FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Yong Deng
AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.A.Sc. (Electrical Engineering)
GRADE / DEGREE

School of Information Technology and Engineering
FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Design and Implementation of Signaling and Traffic Control for AAPN

TITRE DE LA THÈSE / TITLE OF THESIS

Prof. O. Yang
DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Prof. H. Mouftah

Prof. C.C. Huang

Prof. G. Bochmann

Gary W. Slater
Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

Design and Implementation of Signaling and Traffic Control for AAPN

Yong Deng

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of requirements
For the Master of Applied Science degree in

Electrical Engineering

Department of Electrical and Computer Engineering
School of Information Technology and Engineering
Faculty of Engineering
University of Ottawa
Ottawa, Ontario
Canada

Your file Votre référence
ISBN: 978-0-494-49185-0
Our file Notre référence
ISBN: 978-0-494-49185-0

Abstract

Although all-photonic network is a hot topic in recent years, most networks still do not have the ability to achieve high utilization and to allocate bandwidth dynamically and efficiently to an edge node. Many theoretical works and publications have been done in this field. However, up to now there is no commercial or lab product in this area. AAPN (Agile All-Photonic Network) is one research project focusing on these issues while at the same time avoid blocking and starvation. We designed and implemented the whole AAPN signaling protocols under Linux and produced a lab prototype under the AAPN framework. To the best of our knowledge, this is the first AAPN prototype in the world.

While bandwidth has increased so much after the Internet first came to reality, user's request increases even much faster. Congestion can not be avoided if there is no algorithm to control it. The XCP (eXplicit Control Protocol) algorithm is a promising congestion control method that uses explicit feedback. However, XCP does not work when the link capacity changes with time. In this research we designed a modified version, the XCP-CL (XCP of Cross-Layer design) algorithm, that works successfully on the AAPN network. Extensive simulation results have verified that XCP-CL has a much better performance compared to the original XCP algorithm.

Acknowledgment

I would like to thank my supervisor, Prof. Oliver Yang, for his guidance, support, encouragement and his accessibility to my requests for help throughout my study at the University of Ottawa. He introduced me to the research area of Internet and photonic networks and provided help to my family.

I would like to thank Prof. Gregor v. Bochmann for his partial financial support (an RA for a year) to work on the control platform implementation of the AAPN project, the fruitful discussions about the design the of protocols, the architecture of the software and his guidance for part of Chapter 2 and whole Chapter 3 on signaling design and implementation of this thesis.

The interface of the control platform to the scheduling algorithm and the input buffer management is discussed with Dr. Cheng Peng and Dr. YiMing Zhang. QBvN algorithm is implemented by Dr. Cheng Peng. The input buffer management and file transfer are implemented by Dr. YiMing Zhang.

Thanks also go to Dr. Yang Hong, Dr. Yiming Zhang and Dr. Cheng Peng, for their help, great suggestions and fruitful discussions on my thesis. I will remember the happy time that we worked together.

The last but not the least, I would like to thank my family, my wife, QingHua, my son Rickey, my daughter Judy, for their understanding and support without which my study could never have started. I would also like to thank my parents, for their understanding and endless love, even though I have not visited them for a long time.

This research has been partially supported by University of Ottawa Admission Scholarship 2005~2006 and the AAPN (Agile All-Photonic Networks) Research Network through its industrial and government partners.

Table of Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

		Section # of 1st appearance
AAPN	Agile All-Photonic Networks	1
ACK	ACKnowledgement	1.1.4
AIMD	Additive-Increase Multiplicative-Decrease	1.1.4
APOSN	All-Photonic Overlaid-Star Network	1.1.2
AQM	Active Queue Management	1.1.4
ARED	Adaptive RED	1.1.4
AVQ	Adaptive Virtual Queue	1.1.4
BDP	Bandwidth-Delay Product	1.1.4
CAS	Channel-Associated Signaling	1.1.1
CCS	Common Channel Signaling	1.1.1
CIOQ	Combined Input and Output Queue	2.2
cwnd	Congestion WiNDow size	4.2
DRED	Dynamic RED	1.1.4
DTMF	Dual-Tone Multi-Frequency	1.1.1
EC	Efficiency Controller	1.4.2
ECN	Explicit Congestion Notification	1.1.4
ER	Edge Receiving part	1.1.2
ET	Edge Transmission part	1.1.2
ETEN	Explicit Transport Error Notification	1.1.4
FC	Fairness Controller	1.4.2
FPGA	Field Programmable Gate Array	2.3.2
FTP	File Transfer Protocol	5
GCC	GNU Compiler Collection	1.4.1
GNU	GNU's Not Unix	1.4.1
GPIO	General Purpose Input and Output pins	2.3.3
GPL	GNU General Public License	1.4.1
IQ	Input Queue	2.2
LAN	Local Area Networks	1.4.1
LPF	Low-Pass Filter	1.1.4
MIMD	Multiplicative-Increase Multiplicative-Decrease	1.1.4
OBS	Optical Burst Switching	1.1.2
OEO	Optical-Electrical-Optical	2.1
OQ	Output Queue	2.2
OS	Operating System	3.6.1
OSE	Operating System Embedded	3.8.1
OTcl	Object Tcl	1.4.3
OTDM	Optical Time Division Multiplexing	1.1.2

PI	Proportional & Integrate	1.1.4
PSTN	Public Switched Telephone Network	1.1.1
PTP	Performance Transparency Protocol	1.1.4
QBvN	Quick Birkhoff-von Neumann	1.1.3
QFCP	Quick Flow Control Protocol	1.1.4
QoS	Quality of Service	1.1.4
RAP	Rate Adaptation Protocol	1.1.4
RCP	Rate Control Protocol	1.1.4
RED	Random Early Detect	1.1.4
REM	Random Exponential Marking	1.1.4
RSVP	Resource ReSerVation Protocol	1.1.4
RTT	Round Trip Time	4.1.1
SIRENS	Simple Internet REsource Notification Scheme	1.1.4
SONET	Synchronous Optical NETworking	1.1.2
SRED	Stabilized RED	1.1.4
SS7	Signaling System #7	1.1.1
Tcl	Tool Command Language	1.4.3
TCP	Transmission Control Protocol	1.1.4
TCP SACK	TCP Selective Acknowledgment Options	1.1.4
TDM	Time Division Multiplexing	1.1.2
TFRC	TCP Friendly Rate Control	1.1.4
VCP	Variable structure Congestion Control	1.1.4
WDM	Wavelength Division Multiplexing	1.1.2
WXCP	Explicit Congestion Control for Wireless	1.1.4
XCP	eXplicit Control Protocol	1.1.4
XCP-CL	eXplicit Control Protocol of Cross-Layer Design	1.3
XCP-i	XCP incremental employ	1.1.4

List of Notations and Symbols

		Section # of 1st appearance
a, β	Constant parameter in XCP	Appendix A
α	Queue averaging parameter	1.1.4
γ	Shuffling parameter in XCP	Appendix A
η	Protocol efficiency	3.1.1
ϕ	XCP feedback	Appendix A
$\tau(t)$	Average round-trip time	1.1.4
δ	Sampling frequency	Appendix C
d	Average RTT, control interval	2.4
$f1, f2$	Marking probability factor	1.1.4
h	Shuffling traffic	Appendix A
min_{th}	Min queue threshold	1.1.4
max_{th}	Max queue threshold	1.1.4
p	Dropping probability	1.1.4
pm	Marking probability	1.1.4
t_1, t_2, t_3, t_4	Time instants	3.4.1
$x(t)$	Average queue length	1.1.4
w	Number of wavelengths	2.1
BW	Bandwidth	4.1.2
C	Link capacity	1.1.4
M	Number of edge nodes	2.1
N	Number of core nodes	2.1
N_f	Number of frames	3.8.3
N_S	Number of assigned slots	3.6.3
$N(t)$	Traffic load	1.1.4
P_{in}, P_{jn}	Switch port number	4.3
Q	Persistent queue	Appendix A
R_i, R_n	Router ID	2.4
R_{i1}	Edge node ID	2.4
S	Spare bandwidth	Appendix A
T_d	Measured time difference, equals $T_{clockdiff} + T_{delay}$	3.2.1
$T_{clockdiff}$	Clock difference between a master and a slave	3.2.1
T_{delay}	Propagation delay	3.2.1
T_f	Frame period	3.6.3
T_g	Time constraint, guard time	3.2.1
T_{master}	Master local time	3.2.1
$T_{receive\_mastere}$	Master receiving time	3.2.1
T_s	Slot period	3.6.3

T_{send_slave}	Slave sending time	3.2.1
T_{slave}	Slave local time	3.2.1
$W(t)$	Average window size	1.1.4

Chapter 1. Introduction

All-Photonic Network is currently a hot topic in telecommunication network research. It has a potential large bandwidth to meet our increasing Internet traffic demand. There are many photonic test bed activities around the world, e.g. OMNI project [OMNI07] and StarLight project [StLi07]. AAPN (Agile All-Photonic Networks) project [AAPN07] [BoCo04] is one such network that aimed at providing bandwidth dynamically according to the requests from edge nodes in order to improve the network bandwidth utilization while avoiding blocking and starvation at edge nodes. This requires the core node (optical switch) to change its bandwidth allocation dynamically and frequently. Much work has been done on the network topology, switch architecture and bandwidth allocations, e.g. [JiYa05a] [JiYa05b] [JiYa06b] [PeBo06] [Peng07] [RaYa05].

Since network resources such as bandwidth and router buffer space are limited, senders usually compete to get more resources along the data path and this often results in congestion at routers. Therefore networks must have some congestion control mechanism to avoid traffic congestion [Jaco88]. The goal of a congestion control is to regulate the source transmission rates based on the network traffic information in order to prevent the network from congestion and collapse. This is very important for the efficient operation of the Internet to achieve high throughput, low delays of packet delivery and stable network operation.

1.1. Literature Review

We shall review existing work of signaling, scheduling and then congestion control.

1.1.1 Signaling

In telecommunication, signaling has the following meanings:

1. The use of signals for controlling communications.
2. In a telecommunications network, the information exchange concerning the estab-

lishment and control of a connection and the management of the network, in contrast to user information transfer.

3. The sending of a signal from the transmitting end of a circuit to inform a user at the receiving end that a message is to be sent.

Signaling can be classified as CAS (Channel-Associated Signaling) and CCS (Common Channel Signaling) or in-band signaling and out-of-band signaling based on the principal properties [LeWi04].

While CAS employs a signaling channel which is dedicated to a specific bearer channel, CCS employs a signaling channel which conveys signaling information relating to multiple bearer channels. These bearer channels therefore have their signaling channel in common.

In the PSTN (Public Switched Telephone Network) in-band signaling is the exchange of signaling (call control) information within the same channel that the telephone call itself is using. An example is DTMF (Dual-Tone Multi-Frequency) signaling, it is also CAS signaling. Out-of-band signaling is telecommunication signaling (exchange of information in order to control a telephone call) that is done on a channel that is dedicated for the purpose and separated from the channels used for the telephone call. Out-of-band signaling is used in SS7 (Signaling System #7), the standard for the signaling that has controlled the world's phone calls for some twenty years. SS7 is also CCS signaling.

1.1.2 AAPN Architecture

The network topology and switching node are the elementary aspects of a network architecture we should take into consideration [JiYa06b] [ViBe00]. The topology has lasting impact on the network design, economics, operation and performance. The optical transport network topology can be classified as mesh (core nodes are interconnected through WDM, Wavelength Division Multiplexing, optic fiber). It allows for continuous connections and reconfiguration around broken or blocked paths by "hopping" from node to node until the destination is reached.), ring (such as SONET ring, Synchronous Optical NETworking) and star (for example PetaWeb [ViBe00]) or multi-hop and single-hop [BlLe01] [Gree01] [Mukh92a] [Mukh92b]. Each topology has its pros and cons.

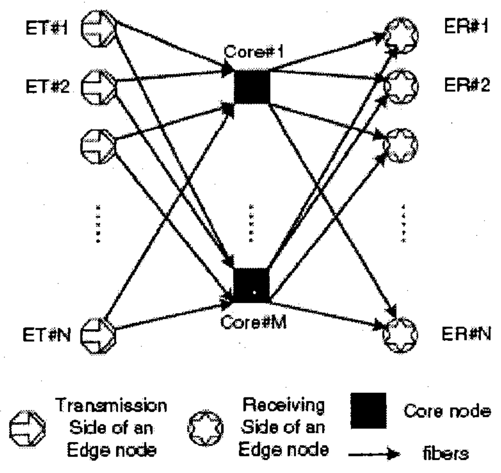
ET#1
ET#2
ET#N
Core#1
Core#M
ER#1
ER#2
ER#N
Transmission Side of an Edge node
Receiving Side of an Edge node
Core node
fibers

Figure 1.1 APOSN Architecture (Courtesy from [JiYa06b])

AAPN has adopted an overlaid star topology to overcome the core node failure that is possible in a single star topology. Compared to complex mesh networks, this overlaid star architecture simplification may result in software simplification, higher reliability, and a reduction of overall network operation costs. There are also advantages of mesh network and star network. While mesh network has been well studied and has less fiber-distance value, it needs expensive wavelength converters and complex algorithm to find a light-path between two edge nodes [ZhYa04]. It requires careful traffic engineering and time-sharing switching modes such as TDM (Time Division Multiplexing) or OBS (Optical Burst Switching) can not be used to improve network utilization and to provide a fine granularity tuning of bandwidth [BlLe01]. Star topology is simple, without the need to perform wavelength conversion, nor wavelength routing while being scalable (by using combiners and splitters [Peng07]). It is not sensitive to spatial traffic distribution and does not need traffic engineering. It may use OTDM (Optical Time Division Multiplexing) switching to increase link utilization which can greatly increase the network agility. However, the failure of the core node is disastrous to the network since all the edge nodes connect each other via the core node. It may require higher fiber-distance value [JiYa06b] [ViBe00].

APOSN (All-Photonic Overlaid Star Networks) [JiYa06b] is a proposal to AAPN networks which introduces agile, microsecond switching device in the core node. It can operate in OTDM or OBS mode. Under the OTDM operation, each optical channel is sliced into time slots (time slot or slot for short, is a terminology of time division multiplexing technology which enables multiple users to share a data link, a user can put a fixed length of traffic in each slot) with some guard time in between. Traffic aggregated by an edge node is sent out through the ET (Edge Transmission part) to a core node. The core node then relays the traffic to ER (Edge Receiving part) of the other desired edge node. All the operation is coordinated through the synchronization unit by a control channel. It provides a feasible and efficient solution for high-speed networks of near future by making use of the currently available optical technologies.

When one edge node is connected to more than one core node, it should decide which core node to transmit its traffic. This is a routing issue. The criteria may depend on the work load of the core node, the cost of the connection, the delay and bandwidth. These kinds of information are exchanged in the signaling protocol. If the edge node decides to use one core node as the traffic relay device, it sends out traffic request to this core node via signaling protocol. Random routing or least-congested-path routing strategy [Stal06] could be used to select one core node to relay the traffic. This is not the topic of this thesis.

1.1.3 Scheduling Algorithms

Notice that optical buffer is usually not provided in the core node of an all-optical network due to the immature optical buffer technology. So we need a scheduling algorithm to coordinate all the edge nodes to exchange traffic in a star network. Much work has been done up to now.

Centralized and distributed scheduling algorithms were proposed [RaYa05b] [RaYa05a]. Centralized scheduling has no collision but with some scheduling delay while distributed scheduling has no scheduling delay but with collision. A parallel time-wavelength assignment algorithm is then presented to solve the routing, wavelength assignment and time slot assignment problems simultaneously for the TDM operation [JiYa06a] [JiYa05b] [JiYa05c]. It uses a heuristic method and a multi-processor concept that reduces computing complexity. QBvN (Quick Birkhoff-von Neumann) decomposition based scheduling algorithm was proposed [PeBo06] [Peng07] with some added computing complexity.

1.1.4 Congestion Control and Classification

Congestion control is a vibrant topic for many researchers since the beginning. From the first time when Jacobson found congestion in Internet in 1988 [Jaco88], many congestion control algorithms have been proposed from then on. Congestion control becomes more difficult as network BDP (Bandwidth-Delay Product) increases. Much work has been done in congestion control such as [ChYa04] [HoYa04] [HoYa05] [HoYa06] [HoYa07] [KaHa02]. Congestion control can be divided into three stages of development historically:

1. Congestion detection and avoidance by end system: In this early stage, no router involves.
2. AQM (Active Queue Management): This can be further divided in two categories: a) Detecting and controlling: When congestion occurs, we try to control it. b) Predicting and controlling: Before congestion occurs (by using a threshold), we indicate the sender to slow down.
3. Using modern control theory to model and control congestion: The end systems and the routers are all involved in the control.

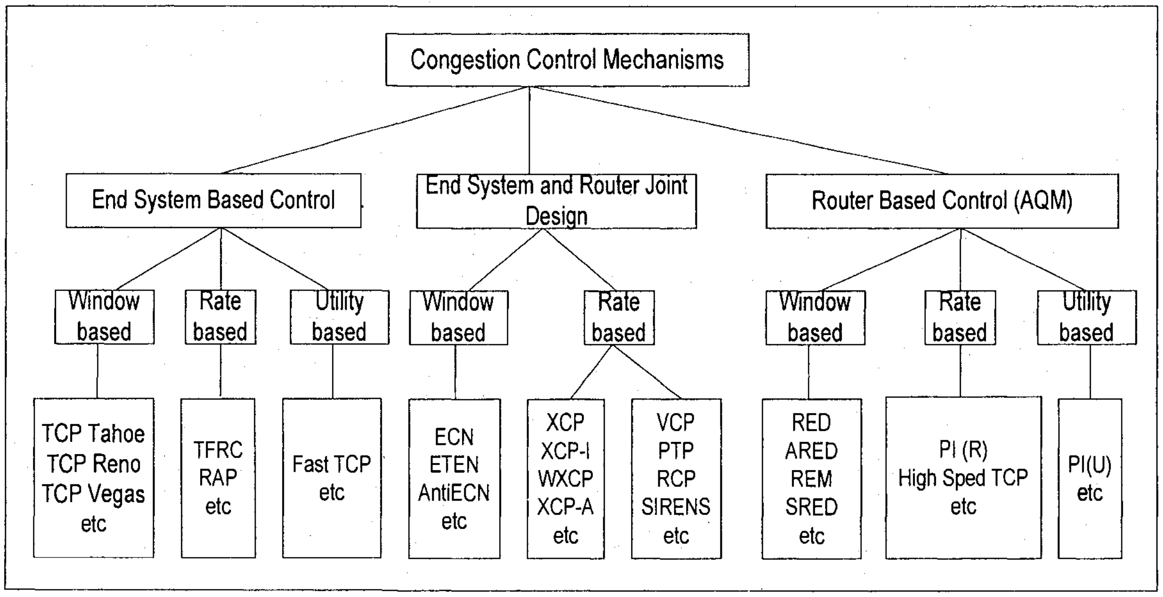

Figure 1.2 Congestion Control Mechanism Classification

According to where an algorithm is placed and whether an end system and/or a router are involved, we classified all algorithms in three categories: end-system-based, router-based (AQM), and the combination of the two.

Based on the control method, Figure 1.2 shows that each category can be further classified into 3 categories. a) window-based: The source is given a maximum number of credits i.e., the window size, to transmit its packets. Once the credits are exhausted, the source has to withhold its transmission until acknowledgments are returned. b) rate-based: The source adjusts its sending rate to support best-effort service traffic and to make the maximum use of network resources, usually used in streaming media transmission, and c) utility-based: The source adjusts its transmission rate based on the feedback of congestion information and its own utility function, in order to make the optimal use of the network resources.

1.1.4.1 End System Based (Source Based)

An end system detects congestion through received ACK (ACKnowledgement) and timeout. If it did not receive an ACK from the receiver in a time interval (timeout), the sender assumes there is congestion in the network. Then it adjusts the sending rate to avoid congestion. This can further be classified as follow (There are some overlaps of the classification category because some authors use more than one method in their algorithms):

a) Window-based

TCP Tahoe [Jaco88] uses slow start with congestion avoidance techniques while TCP Reno [Jaco90] added a new feature - the fast retransmit operation and fast recovery. TCP Vegas [BrOM94] emphasizes on packet delay, rather than packet loss as a signal to help determine the rate at which to send packets. TCP SACK (TCP Selective Acknowledgment Options) [MaMa96] specifies exactly which bytes were missed, better measures the "right edge" of the congestion window.

b) Rate-based

TFRC (TCP Friendly Rate Control) [FlHa00] [HaFl03] is an equation-based congestion control scheme for unicast traffic with slowly changing rate, which is suitable for streaming media applications. The RAP (Rate Adaptation Protocol) [ReHa99] employed an improved AIMD (Additive-Increase Multiplicative-Decrease) source control scheme for real-time stream transmission.

c) Utility-based

The utility-based control [HoYa05a] [JiWe04] [KeMa98] that has emerged recently is a variation of the rate-based source control method. The sender adjusts its transmission rate based on the feedback of the network congestion information and its own utility function, hence is called utility-based control. Utility function is applied in the controller design in order to make the optimal use of the network resources.

1.1.4.2 Router-Based (AQM)

Much work has been done in AQM. In this mechanism, the router runs the algorithm and sends an early signal to the sender to indicate the anticipated congestion. The benefit is congestion control action is taken before the buffer overflows and hence the packet loss is kept as low as possible. Again this can be classified further as follow:

a) Window-based

RED (Random Early Detection) [FlJa93] is the most important one. The basic idea is that the router indicates the sender at a threshold queue value and asks the sender to slow down before congestion occurs. It computes the *p* parameter (drop probability) based on the current average queue length. This parameter is then used as a ratio to drop some packets to indicate the sender to slow down. Because the parameters is hard to determine, so various improvements were proposed.

Some variations include: REM (Random Exponential Marking) [AtLi01], AVQ (Adaptive Virtual Queue) [KuSr01] [KuSr04], ARED (Adaptive RED) [FlGu01], Dynamic RED [AwOu01], Stabilized RED [OtLa99]. BLUE [FeSh02] performs queue management based on the packet loss and the link utilization. REM maintains a marking probability *pm* to either mark or drop the packets. If the queue is continually dropping the packets, *pm* is incremented by a factor of f_1. If the queue is empty or the link is idle, *pm* is decremented by a factor of f_2.

PI (Proportional Integral) [HoMi01] [HoYa04], and pole placement [ChYa04] algorithms use modern control theories to design the congestion controller. These methods use the mature control theory that developed in one decade or more ago to model, design and analyze the network controller which is a new branch in congestion control.

b) Rate-based

A rate-based control allows an end system to adjust its sending rate to support best-effort service traffic and to make the maximum use of network resources. Thus it offers the most effective solution under a network environment with long round trip delays and dynamic changes in available bandwidth where the AIMD control performs poorly in terms of sending rate fluctuation and queue oscillation. Much work has been done in [GeLo02] [HaFl03] [HoYa05b] [HuXu03] [KaKa00].

c) Utility-based

In utility-based control [HoYa05a] [HoYa06] [JiWe04] [KeMa98], the source in the IP networks adjusts its transmission rate based on the feedback network congestion information and its own utility function. Again utility function is applied in the controller design in order to make the optimal use of the network resources.

1.1.4.3 End System and Router Joint Design (Explicit Congestion Notification)

This design category combines the end system and the router together. The router sends back congestion notification explicitly to the sender. The sender gets this information and does some according adjustment to avoid congestion. The sub-categories are as follow:

a) Window-based

ECN (Explicit Congestion Notification) [RaFl01], AntiECN [Kunn03] provide the information on the queue length at the bottleneck link. ETEN (Explicit Transport Error Notification) [KrSt04] provides a notification scheme of the packet loss rate especially for wireless networks.

b) Rate-based

XCP (eXplicit Control Protocol, also categorized as a window-based protocol) [KaHa02] [Kata03] is first introduced in 2002 and results in a shock wave in the Internet community. In this algorithm, traffic is counted as an aggregated value. This value is used to control the efficiency of the link capacity. The output, i.e. aggregated feedback, is used to control the fairness. Then the per-packet feedback is brought back to the sender through ACK to adjust its sending rate. It uses explicit feedback to tell the sender to what degree is the congestion. Details are summarized in Appendix A.

Since XCP algorithm separates congestion control and fairness control, and uses MIMD (Multiplicative-Increase Multiplicative-Decrease) and AIMD control laws separately, it outperforms most of the up-to-date algorithms in high BDP (Bandwidth-Delay

Product) networks. It achieves full utilization and fairness very quickly, has almost zero queue length with almost no queuing delay, at the same time there is almost no packet loss. However, XCP algorithm needs to know all the congestion information through the data path, and the link bandwidth should not change with time. In other words, XCP will not work in the following two cases: (1) One of the routers in the path does not support XCP and this router is the bottleneck of the path; (2) The bandwidth changes with time, XCP can not get the actual required information to compute the feedback. However, usually photonic switch can not provide network information (such as the bandwidth parameter) to a router where XCP algorithm runs. So XCP algorithm can not be implemented in these photonic networks. It needs some modification to make this algorithm to work in these networks.

Much analysis and improvement work has been done since XCP was proposed. As we can see from the XCP algorithm discussed in [AbRi06] [LoAn05] [LoPh05] [ZhAh05] [ZhHe05], XCP needs supports from all routers through the whole packet path and assumes the link capacity does not change with time. In [LoPh06a] and [SuGr05] the authors use estimation method to estimate the bandwidth when it changes with time, and make the XCP algorithm work. However, in all these proposals, the performance is based on the accuracy of the estimation. If the estimation is bad, the performance is also bad. This is also discussed in [[AbRi06]] [LoAn05] [ZhAh05], as the issue of difference between the real link capacity and the rated link capacity. We will describe our design in details later.

For VCP (Variable structure Congestion Control) [XiSu05], we can say that it is a simplified version of XCP. It uses the ECN bits to indicate the congestion degree as a segment approximation of XCP. So it is simple but can achieve the same advantage to some degree as the original XCP. The most important point is this algorithm uses the existing two bits already used in current TCP/IP RFC, so it does not need to change the current Internet protocol too much.

WXCP (Explicit Congestion Control for Wireless) [YaGr05] gave a solution to situations where bandwidth varies with time using estimation skill. XCP-i (XCP incremental employ) [LoPh06] solved the problem of which XCP does not work when there is a router in the data path that does not support XCP and this router is the congestion point.

It uses a bandwidth estimation technique to feedback congestion information. So the XCP algorithm can be incrementally employed in our current Internet network without fundamental reconstruction.

SIRENS, an explicit notification framework for Internet congestion control [NaKo06], proposes a framework to feedback everything (link bandwidth and available bandwidth, packet loss rate and link error rate, queue length and link delay) of each router (per hop information) to the sender. So the sender can decide which parameters to use to get the optimum performance. It captures a snapshot of the path status on a per-hop basis and enables the receivers to freely make use of this information and to perform more precise and flexible congestion control.

RCP (Rate Control Protocol) [DuKo05] proposes to use per-flow rate instead of per packet window adjustment as the feedback from routers. PTP (Performance Transparency Protocol) [Welz00] collects information from routers and determines the bottleneck bandwidth along the path from a sender to a receiver. QFCP (Quick Flow Control Protocol) [PuHa06] allows flows to start with a high initial sending rate and to converge to the fair-share rate quickly based on the feedback from routers.

1.1.4.4 Modeling

When we talk about congestion control, we should mention the fluid model for TCP/AIMD/AQM [MiGo00] [HoMi01a] [HoMi01b] which is a milestone in this research area. It introduces a mathematical model of congestion control that can be used when we design and analyze the controller. The non-linear dynamic model is described by two coupled, nonlinear differential equations. It shows how the average queue length $x(t)$ reacts to the changes on the average window size $W(t)$, the average round-trip time $\tau(t)$, the traffic load $N(t)$, the link capacity μ and the nonlinear mechanism of the AQM algorithm (e.g., RED [FlJa93]). Transfer functions are also used extensively in frequency-domain modeling. More details can be found in Appendix B.

The compensator studied in [HoMi01b] is the well-known RED controller [FlJa93] consisting of a LPF (Low Pass Filter) and nonlinear gain element. The form of the LPF was derived in [HoMi01b] while nonlinear gain element is a mechanism that marks packets with a dropping probability p as a function of average queue length x_{avg}. The parameter p is varying between two queue thresholds min_{th} and max_{th}, with a slope of

$L_{RED} = p_{max}/(max_{th}-min_{th})$. Combining the two elements, the transfer function for RED [FlJa93] is $C_{RED}=L_{RED}/(S/K+1)$ where $K=log_e(1-\alpha)/\delta$ and α is the queue averaging parameter while δ is the sampling frequency.

1.2. Motivation

Up to now, although much theoretical research work has been done on AAPN, there is no commercial product or laboratory prototype to the best of our knowledge. It is desirable to produce a prototype for demonstration, to evaluate the performance and verify the theoretical work.

The implementation is also required by the AAPN project. All the partners expect to see a state-of-the-art new prototype that works, not only theoretical research and publications. Therefore we would like to implement it and evaluate the performance of the prototype system.

As congestion is not avoidable in real networks, congestion control is a must in AAPN network. We need a technique to provide a mechanism of congestion control. As we can see from the literature review, the XCP algorithm decouples the efficiency controller and fairness controller. It has a good performance such as full utilization, almost no queuing delay, almost no packer drop, and fairness that outperform the up-to-date techniques. It does not need to tune any parameters, and there is no per-flow state. These properties let it outperform most of other congestion control techniques up to now (except for the PI-rate controller [HoYa07] for example). Hence we are interested to use XCP as the congestion control mechanism. However, XCP can not work when the link bandwidth changes with time which is the case in our AAPN network. AAPN needs to change the core switch dynamically to allocate the requested bandwidth as an edge node desired, which results in the link bandwidth change with time. To solve this dilemma, we need to modify and improve the XCP algorithm, so it can work successfully on our AAPN networks.

1.3. Thesis Objectives

The general objective of our work is the research and development of AAPN signaling and traffic control. Specifically we would like to:

1. Design and implement AAPN control platform (framework) signaling protocols.
2. Design and evaluate XCP congestion control on AAPN networks.

1.4. Approaches and Methodologies

In order to fulfill our objectives, we would like to implement the signaling protocol under Linux environment to evaluate our design, modify XCP algorithm by using cross-layer design as our congestion control mechanism then use NS2 as the simulation tool to evaluate the modified algorithm.

1.4.1 Design and Implement the Signaling Protocols under Linux

We designed our signaling protocols according the AAPN specifics. In order to evaluate our AAPN signaling protocols, we implemented our AAPN protocol design under Linux operating system with an all-photonic space switch as the core node and a few PCs as the edge nodes. A dynamic scheduling algorithm and a virtual input queue are integrated in the software. Then we did some experiments to measure the synchronization performance and provided some analysis.

Linux operating system is chosen because it is one of the most prominent examples of free software and open source development. It uses the GPL (GNU General Public License). It is a Unix-like computer operating system. Its underlying source code can be modified, used, and redistributed by anyone freely. We have full access to the kernel of this operating system. The GCC (GNU Compiler Collection) compiler is the most popular currently in Unix-like systems including some commercial compilers due to its good compatibility.

C language is used for the programming because we want to move the whole system in an FPGA (Field Programmable Gate Array), or an embedded system. We can use System C to implement it easily in an FPGA. So we could reuse most of the source code as we implemented in Linux and reduce the work load. C also reduces the object code size that is critical in an embedded system which has limited memory. We measure the synchronization error in order to evaluate our implementation. Scheduling time is measured with a different number of edge nodes and different frame sizes.

Real system implementation has the following advantages:

1. It can realistically reflect the physical constraints that exist in the real world, such as the propagation delay, and various synchronization issues.
2. It can verify a protocol by sending real traffic between edge nodes. Through the experiments with the prototype, we can improve it as a final commercial product.
3. It can test the current enabling technologies such as what is the fastest switching time the current commercial optic switch can achieve. This is important because if the enabling technology can not provide fast enough switching time, the switching overhead may degrade the operation of AAPN.

While implementation has these advantages, it also has some shortcomings. It is time consuming to implement the design, especially in the debug stage. The hardware may not be stable as we expected. We need to debug the software and at the same time debug the hardware. Due to technology constraints, we can not always test all theories that published in recent years which made some assumptions that are quite different from the current technology.

1.4.2 Cross-Layer Design on XCP as the Congestion Control Mechanism

As discussed before, despite the good features of XCP algorithm, it does not work with AAPN whose bandwidth is constantly changing. Therefore we need to modify the original XCP algorithm to let it work in our AAPN network (we call it XCP-CL algorithm). We shall use a cross-layer design method to get the exact bandwidth information from the core node through AAPN signaling. The detail will be discussed later.

Due to time limitation and hardware constraints, we do not have access to all the hardware to implement a real system, we only implemented the algorithm by simulation which is commonly used in the literature. Therefore, we shall use simulation to verify its operation and demonstrate the capability of this design.

1.4.3 NS2 to Simulate the XCP-CL Algorithm for Performance Evaluation

In order to evaluate our XCP-CL algorithm and to compare the performance (link utilization, window size and sending rate) of the original XCP and XCP-CL algorithms on AAPN, we choose NS2 (Network Simulator version 2) [ISI07] as the simulation tool.

NS2 is chosen because it is free with an open source, and the license is GPL. Therefore, we can have full access to the source code of the software, modify or add new features to the simulator. Most of the newest designed protocols and algorithms are implemented under NS2 and they have the source code available on the web. This reduces much of the coding development effort.

We shall download the C source code of the original XCP under NS2, and modify it according to our cross-layer design. We construct the simulation network as shown Section 4.3. Tcl script language is used to describe the network model. Then we run the simulation with specified network parameters to obtain various performance measures.

Time-evolution plot is used to evaluate and analyze the performance of our XCP-CL design. We compare the original XCP with our XCP-CL in terms of congestion window size, sending rate, link utilization, number of dropped packets, queue length and required buffer size. During the investigation of the time-evolving performance, we shall also apply step changes to the environment such as big sudden changes in the bandwidth and propagation delay. According to control theory, step response is a good approach to test the capability of a system (e.g., stability and rise time) and therefore its performance [Nise04].

To verify the system performance as designed, we compute the expected window size and sending rate of the source which are dependent on the round trip time as well as the congestion in the routers. If the simulation results are similar to the computed value, we have good confidence that the simulation result is correct.

1.5. Thesis Contributions

The main contributions of this thesis are:

1. AAPN control platform design and implementation under Linux. Our experimental results have shown that the whole system works very well.
2. Synchronization signaling performance measurement and signaling protocol overhead analysis.
3. Cross-layer design of a congestion controller for AAPN.
4. NS2 simulation for the XCP-CL algorithm.

5. Performance evaluation and analysis of the XCP-CL algorithm. The results have shown that XCP-CL algorithm has much better performance compared to the original XCP algorithm.

1.6. Thesis Organization

This thesis is organized as follow: Chapter 2 describes the AAPN network architecture, operation, modeling and some assumptions we used in our AAPN design and implementation. Chapter 3 presents detailed implementation, experimental result and analysis of the AAPN signaling protocols. Chapter 4 details the XCP-CL design including cross-layer design, network modeling. Chapter 5 presents simulation results and analysis. Chapter 6 provides some design guidelines. Chapter 7 concludes our work and makes some proposals for future work.

1.7. Publications

The following are the publications based on our work:

1. Yong Deng, Oliver Yang and Yang Hong, "Cross-Layer Design of XCP on Agile All-Photonic Network (AAPN)", MilCom2007, Orlando, Oct 29-31, 2007
2. Yong Deng, Yang Hong, Oliver Yang and Gregor v. Bochmann, "Cross-Layer Design of XCP on AAPN", AAPN's 2007 Annual Research Review Conference, Nortel Networks, Ottawa, June 14-15, 2007
3. Jonathan Couturier, Yong Deng, and et al, "AAPN Demonstrator Prototype - Control Platform", AAPN's 2007 Annual Research Review Conference, at Nortel Networks, Ottawa, June 14-15, 2007
4. Gregor v. Bochmann, Yong Deng, and et al, "Software Development for the AAPN Control Platform", AAPN's 2007 Annual Research Review Conference, at Nortel Networks, Ottawa, June 14-15, 2007

Chapter 2. Network Architecture, Modeling, Operation and Assumptions

In this chapter we first describe the network topology, switch architecture, network model and the assumptions we made in our design. Finally, the implemented network for the AAPN signaling design was brought out.

2.1. The Overlaid Star Topology and Operation

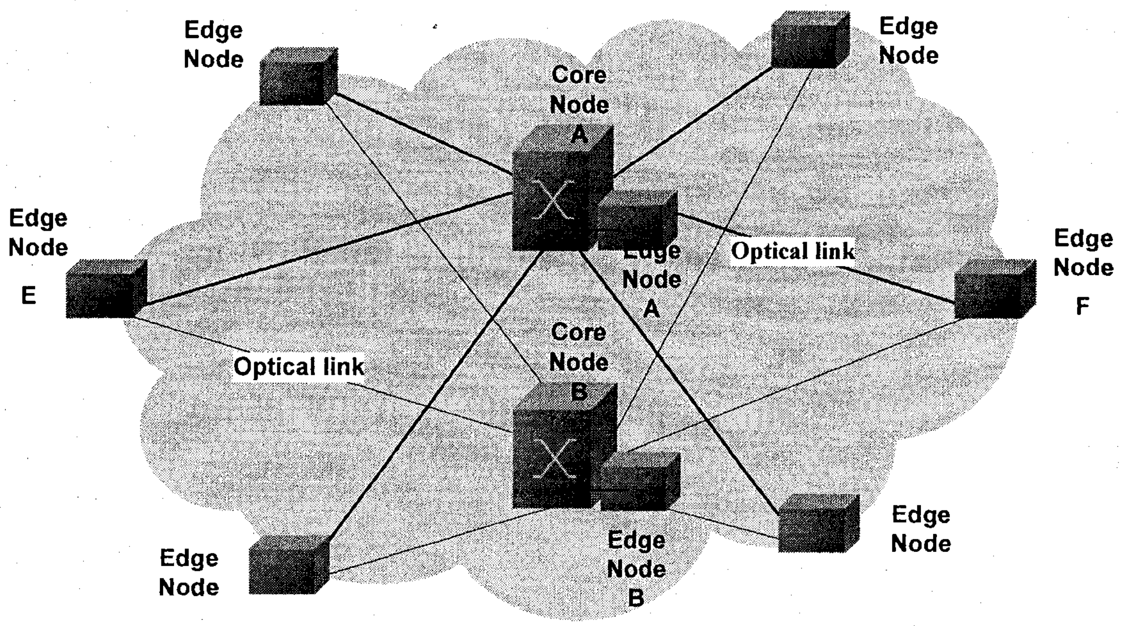

Figure 2.1 AAPN Star Topology with N=2 Core Nodes and M=8 Edge Nodes

Figure 2.1 depicts the overlaid star topology of the AAPN network [JiYa06b] [LoVi06] [MaVi06] [Peng07] to be investigated in our research. It consists of M edge nodes communicating with each other through the N core nodes.

A core node is a non-blocking all-optical space switch (see next section). It relays traffic between edge nodes. An edge node is an ordinary hybrid photonic/electronic router with buffers and runs the AAPN signaling protocol. It has connected to one or more core nodes by optical fibers. It aggregates traffic from outside of AAPN and is responsible for distributing traffic to legacy networks. An edge node (called master node, or master in short, other edge nodes are called slave edge nodes or slave in short) also co-resides with a core node. It runs a scheduling algorithm to allocate bandwidth to each edge node (including the master node itself) and controls the core switch through a direct connection.

The network operation can be illustrated by Figure 2.1. Assume some traffic need to go from a network which is extended to edge node E to a network which is extended to edge node F. After edge node E receives traffic from the legacy network outside the AAPN, it assemblies the traffic into slots. Then edge node E sends a traffic request to the master node A (can also be B, depending on the routing result) through a signaling channel (see the signaling protocol later). Master node A co-resides with core node A which has a scheduler. After master node A receives all the requests from all edge nodes, it runs an algorithm to compute the bandwidth allocation based on the traffic request and sends the results out to each edge node (the master node itself has the traffic information of its own). After edge node E received the allocation, it knows the time and the slot number it should use to send out traffic. When the slots arrive at core node A, master node A has reconfigured the switch to relay the traffic to edge node F. Edge node F receives the traffic and forwards the traffic to the destination.

All these operations within the network (shown by a grey cloud) are performed in the photonic domain using WDM (Wavelength Division Multiplex) via *w* wavelengths. We will talk about the signaling process in detail later. No routing is described here since we only have one core node in our implementation.

Comparing to the current optical networks, AAPN performs not only transmission but also switching in optical domain. The absence of OEO (Optical-Electrical-Optical) conversion at the core nodes leads to two important advantages: greatly increased capacity and the transparency of data format and bit rate. Good bandwidth sharing is achieved through the "agile" property that allocating bandwidth on demand at fine granularity. It also allows carriers to provide and deploy services rapidly.

2.2. The Switch Architecture and Operation

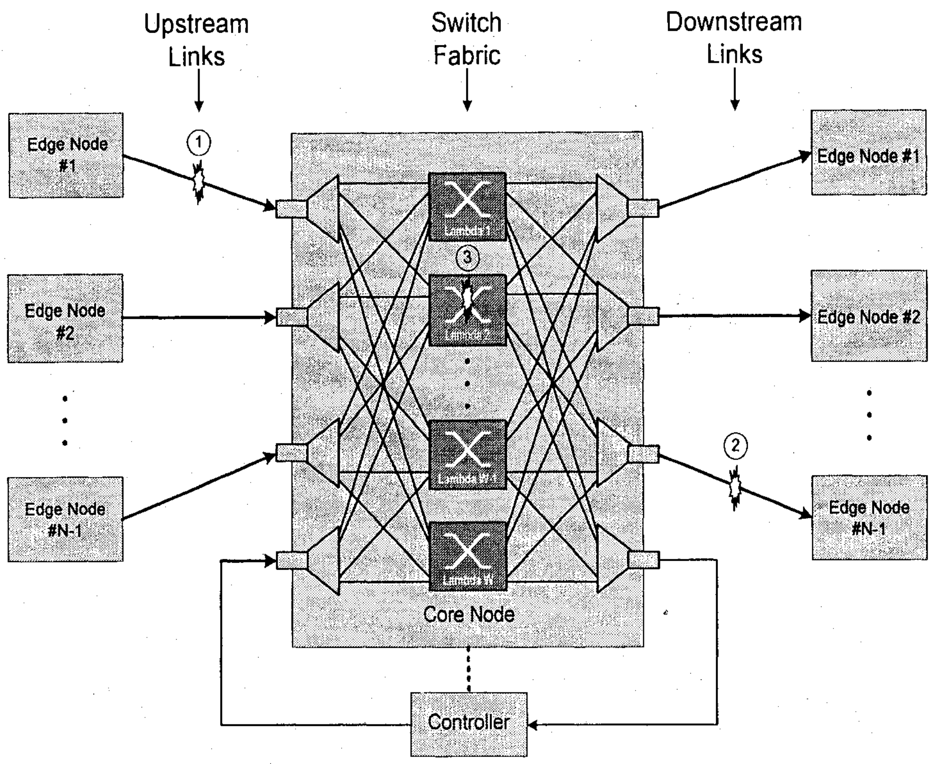

Figure 2.2 The AAPN Switch with One Core Node (Courtesy from [Peng07])

Figure 2.2 depicts the detailed switching architecture of an AAPN core node that is connected to three edge nodes [AAPN07] [JiYa06b]. The core switch is an optical non-blocking space switch. At the input ports on the left, the WDM link is de-multiplexed and each wavelength goes to a separate fabric for switching. Each switching fabric switches only one wavelength. The wavelengths which go to the same output port (on the right side) are then multiplexed onto a single fiber link to the dedicated edge node. The controller receives control information and reconfigures the switch to the desired configuration.

