Network Traffic Control
Based on Modern Control Techniques:
Fuzzy Logic and Network Utility Maximization

Jungang Liu

Thesis submitted to the
Faculty of Postdoctoral and Graduate Studies
in partial fulfillment of the requirements
for the Doctorate in Philosophy degree
in Electrical and Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

This thesis presents two modern control methods to address the Internet traffic congestion control issues. They are based on a distributed traffic management framework for the fast-growing Internet traffic in which routers are deployed with intelligent or optimal data rate controllers to tackle the traffic mass.

The first one is called the IntelRate (Intelligent Rate) controller using the fuzzy logic theory. Unlike other explicit traffic control protocols that have to estimate network parameters (e.g., link latency, bottleneck bandwidth, packet loss rate, or the number of flows), our fuzzy-logic-based explicit controller can measure the router queue size directly. Hence it avoids various potential performance problems arising from parameter estimations while reducing much computation and memory consumption in the routers. The communication QoS (Quality of Service) is assured by the good performances of our scheme such as max-min fairness, low queueing delay and good robustness to network dynamics. Using the Lyapunov's Direct Method, this controller is proved to be globally asymptotically stable.

The other one is called the OFEX (Optimal and Fully EXplicit) controller using convex optimization. This new scheme is able to provide not only optimal bandwidth allocation but also fully explicit congestion signal to sources. It uses the congestion signal from the most congested link, instead of the cumulative signal from a flow path. In this way, it overcomes the drawback of the relatively explicit controllers that bias the multi-bottlenecked users, and significantly improves their convergence speed and throughput performance. Furthermore, the OFEX controller design considers a dynamic model by proposing a remedial measure against the unpredictable bandwidth changes in contention-based multi-access networks (such as shared Ethernet or IEEE 802.11). When compared with the former works/controllers, such a remedy also effectively reduces the instantaneous queue size in a router, and thus significantly improving the queueing delay and packet loss performance.

Finally, the applications of these two controllers on wireless local area networks have been investigated. Their design guidelines/limits are also provided based on our experiences.

Acknowledgments

I would like to extend my greatest gratitude to Prof. Oliver W.W. Yang, my thesis supervisor, for his invaluable and tireless guidance throughout my PhD career at University of Ottawa. His great intuition, profound knowledge, remarkable research expertise and inspiring comments made the work challenging while hopeful and fruitful, and enabled me to succeed in my study.

Many thanks go to the members of the CCNR (Computer Communication Network Research) laboratory for their help and unforgettably nice time being together.

I owe a great debt of thanks to my family. My wife, Xin, encouraged me to pursue a PhD, and I thank her for the constant encouragement, love, support and cooking throughout the years. I thank my daughter, Emily, who could always reenergize me with her sweet smiles and interesting games when I came across difficulties doing my PhD. Lastly, I thank my parents and brother for their warm love.

Finally, I gratefully acknowledge the financial support from the following scholarships or programs: Admission Scholarship (from Faculty of Engineering and Faculty of Graduate and Postdoctoral Studies) of University of Ottawa, OGS (the Ontario Graduate Scholarship), the Research Discovery Grant and Accelerated Grant from NSERC (Natural Sciences and Engineering Research Council of Canada).

Table of Contents

List of Figures

List of Tables

Table of Acronyms and Abbreviations

		Section of 1st appearance
ACK	ACKnowledgment	2.1
AP	Access Point	5.1
API-RCP	Adaptive Proportional-Integral Rate Control Protocol	1.1
AQM	Active Queue Management	1.1
ATM	Asynchronous Transfer Mode	1.1
BDP	Bandwidth Delay Product	1.2.1
BG	Big	3.1.1
BIBO	Bounded Input and Bounded Output	1.7
CA	Center Average	Appendix F.4
COG	Center of Gravity	3.1.1
EB	Extremely Big	3.1.1
ECN	Explicit Congestion Notification	1.2.1
ES	Extremely Small	3.1.1
FIFO	First-In-First-Out	2.3
FLC	Fuzzy Logic Control	1.4
FS	Fuzzy Smoother	3.1.1
ftp	File Transmission Protocol	3.3.1.1
IntelRate	Intelligent Rate Controller	1.8
IP	Internet Protocol	1.4.1
IQSize	Instantaneous Queue Size	3.1.1
KKT	Karush–Kuhn–Tucker	1.3
LDM	Lyapunov's Direct Method	1.4.1
LTI	Linear Time Invariant	1.3
LVs	Linguistic Values	3.1.1
MAC	Media Access Control	1.4.1
MC	Max Criterion	Appendix F.4
MD	Medium	3.1.1
MFs	Membership Functions	1.4.1
MIMO	Multiple-Input-Multiple-Output	1.3
MPEG	(Moving Picture Experts Group)	1.4.1
MPLS	(Multiprotocol Label Switching)	1.7
MX	Maximum	3.1.1
NL	Negative Large	3.1.1
NM	Negative Medium	3.1.1
NS	Negative Small	3.1.1
NUM	Network Utility Maximization	1.2.2
NV	Negative Very Large	3.1.1
OFEX	Optimal and Fully EXplicit Controller	1.8
PI	Proportional-Integral	1.4.1
PID	Proportional-Integral-Differential	1.4.1
PL	Positive Large	3.1.1
PM	Positive Medium	3.1.1
PS	Positive Small	3.1.1

PV	Positive Very Large	3.1.1
QFCP	Quick Flow Completion Protocol	1.2.2
QoS	Quality of Service	1.2.2
RCP	Rate Control Protocol	1.1
RED	Random Early Detection	1.1
RM	Revenue Management	1.4.2
RSSI	Received Signal Strength Indicator	5.1
RTPD	Round Trip Propagation Delay	2.2
RTT	Round Trip Time	1.2.1
SISO	Single-Input-Single-Output	1.3
SM	Small	3.1.1
SNR	Signal-Noise-Ratio	5.1
TBO	Target Buffer Occupancy	1.2.2
TCP	Transmission Control Protocol	1.1
TISO	Two-Input Single-Output	3.1.1
UDP	User Datagram Protocol	3.2.1
UOD	Universe of Discourse	3.1.1
VB	Very Big	3.1.1
VS	Very Small	3.1.1
WLAN	Wireless Local Area Network	3.4
XCP	eXplicit Control Protocol	1.1

Table of Symbols

		Section of 1st appearance
A	Edge value of MFs of $e(t)$, beyond which the MFs saturate	3.1.1
B	Buffer capacity	3.1.1
C	Edge value of MFs of $g(t)$, beyond which the MFs saturate	3.1.1
C'	Estimated bandwidth of a link	E.1.1
Cp	Incoming traffic rate or outgoing traffic rate in packets/second	E.1.1
D	Outermost edge value of MFs of $u(t)$	3.1.1
H	Strictly positive definite diagonal matrix	I.2
K_1	The synonym of $\overline{\lambda}_l$	4.4
K_2	The synonym of $\delta(c_l - \gamma\overline{q})$	4.4
L	Set of network links	2.1
Lc	Lipschitz constant	4.3
N	Number of LVs	3.1.1
N'	Estimated number of flows	E.1.2
R	Packet size	E.1.1
S	Set of Internet sources	2.1
$S_l(t)$	Subset of S	2.1
S_j	Area of a triangle	3.1.1
T	Control Interval	E.1.1
P	Estimated frequency of hits	E.1.2
$\overline{P}_i^j$	LVs assumed by input linguistic variables $\widetilde{e}(t)$ or $\widetilde{g}(t)$	3.1.1
$U(\cdot)$	Function used in the social welfare function for the OFEX controller	4.1
$\overline{U}^j$	Linguistic values assumed by output linguistic variable $\widetilde{u}(t)$	3.1.1
$V(x)$	Lyapunov function	3.2.2
W	Aggregate incoming traffic in bits.	4.2.3
Xi	Clock cycles, i=1,2,3	E.2.1
U_l	Feasible domain of source data rate	4.1
X	Total operations in one control interval T	E.2.1
Yin	Incoming traffic rate in packets/second	E.1.1
$Yout$	Outgoing traffic rate in packets/second	E.1.1
c_j	Bottom centroid of an MF	3.1.1
c_l	Nominal bandwidth (or say, capacity) of link l	4.1
$c(t)$	Time-variable bandwidth (or say, capacity) of a link	2.2
d	Operation cycle of the IntelRate controller	3.1.3
$e(t)$	Queue error between TBO q_0 and IQSize $q(t)$ of AQM router.	3.1.1
$\widetilde{e}(t)$	Linguistic variable to describe input $e(t)$	3.1.1
$f(.)$	Representation of the IntelRate controller	3.1.1
$f_i(.)$	Function of MFs, i=1,2,3,	3.1.1
$g(t)$	Integration of $e(t)$, the other input of the IntelRate controller	3.1.1
$\widetilde{g}(t)$	Linguistic variable to describe input $g(t)$	3.1.1
$h(.)$	Output change function	3.2.2

k	Number of rules for a fuzzy logic controller	3.1.1
l	A instance in set L	2.1
m	Width Parameter of MFs	3.1.1
$\hat{m}$	A parameter for strong convexity of the OFEX controller	4.3
s	A instance in set S	2.1
p_i	crisp input, i=1,2	3.1.1
q	queue size	E.1.1
q_{ave}	Weighted moving average of queue size	E.1.1
q_0	TBO (Target Buffer Occupancy) of AQM router	3.1.1
q_s	Steady state queue size	3.2.1
$\overline{q}$	Local equilibrium point of queue size in the OFEX controller	4.4
$q(t)$	Instantaneous queue size (IQSize)	1.7.2
u_s or $u_s(t)$	Controller output of source s, i.e., the allowed sending rate of source s	2.2
u or $u(t)$	Generalized symbol representing u_s or $u_s(t)$ of any individual flow	3.1
$\tilde{u}(t)$	Linguistic variable to describe the output $u(t)$	3.1.1
$\hat{u}_i$	Value recorded in *Req_rate* of a packet	6.1.1.3
$u_i'(t)$	Current source sending rate	2.2
$v(t)$	Aggregate uncontrolled incoming rate to router queue	2.2
w	Weight number	E.1.1
$w(t)$	The synonym of $-\gamma q(t)$	4.5.1
x_i	Lyapunov variables i=1,2	3.2.2
$\overline{x}_s^l$	Local equilibrium of source rate in the OFEX controller	4.4
y_s	Steady state output	3.2.1
$y(t)$	Aggregate controller output or incoming traffic rate into queue	2.2
$\triangle q$	Deviation of $q(t)$ from q_s in the IntelRate controller	3.2.2
$\Delta x_s^l(t)$	Deviation of u_s from $\overline{x}_s^l$ in the OFEX controller	4.4
$\triangle y$	Deviation of $y(t)$ from y_s in the IntelRate controller	3.2.2
$\Delta\lambda_l(t)$	Deviation of λ_l from $\overline{\lambda}_l$ in the OFEX controller	4.4
Θ	Number of packets passing through a router in one interval	E.1.1
Λ_{lm}	Minimum unit bandwidth price of link l	4.3
Λ_{lM}	Maximum unit bandwidth price of link l	4.3
Π	Operations to estimate bandwidth	E.1.1
$\Pi^*(w)$	Optimal revenue of the disturbed system in the OFEX controller	4.5.1
$\Omega(\cdot)$	Dual function	4.2
α	Bandwidth estimation parameter	E.1.1
γ	Threshold parameter for $q(t)$ in the OFEX controller	4.1
δ	Step size of the OFEX controller algorithm	4.2.2
ε	Any small positive number	3.2.2
λ_l	Lagrange Multiplier (congestion cost in link l)	4.2.1
$\overline{\lambda}_l$	Local equilibrium of bandwidth price in OFEX controller	4.4
ψ_S	Weight (or say, price Source s would like to pay)	4.1
$\eta(\cdot,\cdot)$	Lagrangian of the primal problem	4.2.1

$\mu_{P_i^j}$	Input fuzzy set of the IntelRate controller	3.1.1
μ_{U^j}	Output fuzzy set of the IntelRate controller	3.1.1
$\mu_{XY}(u(t))$	Certainty degrees of MFs	3.1.1
τ_{fi1}	Time delay of a packet from source *i* to the router	2.2
τ_{fi2}	Time delay of the packet of source *i* from the router to its destination	2.2
τ_{bi}	Feedback delay from destination *i* back to its source	2.2
τ_i	Round trip time (RTT) of path *i* (generalized as τ)	2.2
ω	Angular speed in frequency domain	4.4

Chapter 1
Introduction

Since the infancy of the Internet, the network traffic control has been employed to prevent the network from severe congestion which could degrade the Internet performance such as in packet loss, low-throughput, bandwidth under-utilization, excessive transmission delay, and even network collapse. The Internet bandwidth has been ever-growing, and in the meantime the Internet applications are also becoming more traffic-intensive, e.g., the wide-spread file transfers and video steaming nowadays. As an effective approach to manage the network traffic, Internet congestion control is still a hot topic in both research communities and industries.

In this chapter, different classification methods for the existing congestion control protocols are first reviewed along with some of their issues that are then explained and discussed in the subsequent sections for motivation, objectives, methodology. Contributions of the thesis are summarized before the chapter is concluded.

1.1 Classification of Congestion Control Protocols

The existing Internet congestion control protocols can generally be classified in the following three methods.

1) A bandwidth-allocation method in which the control algorithms are classified into window-based and rate-based [Hong06]. In window-based control, a source (or say, end system) transmits its data at most with a congestion window size which is halved whenever congestion is detected. This is the basis of the TCP (Transmission Control Protocol) that uses AIMD (Additive-Increase Multiplicative-Decrease) to exercise congestion control [Jaco88]. In rate-based control, a source transmits its data by adjusting its sending rate according to the traffic intensity of network such as some ATM (Asynchronous Transfer Mode) congestion control protocols [ChCh94], the RCP (Rate Control Protocol) [DuMc06], and the API-RCP (Adaptive PI-Rate Control Protocol) [HoYa07]. The basic difference between the two is that the window-based reactively acts on congestion, i.e., it only takes actions after the congestion happens, whereas the rate-based operates proactively on congestion so as to prevent the congestion from happening.

2) A location-based method in which the control algorithms are classified into source-based and cooperation-based [Chen06]. In source-based control, the control algorithm only resides in end systems, e.g., TCP Tahoe [Jaco88], TCP Reno [Jaco90], TCP New Reno [FlHe04], TCP Vegas [BrPe95], FAST TCP [JiWe05, NiCh06]. In cooperation-based control, the control algorithm resides in both the end systems and routers, e.g., RED (Random Early Detection) +TCP [FlJa93], BLUE+TCP [FeSh02], XCP (eXplicit Control Protocol) [KaHa02] and RCP [DuMc06]. The basic difference between the source-based and the cooperation-based is that the routers in the latter one proactively provide traffic intensity information to sources while the former one has to infer the traffic intensity of the network via packet loss and/or delay.

3) A congestion-signal method in which the control algorithms are classified into implicit and explicit [Kata03, KeRa10]. The implicit control broadly corresponds to the congestion control mechanisms where noisy feedback from the network is averaged at the sources using an increase/decrease rule and generalizing those approaches of various TCP and TCP+AQM (Active Queue Management) algorithms [KeRa10]. The explicit congestion control protocols broadly correspond to the congestion control mechanisms where the routers, according to some averaging methods, feeds explicit traffic intensity information back to sources for adjusting the sending rate generalizing those approaches of XCP [KaHa02] and API-RCP [HoYa07].

In the above, there is no advantage of one classification method over another. Instead, each of these classification methods describes an important characteristic of a control protocol. That is to say, every control protocol at least has three characteristics. For example, TCP can be described as a window-based, source-based and implicit congestion control protocol; XCP can be described as a rate-based, cooperation-based and explicit congestion control protocol.

Since the issues to address (to see later on) in this thesis are more related to the congestion signals, the literature review below will be based on the third classification method.

1.2 Implicit versus Explicit Congestion Control

The implicit and explicit congestion control protocols are also separately called primal

algorithms and dual algorithms from the perspective of convex optimization theory [KeRa10].

1.2.1 Implicit Congestion Control Protocols

In this type of algorithms such as TCP, since network does not actively provide congestion information to sources, the sources have to react to congestion by inference based on some events such as packet loss and/or delay variation.

The remarkable success of TCP [Jaco88, Jaco90] for congestion control is attributed to its important feature that a source adjusts its congestion window size based on the notion that the network is a "black box" [RaFl99], i.e. all it relies on is just the inferred packet loss signal, upon which TCP sources multiplicatively decrease their congestion window.

As an implicit congestion control protocols, TCP encounters various performance problems (e.g., bandwidth utilization, fairness and stability) in today's Internet and particularly when the Internet BDP (Bandwidth-Delay Product) continues to increase.

TCP variants have been proposed to overcome the shortcomings of TCP Reno. For examples, FAST TCP [WeJi07] uses queueing delay as the primary measurement of congestion to adjust the congestion window. Although this scheme reduces packet-level oscillations [JiWe05], it may also artificially introduce queueing at the bottleneck router [QaZn09]. HSTCP (HighSpeed TCP) [Floy03] proposes a large congestion window to improve the flow throughput [Floy03], but it is still packet-loss based. TCP Vegas [BrMa94] estimates the available bandwidth to improve the utilization and to reduce the packet loss, but investigation [HeBo00] found that its congestion avoidance mechanism has fairness problems even if all competing connections operate with the same round trip time. STCP (Scalable TCP) [TeSz05] improves the link utilization in high BDP networks but unfriendly pushes out other existing regular TCP flows [TeSz05]. In summary, although these TCP variants indeed show the improved performance to TCP Reno they still suffer some potential problems in one aspect or another.

In order to make TCP (as an implicit protocol) behave more efficiently and overcome the defect of Drop-Tail mechanism in the router, AQM algorithms were proposed. For examples, the RED algorithm [FlJa93] probabilistically discards/marks the packets by observing the average queue size so that the queue can operate in a reasonably size [FlJa93]. Instead of

dropping/marking packets, the ECN (Explicit Congestion Notification) protocol [RaFl99] explicitly feeds back congestion signal with ECT (ECN-Capable Transport) code-points to sources. The advantage of both the RED and the ECN methods is that they only use the queue size to decide if a packet should be dropped or marked in that they do not need to estimate link states such as bandwidth capacity. Nevertheless, they have some other problems, for examples, TCP+ RED inevitably causes severe oscillations in the source throughput as the link bandwidth or latency increases [LoPa02], while TCP+ECN still inherits the bias against connections with long RTT (Round Trip Time) as well as the unfairness towards new connections [LeMo01]. These have also been widely investigated along with or without AQM schemes by [KaHa02, FeVa03, HaBe07, HoYa10], some of which are explicit congestion control protocols to be discussed in the next section.

1.2.2 Explicit Congestion Control Protocols

In view of the types of the explicit signals fed back from the networks, the explicit congestion control protocols can be further classified into relatively explicit protocols and fully explicit protocols.

The relatively explicit protocols refer to the congestion control mechanisms where sources adjust their sending rate according to cumulative link information, i.e., congestion signal (e.g., link prices) that is the summation over all links on a flow path. The existing NUM (Network Utilization Maximum)-based traffic control algorithms (e.g., [KeMa98, LoLa99, AtLo00, LiSh06, MaMa08]) fall under this category, in which the sources have an end-user control equation or a demand function to calculate in what sending rate they can transmit data according to the cumulative congestion signal.

The fully explicit algorithms refer to congestion control protocols where the network explicitly feeds back the traffic intensity signal (e.g., the allowed sending rate or rate changes) from the most congested link, and the sources must follow such signal to send their data out. Examples of fully explicit congestion control protocols include XCP [KaHa02], RCP [DuMc06], API-RCP [HoYa07], JetMax [ZhLe06], MaxNet [WyAn03] and their enhancements in wireless networks such as QFCP (Quick Flow Control Protocol) [PuHa07, PuHa08], Blind, ErrorS and MAC [AbAr11].

Unlike the implicit congestion control protocols that have to infer from indirect

information whether the network is congested or not, the explicit congestion control protocols: 1) can explicitly broadcast or feed the network congestion or non-congestion information back to sources (e.g., in a dedicated field of packet header) which adjust their sending rate accordingly; 2) do not need to maintain per-flow state; 3) can make their sending rates to converge to their optimum in order to achieve an optimization objective [ZhLe06]. In doing so, the explicit congestion control protocols "may allow the design of a fair, stable, low loss, low delay and high utilization network." [KeRa10]. With the explicit congestion controllers, the sources can transmit data more rationally and precisely (e.g., because they feed the required sending rate to sources) as per the network traffic intensity, and hence link bandwidth can be utilized more efficiently among different flows and the network is more stably controlled.

Most of these explicit congestion control protocols use a fixed value for the bottleneck bandwidth in order to compute the allowed source sending rate or link price. Recent studies show that variations of link bandwidth (e.g., in link sharing networks or wireless networks) may easily occur and can cause significant fairness and stability problems [ZhAh05, ZhHe05, AbRi06]. The latest protocols on wireless applications such as QFCP (Quick Flow Control Protocol) [PuHa08] and the three protocols of Blind, ErrorS and MAC [AbAr11] have introduced bandwidth estimation mechanism into their controllers. They have shown better performance such as in link utilization and throughput fairness. However, they have the fundamental problem of inaccurate estimation resulting in performance degradation. In addition, their bandwidth probing speed may be too slow when the bandwidth fluctuates drastically. Also, they cannot keep the queue size stable, which in turn affects the stability of their source sending rates.

Some explicit protocols appear to compute the sending rates based solely on the queue size, but in fact they still need to estimate the number of active flows in a router, and this consumes CPU and memory resources. Examples are the rate-based controllers [BeMe93, HoYa07, HuXi09] for packet switching networks and the ER (Explicit Rate) allocation algorithm [ChLe01] for ATM networks. For the API-RCP controller [HoYa07], both the original method (a truncated network model) and the improved method [HoYa10] face a memory problem when dealing with many flows (that numbers in millions) arriving to a core router every hour [RiYe08]. Furthermore, the original method of API-RCP requires extensive

CPU computations. In some other controllers (e.g., [ChLe01]), the TBO (Target Buffer Occupancy) is designed to be as high as three times of the BDP, which can cause large queueing delay and thus degrading network performance. Historically, the ER allocation algorithms in ATM networks need to evaluate the link bandwidth and/or the numbers of active VCs (Virtual Circuits), and therefore share the same problems as XCP or API-RCP. Some others (e.g.,[Robe94]) adjust the source sending rates in binary-feedback switches or explicit feedback switches according to a few queue thresholds, which may cause unfairness as well as high cell loss rate [Jain96, LeHo00].

From the perspective of network and service management, the aforementioned explicit congestion control protocols either have QoS (Quality of Service) problems in terms of packet loss, delay, throughput instability, fairness and etc or consume too much computational resources of routers.

1.3 Modern Control versus Classical Control

The works reviewed in the previous section have some common features. Most of them have adopted or can be represented by a control system model of transfer functions in the frequency domain [Nise11]. Then the system behaviours such as transient responses and stability can be investigated with some standard classical control theories. For examples, the explicit congestion control protocols (e.g., XCP, RCP) mostly apply techniques such as the Nyquist stability criterion to do their stability analysis. API-RCP uses the phase margin method to achieve the stability. Many of these analysis techniques are well known in other traditional areas like electric circuit systems, mechanical systems with well-established theorems. Therefore, the term “classical control” is used here to loosely encompass this type of congestion controllers. In a nutshell, classical control is a frequency domain technique which is based on converting differential equation of a system to a transfer function, thus generating a mathematical model of the system that algebraically relates a representation of the output to a representation of the input [Nise11].

Due to the historical development, there are limitations to the controllers using classical control theory. For example, a system has to be or approximated by a LTI (Linear Time-Invariant) system [Ogat02]. Although there are works in the time-domain, they usually rely on the frequency domain theory to support their argument. Besides, the system has to be

modeled or decomposed as SISO (Single-Input-Single-Output) systems in order not to complicate the analysis such as dependency among different states. Furthermore, the system initial states are usually assumed to be zeros so that the modeling and the analysis could proceed, e.g., to model a system with the transfer function technique.

In contrast, “modern control” is defined/introduced to be essentially a time-domain approach applicable to both SISO and MIMO (Multiple-Input-Multiple-Output) systems. They can be linear or nonlinear as well as time-invariant or time-variant [Ogat02]. Therefore, modern control provides a unified method for modeling, analyzing and designing a wide range of systems including those with non-zero initial system conditions, backlash, saturation and dead zone [Nise11].

The applications of modern control approaches to analyze congestion control include mathematical generalizations to TCP and some of its variants with optimization approaches (e.g., [LoPa02]), primal-dual methods (e.g., some literatures to review in Section 1.4.2 below) or game theory (e.g. [AlBa02, FeHu06]), RED-like controller designs with fuzzy control (e.g., those literatures to review in Section 1.4.1 below) or H-infinity (e.g., [QuOz04, ChYa05]), protocol design with economic methods (e.g., [DuNi03, Stid04]) and so on. Some modern control methodology actually has a long history such as the Lyapunov’s stability criterion [RoHa77], and the KKT (Karush–Kuhn–Tucker) conditions [Bert99], and the differential equations of the control system can be expressed in a matrix form with the states, inputs and outputs. It is noteworthy that the terminology “modern control” is also used by different people such as [Kozi08]. We shall use it here to loosely encompass controllers with this kind of approach in order to highlight the latest control methodology in dealing with Internet congestion control. Without further emphasis, we shall use this terminology from now on.

1.4 Two Modern Control Methods

In this section, the congestion control literatures are reviewed based on two of the main modern control (as defined above) approaches that are of interest to us, i.e., FLC (Fuzzy Logic Control) and NUM (Network Utility Maximization) approach. At the end, the fundamental queueing models that the two modern control approaches are based on to design the congestion controllers are reviewed. Please note that the "modern control" wording is

chosen here to encompass the two control approaches in the thesis for brevity.

1.4.1 Congestion Control Using FLC

There have been many FLC applications in network congestion control since 1990s. These FLC schemes have shown good performances, while they also have various shortcomings.

In the early stage, FLC was used to do the rate control in ATM networks, e.g., [ChCh94, PiSe97, Qiu97, HaPu99, LaPh99], to improve the QoS. These control algorithms are explicit in nature, but they have various shortcomings including high cell loss rate, queue size fluctuations, poor network latency, stability and low utilization [ChCh94]. Later, FLC was used in the RED algorithm in TCP/IP (Internet Protocol) networks, e.g., [ChPi03, AoNa04, ChPi06, NyDa06, GuYa07, NyDa07, SuZu07], to reduce packet loss rate and improve utilization. However, they are still providing implicit or imprecise congestion signals, and therefore cannot overcome the throughput fluctuations of the TCP sources [KaHa02]. In addition, these existing methods do not depend on TBO to adjust the allowed sending rate. Instead, they depend on the absolute queue length. Such kind of design is prone to cause queue size fluctuations (as they don't have closed-loop control on queue size) as well as packet loss under heavy traffic conditions, and so potentially suffering from problems such as latency jitter, network instability and link under-utilization. A review of the application of the FLC for congestion control in TCP/IP networks can be found and also the shortcomings and challenges of the designed controllers are discussed [ChPi09], and some of the relevant ones have also been reviewed in this chapter.

To adapt the media encoding rate is another application of FLC to mitigate the congestion. For examples, a controller is proposed for the all-IP networks, where the encoding rate is calculated based on round trip delay [JaFl09]. The simulation results show the controller has good ability to cope with the input uncertainty and packet loss. However, this kind of delay-dependent protocols tends to become unstable under certain circumstances [ZhLe06]. On the other hand, congestion is not the only factor contributing to the network delay, e.g. latency may be caused by MAC (Media Access Control) scheduling or route failure. A neural-fuzzy controller is proposed to adjust the MPEG (Moving Picture Experts Group) video streaming rate for the Bluetooth channel [KaMe06], in which neural network is used to train the membership functions of fuzzy control. Although this method reduces the

number of dropped data packets, and produces better and stable video image quality, the complexity involved in the neural network training makes the real-time implementation very challenging. Finally, this type of streaming data control algorithms reside in the application layer and is dedicated to do the video encoding rate adaptation only.

The fuzzy logic congestion control applications are also used in wireless sensor networks such as [MuYu07], where fuzzy logic controller is used to estimate the congestion level by monitoring the incoming traffic into the queue in sensor nodes. It shows good performance such as in bandwidth utilization or energy consumption. However, just as the abovementioned ATM fuzzy logic controllers, they also depend on absolute queue size and not on TBO. In addition, it does not provide the explicit information to signal the network congestion level and thus sharing the similar shortcomings with TCP.

Stability analysis of FLC (Fuzzy Logic Control) systems has been around since 1990s covering areas such as mass-spring-damper systems (e.g., [LaLe99, LiLi12]), electrical power systems (e.g., [SiTe01, GuAd04]), mechanical and motor control systems (e.g., [HaSc03, YuGr08]), robotic systems (e.g., [SaKe04, MeSa09]) and others. These studies have used different models including the Takagi-Sugeno model [e.g. Hanh08, WaLi09, WuSh12], the sliding-mode control model [e.g., ZhSh10] and the nonlinear strict-feedback model [e.g., ZhLu11]. These models can be applied to either the continuous-time or the discrete-time systems.

Different approaches have been used in the stability analysis of FLC systems. Lyaponov's stability criteria are the most frequently used approaches (e.g., [LaLe99, GuAd04, YuGr08, ZhSh10, WuSu11, LiLi12, WuSh12]) once the state equations of the plant are known. Other approaches include the describing function (e.g., [SiTe01]), circle criterion (e.g., [HaSc03]), Popov criterion and hyperstability criterion [MiKl06]. Each approach has its limitation, and their applicability depends on the system structure and the type of information describing the plant. For example, the describing function technique can be applied to Single (or Multiple)-Input-Single-Output fuzzy controllers, but is prohibitive for more than three inputs in the fuzzy controller [PaYu98] and for the systems that cannot be separated into a linear and a nonlinear part [MiKl06].

There are some FLC stability studies related to data network congestion control [AoNa04, GuYa07, JiCh10]. The RED-like controller design (e.g., [GuYa07]) is actually

mixing FLC with the sliding mode control. Its proof of stability is based on linear control theory (i.e., Nyquist Criterion) by linearizing the plant model around the equilibrium point. LDM (Lyapunov's Direct Method) was used to obtain the system stability conditions of an AQM control system (e.g., [AoNa04]), but the stability conditions cannot be guaranteed. It is also not clear whether those conditions are eventually met with just three rules designed for their controller because a small number of rules can have many potential performance problems such as big overshoot, long settling time or instability due to the high dynamics of the Internet traffic. At least a 300% of overshoot and more than 20 seconds of settling time have been observed in their simulation results. Furthermore, the stability analysis in these existing works is only for window-based AQM congestion control protocols.

A fuzzy-logic-based controller is inherently nonlinear, and so far there is very limited work on its mathematical models. A very simple fuzzy logic controller (with two MFs (Membership Functions) for the inputs and three for the output) has been analytically modeled in [YiSi90, Ying93]. It has been proven to be a nonlinear controller which has its proportional-gain and integral-gain changing according to the inputs. The discussion of the two inputs is later extended to three inputs in [HaYi04], and is shown to be structurally equivalent to a nonlinear PID (Proportional-Integral-Differential) controller with time-varying gains under some given conditions. The equivalence of the fuzzy logic controller to a nonlinear PI (Proportional-Integral) controller is also investigated in [ChWa99] but with another type of membership function.

