measured value, therefore the shifted priority value of next packet, once assigned, remains unchanged, the repeated priority computation is not needed.

In *S-WTP*, packets are timestamped upon their arrival and put into the corresponding class. Two operations are performed independently to update the priority value of each classes.

### 5.3.1 Local Update

*Local Update* is triggered whenever a packet departs. The scheduler computes the exact weighted waiting time of this packet according to (5.1). This value serves as the priority value of the next packet in the same class, called *shifted priority*  $P_s$ . Next time when the router is ready to send packet, the scheduler chooses the class with the largest  $P_s$  to transmit. Then same procedure is performed for the transmitted class.

### 5.3.2 Global Update

Using *Local Update* scheme, the priority value of the HOL packet remains unchanged until the departure of the packet. This may cause starvation in some special situations. Suppose that a packet in class  $C$  is forwarded very quickly, as a result, the next packet in class  $C$  will receive a very small priority value. Then regardless of how long this packet has waited, its priority value is never updated. If at the same time, there are a burst of packets arriving in other classes, their priority values are larger than that of class  $C$ , this will force the packets in class  $C$  to wait for an unnecessary long time than it should do until its priority value is larger than any others. This starvation can persist for an arbitrary long time depending on the size of the bursts in other classes. In the end it will lead to large deviation of delay ratio between the classes.

This starvation can be avoided if the class  $C$  gets updated at a proper time. S-WTP employs *Global update* to achieve this by refreshing the priority value of each class in a round robin way. Specifically, after each *Local update*, the scheduler selects one of other class and update its shifting priority  $P_s$  with the exact waiting time of its current HOL packet. In other words, let  $P_c$  be the current weight waiting time of the HOL packet, if  $P_c$  is larger than  $P_s$ , then substitute  $P_s$  with  $P_c$ . The underlying reason is that  $P_s$  reflects the waiting time of the previous departed packet, if current packet has waited a longer time than its precedent, then we should use its actual waiting time as the priority value to make it more competitive. The *Global update* is independent of *Local update* as the updating class is chosen in a round robin order.

### 5.3.3 Computational overhead

In terms of computational complexity, S-WTP requires one subtraction and one multiplication to compute the shifted priority value in each *Local Update*. This shifted priority value can be stored in a sorted array. A new shifted priority value should be inserted into the proper position of the list, this can be done using *binary insertion* with a computational complexity of  $\log_2 N$ . The computational overhead of *Global Update* is the same as *Local Update*, so for S-WTP, the total computation requirement is  $2 * (2 + \log_2 N)$  operations for each transmission. More importantly, unlike WTP that computes the priority

value of each class at the time of transmission, all these computations have been done before the decision of transmission candidate, no extra computation delay is involved at the time of transmission.

## 5.4 Simulation Results

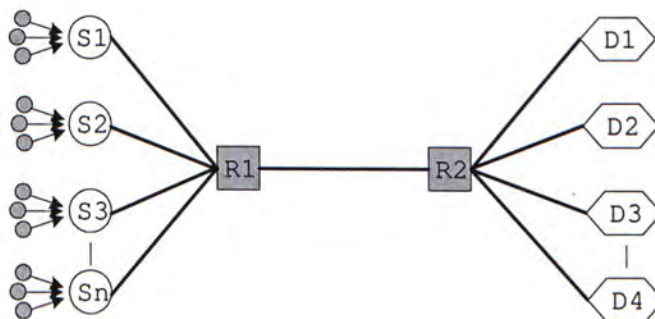


Figure 5.2: Simulation network

S-WTP inherits the fundamental scheduling principle from WTP, we expect it to approximate the functionality of WTP as close as possible. In this section, we conduct a sets of simulations to examine the effectiveness and efficiency of S-WTP. Our simulations are conducted with NS-2 using the network as shown in Figure 5.2: a number of connections share the link between R1 and R2. Each connection consists of several flows which belong to different classes. The packets in each classes are mixed with the similar rules: 40% of 100 bytes, 40% of 500 bytes and 20% of 1000 bytes. Each flow generates the packets following a Pareto distribution with a shape parameter  $\alpha = 1.9$ . The scheduling algorithm (S-WTP and WTP) are implemented in R1 to scheduling the packets on the link between R1 and R2.