An ordinary electrical switch usually uses OQ (Output Queuing) or IQ (Input Queuing) or CIOQ (Combined Input and Output Queuing). Unlike an ordinary electrical switch, the core switch is made up of only a number of buffer-less transparent photonic space switches (one for each wavelength). Hence traffic from edge nodes must be coordinated to pass the core switch. This requirement induces much work about many scheduling algorithms.

2.3. AAPN Prototype

The design and implementation of the AAPN prototype has undergone two stages: the slow AAPN prototype and the fast AAPN prototype.

2.3.1 Slow AAPN Prototype

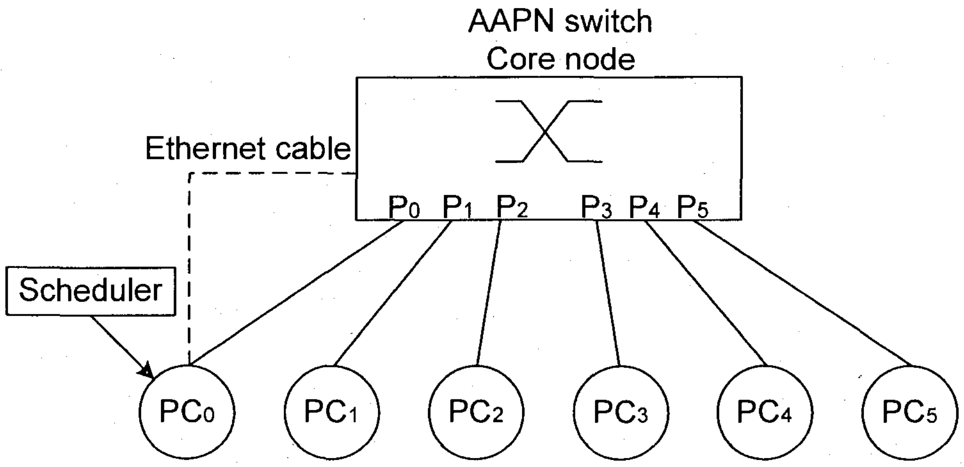

Figure 2.3 Logical Architecture of Slow AAPN Prototype

Figure 2.3 is the logical architecture of a slow AAPN prototype implemented with a few PCs in our lab. It has one core node (the optical switch) and 6 edge nodes (PC_0-PC_5). PC_0 is the master edge node with ID #0. It runs the scheduling algorithm and controls the switch. PC_1-PC_5 are slave edge nodes. The core node is a 6-port optical switch (BigBangWidth's light path accelerator) which connected to the master node through an Ethernet cable. The master node can Telnet to this core node and control it to connect or disconnect two ports. All edge nodes are connected together with optic fibers (1Gbps Ethernet port).

Figure 2.4 shows the equipment in our lab, and Figure 2.5 shows the optical switch - the core node used in our network. We developed this slow AAPN prototype to verify our signaling protocol design: the synchronization and bandwidth allocation protocol, the scheduling algorithm interface, the time constraints of the signaling and scheduling. Signaling protocol and application software runs on all of these PCs. All our experiment and measurement are based on this configuration.

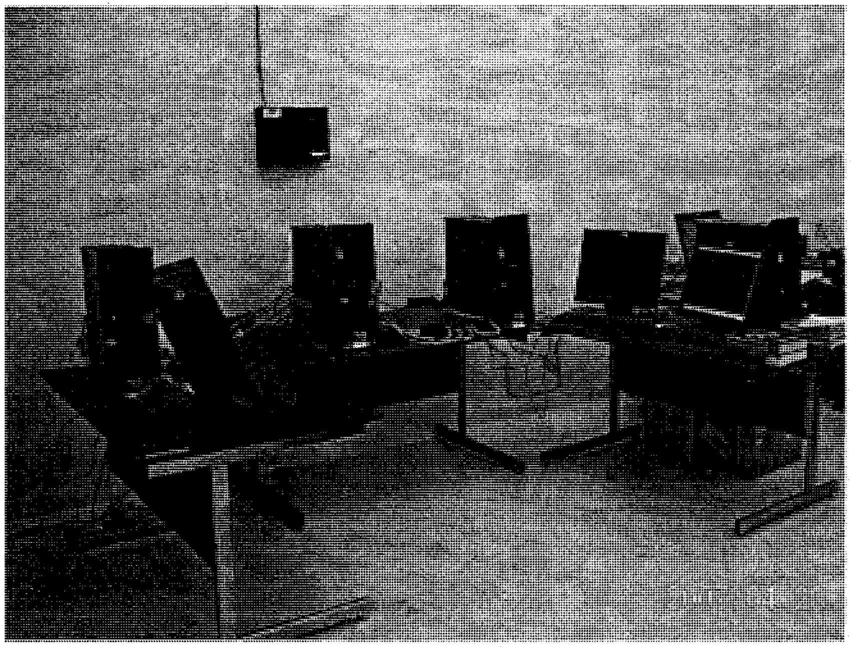

Figure 2.4 View of the Slow AAPN Prototype

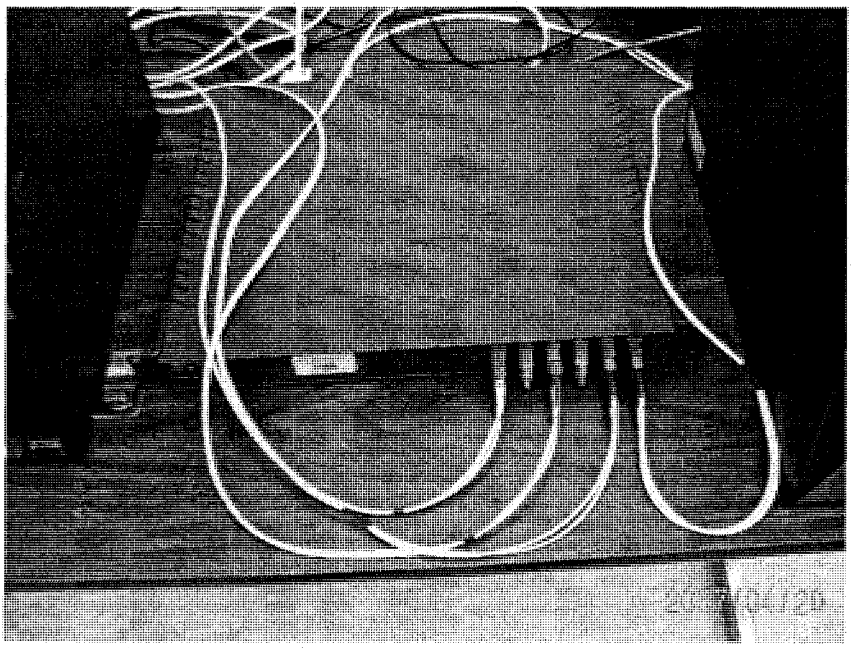

Figure 2.5 Core Switch of Slow AAPN Prototype

In this slow AAPN prototype, we use an Ethernet cable to control the core switch. It has some delay because the command is sent out through Telnet to the core switch and then the switch reconfigures the connection of the input–output port pairs. Also we can not send one command to connect two or more pairs of edge nodes because of the limitation of the software design. Each command can only connect one pair of edge nodes. If we need to connect two pairs of edge nodes, we need to send out two commands.

2.3.2 Fast AAPN Prototype: Work in Progress

In the slow AAPN prototype, the switching time is slow (in the order of seconds) and not fast enough to achieve the benefits of the AAPN concept. The synchronization precision is not good (it depends on the operating system, see analysis in Section 3.8 in later chapter). The measurement and problems are presented later.

Currently we are designing and implementing an intermediate AAPN prototype (Version A) to solve these problems by using a fast core switch (with a switching time of 10us, and a slot period of 250us). We have moved the synchronization processing into the FPGA part and let the PC part processes the protocols only. So the synchronization processing is deterministic and will not oscillate.

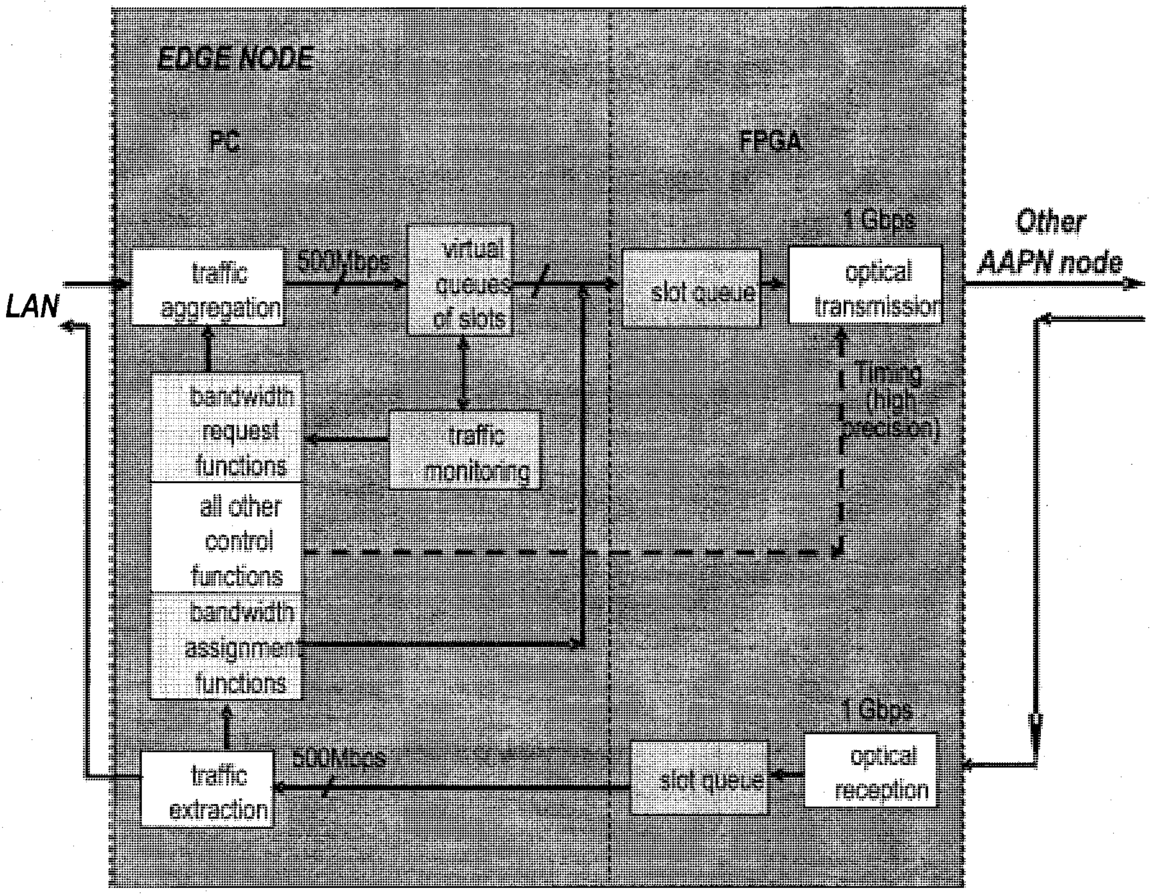

Figure 2.6 Edge Node Architecture of the Fast Prototype Based on FPGA

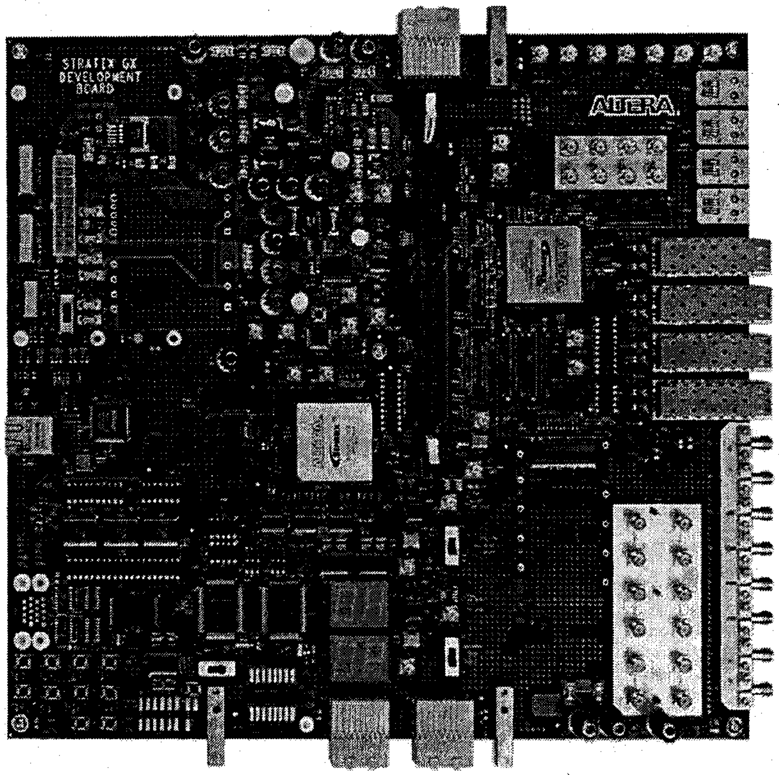

Figure 2.7 FPGA Board for an Edge Node in the Fast Prototype

The typical edge node architecture of FPGA version A is shown in Figure 2.6 while Figure 2.7 shows the FPGA board we are currently using.

The concept is the same as before, but the edge node function is realized jointly by the PC and the FPGA. The PC aggregates data from outside the AAPN and assemblies them into slots. Then it sends these slots to the FPGA in the order they are transmitted to the core node with the desired FPGA sending time. The FPGA then sends them out through optical fiber to other edge node via the core node at the exact sending time. For the receiving part, it is in a reverse direction. Data (in AAPN slot format) from other edge node is received and timestamped by the FPGA receiver, then it is sent to the PC. The PC processes the signaling protocol and extracts and distributes the data to the networks outside of the AAPN. Now the sending and receiving functionality is finished by the FPGA with fixed, deterministic processing time.

The function of virtual input queues, bandwidth requests and allocations are still in the PC so we can take advantage of the PC's programming power, while precise timing is in the FGPA in order to benefit from the quick processing power of the hardware. In the final design (Version B), we will put everything in the FPGA to produce an on-shelf product which is the future work we aim to do.

2.3.3 Switch Control Interface in the Fast Prototype

The speed of a switch control interface depends on the switch that is used. Unlike the slow AAPN prototype, we constructed the core switch by ourselves and have full access to the switch. We use the FPGA to control the core switch directly through the GPIO (General Purpose Input and Output pins) of the FPGA to increase the response time.

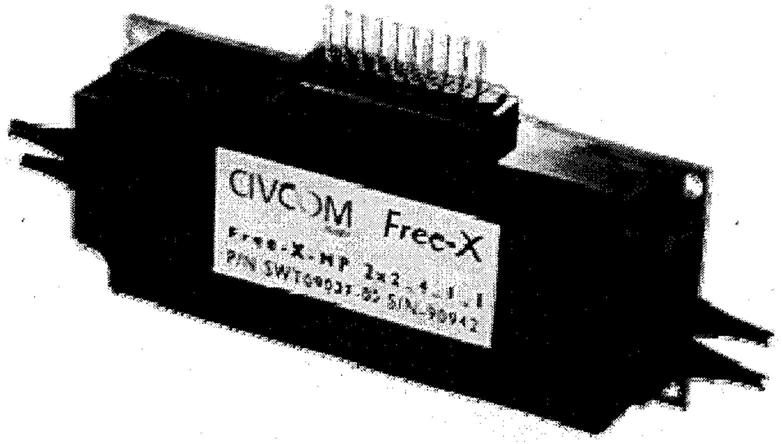

Figure 2.8 2X2 Optical Switch in Fast Prototype

Figure 2.8 shows the switch we used in our immediate prototype. We use this 2X2 switch to construct our own 4X4 no-blocking switch. This work is done by the other members of the project.

2.3.4 Communication between the PC and FPGA

Figure 2.9 shows the data format and the interface between the PC and FPGA in our fast prototype. A control information field (20 octets) is added to each slot data. Then this slot is chopped into Ethernet packets and sent to FPGA board over an Ethernet interface. FPGA takes out the switch control data information and reassemblies those packets into the original slot. The switch control data is used to control the optical switch while the original slot is sent out through an optical fiber. From the user's point of view, a slot is sent to FPGA and is sent out to optical fiber as we can see the logical function on the bottom in this figure.

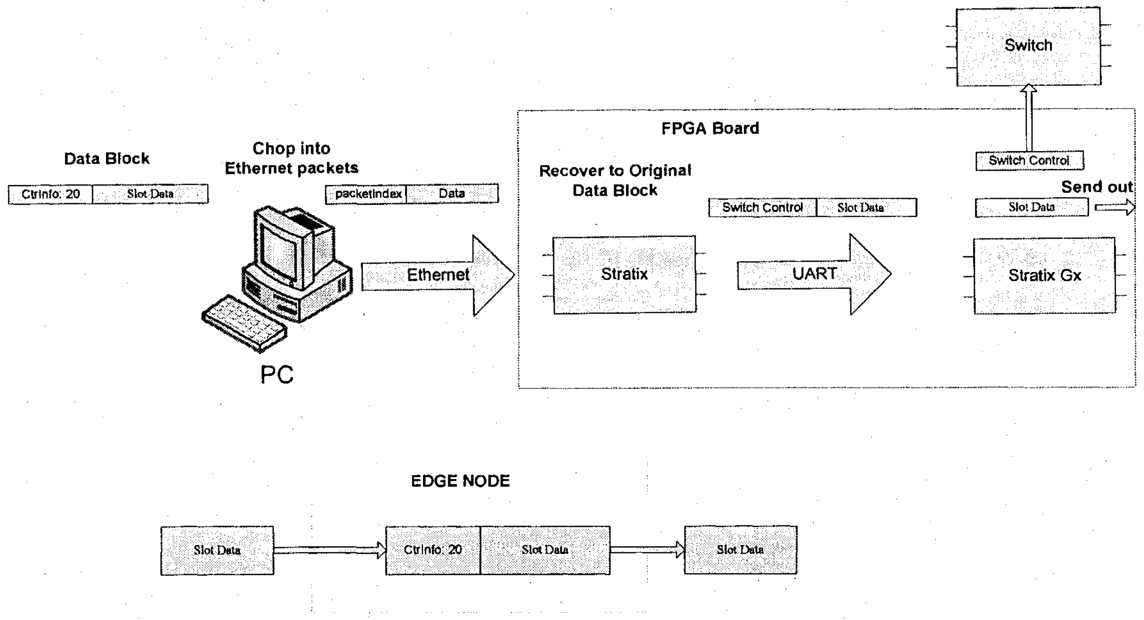

Figure 2.9 Interface between the PC and FPGA in the Fast Prototype

Below is the step-by-step procedure of the communication (assume traffic is aggregated and some slots are allocated to send traffic):

1. Take a slot from the slot buffer.
2. Add the control information field (switch configuration parameter from scheduling results) to this slot.
3. Segment the whole part into Ethernet packets with packet index at the head of each packet.
4. Send out all packets to the FPGA.
5. Receives all packets at the receiving FPGA.
6. It takes out the control information field to control the switch.
7. It assembles the original slot by using the packet index.
8. Then it sends the slot out through the optical fiber to another edge node.

Note only the master node uses the "Switch_Control" field to control the switch, the slave node does not use it. The receiving side does the reverse operation except the control field. We do not need the switch control information in the reverse direction.

2.4. Network Model and Assumptions

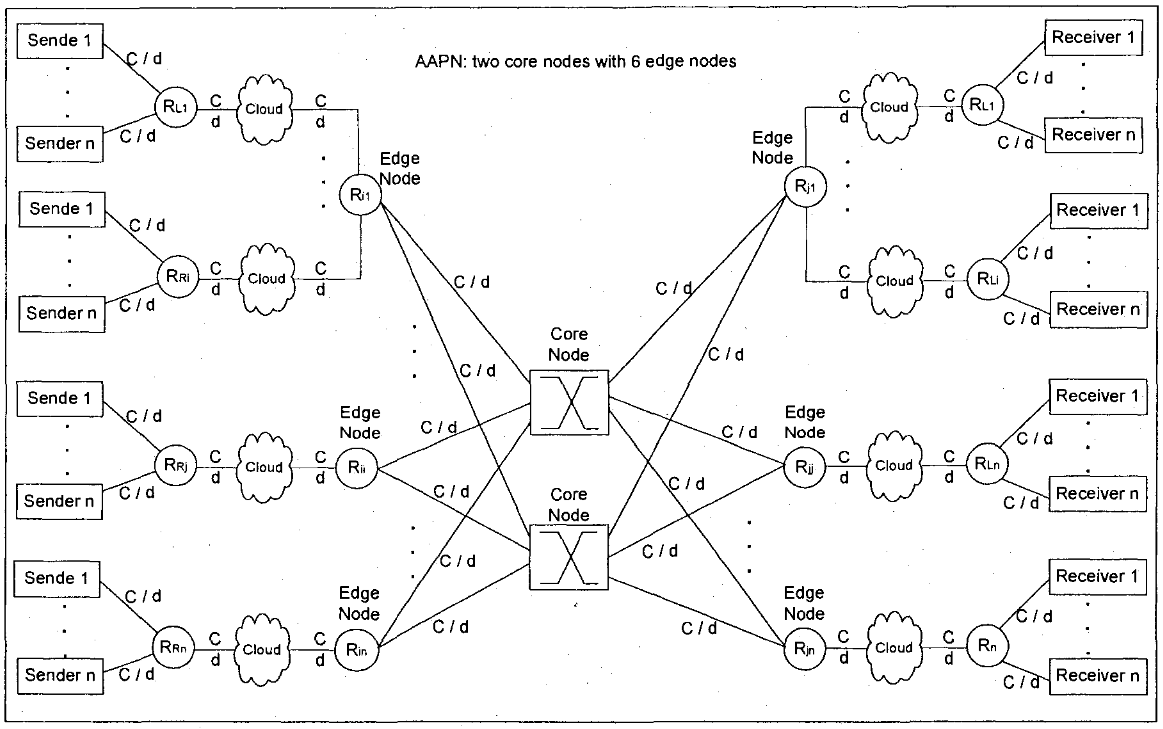

Figure 2.10 The AAPN Network Model with 2 Core Nodes and 2n Edge Nodes
Note: All the links are full-duplex. Each group of *n* senders and *n* receivers is different.

Figure 2.10 shows the network model with 2 core nodes and *n* edge nodes where each link has a capacity *C* and a delay *d*. This is a typical AAPN application in metro networks. Each edge node is connected to two core nodes via optic fiber. The edge node is also connected to one or more legacy networks (sub-network) and aggregates traffic from these sub-networks.

Based on our design, the following assumptions are used in our signaling implementation and XCP design & simulation.

1. A master node has negligible propagation delay to the core node because the master node is co-located with the core node.
2. The time drift of two edge nodes within one frame is negligible. This is guaranteed by clock precision used in PC or FPGA.
3. The propagation delay may change due to temperature. However this is slow when compared to the frame period as designed.

4. There is no optic buffer in a core node. This is reasonable considering the current optical enabling technology.
5. Each edge node is an ordinary electrical router connected to a legacy networks. It uses an electronic buffer and has an E/O interface for transmission to the core node.
6. The receiver buffer (advertised window) is big enough so it does not constraints the transmission.

Chapter 3. Signaling Design, Analysis and Implementation

In this chapter, we first present several choices of the signaling methods commonly used in telecommunication. Based on this analysis and our specific situation, we describe the frame and slot structures in our design, and introduce the synchronization method. These are used to implement a slow prototype from which more experiment measurement and their analysis are presented at the end. The following is the development environment we use to develop the slow prototype. Network topology has been introduced in the previous chapter.

1. Operating system is Linux with the following settings:
   Distribution is Mandrake Linux 2005
   Linux kernel version is 2.6.11
   GCC version is 3.4.3
   Eclipse version is 3.2
2. Edge nodes consist of 6 PCs equipped with optical Ethernet cards. One edge node is assigned as the master node. The others are slave edge nodes.
3. The core node is a BigBangWidth's [BiBa07] light path accelerator - an optical switch with optical Ethernet ports. It has an electrical Ethernet port that can be controlled through Telnet protocol.

3.1. Signaling Overhead Analysis

In this section we analyze the signaling overhead which is the ratio of the payload to the whole protocol data unit. We also compare the relationship of overhead and scheduling delay between frame-based signaling and slot-based signaling. It is clear that we desire an overhead as small as possible. Big scheduling delay degrades traffic estimation which we rely on to get a good scheduling result. So a small scheduling delay is desirable. There should be some trade off between signaling overhead and scheduling delay.

3.1.1 Frame-Based Signaling

Frame-based signaling means in each frame a slave node talks with a master node once to get a schedule and send a request.

According to our frame structure, in one whole frame, some slots (the number equals the system size) are reserved for signaling, the others are for data transmission. We can compute the signaling overhead by using the following formula:

$$\eta_1 = \frac{nodeSize}{frameSize} \tag{3.1}$$

where *frameSize* is the number of slots in a frame, *nodeSize* is the number of nodes which is the same with the number of slots reserved for signaling. This is because we reserve one slot for each node.

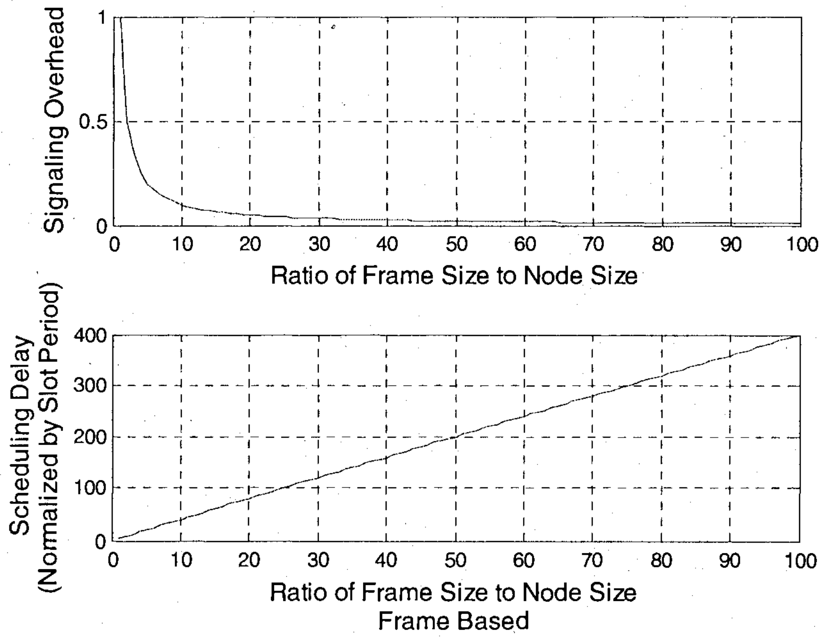

Figure 3.1 Frame Based Signaling

In Figure 3.1, the x-axis is normalized in terms of node number while y-axis is normalized in terms of slot period. We can see from the figure, when frame size increases, signaling overhead decreases too. It decreases very fast with the frame size increases from 1 to 5 in terms of node number. After that it decreases slowly. Scheduling delay is linearly proportional to the frame size. A good choice of the frame size is set it to 5 or 10 times of edge node number. For example, if we have 100 edge nodes, the frame size should be more than 500 slots but lower than 1000 slots. This would balance the signaling overhead and the scheduling delay. We also should choose a small slot period to get a small absolute scheduling delay. Note that in this case we preferred to use the length of the signaling protocol as the slot length. So there is no waste in the signaling slot because signaling data fully fills the whole slot. Another choice is that use different slot lengths. A short slot is for signaling and a long slot is for data transmission. However, this increases the complexity of the software.

3.1.2 Slot-Based Signaling

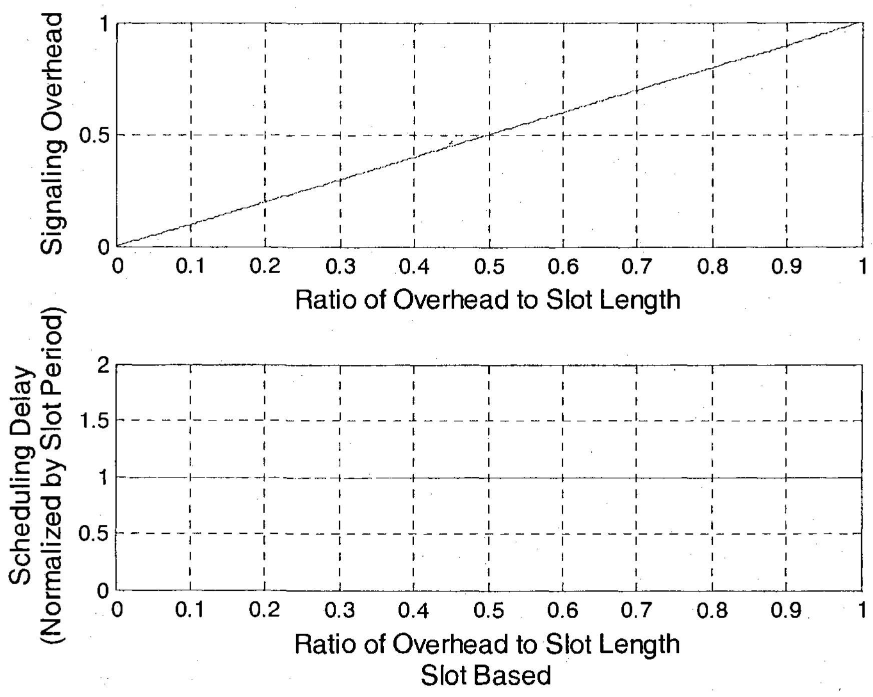

Figure 3.2 Slot-Based Signaling

Slot-based signaling means in each slot, a slave node talks with a master node to get schedule and send request. We can compute the signaling overhead using the following formula (variables are in bytes):

$$\eta_2 = \frac{overHead}{slotLength} \tag{3.2}$$

where *slotLength* is the total length of a slot in bytes, *overHead* is the length of the overhead in bytes.

As shown in Figure 3.2, the scheduling delay is fixed to one slot period, because in each slot an edge node signals to a core node. Signaling overhead is based on the ratio of the overhead to the total length of one slot. In our design, we have an overhead of 60 octets (the value we are currently using). If we desire a signaling overhead that is below 0.05, the slot length should be 60/0.05 = 1200 octets or larger.

In this case, the scheduling time is constrained to one slot period while in frame based scheduling, the scheduling time is relaxed to frame size deducts the number of edge node (equals the number of slots that reserved for signaling) times the slot period. If the slot period is small and the scheduling algorithm is a little complex, a frame based scheduling is preferred due to the scheduling time constraint.

Based on the discussion, frame-based scheduling has the flexibility of scheduling time. We also desired to integrate QBvN algorithm which is based on frame-based scheduling to do some experiments and measurement, hence we use frame-based scheduling in our design.

3.2. In-band Signaling and Out-band Signaling Discussion

We have two choices for the signaling: one is in-band signaling which uses a dedicated slot in the data transmission fiber to exchange control information between a slave node and a master node. The other is out-band signaling which uses a dedicated control channel fiber or wavelength to exchange control information.

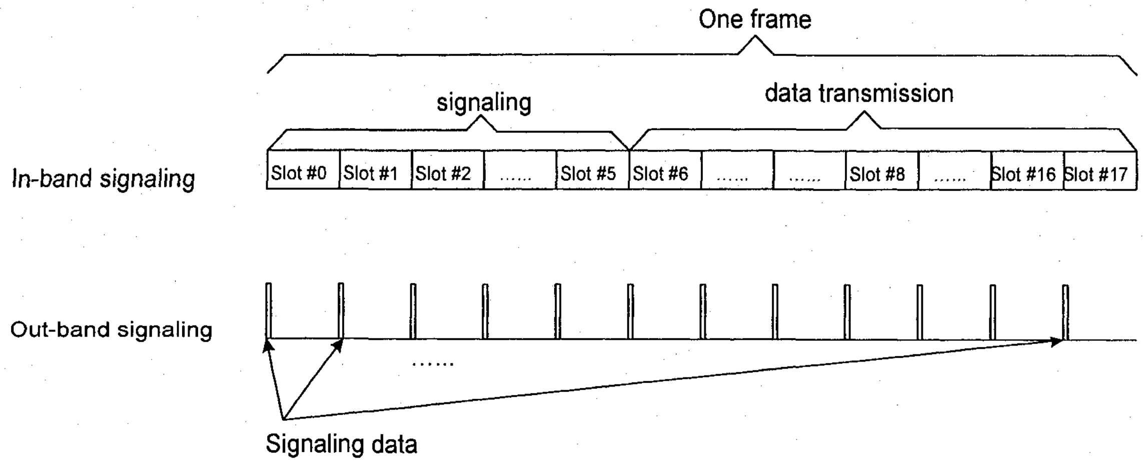

Figure 3.3 In-band and Out-band Signaling

Figure 3.3 shows the slot structures of these two signaling methods. In-band signaling only uses some slots for signaling. The master node gets all the traffic information at the end of slot 5 shown in this figure. There is always a frame period delay of traffic information. In this case we should use an estimation method to predict traffic. The benefit is that the design is simple and it is easy to implement it.

Out-band signaling uses an extra fiber. In each slot time the master node can get the traffic information from all slave nodes. Hence the traffic information is "fresh" and real-time compared to in-band signaling. The scheduler can be more accurate for bandwidth allocation. But the design and implementation is complex and some bandwidth is wasted (note we only use a very small part of bandwidth of the signaling fiber). Due to our current core switch only has one wavelength for each fiber and there is no dedicated extra fiber for control channel, we use in-band signaling in our design.

3.3. Data Structure Design

This section describes the details of the data structure design. In order to design and implement the signaling protocol, we should design the frame structure, slot structure and the related data structures. All the information exchanged between any two edge nodes uses the corresponding data structures.

3.3.1 Frame Structure

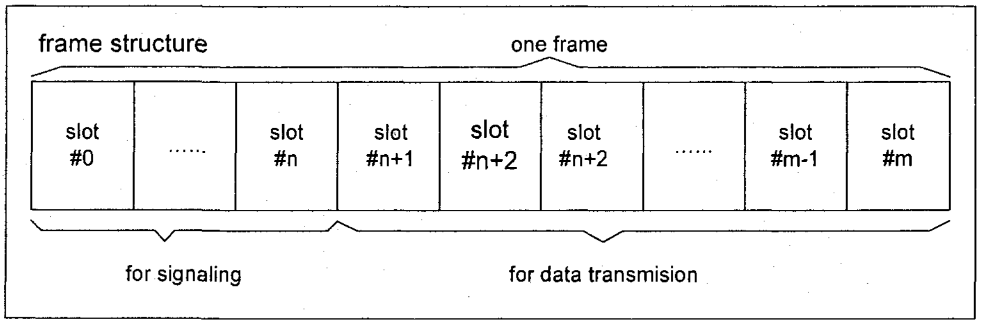

Figure 3.4 Frame Structure

Frame structure is shown in Figure 3.4. Each frame is constituted with a number of *m* slots. The first n slots are reserved for signaling while the other *m-n* slots are used to transmit data (Assume the number of slave edge nodes is *n*). So in each frame each slave node has one slot dedicated to communicate with the master node for signaling.

3.3.2 Slot Structure

Data is transmitted in slot format through optical fiber between any two edge nodes via a core node.

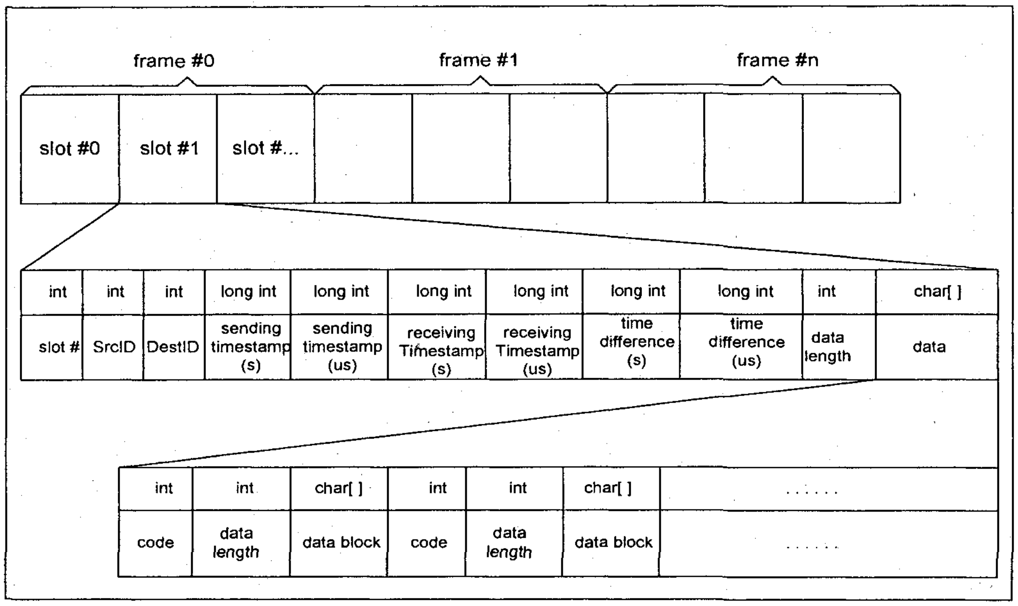

Figure 3.5 Slot Structure

The structure of the slot format is shown in Figure 3.5. This structure defines how the slot is constituted by different fields. It is implemented in C language as follow:

```
struct SLOT
{
   int slotSequence;      //slot sequence number
   int SrcID;             //source node ID number
   int DestID;            //destination node ID number
   long long int sendingTime;          //64 bits
   long long int receivingTime;        //64 bits
   long int timeDifference;            //master - slave
   int dataLength;                     //data length in data[]
   char data[MAX_SLOT_DATA];
 };
```

The meaning of each field is usually self-explained by its name. "slotSequence" is the slot sequence number. "srcID" and "DestID" are source node ID number and destination node ID number separately, because we need to track where the slot comes and where it should go. “sendingTime” and “receivingTime” (include second part and micro second part, “timeDifference” is the same) are the sending time at a source and the receiving time at a destination. We use these two fields to compute and update the time difference field for synchronization purpose. They are 64 bits in length which is long enough to avoid wrap back problem. “timeDifference” is the clock difference plus the propagation delay between two edge nodes. This value is used for synchronization purpose. “dataLength” is the length of the data in data[] field in bytes.

Note the field “data []”. It self has a sub-structure architecture. The messages (described in detail as follow) have code (indicates the type of data), length and data block (the upper layer data) format. They are encapsulated in this field.

3.4. Synchronization and Network Configuration Protocol

This section describes the synchronization and the network configuration protocol. As we said before, because the core node is buffer-less, all edge nodes should coordinate to transmit their traffic. Otherwise the traffic passing through the core node will collide or will be dropped by the core node. The principle of how to synchronize all the edge nodes with different propagation delay is critical in AAPN design [RaYa03] [RaYa05c].

3.4.1 Synchronization Principle

Each time we require the slot from different edge nodes with different propagation delay to arrive at the core node at exactly the same time. So the core node can connect the desired connection and let the data slots go through it within the allocated time slot. Otherwise, the data will be dropped or corrupt.

The first problem to solve is how to measure the propagation delay and the clock difference between the master node and the slave nodes. These values are required by the synchronization mechanism.

When a slave node sends a slot to the master node, it stamps the sending time field T_{send_slave} (slave local clock). This slot goes to the master node through the optical fiber with some delay and arrives at the master node at time $T_{receive\_master}$ (master local clock). Propagation delay plus time difference (sum together as T_d) between these two nodes is

$$T_d = T_{receive\_master} - T_{send\_slave} = T_{clockdiff} + T_{delay} \tag{3.3}$$

$$T_{clockdiff} = T_{master} - T_{slave} \tag{3.4}$$

where $T_{receive\_master}$ is the master receiving time, T_{send_slave} the is slave sending time, $T_{clock\text{-}diff}$ is the time difference between the master and the slave, T_{delay} is the propagation delay, T_{master} is the master's local time and T_{slave} is the slave's local time. We do not need to measure $T_{clockdiff}$ and T_{delay} separately, we only need their sum as we can see in the following computation.

When a master asks a slave node to send data at time T_{send_slave} which is expected to arrive at the core node at T_{master} (master local time) or T_{slave} (slave local time), so the slave node should send its data at:

$$T_{send\_slave} = T_{master} - T_d = T_{master} - (T_{master} - T_{slave} + T_{delay}) = T_{slave} - T_{delay} \tag{3.5}$$

This equation has two meanings. The first meaning is that the slave sending time T_{send_slave} is the master's desired receiving time T_{master} minus T_d (the sum of the time difference and the propagation delay), i.e. the second term in the equation. The second meaning is a slave should send its data at T_{slave} - T_{delay}. After the data transmits a time of T_{delay}, it arrived at the master at the expected time, which is T_{master}.

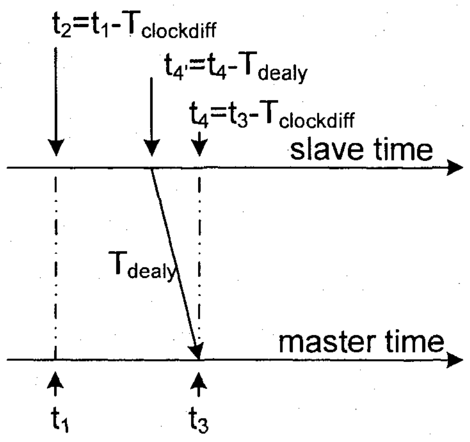

Figure 3.6 Sending Time Computation

Note: *t* means time instant, *T* means time variable with a specific value.

The computation is shown more clearly in Figure 3.6. Time instants t_1 and t_2 are at the same time instants but in different axis (the same meaning in time zone); t_3 and t_4 are the same. Assume a slave sends a slot at $t_{4'} = t_4 - T_{dealy}$. This slot will take T_{dealy} to propagate and arrive at the core node at t_4. This t_4 will be the same instant as t_3 which is we desired in master time scale.