1.4.2 Congestion Control Using NUM

Since the seminal work laid by Kelly [KeMa98], the NUM has emerged as a promising framework in the network congestion control protocol design, and received wide investigation. Many literatures (e.g., [LoLa99, QiMa03, Hark05, ZhWu05, LiSh06, KeRa10, PrVa11]) have been published ever since. There are also books or tutorials on the theory of NUM, e.g., [Srik04, PaCh06].

A primal and a dual rate control protocol based on utility functions have been proposed [KeMa98], in which each end user has a utility $U_s(u_s)$ (by assuming the sending rate of user s is u_s). The objective of their problem is to maximize the aggregate utility of all end users based on a series of network capacity constraints. Each user implements an operation to

additively increase and multiplicatively decrease its data rate based on feedback information cumulated from the links [MoWa00], and proportional fairness is achieved among the users. A method similar to [KeMa98] was used to solve the utility-function-based problem using different marking implementations of the algorithm [LoLa99]. They proved the convergence of the algorithm in both synchronous and asynchronous conditions. Specifically, each link updates its congestion signal based on the received rates. Then, the relatively explicit congestion signals from all links along the path are summed up and fed back to the sources which then use a demand function to compute their new sending rates.

Based on the implicit congestion information as in TCP, a pricing mechanism [LaAn02] modeled with NUM was introduced to not only guarantee the fairness but also maximize the overall utility of the end users. The algorithms can adjust its congestion prices (i.e., the relatively explicit congestion signal) that the end users will use to regulate their window size so that the system optimum is achieved. Further, the equilibrium prices are such that the system optimum can achieve a weighted proportional fairness.

Contrast to solving the congestion control problem for single-path routing with NUM, the utility function method for multi-path routing networks has been applied [LiSh06] where a distributed algorithm was developed and analyzed to solve a NUM problem. Specifically, in their algorithm, a "dual update algorithm" computes the relatively explicit congestion signals attached to the dual variables, and a "primal update algorithm" computes the source sending rate for different paths given the congestion signal.

To understand the connection between the existing network congestion control algorithms and the optimal control theory, the utility functional that can be maximized by the existing algorithms have been investigated [LaDo10], and the results showed that there exists meaningful utility functional whose maximization can lead to the celebrated primal, dual and primal-dual algorithms.

More NUM-based schemes can be found such as [AtLo00, KaSa01, MaMa08, LiEr10] and the references therein. However, our investigation reveals that they are all relatively explicit protocols and share the following issues.

1) The algorithms have a fairness problem, i.e., they "bias" the multi-bottlenecked flows in terms of their source sending rates (or throughput) and their convergence performance due to their source adjusting the sending rates according to the cumulative congestion signals. The

more bottlenecks a flow passes, the more "bias" it encounters. Such an issue is unavoidable in such kinds of duality models because of their network-wise proportional fairness mechanism [LoPa02]. As a matter of fact, such an issue has already been considered in TCP/IP networks, and the correcting measures have been discussed in [Floy91] accordingly. Our study for the explicit congestion control algorithms here is based on the same consideration. That is, there is no compelling reason that the long flows passing through multiple bottlenecks should suffer from low throughput and slow convergence [Floy91].

2) Most of the algorithms take on a fixed link bandwidth assumption that can make the contention-based networks ineffective (such as the multi-access Ethernet or IEEE 802.11 wireless network) because deterministic link capacity is usually unavailable in such networks. Furthermore, there is no easy way to predict the contentions/interferences that can happen anytime in the highly dynamic networks, and therefore estimations to link bandwidth are quite often inaccurate (due to conservative or over-estimation) or simply wrong; these have resulted in stability issues [ZhHe05, AbRi06]. There are some NUM-based works that have considered varying link capacity. For example, the stability/sensitivity issues were studied in a network with varying link capacity (e.g., [ZhWu05]), but no remedial measure against the bandwidth variations is proposed. Others (e.g., [Chia05, PaDe08, BeBa09]) have only modeled the link capacity as a function of the battery power of a node in wireless ad hoc networks which are not applicable to the wired networks.

3) The method to match the incoming traffic rate to the link capacity in these algorithms may incur a long queueing delay or even packet loss after an overload. For example, a sudden influx of data can cause the incoming traffic rate to become greater than the link capacity. If this overload persists, the queue will quickly build up and even overflow the buffer. The queue can never dissipate even if the incoming traffic just matches (remains equal but not less than) the link capacity after. It has been noticed that the traditional controllers XCP [KaHa02] and RCP [DuMc06] (both of which are based on the classical LTI control models, not NUM-based) relax such an issue by considering persistent or instantaneous queue size in their models under fixed link capacity. However, the limits on their design parameters α and β affect their stability performance when there are big traffic or bandwidth variations.

An observation has also been made that many works so far consider bandwidth less than 150 Mbps which is not representative of the high-speed network nowadays, e.g., [LaAn02,

AnQi05, ZhWu05, LiSh06]. The works that consider the high link bandwidth like [ChLi09] that are not fully explicit.

Finally, for the relatively explicit protocols, the Lagrange multiplier used in their algorithms is often economically interpreted with RM (Revenue Management) as a total price that a source has to pay for the network [BoDi03].

1.4.3 Queueing Model

Queueing modeling has been a common technique to abstract the network operation in order to facilitate different analysis and protocol design. Queueing is simply the interaction of the arrivals and departures (services) of customers (as introduced in many queueing textbooks such as [Klei75, HaGa04, GrSh08]). The arrivals and departures can have arbitrary characteristics. Some models such as M/M/1 [e.g., Klei75] (where both the arrival and service have an exponential distribution) can have closed form expressions. For the general models such as G/G/1, only approximation formulas can be obtained [e.g., Klei75, GrSh08] under certain assumptions. For example, fluid approximation is used under a heavy traffic condition by replacing the discrete nature of queue (since queue can only take on integers) with smooth continuous functions of time [e.g. Klei76, HaGa04, Bhat08]. Further assumptions such as a diffusion process can be used to obtain closed form expression approximation [Klei76]. The manuscript [Meyn07] has attempted to use the fluid approximation to generalize many “network” processes, from which many interesting analysis and applications arise [ArKa58, Harr98, Huit99, DaHa04]. While the fluid model therein may be exact for some networks such as an electricity grid, some can only remain as an approximation as in the case of traffic queueing in Internet.

When a queueing system is complex such as the case with many subqueues in the system, the queueing performance can be analyzed as a queueing networks [FrAl09, Trib13]. It can also be approximated as a “super queue” where there is only a single queue while the effect of other queues and their interferences and interactions are included in a queueing model described by the relationship among the queue change rate, the traffic arrival rate and the departure rate [Kulk97, Meyn07]. For example, the general fluid "super queue" has been separately modelled with differential equations for an infinite buffer as well as with a finite buffer [Kulk97]. For the Internet traffic management, the "super queue" model has been

commonly used to model the queue in routers/switches for congestion control protocols. For examples, in the aforementioned XCP [KaHa02] and RCP [DuMc06] congestion control protocols, the queue in the TCP layer is modeled while the detailed operation arising from different interactions (such as queueing according to service priorities) inside the architecture of a router/switch along with anything below TCP can be regarded as packet processing time.

1.5 Motivation

Our motivation to design new explicit congestion controllers comes from the literature review in previous sections that have revealed different aspects and issues to be desired of a congestion controller. One aspect is for the controller to be robust[1] to the dynamics of network parameters such as link bandwidth. Another aspect is to have a small and stable queue size to obtain good performance on queueing delay jitters. The sources should also be made to send their data with a rate required by the most congested router on their paths to improve source throughput. More motivations/issues specific to each of the two controllers under study are the following.

1.5.1 The FLC-based Controller

From our review in Sections 1.2.2 and 1.4.1 for different protocols and their shortcomings, one can see that the fuzzy logic congestion control applications so far have received little considerations in the design of a rate-based explicit congestion controller suitable for the TCP/IP networks nowadays or for the next generation networks. In view of this, we are motivated to design a FLC-based distributed traffic management scheme for the current or the next generation networks, in which routers are deployed with explicit rate-based congestion controllers.

FLC has been considered for IC (Intelligence Control) [PaYu98]. It is a methodology used to design robust systems that can contend with the common adverse synthesizing factors such as system nonlinearity, parameter uncertainty, measurement and modeling imprecision [JaFl09]. In addition, fuzzy logic theory provides a convenient controller design

[1] A robust control system in general should maintain certain desired performance features despite the presence of significant system uncertainty such as parameter changes or disturbances [DoBi08]. Regarding the different mathematical definitions about robustness, the definitions of robustness used for the IntelRate controller and the OFEX controller can be found in Section 3.3.1.3 and the 2nd paragraph of Section 4.5, respectively.

approach based on expert knowledge which is close to human decision making, and readily helps engineers to model a complicated non-linear system. In fact, fuzzy logic control has been widely applied in industrial process control and showed extraordinary and mature control performance in accuracy, transient response, robustness and stability [HaKi92, ReJo95, HuRe96, Vane97, KiSa01]. All of these will facilitate the design of an explicit congestion controller for highly dynamic networks.

The stability of a fuzzy logic controller is also of particular importance and interest to our study. This is because the fuzzy-logic-based controller is inherently nonlinear, unlike other explicit congestion control protocols (such as XCP, RCP and API-RCP mentioned before) that are designed with the linear PI (Proportional-Integral) controller ideas based on established knowledge of classical control theory. The difficulty to prove the stability of a FLC system comes mainly from the rule base (which contains a logical quantification of heuristic information) that cannot be mathematically quantified. That is to say, it is very hard (if not impossible) to express a fuzzy-logic-based controller in an analytical form [MiKl06]. In addition, one cannot easily verify if the stability is guaranteed from the basic design process of a fuzzy logic controller. As such, we are motivated to prove the stability for our new FLC-based scheme.

Finally, we are also interested to investigate why the FLC-based controller is superior to the traditional PI controllers and how complex its computation is.

1.5.2 The NUM-based Controller

We are also motivated to propose a new NUM-based congestion control scheme in view of the shortcomings of the current NUM-based protocols, as reviewed in Section 1.2.2 and Section 1.4.2. This new scheme has to overcome the issues that the relatively explicit control protocols bias multi-bottlenecked flows, and can account for the network-wise proportional fairness issue which results in the bias. Besides, the fixed link capacity assumption in the current NUM-based protocols often leads to restrictive applications because there is no easy way to predict the contentions/interferences that can happen anytime in highly dynamic networks [ZhHe05]. This motivates us in the new scheme to relax the unpractical static assumption on link bandwidth and allow our controller to take on more general applications.

1.6 Objectives

The general objective of our work is to design rate-based, explicit congestion controllers with modern control approaches. Specifically, we would like to

1) investigate and design two types of rate-based explicit congestion controllers: one based on FLC and the other based on NUM.
2) provide a fairness mechanism in each controller to achieve an effective bandwidth allocation.
3) analyze some properties such as stability (or convergence) and computation complexity for each new scheme.
4) evaluate the controller performances and make comparisons with the existing controllers.

1.7 Methodology and Approaches

We shall design cooperation-based rate controllers so as to provide explicit congestion signals to the sources. Our design will start with the modeling of a router, and then apply modern control techniques to allocate link bandwidth to all the flows.

To model the traffic queueing operation in a router, the popular "super queue" model mentioned in Section 1.4.3 is used for the following two considerations: 1) it can abstract the complicated queueing process in a router/switch such as the muti-queue operation, and anything below the protocol layer that the congestion control resides in can be regarded as part of the packet processing time; 2) it is very helpful to analyze the performance of a system such as the system stability. More benefits of the "super queue" model will be discussed in Section 2.2 where we shall show how this arises from the fundamental queueing system model.

After their design stage, stability will be analyzed for each new control scheme. Of the different definitions of stability (e.g., [Nise11]), we shall use the BIBO (Bounded Input Bounded Output) definition for the queueing operation for each of the two control schemes. That is, the queue in the router is stable if every bounded input yields a bounded output. Note that our work will consider marginal stability (e.g., when the natural response remains severely oscillatory) as part of an unstable systems but this is something we want to demonstrate out controller are capable to minimize/reduce.

Computational complexity will be analyzed for the two new congestion controllers to

illustrate their deployment feasibility as well as to understand the convergence of their algorithms.

We shall use simulations to support the correctness of our algorithm and to evaluate their performances. Simulation has been the traditional tool used for performance evaluation, and accepted by many researchers including industry where hardware implementions can be expensive or emulations (e.g., by Linux) can be time consuming for initial study. Software simulation is widely used because it is very flexible to set up different scenarios quickly, and it can quickly and effectively verify the performances of the schemes under study.

Of the different simulation packages available, we shall use the OPNET simulator [Opnet05] because of its popularity and versatility in modeling the networks. As a network simulation tool, OPNET has many features and toolsets. These include a process model and associated packet format for abstracting different protocols and behavior of a particular network component, a node model for specifying network component interface, a project window for defining the topology of the network and various linkages, and a simulation window that is able to capture and/or show the results of network simulation [Qwha14]. OPNET Modeler also has a development environment that can model any network type and various other technologies like MPLS (Multiprotocol Label Switching), IPv6, and TCP. It also provides a C language for us to implement our algorithms/protocols in details via finite state machines for integration with existing network protocols. According to some experts, "OPNET Modeler offers the fastest discrete event simulation engine when compared with other networking solutions in the industry" [Qwha14].

Furthermore, we already have an existing OPNET database in our CCNR (Computer Communication Network Research) lab from previous research work such as the TCP environment to facilitate our OPNET development. Therefore, we shall use OPNET modeler to verify and evaluate the performances of our new controllers under various network scenarios. We shall also make comparison to other existing protocols that are being studied using simulation. Besides, we use the worst traffic scenarios in our simulation to test the rigorousness of our controllers with greedy data sources. Finally, we provide application examples to further verify the effectiveness of our schemes. More approach specific to each type of design are provided in the following.

1.7.1 The FLC-based Controller

This new scheme pays attention to the following methodologies as well as the merits of the existing protocols. Firstly, in order to keep the implementation simple, like TCP, the new controller treats the network as a black box in the sense that queue size is the only parameter it relies on to adjust the source sending rate. The adoption of queue size as the unique congestion signal is inspired by the design experience of some previous AQM controllers (e.g., RED and API-RCP) in that queue size can be accurately measured and is able to effectively signal the onset of network congestion. Secondly, the controller retains the merits of the existing rate controllers such as XCP and RCP by providing explicit multi-bit congestion information without having to keep per-flow state information. Finally, we rely on the fuzzy logic theory to design our controller to form a traffic management procedure due to its capability and designing convenience as discussed above.

To eliminate the greedy behavior of the sources in the present TCP/IP networks, we assume that every source in the Internet takes the initiative just to demand the necessary bandwidth as per its applications. To prove the stability of our new FLC-based control system, we shall first use the LDM (one of the Lyaponov's stability criteria to be discussed later) to derive the stability preconditions in the time domain. Then we exhaustively show how the stability of the control system is guaranteed in whatever network traffic conditions.

The controller will be shown that it has a general form similar to a classical PI controller but with nonlinear attributes. Then based on the properties and computation logics of our controller, we will derive an instance equation to illustrate the analytical form of our scheme, which would help us understand why our controller is intelligent and superior to its counterparts (i.e., traditional linear PI controllers) in controlling a time-delayed nonlinear network congestion control system.

Finally, to analyze the computational complexity of our new scheme, the controller algorithm will be dissected, and the operations will be converted into machine cycles before it is compared with other controllers.

1.7.2 The NUM-based Controller

Inheriting the preliminary NUM method from [KeMa98], we would like to extend the existing works to formulate our new NUM-based scheme.

To model a new NUM-based explicit congestion controller, social welfare[2] will be considered, instead of end user utility (as in the existing works). To do so, the problem is formulated based on the links of a network because they manage how much resource (i.e., bandwidth) each end user should obtain, just like the distribution of well-being by governments. However, the fundamental difference of our approach is that an optimization is performed for each link rather than the global optimization of all users and all links together as done in other existing works (e.g., [Hark05, LiSh06, PrVa11]). This allows a distributed algorithm where each link along a data path can desire a source sending rate so that we can design our new scheme with the congestion signal fed back from the most congested link (which results in network-wise max-min fairness [Welz05]), instead of the cumulative congestion signal over the flow path. In this way, a centralized global optimization is avoided, and the traffic required to broadcast the link prices is reduced in the existing works. On the other hand, the allocation mechanism of the proportional fairness becomes link-wise in our scheme, instead of network-wise like in the relatively explicit controllers.

Theoretically, we will formulate and solve our problem with the convex optimization approach. Based on the duality technique, the scheme can be converted into a dual problem so that we can economically interpret the Lagrange multiplier as "a unit bandwidth price" that a router uses to distribute (or say, "sell") its link bandwidth. Then we employ the RM (Resource Management) theory from economics [YeMc11] to interpret such a "price" under various traffic conditions, just like the plane ticket price fluctuations in different travelling seasons.

In order to relax the unpractical static assumption of link bandwidth in former works, we would like to include the time-varying feature of networks to allow our controller to take on more general applications. To this regard, we would incorporate the queue size $q(t)$ into our controller model as a remedial measure. The rational is that when the link bandwidth c_l suddenly shrinks, $q(t)$ has to build up due to the decreased bandwidth supply. Such queue dynamics can provide immediate information to sources which can then decrease their sending rates. In the meanwhile, as a side benefit, the introduction of $q(t)$ in the new NUM-based controller model will also significantly reduce the queue size that the relatively explicit controllers cannot achieve when matching the incoming traffic to the bandwidth

[2]Social welfare originally means the provision of a minimal level of wellbeing and social support for all citizens from government or organizations [Wiki13].

supply of a link, and hence overcomes the potential packet loss and high queueing delay problems.

To present more merits of the new NUM-based controller, we will investigate the following three aspects: the convergence property, the stability in the presence of time delay, and the robustness property. For the convergence analysis, we will take a reasonable assumption on the bandwidth price that will allow us to prove with a lemma that the objective function of the dual problem of the new NUM-based controller is strongly convex. Then we will propose the other lemma based on a Lipschitz continuity condition for a preparative purpose. Finally, we will prove that the new NUM-based controller algorithm is able to converge to its equilibrium point under certain conditions. To prove system stability of the controller with time delay, we will resort to the linearization technique and employ the Nyquist criterion to obtain the sufficient conditions for the system to be locally stable. Finally, robustness is demonstrated via an analysis of how the optimal revenue of a link reacts to the dynamics of link bandwidth.

1.8 Contributions

This thesis has contributed two new rate-based explicit congestion controllers that can combat the network parameter dynamics and improve sources and queue performances. Also, we have provided their implementations on wireless local area networks which are ubiquitous everywhere.

The contributions for the FLC-based controller are as follows.

1) An intelligent rate-based explicit traffic management scheme (called the IntelRate controller) is designed for the high-speed networks using the fuzzy logic theory ;
2) Less performance parameters is used while providing better performances than the existing explicit control protocols with the application of such a fuzzy logic controller;
3) Max-min fairness is provided even under large network dynamics that usually render many existing controllers unstable;
4) For the first time to our best knowledge, a fuzzy-logic-based congestion control system is proved to be globally asymptotically stable for the rate-based explicit congestion control.

The contributions for the NUM-based controller are as follows.

1) A new class of NUM-based algorithm, called the fully explicit congestion control protocol, is identified for the proper formulation of an optimized traffic congestion controller;
2) The drawback of the existing relatively explicit congestion controllers "biasing" the multi-bottlenecked users is overcome by extending the NUM approach to formulate a distributed OFEX (Optimal and Fully EXplicit) congestion controller, which exercises link-wise proportional fairness to achieve network-wise max-min fairness;
3) $q(t)$ is proposed as a remedial measure against the time-varying link capacities, thus making the OFEX controller applicable in general networks. Furthermore, with the remedial measure, the queue size can be kept at a low level upon congestion and thus improving queueing delay performance and reducing packet loss probability.

1.9 Thesis Organization

Chapter 2 provides the network modeling and assumptions for our new schemes. Chapters 3 and Chapter 4 present the design, analysis and performance evaluation of the IntelRate controller and the OFEX controller, respectively. We provide applications of our new controllers in wireless local area networks in Chapter 5. More insights about the controller design are discussed in Chapter 6. With Chapter 7, we conclude the thesis.

In order to make our thesis concise, we have decided to move some theoretical backgrounds, analyses and experimental performances of the IntelRate controller and the OFEX controller to the appendices. Appendices A and B are the introductions of the fuzzy logic control and Lyapunov's Direct Method. The stability proofs, characterization, computational analysis and other design considerations of the IntelRate controller are shown in Appendices C to F. Appendices G to J provide the theoretical concepts, computational complexity analysis as well as experimental investigation on the convergence of the OFEX controller.

1.10 Publication

A good portion of our work has been published to the following conferences and journals.

Conferences

[LiYa10] J. Liu, O. W.W. Yang, "Design and evaluation of an intelligent controller for

heterogeneous networks", *Proceedings of the IEEE Global Communications Conference* (*GLOBECOM2010*), Miami, U.S.A, Dec.6-10, 2010.

[LiYa11a] J. Liu, O. W.W. Yang, "Stability analysis and evaluation of the IntelRate controller for high-speed heterogeneous networks," *Proceedings of the IEEE International Conference on Communications,* (*ICC2011*), Kyoto, Japan, Jun. 5-9, 2011.

[LiYa11b] J. Liu, O. W.W Yang, "Characterization of the IntelRate controller for high-speed networks", *Proceedings of the 9th Annual Conference on Communication Networks and Services Research* (*CNSR2011*), pp.63-68, Ottawa, Canada, May 2-5, 2011.

[LiYa13a] J. Liu, O. W. W. Yang, "An optimal and fully explicit rate controller for high-speed networks", *Proceedings of the IEEE International Conference on Communications,* (*ICC2013*), Budapest, Hungary, Jun.9-13, 2013.

[LiYa13b] J. Liu, O. W.W. Yang, "The number of linguistic variables in the design of the IntelRate controller for Internet traffic control", *Proceedings of the 10th International Conference on Fuzzy Systems and Knowledge Discovery* (*FSKD2013*)*,* Shenyang, China, Jul. 23-25, 2013.

[LiYa13c] J. Liu, O. W.W. Yang, "Using the IntelRate controller to improve throughput and queue size in high-speed WLAN", *Proceedings of the IEEE Military Communications Conference* (*MILCOM*), San Diego, USA, Nov.18-20, 2013.

[LiYa14a] J. Liu, O. W.W. Yang, "Convergence performance of the OFEX controller for high-speed networks", *to appear in Proceedings of the IEEE International Conference on Communications,* (*ICC2014*), Sydney, Australia, Jun. 10-14, 2014.

Journals

[LiYa13d] J. Liu, O. W.W. Yang, "Using fuzzy logic control to provide intelligent traffic management service for high-speed networks", *IEEE Transactions on Network and Service Management* (*TNSM*), Vol.10, No.2, pp.148-161, Jun. 2013.

[LiYa13e] J. Liu, O. W.W. Yang, "A stable fuzzy logic controller using the least parameter for explicit traffic control", *International Journal of Innovative Computing, Information and Control*, Vol.9, No.7, pp.2801-2819, Jul. 2013.

[LiYa13f] J. Liu, O. W.W. Yang, "How does the IntelRate controller save router computation resources?" *Journal of Convergence Information Technology*, Vol.8, No.5, pp.1-10, Jul. 2013.

[LiYa13g] J. Liu, O. W.W. Yang, "Experimental investigation of two important parameters of the IntelRate controller for high-speed networks", *International Journal of Control and Automation*, Vol.6, No.4, pp.243-258, Aug. 2013.

[LiYa14b] J. Liu, O. W.W. Yang, "An OFEX controller to improve queueing and user performance in multi-bottleneck networks", *to appear in Electronics and Telecommunications Research Institute* (*ETRI*) *Journal*, June 2014.

[LiYa14c] J. Liu, O. W.W. Yang, "Convergence, stability and robustness analysis of the OFEX controller for high-speed networks", *under preparation for journal submission*, Feb. 2014.

Chapter 2

Network Operation, Modeling and Assumptions

This relatively short chapter will introduce the general network layout as well as the general operation and queueing model of our new congestion control schemes to be designed in later chapters. It also introduces the common notations, symbols and terminologies for later use while any notations particular to a controller will be introduced upon their designs.

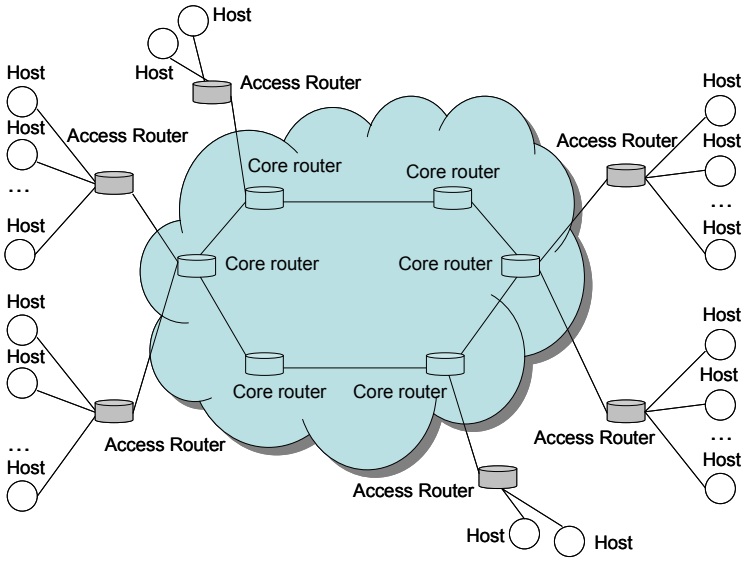


Fig. 2.1: Network Topology

2.1 Network Layout and Operation

Fig. 2.1 depicts a general packet-switching network under our investigation. It is a backbone network composed of high-speed core routers and access routers. Hosts (such as Internet users/sources/receivers) are attached to the access routers and cooperate with the core routers to enable end-to-end communications. In general, we consider the network to consist of a set of sources $S=\{1, 2, \ldots, s,\ldots\}$ and links $L=\{1,2, \ldots, l,\ldots\}$. The total number of sources and links in the network are denoted by $|S|$ and $|L|$, respectively. Each link l has a set of flows, designated by $S_l(t)$.

Congestion occurs when many flows traverse a router and cause its queue to overflow, thus making it a bottleneck in the Internet. Since any router may become a bottleneck along an end-to-end data path, we would like each router to be able to manage its traffic. Below is a

general operation principle of our new traffic management/control algorithms (i.e., the IntelRate controller and the OFEX controller) in each router.

Each packet has a dedicated field *Req_rate* inside the header, which can be updated by any router when the packet travels from its source to destination. Specifically, each router along the data path will compute an allowed source transmission rate according to the queue size or available bandwidth, and then compare it with the rate already recorded in *Req_rate* field of a packet header. If the former one is smaller than the latter one, the *Req_rate* field in the packet header will be updated; otherwise it remains unchanged. After the packet eventually arrives at the destination, the value of the *Req_rate* field reflects the allowed rate from the most congested router along the path. The receiver then sends this value (as an explicit congestion signal) back to the source via an ACK (acknowledgment) packet, and the source uses it to update its current sending rate. If no router modifies the *Req_rate* field, it means that all routers en route allow the source to send its data with the desired rate.

Further operation details of the IntelRate controller and the OFEX controller will be discussed in Chapter 3 and Chapter 4, respectively.

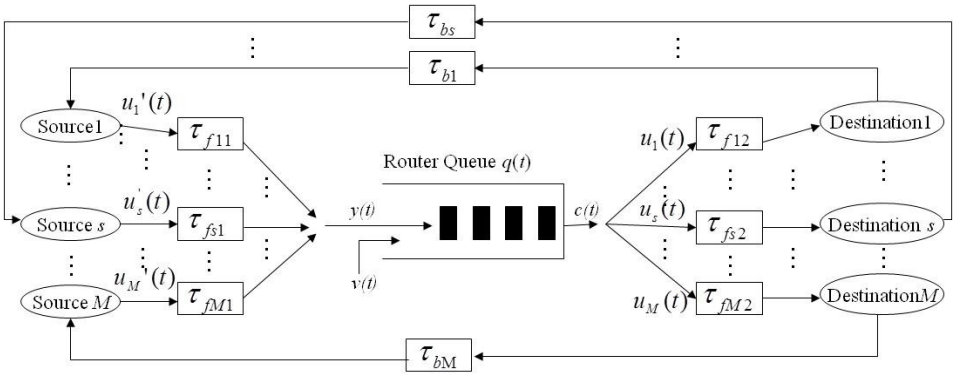


Fig. 2.2: End-to-End Congestion Control System Model

2.2 Modeling

In order to analyze and implement our new controller in each router, we model a typical AQM router in Fig. 2.2 with M sources sending their Internet traffic to their respective destinations. For s=1,2,…,M, $u_s'(t)$ is the current sending rate of source s; $u_s(t)$ is the sending rate of source s determined by the routers. In the text, the variable $u(t)$ is also used as a general variable representing the output of a flow in the controllers, and has the same

meaning as $u_s(t)$ in Fig. 2.2 when we are dealing with an individual flow-s. The variable $y(t)$ is the incoming aggregate controlled flow rate; $v(t)$ is the incoming aggregate uncontrolled flow rate, and $c(t)$ is the link capacity, or say, link bandwidth (measured in bits/second (i.e., bps) or packets/second), which is the sum of the available bandwidth and the occupied bandwidth of a link). It is noteworthy that the incoming flow rate and the bandwidth of a link may both vary with respect to the time. For a particular source-destination pair i, the parameter τ_{fs1} is the time delay of a packet from source s to the router, and τ_{fs2} is the time delay of the packet of source i from the router to the destination i, while τ_{bs} is the feedback delay from destination i back to source i. Obviously, $\tau_{ps} = \tau_{fs1} + \tau_{fs2} + \tau_s$ is the RTPD (Round Trip Propagation Delay). Considering other delays en route (e.g., queueing delay), source s may update its current rate $u_s'(t)$ according to the $u_s(t)$ when the ACK packet arrives after one RTT τ_s.