In these simulation experiments, the delay ratios between classes are measured in successive time windows of every  $K$  packet departures.  $K$  determines the timescale in which we measures the class delays and compute their ratio, which varies from tens to thousands packet units depending the situation.

### 5.4.1 Microscopic View of Individual Packet Delay of S-WTP and WTP

Figure 5.3 shows an instance of packet transmissions with three classes. Each mark in this figure is the queuing delay of an individual packet in the unit of packet transmission time at the time of its departure from the queue. As observed in this figure, there is no notable difference between these two scheduling algorithms in achieving the delay

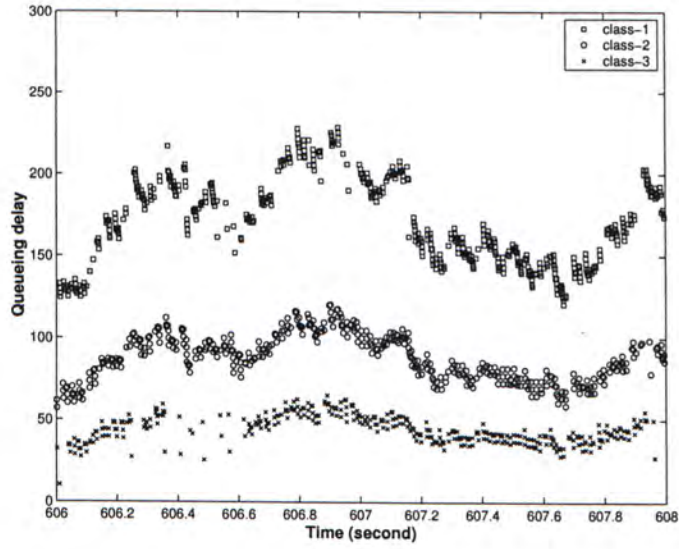
differentiation at the packet level. Similar to WTP, S-WTP also guarantees that the higher classes always get smaller delay than the lower classes. The queueing delay of individual packet ranges from tens to hundreds of packet transmission time from lower class to higher class.

#### 5.4.2 Delay Ratios in Different Timescales

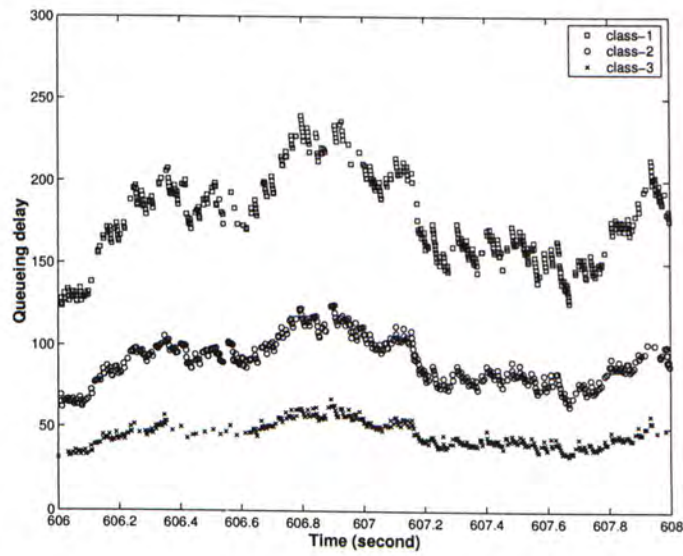
Figure 5.4 compares the achieved delay differentiation with S-WTP and WTP in different timescales. The delay ratios are measured in four different timescales, corresponding to  $K=500, 1000, 5000$ , and  $10000$  respectively. The figure shows five percentiles of the distribution of delay ratios between consecutive classes, which are median delay ratio (50th percentile), the 25th and 75th percentiles, as well as two tail delay ratios (5th and 95th percentiles). The target ratio is 2.0 and offered traffic load is 95%. As this figure shows, the distribution of achieved delay ratios with S-WTP are quite close to that of WTP. In the short timescale, the delay ratios fall in a wide range from 1.6. to 2.3. With the increase of measurement timescale, the delay ratios converge gradually to a narrow range from 1.8 and 2.0 in case of  $K=10000$ , which are quite close to the target ratios.