Note that some clock drift with time can occur between two nodes. This is resolved by dynamically updating the T_d value through our signaling in each frame. (We assume that the time drift of two edge nodes within one frame is negligible.)

For the purpose of precise measurement, the sending time stamp must be set at exactly the sending instant. The receiving time stamp must be set at exactly the receiving instant. In a PC using Linux operating system, we can not get a precise measurement because of the processing delay and the operating system scheduling delay. We will talk about this later.

3.4.2 Sample Signaling Timing Sequence

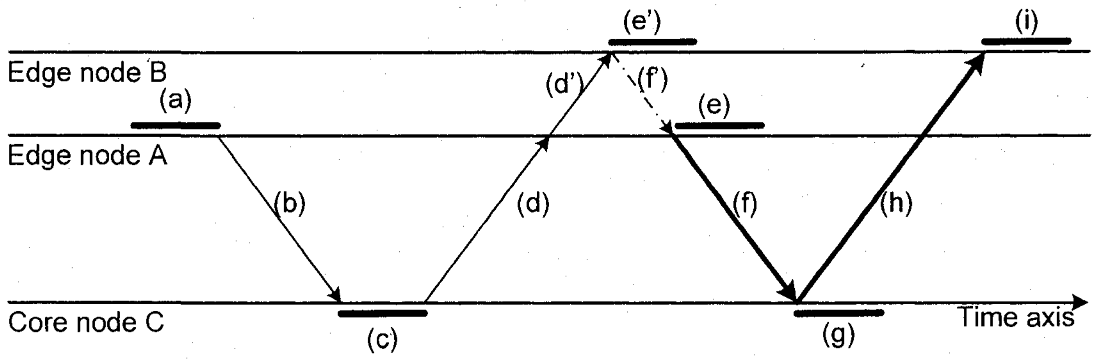

Figure 3.7 Timing Sequence of Frame-Based Signaling

Figure 3.7 shows a typical timing diagram of the frame-based signaling for the case of edge Node A sending some data bursts to edge Node B via a core Node C [Peng07]. A sample signaling scenario is described as bellow.

(a) Node A collects the traffic information from outside, and aggregates it into slots during the current frame.

(b) Node A sends this traffic information to Node C.

(c) Node C calculates the future schedule after it receives all the traffic information from all slave edge nodes if any.

(d) Schedule is then sent to Node A.

(d') Schedule is also propagated to Node B who receives it at time e'. (Similarly to all other nodes each with different propagation delays if any).

(e) Node A waits until the right time (after Node B receives the schedule) due to the synchronization required between the edge nodes, i.e. some other edges may be further away from the core node and requires more time to receive the schedule (see line d').

(e') If Node B has data to send to Node A, it should send it before Node A does. Hence all the data will arrive at Node C at the exact time.

(f) Some data bursts are sent from Node A destined to Node B.

(g) When the bursts pass through the core node, it is re-configured as desired to transfer these bursts to Node B.

(h) The bursts travel to Node B.

(i) Node B has the information at what time the bursts will come (according to the schedule) and receives the data.

3.4.3 Slot Assignment Explanation

In AAPN, we desire the property of dynamically allocating bandwidth to traffic flows as the traffic demand varies, which results in rapid reconfiguring the all-photonic space switch in the core node.

We use a frame scheduling and consider 6 slave edge nodes, one master edge node (which has a scheduler and co-resides with a core node) and one core node.

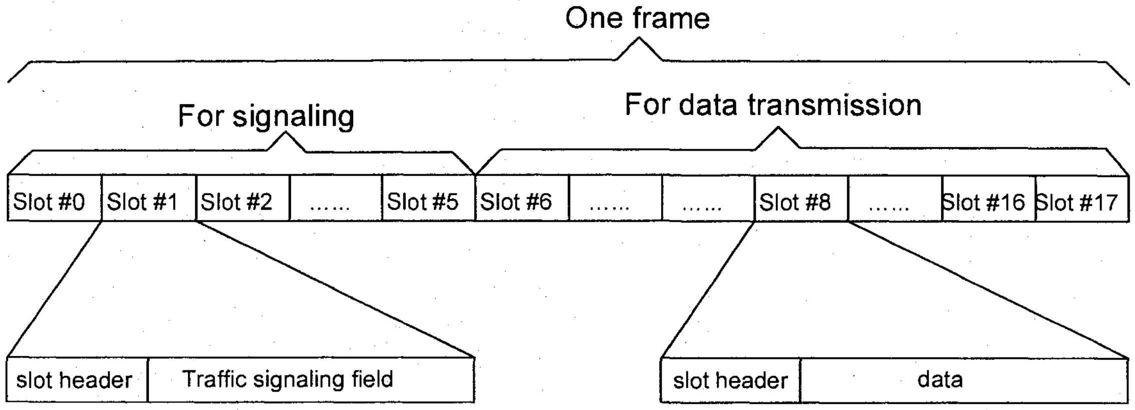

Figure 3.8 AAPN Slot Assignment Explanation

We use a frame scheduling to allocate bandwidth, and in-band signaling through a dedicated slot to coordinate data transmission over a fiber. Figure 3.8 is an example of a frame structure to support 6 slave edge nodes with ID#1-6 and a master node with ID #0. We use the centralized OTDM method [Peng07] [RaYa05] and each frame has 18 slots. The first 6 slots were reserved for signaling (note that a master node does not need a slot for signaling) and the last 12 slots are for data transmission. Each signaling slot has a control field (i.e. the slot header) and a traffic signaling field. The traffic signaling field from a slave node to a master node is called "traffic request", and that from a master node to a slave node is called "traffic allocation".

Each slave node signals in one slot of a frame in order to communicate with the master node and uses some other slots for data transmission. For example, through slot#0 in a frame, edge node#1 can send its traffic request to and receives traffic allocation from the master node. Likewise, edge node#6 can communicate with the master through slot#5, etc. After a master node receives all traffic requests (by the end of slot#5), it executes the scheduling algorithm to allocate bandwidth to each slave node by assigning slots. All these new scheduling information will be sent out in the next frame.

Since slave nodes only receive and use the schedule in the next frame, there is always some delay. That is, scheduler schedules the allocation according to the requests in the previous frame. The edge nodes receive and use the schedule to send out traffic in a future frame. This can be compensated by an estimation/prediction method [FeSh04] [ShYu05] of future traffic. Anyway, an edge node can have the exact allocated bandwidth information that it can use when sends out traffic.

3.4.4 System Starts Up

When the system starts up, the master version software prints out some system information such as the software version, the software title. It reads out some configuration information such as system size (defined to be the number of edge nodes), switch size (defined to be the number of ports), frame size (defined to be the number of slots in a frame), slot period (defined to be the duration in seconds of a slot) etc. from a configuration file.

The slave version software prints out some system information and waits for invitation message from a master node.

3.4.5 Invitation Procedure

After the AAPN master software starts up, the master node invites each slave edge node to join the network. It sends an invitation message to each slave node at a time, waits for the response from a slave edge node. When a slave node receives an invitation message, it sends back response message (same structure as invitation message) immediately. If within a time threshold, the master node can not receive a response, it thinks the slave node is not start up and sets this slave node's status as inactive.

Edge Node ID field is used in the message to distinguish which edge node to be invited in case there is a combiner/splitter. In this case, the number of edge nodes is larger than the number of switch ports, two or more edge nodes are connected to one switch port. We must indicate which edge node is to be invited by using the ID number. Through this method, we can solve the scalability problem.

In this process, we also measure the propagation delay from a slave edge node to a core node plus their clock difference. When a slave node receives invitation message, it puts its local clock T_s to the sending time field in respond message. When the master node receives this message, it uses its local clock T_m minus T_s. This value T_d is the propagation delay plus clock difference between the master and the slave edge node. This value will be sent to the slave node later for synchronization purpose.

3.4.6 Configuration Procedure

After all slave nodes are invited, the master node enters this process. In this process, the master node sends configuration messages to all slave nodes. It tells the slave nodes what system parameters were used currently.

The message includes slave node status (active or inactive), so each slave knows who are currently presented in the networks. Hence salve edge nodes know how the network is configured, to whom they can send traffic. It also includes switch size, slot period, frame size, master start time information, hence slave nodes can compute the exact sending time to synchronize to the master node. Other system parameters such as switching guard time are also sent to each slave node.

3.4.7 Normal Working Procedure

After the configuration process, all slave edge nodes can exchange control information with the master and exchange data among themselves (including the master node) via the core node. The master node would reconfigure the optical switch according to the schedule result.

A master creates three threads: (1) a receiving thread to receive traffic including signaling and data slots, (2) a sending thread to send traffic including signaling and data slots, (3) a scheduling thread to schedule allocation after it receives all requests. These threads repeat the “receiving request”, “schedule”, “sending allocation”, “sending data”, “receiving data” iteration all the time until the system stops.

A slave node creates only sending and receiving threads to finish the “sending request”, “receiving allocation”, “sending data”, “receiving data” iteration.

3.5. Traffic Allocation Protocol

All slave nodes and the master node exchange traffic request and allocation information defined in this protocol.

3.5.1 Traffic Request Message

This message is sent from a slave node to a master node to report the traffic (in slot unit) which it wants to send to other edge node (a master node has its own traffic information). By this message, it asks the master to assign some time slots to send its traffic.

3.5.2 Traffic Allocation Message

This message is sent from a master to a slave node to tell it which slots in a frame are assigned it to send traffic. When slave nodes receive this message, they know in what slots and at what time to send out traffic to whom.

3.5.3 Traffic Request and Allocation Matrixes

Traffic request matrix:

```
int  trafficReq[NUM_EDGE_NODES][NUM_EDGE_NODES];
```

The first index indicates the traffic source, and the second index indicates the traffic destination.

Traffic allocation matrix:

```
int  allocation[NUM_EDGE_NODES][FRAME_SIZE];
```

The first index indicates the traffic source, and the second index indicates which slot in a frame should send out the traffic.

A careful reader should notice the little difference of the second index in these two matrixes. In allocation matrix, the slave node only needs to know which slot number in a frame it should send out traffic. That's all. The master node will configure the core node switch correctly, when the traffic comes from a slave node arrives, it will direct the traffic to the desired destination edge node.

3.6. Fault Monitor Protocol

When a master node and slave nodes exchange control information, they can also exchange fault monitor message. More information on fault detection and localization can be found in [ZhPe06].

If a master needs to shut down or has some other faults, it broadcasts the message to all slave nodes, so slave nodes can select another master if possible.

When a slave node has some faults such as power failure, no signal, synchronization error to report to the master, it sends this message to the master node to report the fault it has. So the master node can take some actions and broadcasts it to other slave nodes.

3.7. Software Architecture

In the previous sections we described each protocol separately. Now we take a look at the whole structure of the software.

Figure 3.9 shows the software architecture of a master node and a slave node. Modular design method and software engineering [Vlie00] method are used in the whole software design project. Each block is a module to decouple the functions for easy maintenance and software evolution. It is clear how all the functional modules are integrated and in what way they interact with each other. We describe them as follow.

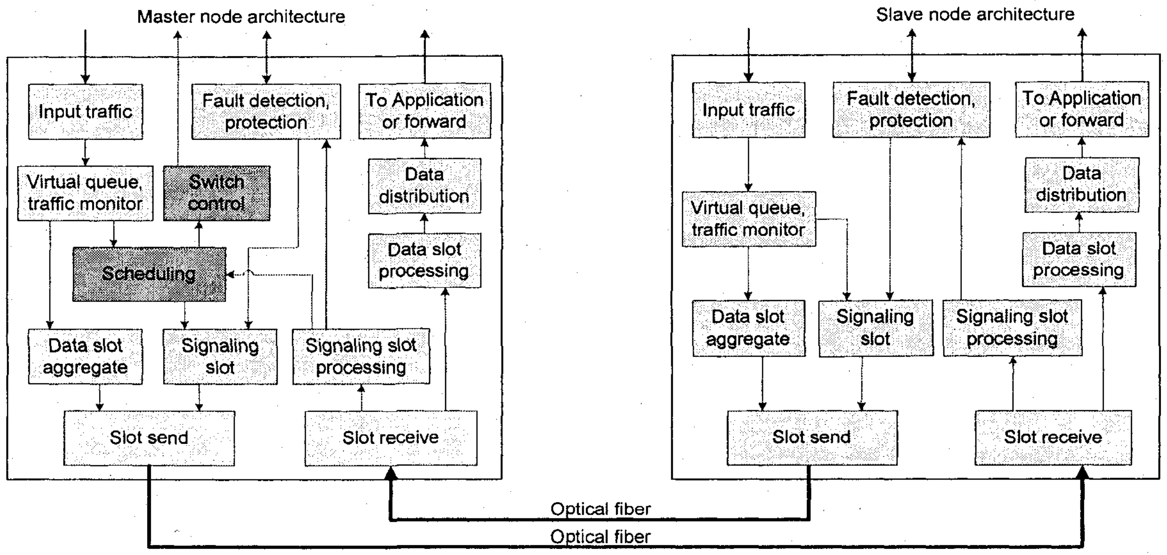

Figure 3.9 Software Architecture of a Master Node and a Save Node

In the master node architecture, the "Input traffic" module receives traffic from outside AAPN. Then the "Virtual queue, traffic monitor" module puts this traffic into a queue, and records the traffic information for monitoring use. When the time comes for this node to send data, the "Data slot aggregate" module takes an item from the queue, adds some sending control information and aggregates them into a slot. This slot is then sent out through the "Slot send" module via optical fiber.

The "Slot receive" module receives a slot if any. If one is received, then different modules are used depending on the type of the received slot. The "Signaling slot processing" module processes it if it is a signaling slot. The "Data slot processing" module processes it if it is a data slot. This module will deliver the data to the "Data distribution" module, then passes it to the up layer or forwards it by the "To application or forward" module. Further for the signaling slot, if it is about traffic information, this information is sent to the "Scheduling" module. If it is about fault, it is sent to the "Fault detection, protection" module for further processing. The "Fault detection, protection" module also detects any faults and passes this information to the "Signaling slot" module for further processing.

The "Scheduling" module receives all the traffic requests from all slave edge nodes, runs a scheduling algorithm to schedule traffic allocations. The results are passed to the "Signaling slot" module for transmission. The scheduled result is also passed to the "Switch control" module, which reconfigures the core switch connection as desired.

In the slave architecture, there is no "Scheduling" or "Switch control" module. All the other modules have the same function as in the master node with the exception as follow. The "Virtual queue, traffic monitor" module passes traffic information directly to the "Signaling slot" module for processing. The "Signaling slot processing" module only need to pass fault information to "Fault detection, protection" module.

The interface of virtual input queue and scheduling algorithm are clearly defined in our software. It is easy to add new algorithms to them, so we can compare and evaluate the performance of different algorithms.

3.8. Measured System Parameters and Analysis

Some system parameter measurements are shown in this section. We are interested in the synchronization precision and scheduling time of each frame.

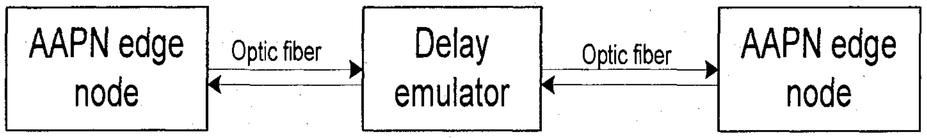

Figure 3.10 Logical Test Network Configuration (Core Switch is not Shown)

3.8.1 Synchronization Parameters

Figure 3.10 shows the logical test network configuration. We developed a program to simulate optical fiber propagation delay. The principle is like this: each time a receiver receives a slot, it stores the whole slot in memory, wait for a specific time T_{delay}, then sends it to another edge node. T_{delay} can be configured in the command line. The purpose of this program is to simulate the propagation delay and see if our program works for different propagation delays. The better way is to use a fiber delay to measure the synchronization parameter.

We set different delay parameters as shown in Table 3.1 to evaluate our synchronization protocol.

Table 3.1 Delay Parameters for Synchronization Measurement

	Set and Expected delay (s)	Measured delay(s)
Case 1	0.0000	0.0015
Case 2	2.0000	2.0144
Case 3	6.5000	6.5024
Case 4	17.0000	17.0031

Figure 3.11 to Figure 3.14 show measured synchronization errors (expected value minus measured value) of the three cases. A negative error means a receive slot arrives at the core node after the expected time, while positive error means slots received by the core node are early than the desired time. Figure 3.11 shows large spike value than the other two cases. Possibly this is because receiving and sending threads work at almost exactly the same time, so squeeze the computer system processing capability.

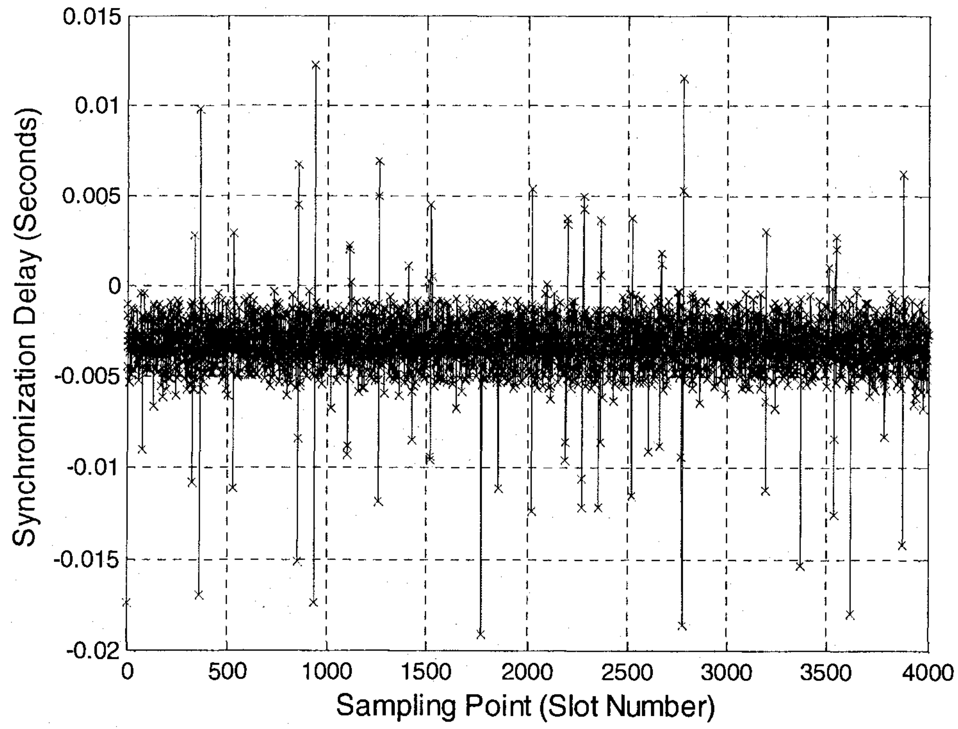

Figure 3.11 Synchronization Measurement Case 1

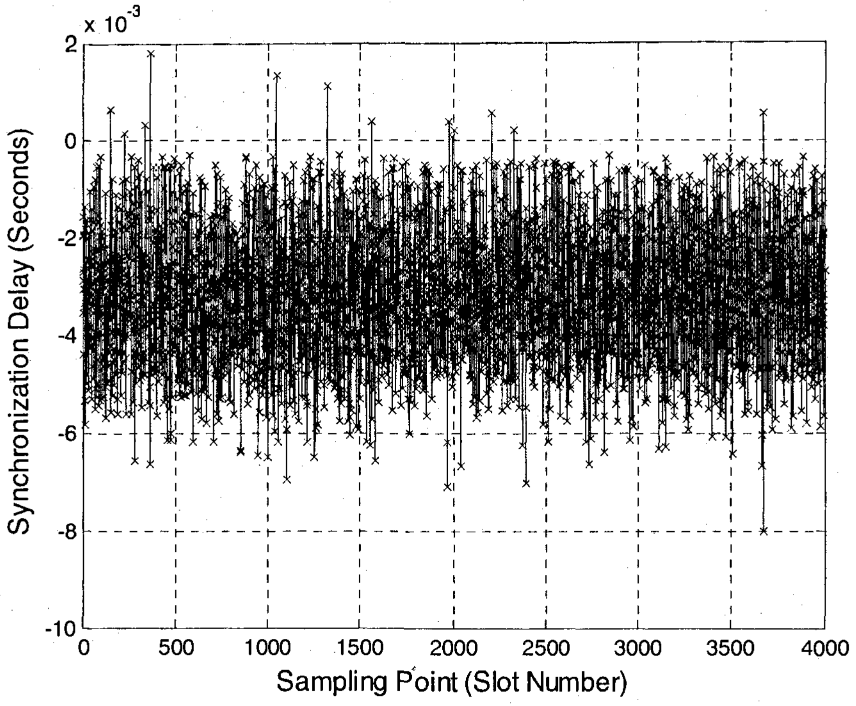

Figure 3.12 Synchronization Measurement Case 2

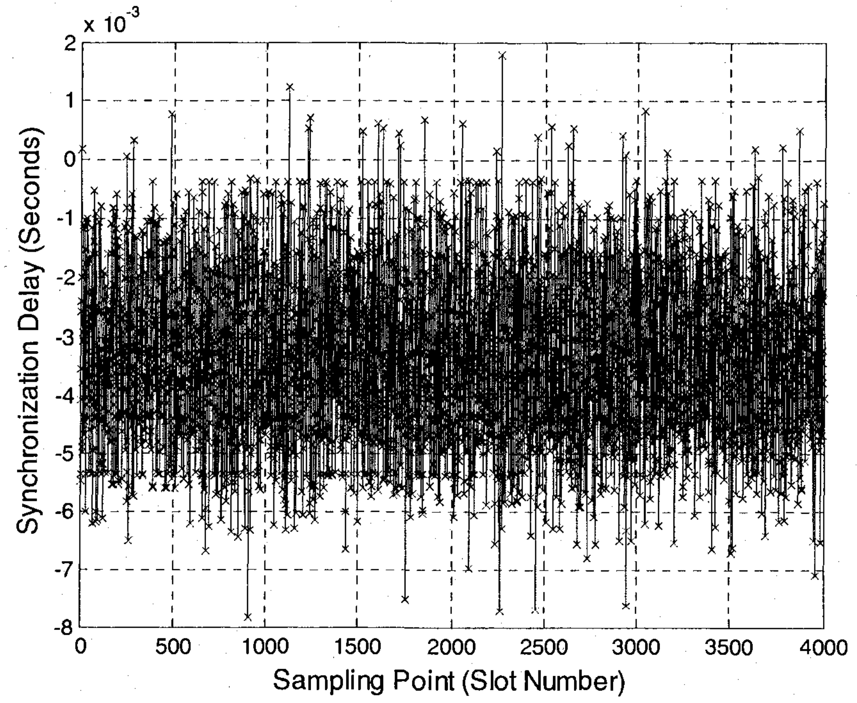

Figure 3.13 Synchronization Measurement Case 3

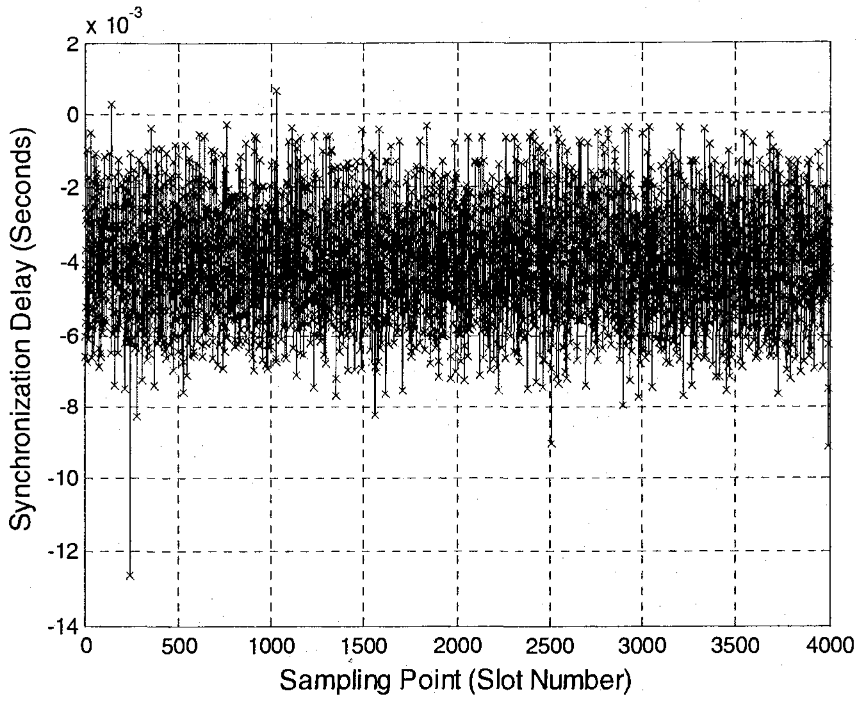

Figure 3.14 Synchronization Measurement Case 4

Table 3.2 Synchronization Error Measurement

	Maximum (ms)	Minimum (ms)	Average (ms)
Case 1	12.303	-19.111	-3.139
Case 2	1.757	-8.025	-3.148
Case 3	15.180	-21.696	-3.350
Case 4	0.658	-12.651	-3.978

Table 3.2 shows the synchronization error measurement of these threes cases. As average values in all cases are negative, it means slots always arrived after the expected time. This is intuitively correct due to that there is always a delay. This is because the software processes the schedule first then sends out the data. It resulted in the processing delay. The maximum and minimum values are random in a sense. This is because the OS (Operating System) is not deterministic and we can not assure the timer function (a system call) is fired as exact as we desired. In all cases slots arrive at the core node in tolerance range (the guard time must larger than the maximum value the oscillation), so they

can be transmitted through core node without any data loss.

Note the big peak shoot at times. This is because the operating system flushes data to disk or runs other tasks at the same time. Sending and receiving timers can not be fired at the exact desired time. Note also the propagation delay simulator is a piece of software, it introduce another processing delay and uncertainty.

Linux is not a real time OS, so some critical timing is not as precise as we expected. System response such as timer is not deterministic. It depends on the workload of the system. We set the threads related to time stamping in high priorities which mainly impact the synchronization. However, this does not improve the performance as much as we desired. Commercial real time OS such as VxWorks, QNX or OSE perhaps can fulfill this requirement. Hardware solution is another possibility and a better one.

3.8.2 Slow Prototype Synchronization Problems and Solutions

As seen from the above section, synchronization precision is not good when we implemented the system under Linux because of the undetermined processing delay and OS scheduling uncertainty. Since the root reason is the indeterminate processing delay, our solution is to use FPGA hardware for synchronization processing.

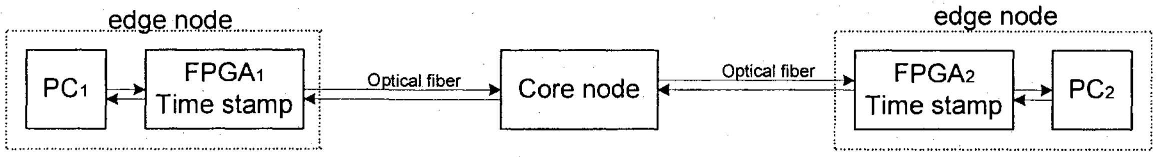

Figure 3.15 System Architecture of FPGA Version A

Figure 3.15 describes the logical architecture of the system (detailed FPGA architecture was described in Section 2.3.2). PC and FPGA inside the dot line constitute the edge node. In this case we only need to measure the time difference and propagation delay between the two FGPA (each FPGA has a clock). This is because FPGA is the real sending and receiving device that should coordinate with the core node.

The principle is as follow: PC_1 sends out a slot to $FPGA_1$, $FPGA_1$ timestamps the sending time. Then it sends out the slot to $FPGA_2$ through the core node, $FPGA_2$ receives this slot and timestamps the receiving time. Note now the time stamping process is done by FPGA, it is deterministic. This slot goes to PC_2. PC_2 then computes the time difference plus the propagation delay which sums to T_d. At PC side, PC only computes T_d and processes signaling protocol. Note we do not care if there is some delay between PC_i and $FPGA_i$ on condition that PC_i sends slot to $FPGA_i$ before the required sending time as indicated by the scheduling at which the slot should be sent out. This assures the slot arrives at the core node at the desired time.

A picky reader will ask if there is still some processing delay. Theoretically there is some delay. However, this delay is negligible compared to our time precision which is in micro-second order. The good thing is that even it is not negligible we can compensate it. Notice this delay is determined by the design of the hardware system. When we finished the design, the delay is fixed. So we can compensate this delay by deducting this value from our measured time difference.

The rule is using deterministic technique to process the timing, hence eliminate the undetermined operating system processing time. If the processing time is fixed and has some delay, we can compensate it to achieve a good synchronization. The other protocol is not time critical, and could use general purpose CPU or micro controller to process it. Hence we can take advantage of the both: the quick and fixed processing time of FPGA and the great computing power of CPU.

3.8.3 Scheduling Time Measurement

Table 3.3 shows the scheduling time with different system sizes and frame sizes with a slot period of 250us. We will plot a curve of this data later for a clearer view and easier analysis. We care about the maximum scheduling time because scheduling must be finished in each frame even in the worst case. The scheduling time is constrained by the slot period and the ratio of the frame size to the system size.

The time constraint T_g (guard time) is the slot period multiplies the difference of the frame size and the system size.

$$T_g = (N_f - N_s) * T_s \qquad (3.6)$$

If we set N_f (frame size) to 16, N_s (system size) to 32, T_s (slot period) to 250us, we have T_g =(32-16)*0.25 = 4.0ms for the scheduler to finish the schedule algorithm (see the first row in the table). Actually we can set the slot period to 0.203ms/16 = 12.7us in this case. For a system size of 64 edge nodes and a frame size is 128, we have T_g = (128-

64)*250us = 16.0ms.

Table 3.3 Scheduling Time vs System Size and Frame Size

System size (nodes)	Frame size (slots)	Allowed time (ms)	Max scheduling time (ms)	Min scheduling time (ms)	Mean scheduling time (ms)	Result
16	32	4	0.203	0.072	0.103	OK
16	64	12	3.503	0.102	0.130	OK
16	128	28	2.948	0.141	0.177	OK
32	64	8	13.161	0.909	11.158	OK
32	128	24	13.357	1.185	11.297	OK
32	256	56	14.467	1.849	11.319	OK
64	128	16	14.617	3.206	12.058	OK
64	256	48	15.278	3.347	12.104	OK
64	512	112	15.790	5.441	12.546	OK

Note: OK means measured scheduling time is less than the allowed time.

Note we should use the maximum measured scheduling time to compare. This is because we must finish the computing in the worst case. The time constraint can be relaxed if we use a larger frame size. As we will see later, the rule is that we should set the frame size bigger than 3 times the system size to achieve high protocol efficiency and scheduling flexibility. So the scheduler can have enough time to compute the allocation.

Our new switch aims at a slot period of 210us, the fastest speed it can has. When the frame size is more than 3 times the system size, this requirement can be fully fulfilled. It is computed as follow: T_g = (64*3-64) * 210us = 26.88ms

It is larger than any of the measured scheduling time. We draw a curve of this data for a clearer view and easier analysis.

The curve of scheduling time vs the system size and the frame size is shown in Figure 3.16. One can see from the figure, with a fixed number of nodes (16 nodes, the bottom curve), the scheduling time increases when the frame size increases, from 0.103ms of 32 frame size to 0.130ms of 64 frame size, and 0.177ms of 128 frame size. Curves with other system sizes have the same behavior.

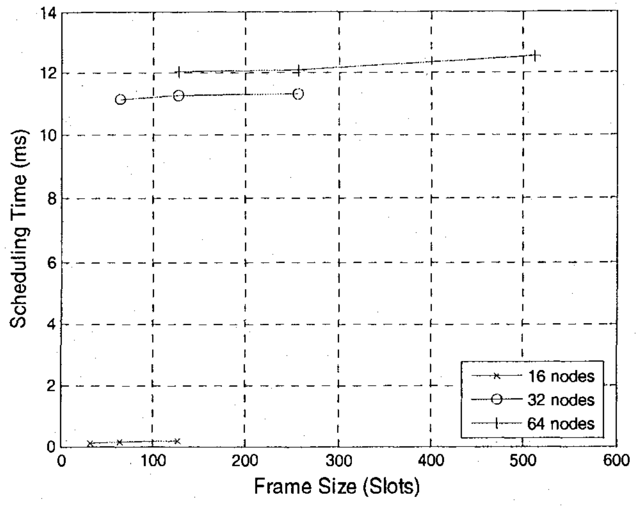

Figure 3.16 Scheduling Time vs Frame Size

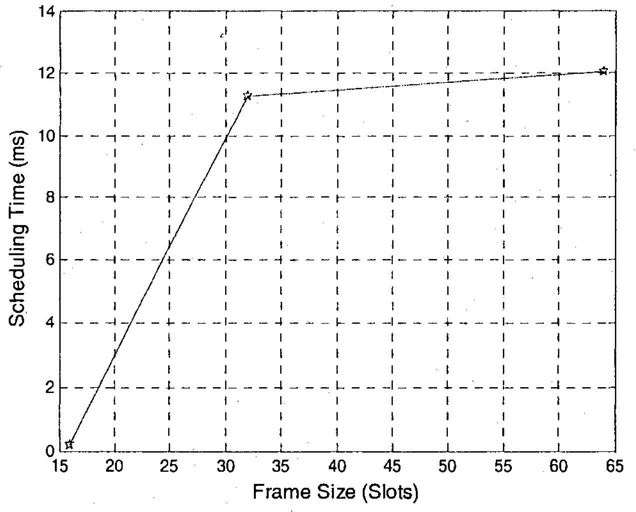

Figure 3.17 Scheduling Time vs System Size with Frame Size Fixed at 128

With a fixed frame size, the scheduling time also increases when the system size increases. Also note that scheduling time increases very fast when system size increases from 16 to 32. But it is much slower when it increases from 32 to 64. This is because of the intrinsic property of the scheduler algorithm. See [Peng07] for more details.

Figure 3.17 shows the curve of scheduling time vs system size while we fixed frame size to 128 for a clearer view. When system size increased from 16 to 32, the scheduling time increased from 0.177ms to 11.159ms. The scheduling time increased from this value to 12.058ms when system size increased from 32 to 64. This says when system size is small, scheduling time increases very fast. When system size is big, it increases much slower. Our measured result has the same trend as said in [Peng07].

3.9. Concluding Remarks

Based on the analysis of the frame-based and slot-based signaling overhead and discussion of in-band signaling and out-band signaling, we have chosen frame-based and in-band signaling as our design. We have successfully designed and implemented the AAPN control platform, including synchronization signaling, bandwidth allocation signaling, traffic monitor protocol and fault monitoring protocol. We have also integrated the QBvN scheduling algorithm to test the scheduling time.

We measured the synchronization precision and the scheduling time with different frame sizes and slot sizes. Both measurements meet our design requirement with a reasonable guard time between slots for safe data transfer.

Chapter 4. AAPN Traffic Control

Having discussed the implementation of AAPN signaling protocol, we now consider the design of our congestion control. We shall extend the XCP algorithm using a cross-layer design. Simulation modeling and environment are provided.

4.1. Crosse-Layer Design of XCP-CL Algorithm

We first introduce the XCP congestion header, the method to pass the real time bandwidth change information to the XCP algorithm by cross-layer design then provide step-by-step procedure of the algorithm, an example is provided after. Simulation network model is provided at the end.

4.1.1 XCP-CL Congestion Header

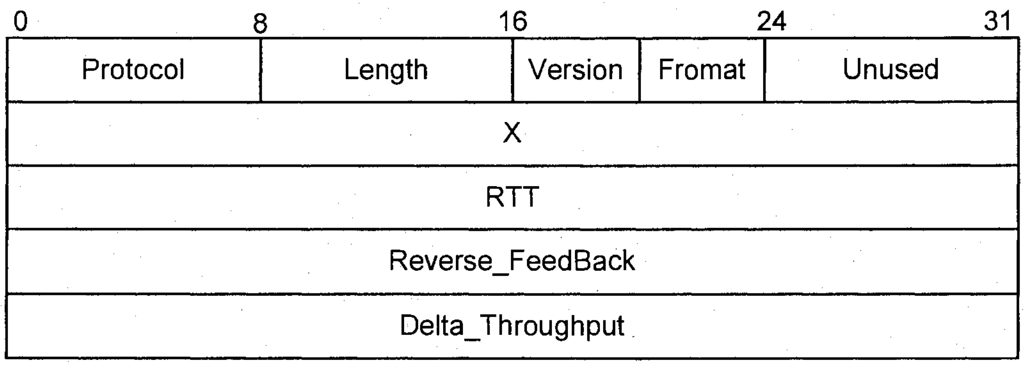

Figure 4.1 XCP-CL Congestion Header

As discussed in motivation and methodology in Chapter 1, the original XCP algorithm needs to know all the congestion information along the data path, and assumes the link bandwidth does not change with time. However, AAPN core node works in TDM mode and reconfigures according to the scheduling of edge node requests. As a result, the core node bandwidth changes frequently. To resolve these problems, we need to modify

XCP algorithm and take advantage of AAPN signaling protocol. We use the same XCP congestion header described as shown in Figure 4.1 to pass the bandwidth change information to the XCP algorithm.

Our XCP-CL uses the "Delta_Throughput" field to bring congestion information to the receiver. This field is initialized by the sender with the desired value and can be modified by any routers through the path. At each router, the XCP-CL algorithm makes use of an EC (Efficiency Controller) and an FC (Fairness Controller) to allocate bandwidth based on congestion at the current router. The details of EC and FC can be found in [KaHa02a] [KaHa02b] [Kata03], and are summarized in Appendix A. After receiving a packet, the receiver sends this information back to the sender to adjust its sending rate through "Reverse_Feedback" field. "RTT" field is the estimated RTT (Round Trip Time). We do not use other fields of the header such as protocol type, length, version and format. Their discussion is omitted.

4.1.2 Passing Real Time Bandwidth Information

As described before, a scheduler allocates bandwidth to each edge node using an AAPN signaling protocol in the data link layer. Hence each edge node knows the bandwidth allocation in terms of the number of time slots. We may take advantage of this signaling protocol.

After receiving the allocation information, an edge node can compute the assigned bandwidth using the frame period and the transmission speed. It computes an equivalent BW (BandWidth, the actual bandwidth) in Mbps using the following formula:

$$BW = N_S * C * T_S / T_f \tag{4.1}$$

where N_S is the number of assigned slots, C is link capacity in Mbps, T_f is frame period, T_S is slot period. Note: T_f / T_S is the frame size in slots. Then the XCP-CL algorithm can use this equivalent BW to compute allowable bandwidth for each flow using the EC and FC controllers. When an edge node receives an XCP-CL packet, it compares the computed result with "Delta_Throughput" field in the XCP-CL header and updates this value when the computed result is less.

So after the packet passes through the whole data path and returns to the sender, it carries the bottleneck information of the path. The sender can adjust its sending rate according to this information to avoid congestion. Hence XCP-CL algorithm can work correctly in AAPN.

Note that this design is a cross-layer design. The core node sends out bandwidth information in data link layer, an edge node takes it out and uses it in transport layer for the XCP algorithm. We can see the benefits of cross-layer design in the following chapter.

4.2. The XCP-CL Algorithm

Below is a step-by-step procedure of the XCP-CL algorithm:

1. The core node receives bandwidth requests from all edge nodes.
2. It runs the scheduling algorithm to compute the bandwidth allocation in terms of number of time slots for each edge node.
3. It uses the AAPN signaling protocol to send the bandwidth allocation to each edge node.
4. An edge node (a router running the XCP-CL algorithm) receives the bandwidth allocation, and converts it to data rate in bits per second.
5. It runs the XCP-CL algorithm to compute the allowable increase in data rate for each flow.
6. When the edge node receives an XCP-CL packet from a traffic flow of a sender, it extracts the "Delta_Throughput" from the "Delta_Throughput" field in the XCP-CL header. Then it compares the computed result with the "Delta_Throughput" and updates this value when the computed result is less.
7. This XCP-CL packet passes the whole data path and arrives at the receiver.
8. The receiver copies the "Delta_Throughput" in the "Reverse_Feedback" field and sends it back to the sender via an ACK packet.
9. The sender uses this information and computes the allowable sending rate and the congestion window size.
10. The sender uses the new computed window size to send traffic until the next update.

For example, assume the link is 45Mbps, and the core node allocates 12 slots out of 18 slots in one frame to one edge node. One can obtain an equivalent bandwidth BW =

45Mbps *12 /18 = 30Mbps. Let's assume RTT be 0.020s, packet size 1500 bytes. And there are 10 flows. Then one flow should have a fair sending rate of 3Mbps. If the previous cwnd (Congestion WiNDow size) is 4 packets (sending rate of 2.4Mbps), then Delta_Throughput = (3Mbps – 1500*8*4 /0.020) = 600Kbps (1 packet increment in cwnd). This value will be copied to "Reverse_Feedback" by the receiver then sent back to the sender to adjust its sending rate.

4.3. Simulation Network Modeling

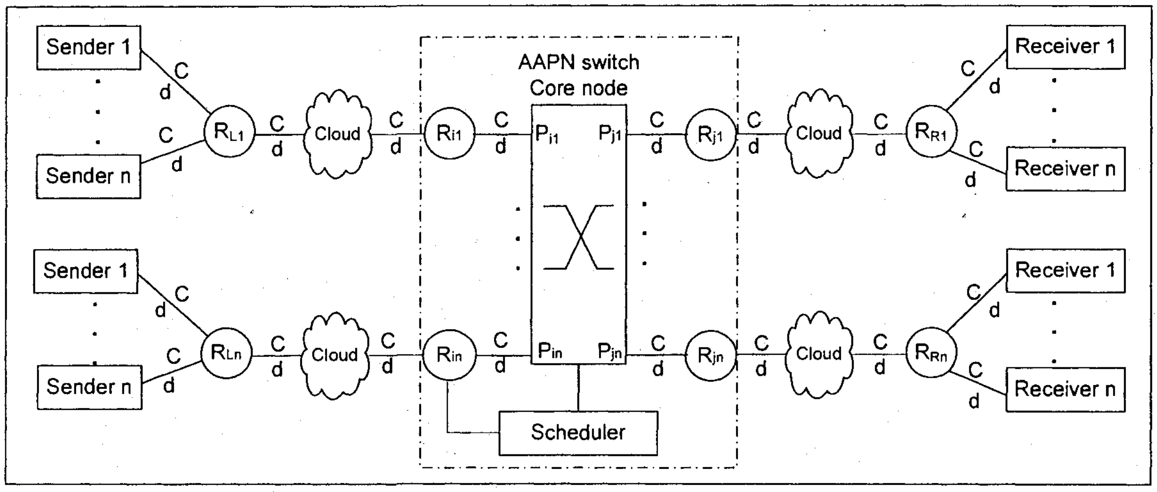

Figure 4.2 AAPN Network Model, One Core Node and M=2n Edge Nodes

Figure 4.2 shows the general network model of an AAPN network application with one core node in our simulation. The dot-lined part is the AAPN network with one core node. As we introduced before, the AAPN optical switch is the core node. There are *2n* edge nodes (routers running AAPN signaling protocol), R_{i1}-R_{in} and R_{j1}-R_{jn}, each of which is connected to the core node via an optic fiber. A core node has *2n* full-duplex ports, P_{i1}-P_{in} and P_{j1}-P_{jn}, each of which is connected to a corresponding edge node. One edge node (R_{in}) co-resides with the core node and is designated as the master node. It controls the core node switch and runs the scheduler. Other edge nodes are slave nodes. Routers R_{L1}-R_{Ln} and R_{R1}-R_{Rn} are ordinary routers. The XCP-CL algorithm runs in these edge nodes and routers so that it can compute congestion feedback to the senders. There are n^2 flows (sender-receiver pair) in the networks and each link has a capacity of *C* and a delay of *d*.