As reviewed before, we shall use the "super queue" model to capture the queueing in the router queue where $q(t)$ is the number of packets at time t which is the interaction of the cumulative arrival (denoted as $Y(t)$) and the cumulative departure (denoted as $C(t)$) in (0, t) subject to $Y(t) \geq C(t)$. For considering both the arrival and the departure of packets as point processes [Klei75], we have

$$q(t)=\max(0, Y(t)-C(t)) \tag{2-1}$$

Note that this is an exact depiction of the queueing process of an infinite buffer system without any assumptions on the characteristics of the arrival and departure processes, nor the traffic levels (whether it is light or heavy traffic). Differentiating (2-1), one can obtain

$$\dot{q}(t) = \begin{cases} y(t) - c(t) & q(t) > 0 \\ \max(0, y(t) - c(t)) & q(t) = 0 \end{cases} \tag{2-2}$$

that can be found in many literatures as reviewed before (e.g., [KaHa02, DuMc06, KeRa10]). In Eqn. (2-2), $y(t) = \mathrm{d}Y(t)/\mathrm{d}t$ and $c(t)=\mathrm{d}C(t)/\mathrm{d}t$ which can take on the general meaning of arrival rate and departure/service rate as we generally know. By also considering other arriving traffic rates such as the uncontrolled arriving traffic rate $v(t)$ in Fig. 2.2, we have

$$\dot{q}(t) = \begin{cases} y(t) + v(t) - c(t) & q(t) > 0 \\ [y(t) + v(t) - c(t)]^+ & q(t) = 0 \end{cases} \tag{2-3}$$

Physically, the change rate of the queue is determined by the arrival rates of the controlled

and uncontrolled traffic as well as their service rate. To prevent the queue from overflowing, $y(t)$, $v(t)$ and $c(t)$ should satisfy a demand-supply model, i.e.,

$$y(t)+v(t)\leq c(t). \quad (2\text{-}4)$$

Several points are worth mentioning here. First of all, queue length is not the output of our controllers. Our controller output is the required sending rates (throughput) of the sources which will interact with other traffic streams at the controller (along with its buffer in the router) to produce the queue length as defined in (2-3). Secondly, the queue length here is defined and can be measured exactly at any particular time points, which is the feature of our control algorithms. There is no approximation modeling is required. For instance, the queue length can be measured at customer departure points to execute the algorithms in the computers/routers/switches. Specific details will be found in later chapters, and also, the discretization of continuous-time systems for implementation can be found in references such as [LaZi06]. Finally, since the input queue in a router is usually not configurable/modifiable, the "super queue" model is generally applied to the output queue (e.g., at the egress interface) of a router.

2.3 Assumptions

Unless otherwise specified, the following assumptions for the remainder of the thesis pertain.

1) A destination always has enough buffer space to receive data from its source. This is because the destination will not impose any constraint on the source sending rate when we verify the effect of our new control scheme in a bottlenecked router.
2) The propagation delay and the queuing delay along the data path are the two dominant components of the RTT while other components like processing delay of a packet in routers or hosts are negligible in comparison.
3) A link means an output interface of a router. Furthermore, a link has only one queue with the queueing discipline of FIFO (First-In-First-Out).
4) Long-lived flows with infinitely long files are used to approximate the greedy behavior of a source when active. This would generate the severest traffic in order for us to rigorously verify the robustness of our new schemes. They are more demanding than other traffics such as those with Poisson arrivals or with a self-similar characteristic.
5) Network connections are very reliable so that the effect due to link error/breakage

(which cause bit errors, data and ACK package losses) is negligible in order to facilitate the analysis and performance evaluation of our controllers.

6) Buffer size is large enough to have negligible loss rate so that an infinite buffer assumption can be approximately true. As can be seen from our design, proof of stability and performance evaluation, our controllers can stabilize or reduce the queue length to some targeted queue lengths without packet loss.

7) Steady state means the output of a control system achieves a final value after all the transient constituents of the response have faded as $t \to \infty$ [DoBi08]. Since we have proved (or if it exists), we would assume such a value would be achieved in a limited time[3].

[3] Just like the steady state of the output of a first-order control system can be achieved after 5 times of the system time constant without $t \to \infty$ [Nise11].

Chapter 3
The IntelRate Controller

In this chapter, we shall design an intelligent rate controller, called the IntelRate controller, to tackle the traffic mass, and then conduct analysis and performance evaluation. Since the IntelRate controller employs FLC (Fuzzy Logic Control), the principles and details of FLC can be found in many textbooks such as [PaYu98, ZhPh05, MiKl06]. Some basic FLC definitions and terminologies are also summarized in Appendix A.

We describe the design of the IntelRate controller in Section 3.1, and analyze the stability of the IntelRate controller in Section 3.2. We then extensively evaluate the IntelRate controller performance with OPNET modeler in Section 3.3. For further analysis on the IntelRate controller, the characterization can be found in Appendix D, and the computation complexity can be found in Appendix E which indicates how well the controllers can be used for real-time operation.

3.1 The IntelRate Controller

A fuzzy logic controller design usually involves four steps, i.e. designing a rule base, fuzzification, making inference and defuzzification. We use these 4 steps to formulate our FLC-based controller along with the fuzzy linguistic descriptions and the membership functions. For some common notations or symbols used in this chapter, please refer to Chapter 2.

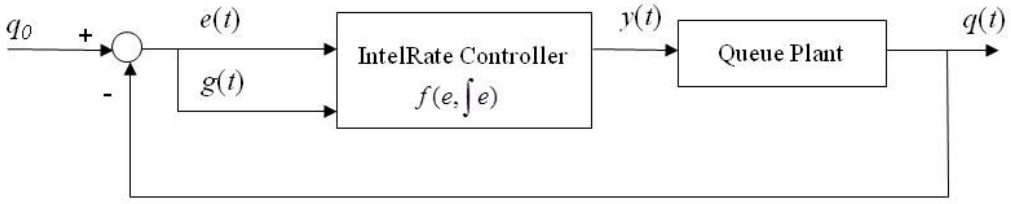


Fig. 3.1: The Closed-Loop IntelRate Control System

3.1.1 Controller Design

Fig. 3.1 depicts the components of our fuzzy logic congestion control system. As seen, the controller (called the IntelRate controller) is a TISO (Two-Input-Single-Output) controller. The TBO (Target Buffer Occupancy) q_0>0 is the queue size level we aim to achieve upon

congestion. The queue deviation $e(t)=q_0-q(t)$ is one of the two inputs of the controller. As defined in Chapter 2, the variable $y(t)$ represents the aggregate input rate from all controlled sources into the queue.

In order to remove the steady state error, we choose the integration of $e(t)$ as the other input of the controller, i.e. $g(t)=\int_0^t e(\phi)d\phi$. Under heavy traffic situations, the IntelRate controller would compute an allowed sending rate $u_s(t)$ for flow i to the current IQSize so that $q(t)$ can be stabilized around q_0. This is the rate sent to the queue of the router one RTT τ_i later by flow i. So effectively, the present aggregate incoming rate $y(t)$ to the queue of the router is the total sending rates determined from one RTT τ_i earlier, i.e., $y(t)=\sum u_s(t-\tau_i)$.

In this system, one can see that the IQSize $q(t)$ is the only parameter each router needs to measure in order to complete the closed-loop control. Unlike other controllers reviewed in Chapter 1, $q(t)$ is the only major state variable to be kept track of in the algorithm and hence contribute to a low complexity of our algorithm. The interested reader can refer to the definition of IQSize in Section 3.3.1.3 later for the time points that $q(t)$ is measured and Appendix E1.3 for the computation complexity of obtaining $q(t)$.

To start, we define the crisp inputs $e(t)$, $g(t)$ and output $u(t)$ with the linguistic variables $\tilde{e}(t)$, $\tilde{g}(t)$ and $\tilde{u}(t)$, respectively. Then we apply N LVs (Linguistic Values) to each of these linguistic variables. We let $\overline{P}_i=\{\overline{P_i^j}: j=1,2,\cdots,N\}$ be the input LVs with i=1 for $\tilde{e}(t)$ and i=2 for $\tilde{g}(t)$. For example, for N= 9, we can designate $\tilde{e}(t)$ and $\tilde{g}(t)$ with the following LVs, which were obtained based on experiences (i.e., "expert knowledge") [PaYu98].

$\overline{P}_i^1$="NV(Negative Very Large)",
$\overline{P}_i^2$="NL (Negative Large)",
$\overline{P}_i^3$=" NM (Negative Medium)",
$\overline{P}_i^4$=" NS (Negative Small)",
$\overline{P}_i^5$=" ZR (Zero)",
$\overline{P}_i^6$=" PS (Positive Small)",
$\overline{P}_i^7$=" PM (Positive Medium)",
$\overline{P}_i^8$=" PL (Positive Large)" and
$\overline{P}_i^9$=" PV (Positive Very Large)".

Similarly, we designate N=9 output linguistic values with $\overline{U}=\{\overline{U}^j: j=1,2,\cdots,9\}$ for

$\widetilde{u}(t)$, in which

$\overline{U}^1$="ZR (Zero)",

$\overline{U}^2$="ES (Extremely Small)",

$\overline{U}^3$="VS (Very Small)",

$\overline{U}^4$="SM (Small)",

$\overline{U}^5$="MD (Medium)",

$\overline{U}^6$="BG (Big)",

$\overline{U}^7$="VB (Very Big)",

$\overline{U}^8$="EB (Extremely Big)",

$\overline{U}^9$="MX (Maximum)".

Table 3.1: Rule Table for the IntelRate Controller

Allowed Throughput *u(t)*		*e(t)*								
		NV	NL	NM	NS	ZR	PS	PM	PL	PV
g(t)	NV	ZR	ZR	ZR	ZR	ZR	ES	VS	SM	MD
	NL	ZR	ZR	ZR	ZR	ES	VS	SM	MD	BG
	NM	ZR	ZR	ZR	ES	VS	SM	MD	BG	VB
	NS	ZR	ZR	ES	VS	SM	MD	BG	VB	EB
	ZR	ZR	ES	VS	SM	MD	BG	VB	EB	MX
	PS	ES	VS	SM	MD	BG	VB	EB	MX	MX
	PM	VS	SM	MD	BG	VB	EB	MX	MX	MX
	PL	SM	MD	BG	VB	EB	MX	MX	MX	MX
	PV	MD	BG	VB	EB	MX	MX	MX	MX	MX

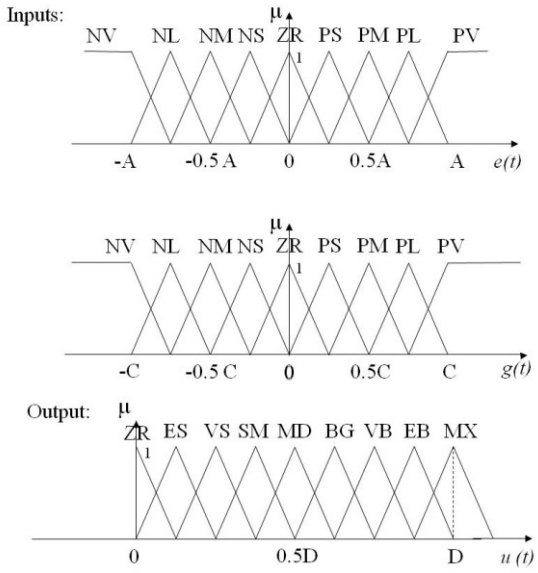


Fig. 3.2: Membership Functions without FS

Table 3.1 illustrates how the inputs and the output are quantified into N=9 LVs. It depicts the rule base which is the set of conditions mapping the inputs to the output using

the *"If…Then…"* format [PaYu98], e.g. "If $e(t)$ is Zero(ZR) and $g(t)$ is Positive Small(PS), Then $u(t)$ is Big(BG)." In the following sections, we refer to a rule in this table by the notation $(\overline{P}_1^j, \overline{P}_2^k, \overline{U}^l)$, where j,k,l=1,2,…,N, e.g. $(\overline{P}_1^5, \overline{P}_2^2, \overline{U}^2)$=(ZR, NL, ES).

The IntelRate controller employs the isosceles triangular and trapezoid-like functions as its MFs (Membership Functions). Fig. 3.2 describes the MFs used to determine the certainty of crisp input or output (where N=9). We define P_1 as UOD (Universe of Discourse) [4] [PaYu98] for the input $p_1=e(t)$, and P_2 is the UOD for the input $p_2=g(t)$. The value of MFs (i.e., the certainty degree) for crisp inputs p_i ($i=1,2$) is designated by $\mu_{P_i^j}(p_i)$. Similarly, we define Z as the UOD of the output $z=u(t)$, and the certainty degree of crisp output z is designated by $\mu_{U^j}(z)$. The above $\mu_{P_i^j}(p_i)$ or $\mu_{U^j}(z)$ is obtained by the "Singleton Fuzzification" method [PaYu98]. Thus the input and output fuzzy sets can be defined with $P_i^j = \{(p_i, \mu_{P_i^j}(p_i)) : p_i \in P_i\}$, and $U^j = \{(z, \mu_{U^j}(z)) : z \in Z\}$, j =1,2,…,N, respectively. The parameters A and C are the boundary values for the input triangle MFs, and D is the boundary value for the output.

To determine how much a rule is certain to current situation, our controller applies the Zadeh AND logic [YiSi90] to perform the inference mechanism, e.g. for the crisp inputs p_1 and p_2, the final certainty (also referred to as the *firing level*) of a rule is computed with $\mu_{P_1^m \cap P_2^n} = \min\{\mu_{P_1^m}(p_1), \mu_{P_2^n}(p_2) : p_i \in P_i\}$, $i=1,2$ and $m,n=1,2,\ldots,N$, where *min* is the *minimum operation* in the Zadeh AND logic.

Note that Fig. 3.2 describes the usual way of designing MFs with FLC to determine the certainty of a crisp input or output, where $e(t)$ and $g(t)$ have the same number of MFs without setting boundaries for the input $g(t)$. This type of design is not suitable when FLC is applied to our Internet traffic controller. The reason is twofold. 1) When e(t) and $g(t)$ have the same number of MFs, the output $u(t)$ has a tendency to change fast when the queue size has drastic variations. This would not give us a smooth source sending rate. 2) Since it is possible that a router operates under light traffic situations sometimes, $g(t)$ will increase indefinitely if it is not bounded. Therefore, a limit has to be imposed on $g(t)$. In light of the above issues, we require the $e(t)$ and $g(t)$ in our controller to operate on different number of MFs, and add an

[4] UoD is the crisp set in which the input or output takes the crisp values [PaYu98].

upper limit mq_0 to the highest LV of $g(t)$. The width of each MF for the inputs $e(t)$ and $g(t)$ is determined by the parameter m ($m \geq 1$) in order to realize a smaller TBO (i.e., q_0) and thus smaller queueing delay upon heavy traffic. Together with the N LVs, we call our design mechanism the FS (Fuzzy Smoother). The choice of parameters like N and m will be discussed later on.

While Fig. 3.3 is used to illustrate the design of a general FS, the designated values actually come from an example of N=9 LVs with the absolute values of both the upper and the lower limits of $g(t)$ set to mq_0. Since $e(t)$ is bounded by the physical size of a queue, we have the boundaries according to the limits $q_0\text{-}B \leq e(t) \leq q_0$. The boundary D of the output $u(t)$ is defined as the maximum value of the *Req_rate* among the active passing flows of a router. The vertical dashed lines in Fig. 3.3 denote those boundaries of inputs or output. Accordingly, the dashed box in Table 3.1 contains the rules that the IntelRate controller operates on. There are generally three LVs that $e(t)$ can take (i.e., "NS", "ZR" and "PS") compared with the nine values (from NV to PV covering all the LVs) for $g(t)$. As shown, nine different LVs (from ZR to MX) are used to give a gradual change in the output $u(t)$ for each combination of $e(t)$ and $g(t)$. Section F.1 in Appendix F provides experimental demonstration about the benefit of FS.

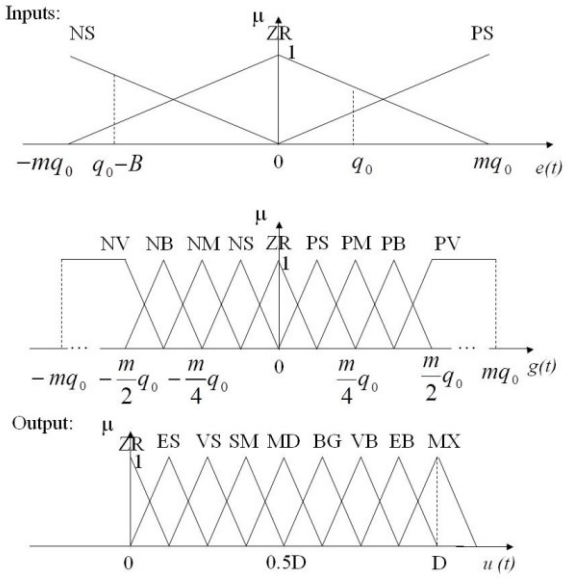


Fig. 3.3: Membership Functions with FS

From Eqn. (A-1) and (A-2) in Appendix A, the membership functions (N=9) for an input $e(t)$ in Fig. 3.3 can be expressed as follows:

$\mu_{NS}(e(t)) = f_1(e(t), NS_0, NSw_L, NSw_R)$;

$\mu_{ZR}(e(t)) = f_1(e(t), ZR_0, ZRw_L, ZRw_R)$;

$\mu_{PS}(e(t)) = f_1(e(t), PS_0, PSw_L, PSw_R)$,

where the parameters in $f_1(.)$ have the same notions as in Eqn. (A-1), and $\mu_{NS}(e(t))$, $\mu_{ZR}(e(t))$ and $\mu_{PS}(e(t))$ are respectively the certainty degrees $\mu_{P_1^4}(p_1)$, $\mu_{P_1^5}(p_1)$ and $\mu_{P_1^6}(p_1)$.

For an input *g*(*t*), the MFs are:

$\mu_{NV}(g(t)) = f_2(g(t), NV_0, NVw_R)$;

$\mu_{NL}(g(t)) = f_1(g(t), NL_0, NLw_L, NLw_R)$;

$\mu_{NM}(g(t)) = f_1(g(t), NM_0, NMw_L, NMw_R)$;

$\mu_{NS}(g(t)) = f_1(g(t), NS_0, NSw_L, NSw_R)$;

$\mu_{ZR}(g(t)) = f_1(e(t), ZR_0, ZRw_L, ZRw_R)$;

$\mu_{PS}(g(t)) = f_1(g(t), PS_0, PSw_L, PSw_R)$;

$\mu_{PM}(g(t)) = f_1(g(t), PM_0, PMw_L, PMw_R)$;

$\mu_{PL}(g(t)) = f_1(g(t), PL_0, PLw_L, PLw_R)$;

$\mu_{PV}(g(t)) = f_3(g(t), PV_0, PVw_L)$,

where the parameters in $f_1(.), f_2(.)$ and $f_3(.)$ have the same notions as in Equation (A-1), (A-3) and (A-4), $\mu_{NV}(g(t))$, $\mu_{NL}(g(t))$, $\mu_{NM}(g(t))$, $\mu_{NS}(g(t))$, $\mu_{ZR}(g(t))$, $\mu_{PS}(g(t))$, $\mu_{PM}(g(t))$, $\mu_{PL}(g(t))$ and $\mu_{PV}(g(t))$ are the certainty degrees $\mu_{P_1^j}(p_2)$ *j=1,2...N*, respectively.

Similarly, for an output *u*(*t*), the MFs are:

$\mu_{ZR}(u(t)) = f_1(u(t), ZR_0, ZRw_L, ZRw_R)$;

$\mu_{ES}(u(t)) = f_1(u(t), ES_0, ESw_L, ESw_R)$;

$\mu_{VS}(u(t)) = f_1(u(t), VS_0, VSw_L, VSw_R)$;

$\mu_{SM}(u(t)) = f_1(u(t), SM_0, SMw_L, SMw_R)$;

$\mu_{MD}(u(t)) = f_1(u(t), MD_0, MDw_L, MDw_R)$;

$\mu_{BG}(u(t)) = f_1(u(t), BG_0, BGw_L, BGw_R)$;

$\mu_{VB}(u(t)) = f_1(u(t), VB_0, VBw_L, VBw_R)$;

$\mu_{EB}(u(t)) = f_1(u(t), EB_0, EBw_L, EBw_R)$;

$\mu_{MX}(u(t)) = f_1(u(t), MX_0, MXw_L, MXw_R)$,

where $\mu_{ZR}(u(t))$, $\mu_{ES}(u(t))$, $\mu_{VS}(u(t))$, $\mu_{SM}(u(t))$, $\mu_{MD}(u(t))$, $\mu_{BG}(u(t))$, $\mu_{VB}(u(t))$, $\mu_{EB}(u(t))$ and $\mu_{MX}(u(t))$ are the certainty degrees $\mu_{U^j}(z) j=1,2...N$, respectively.

For the defuzzification algorithm, the IntelRate controller applies the COG (Center of Gravity) method to obtain the crisp output with the equation $u(t) = (\sum_{j=1}^{k} c_j S_j)/(\sum_{j=1}^{k} S_j)$[PaYu98], where k is the number of rules, c_j is the bottom centroid of a triangular in the output MFs, and S_j is the area of a triangle with its top chopped off as per $\mu_{P_1^m \cap P_2^n}$. Since each parameter in the crisp input pair (p_1, p_2) can take on two different values in the IntelRate controller, we have altogether k=4 rules for defuzzification each time.

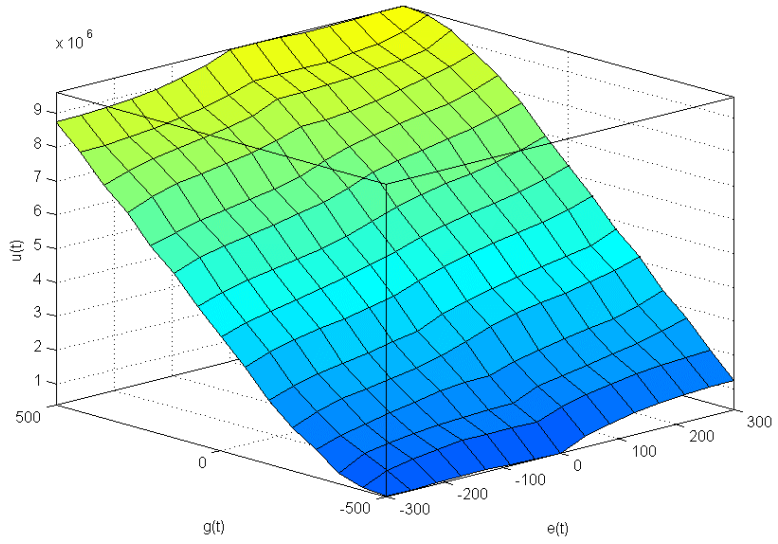


Fig. 3.4: A Three-Dimensional Control Surface Example

Fig. 3.4 shows the three-dimensional non-linear control surface using the above constructed rule base and the 9 LVs of inputs and output variables. In this example, the control surface is shaped with an input $e(t)$ having a range of [-300, 300], an input $g(t)$ in a

range of [-500, 500] and the $u(t)$ in a range of $[0, 10^7]$.

The control surface graphically shows the relationships between the inputs $e(t)$, $g(t)$ and the output $u(t)$ under the designed rule base and LVs. For example, -300 is the minimum value for $e(t)$ in Fig. 3.4. By looking at the dashed box of Table 3.1, one sees that this corresponds to NS as the smallest LV of $e(t)$. Likewise, the minimum value -500 corresponds to the NV as the smallest LV of $g(t)$, and the minimum value 0 corresponds to ZR for the output $u(t)$. Hence, (-300, -500, 0) reflects the rule (NS, NV, ZR). With the same analysis, other points in Fig. 3.4 can be mapped to those rules in Table 3.1.

Note that the control surface is obtained using Matlab [Matl13] and the ranges for the inputs/outputs are for illustration purpose. In a practical implementation (e.g., in OPNET [Opnet05] demonstrated later on), $e(t)$, $g(t)$ and $u(t)$ can take on any practical ranges determined by the specific TBO and buffer capacity of a router as well as the desired sending rate of sources.

3.1.2 Design Parameters

From our design above, one can see there are different parameters (such as N, m and D) which ultimately will affect the performance of our traffic controller. To choose appropriate values for these parameters, we have done extensive experiments (see Section 3.3 and Appendix F) and thorough analysis which will be presented in our design guideline discussion in Chapter 6.

3.1.3 The Control Procedure

As discussed for the operation principle in Chapter 2, we need to include in the congestion header a new field called *Req_rate* in every packet to carry the desired sending rate from the source and which will be continuously updated by the allowed sending rate when the packet passes each router. Below is a summary of the traffic-handling procedure of the IntelRate controller in a router.

(1) Upon the packet departure, the router

(1a) extracts *Req_rate* from the congestion header of the packet.

(1b) samples IQSize $q(t)$ and update $e(t)$ and $g(t)$.

(2) Compare $u(t)$ with *Req_rate* and send the packet:

(2a) if $u(t) < Req\_rate$, it means that the link does not have enough bandwidth to accommodate the requested amount of sending rate. The *Req_rate* field in the congestion header is then updated by $u(t)$;

(2b) otherwise the *Req_rate* field remains unchanged.

(2c) Send the packet out.

(3) If an operation cycle d is over, update the output $u(t)$ and the output edge value of D.

Note that this procedure actually allows the router to perform the max-min fairness in that the greedy flows are always restricted to $u(t)$ by a router under heavy traffic conditions while those small flows whose desired sending rate are smaller than $u(t)$ along their data path have no such a restriction. As mentioned in the operation principle, when the packet arrives at the destination, the receiver extracts *Req_rate* from the header and records it into the ACK packet and sends it back to the source. In addition, the step (1b) mandates that the queue size $q(t)$ is sampled upon each packet departure in order to calculate $e(t)$ and $g(t)$. The implementation efficiency of the control procedure above is analyzed in Appendix E in order to promote the feasibility of the IntelRate controller deployment.

3.2 Stability Analysis

The IntelRate control system can be shown to be globally asymptotically stable. Our approach is to reversely apply the LDM (Lyapunov's Direct Method), i.e., by first deriving what the controller is supposed to behave in order to meet the asymptotical stability conditions, and then showing that the controller is behaving as expected. Appendix B is a summary of the concept and conditions of LDM, where one can appreciate the rationale of applying this method to prove the stability of the IntelRate controller. The readers can also find the detailed theory of LDM in [RoHa77, MiKl06] that we have referred to a few times in this section. To prove the stability of the controller in Section 3.2.3 and Section 3.2.4, we need the assumption and proposition stated below.

3.2.1 The Transformed System Model

To simplify our notation in the analysis below, we let $(y(t), q(t))$ be the state vector of the IntelRate control system described in Fig. 3.1. The following assumption and proposition

would allow us to establish the transformed system model and stability theorems later on.

Assumption 1: For a router using the IntelRate controller, the uncontrolled traffic $v(t)$ is far less than dominant controlled traffic $y(t)$ so that we can neglect $v(t)$, i.e., $v(t)$ =0.

Comment: In the network nowadays, the controlled TCP generates 90-95% of the Internet traffics [Floy08], which means all other uncontrolled traffics such as UDP (User Datagram Protocol) making up only a small portion of the Internet traffic. To our best knowledge, this assumption appears to be practical at the time of this writing.

Proposition 1: For the closed-loop unity feedback system depicted in Fig. 3.1 with the non-zero reference value q_0, let y_s be the steady state controller output (also the aggregate incoming traffic to the router), and q_s be the steady state IQSize. If the system is stable under heavy traffic conditions, then the steady state (y_s, q_s) would be a non-zero vector. Furthermore, $q_s = q_0$, and $y_s = c(t)$ if the system steady state error is negligible.

Proof: This proposition is straightforward from the perspective of control theory. If a closed-loop system with a unity feedback in Fig. 3.1 to be stable as $t \to \infty$, its steady state output would approach the reference input if the steady state error can become negligible [DoBi08], i.e., $q(t)=q_0 \neq 0$ (q_0 is the TBO). Since the inbound and the outbound traffic in the queue must strike a balance to achieve stability at steady state, we have y_s equal to the link capacity $c(t)$, i.e., $y_s = c(t) \neq 0$. The proof is complete. ■

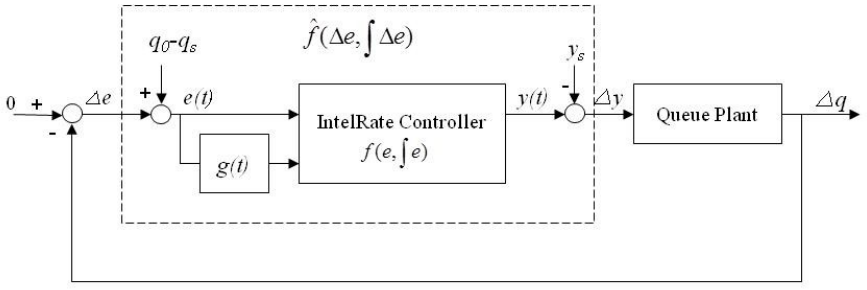


Fig. 3.5: Transformed System Model with Steady State at Zero

Since a non-zero steady state vector (y_s, q_s) does not meet the prerequisite of LDM for the stability analysis [RoHa77], we must perform a transformation for the IntelRate control system before we establish the conditions for the system to achieve stability. To do so, we use the approach described in [MiKl06] to convert (y_s, q_s) into a zero vector. This is done by

defining a new state vector（Δy, Δq) which is the deviation of ($y(t)$, $q(t)$) from (y_s, q_s). Fig. 3.5 shows the transformed equivalent system of Fig. 3.1.

3.2.2 The Stability Conditions

With the definition of stability (Section 1.7) and with the transformed system model in Fig. 3.5, we can now find the conditions that guarantee the stability of the IntelRate control system.

Since（$\triangle y$, $\triangle q$) are the deviations of ($y(t)$, $q(t)$) from (y_s, q_s), we have

$$\begin{cases} \Delta y = y(t) - y_s \\ \Delta q = q(t) - q_s \end{cases} \tag{3-1}$$

and their derivatives are

$$\begin{cases} \dot{\Delta} y = \dot{y}(t) \\ \dot{\Delta} q = \dot{q}(t) \end{cases} \tag{3-2}$$

From Fig. 3.1, the output equation of our IntelRate controller can be represented with

$$y(t) = f(e(t), g(t)) \tag{3-3}$$

Subsequently, the output change rate of the controller is

$$\dot{y}(t) = \frac{\partial f}{\partial e} \cdot \frac{de(t)}{dt} + \frac{\partial f}{\partial g} \cdot \frac{dg(t)}{dt} \tag{3-4}$$

Since $g(t)$ is the integral of $e(t)$, (3-4) can be rewritten as

$$\dot{y}(t) = \frac{\partial f}{\partial e} \cdot \dot{e}(t) + \frac{\partial f}{\partial g} \cdot e(t) \tag{3-5}$$

Because the queue deviation (deviation from the TBO q_0) is $e(t) = q_0 - q(t) = q_0 - (\Delta q + q_s)$, $e(t)$ is a function of $\triangle q$. Consequently,

$$\dot{e}(t) = -\Delta \dot{q} \tag{3-6}$$

(3-6) means $\dot{e}(t)$ is s a function of $\Delta \dot{q}$. The right-hand side of (3-5) is thus a function of $\triangle q$ and $\Delta \dot{q}$ which we can rewrite as $\dot{y}(t) = h(\Delta q, \Delta \dot{q})$. Using (2-1), the lower equation of (3-2) can be written as $\dot{\Delta} q = \begin{cases} \Delta y + y_S + v(t) - c(t) & q(t) > 0 \\ [\Delta y + y_S + v(t) - c(t)]^+ & q(t) = 0 \end{cases}$. One can see that $\Delta \dot{q}$ is a function of $\triangle y$. Therefore, we have

$$\dot{y}(t) = h(\Delta q, \Delta y). \tag{3-7}$$

By carefully checking (3-7), the function $h(\Delta q, \Delta y)$ represents the action of the controller in response to the change of the aggregate arrival rate, i.e., $\dot{y}(t)$.