#### 5.4.3 Effects of aggregate traffic and class load distribution on delay ratio

Figure 5.5 shows the overall delay ratios of three classes under different traffic load condition. The effects of traffic load distribution in each class are also taken into the consideration. We test three typical cases: the heaviest traffic in the lowest priority class ( $\rho_1 : \rho_2 : \rho_3 = 60 : 20 : 20$ ), uniform traffic load in three classes ( $\rho_1 : \rho_2 : \rho_3 = 33.3 : 33.3 : 33.3$ ) and the heaviest traffic in the highest priority class ( $\rho_1 : \rho_2 : \rho_3 = 20 : 20 : 60$ ). In each case, the total traffic load varies from 0.65 to 0.95 while the distribution in each class remains unchanged. The target delay ratio is still 2.0. Seen from these figures, S-WTP achieves close results as WTP under almost all situations. Although in some cases with moderate traffic load, the achieved delay ratios with S-WTP have a little deviation from that of WTP. But this deviation decreases gradually with the increase of the traffic load. Under heavy traffic load condition, the achieved delay ratios with S-WTP and WTP both approach to the target ratio.



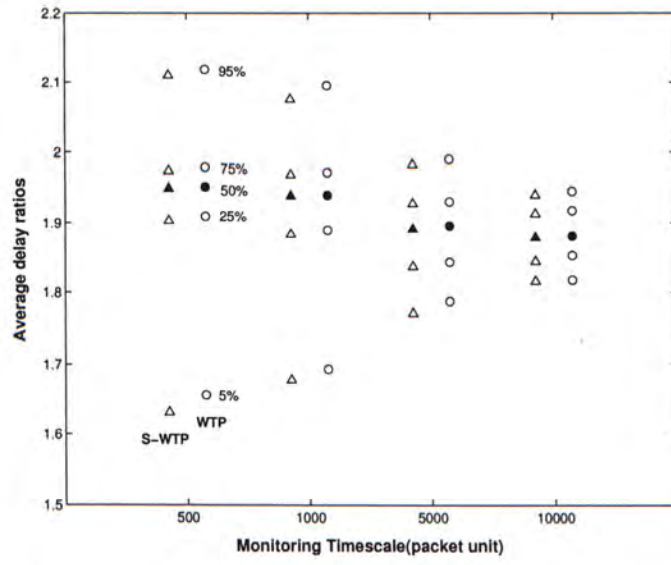
(a) S-WTP



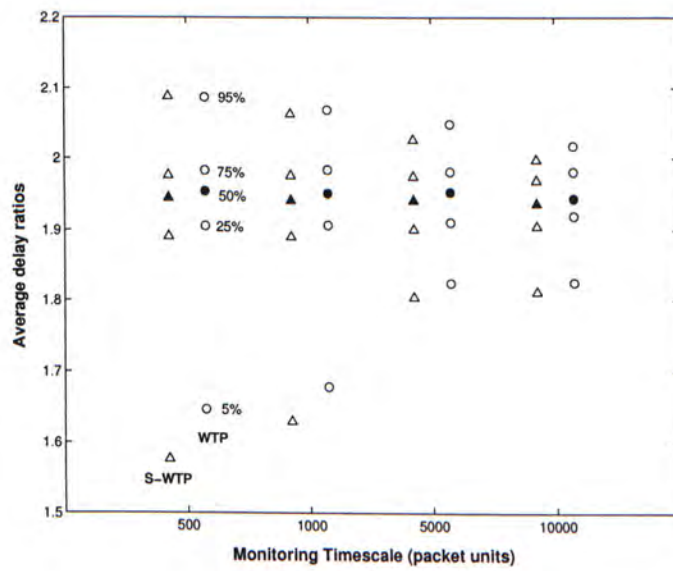
(b) WTP

Figure 5.3: Individual packet queuing delay with S-WTP and WTP)