The following tables provide the delay and bandwidth parameters of 3 cases of network we are going to experiment. We set M=2 for simplification (the upper part in Figure 4.2), because with M equals the other even number, the sub-network is the same. So we can evaluate our design in different bandwidth from 45Mbps, 155Mbps to 1Gbps.

Table 4.1 Delay and Bandwidth Parameters of Case 1

Simulation time	**0-200s**	**200-400s**	**400-600s**
Delay of flows 0-19	110ms	170ms	140ms
Delay of flows 20-39	130ms	190ms	210ms
Delay of flows 40-59	150ms	250ms	110ms
Delay of flows 60-79	130ms	290ms	150ms
Delay of flows 80-99	110ms	150ms	130ms
Bandwidth of R_{i1}-R_{j1}	45Mbps	40Mbps	45Mbps

Table 4.1 shows the delays and bandwidth of 100 flows in Case 1. We choose 100 flows because it is a large number commonly used in most literature. All links have a bandwidth of 45Mbps, the standard T3 speed. During the time interval 0-200s, congestion occurs at R_{L1}, due to 100 flows competing for the 45Mbps bandwidth. During 200-400s, the core node re-allocates the bandwidth between R_{i1} and R_{j1} to 40Mbps. Since this bandwidth is less than 45Mbps, then the bottleneck shifts to R_{i1}. During 400-600s, the bandwidth changes back to 45Mbps.

Table 4.2 Delay and Bandwidth Parameters of Case 2

Simulation time	**0-200s**	**200-400s**	**400-600s**
Delay of flows 0-19	110ms	170ms	140ms
Delay of flows 20-39	130ms	190ms	210ms
Delay of flows 40-59	150ms	250ms	110ms
Delay of flows 60-79	130ms	290ms	150ms
Delay of flows 80-99	110ms	150ms	130ms
Bandwidth of R_{i1}-R_{j1}	155Mbps	140Mbps	155Mbps

Table 4.3 Delay and Bandwidth Parameters of Case 3

Simulation time	**0-200s**	**200-400s**	**400-600s**
Delay of flows 0-19	110ms	170ms	140ms
Delay of flows 20-39	130ms	190ms	210ms
Delay of flows 40-59	150ms	250ms	110ms
Delay of flows 60-79	130ms	290ms	150ms
Delay of flows 80-99	110ms	150ms	130ms
Bandwidth of R_{i1}-R_{j1}	1Gbps	980Mbps	1GMbps

Table 4.2 shows the delays and bandwidth parameters in Case 2. In this case, we set the link bandwidth to 155Mbps, the standard OC-3 speed. Table 4.3 shows the delays and bandwidth parameters in Case 3. In this case we set the link bandwidth to 1Gbps which is normal in the current optical networks. In these two cases, the delays are the same in Case 1, we change the bandwidth to evaluate the system with larger bandwidth to show that our algorithm works in high bandwidth cases.

4.4. Implementation Environment

We run our XCP-CL algorithm simulation on PCs with the following configuration:

1. PC Environment #1:

   OS is Window XP SP2.
   CPU is Pentium® 4, 3.2GHz
   RAM is 1.24 GB
   Cygwin Version is 1.5.24-2
   NS2 version is 2.29

2. PC Environment #2:

   CPU is Pentium® 4, 2.2GHz
   RAM is 504 MB
   Cygwin Version is 1.5.24-2
   NS2 version is 2.29

We use two different PC configurations to ensure the simulation result of the algorithm is independent on the simulation environment. Note that a free hard disk must larger than 10GB to run the simulation, especially for the 1Gbps scenario. This is because the trace files are very large. For example, the 1Gbps scenario, the original trace files are about 4GB and the immediate analysis output files are about 2GB.

The simulated time is 600s which is the duration found to provide steady-state value (approximately) for a performance measure. For the fast bandwidth change scenario, simulation time can be shorted to 100s. It is still long enough to observe the behavior of the system. The statistical mean value of a performance measure is obtained from at least three measurements each with a different seed. We have also obtained the 95% confidence intervals of these measures. However, these confidence intervals are usually quite small compared with the mean. For example, see Table 5.5 in Section 5.6. Therefore, we have omitted their presentations in others places for clarity purpose.

4.5. Concluding Remark

In our XCP-CL design and simulation (presented later), we use an out-band control channel for the signaling and assume there is only one core node and a fixed number of edge nodes. However our algorithm can be extended easily to other settings. For example, it can work with any number of edge nodes and signaling can be in-band. In any case, the master node will send the bandwidth allocation information to each edge node. Each edge node will send out traffic according to this allocation. Otherwise the non-committed traffic will be dropped by the core node. For multiple core nodes configuration, each edge node should first decide which core node as a master node to relay the traffic (the routing problem). After the selection, the system works as the same as a one core node system.

In summary, as long as XCP-CL algorithm knows the bandwidth information, the algorithm works (We can see the performance evaluation in later chapter). Our design of the AAPN signaling protocols guarantees that the real time bandwidth information is passed correctly to the routers which run the XCP-CL algorithm.

Chapter 5. XCP-CL Performance Evaluation

We have evaluated our system using a network of N=1 core node and M=6 edge nodes, each with 100 greedy FTP flows as described in previous chapter. The slot period is set as 200us and the frame period is 3.6ms (that is 18 slots). NS2 simulation [ISI07] is used to obtain the time evolution of the performance measures. To simulate the dynamics of network, we change the propagation delay from each sender to each router R_{Ln}, and the delay from each router R_{Rn} to each receiver periodically.

There are different performance measures used in our study. End-to-end delay is defined to be the duration from the time the first bit of a packet is transmitted from the sender until the last bit of the same packet is received at the receiver. RTT (Round Trip Time) consists of the end-to-end delay of a packet and the end-to-end delay of its acknowledgement packet. Link utilization is defined as the output sending rate of a router port (in bps) divided by the maximum sending rate (the rated speed of a link). We measure the link utilization of each port of the router every RTT (use the highest value seen in flow), and measure the congestion window size when an end system receives an ACK. Flow sending rate is defined as window size divided by RTT. It is measured each time when a packet of a flow departs the router. We also define the fair sending rate to be link capacity divided by the number of flows. The queue length in the buffer is measure each time a packet enters the queue or a packet leaves the queue. The congestion window size is defined as the number of packets (as most of the literature did) that can be outstanding at any time without the receiver's acknowledgement (through ACK packets). The sender dynamically adjusts the *cwnd* according to the condition of the network. Basically the size of the congestion window, to a large degree, controls the speed of transmission as transmission pauses until there is an acknowledgment. End to end delay is defined as the elapsed time from the instant that a packet is sent from a sender to the instant that it is received by a receiver. It is measured each time when a packet is received by the receiver.

We first study three cases in our performance evaluations, with a link capacity of

C = 45Mbps, 155Mbps, and 1Gbps, to show our algorithm works in a wide range of link capacity. The propagation delay d of each flow is shown in the table in the previous chapter. In each case study, we also compared the scenarios when router R_{i1} is aware of the change in link capacity of the core node and the scenario that R_{i1} is not. This would allow us to study the benefit of the cross-layer design of our XCP-CL algorithm. Then we investigate the impact of different queue buffer size on queue length, queuing delay, dropped packet and link utilization.

We evaluate two scenarios under each case. The scenario of using the original XCP algorithm will be used as a reference for comparison with the scenario of using the XCP-CL algorithm. For clearer presentation, we only draw 5 flows (#1, #21, #41, #61, #81) from each group of different RTTs. They are representative of the performance of the system.

5.1. Case 1: Link Bandwidth of 45Mbps

In this case, the rated bandwidth of the connection between R_{i1}-R_{j1} is 45Mbps and the number of XCP flows is 100. More details of the network model can be found in the previous chapter. In particular, Table 4.1 (see previous chapter) provides the changes in delay of different flows and link bandwidth during the 600s of simulation time. Congestion occurs at R_{i1} due to the core node scheduling.

The fair sending rate of each XCP flow is 0.45Mbps (45Mbps/100 which is equivalent to 56.25 kilobyte/second) during 0-200s and 400-600s. The link bandwidth is reduced to 40Mbps after bandwidth re-allocation by the core switch at time 200s. During 200-400s, the fair sending rate of each XCP flow should be 0.4Mbps (40Mbps/100 which is equivalent to 50 kilobyte/second) in order to avoid congestion at R_{i1} if XCP algorithm knows such bandwidth change.

We use a buffer size of one at the edge node, because we plan to implement the edge node in an FPGA which has limited memory (much less than in a PC). Buffer size in other routers are set to the product (bandwidth * propagation delay * 2) as described in some literatures. Our performance evaluation later supports this choice of buffer size and the feasibility of implementing XCP-CL in FPGA.

5.1.1 Scenario 1: Using Original XCP Algorithm

Under this scenario, the edge node R_{i1} has no knowledge of the change in link capacity of the core node when congestion occurs. It has no way to inform the sender to adjust its congestion window size (sending rate) correctly.

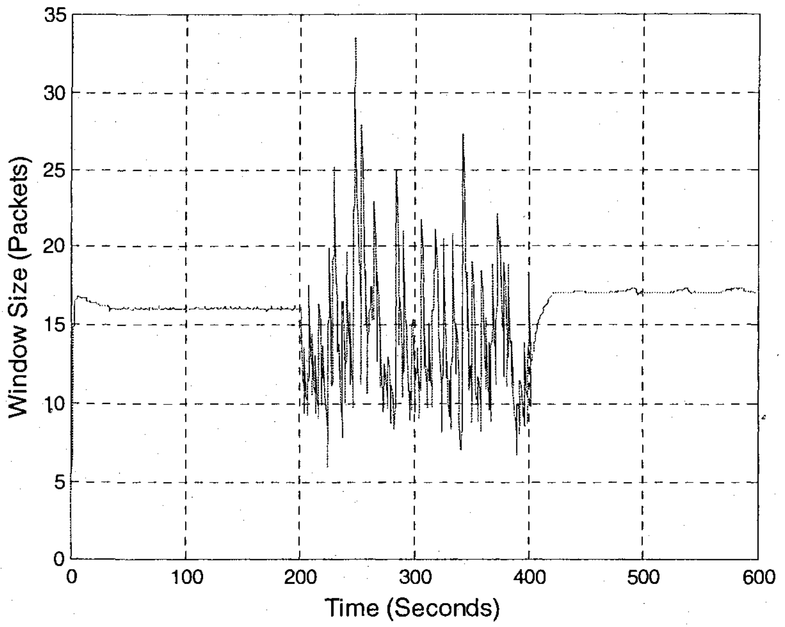

Figure 5.1 Window Size of Flow#1 in Scenario 1, Case 1

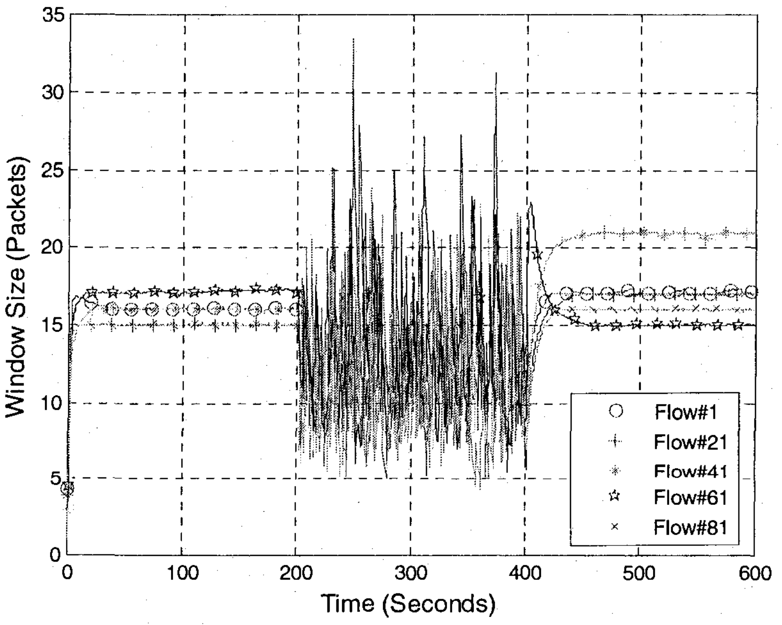

Figure 5.2 Window Sizes of 5 Selected Flows in Scenario 1, Case 1

Window Size

Figure 5.1 shows the time evolution of the window size of Flow#1. Between time 0s and 200s, this flow can quickly stabilize to a window size of 16 packets. There is a small peak at time t=8s probably due to the system adjustment. However, it cannot respond to the congestion and the dynamic changes of the network configuration between 200s and 400s, as it fluctuates heavily (the maximum is 34 packets and the minimum is 6 packets) all the time and can not attain a stable window size. After the bottleneck is removed, one can see that the congestion dissipates beyond t=400s and the window size quickly converges to 17 packets in about 22s.

From Figure 5.2, one can see that other flows have a similar behavior and therefore follow a similar pattern in reaction to the congestion as we expected. Each flow has attained a different window size. This is because the RTT values are different. One can see the fairness in the sending rate performance in the following.

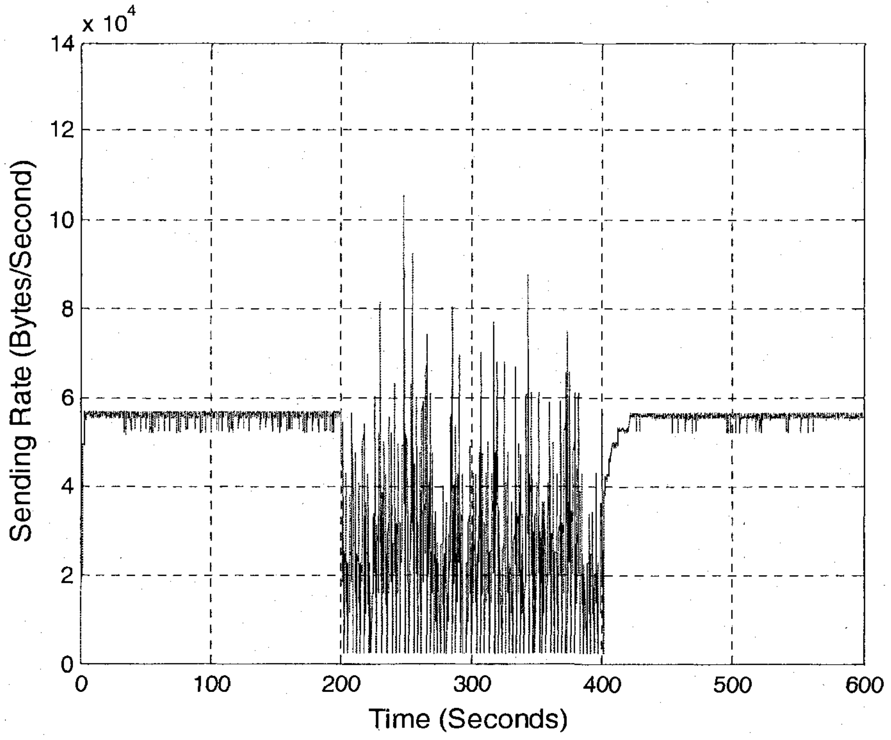

Figure 5.3 Sending Rate of Flow#1 in Scenario 1, Case 1

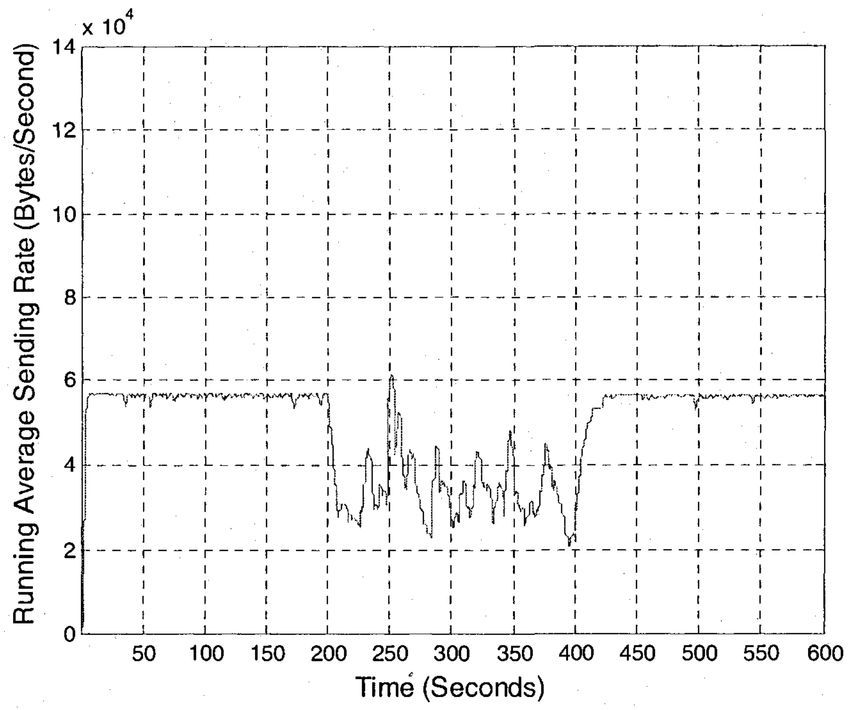

Figure 5.4 Running Average of Sending Rates of Flow #1 in Scenario 1, Case 1

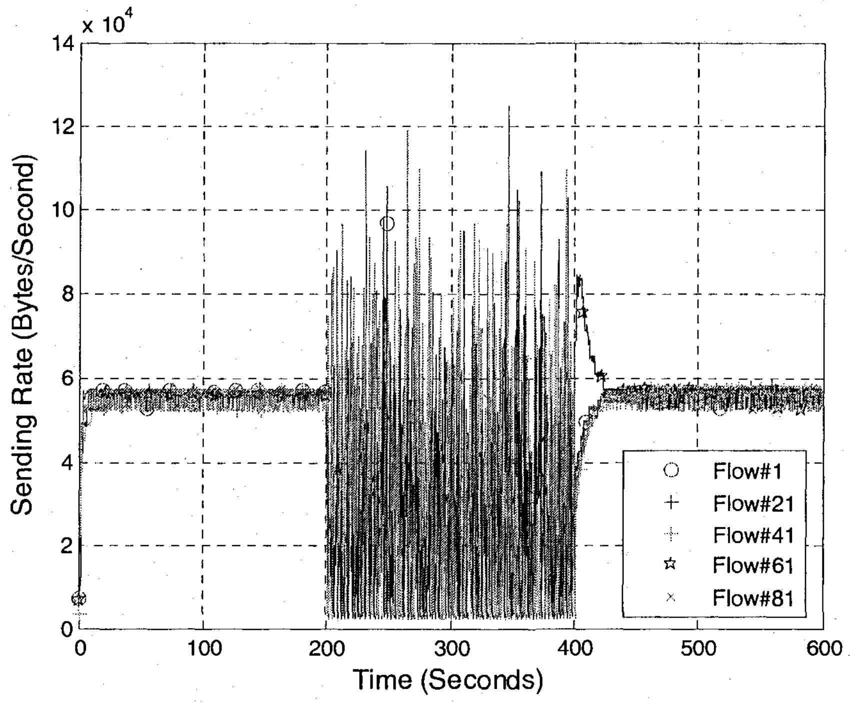

Figure 5.5 Sending Rates of 5 Flows in Scenario 1, Case 1

Sending Rate

Figure 5.3 depicts the sending rate of Flow#1 and Figure 5.4 shows the running average.

During the interval of 0-200s, this flow achieves its fair rate after 3s with a stable mean sending rate of 56 kilobyte/second. There is only a very small fluctuation of about 1 kilobyte/second. During the congestion from 200s to 400s, the rate fluctuates heavily between its maximum of 105 kilobyte/second and its minimum of 2 kilobyte/second. After the congestion disappears beyond 400s, the flow rate goes back to its fair share after a few seconds. Other flows follow the same pattern as show in Figure 5.5.

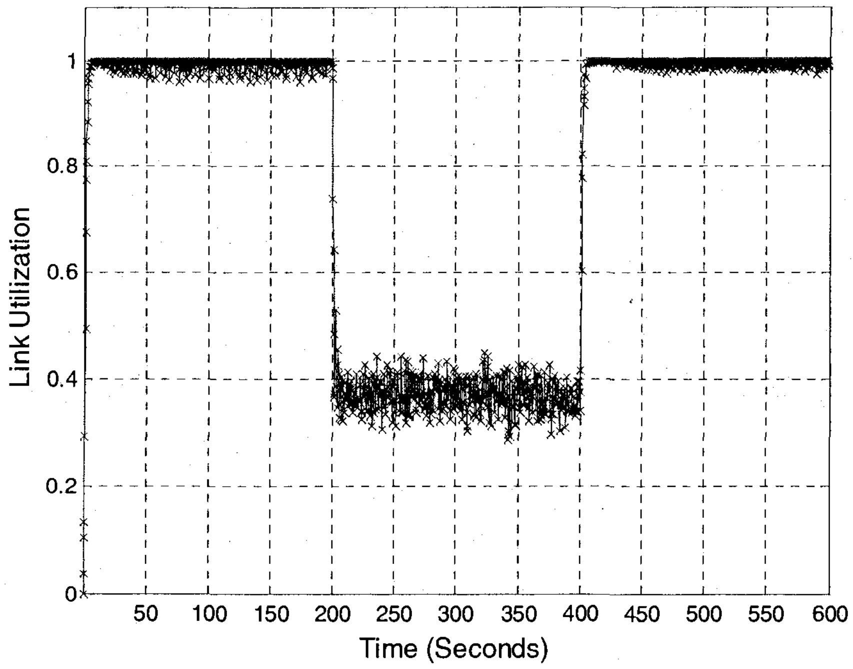

Figure 5.6 Link Utilization in Scenario 1, Case 1

Link Utilization

Figure 5.6 depicts the output link utilization of the router (in Figure 4.2, there is only one output and one input links). One can see that full utilization is attained very quickly within 3s and has a very small fluctuation throughout the period of 0-200s. During the congestion interval from time 200s to 400s, the utilization goes down to 0.4 abruptly and stays there (with only a small fluctuation) due to the congestion and packet drops. This is because R_{i1} does not know the link capacity has been changed from 45Mbps to 40Mbps, and therefore does not send back correct information to its sender. Since the sender does not have the correct information and tries to send at a higher rate than the allowable one.

Thus congestion occurs in router R_{i1} and some of the packets are dropped. The senders can detect this information and half the congestion window size. The senders may detect the packet-drop information and would have halved the congestion window size accordingly. Unfortunately, XCP protocol is aggressive [KaHa02] to use all the spare bandwidth indicated in the ACK no matter what congestion can occur in the router.

After the bottleneck disappears beyond 400s, XCP has the correct congestion information and the utilization goes back to 1 quickly again.

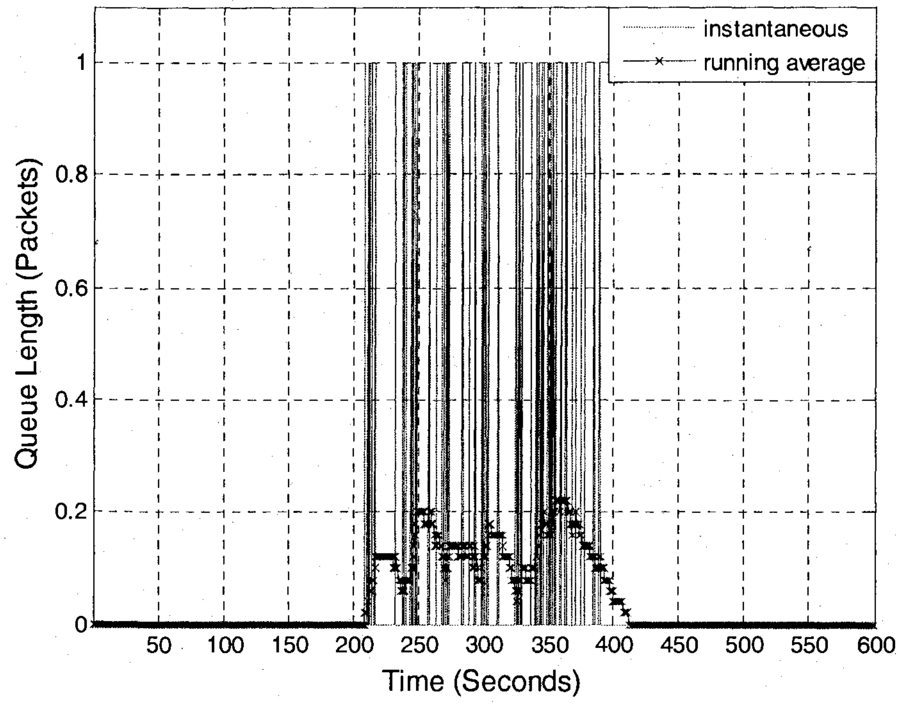

Figure 5.7 Queue Length in Scenario 1, Case 1

Queue Length

Figure 5.7 shows the time evolution of queue length of the router buffer serving all 100 flows. As one can see, when XCP knows the congestion information (during time 0-200s and 400-600s), XCP works well and there is no queue in the router. However, when there is congestion and XCP does not know the exact congestion information during time 200-400s, there is a queue in the router which oscillating between 0 and 1 (the maximum buffer size) with a running average of about 0.1 packet. The running average has less fluctuation as expected and its fluctuations correspond to the instantaneous influx of random arrivals as shown.

5.1.2 Scenario 2: Using XCP-CL, the Modified XCP Algorithm

Under this scenario, the edge node R_{i1} is aware of the change in link capacity of the core node and therefore the congestion information. R_{i1} advertises the correct congestion information to all senders in order to adjust their congestion window size (and sending rate).

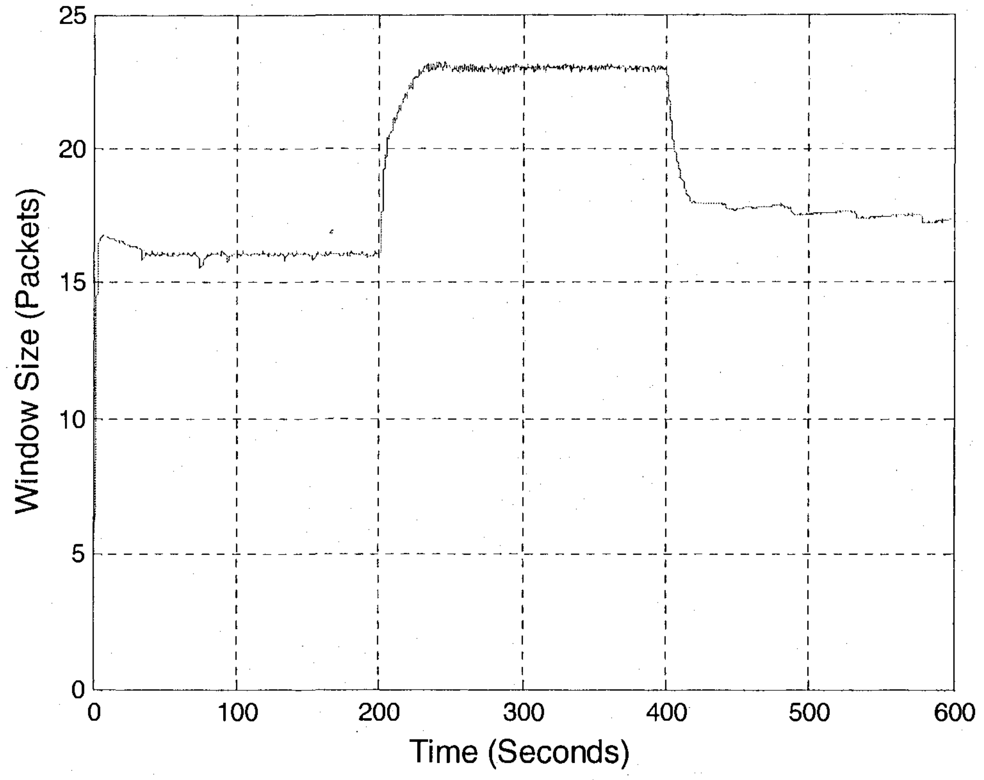

Figure 5.8 Window Size of Flow#1 in Scenario 2, Case 1

Window Size

Figure 5.8 shows the time evolution of the window size of Flow#1. Between time 0s and 200s, this flow can quickly stabilize to a window size of 16 packets with a little peak at time t=8s due to the system adjustment. Between time 200s and 400s, this flow can attain a stable window size of 23 packets in about 30s in response to the congestion and the dynamic change of the network configuration. When the congestion disappears beyond t=400s, the window size also quickly converges to a window size of 17 packets after about 20s.

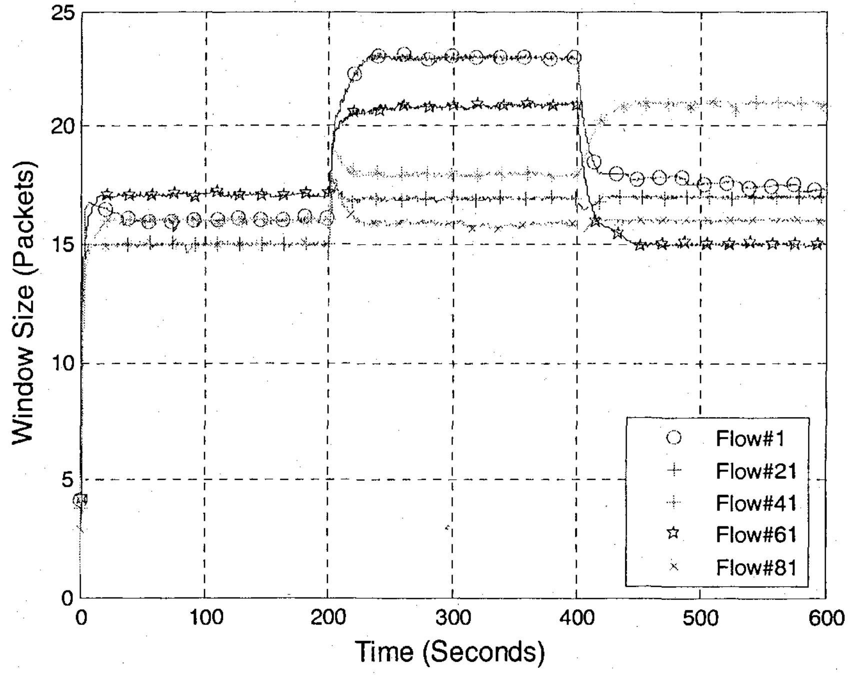

Figure 5.9 Window Sizes of 5 Flows in Scenario 2, Case 1

From Figure 5.9 one can see that other flows have a similar behavior and therefore follow a similar pattern in reaction to the congestion and the dynamic configuration changes. Again each flow would attain a different window size because the RTT values are different.

<u>Sending Rate</u>

Figure 5.10 depicts the sending rate of Flow#1 and Figure 5.11 shows the running average. During 0-400s, Flow#1 achieves its fair rate of 56 kilobyte/second after 5s which reduces to 50 kilobyte/second during congestion and back up to 56 kilobyte/second after the congestion. A dip is noticed at t=200s (with a sending rate of 37 kilobyte/second) and a peak at t=400s (with a sending rate of 68 kilobyte/second). This is due to the effect of the dynamic changes of the network configuration. Because at these points, there is some approximation to the RTT measurement which is required to compute the sending rate (remember it equals congestion window size divided by RTT).

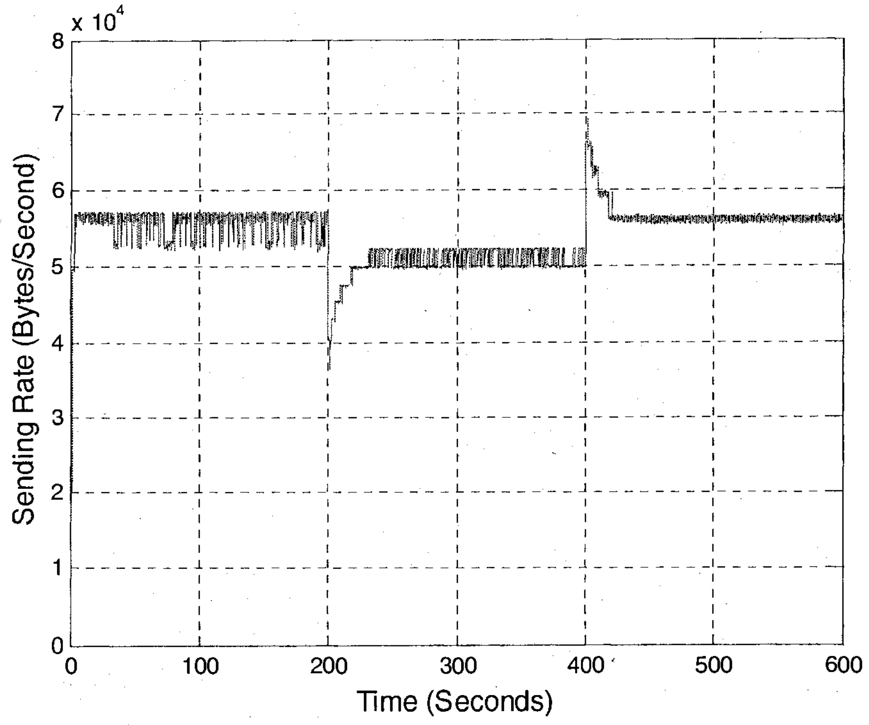

Figure 5.10 Sending Rate of Flow#1 in Scenario 2, Case 1

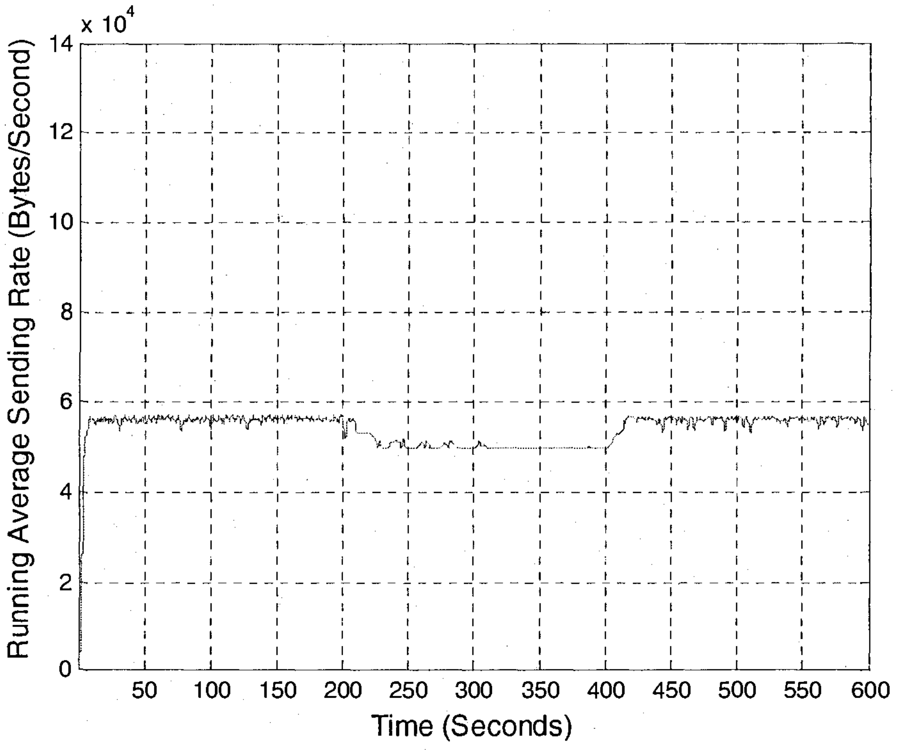

Figure 5.11 Running Average Sending Rate of Flow#1 in Scenario 2, Case 1

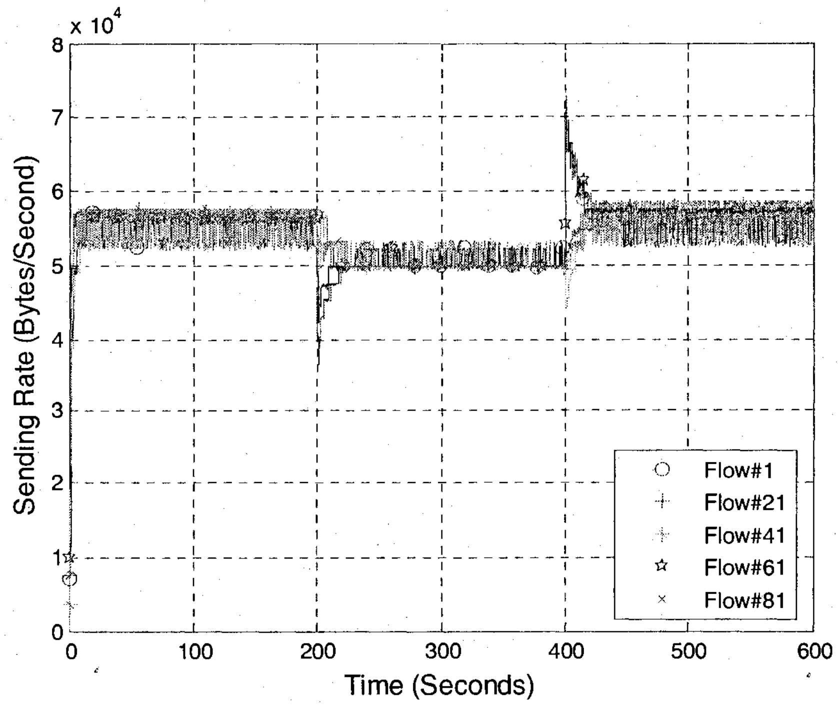

Figure 5.12 Sending Rates of 5 Flows in Scenario 2, Case 1

The sending rate has a small fluctuation of 3 kilobyte/ second but only 2 kilobyte/second during the congestion. In addition to the argument on the approximate measurement of the RTT in the router, this may probably be due to the fact that the router also shuffles the throughput of each flow to achieve fairness among all the flows. Figure 5.12 depicts the sending rates of 5 flows. As one can see these flows have a similar behavior and follow the same pattern except they have an undershoot or overshoot at the instant of time 200s or 400s. This is due to the dynamic configuration changes.

Link Utilization

Figure 5.13 depicts the link utilization of the router. One can see that full utilization is attained very quickly after 5s. Then it stays there all the time even during the congestion interval. The two dips at time 200s (with utilization of 0.75) and 400s (with utilization of 0.97) are due to the dynamic change of the network configuration.

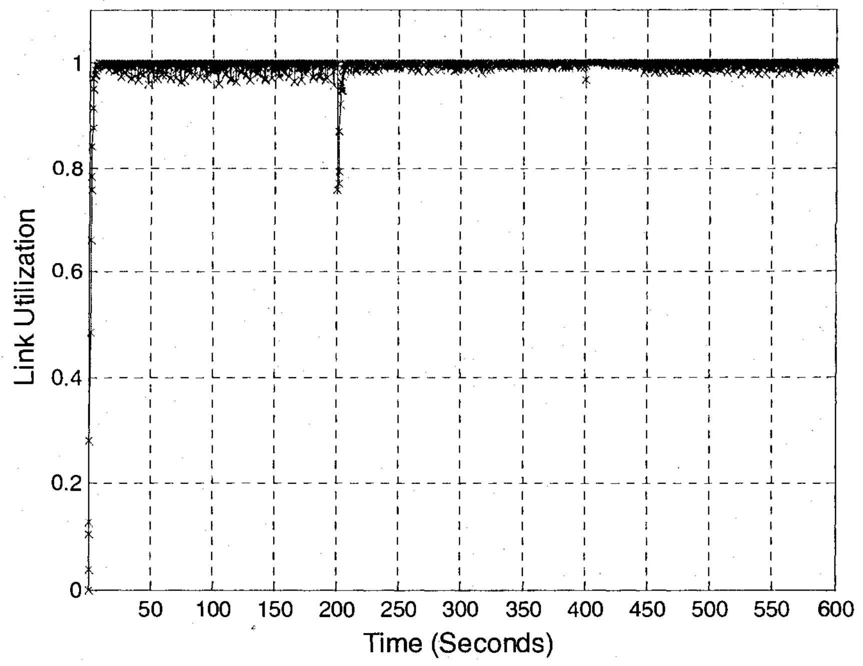

Figure 5.13 Link Utilization in Scenario 2, Case 1

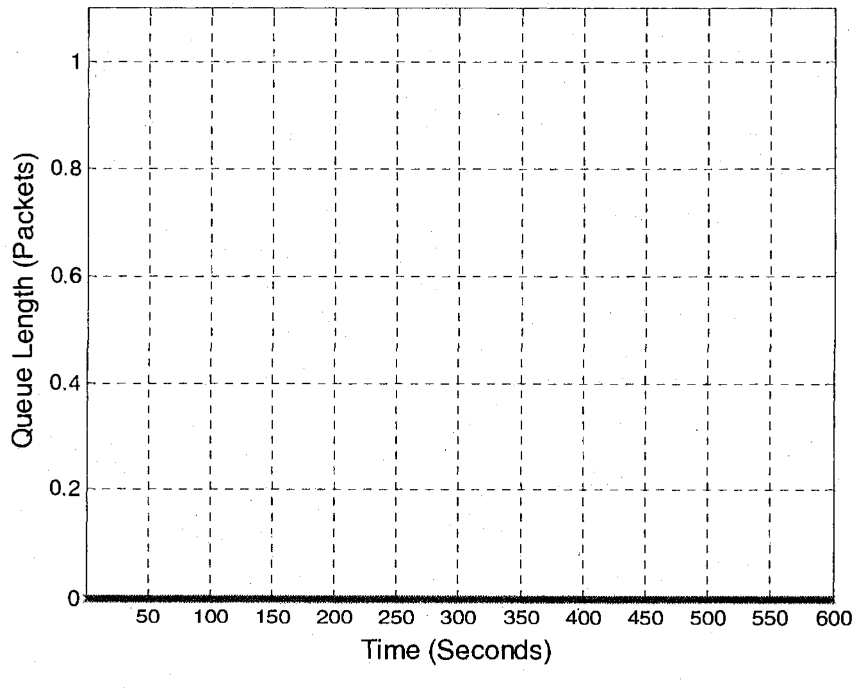

Figure 5.14 Queue Length in Scenario 1, Case 2

Queue Length

As we can see in Figure 5.14, there is no queue (so no queuing delay) in the router. This is because XCP-CL can obtain the correct congestion information all the time. Because of this nice property of our algorithm, we will no longer present the queue length performance of each simulation result in the follow sections.