Till now, the system equation (3-2) can be rewritten as

$$\begin{cases} \dot{\Delta y} = h(\Delta y, \Delta q) \\ \dot{\Delta q} = \begin{cases} \Delta y + y_S + v(t) - c(t) & q(t) > 0 \\ [\Delta y + y_S + v(t) - c(t)]^+ & q(t) = 0 \end{cases} \end{cases} \tag{3-8}$$

Using $\boldsymbol{x} = [x_1(t), x_2(t)] = [\Delta y, \Delta q]$, we obtain a more general form of the state equations,

$$\begin{cases} \dot{x}_1 = h(x_1, x_2) \\ \dot{x}_2 = \begin{cases} x_1 + y_s + v(t) - c(t) & q(t) > 0 \\ [x_1 + y_s + v(t) - c(t)]^+ & q(t) = 0 \end{cases} \end{cases} . \tag{3-9}$$

Without loss of generality, we choose the positive scalar $V(\boldsymbol{x}) = \frac{1}{2}({x_1}^2(t) + {x_2}^2(t))$ as the Lyapunov function. To guarantee the system's stability, we must have $\dot{V}(\boldsymbol{x}) = x_1(t)\dot{x}_1(t) + x_2(t)\dot{x}_2(t) < 0$. Specifically, $x_1(t)h(x_1, x_2) + x_2(t)(x_1 + y_s + v(t) - c(t)) < 0$

Note that we have reasonably neglected the boundary conditions of $\dot{x}_2(t)$ in (3-9) as done in [KaHa02]. Later on we shall show how our IntelRate control system designed with this inequality can maintain stability with respect to each boundary condition.

According to Lyapunov's Direct Method [RoHa77], the system would be asymptotically stable if

$$x_1(t)h(x_1, x_2) + x_2(t)(x_1(t) + y_S + v(t) - c(t)) < -\varepsilon, \quad \text{for any small } \varepsilon > 0, \tag{3-10}$$

Therefore, for $x_1(t) \neq 0$, $h(x_1, x_2)$ should satisfy

$$\begin{cases} h(x_1, x_2) > \frac{1}{x_1(t)}[x_2(t)(c(t) - x_1(t) - y_S - v(t)) - \varepsilon] & if\ x_1(t) < 0 \\ h(x_1, x_2) < \frac{1}{x_1(t)}[x_2(t)(c(t) - x_1(t) - y_S - v(t)) - \varepsilon] & if\ x_1(t) > 0 \end{cases} \tag{3-11}$$

i.e.,

$$\begin{cases} \dot{y}(t) > \frac{1}{x_1(t)}[x_2(t)(c(t) - x_1(t) - y_s - v(t)) - \varepsilon] & if\ x_1(t) < 0 \\ \dot{y}(t) < \frac{1}{x_1(t)}[x_2(t)(c(t) - x_1(t) - y_s - v(t)) - \varepsilon] & if\ x_1(t) > 0 \end{cases} \quad \text{for any small } \varepsilon > 0 \tag{3-12}$$

The two inequalities in (3-12) depict how the IntelRate controller $y(t) = f(e(t), g(t))$ should behave in order to guarantee the asymptotic stability of the system upon different traffic conditions. These behaviors are established by the following two theorems.

3.2.3 Stability Analysis under Light Traffic ($x_1(t) < 0$)

Theorem 1 below establishes the behaviour of the IntelRate controller under light traffic.

Theorem 1: The inequality $\dot{y}(t) > \frac{1}{x_1(t)}[x_2(t)(c(t) - x_1(t) - y_s - v(t)) - \varepsilon]$ *if* $x_1(t) < 0$ in (3-12) corresponds to the light traffic scenario in a router where the IntelRate controller allows each source to increase its sending rate until the desirable rate is reached while maintaining the system stable.

Proof: Please see Appendix C.1 for details.

3.2.4 Stability Analysis under Heavy Traffic ($x_1(t) > 0$)

Theorem 2 below establishes the behaviors of the controller under heavy traffic.

Theorem 2: The inequality $\dot{y}(t) < \frac{1}{x_1(t)}[x_2(t)(c(t) - x_1(t) - y_s - v(t)) - \varepsilon]$ *if* $x_1(t) > 0$ in (3-12) corresponds to the heavy traffic scenario in a router where the IntelRate controller decreases the source sending rate according to max-min fairness until the queue size reaches the TBO q_0 so that the asymptotical stability of the IntelRate control system is guaranteed.

***Proof*:** Please see Appendix C.2 for details.

3.2.5 Global Asymptotical Stability

Finally, the only special case to analyze is when $x_1(t) = 0$, i.e., the incoming rate $y(t)$ to the router remains constant at y_s. We consider this as a heavy traffic scenario because the bandwidth has been fully utilized, i.e., $y(t)=c(t)$. From $\dot{V}(\boldsymbol{x}, x_1(t)) = h(x_2, \cdot$ $+x_2(t)(x_1(t) + y_s + v(t) - c(t)))$, we note that $\dot{V}(\boldsymbol{x}) = 0$ only when $\boldsymbol{x} = 0$. Furthermore, as $|\boldsymbol{x}| \to \infty$, $V(\boldsymbol{x}) = (x_1^2(t) + x_2^2(t))/2 \to \infty$. Therefore, from the stability principle of LDM, along with the established Theorem 1 and Theorem 2 above, we can now conclude that the IntelRate control system is globally asymptotically stable given any initial state of ($y(t)$, $q(t)$).

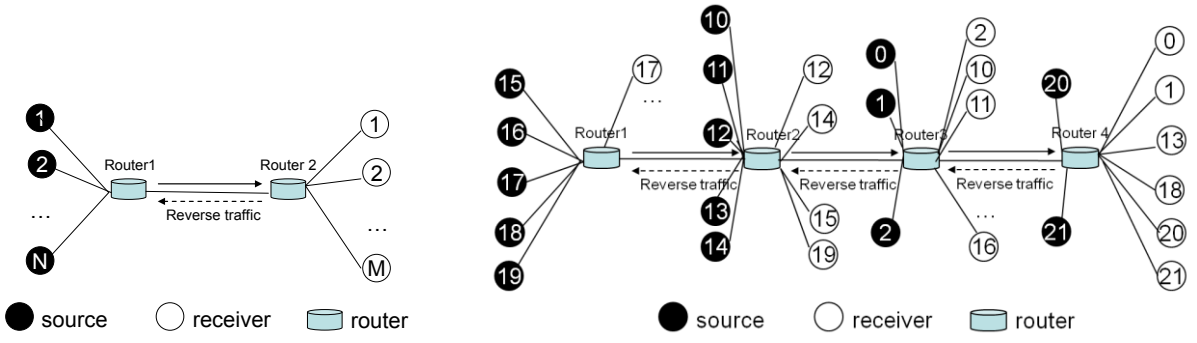
Please note that the global stability stated here is based on the notion that the initial state ($y(t)$, $q(t)$) should be compliant with the system physical limitations, e.g., $q(t)$ should neither be a negative value nor be greater than the buffer capacity B. Also, as a reference value, q_0 is fixed in a link and does not vary with respect to time, which is the base of the above stability proof.

3.3 Performance Evaluation

In this section, the capability of our IntelRate controller is demonstrated by performance evaluations through a series of experiments.

Section 3.3.1 describes simulation settings. Sections 3.3.2 and 3.3.3 demonstrate the IntelRate controller performance under light and heavy traffic scenarios. Sections 3.3.4 and 3.3.5 separately show the system robustness and queueing jitter control. The IntelRate controller performance in a multiple bottleneck network is illustrated in Section 3.3.6. In Section 3.3.7, we compare with other existing controllers.

For the following evaluation, we choose N=9, m=8 and the delay of TBO≤10ms, while B=10*TBO and d=50ms using the design principles for the IntelRate controller (to see in our design guideline discussion in Section 6.1.1). As noted there, some of these "optimum/good" values are the result after many experiments on different combinations, which will be further experimentally investigated in Appendix F.



(a) Single Bottleneck Network **(b) Multi-Bottleneck Network**

Fig. 3.6: Network Simulation Topologies

3.3.1 Simulated Network

Two case studies (i.e., a single bottleneck topology in Fig. 3.6a and a multi-bottleneck topology in Fig. 3.6b) are conducted by OPNET simulations. The single bottleneck is

actually a special case of multiple bottleneck networks, and has been the subject of many analyses, e.g., [KaHa02, DuMc06, GuYa07, HoYa07, JiCh10]. Therefore, it plays an important role as a reference and for comparison with other controllers. Below, we also list the network parameters and their settings. Except the multiple bottleneck performance experiment in Section 3.3.6 which uses multi-bottleneck topology, the experiments from Section 3.3.2 to 3.3.5 and Section 3.3.7 all use the single bottleneck topology.

3.3.1.1 Single Bottleneck Network

The single bottleneck network in Fig. 3.6a is used to investigate the controller behavior of the most congested router. We choose Router 1 as the only bottleneck in the network, whereas Router 2 is configured to have sufficiently high service rate and big buffer B so that congestion never happens there.

Table 3.2: Sources Characteristics in Single Bottleneck Network

Subnet ID	Source ID	Flow NO.	RTPD(ms)
ftp group 1	1-20	ftp 1-20	80
ftp group 2	21-40	ftp 21-40	120
ftp group 3	41-60	ftp 41-60	160
ftp group 4	61-80	ftp 61-80	200
ftp group 5	81-100	ftp 81-100	240
http group 1	101-120	http 1-20	80
http group 2	120-140	http 21-40	120
http group 3	141-160	http 41-60	160
http group 4	161-180	http 61-80	200
http group 5	181-200	http 81-100	240
uncontrolled ftp	201	UDP 1	160

The numbers marked on the nodes in Fig. 3.6a designate the subnets attached to each router. Table 3.2 shows the sources characteristics. There are M=N=11 subnet pairs, which form the source-destination data flows in the network, and they run various Internet applications such as the long-lived ftp, short-lived http, or the unresponsive UDP-like flows (also called uncontrolled ftp flows [HoYa07]). Since the link bandwidth we want to simulate have a magnitude of Giga bits per second, we need to use 20 flows (which are not shown in Fig.3.6a for brevity) in each subnet to generate enough traffic to produce congestion. All

flows within each group are identical with the same RTPD (as seen in Table 3.2) and behavior, but different from other groups. The RTPD includes the forward path propagation delay and the feedback propagation delay, but does not include the queueing delay, which may vary according to our settings of TBO size in the experiments. The reverse traffic is generated by the destinations which send the ACK packets back to the sources.

The TBO and the buffer capacity *B* in Router 1 in each experiment are set according to the approaches discussed in Section 6.1.1. We also adopt some typical values from the experiments of existing works so that we can make meaningful comparisons later on. In particular, all the ftp packets have the same size of 1024 bytes (i.e., 8192 bits) [HoYa07] while the http packet size is uniformly distributed in the range of [800, 1300] bytes [ZhLe06].

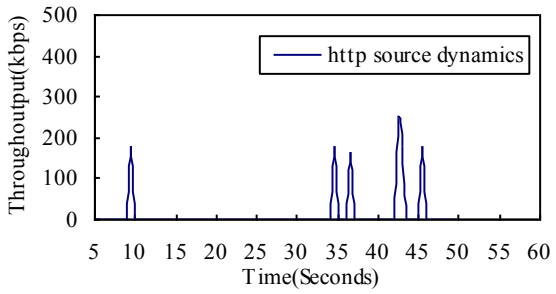
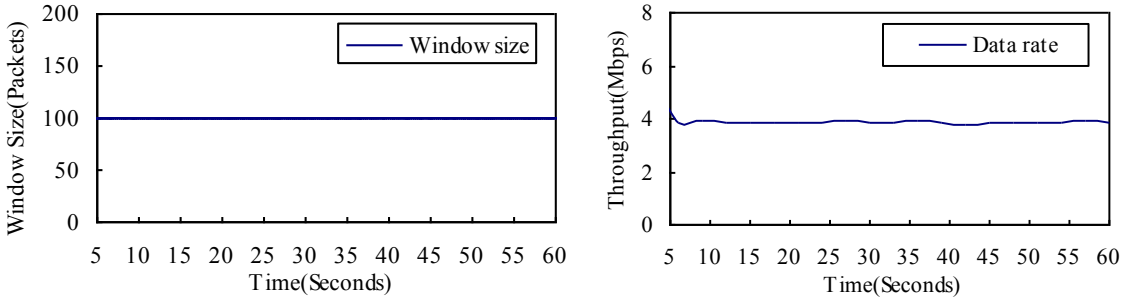


Fig. 3.7: Http Sessions Example



(a) Window Size **(b) Data Rate**

Fig. 3.8: Uncontrolled ftp Flows

In order to demonstrate and discuss the robustness of our IntelRate controller, our experiments would focus on the testing of the 100 long-lived ftp flows, unless otherwise stated. The 100 sporadic short-lived http flows just act as the disturbance to the ftp traffic and their transfer size follows the real web traffic scenario; it has a Pareto distribution [CrBe97] with a mean transfer size of 30 packets [KaHa02]. The arrivals of http flows follow a

think-time [CrBe97] uniformly distributed in [0.1s, 30s]. One of the http session examples is shown in Fig. 3.7. The uncontrolled ftp flow keeps using a window size of 100 packets and operates at an almost constant rate as shown in Fig. 3.8. These http and UDP-like flows generate an aggregate traffic $v(t)$ as discussed in Chapter 2.

Table 3.3: Sources Characteristics in Multiple Bottleneck Network

Router No.	Subnet/Group ID	Source ID	Flow No.	RTPD(ms)
1	0	1-20	ftp 1-20	100
	1	21-40	ftp 21-40	140
	2	41-60	UDP 41-60	160
2	10	1-20	ftp 1-20	140
	11	21-40	ftp 21-40	176
	12	41-60	ftp 41-60	80
	13	61-80	ftp 61-80	240
	14	81-100	ftp 81-100	130
3	15	1-20	ftp 1-20	50
	16	21-40	ftp 21-40	170
	17	41-60	ftp 41-60	80
	18	61-80	ftp 61-80	230
	19	81-100	http 81-100	120
4	20	1-20	ftp 1-20	200
	21	21-40	http 21-40	180

3.3.1.2 Multi-Bottleneck Network

In the multiple bottleneck scenario, we set up four bottleneck routers in the network as shown in Fig. 3.6b. The numbers marked on the nodes have the same notion as in the single bottleneck network. For example, the 20 flows in Subnet 16 will go through Routers 1, 2 and 3 before they reach their destinations. Note that we do not number the subnets consecutively (i.e., all the way from 1 to 21) in Fig. 3.6b to allow for future network extension. Table 3.3 tabulates the flow characteristics for the multiple bottleneck network.

3.3.1.3 Other Settings and Performance Measures

For both the single bottleneck scenario and the multiple-bottleneck scenario, our experiments were conducted using OPNET Modeler 11.5 [Opnet05] on an Intel Core TM2 Quad platform

with 2.40GHz processor. Typical simulated time is set according to the specific experiment scenario. The simulation time depends on bottleneck bandwidth and the simulated time. A typical simulation run usually takes hours or days. For example, in order to observe the source throughput behavior before and after the network parameter changes, we set a longer simulated time for such an experiment than a max-min fairness experiment. The number of packets generated in an experiment is related to the TBO value, the bandwidth, the simulated time and the traffic intensity. The controller is evaluated by the following performance measures.

1) Source throughput (or source sending rate) is defined to be the number of bits (or packets) successfully sent out by a source per second, i.e. bits/second (or packets/second) [Opnet05]. Later we will use throughput and sending rate interchangeably. As mentioned above, the fixed packet size (i.e., 8192 bits) is used in our experiments, the conversion between bits/second and packets/second is just a constant of 8192 or 1/8192.
2) Average source throughput is defined to be the average number of bits successfully sent out by a source averaged over the simulated time [Opnet05].
3) IQSize is the length of the bottleneck buffer queue (in packets), which is measured on packet departures in our experiments [GrSh08].
4) Queueing delay is the waiting time of a packet in the router queue before its service. Measurements are taken from the time the first bit of a packet is received at the queue until the time the first bit of the packet is transmitted. As noted in Section 2.1, queue length and therefore the queueing delay are the results of the arrival rate and departure rates to be controlled by the proposed controller. They can be measured to gauge the success of our control algorithms.
5) Queueing jitter is the variation of queueing delay due to the queue length dynamics, and is defined as the variance of the queuing delay.
6) Utilization is the ratio between the current actual throughput in the bottleneck and the maximum service rate of the bottleneck. It is expressed as a fraction less than 1 or as a percentage.
7) Packet loss rate is defined as the ratio between the number of packet dropped and the number of total packets received per second by the bottleneck.
8) Max-min fairness: A feasible allocation of rates is ‘max-min fair’ if and only if an increase of any rate within the domain of feasible allocations must be at the cost of a

decrease of some already smaller or equal rates [Welz05].

9) Robustness: a system is robust when it exhibits desired performance despite the presence of significant process uncertainty like model changes or disturbances [DoBi08].

For performance measures 1) and 2), a bit is considered successfully sent if it is part of a packet that is successfully sent out [Opnet05].

Since the behavior and performance of the sources within each group are quite similar, in the following experiments we shall show the results of one source from each ftp group or http group for brevity reason. Also, some of the following performance discussion will not involve the http and UDP traffics although they are also part of the incoming traffics. This is because these traffics (as shown in Fig. 3.7 and Fig. 3.8) are negligible compared with the big shares of the ftp flows in the link. Doing so is to rigorously test our controller with these greedy ftp flows so that effectiveness of the controller can be verified under the worst traffic scenario. As such, the occasional bursty traffic is only a special case in our test scenario.

3.3.2 Light Traffic Scenario in Single Bottleneck Network

The light traffic scenario is used to demonstrate that the IntelRate controller can maintain system stability by allowing all flows to send data according to their desired rates, as analyzed in Section 3.2.3. The bandwidth of Router 1 is 5Gbps. The TBO of the bottleneck in this application example is 6000 packets, which can cause a queueing delay of 9.83ms (calculated by 6000 packets*1024bytes*8bits/5Gbps). The buffer capacity B=10*TBO. The desired sending rates from ftp Group 1 to Group 5 are 6.14Mbps, 10.08Mbps, 20.15Mbps, 41.78Mbps and 76.19Mbps, respectively. Note that the total desired rate is less than the bottleneck bandwidth, i.e., 5Gbps, in order not to cause congestion in the bottleneck.

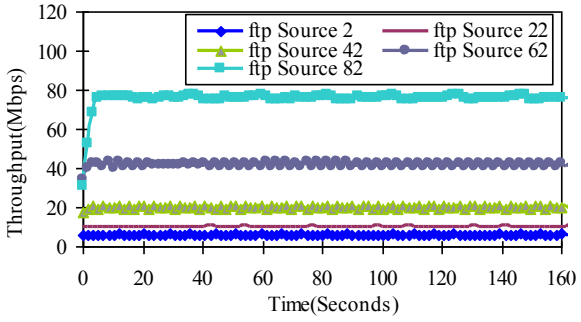


Fig. 3.9: Sources Throughput Dynamics under Light Traffic

Fig. 3.9 shows the throughput of 5 ftp sources, one from each ftp group. In comparison

to their desired rate we set above, one can see that all the flows stably send data in a rate they desired, and this verifies our system stability principle stated in Theorem 1 of Section 3.2.3. Further discussion will be provided shortly.

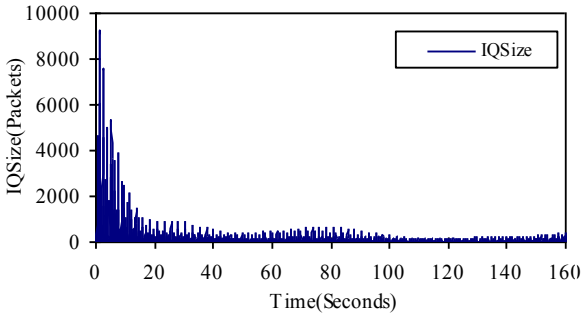


Fig. 3.10: Router IQSize under Light Traffic

Fig. 3.10 shows the router IQSize dynamics. Compared with the TBO value of 6000 packets, the IQSize operates in a very low buffer level most of the time. After the router is started, the controller is trying to increase the queue size to the TBO of 6000 packets. However, because of the total incoming traffic less than the bottleneck bandwidth, it has to settle at a lower level eventually. This indicates that, in a light traffic condition, the IQSize may not reach the TBO value. Since the router is not congested, this does not cause the system instability. As a matter of fact, it is queue overflow that would cause congestion so the system instability, which will be investigated in the heavy traffic scenario later on.

Discussion: It is interesting to review once more the fuzzy logic mechanism that allows the IntelRate sources to send data with their desired rate in light traffic scenario. As shown in Fig. 3.10, when the IQSize $q(t)$ is less than TBO value of q_0, $e(t)$ is always positive (corresponding to the LV "ZR" or "PS" of $e(t)$ in Fig. 3.3). Consequently, $g(t)$ is increasing until it reaches the maximum edge value mq_0 (corresponding to the "PV" of $g(t)$). According to Table 3.1, the system is now working under the rule (ZR, PV, MX) or (PS, PV, MX). Either one means the IntelRate controller allows the "MX" sending rate, where the value D resides. Therefore, every source can send data with its desired rate.

Unlike the heavy traffic scenario to be discussed below, we have no max-min fairness issue here. This is because the max-min fairness is a kind of resource allocation mechanism for many greedy consumers sharing limited resources. This fairness works by first satisfying the small users, and then evenly allocating the remaining resources among the large users (as

will be illustrated in the next section).

3.3.3 Heavy Traffic Scenario in Single Bottleneck Network

In this section, we want to demonstrate that under heavy traffic the IntelRate controller can maintain system stability by allocating the bottleneck bandwidth according to max-min fairness, as analyzed in Theorem 2 of Section 3.2.4. The experiment is conducted with the similar settings as those in Section 3.3.2. The difference is that the ftp flows in group 1 to group 5 would desire higher sending rates, i.e., 24.48 Mbps, 33.18 Mbps, 49.15 Mbps, 90.11Mbps and 114.69Mbps respectively, which are the rates we use to produce congestion.

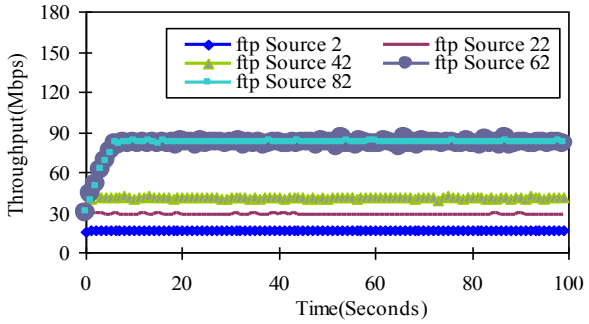


Fig. 3.11: Source Throughput Dynamics under Heavy Traffic

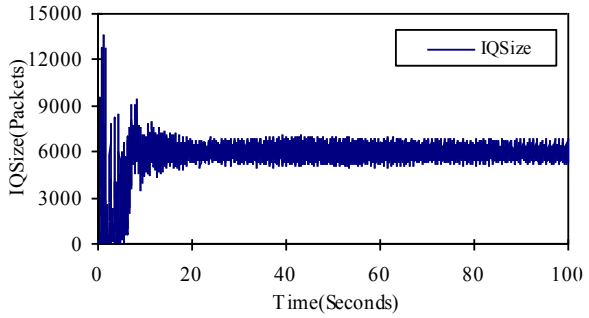


Fig. 3.12: Router IQSize under Heavy Traffic

Fig. 3.11 shows the stable sending rates of 5 sources, one from each ftp group. One can see that the sources in group 1, 2 and 3 (e.g. sources 2, 22 and 42) obtain a throughput they desire, i.e. 24.48 Mbps, 33.18 Mbps and 49.15 Mbps, respectively. However, the sources in group 4 and 5 (e.g. sources 62 and 82) cannot obtain their desired rates 90.11Mbps and 114.69Mbps respectively. Instead they have to share the same portion of bandwidth, i.e., 82.45Mbps. This illustrates and verifies the max-min fairness capability we implemented for the IntelRate controller.

Fig. 3.12 shows the router IQSize is well controlled around the TBO value of 6000 packets. Even though the queue size may rise sharply at the starting stage, but its level is much smaller than the buffer size of 60000 packets. This demonstrates that the IntelRate controller is capable of restricting the IQSize around the TBO so that the buffer is not overflowed and the system is stable. This also leads to another advantage observed in our controller, i.e., there is no packet loss upon heavy traffic.

***Discussion*:** Now we analyze how the IntelRate controller imposes max-min fairness under heavy traffic to maintain the system stable. As mentioned before, the system is stable under heavy traffic situations because the controller is capable of controlling the IQSize around the TBO position. This means that $e(t)$ is hovering around zero depending on the relative values of $q(t)$ and q_0. Consequently, the input $g(t)$ decreases or increases its crisp value. According to Table 3.1, the router outputs an allowed sending rate $u(t)$ for each flow according to the crisp values of $e(t)$ and $g(t)$. The flows whose desired rates larger than $u(t)$ (e.g. the above flow 62 and 82) will be limited to $u(t)$, while those with smaller desired rates are allowed to send data with the rates they desire (e.g. flows 2, 22 and 42 above). This is exactly the max-min fairness the IntelRate controller was designed for.

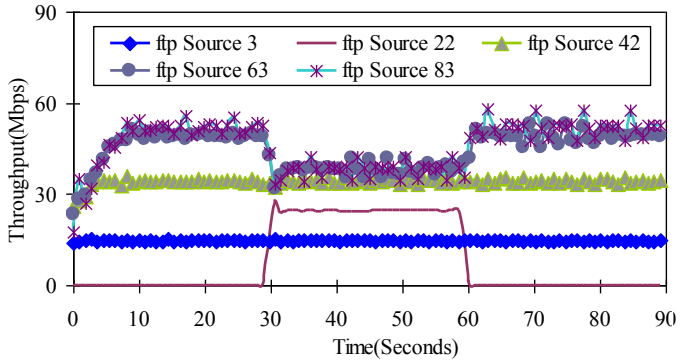
3.3.4 Robustness to Network Parameter Changes in Single Bottleneck Network

The purpose of the experiments is to test the robustness of the IntelRate controller upon the sudden network changes, e.g., the sudden traffic changes or link bandwidth variations. Network traffics can surge/drop when some flows join/leave a link simultaneously. For example, a group of flows in a failed link has to be transferred to another operating link [KeRa10]. The link capacity changes can take place either in a multi-access network due to traffic contention or interferences such as those found in shared Ethernet or IEEE 802.11 wireless network [ZhHe05]. In this section, we will show how the IntelRate controller is capable of dealing with such disruptive network dynamics.

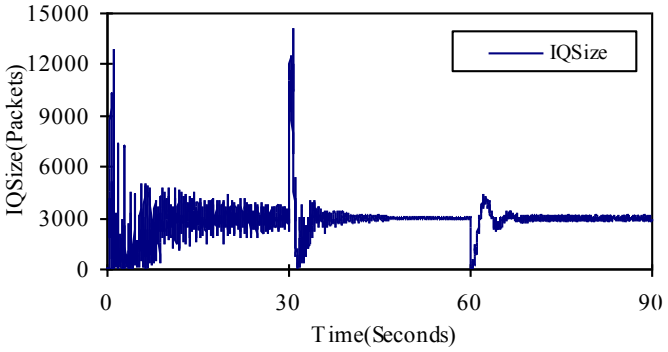
3.3.4.1 Sudden Traffic Changes

Sudden traffic change may occur when the number of long-lived or short-lived flows, or some uncontrolled (unresponsive) flows change at the ingress/egress. To test the controller capability to the dynamics of the real Internet traffic, we conduct an experiment in which the bottleneck bandwidth is 3Gbps. TBO is set to be 3000 packets, which causes queueing delay of about 8.19ms when the queue size stabilizes at TBO (i.e., 3000*1024*8/3Gbps=8.19ms).

The buffer capacity *B*=30000 packets. The source desired rates of ftp groups 1 to 5 are 14.75 Mbps, 24.55 Mbps, 34.41 Mbps, 49.15 Mbps and 65.54Mbps, respectively. During the first 30 seconds, only ftp flows in groups 1, 3, 4 and 5 are active, and at t=30s, 20 ftp flows in group 2 join in the bottleneck traffic and then drop out at t=60s. The objective of this experiment is to see how our controller responds to traffic changes and if the system can remain stable.



(a) Source Throughput



(b) IQSize

Fig. 3.13: Source and IQSize Dynamics under Traffic Change

Fig. 3.13a shows the throughput dynamics of 5 representative ftp sources. The sources in groups 1 and 3 (e.g. sources 3 and 42) can maintain relatively constant throughput. Sources in groups 4 and 5 (e.g., sources 63 and 83) have high sending rates at the beginning, but have to decrease their throughput by about 12Mbps at t=30s, to accommodate the newly joined flows from group 2. Only after the flows of group 2 finish their transmissions at t=60s, they recover their original transmission rates. Our results demonstrate that the IntelRate controller is working with max-min fairness, i.e. the bandwidth to accommodate the new flows is first squeezed from the big users (e.g., sources 63 and 83). As for the response time in this

experiment, from Fig. 3.13a, it takes about 0.9 seconds for the flows to decease their sending rate upon new traffic arrival at t=30s, and then takes the same time to recover after the new traffic withdraws.

The changes in traffic load are well reflected in the dynamics of the router queue size. Fig. 3.13b shows that when the ftp flows in group 2 join in (at t=30s) and leave (at t=60s) the bottleneck traffic, the queue size increases and decreases with some fluctuations. In particular, the swarm-in of the 20 flows at t=30s causes a sharp rise in the queue size, but it does not cause the queue overflow because the maximum queue size of 14000 packets at t=30s just reach half the buffer size of 30000 packets. Besides, we observe that the IQSize in both cases can settle back to the TBO of 6000 packets only within one oscillation cycle, thus demonstrating good stability and robustness of the IntelRate control system upon traffic changes.