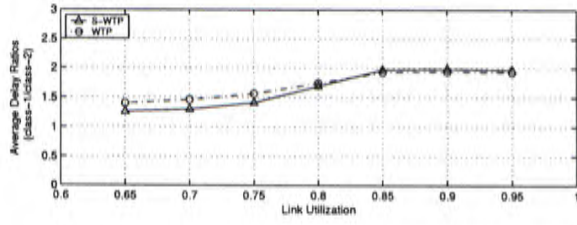


(a) class-1/class-2

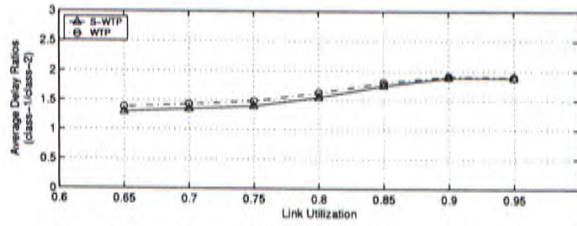
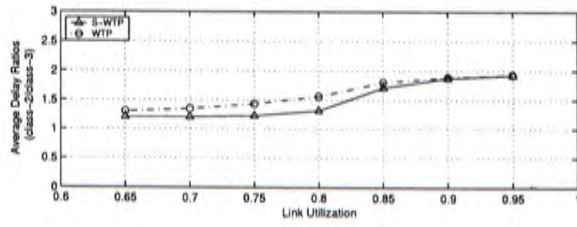


(b) class-2/class-3

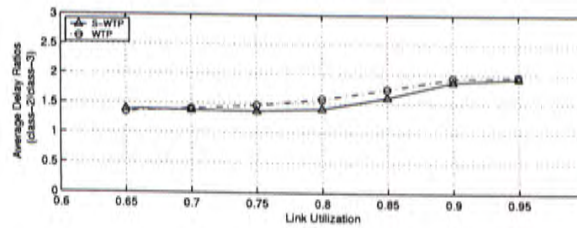
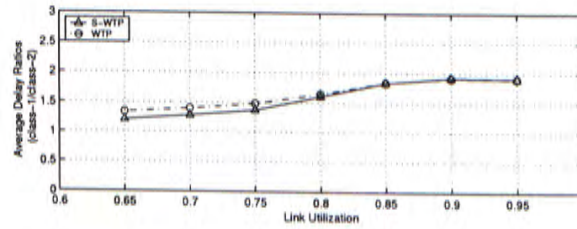
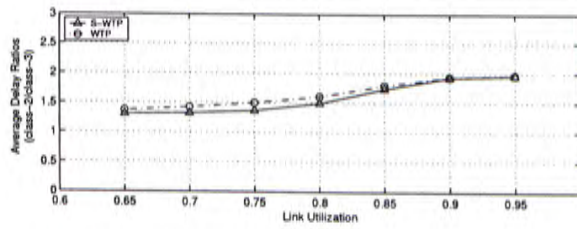
Figure 5.4: Distribution of delay ratios in different timescales( $U = 95\%$ )



(a)  $\rho_1 : \rho_2 : \rho_3 = 60 : 20 : 20$



(b)  $\rho_1 : \rho_2 : \rho_3 = 30 : 40 : 30$



(c)  $\rho_1 : \rho_2 : \rho_3 = 20 : 30 : 50$

Figure 5.5: Delay ratios with different traffic load and distributions ( $\frac{b_{i+1}}{b_i} = 2.0$ )

#### 5.4.4 Delay Ratios with More Classes

In the previous simulation experiments, we just investigated the performance of S-WTP in the case of three classes. In this part, we investigate the effectiveness of S-WTP when more traffic classes are involved. We test three cases with 4 classes, 5 classes and 6 classes respectively. The achieved delay ratios between classes are shown in Tables 5.1 and 5.2 for the cases that target ratios are 2.0 and 1.5 respectively. For all of these cases, it is seen that S-WTP is as effective as WTP even a larger number of classes are introduced. Although it is difficult to show the efficiency of these two schedulers, but for the analytical results previous section, it is obvious that more classes, the more computations are saved with S-WTP than WTP.