5.1.3 Comparison

Comparing Scenario 1 with Scenario 2, we can see that when the system has no congestion both XCP and XCP-CL work very well: full utilization is achieved very quickly; window size converges to steady state in a short time while sending rate converges to the fair share. However, when the system encounters congestion, there is a big difference in performance between theses two algorithms. The utilization of XCP goes down to 0.4 abruptly while XCP-CL stays at 1 all the time (except for a dip at the beginning of the duration due to the dynamic network configuration). The window size and the sending rate of XCP oscillate heavily but XCP-CL works well (with only dips or overshoots at the dynamic network configuration instants). XCP has a queue in the router during the congestion while XCP-CL has no queue all the time.

The reason for this big difference is that XCP can not obtain the actual bandwidth information and feedbacks a wrong value to its senders. On the other hand XCP-CL can obtain the correct bandwidth information. So it performs well.

5.2. Case 2: Link Bandwidth of 155Mbps

Table 5.1 Delay and Bandwidth Parameters of Case 2

Simulation time	**0-200s**	**200-400s**	**400-600s**
Delay of flows 0-19	110ms	170ms	140ms
Delay of flows 20-39	130ms	190ms	210ms
Delay of flows 40-59	150ms	250ms	110ms
Delay of flows 60-79	130ms	290ms	150ms
Delay of flows 80-99	110ms	150ms	130ms
Bandwidth of R_{i1}-R_{j1}	155Mbps	140Mbps	155Mbps

In this case, the rated bandwidth (the maximum sending speed) of R_{i1}-R_{j1} connection is 155Mbps and again the number of XCP flows is 100. As before, Table 5.1 (repeated for clarity) provides the change in delay of different flows during the 600s simulation time. The link bandwidth change is also provided in the bottom row. Since sending rate is redundant information with respect to congestion window, and queue length performance has a similar behavior, we shall only show link utilization and congestion window size performance.

5.2.1 Scenario 1: Using the Original XCP Algorithm

Under this scenario, the edge node R_{i1} has no knowledge of the change in link capacity of the core node when congestion occurs. It has no way to inform the senders to adjust their congestion window size correctly.

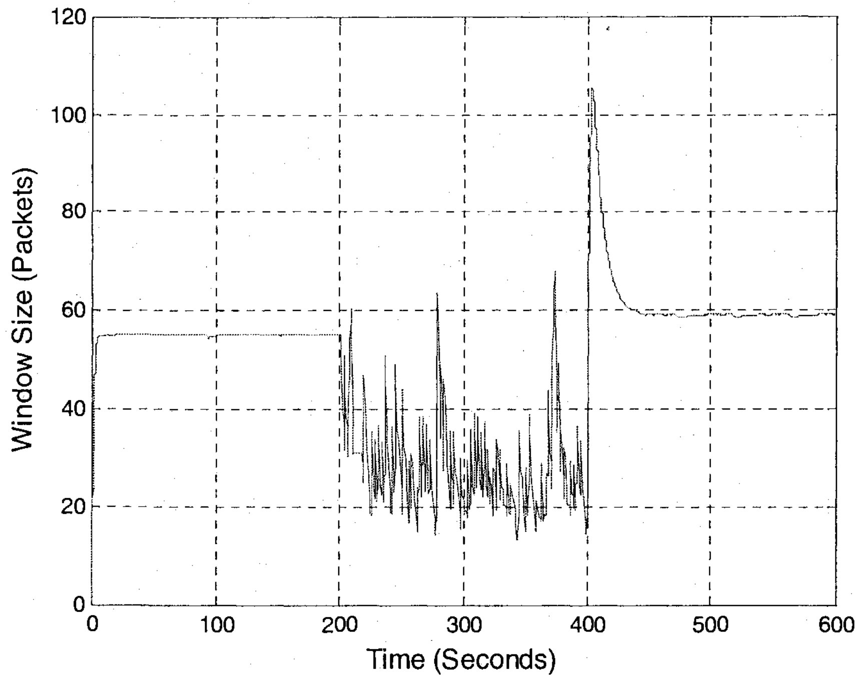

Figure 5.15 Window Size of Flow#1 in Scenario 1 Case 2

Window Size

Figure 5.15 shows the window size performance of Flow#1. During the interval of 0-200s, Flow#1 achieves a stable window size of 55 packets quickly after 3s. But it can not respond properly to the congestion during time 200-400s, as the window size fluctuates heavily (the maximum is 68 packets and the minimum is 13 packets) all the time and can not attain a stable window size. After the bottleneck disappears, the window size quickly converges to 59 packets after 26s with a peak at 400s. This is due to the system adjustment to the dynamic network configuration.

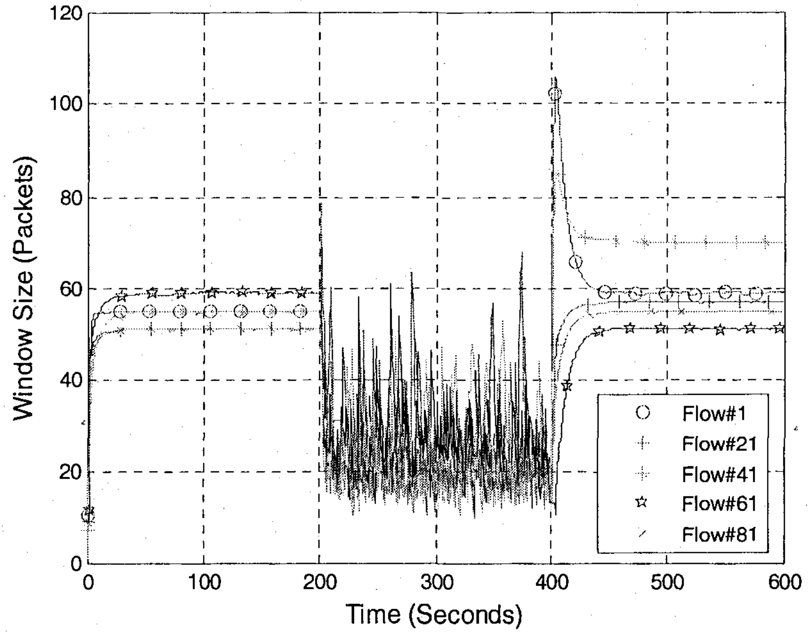

Figure 5.16 Window Sizes of 5 Flows in Scenario 1 Case 2

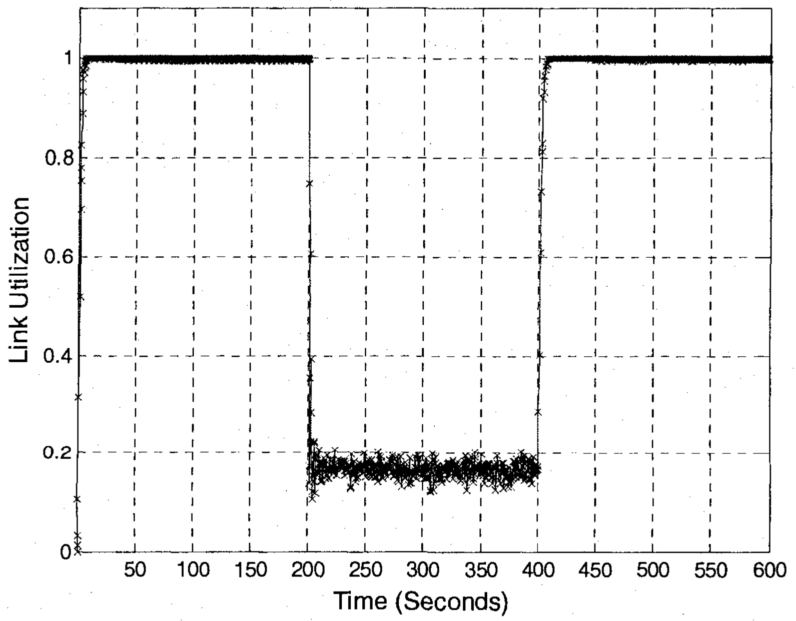

Figure 5.17 Link Utilization in Scenario 1 Case 2

Figure 5.16 shows the time evolution of the window sizes of 5 flows. They have a similar behavior and therefore follow a similar pattern in reaction to the congestion which each has a different window size due to the different RTT values.

Link Utilization

Figure 5.17 depicts the link utilization of the router. One can see that full utilization is attained very quickly within 3s. During congestion interval of 200-400s, the utilization goes down to 0.14 abruptly and stays at 0.16 (with a small fluctuation) due to congestion. After the bottleneck disappears beyond t=400s, XCP has the correct congestion information and the utilization goes back to 1 quickly again.

5.2.2 Scenario 2: Using XCP-CL, the Modified XCP Algorithm

Under this scenario, the edge node R_{i1} knows the change in the link capacity of the core node and sends congestion information correctly back to the senders.

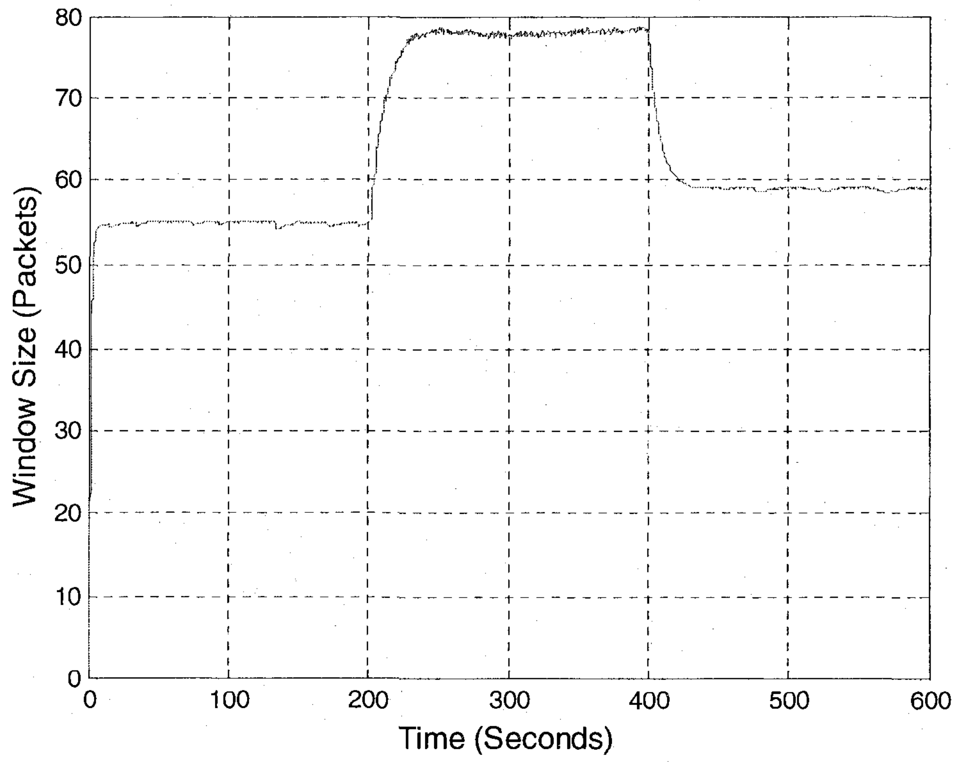

Figure 5.18 Window Size of Flow#1 in Scenario 2 Case 2

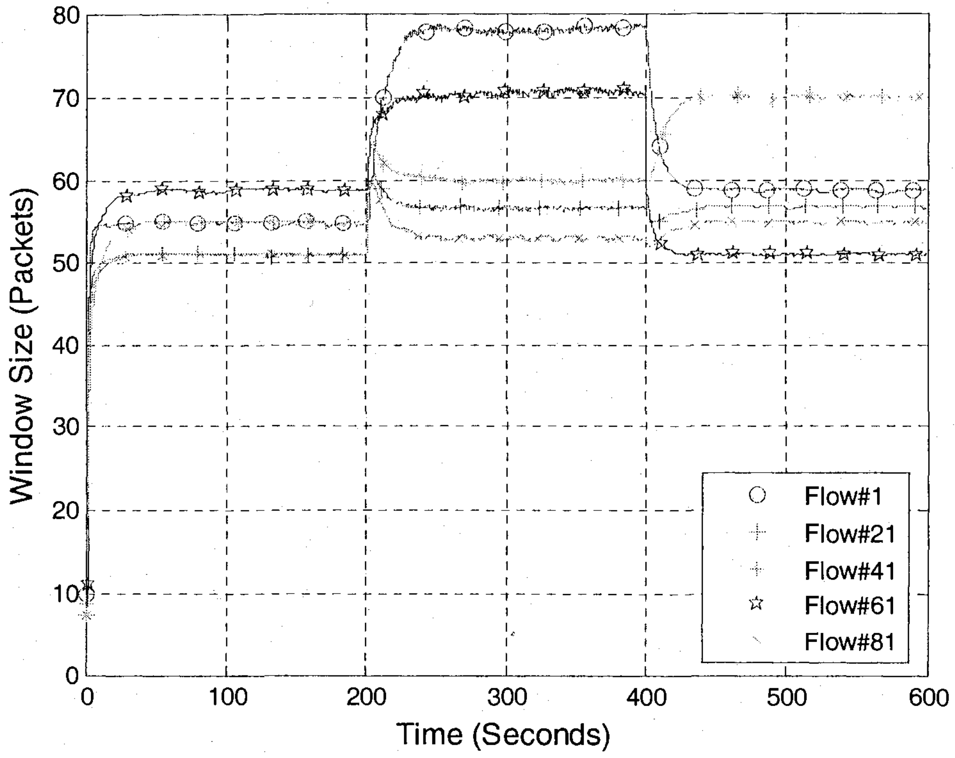

Figure 5.19 Window Sizes of 5 Flows in Scenario 2 Case 2

Window Size

Figure 5.18 depicts the window size performance of Flow#1 under XCP-CL algorithm. It attains a window size of 55 packets after 3s between 0s and 200s. Between 200s and 400s, this flow can get a stable window size of 78 packets in about 15s in response to the congestion and the dynamic change of the network configuration. When the congestion disappears beyond t=400s, the window size quickly converges to 59 packets in about 10s. Other flows have a similar behavior and therefore follow a similar pattern as one can see from Figure 5.19.

Link Utilization

Figure 5.20 depicts the link utilization of the router. Full utilization is attained very quickly after 2.5s. Then it stays there all the time even during the congestion interval. The two dips at time 200s (with utilization of 0.67) and 400s (with utilization of 0.96) are because of the dynamic change of the network configuration. It is the same as before.

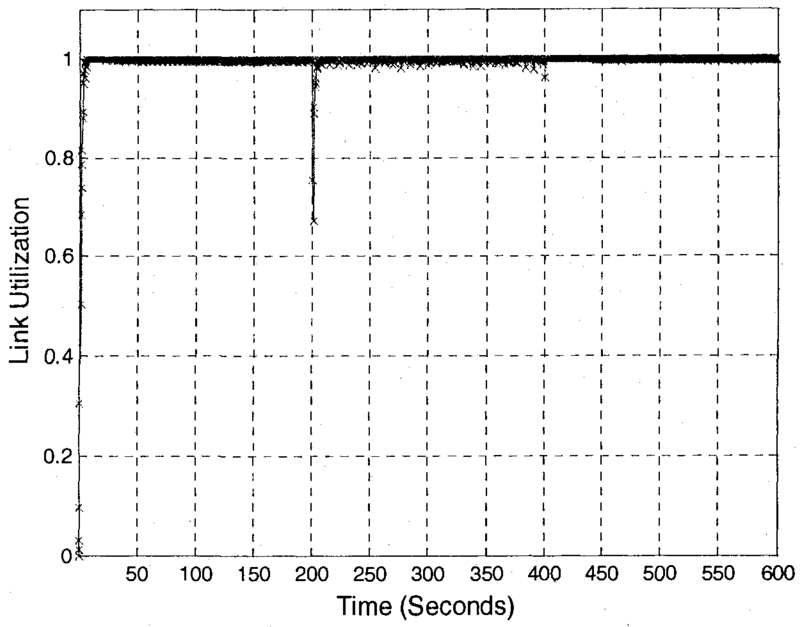

Figure 5.20 Link Utilization in Scenario 2 Case 2

5.3. Case 3: Link Bandwidth of 1Gbps

In this case, the topology and number of flows are the same as in Case 1 but now the rated bandwidth of R_{i1}-R_{j1} connection is 1Gbps (the normal speed of photonic networks).

Table 5.2 (repeat for clarity) provides the change in delay of different flows during the 600s simulation time. The link bandwidth change also provided in the bottom row. A bottleneck is created at R_{i1}-R_{j1} during 200-400s when the link bandwidth changed to 980Mbps due to scheduling.

Table 5.2 Delay and Bandwidth Parameters of Case 3

Simulation time	0-200s	200-400s	400-600s
Delay of flows 0-19	110ms	170ms	140ms
Delay of flows 20-39	130ms	190ms	210ms
Delay of flows 40-59	150ms	250ms	110ms
Delay of flows 60-79	130ms	290ms	150ms
Delay of flows 80-99	110ms	150ms	130ms

Bandwidth of R_{i1}-R_{j1}	1Gbps	980Mbps	1GMbps

5.3.1 Scenario 1: Using Original XCP Algorithm

Under this scenario, the edge node R_{i1} has no knowledge of the change in link capacity of the core node when congestion occurs. It has no way to inform the senders to adjust their congestion window size correctly.

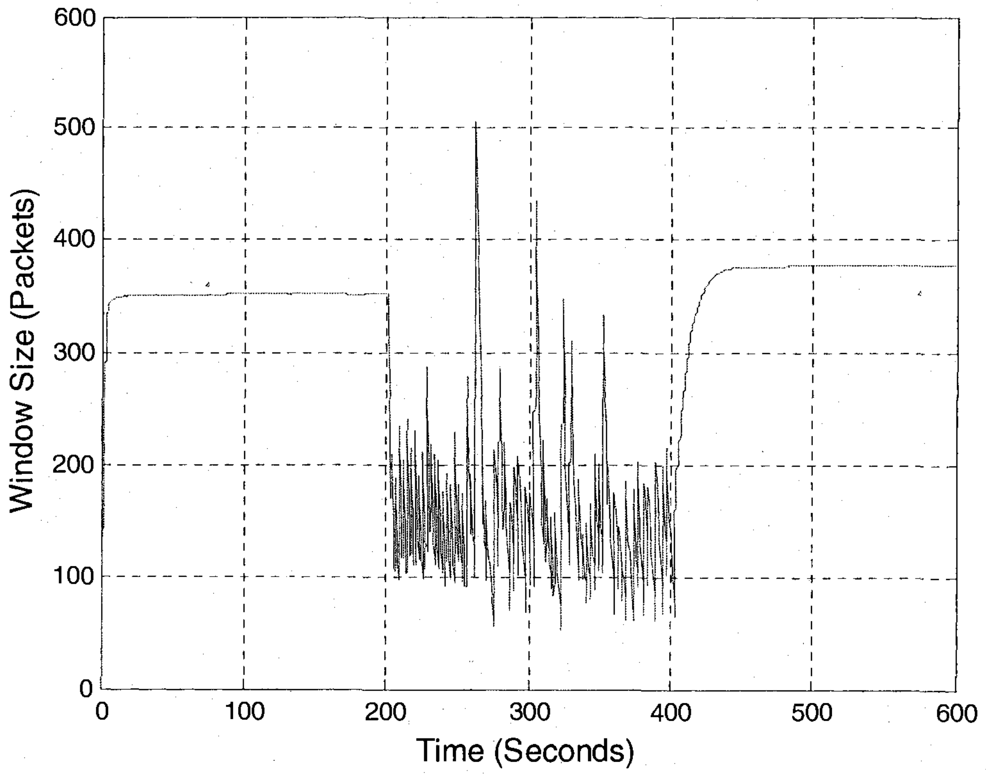

Figure 5.21 Window Size of Flow#1 in Scenario 1 Case 3

Window Size

Figure 5.21 shows the time evolution of window size performance of Flow#1. During the interval of 0-200s, this flow achieves a stable window size of 350 packets quickly after 3s. But it can not respond to the congestion properly during time 200-400s, as the window size fluctuates heavily (the maximum is 504 packets and the minimum is 55 packets) all the time and can not attain a stable window size. After the bottleneck disappears beyond t=400s, the window size quickly converges to 376 packets after 26s. Again other flows have a similar behavior and therefore follow a similar pattern in reaction to the congestion as one can see from Figure 5.22.

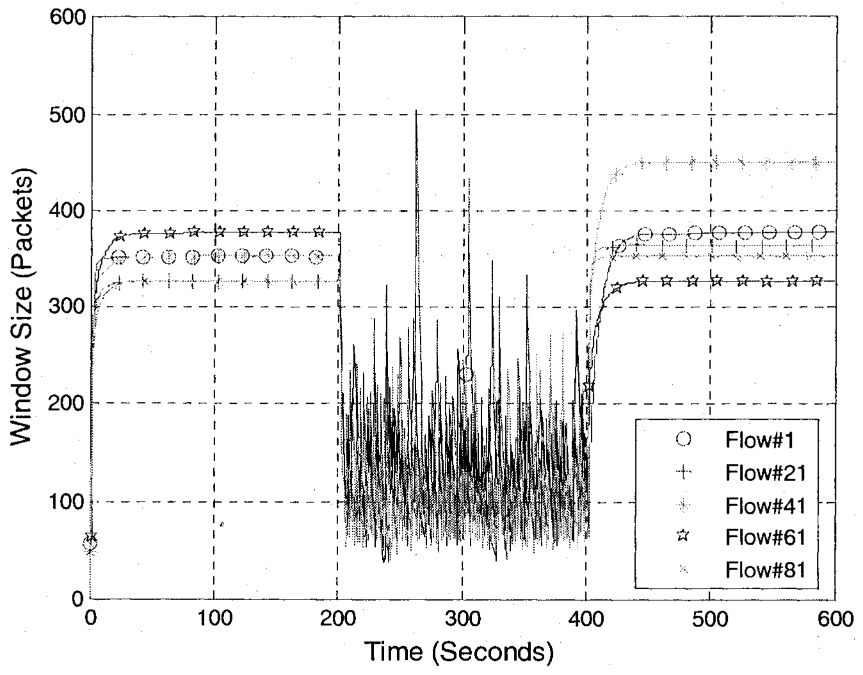

Figure 5.22 Window Sizes of 5 Flows in Scenario 1 Case 3

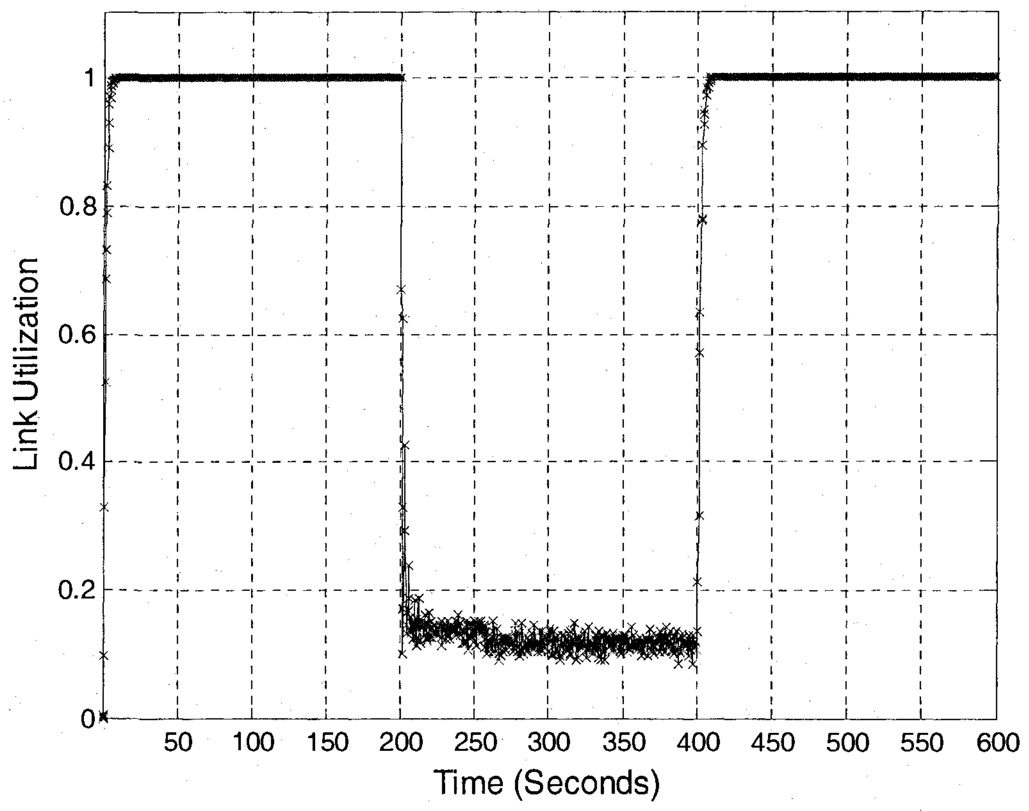

Figure 5.23 Link Utilization in Scenario 1 Case 3

Link Utilization

Figure 5.23 depicts the link utilization of the router. One can see that full utilization is attained very quickly within 3s. During time 200-400s when there is congestion, the utilization goes down to 0.14 abruptly and stays at 0.12 (with a small fluctuation) due to congestion. After the bottleneck disappears beyond t=400s, XCP has the correct congestion information and the utilization goes back to 1 quickly again.

5.3.2 Scenario 2: Using XCP-CL, the Modified XCP Algorithm

Under this scenario, the edge node R_{i1} knows the change in the link capacity of the core node and sends congestion information correctly back to the senders.

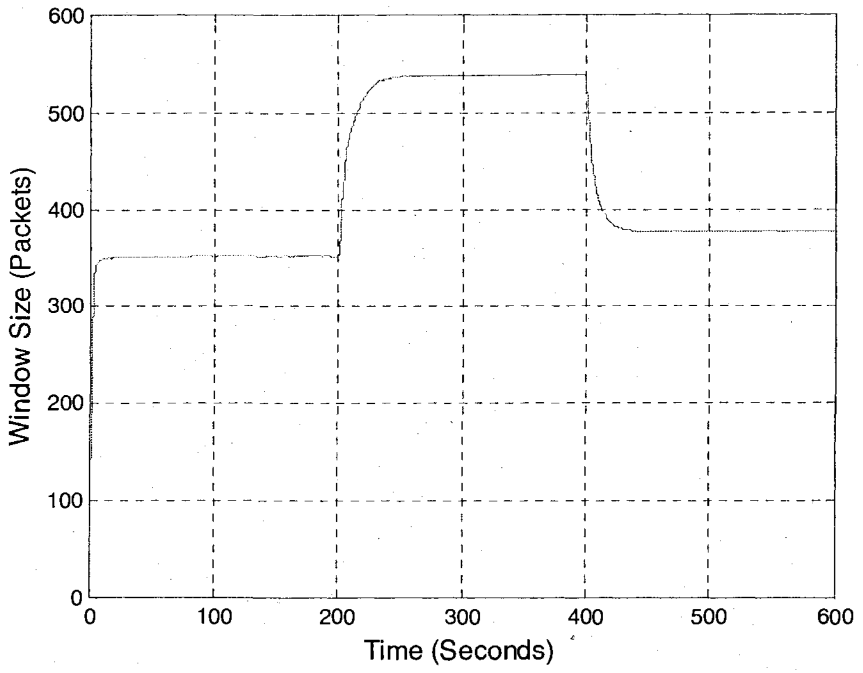

Figure 5.24 Window Size of Flow#1 in Scenario 2 Case 3

Window Size

Figure 5.24 depicts the time evolution of the window size performance of Flow#1. As one can see this flow attains a window size of 350 packets after 35s between time 0s and 200s. Between time 200s and 400s, this flow can get a stable window size of 538 packets in about 8s in response to the congestion and the dynamic change of the network configuration. When the congestion disappears beyond t=400s, the window size quickly converges to 376 packets in about 10s. Other flows have a similar behavior and therefore follow a similar pattern in reaction to the congestion as one can see from Figure 5.25.

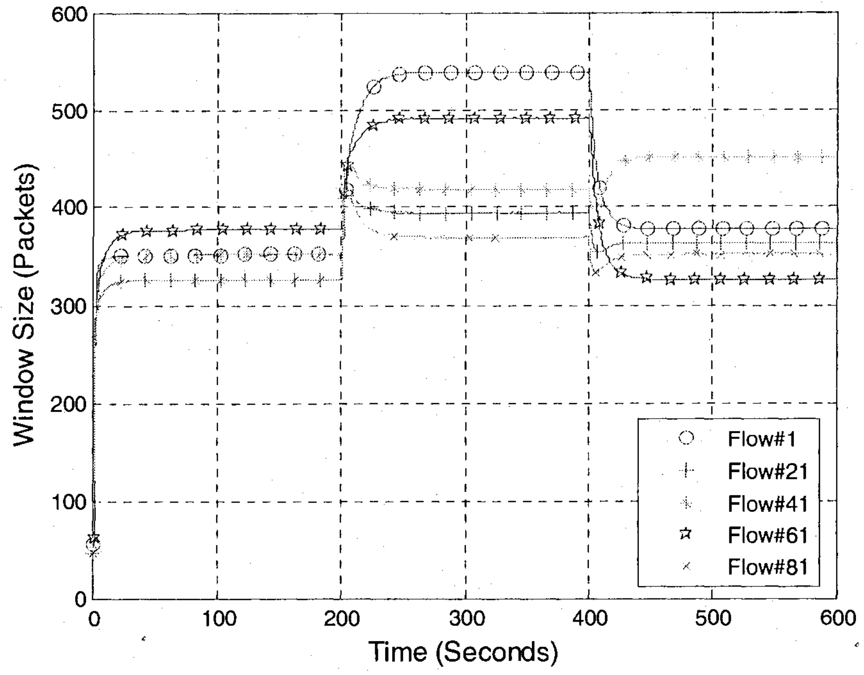

Figure 5.25 Window Sizes of 5 Flows in Scenario 2 Case 3

<u>Link Utilization</u>

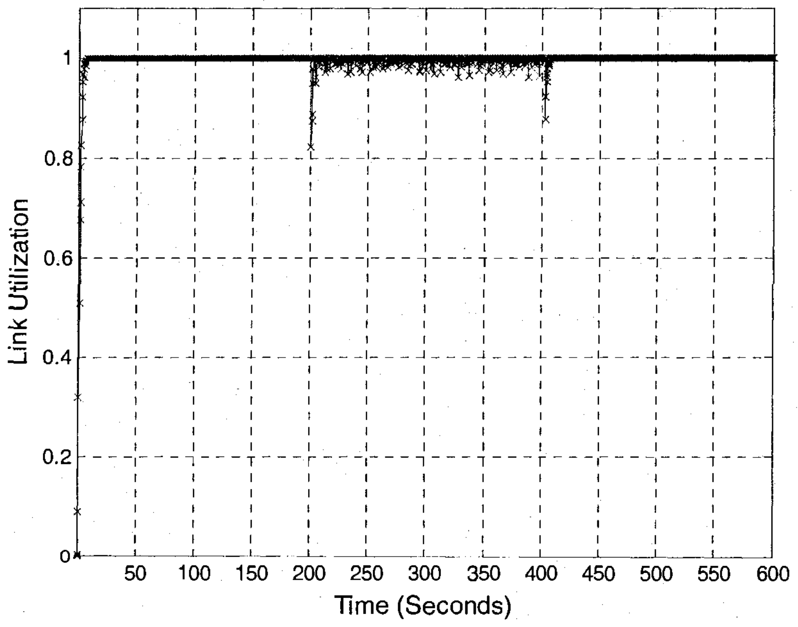

Figure 5.26 Link Utilization in Scenario 2 Case 3

Figure 5.26 depicts the link utilization of the router. Full utilization is attained very quickly after 3s. Then it stays there all the time even in the congestion interval. The two dips at time 200s (with utilization of 0.8) and 400s (with utilization of 0.9) are because of the dynamic changes of the network configuration.

5.4. Performance Comparison of XCP and XCP-CL Algorithms

Table 5.3 Performance Comparison of XCP and XCP-CL During Congestion

Speed	**45Mbps**		**155Mbps**		**1Gbps**	
Algorithm	XCP	XCP-CL	XCP	XCP-CL	XCP	XCP-CL
cwnd	Bad	Good	Bad	Good	Bad	Good
Fairness	Bad	Good	Bad	Good	Bad	Good
Queue Length	Non-Zero	Zero	Non-Zero	Zero	Non-Zero	Zero
Utilization	Bad	Good	Bad	Good	Bad	Good

Notes: 1. *cwnd* is the congestion window size

2. Good and Bad: the relative performance during the congestion

Compare the three cases of different data rates, when there is no congestion at the edge node R_{i1}, both algorithms work very well, with a good link utilization, a fair sending rate, and a zero queue length. However, when there is congestion, there is a big difference between these two algorithms as shown in Table 5.3. XCP-CL still works very well in any case as before, but for XCP during the congestion interval, each flow can no longer achieve a stable window size, nor a fair sending rate. There is always an oscillating queue in the router. The link utilization is also not good. When the bandwidth is 45Mbps, the link utilization goes down sharply to 0.4, while in 155Mbps case, it is 0.16. When the bandwidth is 1Gbps, the link utilization goes down to 0.1. It is worse than the former two cases. The reason for this big difference is that XCP can not obtain the actual bandwidth information and feedbacks a wrong value to its senders. On the other hand XCP-CL can get the correct bandwidth information. So it performs well.

5.5. Impact of Buffer Size on Queue Length, Packet Drops, Link Utilization

We use the simulation model in Figure 4.2 and the parameters in Table 4.1 (Case 1 study, 100 flows with a link bandwidth of 45Mbps) to investigate the impact of different queue buffer sizes on the performance of queue length, packet drop, link utilization. With other parameters, the system behavior is similar. We set the buffer size to 1, 10, 100 and 1000 packets and measure the queue length, the queuing delay, the number of dropped packets, and the link utilization of our XCP-CL algorithm and summarize them in the following section. Again the performance of our algorithm is much better than the original one. The results of the original XCP algorithm can be found in Appendix D.

Figure 5.27 to Figure 5.29 show the performance of queue length, queuing delay, number of dropped packets and link utilization performance when the buffer size is set to 1, 10, 100, or 1000 packets. Queue length, queuing delay, and number of dropped packets are all zero all the time. Link utilization is 1 all the time (with dips during the network dynamic configuration instants) with any buffer sizes. Our algorithm works well even there is congestion and the bandwidth changes with time.

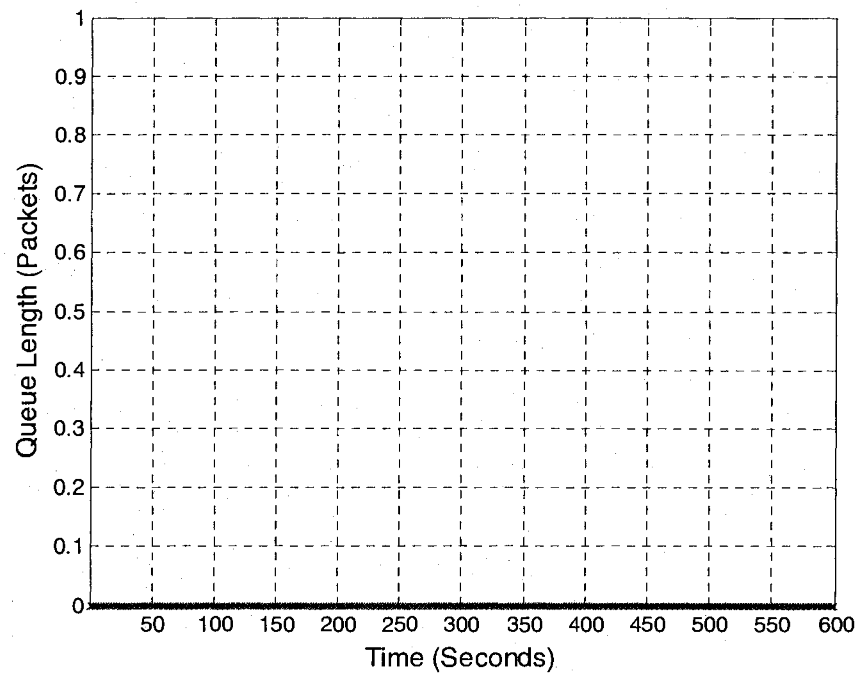

Figure 5.27 Queue Length (Buffer Size= 1, 10, 100, 1000 Packets, XCP-CL)

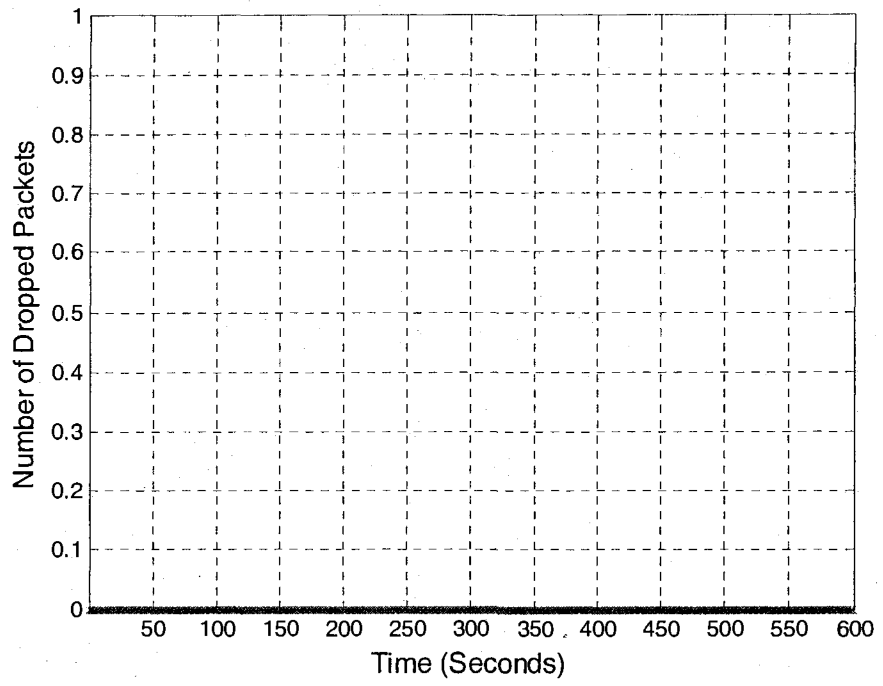

Figure 5.28 Number of Dropped Packets (Buffer Size= 1, 10, 100, 1000 Packets, XCP-CL)

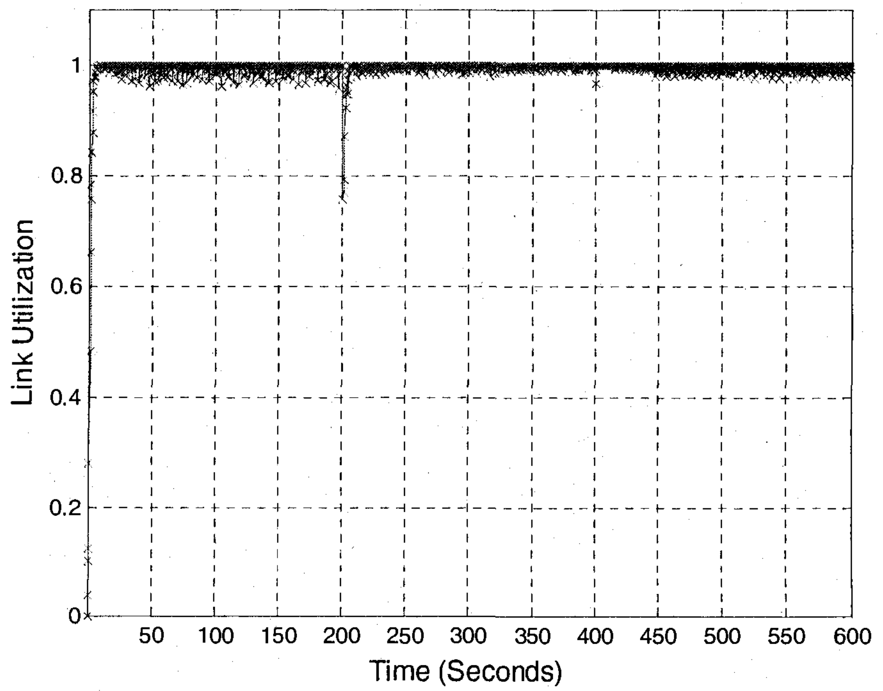

Figure 5.29 Link Utilization (Buffer Size = 1, 10, 100, 1000 Packets, XCP-CL)

Table 5.4 Queue Length, Queuing Delay, Dropped Packets & Link Utilization vs Different Buffer Size (XCP)

Buffer Size (Packets)	**1**	**10**	**100**	**1000**
Queue Length (packets)	0.10±0.001	2.5±0.001	20±0.001	700±0.001
Queuing Delay(ms)	0.02±0.001	0.4±0.001	4±0.001	120±0.001
Number of Dropped Packets	50	40	40	10
Percentage of Dropped Packets	2.1	0.87	0.0016	0.0001
Link Utilization	0.39	0.70	0.80	1.0

Table 5.5 Queue Length, Queuing Delay, Dropped Packets & Link Utilization vs Different Buffer Size (XCP-CL)

Buffer Size (Packets)	**1**	**10**	**100**	**1000**
Queue Length (Packets)	0	0	0	0
Queuing Delay(ms)	0	0	0	0
Dropped Packets	0	0	0	0
Percentage of Dropped Packets	0	0	0	0
Link Utilization	1	1	1	1

Table 5.4 and Table 5.5 summarize the performance of the two algorithms. While XCP-CL has no queue, no dropped packets, and full link utilization with any buffer size, the original XCP has a queue, drops some packets in all cases. Link utilization is under 1 in all cases except when buffer size is 1000 packets. More details are in Appendix D.

Please also note that when we set larger buffer size in the original XCP algorithm, the performance of packets drop and link utilization can be improved with the trade off for a larger queue length and queuing delay. But the requirement of the physical memory of the system design is much bigger than the XCP-CL algorithm. Note we need 1000 packets * 1000 bytes = 1000000 bytes (1MB) memory in this simple scenario (only with one input port). In XCP-CL design, we only need 1KB memory for only one packet. From a system design point of view, we prefer less memory size even the memory is not expensive these days. This simplifies the system design and improves the reliability of the whole system.

5.6. End to End Delay Performance

In this section we measured the performance of end to end delay of the system of our XCP-CL algorithm with different buffer sizes. We allow the path between a source and destination to change as time evolves. One scenario is an alternate path is found to by pass a failed link, thus resulting in a longer path length and therefore longer propagation delay. We summarize the results in the following section. The results of the original XCP algorithm are provided in Appendix E.