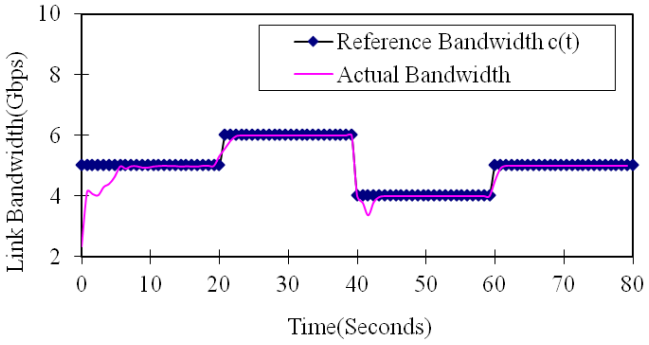


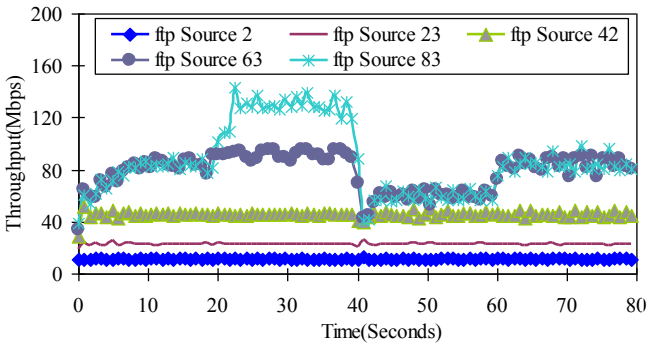
Fig. 3.14: Bandwidth Variations

3.3.4.2 Sudden Bandwidth Changes

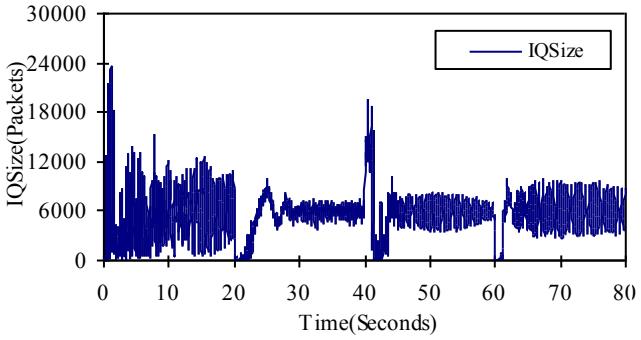
Since a deterministic amount of bandwidth is usually not available in contention-based multi-access networks or wireless networks, their bottleneck bandwidth can change suddenly [ZhHe05]. To demonstrate how our IntelRate controller can be robust to link bandwidth dynamics, we start an experiment with a bandwidth of $c(t)$=5Gbps, q_0=6000 packets and B=60000 packets in the bottleneck. The source desired rates of ftp groups 1 to 5 are 11.47Mbps, 22.94Mbps, 45.88Mbps, 91.75Mbps and 131.07 Mbps, respectively.

In Fig. 3.14, the reference bandwidth represents the amount of bandwidth $c(t)$ available for all traffic flows in the control system while the actual bandwidth is the total amount of sending rates computed by the router as time evolves in order to fully utilize the reference bandwidth. One can see the actual bandwidth closely follows the reference bandwidth except for the first 5.5s of the warm-up stage. The biggest mismatch happens at t=40s when the

bandwidth drops to 4Gbps from 6Gbps (a 33.33% bandwidth drop). This is because the IntelRate controller has to quickly reduce the rates of the big flows in response to the queue size surge, as will be discussed later.



(a) Source Throughput



(b) IQSize

Fig. 3.15: Source and IQSize Dynamics under Bandwidth Change

Fig. 3.15a shows the throughput dynamics of 5 ftp sources. Similar to the traffic change scenario, the sources are working with the max-min fairness and the flows who desire higher sending rate adapt their throughput to the time-varying bandwidth. For example, when the bottleneck bandwidth increases from 5Gbps to 6Gbps at t=20s the sending rates of sources in ftp groups 1, 2, 3 and 4 (e.g., sources 2, 23, 42 and 63) remain unchanged, i.e., they maintain their throughput at their desired rate 11.47Mbps, 22.94Mbps, 45.88Mbps and 91.75Mpbs. However, the sending rate of sources in ftp groups 5 (e.g., source 83) increases to 130.38Mbps from 91.75Mbps responding to the enlarged link capacity. The similar trend can be seen after t=60s when the bandwidth increases to 5Gbps from 4Gbps in that the greedier flows in groups 4 and 5 increase their throughput and the flows in other groups are not affected. When the available bandwidth suddenly shrinks at t=40s from 6Gbps to 4Gbps, the sources in group 4 and 5 (e.g., the sources 63 and 83) perform rate reduction accordingly and

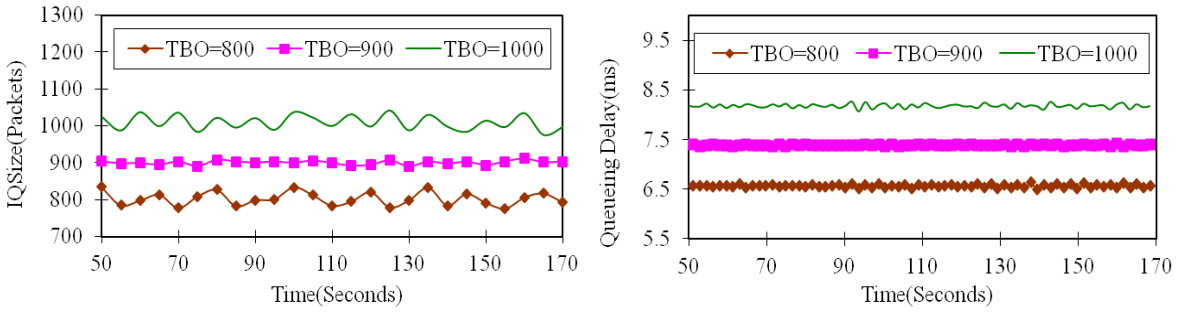
share the same sending rate of 52.37Mbps after stabilized. As with the bandwidth increase, the less greedy flows in group 1, 2 and 3 do not show throughput variation.

Fig. 3.15b shows that the link bandwidth variations are also well reflected by the IQSize dynamics. Queue size fluctuates upon bandwidth changes but it can always tend to settle back to the TBO of 6000 packets, thus showing good stability and robustness of the IntelRate controller to sudden bandwidth changes. For example, when the bandwidth increases from 5Gbps to 6Gbps at t=20s, the IQSize decreases temporarily due to the 1Gbps newly available bandwidth, but is regulated back to the TBO position. In contrast, at t=40s, when the bandwidth decreases from 6Gbps to 4Gbps, the IQSize increases to 18000 packets responding to the sudden loss of 2Gbps bandwidth, but settles down in one oscillation cycle. In addition, in all transients, the IQSize is much lower than the buffer size of 60000 packets and thus there is no packet loss during all the bandwidth variations.

It is interesting to observe that this sudden bandwidth change experiment is equivalent to our traditional test where there is uncontrolled traffic such as UDP that takes away a chunk of bandwidth all of sudden from the link.

3.3.5 Queueing Jitter Control in Single Bottleneck Network

One main source of the network latency oscillations comes from the dynamics of queueing delay in the routers. In most cases, whenever the queue size is out of control or oscillating too much, the queueing jitter performance degrades. In this experiment, we want to check under heavy traffic conditions how the queueing delay fluctuates under the different TBOs in the IntelRate controller, and how big the queueing jitters can be. The experiment is conducted in a 1Gbps single bottleneck network with B=10000 packets.



(a) IQSize **(b) Queueing Delay**

Fig. 3.16: IQSize and Queueing Delay under Different TBOs

Fig. 3.16 illustrates the queueing jitter performance under the steady state of the IntelRate controller. Fig. 3.16a shows the IQSize under TBO=800 packets, 900 packets and 1000 packets. Fig. 3.16b shows the corresponding queueing delays. One can see the queueing delay is well controlled to a certain level with respect to different TBOs. For example, the TBO of 800 packets produces a queueing delay of about 6.5ms. The queueing jitters obtained from these measured queueing delays under the three TBOs are 0.171×10^{-6}, 8.738×10^{-6} and 5.043×10^{-6}, respectively. The very small jitters are due to the capability of the IntelRate controller to control its queue size around the TBO, and thus controlling the oscillations of the queuing delay to a very low level.

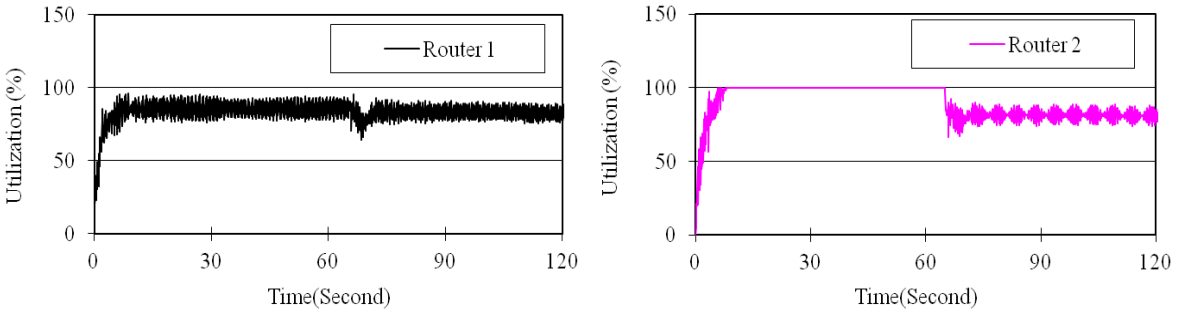
3.3.6 Multi-Bottleneck Performance

The purpose of this experiment is to observe the response and behavior of sources and routers in a multiple bottleneck network upon bottleneck switching. Bottleneck switching means the Internet congestion may shift from one router to another (e.g., which can happen from time to time due to the network traffic dynamics).

The experiment is conducted with the network depicted in Fig. 3.6b. To produce a drastic bottleneck switching, at t=65s, we will have the 20 flows of subnet 12 attached to Router 2 simultaneously retreat from the network while 20 flows of subnet 20 attached to Router 4 join in the network at the same time. With such a scenario, we expect a sudden bottleneck switching from Router 2 to Router 4 during the simulation. To make the experiment more complicated, we add a traffic change during the simulation by having Source 8 from Subnet 18 operating only between t=70s and t=110s.

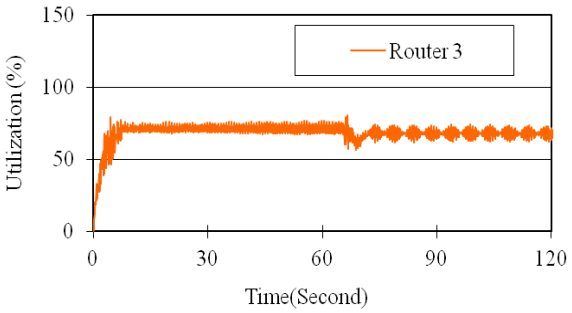
The bandwidth from Router 1 to 4 is 1.38Gbps, 2.46Gbps, 3.05Gbps and 1.60Gbps, respectively. Their TBOs are separately set to be 1300 packets, 2500 packets, 3000 packets and 1500 packets. Upon heavy traffic, these TBOs may cause a queueing delay of 7.71ms, 8.33ms, 8.06ms and 7.68ms in the router it resides in. The buffer size of the four routers is set to be 10 times big of their TBO as mentioned in our design, e.g., Router 1 has a buffer size of 13000 packets. For brevity, we shall omit listing the source desired rates of all the subnets, and mention them wherever needed.

First, by checking the link utilization, we will see how the bottleneck is being switched with the above settings, or say, how the bottleneck switching happens between Router 2 and Router 4 as expected.

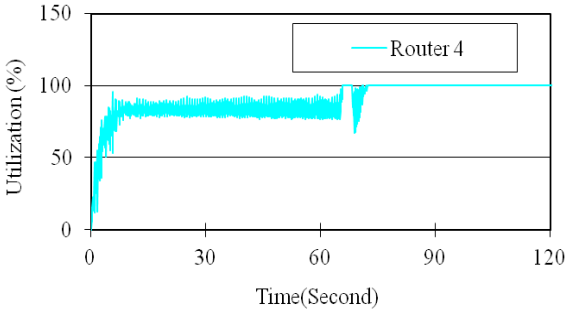


(a) Utilization of Router 1

(b) Utilization of Router 2



(c) Utilization of Router 3



(d) Utilization of Router 4

Fig. 3.17: Link Utilization of Each Router

Fig. 3.17 depicts the link utilization in the four routers. Before t=65s, they separately have a utilization of 89%, 100%, 72% and 82%, which means the bottleneck happens at Router 2 of the network, whereas other routers work under light traffic situation. After t=65s, their utilizations become 83%, 81%, 70% and 100%, which means the bottleneck has shifted to Router 4 from Router 2 due to the withdrawal and joining of the traffics configured with the above settings. In addition, we also find that Router 1 and Router 3 slightly undergo lower utilizations after the bottleneck switching, and the reason will be evident after we analyze the source behaviors attached to these routers later.

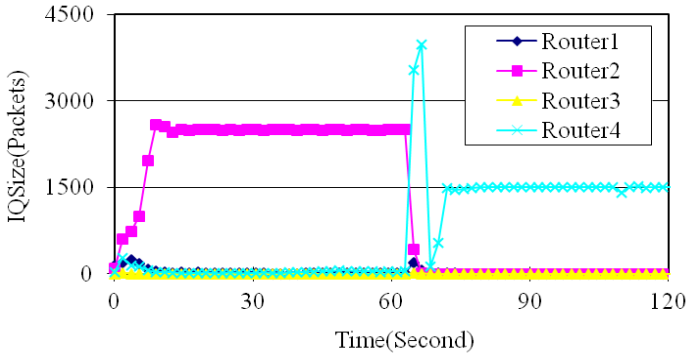


Fig. 3.18: IQSize of Each Router

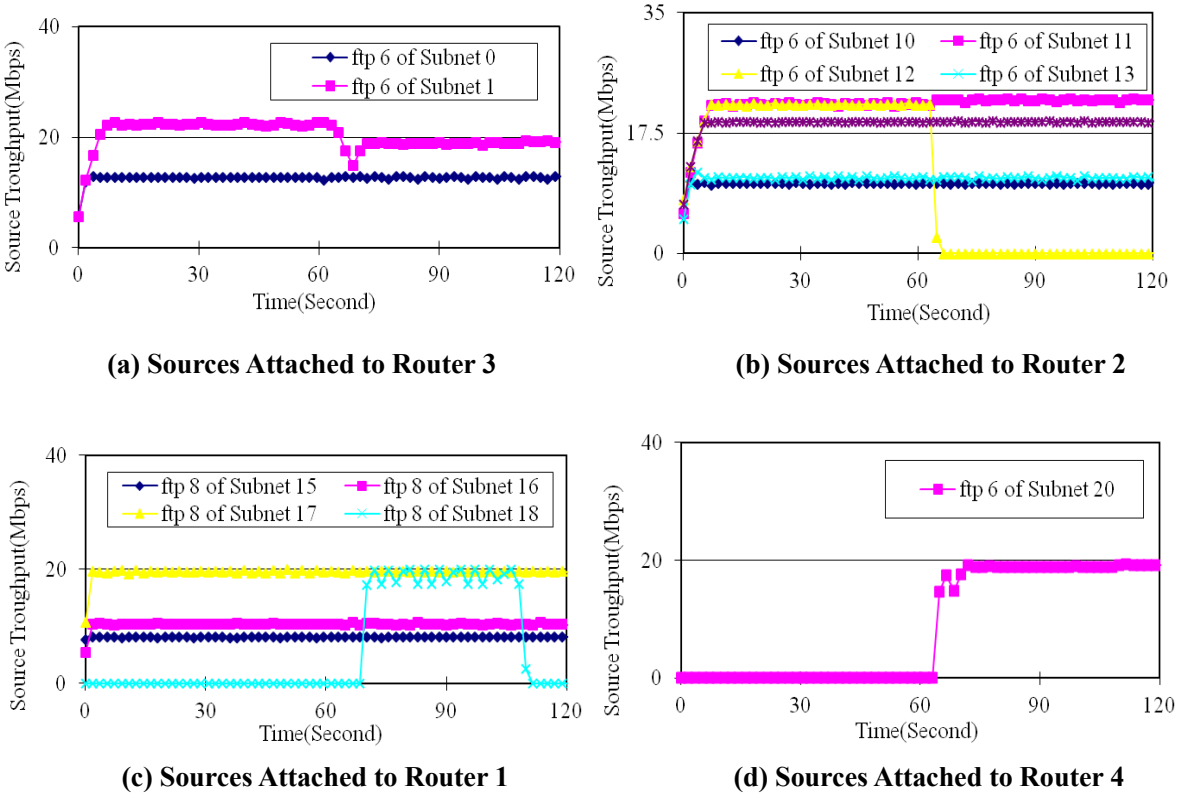
The corresponding queue size performance of the 4 routers is plotted in Fig. 3.18. Before t=65s, the queue size of Router 1, 3 and 4 are kept in operation near zero, while the queue size of Router 2 is stabilized at 2500 packets, which is exactly the TBO we set in Router 2. The near-zero queue size performance of Router 1, 3 and 4 before the bottleneck switching also implies these routers are working in light traffic situation as discussed in Fig. 3.17a, Fig. 3.17c and Fig. 3.17 d.

After t=65s, the queue size of Router 2 plummets to zero from 2500 packets, while that of Router 4 increases to its TBO position of 1500 packets from zero. Note that even though the oscillation in Router 4 first shoots up to about 4000 packets at t=65s, it is much lower than the buffer capacity of 15000 packets, so it does not cause the packet loss at all. Router 1 and 3 remain their queue size to near zero position all the time. Such drastic transitions in the queue size of Router 2 and Router 4 is caused by the swarm-in or the swarm-out of the flows we set above in the two routers. The transitions reflect how the IntelRate controller responds to bottleneck switching to prevent congestion, i.e., the IntelRate controller just simply rises to control the router queue size to be operating around the TBO upon the congestion presence. For the routers which work in light traffic situations, they leave their queue size operate near zero level.

The source behaviors of one representative flow from each subnet are illustrated in Fig. 3.19. Fig. 3.19a shows the source behaviors in Subnet 0 and Subnet 1, which are attached to Router 3. Before t=65s, all the sources in the two subnets send data with their desired rate. After t=65s, the source sending rate of Subnet 0 remains the same irrespective of the bottleneck switch, i.e., 12.74Mbps, whereas due to the max-min fairness the sources in Subnet 1 undergo the sending rate decrease, which first drops to around 13.37Mbps from 22.35Mbps and eventually stabilizes at 19.62Mbps.

Fig. 3.19b is the source behavior of Subnet10-14 associated to Router 2. Before t=65s, the sources of Subnet 10, 13 and 14 send their data with their desired sending rates, while the sources of Subnet 11 and 12 send data with the same sending rates, i.e., 21.63Mbps. When the 20 flows of Subnet 12 withdraw from the traffic, the sources in Subnet 11 reach their desired sending rate by slightly increasing their throughput from 21.63Mbps to 22.47Mbps. Thus after the flows of Subnet 12 retreat, all the flows attached to Router 2 are able to

operate with their desired sending rate.



(a) Sources Attached to Router 3

(b) Sources Attached to Router 2

(c) Sources Attached to Router 1

(d) Sources Attached to Router 4

Fig. 3.19: Source Sending Rates

The source behavior of Subnet 15-18 attached to Router 1 is plotted in Fig. 3.19c, in which the sources of Subnet 15-17 have a throughput of their desired rate all the time. After Source 8 of Subnet 18 begins to operate, it sends data with a rate of around 19.62Mbps, which is lower than its desired rate of 23.73Mbps because its sending rate is restricted by the currently bottlenecked Router 4 (Note that Source 8 of Subnet 18 has its destination attached to Router 4).

Finally, Fig. 3.19d is the source throughput from Subnet 20 attached to Router 4. After the 20 sources of Subnet 20 join in the traffic, they send data with a rate of 19.62Mbps, which is the maximum data rate allowed by the bottleneck Router 4 under the max-min fairness, so they cannot have a desired throughput of 21.62Mbps.

As a summary, when the bottleneck switching happens, the throughput of all the sources behaves sensibly. Some flows may have to yield (e.g., the sources of Subnet 1) such that the new flows can be accommodated into the network, and some flows might be able to increase their throughput (e.g., the sources of Subnet 11) due to the transmission completion of other

flows. On the other hand, it shows how the flow throughput is determined by the most congested link en route (e.g., Source 8 of Subnet 18 has its bottleneck in Router 4).

3.3.7 Comparison with Other Controllers

The existing explicit protocols such as XCP, RCP, JetMax and MaxNet all need to estimate bottleneck bandwidth for computing the source sending rate or link price. As investigated for XCP in [ZhHe05], the potential mis-estimation of bottleneck bandwidth tends to cause throughput convergence problem and instability. Our IntelRate controller does not have this problem (to be demonstrated in Section 3.3.7.1). As discussed before, to estimate the number of flows consumes the CPU and memory resources in the API-RCP controller. In this regard, Section 3.3.7.2 will show that the IntelRate controller, without estimating number of flows, can achieve an equivalent or even better performance than API-RCP upon the traffic change or the bandwidth change. Finally, in Section 3.3.7.3, we show that IntelRate controller can also achieve full link utilization and zero packet loss rate performance, as with other controllers. All the above performance comparisons are carried out under the single bottleneck network as depicted in Fig. 3.6a.

3.3.7.1 Robustness to Bandwidth Variation

We set up a 45Mbps bottleneck and the flows have a common round trip time of 40ms, just as described in XCP [KaHa02]. 100 long-lived ftp flows share the bottleneck and greedily consume it up. To test XCP, at t=20s, we change the bottleneck bandwidth from 45Mbps to 40Mbps to mimic the error in bandwidth estimation. Then we test IntelRate controller with the same bandwidth change.

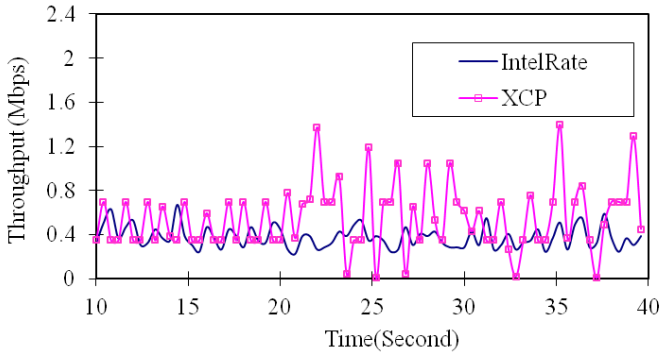
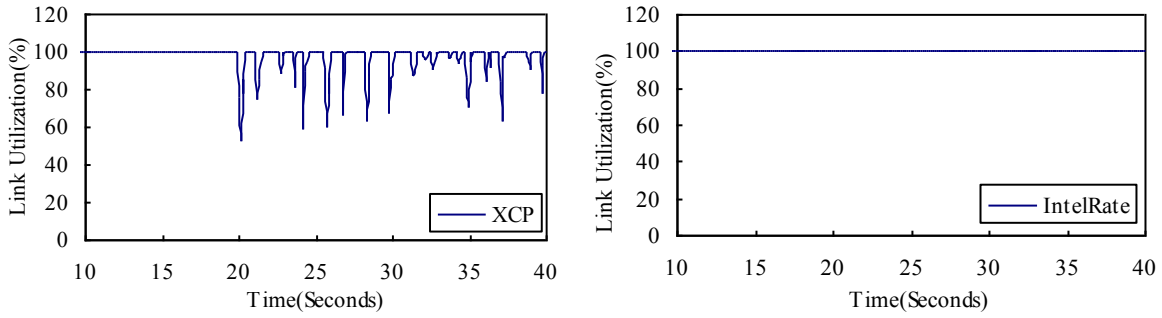


Fig. 3.20: Source Throughput Dynamics Comparison upon Bandwidth Variation

As shown in Fig. 3.20, the throughput of XCP flow becomes oscillating after the bandwidth changes from 45Mbps to 40Mbps at t=20s, and cannot converge to a relatively smooth state. The reason is evident if we check its efficiency equation $\phi = \alpha \cdot (c - y(t)) - \beta \cdot Q / d)$, we will find it depends on the accurate estimation of bottleneck bandwidth *c*. If *c* is somehow mis-estimated like emulated in this experiment, the performance of XCP controller is degraded. On the other hand, our IntelRate controller is based on the heuristic expert knowledge so that the source is able to find a new rule according to the queue size variation, and thus adapting itself to the new bandwidth condition. This is why IntelRate controller can maintain the flow throughput relatively smooth and the system stable under quick fluctuating bandwidth situations.



(a) XCP **(b) The IntelRate Controller**

Fig. 3.21: Link Utilization upon Bandwidth Variations

Fig. 3.21 shows the link utilization of XCP degrades accordingly when its flow rate becomes oscillating, while IntelRate controller keeps full utilization due to its relatively stable sending rate.

3.3.7.2 Response Performance Comparison with API-RCP

API-RCP [HoYa07] depends on an additional mechanism, called self-adaptation, to adapt its PI (Proportional-Integral) coefficients to the new traffic conditions. With such a mechanism, API-RCP can detect and deal with the changes of the number of flows so that the system can keep stable [HoYa07]. However, the IntelRate controller does not need to estimate the number of flows at all, so it has a lower requirement on computational and memory resources than API-RCP while achieving better performances, as will be seen. The

comparisons of the computational intensity and memory requirement with other controllers can be found in Appendix E. In the following, we will demonstrate the superiority of the IntelRate controller through traffic and bandwidth changes experiments.

(1) Response Performance upon Traffic Change

We set up a 45Mbps bottleneck bandwidth with B=1000 packets and TBO=300 packets, as described in [HoYa07]. We set M=N=100 ftp sources. In this experiment, at the first 80 seconds, we only have the 80 ftp flows in group 2, group 3, group 4 and group 5 operating, then the 20 ftp flows in group 1 join in the bottleneck traffic and retreat at t=120s. We test the two controllers with the same scenario successively. The API-RCP is working with a phase margin of 45 degrees (which is considered to be within an optimum range of setting [HoYa07]). In the IntelRate controller experiment, each source desires a greedy sending rate of 0.9Mbps.

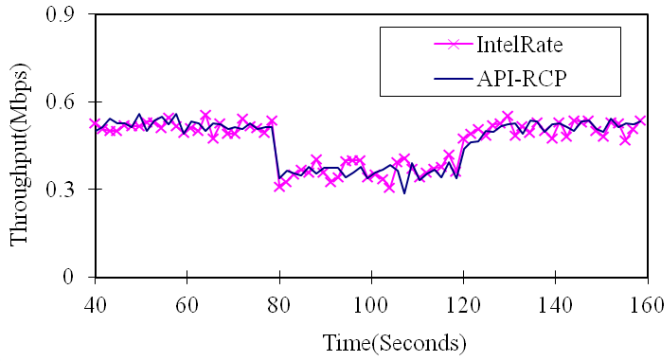
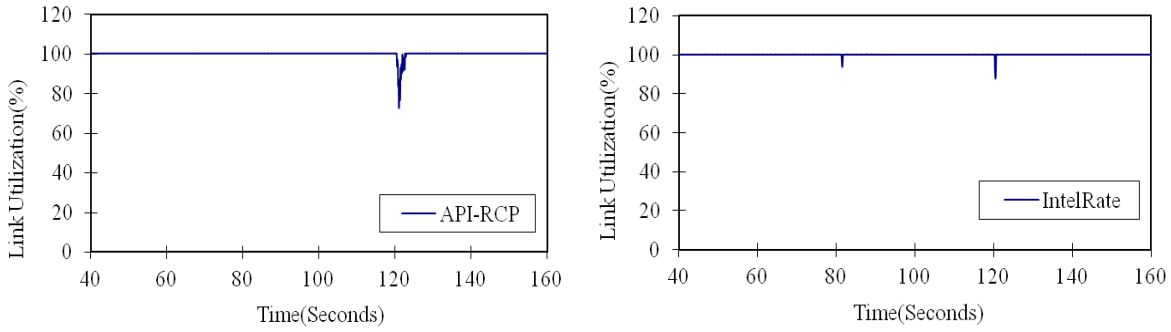


Fig. 3.22: Source Response upon 20 Flows Joining and Retreat

Fig. 3.22 shows the behaviors of the sending rate of the ftp source under the two experiments. The two controllers show very similar response. At t=80s, when the new 20 flows in group 1 join in the traffic, API-RCP spends about 1.5 seconds transitioning to a new sending rate. When the 20 flows in group 1 retreat from the traffic at t=120s, it spends about 3 seconds to recover the sending rate (later on we call this process "recovery stage"). The IntelRate controller renders a similar transitional time during such a process, but shows a slightly higher sending rate than API-RCP in the recovery stage. Although the two appear to be close in Fig. 3.22, the data collected to depict Fig. 3.22 show that the IntelRate has a slightly higher sending rate than API-RCP in the recovery stage (e.g., at t=122s, the sending rate of the API-RCP is 0.460Mbps while that of the IntelRate controller is 0.491Mpbs). This

will become more noticeable when the traffic change is larger, e.g. 40 flows conducted in the next experiment.



(a) API-RCP **(b) The IntelRate Controller**

Fig. 3.23: Link Utilization upon 20 Flows Joining and Retreat

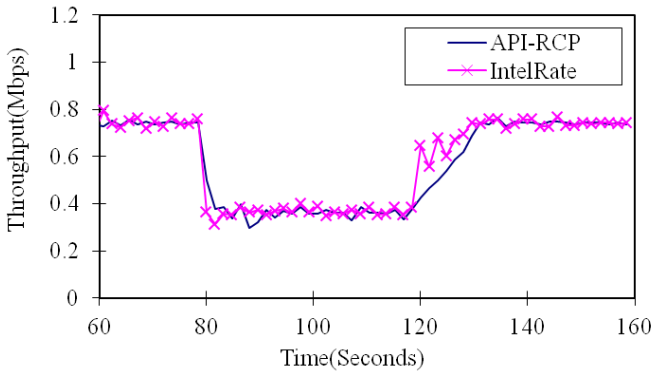
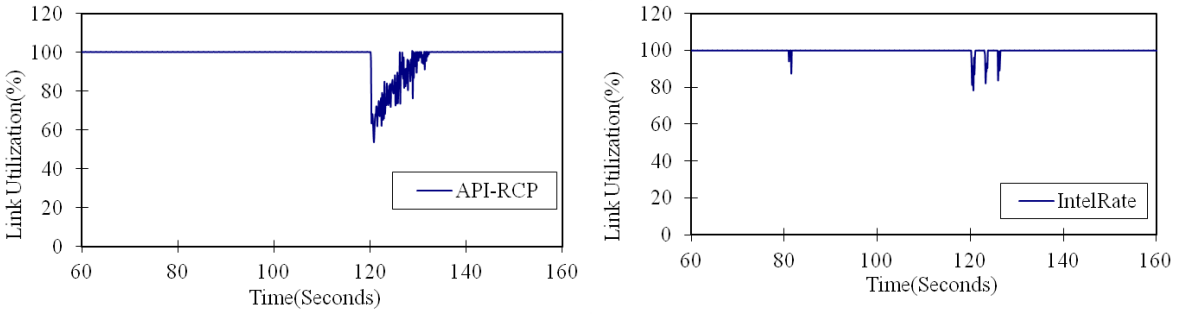


Fig. 3.24: Source Response Upon 40 Flows Joining and Retreat

Fig. 3.23 compares the link utilization of IntelRate and API-RCP during the traffic change. At t=80 seconds, when 20 flows join in the traffic, IntelRate controller has a slight drop in utilization, while API-RCP does not show any impact. However, when the 20 flows depart from the traffic at t=120s, API-RCP shows larger utilization drop than the IntelRate controller. The reason will be discussed in the next experiment based on a more noticeable difference of the source throughput between the two controllers during the recovery stage.

In order to have a more noticeable observation of the difference between the response performances of two controllers, now we get 40 ftp flows to contribute the traffic change. Specifically, at the first 80 seconds, we only allow 60 ftp flows operating from group 2, group 4 and group 5. At t=80s, 40 ftp flows from group 1 and group 3 join in the bottleneck traffic and then retreat at t=120s. Other settings remain the same as in the last experiment. Fig. 3.24 shows the experiment results of the ftp source.