Delay ratio	4 classes		5 classes		6 classes	
	SWTP	WTP	SWTP	WTP	SWTP	WTP
class-1/class-2	1.9843	2.0093	1.9644	2.0015	1.9607	2.0092
class-2/class-3	1.9776	1.9919	1.9892	1.9817	1.9473	1.9822
class-3/class-4	1.9639	1.9620	1.9499	1.9739	1.9834	1.9847
class-4/class-5	-	-	2.0053	1.9880	1.9431	1.9553
class-5/class-6	-	-	-	-	1.9567	1.9617

Table 5.1: Average delay ratios between classes ( $\frac{b_{i+1}}{b_i} = 2.0, U = 95\%$ )

Delay ratio	4 classes		5 classes		6 classes	
	SWTP	WTP	SWTP	WTP	SWTP	WTP
class-1/class-2	1.4370	1.4443	1.4251	1.4118	1.4898	1.5185
class-2/class-3	1.4645	1.4888	1.5327	1.5496	1.4837	1.4953
class-3/class-4	1.4694	1.4993	1.4757	1.4897	1.5033	1.4780
class-4/class-5	-	-	1.5459	1.5052	1.5077	1.5037
class-5/class-6	-	-	-	-	1.4693	1.5069

Table 5.2: Average delay ratios between classes ( $\frac{b_{i+1}}{b_i} = 1.5, U = 95\%$ )

## 5.5 Summary

In current DiffServ architecture there are no any standardized mechanisms as the scheduling algorithms in the core network, and even for the DiffServ model itself, there are still many open problems. In this chapter, we propose the Shifted Waiting Time Priority (S-WTP) for *delay differentiated services*. S-WTP derives the basic scheduling principles

from WTP and uses the packet's waiting time as the priority metric. S-WTP reduces the computational complexity and is more efficient than WTP. We also conduct a series of simulation experiments to show that S-WTP is as effective as WTP in a wide range of environments and achieves close performance as WTP in most cases. This make S-WTP an idea scheduling algorithm for deployment in high speeds network environments to provide delay differentiated services.

# Chapter 6

## Conclusions

The Internet has grown exponentially in the past few years in both the number of applications and the number of users. Facing the challenge to meet the diverse requirements from both applications and users, the Internet evolves gradually towards a global commercial infrastructure which calls for new technologies and solutions to provide high performance guarantee to customers. In this thesis, we address some aspects of the most challenging works as avoiding congestion and providing quality of service guarantee in IP networks.

### 6.1 Congestion Control

TCP is the dominate transport protocol in current IP networks. It employs a host-based congestion control mechanisms which respond to the network state dynamic by monitoring the network state with a end-to-end measurement, such as packet loss, variance of round trip time, or fluctuation of flow's throughput. The weakness of this host-based mechanism has long been discussed. In this thesis, we firstly investigated the performance of TCP Vegas, a TCP protocol employing the round trip time(RTT) as the metric for congestion control, in the network with multiple congested links. We analyzed the congestion control algorithm of TCP Vegas and found that it may lead to unfairness to these flows that pass through multiple-hop path. This unfairness is unacceptable from the perspective of users. We also conduct a series of simulation experiments to verify our analytical results. Our work, as well as many studies by other researchers, revealed that the host-based congestion control is not reliable for providing accurate and effective control as congestion occurre in the network.

Under the belief that the participant of routers in the network can contribute to the congestion control and improve the performance of TCP protocol, we then propose the *Joint Congestion Control* for TCP/IP networks. JCC is a scheme in which routers can provide detailed congestion information of the link via the probing packet from the

end hosts. We propose **Additive-Increase Exponential-Decrease(AIED)** algorithm which can make full use of the *path load reduction factor*, a metric defined to describe the congestion level of a path, to adjust the congestion window smoothly in a proactive way. We also propose a modified RED algorithm to provide the scaled dropping probability as the congestion level in the router, which make it possible to implement the JCC scheme in practice. We also conducted extensive simulation experiments to illustrate the performance of JCC from different aspects and proved that JCC can provide satisfied performance in terms of throughput stability, packet loss rate and fairness comparing to other version of TCP.

## 6.2 Quality of Service Provision

The DiffServ architecture provides a framework within which services providers can offer customers a range of network services, each differentiated based on desired quality of service level. Currently it only specifies the basic mechanism as how the packets should be treated. Much freedom are left for discussion of performance metrics and scheduling algorithms for the building block. In this thesis, we focused on the *Delay Differentiated Services*, a model proposed to provide relative quality of services to delay sensitive application. We studied WTP, an algorithm that is designed for packet scheduling in this model. We suggested that the high computational complexity make it difficult for this algorithm to be deployed in the high speeds network environments. We then proposed S-WTP based on the WTP algorithm. S-WTP derives the basic scheduling principle from WTP but reduces the computational complexity of WTP from  $O(n)$  to  $O(\log(n))$ . Simulation results suggested that S-WTP preserves the basic functionalities of WTP in a wide range of environments, which make it an idea scheduling algorithm for delay differentiated services in high speeds network.