5.6.1 Scenario 1: Using XCP-CL, the Modified XCP Algorithm

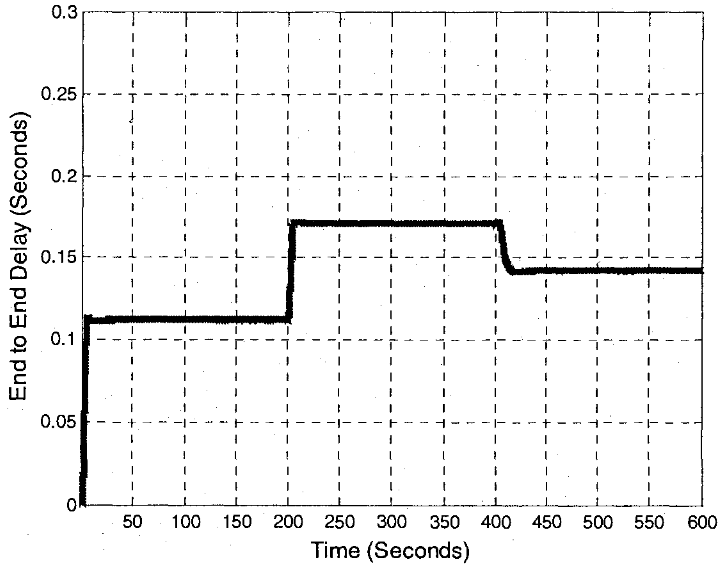

Figure 5.30 End to End Delay of Flow#1 (Buffer Size = 1 Packet, XCP-CL)

Figure 5.30 to Figure 5.34 show the end to end delay performance of 5 flows from the 5 groups of different end to end delay. For example, Figure 5.31 shows the end-to-end delay measurement of Flow#21 which, in each of the three time intervals of 0-200, 200-400 and 400-600s intervals, has increased according to the changes in the propagation delay. There is also an interesting spike at the 400s time point in Figure 5.34, which can probably be attributed to the interaction with the few flows whose propagation delays have increased. These are the expected values and we summarize them in the following table.

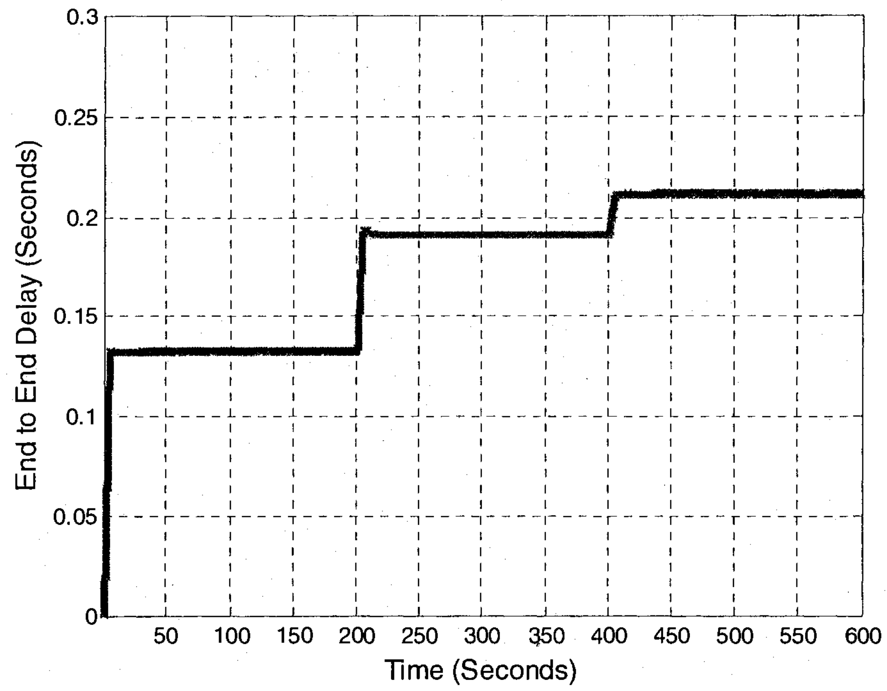

Figure 5.31 End to End Delay of Flow#21 (Buffer Size = 1 Packet, XCP-CL)

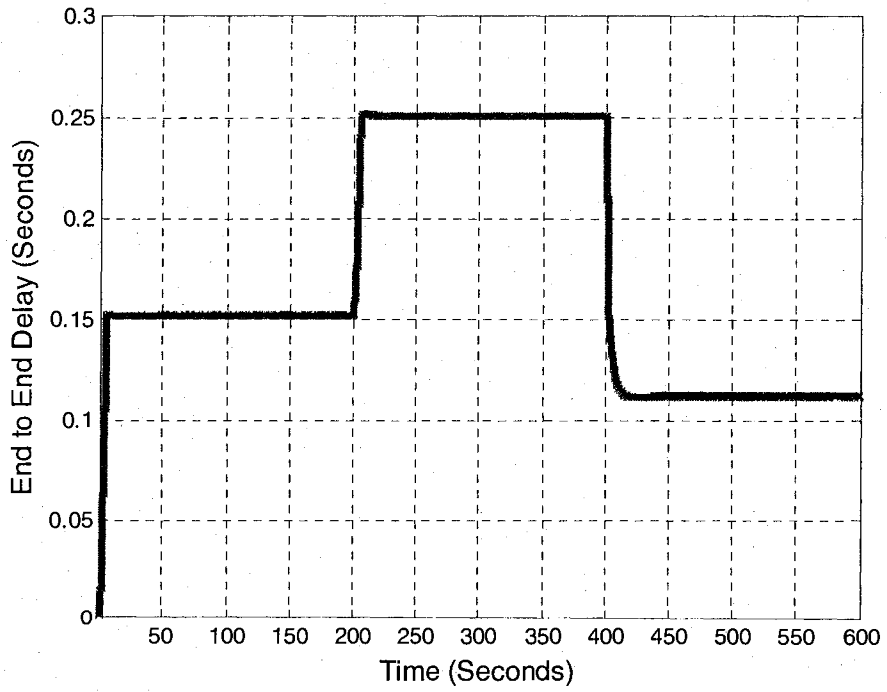

Figure 5.32 End to End Delay of Flow#41 (Buffer Size = 1 Packet, XCP-CL)

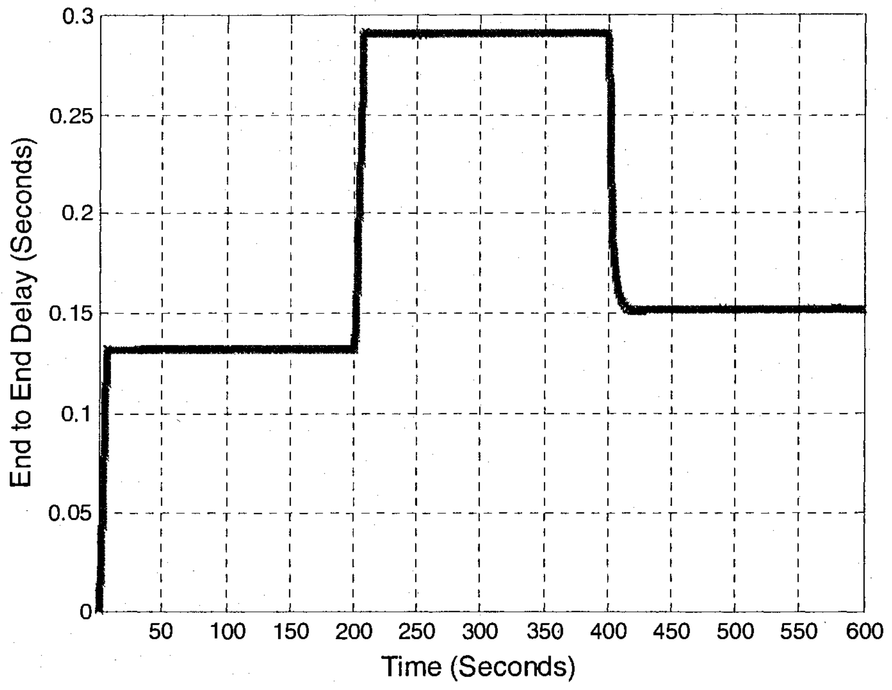

Figure 5.33 End to End Delay of Flow#61 (Buffer Size = 1 Packet, XCP-CL)

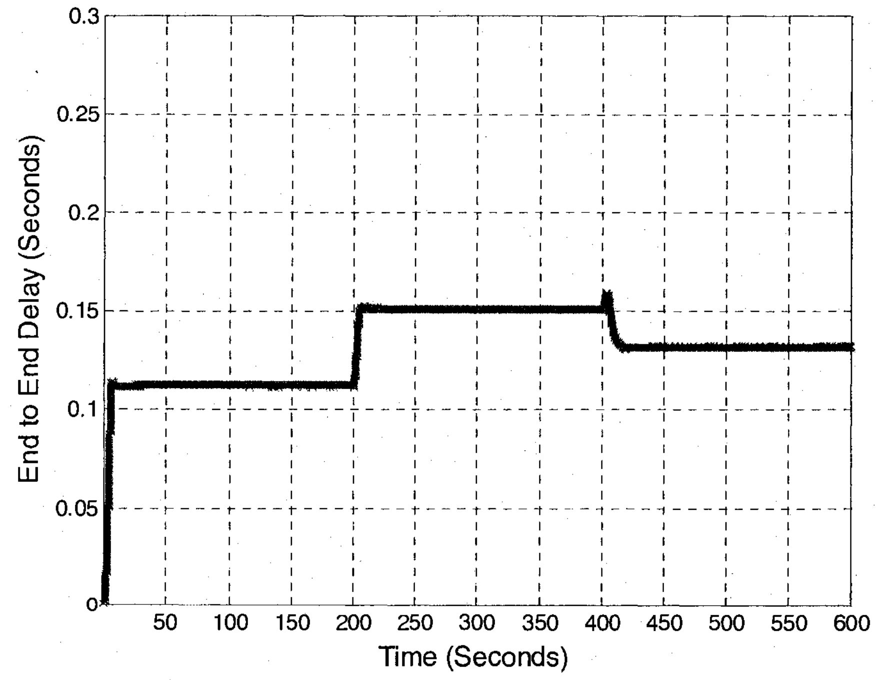

Figure 5.34 End to End Delay of Flow#81 (Buffer Size = 1 Packet, XCP-CL)

Table 5.6 summarizes these mean delay performance. As one can see from the table, the measured end to end delays are almost the same as the expected delay (i.e. the end to end propagation delay.). This is because there is no queue in the system. The end to end delay should be the same as the propagation delay. The reason of the little difference between the expected and the measured value is probably due to the processing delay coming from the simulation software. With a buffer size of 10, 100 or 1000 packets, the end to end delay is almost the same as with a buffer size of 1. This is because there is no queue in the system in these experiments. For clarity, the results are not provided here.

Table 5.6 Expected and Measured End to End Delay (Buffer Size =1 Packet, XCP-CL)

Simulation Time	**0-200s**		**200-400s**		**400-600s**	
Delay	Expected	Measured	Expected	Measured	Expected	Measured
Flow#1(s)	0.1100	0.1117	0.1700	0.1705	0.1400	0.1422
Flow#21(s)	0.1300	0.1316	0.1900	0.1905	0.2100	0.2120
Flow#41(s)	0.1500	0.1516	0.2500	0.2504	0.1100	0.1129
Flow#61(s)	0.1300	0.1316	0.2900	0.2904	0.1500	0.1528
Flow#81(s)	0.1100	0.1117	0.1500	0.1506	0.1300	0.1322

5.7. Fast Change Scenario #1: 45Mbps Bandwidth, 100 Flows

We further evaluated the system with much faster bandwidth changes in order to determine the system capability under a stressful situation. In this case, we let the bandwidth change every 25 seconds between 40Mbps and 45Mbps. The other configuration is the same as before, including the dynamic change of propagation delay.

Table 5.7 and Table 5.8 provide the change of the link bandwidth and the change in delay of different flows during the 600s simulation time. A bottleneck is created at R_{i1} every 25s when the link bandwidth changed to 40Mbps due to scheduling.

Table 5.7 Bandwidth Parameters of Fast Change#1

Simulation time	**0-25s**	**25-50s**	……	……	**550-550s**	**575-600s**
Bandwidth of R_{i1}-R_{j1}	45Mbps	40Mbps	45Mbps	40Mbps	45Mbps	40Mbps

Table 5.8 Delay Parameters of Fast Change#1

Simulation time	0-200s	200-400s	400-600s
Delay of flows 0-19(s)	0.11	0.17	0.14
Delay of flows 20-39(s)	0.13	0.19	0.21
Delay of flows 40-59(s)	0.15	0.25	0.11
Delay of flows 60-79(s)	0.13	0.29	0.15
Delay of flows 80-99(s)	0.11	0.15	0.13

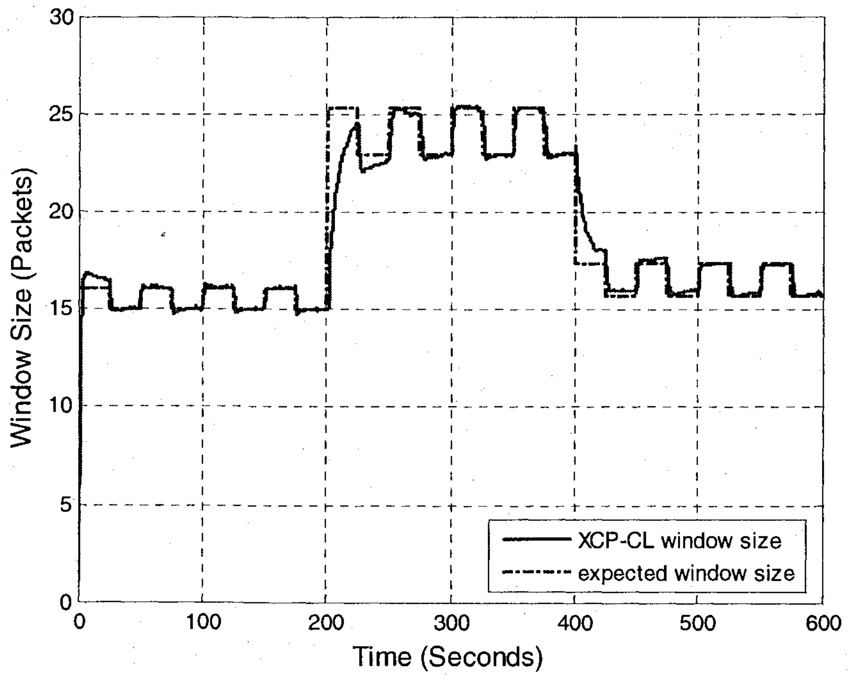

Figure 5.35 Window Size of Flow#1 and its Expected Value of Fast Change#1

Figure 5.35 shows the window size of Flow#1 with its expected window size. Window size of Flow#1 follows the change of the bandwidth. In the first 200s, the window size goes to 17 packets after 5s, then 15 packets and so on according to the changes in bandwidth. During the next 200s, it changes to 25 packets due to the dynamic change of the network configuration, then 22 packets and so on until 400s. After that it goes to 17 packets and 19 packets according to bandwidth changes and configuration change until simulation finished. It follows the expected window size well, while at start up 0s and 200s, 400s there is some disparity due to the dynamical delay changes. The other flows follow the same pattern as one can see from Figure 5.36. Notice that each flow has a different RTT so each of them has a different window size.

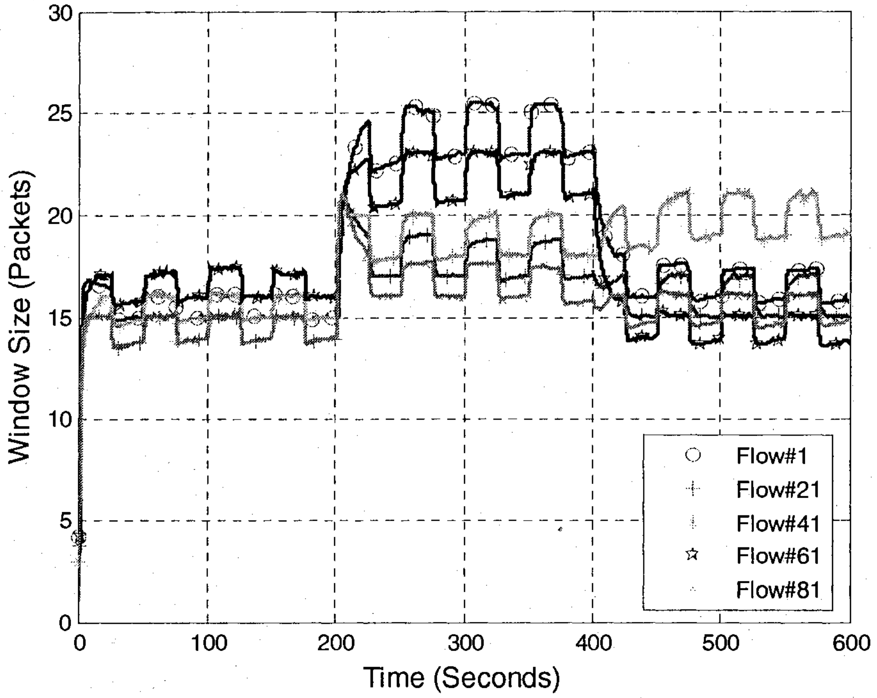

Figure 5.36 Window sizes of 5 Flows of Fast Change#1

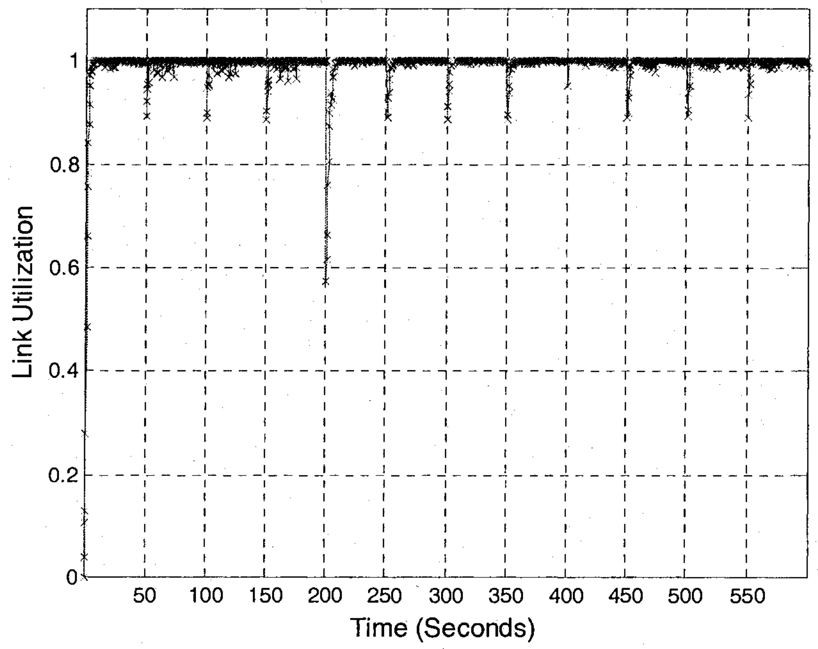

Figure 5.37 Link Utilization of Fast Change#1

Link utilization goes up to 1 quickly and stays there all the time as shown in Figure 5.37. However it has dips at the instants (every 25s) of bandwidth change and the network dynamic changes at 200s and 400s.

Under this algorithm, the edge node R_{i1} knows the link capacity change as the same as before. The simulation results of the original XCP algorithm are worse than that of XCP-CL and are not provided.

5.8. Fast Change Scenario #2: 45Mbps Bandwidth, 100 Flows

We consider a more stressful situation to evaluate our system. In this case we let the bandwidth change every 1 second. The other configuration is the same as before, including the dynamic change of propagation delay.

Table 5.9 Bandwidth Parameters of Fast Change#2

Simulation time	**0-1s**	**1-2s**	**……**	**90-91s**	**……**	**99-100s**
Bandwidth of R_{i1}-R_{j1}	45Mbps	40Mbps	45Mbps	30Mbps	……	45Mbps

Note: during time 90-91s, the bandwidth changes to 30Mbps.

Table 5.10 Delay Parameters of Fast Change#2

Simulation time	**0-33.3s**	**33.3-66.7s**	**66.7-100s**
Delay of flows 0-19(s)	0.11	0.17	0.14
Delay of flows 20-39(s)	0.13	0.19	0.21
Delay of flows 40-59(s)	0.15	0.25	0.11
Delay of flows 60-79(s)	0.13	0.29	0.15
Delay of flows 80-99(s)	0.11	0.15	0.13

Table 5.9 and Table 5.10 provide the link bandwidth change and the change in delay of different flows during the 600s simulation time. A bottleneck is created at R_{i1} during every 1s when the link bandwidth changed to 40Mbps due to scheduling. At t=90s, we changed the bandwidth from 45Mbps to 30Mbps for one second then changed it back to 45Mbps. Each group of flows change their delays to different value every 33.3s as shown in the Table 5.10.

Figure 5.38 shows the window sizes of 5 flows. The window size of Flow#1 goes to 14 packets after 3s and converges to 15 packets with small fluctuation (same with other flows) during 0-33.3s. It goes to 20 packets and down to 18 packets slowly. Then it goes to 19 packets and converges there with small fluctuation during 33.3 – 66.6s. From 66.6s to 100s, it goes to 16 packets slowly and converges to 19 packets with small fluctuation. At t=90s, it has a dip of 17 packets due the big change of the bandwidth. Congestion window size converges well but with some oscillations. Other flows have a same behavior and follow the same pattern but with different window size due to different RTT.

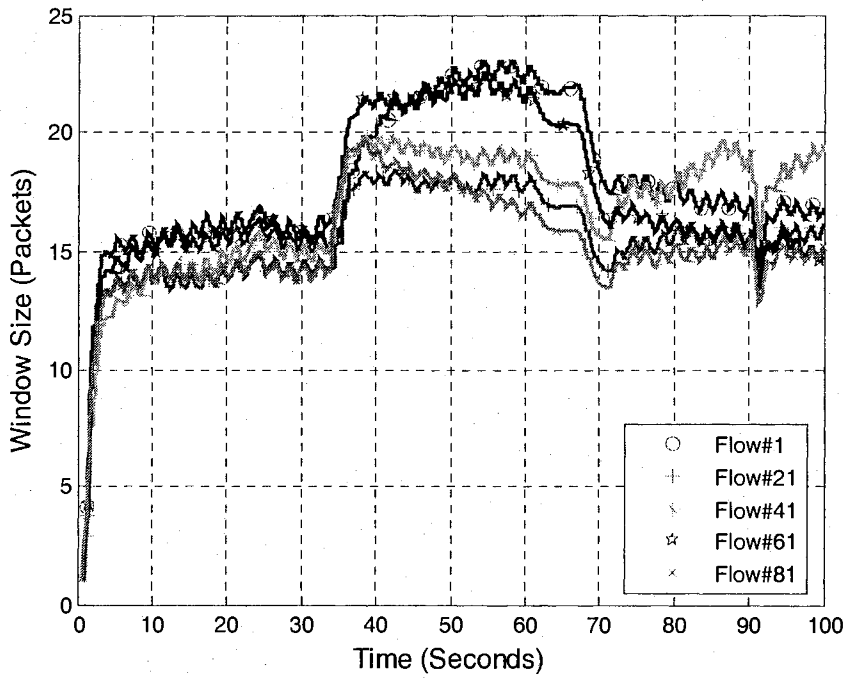

Figure 5.38 Window Sizes of 5 Flows of Fast Change#2

Link utilization goes to 1 after 3s and stay there with only a small fluctuation as shown in Figure 5.39. At time 33.3s, the network configuration changes which results in a dip in the utilization with a value of 0.62. At the sharp bandwidth change at time 90s, it results in a dip in the utilization with 0.81, due to the system adjustment to this change. The reason is the system needs some time to adjust its performance due to the delay of feedback. However, the bandwidth changes faster than the algorithm can handle as we can see from these figures.

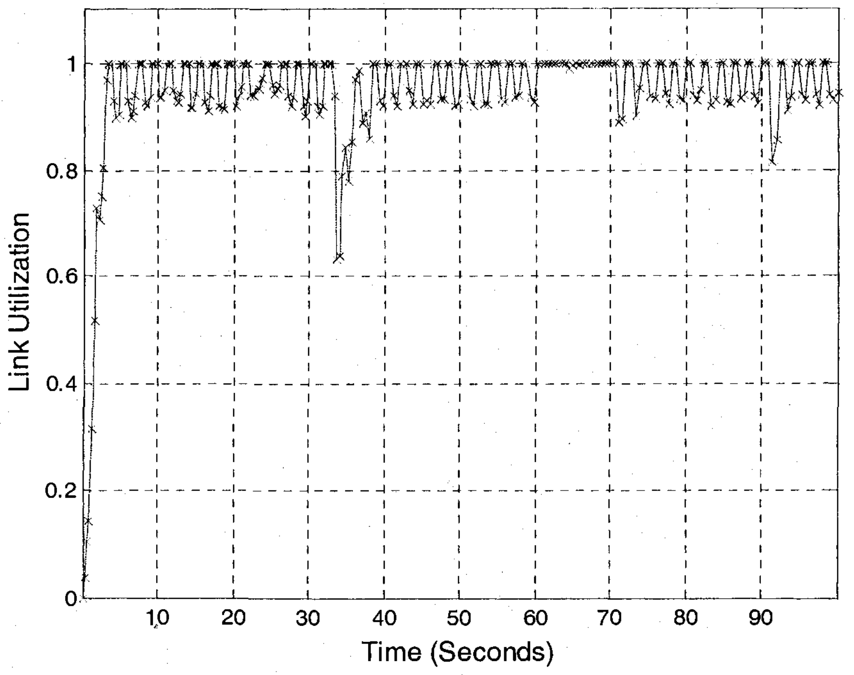

Figure 5.39 Link Utilization of Fast Change#2

XCP-CL algorithm works in this stressful scenario but with degraded performance (window size has disparity from the expected one, some packets are dropped, not shown here). The reason is that the bandwidth changes too fast and the algorithm need some time to adjust to the changes.

5.9. Concluding Remarks

In this chapter, we showed the simulation results of the network model as described in the beginning of this chapter with different link capacity. We then investigated the impact of different buffer size on the queue length, number of dropped packets, and link utilization. In both cases XCP-CL works very well, while the original XCP algorithm can not work during the congestion period as we can see from the results shown before. Finally we evaluated the XCP-CL when bandwidth changes much faster. The performance of the system degraded in this situation but our XCP-CL is still better than the original XCP. The faster the change of the bandwidth is, the worse the performance will be.

Chapter 6. Design Guideline

In this chapter we summarize the design guideline of AAPN signaling implementation and congestion control.

6.1. Scheduling Design

We desire a fast yet effective scheduling algorithm. "Fast" means the algorithm should be finished as quickly as possible in each iteration. "Effective" means the algorithm can maximizes the throughput of the network and deals with the traffic load of each edge node. These are two conflicting requirements. "Fast" requires the algorithm as simple as possible, but "effective" usually requires a high computation complexity. There should be some trade off between these two requirements. If the algorithm is based on frame signaling, it should also use some techniques to estimate the arrival traffic of each edge node to decrease the impact of the scheduling delay.

6.1.1 Slot-Based Signaling

When we choose a slot-based signaling method with a fixed amount of overhead, we prefer a larger slot length to get a better signaling efficiency. Remember that efficiency equals the length of payload (in bytes) divided by the slot length (in bytes). See Section 3.1 for details. In this signaling method, the scheduling has to be completed within one slot period. In fast switches such as the AAPN switch, the time slot is usually set to a very short period. Hence it is difficult to complete a computationally intensive scheduling algorithm.

6.1.2 Frame-Based Signaling

From our previous analysis and experience on frame-based signaling, one needs to use a large frame size to get a good signaling efficiency (defined to be the number of slots assigned to data transmission divided by frame size in number of slots). When frame size increases with a fixed number of slots for signaling, the efficiency increases. See Section 3.1 for details. However the scheduling delay (i.e. the frame period, which depends on the frame size) has to be lengthened. This is a trade off between signaling efficiency and scheduling delay. If scheduling delay is too big, the scheduling result is not good to handle the traffic arrives at the edge nodes. At the same time, queue length in the edge nodes will be increased to absorb the traffic. A longer scheduling delay is usually good for non-bursty traffic. We can use the estimator of the scheduler to handle the scheduling delay by estimating the future traffic. Now the scheduling time in frame-based signaling is relaxed to the interval of the frame period minus the time used in signaling (i.e. the number of slots that reserved for signaling times the slot period). Hence frame-based scheduling is preferred.

6.2. Out-band and In-band Signaling

As analyzed before, our XCP-CL algorithm needs the actual bandwidth in real time and our AAPN signaling protocol can pass the bandwidth information to the edge nodes (where XCP-CL algorithm runs) in each frame period in either out-band or in-band signaling. By using the cross-layer design technique, and passing the link layer information to the upper layer, our XCP-CL can use either out-band or in-band signaling (see Section 3.2 for details).

6.3. Choosing Buffer Sizes

As we can see from our simulation results and the original paper [KaHa02], XCP algorithm needs much less buffer memory than the other algorithms. This is ideal for FPGA implementation which requires as little memory as possible. According to our simulation results in previous chapter, even a buffer size of one is adequate for the modified XCP-CL algorithm. There is no queue and no dropped packet in our simulation. We only need one buffer to hold the arriving packet. For safety reason, we should set the buffer size to the pipe size if we could.

6.4. XCP-CL Algorithm Limitations

As we can see from the simulation results in Chapter 5, when the bandwidth changes too fast for the algorithm to handle, our XCP-CL algorithm does not work very well (even though it is still better than the original XCP algorithm). For example, a minimum interval of 25s to update the bandwidth is still acceptable. After experimenting with different update intervals, our finding is that the control interval *d* from the XCP controller unit has to be shorter than the interval of bandwidth change so that the system can stabilize to a new operation point. Presently the control interval is used as the feedback delay and is set to the average RTT. The value is 0.28s during the period of 0-200s, 0.36s during the period of 200-400s and 0.30s during the period of 400-600s. In other words, if bandwidth is fluctuating at interval shorter than the above value during those periods, the performance of the system would suffer greatly.

Chapter 7. Conclusions

We have successfully designed and implemented AAPN control platform and signaling protocol (including synchronization protocol, traffic allocation protocol, fault monitor protocol etc) under the Linux operating system (with more than 6000 lines of C source code). We did experiments to evaluate our system synchronization precision and scheduling time. Experiments have shown that our synchronization protocol and our traffic allocation protocol work well. Signaling overhead analysis was carried out, and design guidelines are given for our current in process fast type design.

Scheduling algorithm and edge node input buffer were successfully integrated together. Experiments have shown that the scheduling algorithm can be finished in time as required by our design. We have verified the system by correctly transferring a file between two edge nodes.

Based on the AAPN signaling protocol and by using cross-layer design, we have successfully modified and improved XCP algorithm (called XCP-CL) and let it work on our AAPN network. We have evaluated and analyzed our design through NS2 simulation in terms of window size, throughput, link utilization. The impact of different buffer size on these performance metrics is also investigated. Simulation results have shown that XCP-CL has much better performance results compared to the original XCP algorithm.

There are some lessons we have learned from our research which should be taken care of in the future research: a) A stable hardware (the optical switch in this case) would have saved us much time on debugging. Therefore we should have a full specification of the hardware for us to understand well to start with. b) We should be very careful to reduce the number of careless mistakes; for examples, the variable definition and initialization. On the other hand, more advanced debugging skills could have saved much time and effort. c) NS2 is not so well documented on its usage and requires the user to be familiar with both C++ and OTcl languages. OPNET simulator [Opne97] is probably better in this respect.

7.1. Future Work

We can extend our study to the following interesting items:

1. Implement the synchronization protocol in FPGA to eliminate the processing delay of operating system. This is in progress.
2. Implement the whole AAPN signaling protocols in FPGA, so it can be an on-shelf commercial product.
3. Modify the XCP-CL algorithm, so it can work when the bandwidth changes very fast.
4. Implement the XCP-CL algorithm in the whole network as the congestion control mechanism, and measure the performance of the real networks.

References

[802.3ah] IEEE 802.3ah, Ethernet in the First Mile Task Force, http://www.ieee802.org/3/efm/index.html, accessed 2007

[AAPN07] AAPN project, http://www.aapn.mcgill.ca/, accessed 2007

[AbRi06] Filipe Abrantes, and Manuel Ricardo, "XCP for Shared-Access Multi-Rate Media," *Proceeding of ACM SIGCOMM*, pp27-38, Pisa, Italy, Sep 11-15, 2006.

[AgBo05] Agusti-Torra, Gregor v. Bochmann, Cristina Cervello-Pastor, "Retransmission schemes for optical burst switching over star networks", *Proc. 2nd IFIP Intern. Conf.* on Wireless and Optical Communications Networks (WOCN), March 3-2, 2005, Dubai, United Arab Emirates

[AlAr06] Onur Alparslan, Shin'ichi Arakawa, and Masayuki Murata, "Paced XCP for Small Buffered Optical Packet Switching Networks", pp.35-40, *PFLDnet*, February 20, 2006

[AtLi01] Sanjeewa Athuraliya, Victor H. Li, Steven H. Low, Qinghe Yin. "REM: Active Queue Management". *IEEE Network*, 15(3):48-53, May 2001.

[AWK07] AWK software http://www.gnu.org/software/gawk/manual, accessed 2007.

[BiBa07] BigBangwidth's Lightpath Accelerator, http://www.bigbangwidth.com/accelerator.htm, accessed 2007

[BlLe01] Francois J. Blouin, Andrew W. Lee, Andrew J.M. Lee, Maged Beshai, "Comparison of two optical-core networks", *Journal of Optical Network* (1), March 23, 2001, pp. 56–65.

[BoCo04] Gregor v. Bochmann, Mark J. Coates, Trevor Hall, L. Mason, R. Vickers and Oliver Yang, "The Agile All-Photonic Network: An architectural outline", *Proceeding of Queen's University Biennial Symposium on Communications*, May 31–June 3, 2004

[ChYa04] Qiang Chen and Oliver W. W. Yang, "On Designing Self-Tuning Controllers for AQM Routers Supporting TCP Flows Based on Pole Placement, " *IEEE Journal on Selected Areas in Communications*, Vol. 22, No. 10, Dec. 10, 2004, pp.1965-1974.

[Clar82] David Clark, "Window and acknowledgment strategy in TCP", Internet Request For Comments, RFC 813, July 1982

[DeSa04] Jules Degila, Brunilde Sanso, "Topological design optimization of a yottabit-per-second lattice network", *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 9, pp. 1613 - 1625, Nov 20, 2004.

[Dixi03] Sudhir Dixit (Editor),"IP over WDM: Building the Next-Generation Optical Internet", ISBN: 978-0-471-21248-5, March 2003, Wiley InterScience

[DuKo05] Nandita Dukkipati, Masayoshi Kobayashi, Zhang-Shen Rui, "Processor Sharing Flows in the Internet", International Workshop on Quality of Service, 2005

[ElMo00] Jaafar M.H. Elmirghani, Hussein T. Mouftah, "All-Optical Wavelength Conversion: technologies and Applications in DWDM Networks", *IEEE Communication Magazine*, vol. 38, no. 3, pp. 86-92, March 2000.

[ElMo00b] Jaafar M.H. Elmirghani, Hussein T. Mouftah, "Technologies and architectures for scalable dynamic dense WDM networks," *IEEE Communication Magazine* 38, pp.58–66 (2000).

[FaPr05] Aaron Falk, Dina Katabi, Yuri Pryadkin, Internet Draft, "Specification for the Explicit Control Protocol (XCP) draft-falk-xcp-spec-02.txt", http://www.isi.edu/isi-xcp/docs/draft-falk-xcp-spec-02.html, accessed 2007

[FeSh02] Wu-chang Feng, Kang G. Shin, Dilip D. Kandlur, Debanjan Saha, "The Blue: Active Queue Management Algorithms", *IEEE/ACM Transactions on Networking*, Vol. 10, No. 4, August 2002, pp. 513-528.

[FeSh04] Huifang Feng, Yantai Shu, Oliver W.W. Yang and Hua Wang, "Prediction-based dynamic bandwidth allocation in WiFi", Proceed. *SPIE ITCOM2004 (International Symposium IT COM Information Technology and Communication)*, Vol. 5598, pp. 246-253, Oct 25-28, 2004, Philadelphia.

[FlGu01] Sally Floyd, Ramakrishna Gummadi, and Scott Shenker, "Adaptive RED: An Algorithm for Increasing the Robustness of RED," ICIR Technical Report, Available at http://www.icir.org/floyd/papers/adaptiveRed.pdf, August, 2001.

[FlJa93] Sally Floyd and Van Jacobson, "Random Early Detection Gateways for Congestion Avoidance", *IEEE/ACM Transactions on Networking*, Vol. 1, No. 4, August 4, 1993, pp. 397-413.

[Floy03] Sally Floyd. "High Speed TCP for Large Congestion Windows". IETF RFC 3649, December 2003.

[Gree01] Paul Green, "Progress in optical networking," *IEEE Commun. Mag.* 39, 54–61 (2001).

[Grov03] Wayne D. Grover, "Mesh-based Survivable Transport Networks: Options and Strategies for Optical, MPLS, SONET and ATM Networking", Prentice Hall PTR, Upper Saddle River, New Jersey, Aug 24, 2003.

[HaFl03] Mark Handley, Sally Floyed, John Padhye and Joerg Widmer,"TCP Friendly Rate Control (TRFC): Protocol Specification", Internet Engineering Task Force, RFC 3448, January 2003.

[HaSo05] Trevor J. Hall, Sofia A. Paredes, Gregor v. Bochmann. "An Agile All-Photonic Network", *The International Conference on Optical Communications and Networks, ICOCN 2005*; Bangkok, Thailand, 14-16 December 2005, pp. 365-368.

[HoMi01a] Chris V. Hollot, Vishal Misra, Don Towsley, and Wei-Bo Gong, "On Designing Improved Controllers for AQM Routers Supporting TCP Flows", *Proceedings of IEEE/INFOCOM*, pp. 1726 – 1734, April 22-26, 2001.

[HoMi01b] Chris V. Hollot, Vishal Misra, Don Towsley, Wei-Bo Gong, "A Control Theoretic Analysis of RED", *Proceedings of IEEE/INFOCOM*, April 22-26, 2001, pp.1510-1519

[HoYa04] Yang Hong, Oliver Yang and Changcheng Huang, "Self-Tuning PI TCP Flow Controller for AQM Routers with Interval Gain and Phase Margin Assignment", *Proceedings of IEEE Global Telecommunications Conference*

(Globecom 2004), Dallas, U.S.A, November 1-2, 2004, pp. 1324-1328.

[HoYa05] Yang Hong, Oliver Yang, “Design of Utility-Based Congestion Controller for Internet Traffic Based on Pole Placement”, *Proceedings of 39th annual conference on Information Sciences and Systems (CISS 2005)*, Johns Hopkins University, U.S.A, March 12, 2005.

[HoYa06] Yang Hong and Oliver Yang, “Self-Tuning Utility-Based Controller for End-to-End Congestion in the Internet”, *IEEE BROADNETS 2006*, San Jose, California, USA, October 1, 2006.

[HoYa07] Yang Hong and Oliver Yang, “Design of Adaptive PI Rate Controller for Best-Effort Traffic in the Internet Based on Phase Margin”, *IEEE Trans. Parallel and Distributed Systems*, vol. 18, no. 4, April 2007, pp. 550-561.

[IEEE03] IEEE-1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, July 16 2003, www.ieee.org/scv/ims/Meetings/IM_Society_IEEE_1588.pdf

[IrMa98] Rainer R. Iraschko and M. H. MacGregor, "Optimal Capacity Placement for Path Restoration in STM or ATM Mesh Survivable Networks", *IEEE Transactions on Networking* Vol. 6 No.3, June 1998, pp. 325-335

[ISI07] NS2 simulator software, http://www.isi.edu/nsnam/ns/, accessed 2007.

[Jaco88] Van Jacobson, “Congestion Avoidance and Control”, *Proceedings of ACM Sigcomm* 1988. pp. 314-329, Stanford, CA, August 16-18, 1988.

[Jaco90] Van Jacobson, “Modified TCP Congestion Avoidance Algorithm”, message to end2end-interest mailing list, Apr. 1990.

[JaFl05] Amit Jain, Sally Floyd, Mark Allman, and Pasi Sarolahti. “Quick-Start for TCP and IP”. Internet Draft, draft-ietf-tsvwg-quickstart-00.txt, May 2005.

[JiWe04] Cheng Jin, David X. Wei, Steven H. Low, "FAST TCP: motivation, architecture, algorithms, performance", *INFOCOM 2004*. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies, Volume 4, Issue , March 7-11, 2004 Page(s): 2490 - 2501 vol.4

[JiYa05a] Mushi Jin and Oliver W. W. Yang, “An integrated architecture enabling different resource sharing schemes for AAPN networks”, *Proceeding International conference on Optical Communication Systems and Networks, SPIE Photonics North*, Toronto, Canada, Sep 27-29, 2005.

[JiYa05b] Mushi Jin, Oliver Yang, YimingZhang, A.G.P. Rahbar, Wei Yang, “Applying time-division multiplexing in star-based optical networks”, *Electrical and Computer Engineering*, 2005, Canadian Conference, May 1-4 2005 Page(s):1141 – 1144

[JiYa06a] Mushi Jin, Oliver Yang, “A TDM Solution for All-Photonic Overlaid-Star Networks”, *Proceeding CISS2006*, Princeton, New Jersey, Mar 22–24, 2006.

[JiYa06b] Mushi Jin, Oliver Yang, “APOSN: Operation, Modeling and Performance Evaluation”, *Computer Networks*, Volume 51, Issue 6, 25 April 2007, Pages 1643-1659

[JoCh01] Amaury Jourdan, Dominique Chiaroni, and Emmanuel Dotaro, et al., “The perspective of optical packet switching in IP-dominant backbone and metropolitan networks”, *IEEE Commun. Mag.* 39 (3) (2001) 136–141.

[KaHa02] Dina Katabi, Mark Handley, and Chalrie Rohrs. “Congestion Control for

High Bandwidth-Delay Product Networks". *Proc. ACM SIGCOMM '02*, August 19-23, 2002, Pittsburgh, Pennsylvania, USA, Pages: 89 - 102.

[Kata03] Dina Katabi, "Decoupling congestion control from the bandwidth allocation policy and its application to high bandwidth-delay product networks," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, Mar. 2003.

[KrSt04] Rajesh Krishnan, James Sterbenz, Wesley M. Eddy, Craig Partridge, and Mark Allman. "Explicit Transport Error Notification (ETEN) for Errorprone Wireless and Satellite Networks". *Computer Networks*, 46(3):343–362, October 22, 2004.