Similar to Fig. 3.22, the time for the two controllers to reduce the source sending rate at t=80s in Fig. 3.24 is basically the same in that the IntelRate controller is just a little faster than API-RCP. However, the source behavior differs noticeably in the recovery stage at t=120s. In detail, the IntelRate controller increases faster transitioning to the new sending rate during the recovery stage, though undergoes a few slight oscillations. However, API-RCP behaves with a gradual pace, though it shows a smooth behavior. Hence, the IntelRate controller shows a fast response performance, which means a more efficient data transmission during the recovery stage.



(a) API-RCP **(b) The IntelRate Controller**

Fig. 3.25: Link Utilization Upon 40 flows joining and retreat

Fig. 3.25 compares the link utilizations for the two controllers upon traffic change of 40 flows. Similar to Fig. 3.23, during the recovery stage of the source sending rate, API-RCP shows a dramatic and longer utilization drop, whereas the IntelRate controller just shows three small and short drops. This phenomenon can be explained from the Fig. 3.24. Since API-RCP increases the source sending rate in a gradual way during the recovery stage, when the 40 flows depart from the traffic, API-RCP would accordingly takes a longer time to fill the spare bandwidth left by those departure flows. In contrast, the IntelRate controller recovers the source sending rate with a higher speed, which makes it faster take up the spare bandwidth. Due to the slight oscillations in the source sending rate during the recovery stage the link utilization in the IntelRate controller needs to undergo couple of oscillations before completely reaching 100%.

In summary, compared with API-RCP, the experimental results in Fig. 3.22-Fig. 3.25 show that, in general, the IntelRate controller achieves better response performance upon traffic changes. In particular, it shows faster response than API-RCP in fully utilizing the spare bandwidth upon traffic change.

(2) Response Performance upon Bandwidth Variation

In what follows, we conduct the response performance comparisons upon bandwidth variations between IntelRate controller and API-RCP. The experiments start from a bottleneck bandwidth of 45Mbps, and will decrease to 30Mbps at t=80s, and increase to 60Mbps at t=100s. Finally it will change back to 45Mbps. We use 100 ftp flows in operation. Other settings are the same as the traffic change experiments above.

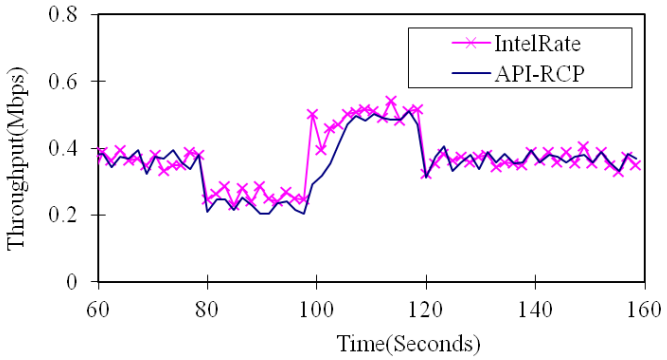
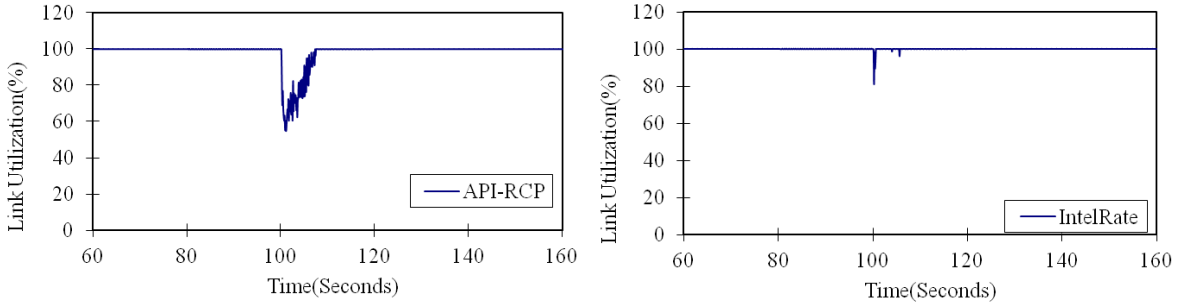


Fig. 3.26: Source Response Upon Bandwidth Variation



(a) API-RCP **(b) The IntelRate Controller**

Fig. 3.27: Link Utilization Upon Bandwidth Variation

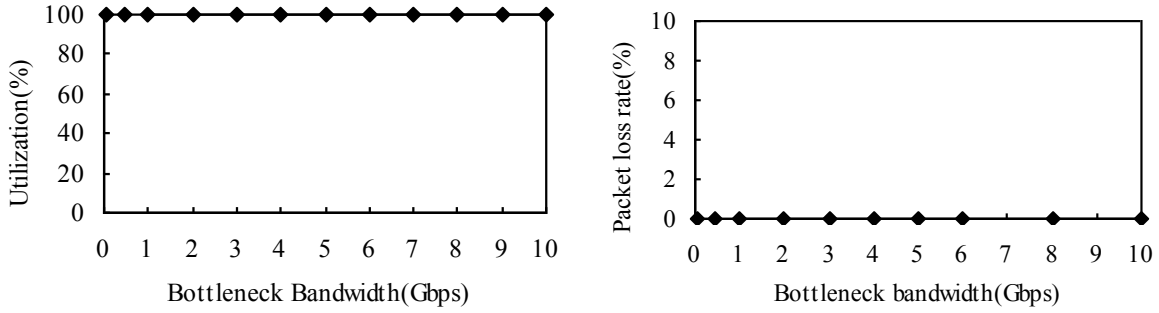
The source behaviors upon sudden bandwidth change are depicted in Fig. 3.26. Similar to the traffic change results above, at t=80s or t=120s, when the bandwidth decreases from 45Mbps to 30Mbps or from 60Mbps to 45Mbps, both controllers take the same time to reduce the source sending rate to a new state. What they differ in the response performance is also at the rate recovery stage from t=100s. When the bandwidth ratchets up from 30Mbps to 60Mbps, the IntelRate controller increases faster than API-RCP, though undergoes a slight oscillation before stabilizing at the new sending rate. In contrast, API-RCP increases the source sending rate still in a gradual way. Therefore, the IntelRate controller also shows a

faster response than API-RCP during the recovery stage.

Fig. 3.27 depicts the link utilization performance upon the bandwidth variations. Like in the traffic change experiment, the IntelRate controller only shows a few slight drops in the link utilization when the bandwidth bounces up from 30Mbps to 60Mbps at t=100s, while API-RCP shows a larger drop. The reason is similar to that in the traffic change. When the bandwidth increases, the spare bandwidth is present. The gradual way to increase the source sending rate hinders API-RCP to fast take up the spare bandwidth, and thus leaves a large dent, as shown in Fig. 3.27a. Nevertheless, the higher source sending rate in the recovery stage in the IntelRate controller makes it grab the available bandwidth faster, thus it can fully utilize the bandwidth in a shorter time, as shown in Fig. 3.27b.

3.3.7.3 Utilization and Packet Loss Rate

The previous protocols such as XCP, RCP, API-RCP or JetMax all have two excellent features, i.e., 100% utilization in general (especially when there is no bandwidth or traffic changes as in the case of XCP) and zero packet loss rate. We also evaluate these performance measures for our IntelRate controller with respect to bottleneck bandwidth or the different settings of TBOs. Since utilization and packet loss rate are tested with the same experiment, we put them in the same subsection. First we vary the bottleneck capacity from 45Mbps to 10Gbps to check the system utilization and packet loss rate.

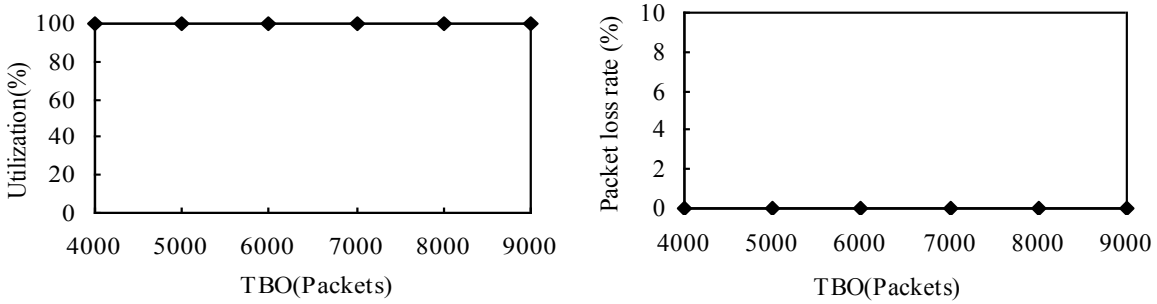


(a) Utilization **(b) Packet Loss Rate**

Fig. 3.28: Utilization and Packet Loss Rate w.r.t Bandwidth

The simulation results are shown in Fig. 3.28. It shows the IntelRate controller is able to maintain the ideal 100% utilization with zero packet loss rate. The reason is because IntelRate controller can control the variations of the IQSize around the TBO position within

25% (observed from our numerous experiments). Therefore, the buffer never overflows and packet never loses, and in the meanwhile, the stable feature in IQSize and throughput guarantees the full bandwidth utilization.



(a) Utilization **(b) Packet Loss Rate**

Fig. 3.29: Utilization and Packet Loss Rate w.r.t. TBO

Next we fixed the bottleneck bandwidth at 1Gbps with buffer size B=24000 packets, and then change TBO from 4000 to 9000 packets to check utilization and packet loss rate again. The experiment results are shown in Fig. 3.29. We can see the bandwidth utilization also maintains 100% under different TBOs with a zero packet loss rate. The same reasons as the last experiment pertain.

3.3.8 Summary

The good performances above demonstrated by the IntelRate controller justify the rationale and verify the feasibility of our controller.

On one hand, the experiments under different link bandwidths above show that a choice of the TBO (which usually causes a queueing delay around 10 ms), N and m values works well in various network conditions. On the other hand, the max-min fairness has shown that the IntelRate controller can guarantee the maximum output according to the biggest rate recorded in *req_rate* among all passing flows. This verifies that the controller meets our design objective of choosing the outmost edge value D of the output. As a complement, Appendix F provides further experimental investigation and analysis of the choices of the value of these design parameters. Besides, our experiments show that the controller can maintain the IQSize at a level much lower than the designed buffer size B, even when the network undergoes big dynamics (such as the traffic change or bandwidth variations). This

means the IntelRate controller can significantly save the buffer resources, and thus a reduction in router design cost and size.

Perhaps the most notable feature in our design, as mentioned in the beginning, is the utilization of the queue size as the only parameter that the controller needs, which avoids mis-estimation and saves a lot of computational and memory resources that are required in other controllers.

Please note that the bandwidth of a router nowadays may be much larger than 5Gbps. However, since the design of our controller only depends on the queue size, theoretically the performance of the IntelRate controller is independent on bandwidth.

3.4 Concluding Remarks

Although network capacity nowadays quite often appears to be over-provisioned, the large number of files transfers (i.e., uploading or downloading) and streaming video applications (e.g., IPTV, video conference, on-line movie and games) pose a great surge of network traffic and therefore great threat on the network congestion. We have demonstrated our IntelRate controller can effectively throttle the source sending rate to avoid network congestion by only relying on one state variable in routers, i.e., the IQSize. It does not need to estimate the link bandwidth, packet loss, latency or the number of incoming flows for providing the explicit congestion signals and hence overcomes the defect of the existing congestion control protocols in potentially mis-estimating network parameters and complicating the computation of the congestion algorithm. The reason that the IntelRate controller outperforms its counterparts upon network dynamics has been analyzed in Appendix D. In the meanwhile, Appendix E demonstrates that our controller is fast enough for real time applications as further illustrated in the WLAN (Wireless Local Area Network) application in Chapter 5.

Chapter 4

OFEX: An Optimal and Fully Explicit Controller

In this chapter, we shall investigate a NUM-based congestion controller, called OFEX (An Optimal and Fully EXplicit) controller, based on the same end-to-end network model depicted in Fig. 2.2. Unlike the IntelRate controller which uses a closed-loop system dedicated to controlling the queue size, the OFEX controller uses the closed-loop system to match traffic demand and the link bandwidth.

The OFEX controller model is formulated in Section 4.1 before solved in Section 4.2. The convergence property and the time-delayed stability are then investigated in Section 4.3 and Section 4.4, respectively. After the robustness analysis in Section 4.5, the extensive experimental evaluation of the OFEX controller via simulation is done in Section 4.6. The computation complexity analysis of the OFEX controller can be found in Appendices I, where the feasibility of the OFEX is discussed for the actual implementation in a router.

4.1 The NUM-based Controller Design

Of the several outgoing links possible at a router, we first focus on all traffic queueing in one outgoing link. A link $l, \forall l \in L$, in a router has a bandwidth capacity c_l with a set of passing flows originated from a set of sources $S_l(t)$, $S_l(t) \subset S$. Each source $s \in S_l(t)$ carries a weight ψ_s, called *Flow_Weight* (which may be a price that the source is willing to pay for using the network), to link l so that it can obtain a proportion of bandwidth to transmit data with a rate of u_s(i.e., *Req_rate* mentioned in Section 2.1). We define "*revenue*" as the total income the resource owner may collect. The bandwidth sharing strategy in link l can thus be formulated as a convex optimization problem called **PP** (Primal Problem) below. Appendix G is also provided for different terminologies in convex optimization. For some common notations or symbols (e.g., $y(t)$ or $q(t)$) used in this Chapter, please refer to Chapter 2.

PP:

$$\max \quad \sum_{s \in S_l(t)} \psi_s U(u_s) \tag{4-1}$$

$$\text{subject to} \quad \sum_{s \in S_l(t)} u_s \le c_l - \gamma q(t) \quad \forall l \in L \tag{4-2}$$

$$u_s > 0 \quad s \in S_l(t) \tag{4-3}$$

Equation (4-1) is the objective function of link l to obtain the maximum "*revenue*" from forwarding traffic for all the passing flows. The function $\sum_{s \in S_l(t)} \psi_s U(u_s)$ is a social welfare function[5], where $U(u_s)$ is increasing, strictly concave and twice continuously differentiable over u_s. The bandwidth is allocated by link l according to different "payment" (from lowest to highest) from the sources, just like the ranking of social states. The parameter c_l is the nominal bandwidth (or say, face value of the bandwidth, which is simply called link bandwidth or capacity thereafter) of an outgoing link which is predefined in a router. The actual link bandwidth, however, can be more or less than c_l [Welz02].

The variable $q(t)$ is the instantaneous queue size which can be accurately monitored and measured at any time point as mentioned in Section 2.1 (e.g., at the departure point of each packet). The constant γ (γ>0) is a threshold parameter for $q(t)$ (to be discussed in Section 4.6.3 and Section 6.2.1.3) and has a unit of s^{-1}, i.e., 1/second. Eqn. (4-2) is a constraint equation stipulating that the aggregate incoming data rate $\sum_{s \in S_l(t)} u_s$ (represented by $y(t)$ as mentioned in Chapter 2) to link $l, \forall l \in L$, cannot exceed the link bandwidth c_l when the instantaneous queue is emptied.

Finally, constraint (4-3) stipulates/guarantees that source data rates are always positive, which means that once a flow is admitted to use link l, it is always allotted a portion of bandwidth. The total number of flows in link l is represented by $|S_l(t)|$. We define $\psi = [\psi_1, \psi_2, \cdots, \psi_{|S_l(t)|}]$ as the weight vector, and define $u_l = [u_1, u_2, \cdots, u_s, \cdots, u_{|S_l(t)|}]$ as the data rate vector of passing flows in link l, $\forall l \in L$. We let U_l be the domain of u_l, i.e., the region where u satisfies (4-2) and (4-3).

There are several differences in the formulation of our problem from others as reviewed in Section 1.4.2:

1) We attach a social welfare function to a link, instead of a utility function to end users. By interpreting the optimum of the formulated problem as "*revenue*", a link has the incentives to

[5]In economics, a social welfare function is a real-valued function that ranks conceivable social states (alternative complete descriptions of the society) from lowest to highest [Wiki13]. We borrow this terminology to draw the analogy between the ranking of social states and the way the sending rate of end users in the OFEX controller is determined.

relay data for the end users. Meanwhile, we can easily evolve such a model into a fully explicit congestion control algorithm.

2) We introduce the instantaneous queue size $q(t)$ in our model as a remedial measure not just to reduce the queue size but to further maintain the queue size at a near-zero level despite the link bandwidth dynamics. The rationale and such capability of the controller will be discussed in the following sections on design/analysis and performance evaluations.

3) As q(t) is the only state variable that needs to be monitored, it simplifies the measurement and the computational complexity of the control algorithm which has also been similarly demonstrated in Chapter 3 for the IntelRate controller.

4) Our model and formulation allow the Lagrange multiplier to take on a different economic notion from the relatively explicit controllers.

4.2 Optimization Framework

Instead of solving **PP** as a primal problem [BoVa04], we want to solve its duality instead so that we can obtain and interpret the Lagrange multiplier as "a price of per unit link bandwidth" (or "a unit bandwidth price"). We can then apply one of the KKT optimality conditions[6] to obtain the optimal data rate for all passing flows.

4.2.1 Problem Transformation

As a popular choice among researchers, we also choose $U(u_s) = \log(u_s)$ for link l because the natural log(.) gives rise to a concave function that would allow resources to be allocated according to the concept of proportional fairness [Kelly97, WaLt09]. However, our modeling approach mandates that the proportional fairness in OFEX controller is link-wise, instead of network-wise (e.g., in the relatively explicit protocols). This is crucial for the OFEX controller to exercise feeding back the congestion signal from the most congested link.

To convert **PP** to a Lagrange dual problem, we need to apply Lagrangian relaxation by associating a Lagrange multiplier $\lambda_l \geq 0, \forall l \in L,$ to the constraint (4-2). Then we use it to augment the objective function. The Lagrangian of **PP** is

$$\eta(\lambda_l, u) = \sum_{s \in S_l(t)} \psi_s \log(u_s) - \lambda_l \left(\sum_{s \in S_l(t)} u_s - c_l + \gamma q(t) \right) \quad . \tag{4-4}$$

[6]KKT optimality conditions [P244, BoVa04] here refer to the series of constraints attached to the primal and dual problems of a convex optimization problem.

Note that the total "revenue" of link l now includes two parts. In addition to the revenue $\sum_{s\in S_l(t)}\psi_s \log(u_s)$ inherited from the objective function, it now has a congestion cost $\lambda_l(\sum_{s\in S_l(t)}u_s - c_l + \gamma q(t))$. If we assume λ_l is the congestion cost per unit bandwidth, then when $\sum_{s\in S_l(t)}u_s > c_l$, (i.e., congestion is occurring because the aggregate incoming rate is greater than the link capacity), link l has to pay an additional cost to deal with the congestion, e.g., uses its queue buffer (i.e., $q(t)>0$) to temporarily accommodate those redundant data. Therefore, to avoid congestion, maintaining the congestion cost $\lambda_l(\sum_{s\in S_l(t)}u_s - c_l)$ to be less than or equal to zero is ideal. We shall visit this issue later on in "Comment 2".

Rearranging (4.4), we have

$$\eta(\lambda_l, x_l) = \sum_{s\in S_l(t)}\psi_s \log(u_s) - \lambda_l \sum_{s\in S_l(t)}u_s + \lambda_l(c_l - \gamma q(t))$$

$$= \sum_{s\in S_l(t)}(\psi_s \log(u_s) - \lambda_l u_s) + \lambda_l(c_l - \gamma q(t)). \quad (4\text{-}5)$$

Hence, the Lagrangian dual problem is

DP:

$$\min \quad \Omega(\lambda_l) \quad (4\text{-}6)$$

$$\lambda_l \geq 0, \quad \forall l \in L \quad (4\text{-}7)$$

where $\Omega(\lambda_l)$ is the dual function given by

$$\Omega(\lambda_l) = \max_{u_s} [\sum_{s\in S_l(t)}(\psi_s \log(u_s) - \lambda_l u_s)] + \lambda_l(c_l - \gamma q(t)). \quad (4\text{-}8)$$

4.2.2 Optimal Solution

Lemma 1 and Comment 1 below will justify legitimacy of our transformation from the problem **PP** to **DP**.

Lemma 1: Strong duality holds between the primal problem **PP** in (4-1)-(4-3) and the Lagrangian dual problem **DP** in (4-6)-(4-8), i.e., they have zero duality gap.

Proof*:* For a convex optimization problem, if the Slater's condition holds, the primal and the dual problem would have a zero duality gap [P226, BoVa04]. To see so, we need to check the

constraints in (4-2) and (4-3) whether they meet the Slater's condition. Since the constraint (4-3) is in the form of y>0 which is a special form of $y>ax+b$, this would meet the Slater's condition automatically. In other words, (4-3) is simply the refined Slater condition [BoVa04]. On the other hand, the constraint (4-2) is not affine. However, if there exists u_s so that (4-2) holds with a strict inequality, i.e., $\sum_{s\in S_l(t)} u_s < c_l - \gamma q(t)$, then the Slater's condition can also be met. By checking (4-2), one sees that there indeed exists a strictly feasible point, e.g., $u_s = 0.5(c_l - \gamma\, q(t))/|S_l(t)|$, $s \in S_l(t)$, and the link bandwidth $c_l>0$. Therefore, the Slater's condition holds, and the proof follows.■

***Comment* 1** (cf: chapter 5 [BoVa04]): The establishment of Lemma 1 enables us to solve the problem **PP** (4-1)-(4-3) by solving the problem **DP** (4-6)-(4-8) induced by the Lagrangian relaxation. The reason is that when the strong duality holds, the optimal values of the primal problem **PP** and the dual problem **DP** are the same, so the optimal points of the problem **DP** is exactly the optimal points of the primal problem **PP**. Hence, to solve the dual problem also solves the primal problem.

We have derived below the optimal results with Theorem 1 and Theorem 2 in a manner similar to [LoLa99, Srik04]. The major difference here is the emphasis of $q(t)$ as the remedial measure to overcome the shortcomings of the relatively explicit controllers that are not adaptive to varying network parameters and incur long queueing delays. Besides, even the result of Theorem 2 seems obviously, it is kept in the text for one's convenience in case one is not familiar with this topic.

Theorem 1: The problem **DP** in (4-6)-(4-8) can be solved by the gradient descent method guided by

$$\lambda_l^{(k+1)} = [\lambda_l^{(k)} + \delta \cdot (\sum_{s\in S_l(t)} u_s - c_l + \gamma q(t))]^+, \tag{4-9}$$

where $[a]^+ = \max(0, a)$, and δ is a constant step size with $\delta > 0$.

***Proof*:** Standard gradient descent method [P466, BoVa04] allows us to write

$$\lambda_l^{(k+1)} = \lambda_l^{(k)} - \delta \cdot \Omega'(\lambda_l) \tag{4-10}$$

From (4-8), we can obtain

$$\Omega'(\lambda_l) = \frac{d\Omega(\lambda_l)}{d\lambda_l} = -\sum_{s \in S_l(t)} u_s + (c_l - \gamma q(t)) \tag{4-11}$$

Therefore,

$$\lambda_l^{(k+1)} = \lambda_l^{(k)} + \delta \cdot (\sum_{s \in S_l(t)} u_s - c_l + \gamma q(t)). \tag{4-12}$$

Consider the constraint (4-7), and use $[a]^+$ to force λ_l in (4-12) to take on only non-negative values, Eqn. (4-9) follows. ■

***Comment* 2**: Eqn. (4-9) allows us to interpret how the link bandwidth can be managed from an economical perspective using RM (Revenue Management) whose principle is summarized in Appendix H.

When congestion is taking place, i.e., when $\sum_{s \in S_l(t)} u_s > c_l$ and $q(t)>0$, λ_l increases in each iteration because $\sum_{s \in S_l(t)} u_s - c_l + \gamma q(t)$ is positive. Otherwise (i.e., when $\sum_{s \in S_l(t)} u_s < c_l$ and $q(t)=0$), λ_l decreases. Economically, one may interpret λ_l as "a unit link bandwidth price". Depending on the traffic loads, link $l, \forall l \in L$, adjusts its λ_l, just as the way plane fare is adjusted by each airline company according to high or low travelling seasons. The parameter λ_l stays unchanged when $\sum_{s \in S_l(t)} u_s = c_l$ (Note: a very small $q(t)$ is allowed in this case, as seen in our simulation later), which means λ_l has converged to its optimum λ_l^*. In such a case, the traffic demand (i.e., the aggregate incoming traffic) and the bandwidth c_l of link $l, \forall l \in L$, have struck a balance.

Upon sudden shrinkage of c_l when the link is already fully utilized, $q(t)$ must build up due to the decreased link output capability. Eqn. (4-9) stipulates that λ_l must increase to reflect the shortage of link bandwidth so that the sources can decrease their sending rate accordingly (to be seen in Theorem 2). Note that the constant c_l in the controller remains the same all the time. The physical bandwidth shrink can be caused by interference, contention or joining in of some uncontrolled traffics such as UDP flows. In contrast, the relatively explicit controllers were designed without considering $q(t)$ in their model; thus it has no ability to detect the bandwidth dynamics, which subsequently degrade its application.

Eqn. (4-9) describes the relationship between $\sum_{s \in S_l(t)} u_s$ (i.e., the incoming traffic $y(t)$), c_l, and $q(t)$. One can see that the OFEX controller tries to match the sum of $\sum_{s \in S_l(t)} u_s$ and $q(t)$ to c_l. Based on this notion, a closed-loop control OFEX system is shown in Fig. 4.1 below.

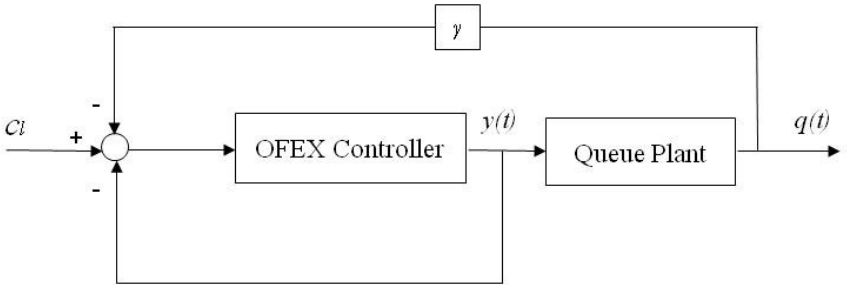


Fig. 4.1: The Closed-Loop OFEX Control System

Fig. 4.1 shows that the OFEX control system in a router needs to measure both $y(t)$ and $q(t)$ to complete the closed-loop control. The parameter $y(t)$ is measured from the incoming traffic to the queue plant in which $q(t)$ is also measured by the controller. The parameter γ in the box is a multiplier constant as indicated in Eqn. (4-9). According to the output of the summation junction for $y(t)$, $\gamma q(t)$ and c_l, the OFEX controller adjusts the required source sending rate until they strike a balance.

Theorem 2: The optimal data rate that each passing flow is allocated by link $l, \forall l \in L$, is simply

$$u_s^* = \frac{\psi_s}{\lambda_l^*} \tag{4-13}$$

***Proof*:** For optimization, we set the gradient $\partial \eta(\lambda_l, x_l) / \partial x_s^l = 0$ from (4-5), i.e.,

$\frac{\partial \eta(\lambda_l, x_l)}{\partial u_s} = \frac{\psi_s}{u_s} - \lambda_l = 0$, thus

$$u_s = \frac{\psi_s}{\lambda_l} \tag{4-14}$$

Hence, the unique optimal data rate u_s^* in (4-13) follows when λ_l converges to λ_l^* with (4-9).

■

***Comment* 3:** Eqn. (4-13) means the optimal portion of bandwidth allocated to source s by link l depends on the optimal "unit bandwidth price" λ_l^* and the source payment ψ_s. As

discussed in Comment 2, the value of λ_l^* will go up upon congestion. So the portion of bandwidth u_s^* source s can obtain under congestion will be smaller than that under non-congestion. This phenomenon is similar to the price of goods in market. When the supply of some goods is tight, its unit price will increase. The customer cannot acquire the same amount of goods with the same amount of payment.

4.2.3 The Control Procedure

With the link-wise proportional fairness, Theorem 1 and Theorem 2 established the computation of the optimal data rate for each flow in a link, which forms the link-wise control procedure of the OFEX controller in Fig. 4.2 below. The control procedure is implemented in each router of the network.

---

(1) Initialize the parameter γ, the step size δ and the Lagrangian multiplier $\lambda_l^{(0)}$ in link $l, \forall l \in L$, and set W=0.
(2) Do forever the following steps (3) to (4).
(3) Perform either A or B depending on a packet arrival or a departure.

A) For the *ith* Packet Arrival, *i*=1, 2, 3 ….
  3A1) Obtain the packet size *r* of the arrived packet.
  3A2) Update W via $W^{(i+1)} = W^{(i)} + r$.

B) For the *jth* Packet Departure, *j*=0,1,2, 3 ….
  3B1) extract the "price" *Flow_Weight* ψ_s from the header of the packet.
  3B2) compute the allowed sending data rate u_s according to (4-13) with $\lambda_l^{(k)}$.
  3B3) compare u_s with the existing *Req_rate* in the packet header.
  3B4) If $u_s <$ Req_rate, record x_s^l in the field *Req_rate*. Otherwise, no update is needed.
  3B5) Send the packet out.

(4) At k_{th} fixed intervals *T*, *k*=0,1,2, 3 ….
  4.1) Compute the aggregate incoming rate W/T.
  4.2) Set W=0.
  4.3) Measure *q*(*t*) and update the Lagrangian multiplier λ_l with equation (4-9) and obtain $\lambda_l^{(k)}$ as mentioned in step 3B2).

Fig. 4.2: Implementation Algorithm of the OFEX Controller

As per the operation principle generalized in Section 2.1, the step 3B1)-3B5) in Fig. 4.2 mandates that the allowed sending rate eventually fed back to a source is from the most

congested link along the data path. Therefore, like the IntelRate controller, from the perspective of the network in a whole, the control procedure of the OFEX algorithm exercises network-wise max-min fairness, based on the link-wise proportional fairness. Note that the queue size $q(t)$ as a state variable is measured every interval T during the Lagrangian multiplier λ_l update, instead of at the packet departure points of the IntelRate Controller in Section 3.1.3.

The computation complexity of the above OFEX control procedure has been analysed in Appendix I. It shows that Step (4) (i.e., cycle-level manipulations) shares more than half of the manipulations which will burden the router computation resources if they are all implemented per packet. Furthermore, the comparison in Appendix I shows that the control procedure above just needs one more manipulation than the relatively explicit controller. Therefore its complexity is not significantly higher than the existing NUM-based controllers.

4.3 Convergence analysis

Basically routers would like to collect the maximum "revenue" by selling its bandwidth with prices λ_l. Our analysis below will show how a router can collect (converge to) its optimal "*revenue*" fast. Besides, we discuss how the controller can achieve convergence even under dynamic bandwidth variations. An assumption and two lemmas will be first presented below to facilitate our argument and proof later on.

Assumption: The unit bandwidth price λ_l satisfies $0 < \Lambda_{l\,m} \le \lambda_l \le \Lambda_{l\,M} < \infty$.