## 6.3 Final Remarks

Congestion control and quality of service provision in IP networks are important but difficult tasks. While this thesis has examined several problems and has provided a number of solutions, there are undoubtedly still many issues which need to be addressed in the future.

One of the problem in congestion control is the control of irresponsible traffic. Currently, most of the real-time applications are based on UDP protocol, which is well known as a connection-less protocol without any congestion control schemes. So UPD traffic is said to be irresponsible traffic as it does not make cooperative action in case of congestion. Currently the router-based congestion avoidance schemes are not effective to handle this kinds of irresponsible traffics. Some people have discussed possible solutions

to make them "TCP-friendly" [24, 33], i.e, respond to congestion in the similar way as TCP does. Our JCC scheme, although is proposed for TCP protocol, can be extended to incorporate into UDP protocol to make it responsible as TCP does.

For the provision of quality of service in the Internet, current interests in both academic research and industry are focused on DiffServ model. At present, the DiffServ only specific the basic mechanism on ways as how to treat packets. Variety of service can be built into this model by using these mechanisms. In this sense, there is still much space on the discussion of the scheduling algorithm and measurement metrics. This thesis just touch a small piece of this work by studying the S-WTP algorithm for delay differentiated services. More works can be done in this direction.

# Bibliography

- [1] The network simulator - ns-2. Available at : <http://www.isi.edu/nsnam/ns/>.
- [2] J. Ahn, P. Danzig, Z. Liu, and L. Yan. An evaluation of tcp vegas: Emulation and experiment. *Computer Communications Review*, 25(4), Oct. 1995.
- [3] J. M. an Richard J. La, V. Anantharam, and J. Walrand. Analysis and comparison of tcp reno and vegas. In *Proceedings of INFOCOM'99*, 1999.
- [4] S. Blake. An architecture for differentiated services. RFC2475, IETF Diffserv Working Group, Aug. 1998.
- [5] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, Estrin, S. Floyd, V. Jacobson, G. Minshall, Fiberlane, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. Recommendations on queue management and congestion avoidance in the internet. RFC2309, IETF Network Working Group, Apr. 1998.
- [6] D. Chiu and R. Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems*, 17:1-14, 1989.
- [7] D. Clark and W. Fang. Explicit allocation of best-effort packet delivery service. *IEEE/ACM Transactions of Networking*, 6(4):362-373, Aug. 1998.
- [8] D.Clark. The design philosophy of the darpa internet protocols. In *Proceeding of ACM SIGCOMM*, pages 106-114, Aug. 1988.
- [9] D.Clark, S. Shenker, and L. Zhang. Supporting real-time applications in an integrated services packet networks: Architecture and mechanism. In *In proceeding of ACM SIGCOMM*, pages 14-26, Aug. 1992.
- [10] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *Journal of Internetworking Research and Experience*, pages 3-26, Oct. 1990.