[Kunn03] Srisankar S. Kunniyur, "AntiECN Marking: A Marking Scheme for High Bandwidth Delay Connections". *Proceeding ICC '03*, Anchorage, Alaska, USA, May 11-15, 2003.

[KuSr01] Srisankar Kunniyur and R. Srikant, "Analysis and Design of an Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management". *SIGCOMM'01*, August 11-13, 2001, San Diego, CA, USA.

[KuSr04] Srisankar Kunniyur and R. Srikant "An adaptive virtual queue (AVQ) algorithm for active queue management", *IEEE/ACM Transactions on Networking*, Vol. 12, No. 2, April 2, 2004, pp. 286-299.

[LeWi04] Alberto Leon-Garcia, Indra Widjaja, "*Communication Networks: Fundamental Concepts and Key Architectures*", McGraw Hill, second edition, 2004

[LiCh03] Soung Y. Liew, H. Jonathan Chao, "On slotted WDM switching in bufferless all-optical networks", *Proceeding. the 11th Symposium on High Performance Interconnects (HOTI'03)*, Aug 03, 2003, pp. 96-101.

[Linu07] Linux website, http://www.linux.org/, accessed 2007

[LoAn05] Steven H. Low, Lachlan L. H. Andrew, Bartek P. Wydrowski, "Understanding XCP: equilibrium and fairness", *Proceeding Of IEEE INFOCOM*, Miami, FL, March 13-17, 2005, Volume: 2, pp 1025- 1036 vol. 2.

[LoPa02] Steven H. Low, Fernando Paganini, Jiantao Wang Sachin Adlakha John C. Doyle, "Dynamics of TCP/RED and a scalable control", *Proceeding Of IEEE INFOCOM*, New York, NY, June 23-27, 2002, pp 239- 248 vol.1.

[LoPh05] Dino M. Lopez Pacheco, Congduc Pham, "Robust Transport Protocol for Dynamic High-Speed Networks: enhancing the XCP approach", *Proceeding of the IEEE MICC-ICON*, Nov 16-18, 2005. Volume: 1, pp.404-409.

[LoPh06a] Dino M. Lopez-Pacheco, Congduc Pham, "XCP-i: eXplicit Control Protocol for heterogeneous inter-networking of high-speed networks", *IEEE GLOBECOM 2006*, San Francisco, CA, Nov 10, 2006, pp1-6.

[LoPh06b] Dino M. Lopez-Pacheco, Congduc Pham, "Enabling Large Data Transfers on Dynamic, Very High-Speed Network Infrastructures;" Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, 2006, International Conference, April 23-29, 2006 Page(s):40 – 46.

[LoWy05] Steven H. Low, Lachlan L. H. Andrew, Bartek P. Wydrowski. "Understanding XCP: equilibrium and fairness", *Proceeding Of IEEE INFOCOM*, Miami, FL, March 13-17, 2005, Pasadena, CA, USA.

[MaGa93] Francesco Masetti, Paulette Gavignet and Dominique Chiaroni, "Fiber Delay Lines Optical Buffer for ATM Photonic Switching Applications", *Proceeding of INFOCOM* March 28-April 1, 1993, San Francisco, CA, USA.

[MaLa07] MatLab software, http://www.mathworks.com/, accessed 2007.

[Mand07] Mandrake Linux distribution, http://www.mandriva.com/, accessed 2007.

[MaSi01] Mike J. O' Mahony, Dimitra Simeonidou, David K. Hunter, and Anna Tzanakaki, The application of optical packet switching in future communication networks, *IEEE Commun. Mag.* 39 (3) (2001) 128–135.

[MaTh01] Carmen Mas Machuca, Patrick Thiran, "A review on fault location methods and their application to optical networks," *SPIE Optical Networks Magazine*, Vol. 2, No. 4, Jul 19-Aug 1, 2001, pp. 73-87.

[MaVi06] Lorne Mason, Anton Vinokurov, Ning Zhao, David Plant, "Topological design and dimensioning of Agile All-Photonic Networks", *Computer Networks*, Volume 50, No.2, 2006, pp.268–287.

[MiGo00] Vishal Misra, WeiBo Gong, Don Towsley, "Fluid-Based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED," *Proceeding ACM SIGCOMM 2000*, August 28 - September 1, 2000, Stockholm, Sweden, Pages: 151 - 160.

[Mose05] Petter Mosebekk, "A Linux implementation and analysis of the eXplicit Control Protocol (XCP)", Master thesis, University of OSLO, Norway, May 2005.

[Mukh92a] Biswancrth Mukherjee, WDM-based local lightwave networks. I. Single-hop systems, *IEEE Network* 6 (3) (1992) 12–27.

[Mukh92b] Biswancrth Mukherjee, WDM-based local lightwave networks. II. Multihop systems, *IEEE Network* 6 (4) (1992) 20–32.

[NaDa04] Kshirasagar Naik, David S. L. Wei, et.al., "A Reservation-Based Multicast Protocol for WDM Optical Star Networks", *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 9, pp. 1670-1680, Nov 2004.

[NaKo05] Kiyohide Nakauchi and Katsushi Kobayashi. Studying Congestion Control with Explicit Router Feedback Using Hardware-based Network Emulator. *Proceeding PFLDnet '05*, February 3–4, 2005, pp184-190.

[NaKo06] Kiyohide NAKAUCHI Katsushi KOBAYASHI, SIRENS: An Explicit Notification Framework for Internet Congestion Control, *ICC2006*, Sep. 20-25, Istanbul, Turkey, page(s): 12-17.

[NaWe04] Kshirasagar Naik, David S.L. Wei, et al., "A reservation based multicast protocol for WDM optical star networks", *IEEE J. Selected Areas Commun.* 22 (9) (2004) 1670–1680.

[Nise04] Norman S.Nise, Control System Engineering, John Willey &Sons, Inc, 2004

[OMNI07] OMNI project, http://www.icair.org/omninet/, accessed in 2007.

[Opne07] Opnet Simulator, http://www.opnet.com accessed in 2007.

[Otcl07] Object Tcl, http://www.otcl.org/, accessed 2007.

[OtLa99] Teunis J. Ott, T.V. Lakshman, Larry Wong, "SRED: stabilized RED," *Proceedings of IEEE/INFOCOM*, New York, March 21-25, 1999, pp. 1346 – 1355.

[PeBo06] Cheng Peng, Gregor v. Bochmann and Trevor J. Hall, "Quick Birkhoff-von Neumann Decomposition Algorithms for Agile All-Photonic Network

Cores", *proceeding of 2006 IEEE International Conference on Communications (ICC 2006)*, Istanbul, Turkey, pp.11-15, Sep. 20-25, 2006.

[Peng07] Cheng Peng, "Frame-Based Bandwidth Allocation for Agile All-Photonic Networks", PhD thesis, University of Ottawa, Ottawa, Canada, April 2007.

[PuHa06] Jian Pu and Mounir Hamdi, "QFCP: a Router-Assisted Congestion Control Mechanism for Heterogeneous Networks", *IEEE SITIS 2006*, Marco Polo Hotel, Tunisia, Dec 17-21, 2006, pp. 144-193

[QiYo99] Chunming Qiao, Myungsik Yoo, Optical burst switching (OBS) – a new paradigm for an optical Internet, *Journal of High Speed Networks* 8 (1999), pp 69–84.

[RaFl01] Kadangode K. Ramakrishnan, Sally Floyd, and David L. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC3168, September 2001.

[RaYa03] Akbar Ghaffar Pour Rahbar and Oliver Yang, "Distributed vs Centralized TDM in AAPN ", CCNR presentation, Oct 1, 2003

[RaYa05a] Akbar Ghaffar Pour Rahbar and Oliver Yang, "An Integrated TDM Architecture for AAPN Networks," *Proceeding of International conference on Optical Communication Systems and Networks, SPIE Photonics North*, Toronto, Canada, Sep 27-29, 2005, pp. 727-734..

[RaYa05b] Akbar Ghaffar Pour Rahbar, Yiming Zhang, Oliver Yang, and Sayeed Choudhury, "An integrated TDM architecture for AAPN networks", *Proceedings of SPIE"*, pp. 727-734, Vol. 5970, Feb 27, 2005, Photonic Applications in Biosensing and Imaging.

[RaYa05c] Akbar Ghaffar Pour Rahbar and Oliver Yang, "Distributed vs Centralized Scheduling in AAPN ", UO AAPN meeting, Jan 27, 2005

[ReHa99] Reza Rejaie, Mark Handley, Deborah Estrin, "RAP: An End-to-end Rate-based Congestion Control Mechanism for Real-time Streams in the Internet", *Proceedings of IEEE INFOCOM*, pp 1337-1345 vol.3, March 21-25, 1999, New York, NY, USA.

[RFC2210] RFC2210 - The Use of RSVP with IETF Integrated Services, September 1997

[RFC3448] RFC3448 - TCP Friendly Rate Control (TFRC): Protocol Specification, January 2003

[RyRu04] Seungwan Ryu, Christopher Rump and Chunming Qiao, "Advances in Active Queue Management (AQM) Based TCP Congestion Control", *Journal of Telecommunication Systems*, Volume 25, Numbers 3-4 / March, 2004, Pages 317-351

[SaCh06] Sabit Sayeed, Sadrul H. Chowdhury, Oliver Yang, Yong Deng, "The Peking Express Problem and its Applications", *23rd Biennial Symposium on Communications*, Queen's University, Kingston, Ontario, Canada, May 29 - June 1, 2006

[SeBe96] Seung-Woo Seo, K. Bergman, P.R. Prucnal, Transparent optical networks with time-division multiplexing, *IEEE Journal of Selected Areas Commun.* 14 (5) (1996) 1039–1051.

[ShYu05] YanTai Shu, Oliver Yang, Jiakun Liu, et al, "Wireless Traffic Modeling and Prediction Using Seasonal ARIMA Models", *IEICE Transactions on Com-*

mun, Vol.E88-B No.10, Oct 15, 2005, pp.3992-3999.

[Stal06] William Stallings, Data and Computer Communications, 8th Edition, Prentice Hall, Aug 2006.

[StBa00] Thomas E. Stern and Krishna, “Multiwavelength Optical Networks: A Layered Approach”, Addison-Wesley, 2000.

[StLi07] StarLight project, http://www.startap.net/starlight/, accessed 2007

[StSh03] Ion Stoicay, Scott Shenkerz , Hui Zhang; "Core-stateless fair queuing: a scalable architecture to approximate fair bandwidth allocations in high-speed networks", *Networking, IEEE/ACM Transactions* on Volume 11, Issue 1, Feb. 2003 Page(s):33 - 46

[SuGr05] Yang Su, Thomas Gross, “WXCP: Explicit Congestion Control for Wireless Multi-hop Networks”, *Proceedings of the 13th International Workshop on Quality of Service (IWQoS)*, Passau, Germany; June 21-23, 2005.

[TCL07] TCL software, http://www.tcl.tk/, accessed 2007.

[TLDP07] The Linux Documentation Project, http://www.tldp.org//, accessed 2007.

[TuTe99] Jonathan Turner, “Terabit burst switching”, *Journal of High Speed Networks* 8 (1) (1999) 3–16.

[ViBe00] Richard Vickers and Mike Beshai, “PetaWeb Architecture,” presented at *Networks 2000—Toward Natural Networks: 9th International Telecommunication Network Planning Symposium*, Toronto, Canada, pp.10–15, Sept. 10-15, 2000.

[Vlie00] Hans Van Vliet, "Software Engineering Principles and Practice", John Wiley & Sons Inc, Second edition, 2000

[Welz00] Michael Welzl, PTP: Better Feedback for Adaptive Distributed Multimedia Applications on the Internet. *Proceeding of IEEE IPCCC 2000*, February 20-22, 2000, pp: 330-336.

[WuLa90] Tsong-Ho Wu and Richard C. Lau, "A Class of Self-Healing Ring Architectures for SONET Network Applications", *Proceeding of IEEE GLOBECOM '90*, San Diego, Dec 2-5, 1990, USA, pp. 444-449.

[XiSu05] Yong Xia, Lakshminarayanan Subramanian, Ion Stoica, Shivkumar Kalyanaraman. “One More Bit Is Enough”. *Proceeding ACM SIGCOMM ’05*, pp 37-48, Philadelphia, PA, August 21-26, 2005.

[YaMu00] Shun Yao, Biswanath Mukherjee and Sudhir Dixit, Advances in photonic packet switching: an overview, *IEEE Commun. Mag.* 38 (2) (2000) 84–94.

[ZhAh05] Yongguang Zhang and Mohin Ahmed, “A control theoretic analysis of XCP, *Proceeding of IEEE GLOBECOM*”, pp.73-77, Miami, Florida, Mar 13-17, 2005

[ZhHe04] Bin Zhou, Peng He, and Gregor v. Bochmann," Blocking analysis for time-space switched all-optical networks ", pp. 756-761, *proceedings of the 4th IASTED international multi-conference wireless and optical communication*, July 8-10, 2004, Banff, Canada

[ZhHe05] Yongguang Zhang and Thomas R. Henderson. “An implementation and experimental study of the explicit control protocol (XCP)”. *Proceeding of IEEE INFOCOM*, pp. 1037-1048, Miami, Florida, March 13-17, 2005.

[ZhMo04] Jun Zheng and Hussein T. Mouftah, “Optical WDM Networks: Concepts and Design Principles”, Wiley-IEEE Press, New Jersey, Jul. 2004.

[ZhPe06] Jun Zheng, Cheng Peng, and Gregor v. Bochmann, "A fault detection and localization scheme for all-optical overlaid-star TDM networks," Proc. of 2006 *International Conference on Communications and Networking in China (CHINACOM'06)*, Beijing, China, Oct. 25-27, 2006

[ZhYa04] Yiming Zhang, Oliver Yang and Haomei Liu, "A Lagrangean Relaxation and Subgradient Approach to the Routing and Wavelength Assignment Problem in WDM Networks", *IEEE J-SAC OCN Series* Nov 2004, pp. 1752-1765.

Appendix A: Summary of the XCP Algorithm

XCP algorithm was first proposed in [Kata03]. It decouples efficiency control from fairness control, so we can use different control law for each controller and each flow acquires the spare bandwidth quickly while achieving good fairness.

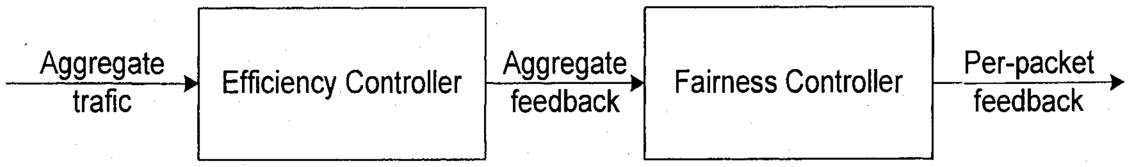

Figure A.1 CP Algorithm Principle

Figure A.1 shows the main idea of the algorithm. Traffic is counted as an aggregated value. This value is used to control the efficiency of the link capacity. The output, aggregated feedback, is used control fairness. Then the "per packet" feedback is brought back to the sender through ACK to adjust the window size. It uses explicit feedback to tell the sender to what degree is the congestion.

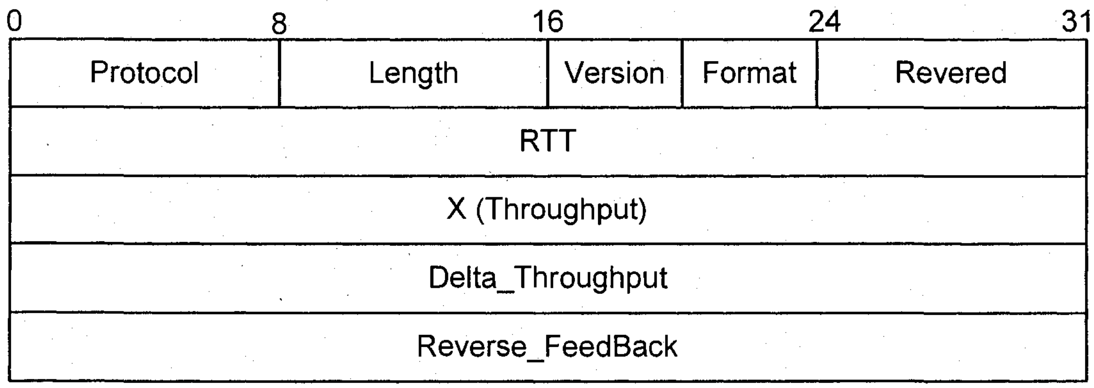

Figure A.2 XCP Header

Figure A.2 show the basic idea of XCP algorithm. XCP introduces a new congestion header. The important filed is the "Reverse_Feedback". It is initialized by the sender as the desired value and modified by the routers through the path if the allowed bandwidth is lower than the previous set value. So after the packet has passed through the whole path and returned to the sender, it has the bottleneck information of the path. The sender then adjusts its sending rate according to this information to avoid the congestion.

EC: Efficiency Controller

The efficiency controller aims at maximizing link utilization while minimizing packet drop rate and persistent queues without considering the fairness issue. The EC computes a desired increase or decrease in the aggregate traffic rate as the input traffic changes. This aggregate feedback can be computed at each control interval as follow:

$$\phi_{(Feedback)} = \alpha.S - \beta.\frac{Q}{d} \qquad \text{(A1)}$$

In the equation, α and β are the control parameters, with constant values 0.4 and 0.226 as set in [KaHa02]. S is spare bandwidth (i.e. link capacity C minus input bandwidth y). Q is the persistent queue. And d is the average RTT, the control interval. As we can see from the equation, the feedback takes the information of the spare bandwidth and the persistent queue length into consideration. It uses an MIMD (Multiplicative-Increase Multiplicative-Decrease) control law to quickly adapt to the spare bandwidth and drain the queue. So XCP algorithm can converge to the full utilization in a short time with almost no queue waiting in the router with comparison to TCP.

FC: Fairness Controller

The goal of the FC is to allocate the feedback to individual packets of each flow to achieve fairness.

The FC uses an AIMD (Additive-Increase Multiplicative-Decrease) control law and computes the per-packet feedback for each flow according to the following rules:

1. If $\phi_{(Feedback)} > 0$, allocate feedback equally to each flow
2. If $\phi_{(Feedback)} < 0$, allocate feedback to flows proportionally to their current throughputs

As long as the aggregate feedback is not zero, controller will tell the sender to adjust the traffic until each flow converges to their fairness.

When efficiency is close to optimal, (that is, the feedback is near zero) the above policy becomes inefficient. In this case, the algorithm uses bandwidth shuffling technique to converge to fairness further. The shuffled traffic is computed as follows:

$$h = \max(0, \gamma . y - |\phi|) \tag{A2}$$

where y is the input bandwidth, ϕ is the feedback. The control parameter γ is usually set to 0.1, so that the utilization can be better than 90% in the shuffling stage when $\phi \approx 0$. In this stage the controller allocates and de-allocates some of the bandwidth to each flow in order to allow it to obtain its fair share of the bandwidth (i.e., the fairness issue), and to ensure its utilization to be near 100% (with some fluctuation due to adjustment) while keeping the total allocated traffic constant.

Since XCP algorithm separates the congestion control and fairness control, and it uses MIMD and AIMD control law separately, it outperforms most of the up-to-date algorithms in high bandwidth-delay product networks.

Appendix B: Flow Charts of Some Important Slot Processing Functions

We list some important flow charts to understand more about the implementation of AAPN signaling protocol.

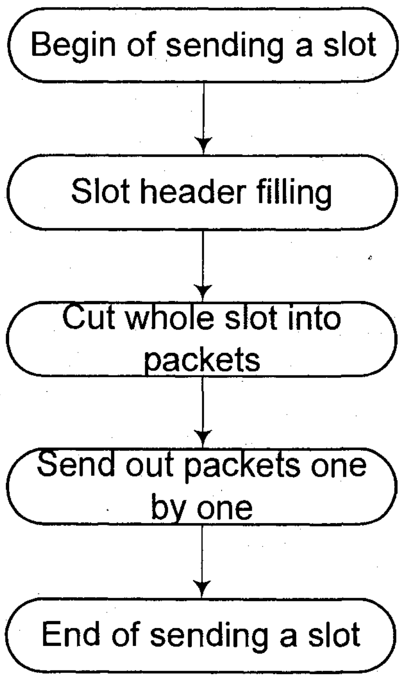

Figure B.1 Flow Chart of Slot Sending Function

Figure B.1 shows the slot sending flow chart (in slow type and FPGA version A). Each slot is taken from the virtual queue buffer, filled with slot header. It then is chopped into Ethernet packets and sent to another PC or FPGA.

Figure B.2 shows the receiving flow chart in slow prototype. It uses the packet sequence, source ID and destination ID as the magic numbers to find the beginning of slot header, then receives the whole slot.

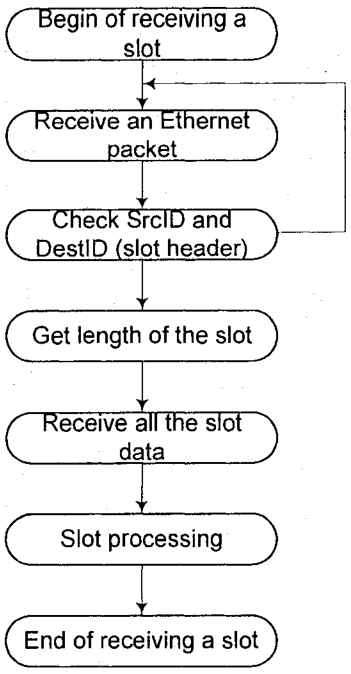
Begin of receiving a slot
Receive an Ethernet packet
Check SrcID and DestID (slot header)
Get length of the slot
Receive all the slot data
Slot processing
End of receiving a slot

Figure B.2 Flow Chart of Slot Receiving Function

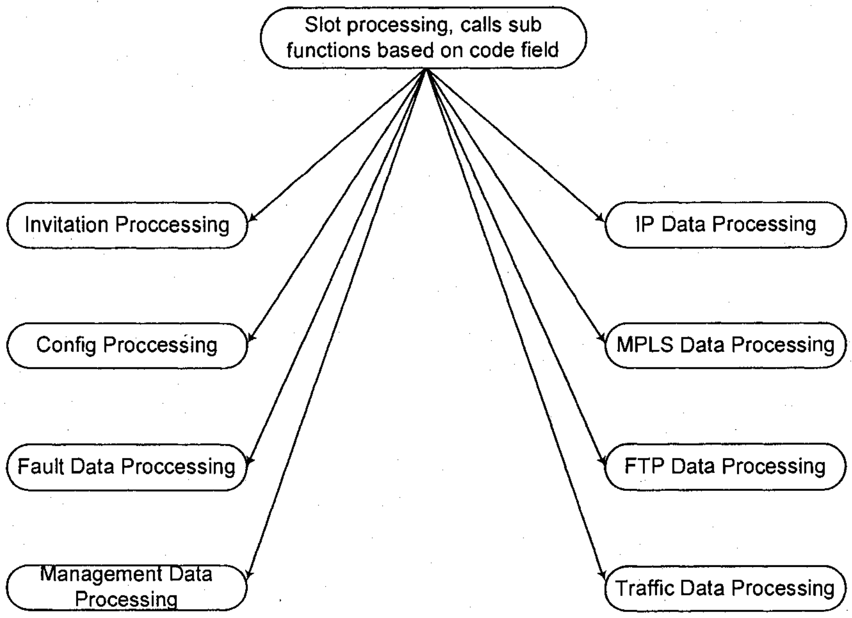

Figure B.3 Flow Chart of Slot Processing Function

Figure B.3 shows the flow chart when the scheduler receives a slot. It first checks the code field to see what type of slot it received and then takes related actions by calling the sub-functions to finish the processing.

Appendix C: Fluid Modeling

When we talk about congestion control, we should mention fluid model [MiGo00] [HoMi01a] [HoMi01b] which is a milestone in this research area. It introduces a mathematic model of congestion control that can be used when we design and analyze the controller.

The non-linear dynamic model for TCP/AIMD/AQM is described by the following coupled, nonlinear differential equations:

$$\dot{W}(t) = \frac{1}{\tau(t)} - \frac{W(t)W(t-\tau(t))}{2\tau(t-\tau(t))} p(t-\tau(t)) \qquad \text{(C1)}$$

$$\dot{x} = \frac{W(t)}{\tau(t)} N(t) - \mu \qquad \text{(C3)}$$

where $\dot{x}$ is the time-derivative of x, μ is link capacity in packets/sec. It shows how the average queue length $x(t)$ reacts to the changes on the average window size $W(t)$, the average round-trip time $\tau(t)$, the traffic load $N(t)$, the link capacity μ and the nonlinear mechanism of the AQM algorithm (e.g., RED [FlJa93]).

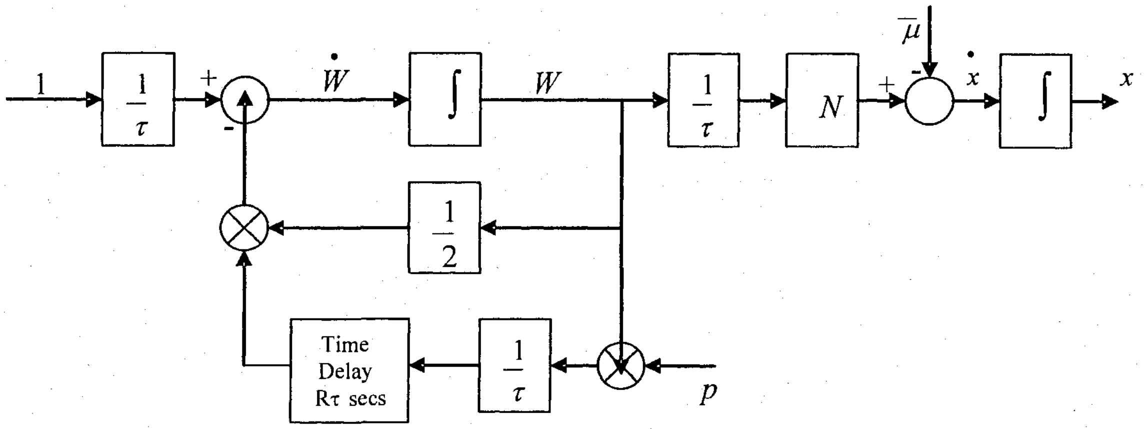

Figure C.1 Fluid Model of TCP Window Based Congestion Control

Figure C.1 shows the compensator studied in [HoMi01b], the well-known RED controller [FlJa93]. It consists of an LPF (Low-Pass Filter) and nonlinear gain element.

The form of the LPF was derived in [HoMi01b] while nonlinear gain element is a mechanism that marks packets with a dropping probability p as a function of average queue length x_{avg}. Parameter p is varying between two queue thresholds min_{th} and max_{th}, with a slope of $L_{RED}=p_{max}/(max_{th}-min_{th})$. Combining the two elements, the transfer function model for RED [FlJa93] is $C_{RED}=L_{RED}/(S/K+1)$ where $K=\log_e(1-\alpha)/\delta$ and α is the queue averaging parameter while δ is the sampling frequency.

Based on the linearized mathematic model, we can use advanced control techniques to design the controller and analyze our design.

Appendix D: Impact of Buffer Size on Queue Length, Packet Drops, Link Utilization

This appendix shows performance of queue length, queuing delay, packet drops and link utilization of the original XCP algorithm with different buffer sizes. The performance of our XCP-CL is provided in Section 5.5.

Buffer Size = 1 Packet

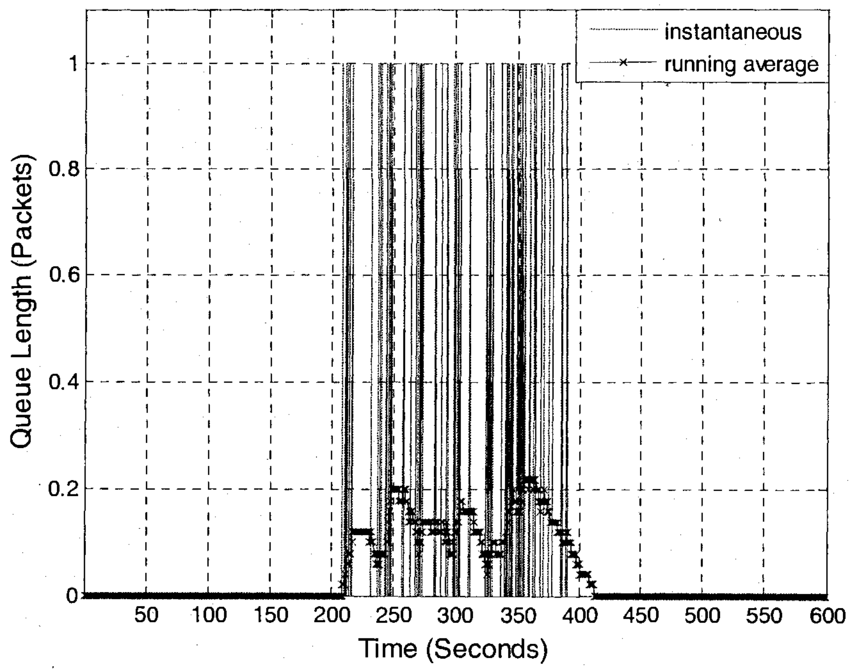

Figure D.1 Queue Length (Buffer Size =1 Packet, XCP)

Note: the thick line during time interval 0-100s and 420-600s is due to the over crowding of 'X' in the running average curve. It does not mean there is a nonzero queue length. It is likewise for other figures later on.

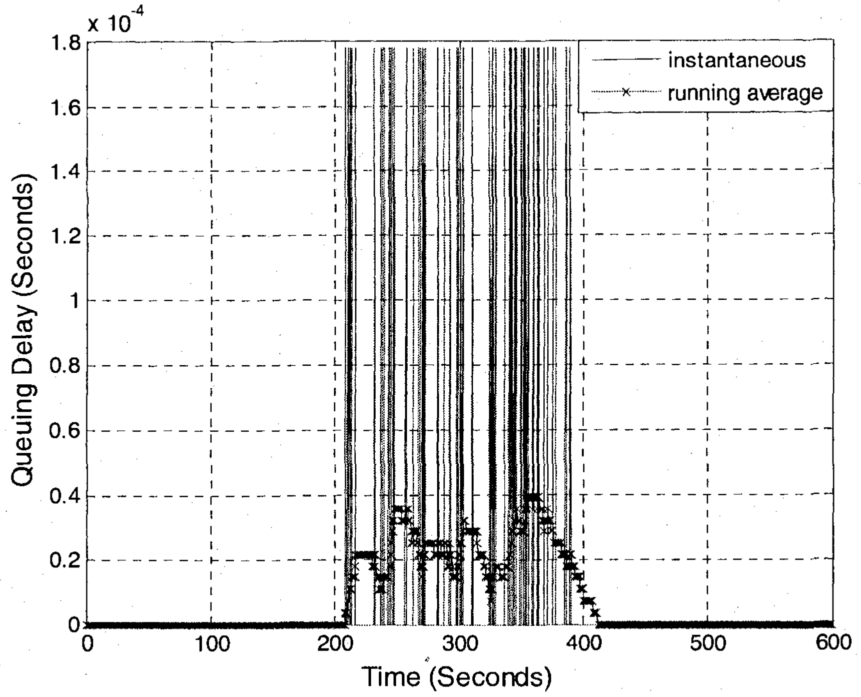

Figure D.2 Queuing Delay (Buffer Size =1 Packet, XCP)

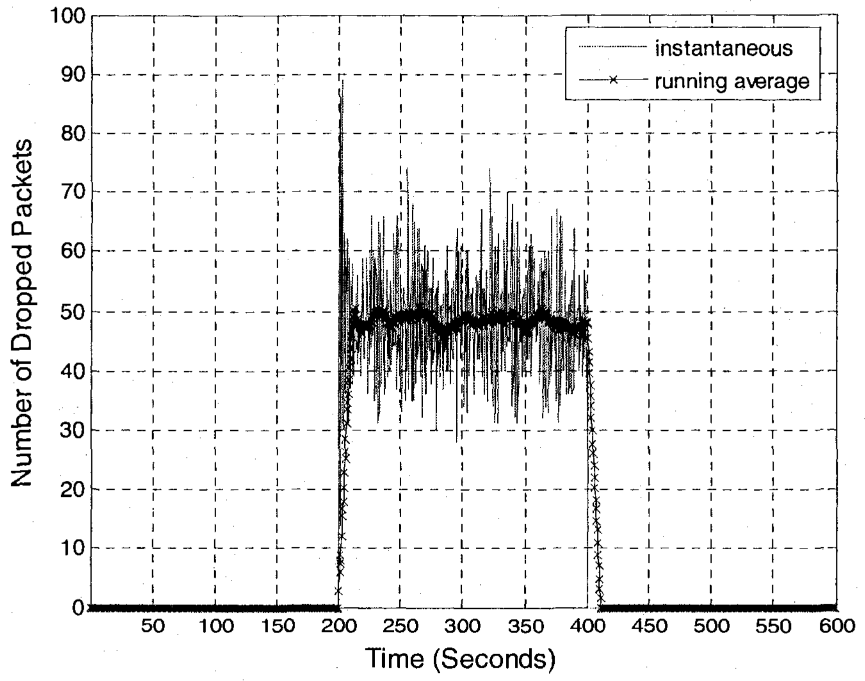

Figure D.3 Number of Dropped Packets (Buffer Size =1 Packet, XCP)

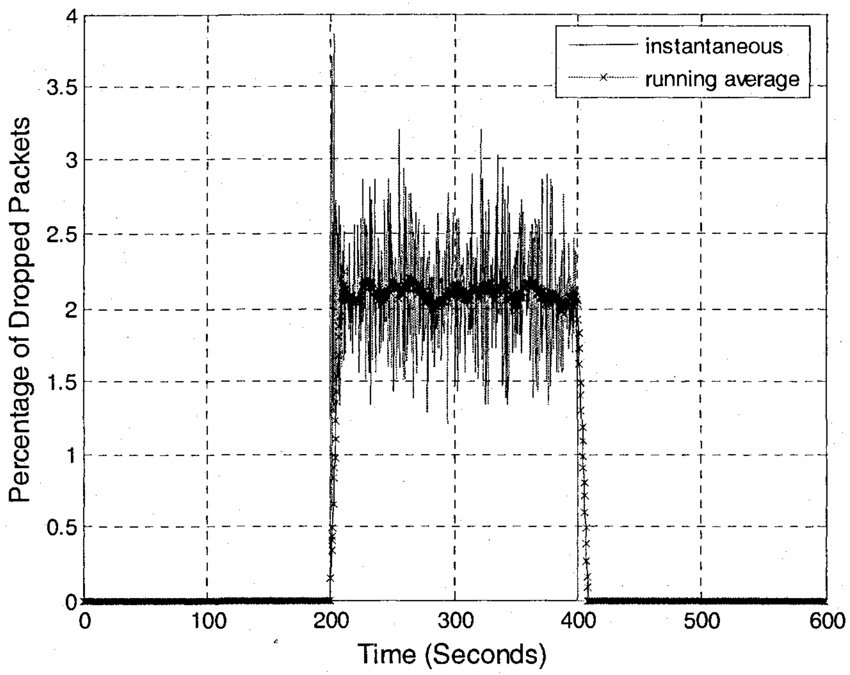

Figure D.4 Percentage of Dropped Packets (Buffer Size =1 Packet, XCP)

Figure D.1 and Figure D.2 show the queue length and queuing delay respectively. We plot the running average and the instantaneous queue length and queuing delay to have a good view of these parameters. Running average is used to smooth out short-term fluctuations, thus highlighting longer-term trends. As one can see from these figures, there is no queue during the interval 0-200s and 400-600s when the system has the correct capacity information. However, during the interval 200-400s, when the system has wrong capacity information, it has non-zero queue length. The instantaneous queue length oscillates between 0 and 1 packet with a running average of about 0.1 packet. Because the instantaneous queue length is random oscillation, the running average is also random between the maximum and minimum boundaries (the other curves are the same). Consequently, the instantaneous queuing delay oscillates between zero and $1.8\text{x}10^{-4}$s with the running average of about $0.2\text{x}10^{-4}$s.

During the congestion interval, the senders are overloading the system and there are not enough buffers to absorb the overloaded traffic since we set the buffer size as one in this experiment. As a result, it incurs a high packet drop with a running average of 50 packets as shown in Figure D.3. The number of dropped packets shown in this figure is defined as the accumulated dropped packets in a time interval of *d*, the control interval. One can see there are always a number of packets are dropped. We plot the curve of the percentage of dropped packets in Figure D.4 as well. The percentage is defined as the number of dropped packets in a time interval divided by the total number of packets the router should receive in this time interval. It has the same pattern of the number of dropped packets (it is a scale down version). The running average is 2.1%.

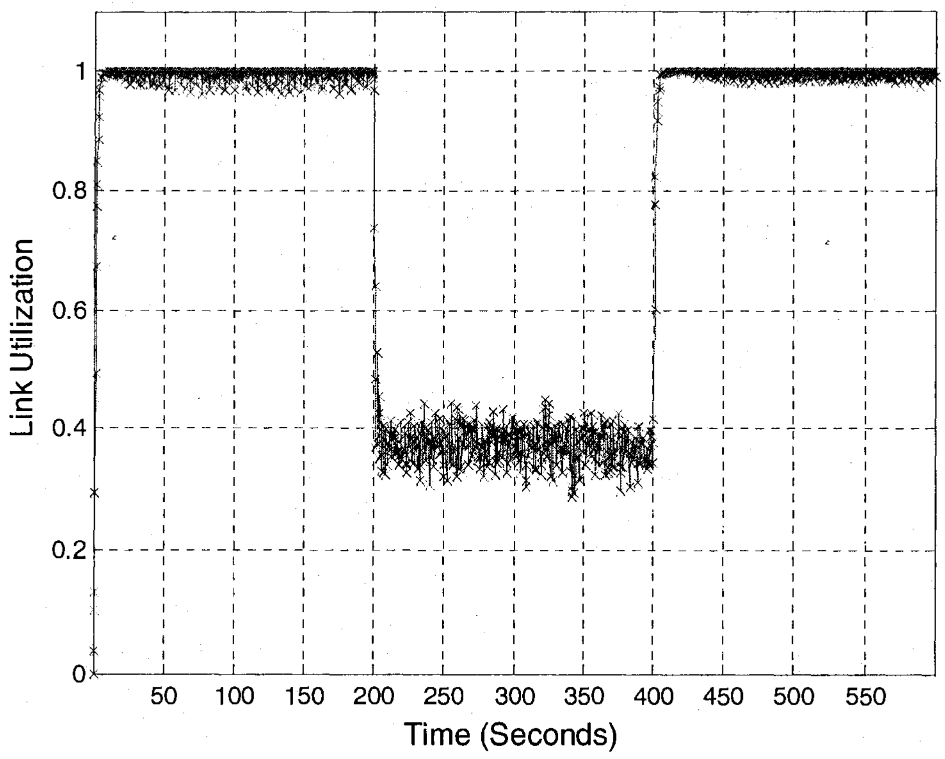

Figure D.5 Utilization (Buffer Size =1 Packet, XCP)

Figure D.5 shows the link utilization performance. During interval 0-200s, the utilization goes to 1 in about 3s and stays at full utilization. During congestion interval of 200-400s, when the system does not have the correct congestion information, the router drops packets and the utilization goes down to about 0.39. During time 400-600s, the congestion disappeared, and the utilization goes back to 1 again very quickly.