***Comments*:** This assumption is reasonable because λ_l (as introduced in Section 4.2) usually bounded by some realistic limits. The lower bound $\Lambda_{l\,m}$ of λ_l is the minimum payment required by link l, and it stays as a non-zero parameter in the denominator of (4-13). Likewise, there is an upper bound $\Lambda_{l\,M}$ to represent the maximum unit bandwidth price levied by link l. ■

Lemma 1: The objective function $\Omega(\lambda_l)$ (4-6) is strongly convex.

***Proof*:** $\Omega(\lambda_l)$ is strongly convex if only if there exists $\hat{m} > 0$ such that the second derivative $\Omega''(\lambda_l) \ge \hat{m}$ [P677, Bert99]. The first derivative

$$\Omega'(\lambda_l) = -\sum_{s \in S_l(t)} x_s^l + (c_l - \gamma q(t)). \tag{4-15}$$

To derive $\Omega''(\lambda_l)$, we plug (4-14) into (4-15),

$$\Omega'(\lambda_l) = \frac{d\Omega(\lambda_l)}{d\lambda_l} = -\sum_{s \in S_l(t)} \frac{\psi_s}{\lambda_l} + (c_l(t) - \gamma q(t)). \tag{4-16}$$

Differentiate (4-16), we obtain

$$\Omega''(\lambda_l) = \frac{d^2\Omega(\lambda_l)}{d\lambda_l^2} = \frac{\sum \psi_s}{\lambda_l^2} \tag{4-17}$$

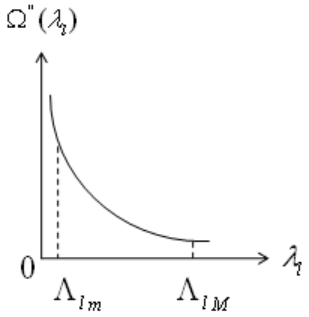


Fig. 4.3: The Second Derivative $\Omega''(\lambda_l)$

Fig. 4.3 shows the general characteristic curve of $\Omega''(\lambda_l)$. The assumption stated above allows us to conclude that $\Omega''(\lambda_l) \geq \Omega''(\Lambda_{lM}) = \frac{\sum \psi_s}{\Lambda_{lM}^2}$. Therefore, there exists $\hat{m} = \Omega''(\Lambda_{lM})$, which completes the proof. ■

From Lemma 1 and our Assumption, we made the following observations in passing: (1) $\Omega''(\lambda_l) \leq \Omega''(\Lambda_{lm}) = \frac{\sum \psi_s}{\Lambda_{lm}^2}$. This relationship can be constructed from Fig. 4.3 along with Eqn. (4-17); (2) $\Omega(\lambda_l)$ is twice differentiable as required by the derivation from Eqn. (4-16) to Eqn. (4-17). These useful observations are required by Lemma 2 below.

Lemma 2: The first derivative $\Omega'(\lambda_l)$ of the objective function is Lipschitz continuous with the Lipschitz constant $L_c = \Omega''(\Lambda_{lm})$.

***Proof*:** If there exists some $L_c > 0$ such that the curvature of $\Omega(\lambda_l)$ is no more than L_c, the

proof is complete [P48, Bert99]. Therefore, we check the existence of L_c>0.

Because $\Omega(\lambda_l)$ is twice differentiable, the Lagrange Mean Value Theorem [P667, Bert99] allows us to write, for any given j, k>0,

$$\Omega'(j)-\Omega'(k)=\Omega''(\xi)(j-k),\quad \xi\in[\Lambda_{lm},\Lambda_{lM}]. \tag{4-18}$$

$$\begin{aligned}\text{Thus } \left|\Omega'(j)-\Omega'(k)\right| &= \left|\Omega''(\xi)(j-k)\right| \\ &= \left|\Omega''(\xi)\right|\left|(j-k)\right|\end{aligned} \tag{4-19}$$

Since $\left|\Omega''(\xi)\right|$ is upper bounded by $\Omega''(\Lambda_{lm})$,

$$\left|\Omega'(j)-\Omega'(k)\right|\le\Omega''(\Lambda_{lm})\left|(j-k)\right|. \tag{4-20}$$

According to Lipschitz continuity condition [P47, Bert99], (4-20) shows the curvature of $\Omega(\lambda_l)$ is no more than $\Omega''(\Lambda_{lm})$, i.e., there exists a Lipschitz constant $L_c=\Omega''(\Lambda_{lm})>0$, which yields the desired result. ■

Theorem 3: Given the initial unit bandwidth price $\Lambda_{lm}\le\lambda_l^{(0)}\le\Lambda_{lM}$, $\forall l\in L$, if the step size satisfies $0<\delta\le(1-\sqrt{1-(1-\alpha)L_c/\hat{m}})/L_c$ or $(1+\sqrt{1-(1-\alpha)L_c/\hat{m}})/L_c\le\delta<2/L_c$ ($0<\alpha<1$), the control algorithm in (4-9) and (4-13) converges to the equilibrium (λ_l^*, x_l^*).

***Proof*:** Using the Descent Lemma [P667, Bert99], we can obtain

$$\Omega(\lambda_l^{(k)}-\delta\cdot\Omega'(\lambda_l^{(k)}))\le\Omega(\lambda_l^{(k)})+(-\delta\cdot\Omega'(\lambda_l^{(k)}))\Omega'(\lambda_l^{(k)})+\frac{L_c}{2}\left|-\delta\cdot\Omega'(\lambda_l^{(k)})\right|^2. \tag{4-21}$$

Since $\lambda_l^{(k+1)}=\lambda_l^{(k)}-\delta\cdot\Omega'(\lambda_l)$, (4-21) can be written as

$$\Omega(\lambda_l^{(k+1)})\le\Omega(\lambda_l^{(k)})+(-\delta\cdot\Omega'(\lambda_l^{(k)}))\Omega'(\lambda_l^{(k)})+\frac{L_c}{2}\left|-\delta\cdot\Omega'(\lambda_l^{(k)})\right|^2 .$$

Combining the last two terms of the right hand side, we obtain

$$\Omega(\lambda_l^{(k+1)})\le\Omega(\lambda_l^{(k)})+\delta(\frac{L_c}{2}\delta-1)\left|\Omega'(\lambda_l^{(k)})\right|^2 . \tag{4-22}$$

Subtracting a constant, say Ω^*, from both sides of (4-22), we obtain

$$\Omega(\lambda_l^{(k+1)})-\Omega^*\le\Omega(\lambda_l^{(k)})-\Omega^*+\delta(\frac{L_c}{2}\delta-1)[\Omega'(\lambda_l^{(k)})]^2 . \tag{4-23}$$

Using the strong convexity of $\Omega(\lambda_l)$ from Lemma 1, we would have

$$\left|\Omega'(\lambda_l^{(k)})\right|^2\ge2\hat{m}(\Omega(\lambda_l^{(k)})-\Omega^*) \tag{4-24}$$

where $\hat{m}$ is the parameter discussed in Lemma 1.

If we assume $L_c\delta < 2$, we can rewrite (4-23) as

$$\Omega(\lambda_l^{(k+1)}) - \Omega^* \le \Omega(\lambda_l^{(k)}) - \Omega^* + \delta(\frac{L_c}{2}\delta - 1)(2\hat{m}(\Omega(\lambda_l^{(k)}) - \Omega^*)) \tag{4-25}$$

After rearrangement, we have

$$\Omega(\lambda_l^{(k+1)}) - \Omega^* \le [2\hat{m}\delta(\frac{L_c}{2}\delta - 1) + 1]((\Omega(\lambda_l^{(k)}) - \Omega^*)) \tag{4-26}$$

This inequality can be applied recursively on itself to generate

$$\Omega(\lambda_l^{(k+1)}) - \Omega^* \le [2\hat{m}\delta(\frac{L_c}{2}\delta - 1) + 1]^{k+1}((\Omega(\lambda_l^{(0)}) - \Omega^*)). \tag{4-27}$$

Assume $0 < \alpha < 1$, under the condition of

$$0 < \alpha \le 2\hat{m}\delta(\frac{L_c}{2}\delta - 1) + 1 < 1 \quad , \tag{4-28}$$

$\Omega(\lambda_l^{(k+1)})$ converges to Ω^* in (4-27) as $k \to \infty$, i.e., the objective function $\Omega(\lambda_l)$ reaches its optimal value Ω^*, and the unit bandwidth price reaches an optimum λ_l^*. Accordingly, the optimal data rate $x_s^{l^*}$ follows with (4-13). Hence, the system reaches the equilibrium (λ_l^*, x_l^*). Solving (4-28), for convergence, the step size should satisfy $0 < \delta < (1 - \sqrt{1 - (1-\alpha)L_c / \hat{m}}) / L_c$ or $(1 + \sqrt{1 - (1-\alpha)L_c / \hat{m}}) / L_c < \delta < 2 / L_c \, (0 < \alpha < 1)$. The proof is complete. ■

***Comment*:** From (4-27), the term $[2\hat{m}\delta(\frac{L_c}{2}\delta - 1) + 1]^{k+1}$ shows that the error $[\Omega(\lambda_l^{(k+1)}) - \Omega^*]$ converges to zero as $k \to \infty$ subject to Eqn. (4-28), which means the objective function $\Omega(\lambda_l^{(k+1)})$ converges to its optimal point Ω^* at least as fast as a geometric series [BoVa04]. Specifically, this is because of k+1 as the exponential in the term $[2\hat{m}\delta(\frac{L_c}{2}\delta - 1) + 1]^{k+1}$, which leads to a fast decrease of the term (for the convergence of $[\Omega(\lambda_l^{(k+1)}) - \Omega^*]$ to zero) at a speed called geometric series.

Above we have proved that the OFEX controller can converge to its optimum Ω^*, i.e., $\Omega(\lambda_l)$ in (4-8) reaches its maximum, and (4-8) becomes

$$\Omega^* = [\sum_{s \in S_l(t)} (\psi_s \log(u_s^*) - \lambda_l^* u_s^*)] + \lambda_l^* (c_l - \gamma q(t)). \tag{4-29}$$

After arranging (4-29),

$$\Omega^* = \sum_{s \in S_l(t)} (\psi_s \log(u_s^*)) - \lambda_l^* (\sum_{s \in S_l(t)} u_s^* - (c_l - \gamma q(t))) . \quad (4\text{-}30)$$

Next by analyzing the "traffic demand and bandwidth supply" of a link, we shall verify that Ω^* is an authentic optimum for the OFEX controller even under bandwidth variation situations. But before doing so, we shall analyze the optimum Ω^* of the relatively explicit controllers as bandwidth varies. One can find the relatively explicit controller goes to a fake optimum Ω^* under such situations, which brings about the non-convergence problem.

Since the relatively explicit controller does not consider $q(t)$ in their models, their optimum Ω^* will be as follows.

$$\Omega^* = \sum_{s \in S_l(t)} (\psi_s \log(u_s^*)) - \lambda_l^* (\sum_{s \in S_l(t)} u_s^* - c_l) . \quad (4\text{-}31)$$

which is obtained by eliminating the item $\gamma q(t)$ in (4-30). Also, with the relatively explicit controller, by adapting (4-20), their Lagrange multiplier is updated according to

$$\lambda_l^{(k+1)} = \left[\lambda_l^{(k)} + \delta \cdot (\sum_{s \in S_l(t)} u_s - c_l) \right]^+ \quad (4\text{-}32)$$

When the bandwidth c_l (i.e., bandwidth supply) shrinks due to contention or interferences, the relatively explicit controller with (4-31) does not sense it, so (4-32) still tries to match the incoming traffic $\sum_{s \in S_l(t)} u_s$ (i.e., traffic demand) with the face value c_l of the link bandwidth. In fact, now the relatively explicit controller is overwhelming the queue of a router because the actual bandwidth is already less than c_l. In such an "optimal" situation where $\sum_{s \in S_l(t)} u_s$ matches c_l, λ_l^* and u_s^* are fake optimal values which resultantly lead to a fake Ω^* in (4-31). To be seen later in the simulation, such kind of queue overwhelming in routers causes the non-convergence problem in the relatively explicit controllers.

In contrast, when the bandwidth shrinks, the OFEX controller can sense it by sensing a growing $q(t)$. From (4-9), now $\sum_{s \in S_l(t)} u_s$ is matching $(c_l - \gamma q(t))$ by rearranging (4-9) as follows.

$$\lambda_l^{(k+1)} = \left[\lambda_l^{(k)} + \delta \cdot (\sum_{s \in S_l(t)} u_s - (c_l - \gamma q(t)) \right]^+ \quad (4\text{-}33)$$

That is to say, u_s^* must decrease so as to match $(c_l - \gamma q(t))$. At the same time, the queue also avoids being overwhelmed. Please note $(c_l - \gamma q(t))$ exactly reflects the shrunken

bandwidth that the OFEX controller can sense via the measurement of $q(t)$. Therefore, λ_l^* and u_s^* are authentic optima in the OFEX controller as bandwidth varies, and so is the Ω^* in (4-27) or (4-29).

4.4 Stability with Time Delay

In the previous section, the OFEX controller in a link was proved to be able to converge to an optimal equilibrium under certain conditions. In this section, with the end-to-end single router model as shown in Fig. 2.2, we are going to show the time-delayed stability of the OFEX algorithm. Note that the mathematical stability proof of the congestion control in multi-bottlenecked network models or a general network topology remains an open issue [Skri04].

To do so, we will first define differential equations for the OFEX algorithm, and then linearize them around the local equilibrium point. Finally, we will transform them to the frequency domain and employ the Nyquist stability criterion to derive sufficient conditions for the system to be locally stable.

From (4-9), the unit link price λ_l can be defined to be updated according to the following differential equation

$$\dot{\lambda}_l(t) = \delta \cdot \Big(\sum_{s \in S_l(t)} u_s(t) - c_l + \gamma q(t) \Big) \tag{4-34}$$

which reflects the change of the unit link price without time delay. With time delay considered, (I-1) will become

$$\dot{\lambda}_l(t) = \delta \cdot \Big(\sum_{s \in S_l(t)} u_s(t-\tau) - c_l + \gamma q(t) \Big) \tag{4-35}$$

This is because the explicit data rate u_s only takes effect after one RTT, i.e., τ. To see this, we have the following discussion. The variable u_s is the allowed sending rate calculated by link l and recorded in the header of a packet on its way to the destination. When the packet arrives at its destination, u_s is piggybacked in an ACK packet to the source. The source then extracts u_s, and uses it to update the data sending rate accordingly. When the data with the new sending rate reach link l, it has been exactly one RTT τ since the new u_s was issued. The link price λ_l is then updated according to the new incoming rate from sources.

Since the allowed source sending rate u_s has to be matched to the price the source *s* pays (i.e., ψ_s), the update of u_s by link *l* can be defined with the following differential equation

$$\dot{u}_s(t) = \psi_s - u_s(t)\lambda_l(t) \tag{4-36}$$

The second term $u_s(t)\lambda_l(t)$ on the right-hand side of (4-36) is the product of the data rate (in bps) authorized to source *s* and the unit bandwidth price of link *l* (e.g., in dollars/bps), and represents the total amount that the link needs to collect. The update of u_s in (4-36) eventually matches $u_s\lambda_l(t)$ to the payment, i.e., ψ_s. We introduce $\Delta\lambda_l(t) = \lambda_l(t) - \bar{\lambda}_l$, $\Delta x_s^l(t) = u_s(t) - \bar{x}_s^l$ and $\Delta q(t) = q(t) - \bar{q}$, where $\bar{\lambda}_l$, $\bar{x}_s^l$ and $\bar{q}$ are the local equilibrium points of $\lambda_l(t)$, u_s and $q(t)$. Among them, the time delay with u_s is $\Delta x_s^l(t-\tau) = u_s(t-\tau) - \bar{x}_s^l$. By putting $\lambda_l(t) = \Delta\lambda_l(t) + \bar{\lambda}_l$, $u_s(t-\tau) = \Delta x_s^l(t-\tau) + \bar{x}_s^l$ and $q(t) = \Delta q(t) + \bar{q}$ in (4-35), we have

$\Delta\dot{\lambda}_l(t) = \delta \cdot (\sum_{s\in S_l(t)} (\Delta x_s^l(t-\tau) + \bar{x}_s^l) - c_l + \gamma(\Delta q(t) + \bar{q}))$, and rearrange it,

$$\Delta\dot{\lambda}_l(t) = \delta \cdot (\sum_{s\in S_l(t)} \Delta x_s^l(t-\tau) + \gamma\Delta q(t) + \sum_{s\in S_l(t)} \bar{x}_s^l + \gamma\bar{q} - c_l)$$

At equilibrium, when $\sum_{s\in S_l(t)} \bar{x}_s^l + \gamma\bar{q}$ strikes equality with the link bandwidth c_l, one obtains

$$\Delta\dot{\lambda}_l(t) = \delta \cdot (\sum_{s\in S_l(t)} \Delta x_s^l(t-\tau) + \gamma\Delta q(t)) \tag{4-37}$$

Similarly, to linearize (4-37), we have $\Delta\dot{x}_s^l(t) = \psi_s - (\Delta x_s^l(t) + \bar{x}_s^l)(\Delta\lambda_l(t) + \bar{\lambda}_l)$ and it can be rewritten as $\Delta\dot{x}_s^l(t) = \psi_s - (\Delta x_s^l\Delta\lambda_l(t) + \bar{\lambda}\Delta x_s^l(t) + \bar{x}_s^l\Delta\lambda_l(t)(t) + \bar{x}_s^l\bar{\lambda}_l)$. At equilibrium, $\bar{x}_s^l\bar{\lambda}$ matches ψ_s, and they cancel out. Also, the small term $\Delta x_s^l\Delta\lambda_l(t)$ can be dropped. Thus, we have

$$\Delta\dot{x}_s^l(t) = -\bar{\lambda}\Delta x_s^l(t) - \bar{x}_s^l\Delta\lambda_l(t) \tag{4-38}$$

Based on (4-38), the following equation represents the aggregate differential equations of the allowed sending rate of all the sources using link *l*.

$$\sum_{s\in S_l(t)} \Delta\dot{x}_s^l(t) = -\bar{\lambda} \sum_{s\in S_l(t)} \Delta x_s^l(t) - \Delta\lambda_l(t) \sum_{s\in S_l(t)} \bar{x}_s^l$$

Since $\sum_{s\in S_l(t)} \bar{x}_s^l + \gamma\bar{q}$ strikes equal with the link bandwidth c_l, $\sum_{s\in S_l(t)} \bar{x}_s^l = c_l - \gamma\bar{q}$. So

$$\sum_{s\in S_l(t)} \Delta\dot{x}_s^l(t) = -\bar{\lambda} \sum_{s\in S_l(t)} \Delta x_s^l(t) - \Delta\lambda_l(t)(c_l - \gamma\bar{q}) \tag{4-39}$$

For simplicity, we replace the aggregate $\sum_{s\in S_l(t)}\Delta x_s^l(t)$ with $\Delta y_l(t)$. Accordingly, $\sum_{s\in S_l(t)}\Delta x_s^l(t-\tau)$ can be replaced by $\Delta y_l(t-\tau)$. Now, (4-37) and (4-39) can be rewritten as

$$\Delta\dot{\lambda}_l(t) = \delta\cdot(\Delta y_l(t-\tau) + \gamma\Delta q(t) \tag{4-40}$$

and

$$\Delta\dot{y}_l(t) = -\bar{\lambda}\Delta y_l(t) - (c_l - \gamma\bar{q})\Delta\lambda_l(t) \tag{4-41}$$

(4-40) and (4-41) thus form the OFEX control system with the time-delay.

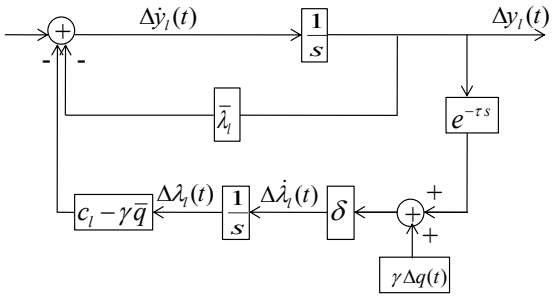


Fig. 4.4: The OFEX Closed-loop Control System

Fig. 4.4 shows the OFEX closed-loop control system based on (4-40) and (4-41). The open-loop transfer function is

$$G(s) = \frac{\bar{\lambda}_l s + \delta(c_l - \gamma\bar{q})e^{-\tau s}}{s^2} \tag{4-42}$$

Note that the other input $\gamma\Delta q(t)$ has to be set to zero when we were to find the transfer function $G(s)$ between the output $\Delta y_l(t)$ and the input on the left side [DoBi08].

Let $K_1 = \bar{\lambda}_l$ and $K_2 = \delta(c_l - \gamma\bar{q})$. Then

$$G(s) = \frac{K_1 s + K_2 e^{-\tau s}}{s^2} \tag{4-43}$$

Using the first-order Padé approximation for the delay term $e^{-\tau s}$, i.e., let $e^{-\tau s} \approx \frac{-\tau s + 2}{\tau s + 2}$, (4-43) becomes

$$G(s) = \frac{K_1 s + K_2 \dfrac{-\tau s + 2}{\tau s + 2}}{s^2} \tag{4-44}$$

To apply the Nyquist criterion, we substitute *s* with $j\omega$,

$$G(j\omega) = \frac{jK_1\omega + K_2\dfrac{-j\tau\omega + 2}{j\tau\omega + 2}}{(j\omega)^2} \tag{4-45}$$

Rewriting $G(j\omega)$ in its Cartesian form of $Re+jIm$,

$$G(j\omega) = \frac{K_2(4-\tau^2\omega^2) + j\omega(4K_1 - 4K_2\tau + K_1\tau^2\omega^2)}{-\omega^2(4+\tau^2\omega^2)} \tag{4-46}$$

From Nyquist plot, one sufficient condition to guarantee the system stability is to have the crossover point with the horizontal axis on the right of (-1, 0). To determine this crossover point, we let the imaginary part of (4-46) to be zero, i.e.,

$\dfrac{\omega(4K_1 - 4K_2\tau + K_1\tau^2\omega^2)}{-\omega^2(4+\tau^2\omega^2)} = 0$, from which we obtain

$$\omega^2 = \frac{4(K_2\tau - K_1)}{K_1\tau^2} \tag{4-47}$$

Note we have omitted another solution $\omega = 0$ because of the non-zero requirement of denominator in (4-46). Substituting (4-47) into the real part of (4-46), and let the real part be greater than -1 to have the crossover point on the right of (-1,0) to ensure stability, we have

$$K_2\left(4 - \tau^2 \cdot \frac{4(K_2\tau - K_1)}{K_1\tau^2}\right) > -1 \tag{4-48}$$

After rearranging (4-48), and factoring on both sides, we obtain

$K_1^2 + (4\tau - K_2\tau)K_1 - 4\tau^2K_2 < -K_1^2$.

$$(K_1 - K_2\tau)(K_1 + 4\tau) < -K_1 \cdot K_1 \tag{4-49}$$

From (4-49), we can determine two conditions

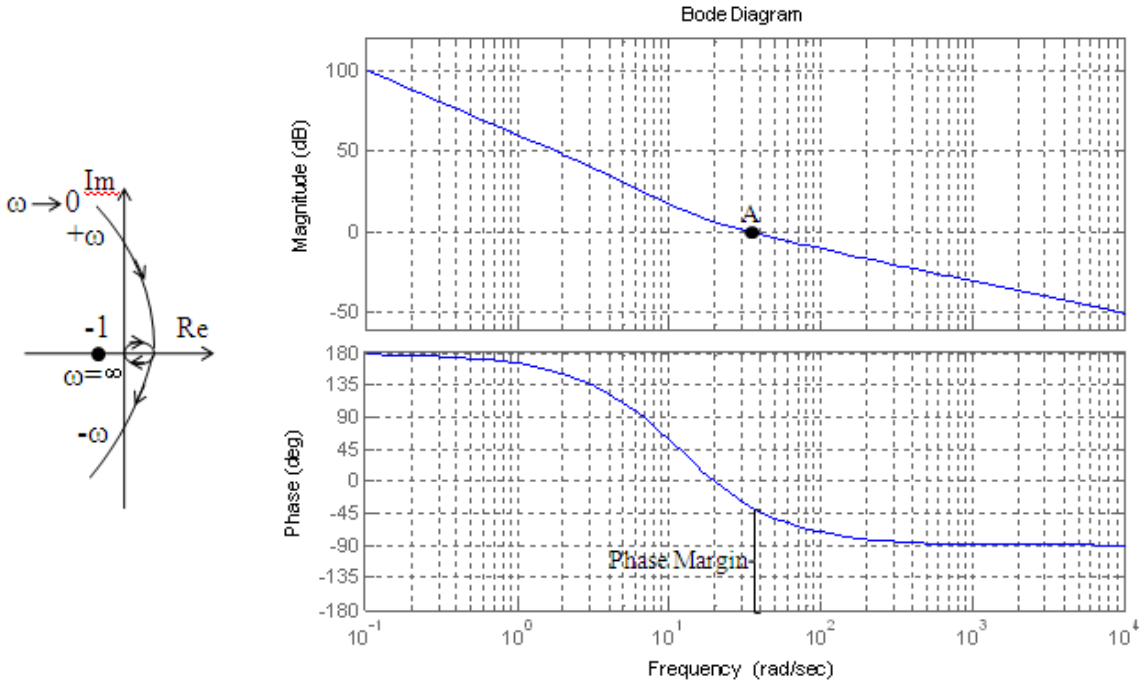
$$\begin{cases} K_1 - K_2\tau < -K_1 \\ K_1 + 4\tau > K_1 \end{cases} \tag{4-50}$$

The second inequality in (4-50) holds automatically (in that the RTT τ is always greater than zero) while the first inequality of (4-50) is true if

$$\frac{K_1}{K_2} < \frac{\tau}{2}, \text{ i.e., } \frac{\bar{\lambda}}{\delta(c_l - \gamma\bar{q})} < \frac{\tau}{2}, \tag{4-51}$$

which is the sufficient condition for the OFEX control system with time delay to be locally stable.

Fig. 4.5a and Fig. 4.5b show one example Nyquist plot and Bode diagram with K_1=30, K_2=1000 and the time delay τ=0.3s, where K_1, K_2 and τ satisfy (I-18). The phase margin is about 135°, and the system is stable.



(a) Nyquist Plot **(b) Bode Diagram**

Fig. 4.5: Nyquist Plot and Bode Diagram

4.5 Robustness

In this section, another property of the OFEX controller will be investigated. Specifically, we shall show how the optimal link price λ^* is related to the disturbances in the link bandwidth c_l based on the strong duality. In particular, the optimal value of the OFEX controller is upper bounded in the presence of the disturbances in bandwidth c_l. We also analyze the local sensitivity.

Here, we adopt the notion of robustness based on the convex optimization theory for the OFEX controller, and define robustness to be the sensitivity of the controller to obtain the maximum "*revenue*" in the presence of disturbances in link bandwidth.

4.5.1 The Perturbed System Model

The perturbed network model arises from the original problem **PP** in Section 4.1 where all

flows are contending in a router whose bandwidth c_l can fluctuate as in the shared Ethernet or IEEE 802.11 WLAN. Our starting point is the equation (4-2) where we replace $-\gamma q(t)$ by $w(t)$. The variable $w(t)$ is a non-positive as $q(t)$ cannot be negative, i.e. $w(t)\leq 0$. Throughout this paper, we always respect this requirement. Consequently, our disturbed network model becomes

RPP:

$$\max \quad \sum_{s\in S_l(t)} \psi_s \log(u_s) \tag{4-52}$$

$$\text{subject to} \quad \sum_{s\in S_l(t)} u_s - c_l \leq w(t) \quad \forall l \in L \tag{4-53}$$

$$u_s > 0 \quad s \in S_l(t) \tag{4-54}$$

We represent the optimal "*revenue*" of the disturbed system with $\Pi^*(w)$, and see how $\Pi^*(w)$ would vary with respect to the disturbances $w(t)$ from the constraint (4-53). Accordingly, $\Pi^*(0)$ is the optimal "*revenue*" that a link can obtain when the system has no disturbance (i.e., $w(t)=0$).

4.5.2 Upper Bound of the Optimum

The following proposition will allow us to obtain an upper bound of the optimal revenue $\Pi^*(w)$ of the OFEX controller upon c_l disturbances.

Proposition 1: Suppose λ^* is the optimal unit link price of the unperturbed system (i.e., when $w(t)=0$), then for any bandwidth decrease ($w(t)<0$) on c_l, the following inequality establishes.

$$\Pi^*(w) \leq \Pi^*(0) + \lambda^* w(t) \tag{4-55}$$

Proof: It is obvious that $\Pi^*(0) = \Omega(\lambda^*)$ by the strong duality of the PP and DP, as discussed in Section 4.2. We also have $\Omega(\lambda^*) \geq \sum_{s\in S_l(t)} \psi_s \log(u_s) - \lambda^* (\sum_{s\in S_l(t)} u_s - c_l)$ by the definition of the dual function. Therefore

$$\Pi^*(0) \geq \sum_{s\in S_l(t)} \psi_s \log(u_s) - \lambda^* (\sum_{s\in S_l(t)} u_s - c_l) \tag{4-56}$$

Since $\lambda^* \geq 0$ and $\sum_{s \in S_l(t)} u_s - c_l \leq w(t)$, $\forall l \in L$, (4-56) can be rewritten as

$$\Pi^*(0) \geq \sum_{s \in S_l(t)} \psi_s \log(u_s) - \lambda^* w(t) \tag{4-57}$$

Rearranging (4-57), we have $\Pi^*(0) + \lambda^* w(t) \geq \sum_{s \in S_l(t)} \psi_s \log(u_s))$. Thus

$$\sum_{s \in S_l(t)} \psi_s \log(u_s) \leq \Pi^*(0) + \lambda^* w(t) \tag{4-58}$$

Eqn. (4-58) is true for any $w(t)$ including the maximum of $\sum_{s \in S_l(t)} \psi_s \log(u_s)$ in (4-52). Therefore, $\Pi^*(w) \leq \Pi^*(0) + \lambda^* w(t)$, and the proof is complete. ■

We can now make the following few sensitivity interpretations directly from the inequality (4-53) with the aid of Fig. 4.6 where the affine function $\Pi^*(0) + \lambda^* w(t)$ is a straight line that can be drawn tangential to $\Pi^*(w)$ at $w(t)=0$ to represent the upper-bound condition.

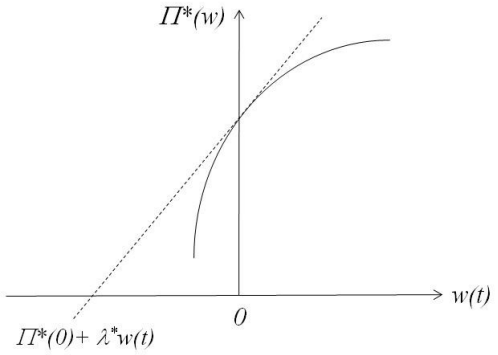


Fig. 4.6: The Inequality (4-55)

1) If the optimal link price λ^* is large, and the bandwidth is decreasing (i.e., $w(t) < 0$), the optimal revenue $\Pi^*(w)$ that the link can collect decreases drastically. This can be seen from Fig. 4.6 where the concavity of the function when $w(t)<0$ implies a steeper slope than the slope (price) λ^* of the affine function.
2) Similarly, if the optimal link price λ^* is small, the optimal revenue $\Pi^*(w)$ that the link can collect decreases a little bit.
3) Since $w(t)$ is non-positive in the OFEX controller, we are not going to interpret the case $w(t)>0$ in detail. In such case, briefly, independent of the optimal link price λ^* (large or small), the optimal revenue $\Pi^*(w)$ that the link can collect may not increase too much. This is because for an inequality as (4-53), the increase of the right hand side does not

guarantee an increase of the left hand side.