- [11] C. Dovrolis, D. Stiliadis, and P. Ramanathan. Proportional differentiated services: Delay differentiation and packet scheduling. In *ACM SIGCOMM 99*, Sept. 1999.
- [12] L. Essafi, G. Bolch, and A. Anders. An adaptive waiting time priority scheduler for the proportional differentiation model. Technical Report TR-I4-00-06. University of Erlangen-Nrnberg, Sept. 2000.
- [13] S. Floyd. Tcp and explicit congestion notification. *ACM Computer Communication Review*, 24(5):10–23, Oct. 1994.
- [14] S. Floyd and V. Jacobson. On traffic phase effects in packet-switched gateways. *Internetworking Research and Experience*, pages 115–126, 1992.
- [15] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4), Aug. 1993.
- [16] V. Jacobson. Congestion avoidance and control. *Computer Communication Review*, 18(4):314–329, Aug. 1988.
- [17] V. Jacobson. Modified tcp congestion avoidance algorithm. end2end-interest mailing list, available at <ftp://ftp.isi.edu/end2end-interest-1990.mail>, Apr. 1990.
- [18] R. Jain. A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks. *Computer Communication Review*, 19(5):56–71, Oct. 1989.
- [19] L. Kleinrock. *Queueing Systems*, volume 1: Theory. New York: Wiley, 1975.
- [20] K. H. Leung, John C. S. Lui, , and David K. Y. Yau. Characterization and performance evaluation of proportional delay differentiated services. In *ICNP 2000*, Nov. 2000.
- [21] C. Li, S.L. Tsao, M. C. Chen, Y. Sun, and Y.M. Huang. Proportional delay differentiation service based on weighted fair queuing. In *Proceedings Computer Communications and Networks 2000*, 2000.
- [22] Y. Lim and J. Kobza. Analysis of a delay-dependent priority discipline in an integrated multiclass traffic fast packet switch. *IEEE Transactions on Communications*, 38(5), May 1990.
- [23] L.S.Brakmo, S.W.OMalley, and L. Peterson. Tcp vegas: New techniques for congestion detection and avoidance. In *1994 ACM SIGCOMM Conference*, pages 24–35, May 1994.

- [24] J. Mahdavi and S. Floyd. Tcp-friendly unicast rate-based flow control. Technical note sent to the end2end-interest mailing list, Jan. 1997.
- [25] Mankin. Random drop congestion processing. In *Proceedings of the SIGCOMM '90 Symposium: Communications Architectures Protocols*, pages 1–29, Sept. 1990.
- [26] J. Martin, A. Nilsson, , and I. Rhee. The incremental deployability of rtt-based congestion avoidance for high speed tcp internet connections. In *Proceedings of SIGMETRICS 2000*, June 2000.
- [27] J. Nagle. Congestion control in ip/tcp internetworks. RFC896, IETF Network Working Group, Jan. 1984.
- [28] K. Nichols, S. Blake, F. Baker, and D. Clack. Definition of the differentiated services field (ds field) in the ipv4 and ipv6 headers. RFC2474, IETF Diffserv Working Group, Dec. 1998.
- [29] K. Nichols, V. Jacobson, and K. Poduri. Expedited forwarding phb group. RFC2598, IETF Diffserv Working Group, June 1999.
- [30] J. Postel. Internet protocol specification. RFC791, IETF, Sept. 1981.
- [31] K. Ramakrisnan and S. Floyd. A proposal to add explicit congestion notification (ecn) to ip. RFC2481, IETF Diffserv Working Group, Jan. 1999.
- [32] R. Braden, D. Clark, and S. Shenker. Integrated services in the internet architecture. RFC1633, IETF Network Working Group, June 1997.
- [33] R. Rejaie, M. Handely, and D. Estrin. Rap: An end-to-end rate-based congestion control mechanism for realtime streams in the internet. In *IEEE Infocom'99*, Mar. 1999.
- [34] J. L. Richard, J. Walrand, and V. Anantharam. Issues in tcp vegas. Available at <http://www.path.berkeley.edu/hyongla>, July 1998.
- [35] S. Floyd. Connections with multiple congested gateways in packet-switched networks part 1: Oneway traffic. *Computer Communication Review*, 21(5), Oct. 1991.
- [36] W. Stevens. Tcp slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. RFC2001, IETF Network Working Group, Jan. 1997.
- [37] S. Vegesna. *IP Quality of Service*. Cisco Press, 2001.
- [38] V. Paxson. Phd. thesis. University of California Berkeley, 1997.

- [39] W.Fen, D. Kandlur, D. Saha, and K. Shin. Blue: A new class of active queue management algorithms. UM CSE-TR-387-99, 1999.
- [40] Z.Wang and J. Crowcroft. A new congestion control scheme:slow start and search(tri-s). *Computer Communication Review*, 21(1):32-43, Jan. 1991.
- [41] C.Q. Hua and Tak-shing P. Yum. The Fairness of TCP Vegas in Networks with Multiple Congested Gateways. *IEEE 5th International Conference on High Speed and Multimedia Communications*, July 2002.



CUHK Libraries



003955613