<u>Buffer Size = 10 Packets</u>

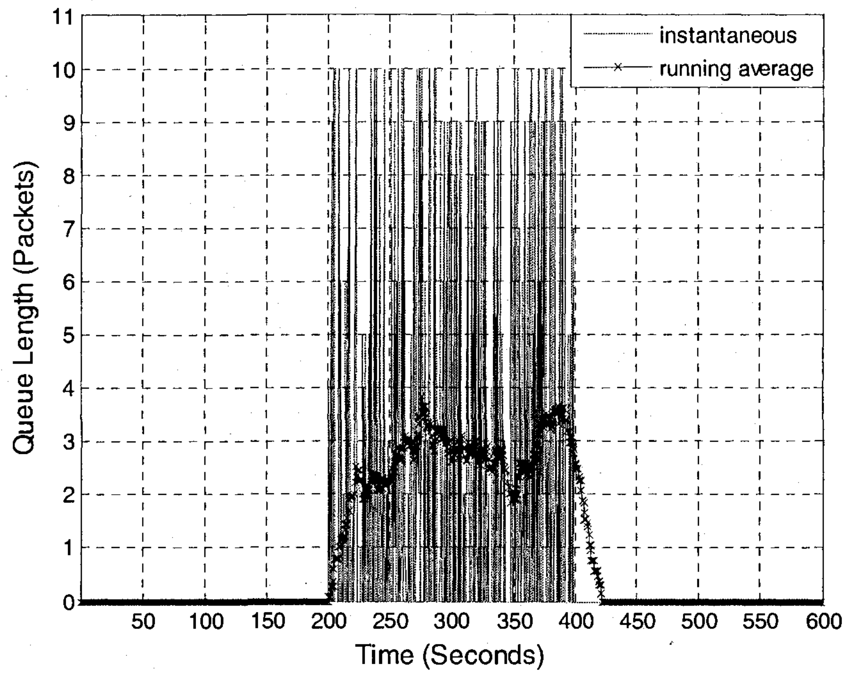

Figure D.6 Queue Length (Buffer Size =10 Packets, XCP)

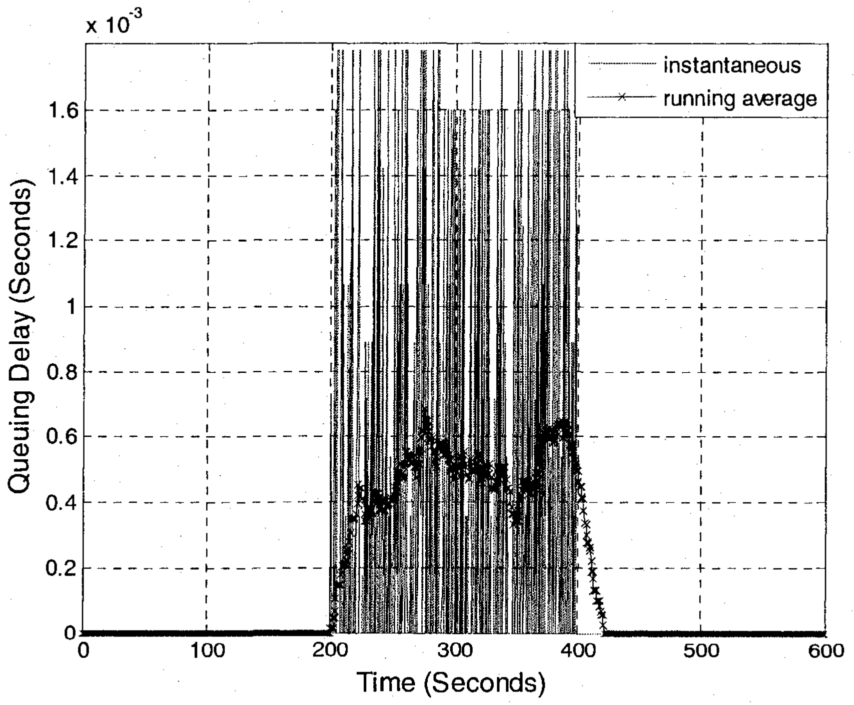

Figure D.7 Queuing Delay (Buffer Size =10 Packets, XCP)

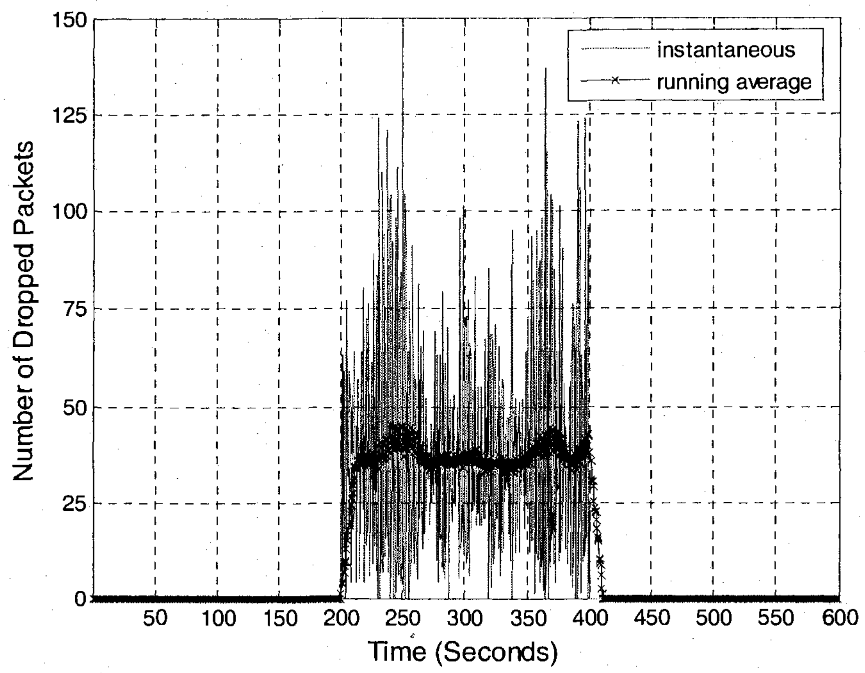

Figure D.8 Number of Dropped Packets (Buffer Size =10 Packets, XCP)

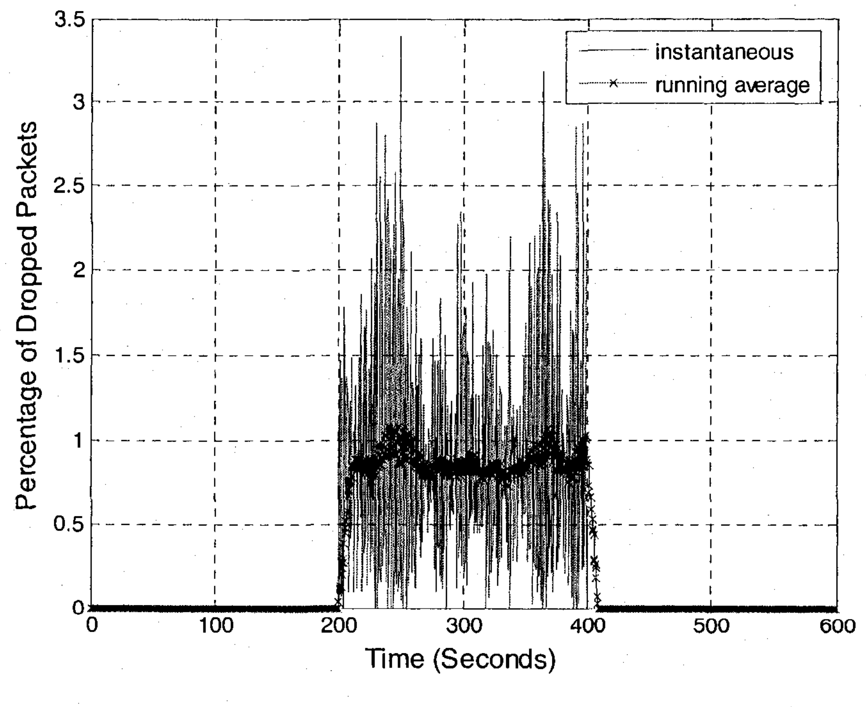

Figure D.9 Percentage of Dropped Packets (Buffer Size =10 Packets, XCP)

Figure D.6 to Figure D.7 show the queue length, queuing delay, packets drop performance separately. The system appears to have the similar performance as before. Again, one can see from these figures, there is no queue during time 0-200s and 400-600s when the system has the correct capacity information. Again there is no queuing delay. During time 200-400s, when the system has wrong capacity information, it has non-zero queue length. The instantaneous queue length oscillates between 0 and 10 packets (the largest buffer size) with a running average of about 2.5 packets. This is due to the buffer size is set to 10 packets and the system has the wrong congestion information, so it tries to send much traffic and uses up all the buffers. Consequently, the instantaneous queuing delay oscillates between zero and $1.8\text{x}10^{-3}$s with the running average of about $4\text{x}10^{-4}$s.

During time 0-200s and 400-600s, there is no dropped packet as before. During congestion period of 200-400s, there are also many packets dropped with a running average of 40 packets as shown in Figure D.8. The running average of the percentage of dropped packets is 0.87% as shown in Figure D.9. Since we set the buffer size as 10 packets in this experiment, fewer packets are dropped compared to the situation as we set the buffer size as 1. Notice during this congestion period, at some points there is no dropped packet while there are always dropped packets in the previous experiment. The reason is that the system now has a buffer size of 10 packets. At these sample points the system has the ability to absorb all the arrived traffic when needed.

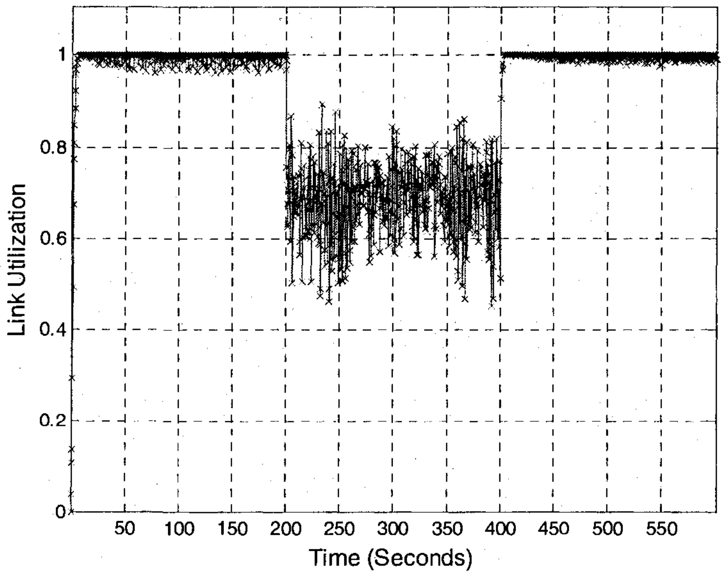

Figure D.10 Link Utilization (Buffer Size =10 Packets, XCP)

Figure D.10 shows the link utilization performance. Again as before, during time 0-200s, utilization goes to 1 in about 3s and stays at full utilization. During congestion interval of 200-400s, when the system does not have the correct congestion information, the router drops some packets. However, in this experiment the buffer size is set to 10 packets. So the system can buffer more packets when there is some overload and drops less packets. Hence the utilization only goes down to about 0.7 (with heavy oscillation), much better than with a buffer size as 1 packet (with the utilization as 0.39). This is the trade off for longer queuing delay. During time 400-600s, the congestion disappears, and the utilization goes back to 1 very quickly again.

<u>Buffer Size = 100 Packets</u>

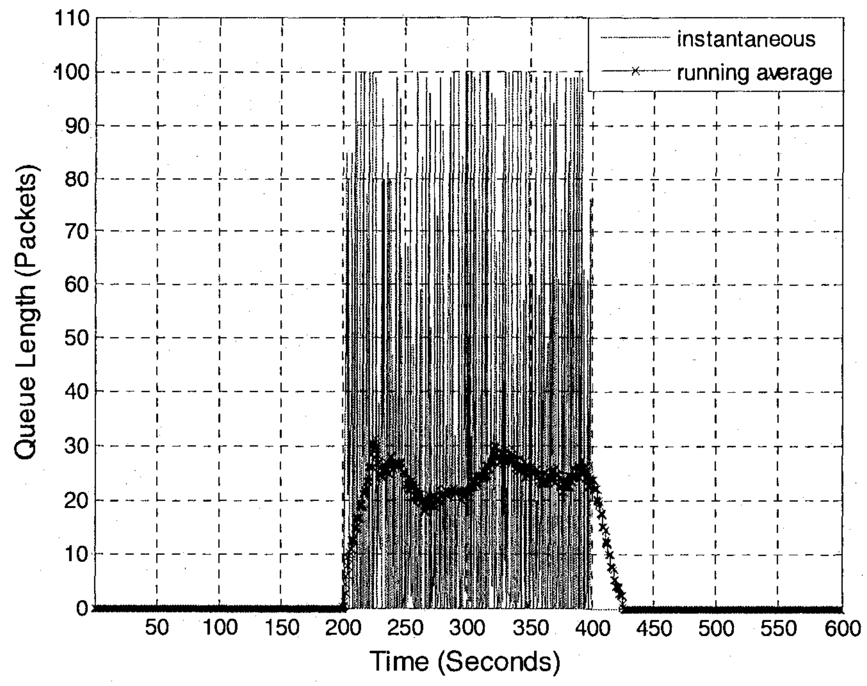

Figure D.11 Queue Length (Buffer Size =100 Packets, XCP)

Figure D.11 to Figure D.12 show the queue length, queuing delay, dropped packets performance separately. Again, it has the similar behavior as previous experiments as one can see from these figures. There is no queue during time 0-200s and 400-600s when the system has the correct capacity information. Again there is no queuing delay. During time 200-400s, when the system has wrong capacity information, it has non-zero queue length. The instantaneous queue length oscillates heavily between 0 and 100 packets with a running average of about 20 packets. Consequently, the instantaneous queuing delay oscillates between zero and $1.8x10^{-2}$s with a running average of about $3.6x10^{-3}$s.

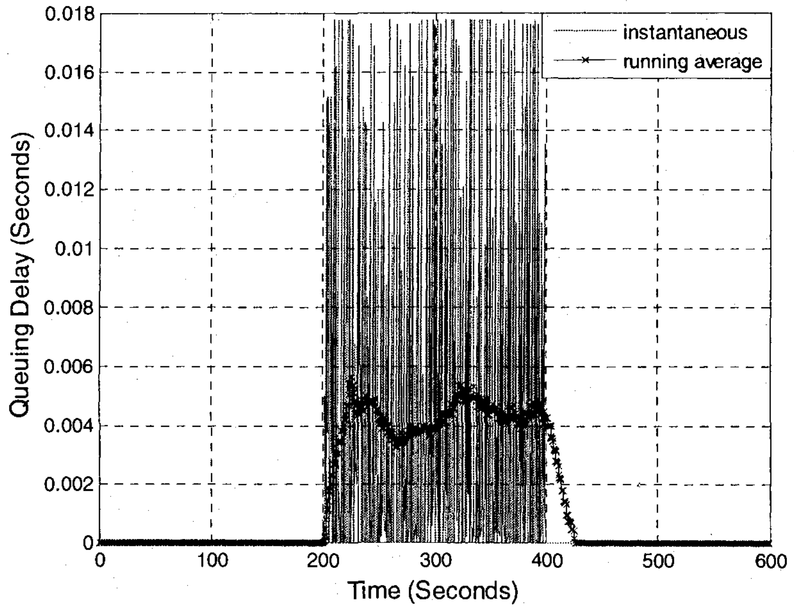

Figure D.12 Queuing Delay (Buffer Size =100 Packets, XCP)

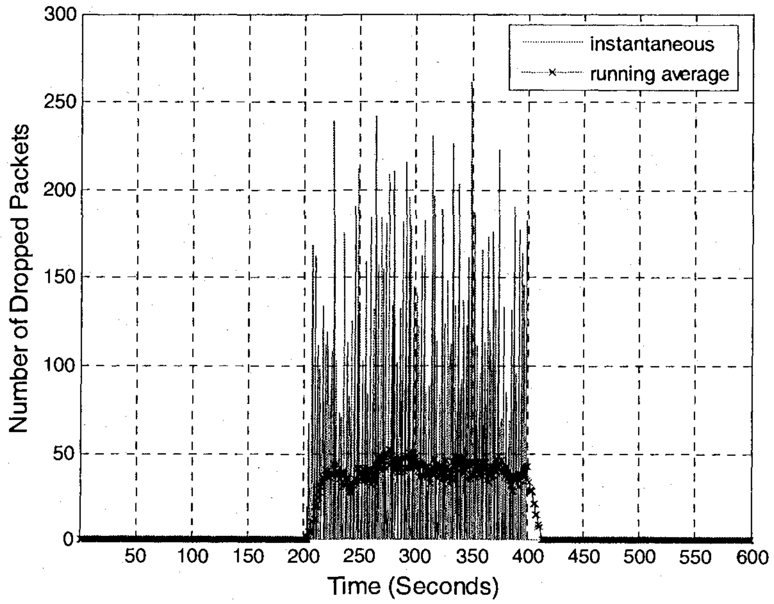

Figure D.13 Number of Dropped Packet (Buffer Size =100 Packets, XCP)

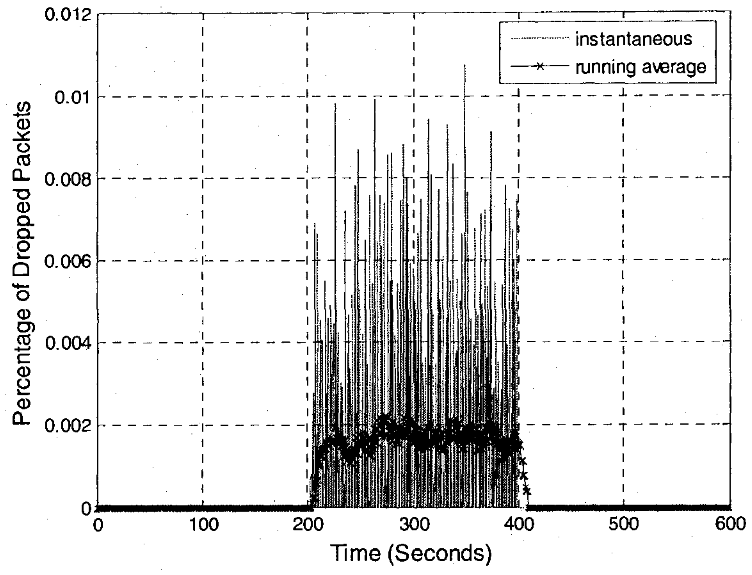

Figure D.14 Percentage of Dropped Packets (Buffer Size =100 Packets, XCP)

There are also many packets dropped with a running average of 40 packets as shown in Figure D.13. This value is approximately the same as when we set the buffer size to 10 packets. The possible reason is the system now have a bigger buffer (100 packets) to absorb the traffic but still has the wrong congestion information. The net result is similar to the situation of setting the buffer size to 10 packets. Figure D.14 shows the percentage of dropped packets which has running average of 0.002%.

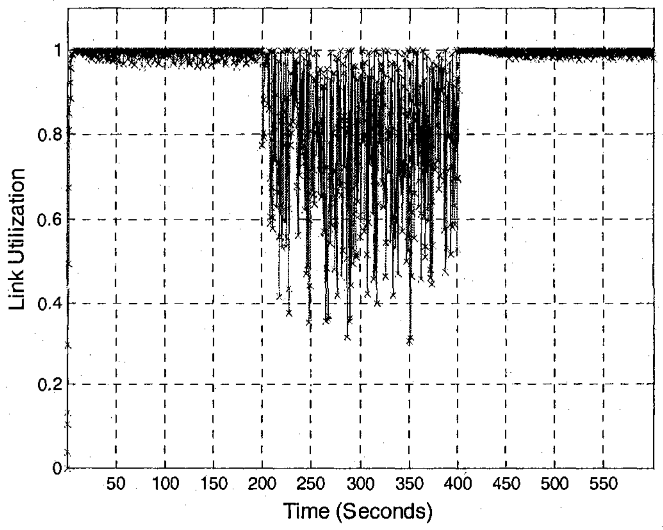

Figure D.15 Link Utilization (Buffer Size =100 Packets, XCP)

Figure D.15 shows the link utilization performance. Again as before, during time 0-200s, utilization goes to 1 in about 3s and stays at full utilization. During congestion interval of 200-400s, when the system does not have the correct congestion information, the router drops some packets. However, in this experiment the buffer size is set to 100 packets. So the system can buffer more packets when there is some overload and drops less packets. Hence the utilization only goes down to about 0.8, a little better than with a buffer size as 10 packets (with the utilization as 0.7). During time 400-600s, the congestion disappeared, and the utilization goes back to 1 again very quickly.

Buffer Size = 1000 Packets

Figure D.16 to Figure D.17 show the queue length, queuing delay separately. Again the system has a similar behavior with some difference after the congestion disappears. One can see from these figures, there is no queue during time 0-200s when the system has the correct link capacity information. Again there is no queuing delay. During time 200-400s, when the system has wrong capacity information, it has non-zero queue length. The instantaneous queue length oscillates heavily between 0 and 1000 packets with a running average of about 700 packets. Consequently, the instantaneous queuing delay oscillates between zero and 1.8×10^{-1}s with a running average of about 1.2×10^{-1}s.

During time 400-600s, the congestion disappears, however, there is a huge queue in the system. It takes a long time for the router to drain this queue as one can see from the figure (we did not plot the curve after the queue is completely drained out, it is enough to show the performance of the system). This phenomenon is much obvious compared to the results as we set the buffer size to 1, 10, and 100 packets.

There are much fewer packets were dropped during congestion with a running average of 10 packets as shown in Figure D.18. This is because we set the buffer size as 1000 packets, so the system can absorb much traffic as needed. There is again the trade off for much longer queuing delay. During time interval 400-600s when the congestion disappears, even there is a huge queue in the system there is still no dropped packet. The algorithm has the correct congestion information (including the queue length), so the system works very well.

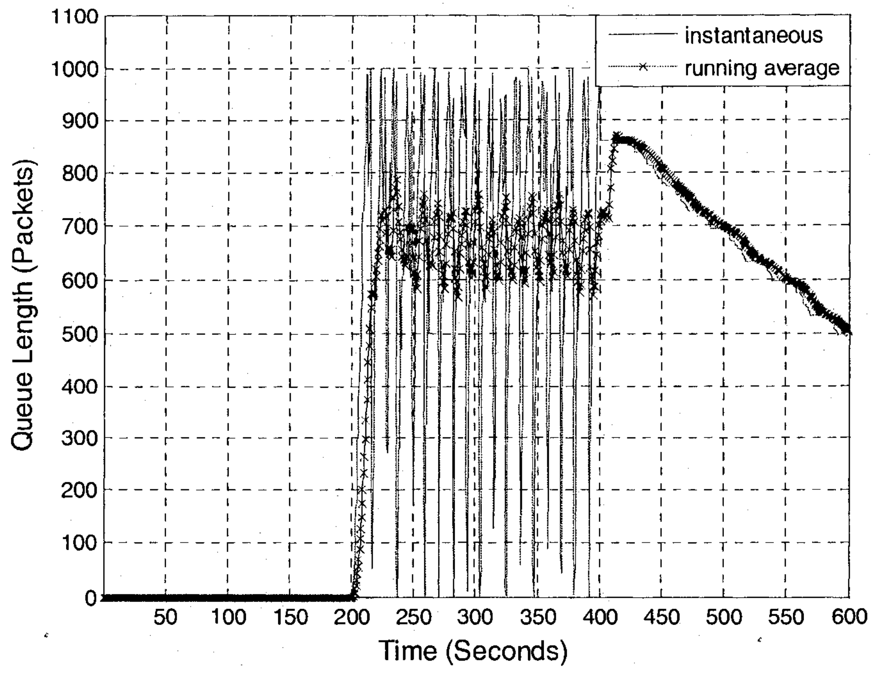

Figure D.16 Queue Length (Buffer Size =1000 Packets, XCP)

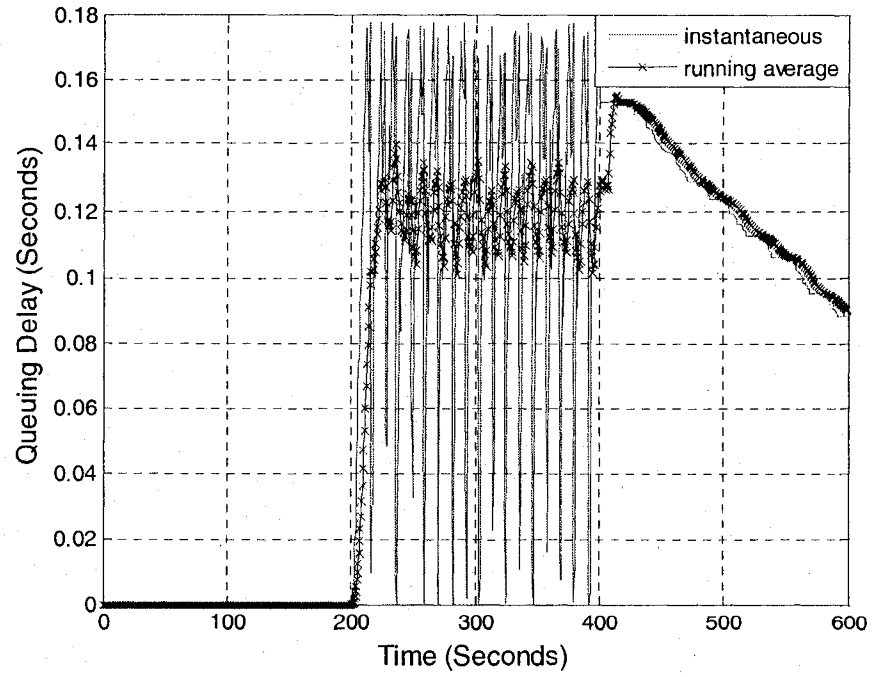

Figure D.17 Queuing Delay (Buffer Size =1000 Packets, XCP)

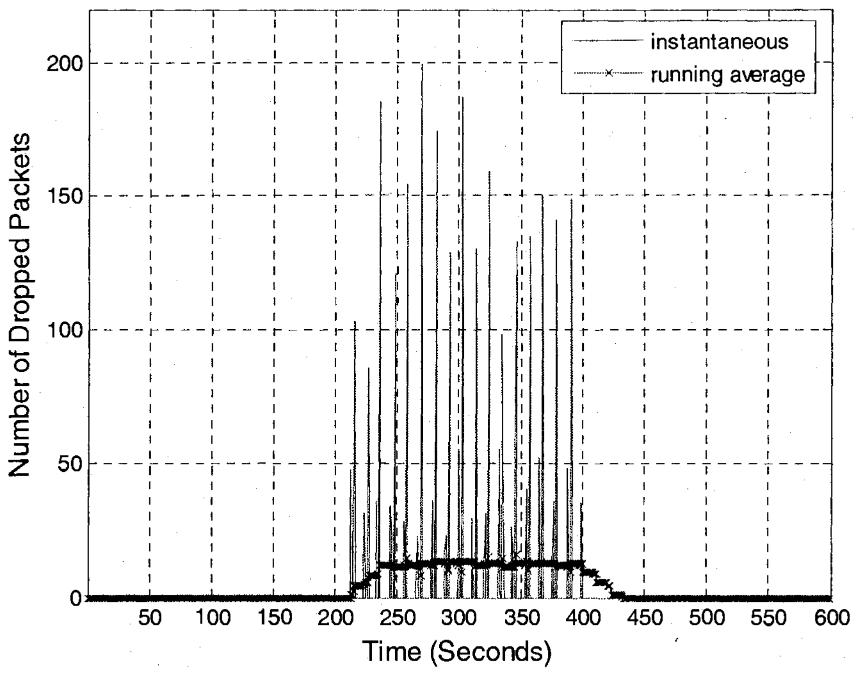

Figure D.18 Number of Dropped Packets (Buffer Size =1000 Packets, XCP)

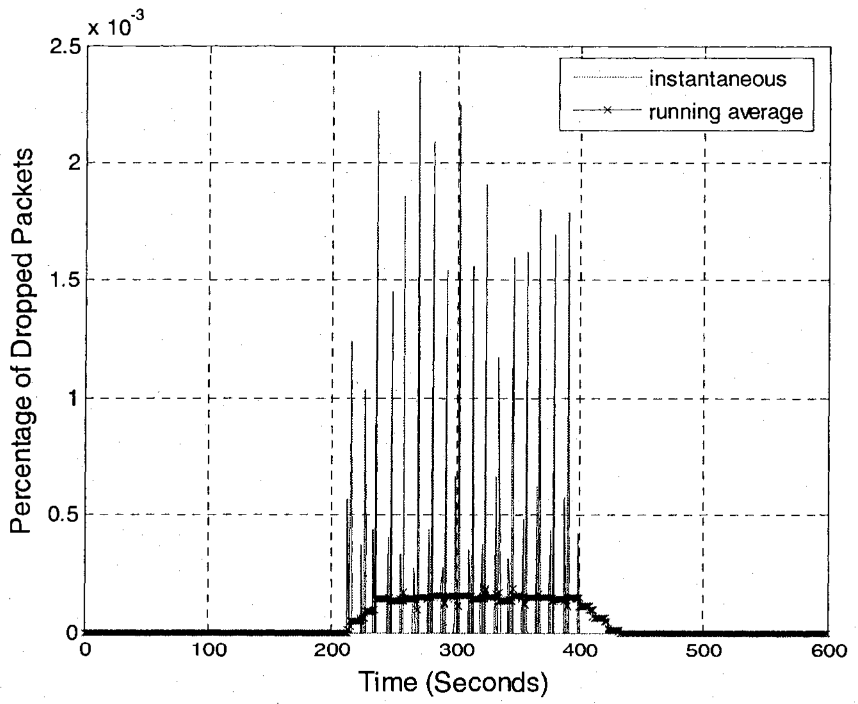

Figure D.19 Percentage of Dropped Packets (Buffer Size =1000 Packets, XCP)

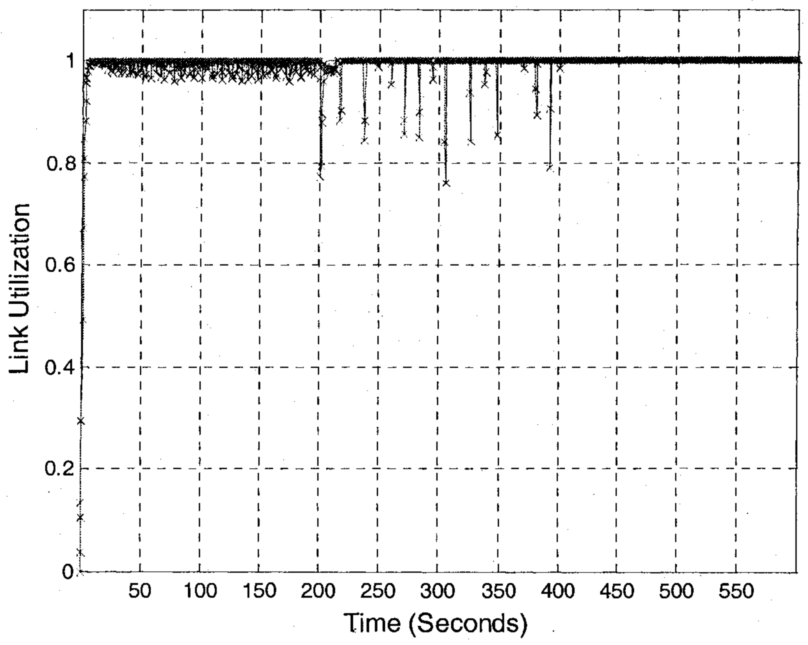

Figure D.20 Link Utilization (Buffer Size =1000 Packets, XCP)

Figure D.20 shows the link utilization performance. Again as before, during time 0-200s, utilization goes to 1 in about 3s and stays at full utilization. During congestion interval of 200-400s when there is congestion, the router drops some packets. However, in this experiment the buffer size is set to 1000 packets, the number of dropped packets is less than the experiments before. Sine the system can buffer more packets when there is some overload. The buffered packets keep the link almost as full utilization with some points of dips. As we said before, this is at the cost of longer queuing delay. During time 400-600s, the congestion disappears, and the utilization goes back to 1 again very quickly.

Appendix E: End to End Delay Performance

In this appendix we show the measured end to end delay performance of the system of the original XCP algorithm with different buffer sizes (1, 10, 100 and 1000 packets). We use the network model as shown in Case 1. Detailed parameters are shown in Table 4.1.

Because the queuing delay is small compared to the propagation delay, the figures of the end to end delay of the original XCP with a buffer size of 1, 10, 100 packets are almost the same with that of our XCP-CL and with not much interests. We omitted here.

Table E.1 Expected and Measured Delay (Buffer Size =1 Packet, XCP)

Simulation Time	**0-200s**		**200-400s**		**400-600s**	
Delay	Expected	Measured	Expected	Measured	Expected	Measured
Flow#1(s)	0.1100	0.1117	0.1700	0.1705	0.1400	0.1415
Flow#21(s)	0.1300	0.1316	0.1900	0.1907	0.2100	0.2114
Flow#41(s)	0.1500	0.1516	0.2500	0.2503	0.1100	0.1119
Flow#61(s)	0.1300	0.1316	0.2900	0.2907	0.1500	0.1519
Flow#81(s)	0.1100	0.1117	0.1500	0.1505	0.1300	0.1315

Table E.2 Expected and Measured Delay (Buffer Size =10 Packets, XCP)

Simulation Time	**0-200s**		**200-400s**		**400-600s**	
Delay	Expected	Measured	Expected	Measured	Expected	Measured
Flow#1(s)	0.1100	0.1117	0.1700	0.1713	0.1400	0.1419
Flow#21(s)	0.1300	0.1316	0.1900	0.1916	0.2100	0.2118
Flow#41(s)	0.1500	0.1516	0.2500	0.2517	0.1100	0.1121
Flow#61(s)	0.1300	0.1316	0.2900	0.2917	0.1500	0.1524
Flow#81(s)	0.1100	0.1117	0.1500	0.1515	0.1300	0.1317

Table E.3 Expected and Measured Delay (Buffer Size =100 Packets, XCP)

Simulation Time	**0-200s**		**200-400s**		**400-600s**	
Delay	Expected	Measured	Expected	Measured	Expected	Measured
Flow#1(s)	0.1100	0.1117	0.1700	0.1763	0.1400	0.1422
Flow#21(s)	0.1300	0.1316	0.1900	0.1959	0.2100	0.2119
Flow#41(s)	0.1500	0.1516	0.2500	0.256	0.1100	0.1131
Flow#61(s)	0.1300	0.1316	0.2900	0.2959	0.1500	0.1541
Flow#81(s)	0.1100	0.1117	0.1500	0.1557	0.1300	0.1325

Table E.1 to Table E.3 show the expected and the measured end to end delay of the same 5 flows with a buffer size of 1, 10 or 100 packets respectively. Again note that the queuing delay is so small compared to the propagation delay that the figures are almost the same as with a buffer size of 1 in XCP-CL algorithm simulation. During time interval 0-200s and 400-600s, the measured end to end delays are almost the same as the expected values. During the time of 200-400s when there is congestion in the system in these experiments, the measured values are a little bigger compared to the expected one. This is because of the queuing delay and the processing delay.

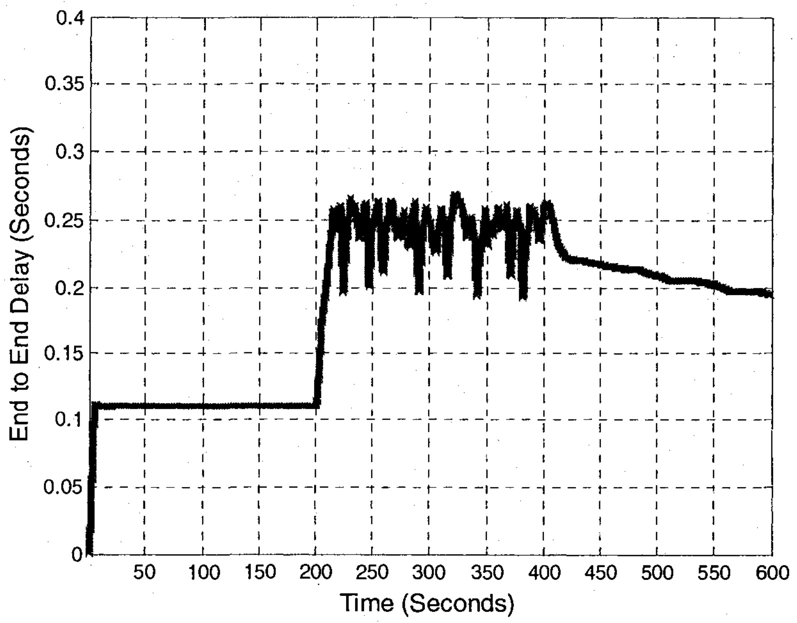

Figure E.1 End to End Delay of Flow#1 (Buffer Size =1000 Packets, XCP)

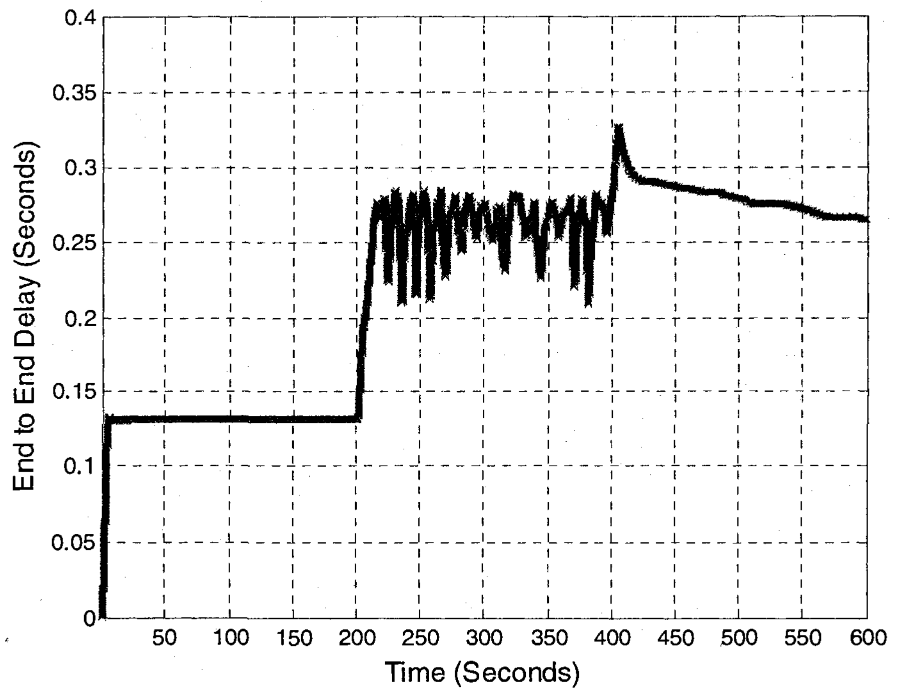

Figure E.2 End to End Delay of Flow#21 (Buffer Size =1000 Packets, XCP)

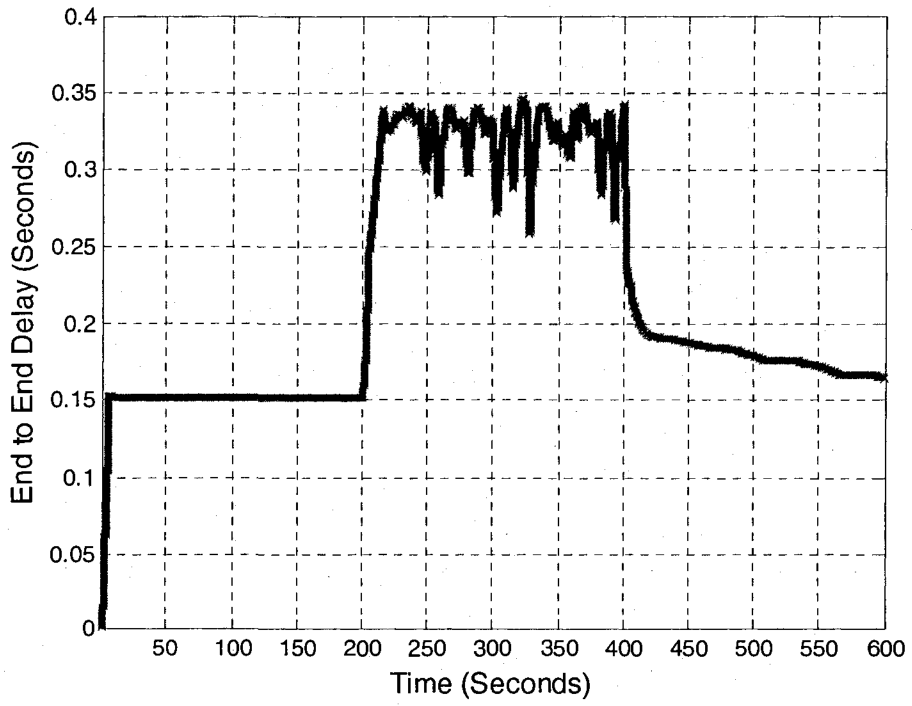

Figure E.3 End to End Delay of Flow#41 (Buffer Size =1000 Packets, XCP)

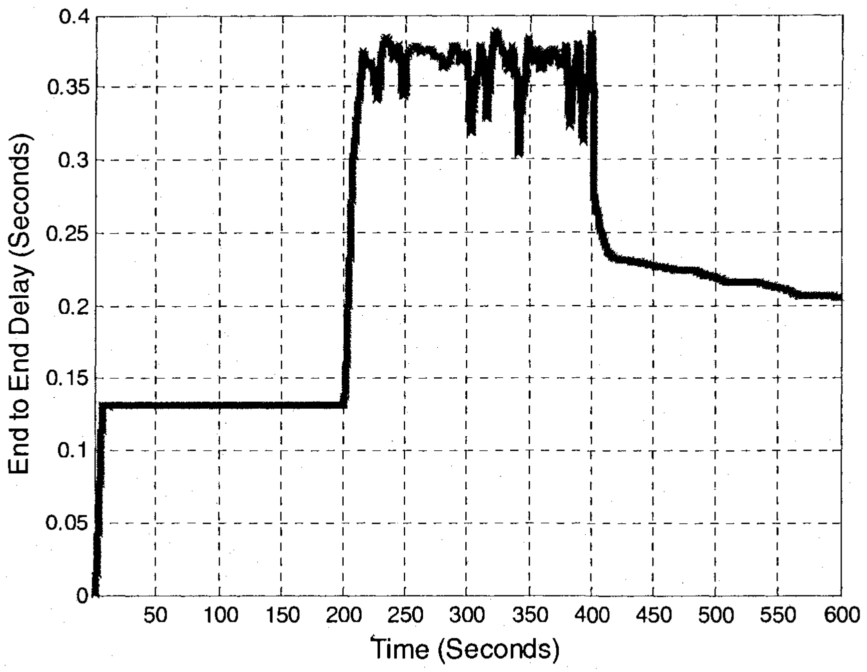

Figure E.4 End to End Delay of Flow#61 (Buffer Size =1000 Packets, XCP)

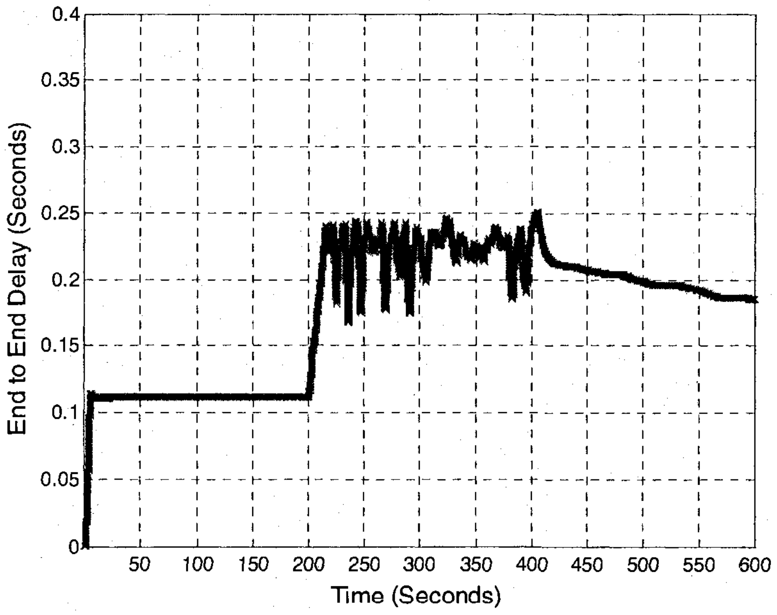

Figure E.5 End to End Delay of Flow#81 (Buffer Size =1000 Packets, XCP)

Figure E.1 to Figure E.5 show the end to end delay of the same 5 flows with the buffer size set as 1000 packets. During time 0-200s, the end to end delays are almost the same as the expected values. During time 200-400s, because of the huge queuing delay in the system in this experiment, the end to end delays oscillate heavily above the expected values (notice the end to end delay equals the propagation delay plus the queuing delay and the processing delay). During time 400-600s the congestion disappears, the system drains the huge queue that left in the previous time. This makes the curves of the end to end delay in these figures have a long trail.

Table E.4 Expected and Measured Delay (Buffer Size =1000 Packets, XCP)

Simulation Time	**0-200s**		**200-400s**		**400-600s**	
Delay	Expected	Measured	Expected	Measured	Expected	Measured
Flow#1(s)	0.1100	0.1117	0.1700	0.2396	0.1400	0.2102
Flow#21(s)	0.1300	0.1316	0.1900	0.2583	0.2100	0.2801
Flow#41(s)	0.1500	0.1516	0.2500	0.3192	0.1100	0.1834
Flow#61(s)	0.1300	0.1316	0.2900	0.361	0.1500	0.2234
Flow#81(s)	0.1100	0.1117	0.1500	0.218	0.1300	0.2003

Table E.4 shows the end to end delay of the same 5 flows with the buffer size set to 1000 packets. One can see that during time 0-200s, the end to end delays are almost the same as the expected values. During time 200-400s, because the huge queuing delays, the end to end delays are much longer than the expected one. During time 400-600s, the congestion disappears. However, because of the previous huge queue and the router need some time to drain this queue, each flow has a much longer end to end delay.