Note that $\Pi^*(w) = \Pi^*(0) + \lambda^* w(t)$ is the special case when the "*revenue*" does not change.

4.5.3 Local Robustness Analysis

If we assume that $\Pi^*(w)$ is differentiable at w=0, then Fig. 4.6 depicts that the optimal dual variable λ^*, i.e., the optimal unit link price, is related to the gradient of $\Pi^*(w)$ at $w(t)$=0. Simply put,

$$\lambda^* = \frac{d\Pi^*(w)}{dw}\Big|_{w=0} \tag{4-59}$$

Therefore, the optimal link price is exactly the local sensitivity measured by the change of the optimal link revenue with respect to the bandwidth variations around $w(t)$=0 .

Eqn. (4-59) also gives us the following quantitative interpretations about the variations of $\Pi^*(w)$ around $w(t)$=0. These interpretations can also be seen from Fig. 4.6, where λ^* is the slope of $\Pi^*(w)$ near $w(t)$=0.

(1) If the link variation $w(t)$ is small and negative, there will be a decrease in $\Pi^*(0)$ of about $-\lambda^* w(t)$;

(2) If both the link variation $w(t)$ and λ^* are very small, they can be ignored and the optimal "*revenue*" is $\Pi^*(0)$.

(3) If the link price λ^* is large, any variation of $w(t)$ may have a big impact on the maximum revenue of $\Pi^*(0)$. That is, the link may lose a lot of "*revenue*" even if the bandwidth decreases a bit.

(4) Since $w(t)$ is non-positive in the OFEX controller, we are not going to interpret the case that $w(t)$ is small and positive.

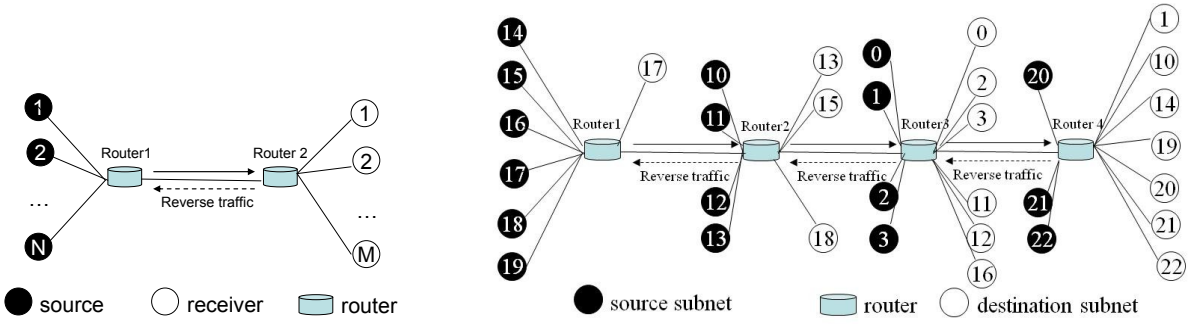
4.5.4 Discussion

Since the OFEX controller is formulated to maximize the link "*revenue*" so that each passing flow can obtain a portion of bandwidth, the effect of the link bandwidth variations is a concern to the incentives with which a link would like to relay data for sources. Using the convex theoretic, we have discussed how the link optimal "*revenue*" may be affected as

bandwidth varies with respect to the link price λ^*. In particular, when λ^* is big, the bandwidth shrinkage may lead to a big loss in the link optimal revenue. Our Eqn. (4-52) implies that the passing flows have decreased their sending rate u_s as a response to the bandwidth reduction so that the maximum value (i.e., the optimal revenue) of $\sum_{s\in S_l(t)} \psi_s \log(u_s)$ decreases accordingly. This is the desired robustness performance the OFEX controller is designed for.

4.6 Performance Evaluation

In this section, the performance and capability of our OFEX controller are evaluated through a series of experiments. Section 4.6.1 describes the simulation settings. Section 4.6.2 and Section 4.6.3 separately show bandwidth allocation and the effect of the parameter γ on the controller performance. Section 4.6.4 shows the robustness of the OFEX controller. Finally, the superiority of the OFEX controller is presented by comparisons in Section 4.6.5.



(a) Single Bottleneck Network **(b) Multi-Bottleneck Network**

Fig. 4.7: Network Simulation Topologies

4.6.1 Simulated Network

A single bottleneck network and a multiple bottleneck network are used to evaluate the OFEX controller performance, as shown in Fig. 4.7.

The single bottleneck network Fig. 4.7a is the same as Fig. 3.6a for the IntelRate controller, and the network settings are therefore basically the same as described in Section 3.3.1.1 for the IntelRate controller except the buffer capacity *B* in Router1 is to set roughly equal to the BDP for the OFEX controller (because, unlike the IntelRate controller, the OFEX controller has no TBO, and thus there is no parameter that can be used as a reference to set up *B*). Therefore, other settings and configurations in Section 3.3.1.1 are also applicable to Fig. 4.7a here. Fig. 4.7a is the topology used to test some basic performances of

the OFEX controller in Sections 4.6.2 -Section 4.6.4 and Section 4.6.5.2, where the optimal data rate allocation, the γ effect, robustness to network parameter changes and some comparisons are conducted.

Table 4.1: Sources Characteristics in Multiple Bottleneck Network

Router No.	Subnet/Group ID	Source ID	Flow No.	RTPD(ms)
1	0	1-20	ftp 1-20	52
	1	21-40	ftp 21-40	80
	2	41-60	ftp 41-60	96
	3	61-80	UDP 1-20	75
2	10	1-20	ftp 1-20	180
	11	21-40	ftp 21-40	160
	12	40-60	http 1-20	150
	13	60-80	UDP 1-20	200
3	14	1-20	http 1-20	100
	15	21-40	ftp 1-20	50
	16	41-60	ftp 21-40	170
	17	61-80	ftp 41-60	80
	18	81-100	ftp 61-80	110
	19	101-120	ftp 81-100	290
4	20	1-20	ftp 1-20	180
	21	21-40	ftp 21-40	166
	22	41-60	UDP 1-20	95

Since the congestion signal of the relatively explicit controller is an accumulation of link price of a series of routers along a flow path, a multiple bottleneck network topology depicted in Fig. 4.7b is used to allow the comparison in Section 4.6.5.1 between the OFEX controller and the relatively explicit controller (The multi-bottleneck network shown in Fig. 3.6b for the IntelRate controller can also be used for the OFEX controller here, but to show the variety of node assignment to the routers, we made Fig. 4.7b different from Fig. 3.6b, e.g., the destination nodes of subnet 10 are attached to Router 3 in Fig. 3.6b, but attached to Router 4 in Fig. 4.7b). The link bandwidth from Router 1 to Router 4 is 1.52Gbps, 3.02Gbps, 2.14Gbps and 1.86Gbps, respectively. The buffer capacity B in each router is set the same way as done for Router 1 in Fig. 4.7a.

Table 4.1 tabulates the source characteristics in the multiple bottleneck network Fig. 4.7b. Other settings such as packet size and buffer capacity B are the same as that in the single network scenario.

The performance measures and the worst traffic scenarios are the same as in Section 3.3.1.3 for the IntelRate controllers, and are omitted here for brevity.

4.6.2 Optimal Rate Allocation

In this section, we show how the OFEX controller allocates the link capacity to each passing flow in proportional fair. With the single bottleneck network, the experiment is conducted under a bottleneck bandwidth of 5Gbps with a buffer size B=120000 packets. We set the "payment" for flows in ftp groups 1 to 5 to be 36621.1, 14648.4, 24414.1, 19531.24 and 9765.6, respectively (which are just some numbers we picked randomly). The more one pays, the more bandwidth one can obtain. Obviously, zero payment leads to no bandwidth allocated). In addition, we set $\lambda_l^{(0)} = 10$, $\gamma = 1$ and $\delta = 0.01$ in the algorithm illustrated in Fig. 4.2. Note that the selection of the parameters $\lambda_l^{(0)}$, δ and γ is based on our numerous experiments and performance tradeoff, which will be investigated in Appendix J and Section 4.6.4.

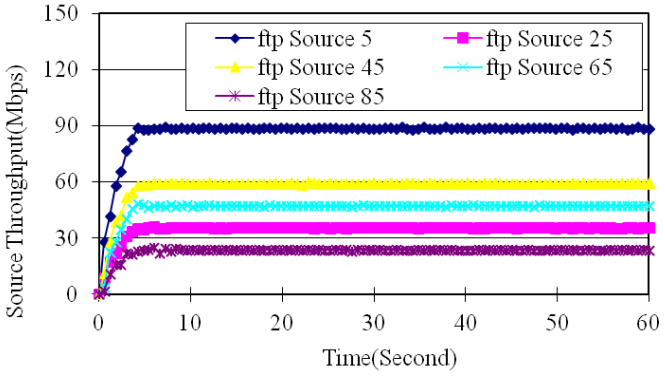


Fig. 4.8: Optimal Rate Allocation

Fig. 4.8 shows the allowed source sending rate allocated by the link to the passing flows. As shown, all the flows transmit data with a constant data rate after they reach the steady state. For example, the flows in ftp group 1 (as represented by Source 5) spend about 3.4s on the transient stage and then stabilize their throughput at 88.34Mbps. Similar observations can also be made to the sources in other ftp groups. Specifically, the allowed sending rates of the

flows from ftp groups 2 to 5 at steady state are 35.26Mbps, 59.51Mbps, 47.22Mbps and 23.70Mbps, respectively. It is also interesting to see that their throughputs are proportional to their payments. For example, the flows in group 1 has made the biggest payment (i.e., 36621.1), and obtained the biggest share of the bandwidth (i.e., 88.34Mbps). Likewise, the flows in group 5 (as represented by Source 85) pay the least, and obtain the least.

To show the link bandwidth of 5Gbps is actually allocated among the flows in a proportionally fair manner and thus optimal, we carry out some simple calculations as follows. First we add all the paid prices from the flows up (in ftp group 1-5), which is $2.1*10^6$ (obtained from (36621.1+14648.4+24414.1+19531.24+9765.6)*20). The portion of bandwidth a flow obtained is the ratio of the price it paid to the total price of $2.1*10^6$ in the link multiplied by the link bandwidth 5Gbps. Thus, 5Gbps $\times(36621.1/2.1*10^6)$ = 87.21Mbps is the allowed sending rate for flows in group 1. Similarly one can obtain the theoretical sending rates of the flows for groups 2-5 to be 34.88Mbps, 58.14Mbps, 46.51Mbps and 23.25Mbps, respectively. As seen, they closely match the experimentally obtained data rates described in the last paragraph. Therefore, the OFEX controller realizes the optimal rate allocation mechanism (i.e., proportionally fair) as originally designed for a link.

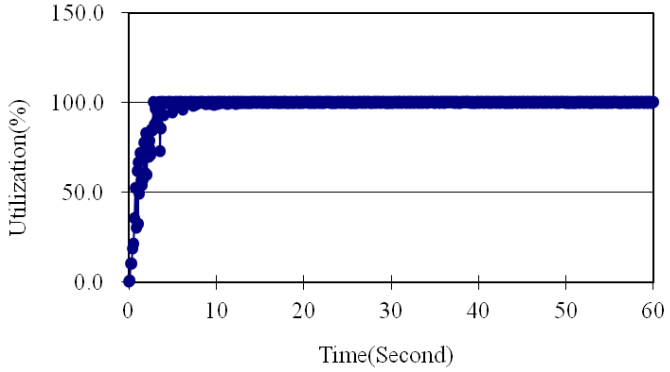


Fig. 4.9: Link Utilization

Fig. 4.9 shows the link utilization reaches 100% (i.e., the link bandwidth of 5Gbps has been fully utilized) after it converges to the steady state. Note that the first 4 seconds in Fig. 4.9 is the initial startup stage of our simulation for the flows to increase their sending rate gradually to use up the link capacity.

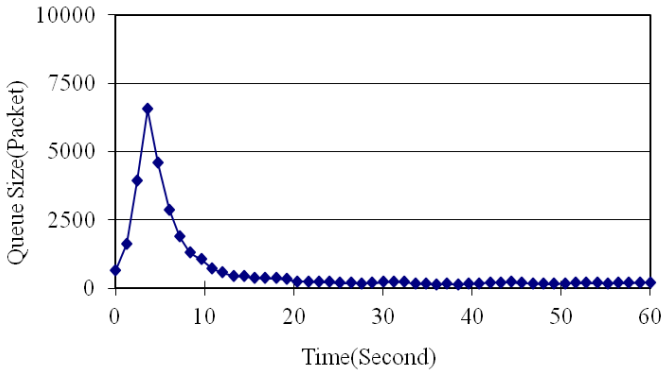


Fig. 4.10: Instantaneous Queue Size

Fig. 4.10 depicts the instantaneous queue size. It shows a gradual increase at the beginning followed by an overshoot but eventually it settles at the near-zero buffer level (i.e., $\overline{q} \approx 0$). This means the OFEX controller tries to vacate the queue and thus making the queueing delay as small as possible. This advantage is achieved by having added the instantaneous queue size $q(t)$ in our model as discussed in Section 4.1. Later on, when we make the comparison with the relatively explicit controller, you may see the relatively explicit controller tends to maintain a big queue size due to the lack of consideration on queue size in their model.

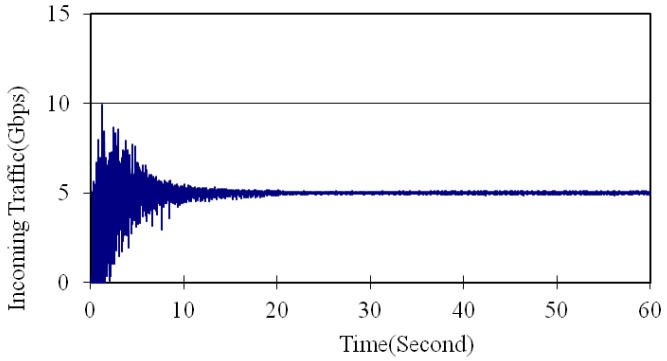


Fig. 4.11: Aggregate Incoming Traffic into the link

Fig. 4.11 shows the aggregate incoming traffic to the link operates around 5Gbps with some oscillations at the beginning after the router switched on to start. To compare Fig. 4.9 and Fig. 4.11, one can verify that the incoming traffic and the outgoing link capacity strike equal at the steady state in that they are both operating at 5Gbps when the system settles. Fig. 4.12 shows the link price which decreases from the initial $\lambda_l^{(0)}$ (i.e., 10) to the equilibrium

point λ_l^* (i.e., $\overline{\lambda}_l$ =3.83) when the system reaches the steady state.

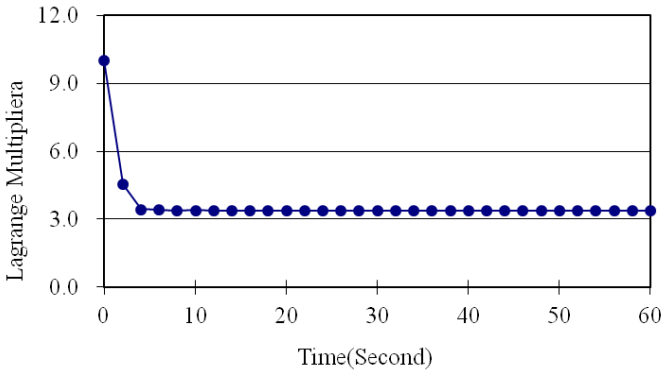


Fig. 4.12: Link Price

We can also use this experiment to verify the theoretical analysis of the system stability with time delay conducted in Section 4.4. As defined in Section 4.4, $K_1=\overline{\lambda}_l$ and $K_2=\delta(c_l-\gamma\bar{q})$. So when the system is stable in this experiment, we have $K_1=\overline{\lambda}_l$ =3.83 and $K_2=\delta(c_l-\gamma\bar{q})=0.01*(5*10^9-1*0)= 5*10^7$. Also, as defined in Table 3.2, the average RTPD is 160ms (i.e., 0.16s), which is obtained by dividing the sum of all the RTPDs by the total number of flows. Note that RTT τ >0.16s if queueing delay is included. For convenience, we use the average RTPD to approximate τ because the queue size as shown Fig. 4.10 is very close to zero. In order to verify the stability condition imposed by (I-18), one can put K_1, K_2 and τ into (I-18) now. It is not hard to find $K_1/K_2=7.66*10^{-6}<\tau/2=0.08$s, i.e., the parameters of the experiment satisfy the theoretical stability requirement, which in turn speaks to the stability of the system as shown from Fig. 4.8 to Fig. 4.12. The same approach can also be used to verify the system stability in other experiments.

4.6.3 Effect of Parameter γ on the Controller Performance

In Section 4.1, the design of the OFEX controllers requires the parameter γ to exercise some control improvement on the system. In this section, through testing different values of γ, we shall experimentally investigate what effect it would have on the control performance. To do so, we set the link bandwidth of Router 1 in the single bottleneck network to be 1Gbps, and the buffer size B=24000 packets. The weights ψ_s in the flows from ftp groups 1-5 are 12207.03, 4882.81, 9765.62, 7324.22 and 2441.4, respectively. We consecutively test the

OFEX controller by setting γ=2, 3 and 5.

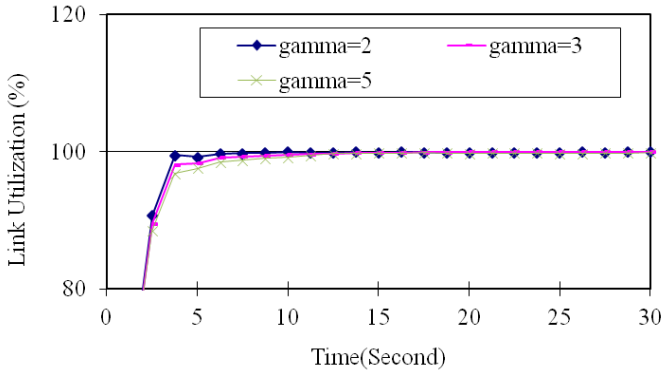


Fig. 4.13: Link Utilization under Different γ

Fig. 4.13 demonstrates the link bandwidth utilization under different values of γ. They show the similar trend, i.e., after the startup, the bandwidth capacity is gradually used up and reaches the steady state afterwards. The difference is the rise time [DoBi08] before they reach the steady state. When γ=2, the rise time is around 3.75s, however, when γ=3 and γ=5, the rise time is lengthened to 8.75s and 11.25s, respectively. Such a trend shows that a bigger γ affects the response time of the OFEX controller, say, the bigger the parameter γ, the longer the response time.

From Fig. 4.13, the longer response time is directly related to the bandwidth utilization performance. When certain capacity of bandwidth is available, obviously the faster it is fully utilized, the better use of the network resources. In this regard, a smaller γ is a good choice for the OFEX controller.

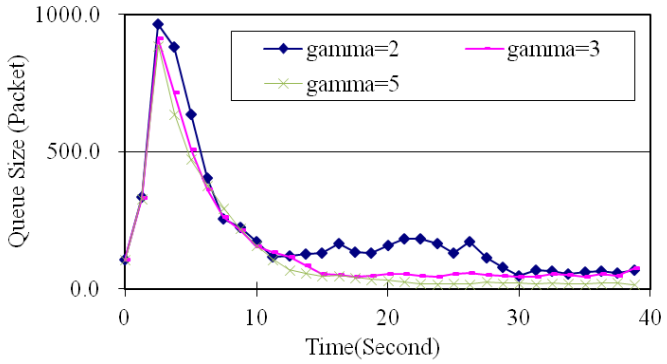


Fig. 4.14: Instantaneous Queue Size under Different γ

The instantaneous queue size is depicted in Fig. 4.14. The big buildup in the first 10

seconds is the overshoot of the queue size after the router starts, which implies the desire that the OFEX controller would like to fast consume the available bandwidth up. When γ=2, the peak value of the overshoot is 960 packets, and then when the overshoot subsides after t=10s, the queue size oscillates at around 130 packets until t=30s. After t=30s, the queue size decreases and is then maintained at around 60 packets. For γ=3, the peak value of the overshoot is 910 packets, which is a bit lower than γ=2. Furthermore, after the overshoot at t=10s, it directly settles at a queue size of around 60 packets. When γ=5, the maximum value of the overshoot is 880 packets, which is lower than that under γ=3, the same observation can be seen after t=10s, i.e., the queue size under γ=5 is 18 packets and lower than that under γ=3.

Fig. 4.14 shows that the bigger the parameter γ, the better the queue performance in terms of its size because a lower queue size is always preferred in order to have small queueing delay in a network. In this regard, a bigger γ is a good choice for the OFEX controller.

As a summary to the performance demonstrated in Fig. 4.13 and Fig. 4.14, there is a conflicting requirement for the value of γ between the bandwidth utilization and queue size performance. Therefore, neither a too small γ nor a too big γ is a good decision for the OFEX controller; instead it should be a tradeoff by considering the performances of the bandwidth utilization and the queue size of a router. The value of γ in all our experiments is selected this way experimentally, i.e., we always choose a value for γ by taking care of the performances of above both measures.

4.6.4 Robustness to Large Network Parameter Changes

As with the experiments in Section 3.3.4 for the IntelRate controller, in view of the real world dynamic Internet traffic, below we shall investigate the performance of the OFEX controllers in the single bottleneck network when faced with drastic network changes. That is, we are interested in the step responses (which are usually adopted in classical control theory [DoBi08]) to the step changes in the number of flows or the available bandwidth.

4.6.4.1 Sudden Traffic Changes

We are going to test our controller using a bottleneck bandwidth of 4Gbps, $\lambda_l^{(0)} = 10, \delta = 0.01$,

γ=5, and B=96000 packets. The weight Ψ_s of the flows from ftp groups 1-5 are 30517.6, 10986.33, 24414.1, 18310.5 and 7324.2, respectively.

To create traffic changes, we only allow the flows running in ftp group 1-4 in the beginning, and then make the ftp flows in group 5 swarm in at t=40s, and run for 20 seconds before retreating from the network.

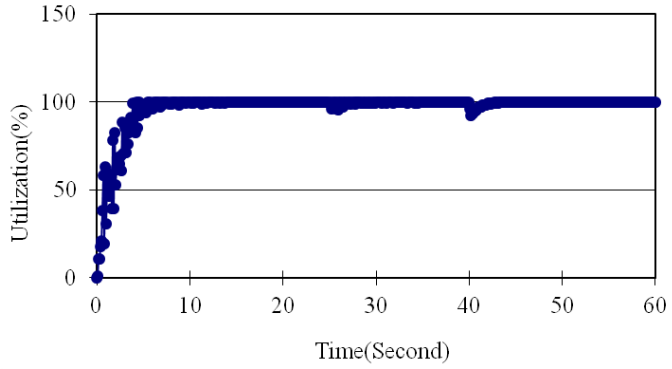


Fig. 4.15: Link Utilization under Sudden Traffic Change

The link utilization performance during the traffic changes is depicted in Fig. 4.15. After a transient in the first 7s, the link bandwidth is fully utilized at 100%. The joining in of the 20 flows in group 5 at t=20s does not have an effect on it. When the 20 flows withdraw from the traffic at t=40s, the link utilization decreases a little bit, i.e., about 5%, but quickly recovers after 3 seconds.

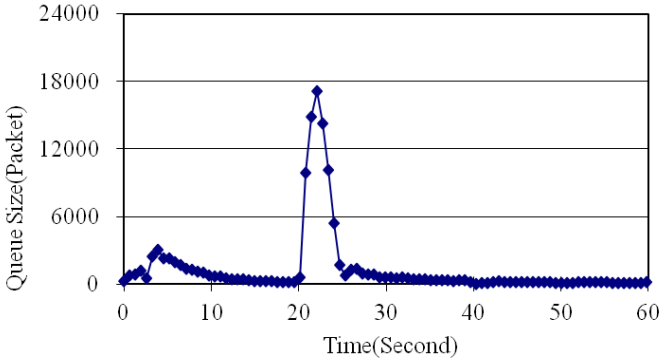


Fig. 4.16: Instantaneous Queue Size under Sudden Traffic Change

Fig. 4.16 shows the dynamics of the instantaneous queue size. After the controller starts, the queue accumulates a little bit until it reaches about 2000 packets at t=3s, and then goes back to the near-zero position. At t=20s, the sudden swarm-in of the 20 flows in ftp group 5 surges the queue size to almost 18000 packets. Compared with the buffer size

B=96000packets, such a surge is just 18.75% of the buffer capacity, so there is no packet loss. Furthermore, the queue size quickly drops back to the near-zero level after 7s, which is the position the queue size stays most of the time during the simulation.

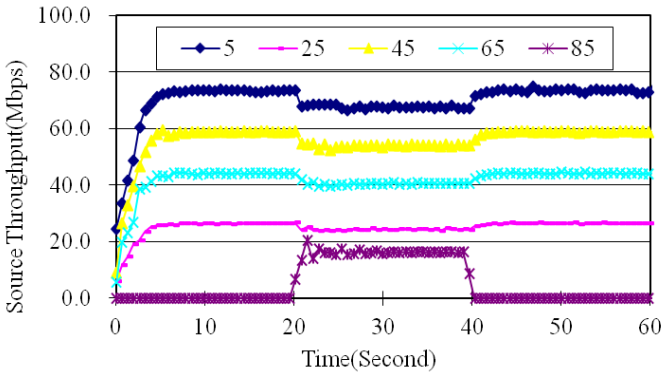


Fig. 4.17: Source Dynamics under Sudden Traffic Change

Fig. 4.17 is the dynamics of the source sending rate. After the system reaches the steady state at t=6s, each flow in groups 1-4 (represented by flows 5, 25, 45, 65) obtains a share of bandwidth to transmit their data. When the flows in group 5 (represented by flow 85) join in at t=20s, all the flows in groups 1-4 have to decrease their sending rate to make room for the new traffic. Furthermore, the bigger the original share of the bandwidth the flows obtain, the more their sending rate is decreased because of the proportional fair mechanism of the OFEX controller. For example, the flows in group 1 which have the biggest throughput (represented by flow 5) decrease their sending rate the most (a decrease of 6.7Mbps), whereas the flows in group 2 which have the smallest throughput (represented by flow 25) decrease their sending rate the least (a decrease of only 0.8Mbps). The decreases (4.9Mbps and 3.6Mbps) observed for other groups in between are also proportional to their original share. All flows in groups 1-4 can recover their sending rates only after the flows in group 5 withdraw from the link at t=40s.

As a matter of fact, the source performance in Fig. 4.17 explains why the link utilization in Fig. 4.15 can always be controlled around the target bandwidth of 4Gbps. This is because when new traffic comes in, the existing flows will decrease their source sending rate according to the proportional fairness as discussed in Section 4.6.2 so that the total traffic going into the router can be maintained at the target bandwidth level after the new traffic swarms in.

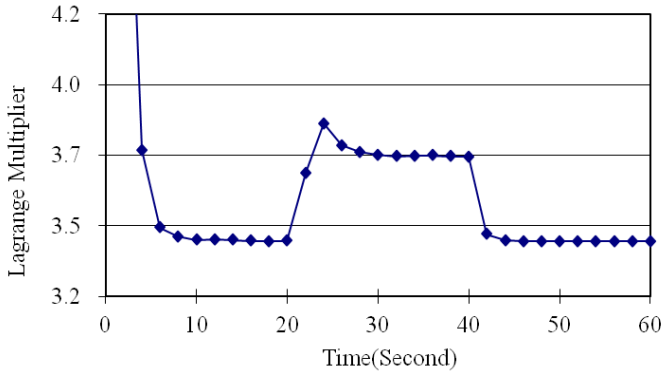


Fig. 4.18: Link Price Dynamics under Sudden Traffic Change

The behavior of the sources in Fig. 4.17 can also be explained from the dynamics of the Lagrange multiplier whose variations can acts like the fluctuation of the "unit link bandwidth price". As shown in Fig. 4.18, after the warm-up stage, the unit link price settles at 3.39. When the new traffic joins in, it increases to 3.83 at first before settling at 3.70. Thus the "unit bandwidth price" has gone up due to the tight bandwidth resource in the router. Consequently, the amount of bandwidth that each source can "purchase" decreases because they are paying the same total amount of money. This is why the source sending rate begins to decrease at t=20s in Fig. 4.17 (also as expected from Eqn. (4-13)). Similar analysis can be applied to t=40s, when the 20 flows retreat, and the unit link price falls back to the original price level. Therefore, the remaining flows can now send at their original rates.

4.6.4.2 Sudden Bandwidth Changes

The experiment is conducted under a bottleneck bandwidth varying between 7-8Gbps. We set $\gamma = 5$, $\lambda_l^{(0)} = 10$, $\delta = 0.01$ and B=192000 packets. The weight ψ_s in the flows from ftp groups 1-5 are 46386.72, 19531.25, 30517.58, 24414.02 and 12207.03, respectively.

The nominal bandwidth of the router is set to 8Gbps. To create bandwidth changes, the bandwidth will decrease from the nominal value of 8Gbps to 7.5Gbps at t=40s first, and then decrease again at t=60s to 7Gbps. With respect to the nominal bandwidth, these two variations represent a change of 6.25%, and 12.5% in outgoing capacity of the router, respectively. Finally, at t=80s, the bandwidth recovers to the nominal value of 8Gbps.

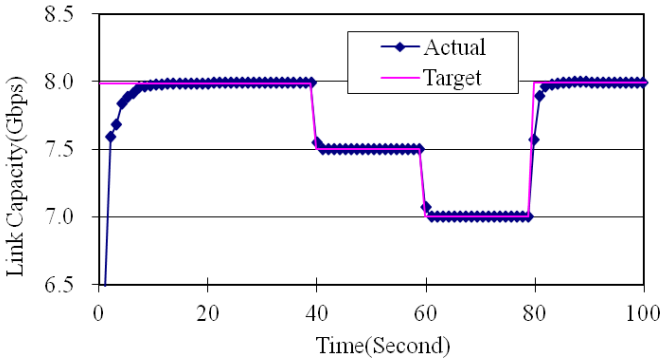


Fig. 4.19: Bandwidth Variation

Fig. 4.19 compares the target bandwidth and the actual bandwidth obtained. Generally, the actual link bandwidth utilization well follows the change of the target bandwidth. After the link utilization reaches the steady state at t=6.5s, it stabilizes at the target bandwidth of 8Gbps in the rest of the first 40 seconds. In the following, we shall make observations when the bandwidth decreases (at t=40s and t=60s) and increases (at t=80s).

Upon the first bandwidth decrease at t=40s, as seen in Fig. 4.19, the actual bandwidth utilization goes down to the target bandwidth 7.5Gbps in 1 second. The second bandwidth decrease from 7.5Gbps to 7Gbps at t=60s presents the similar trend. When the bandwidth suddenly jumps back to 8Gbps at t=80s, the actual bandwidth obtained reaches the target bandwidth 8Gbps within 3 seconds, and stays there until the end of the simulation.

In summary, our controller can obtain a link utilization well matched to the target bandwidth with the exception of the few seconds of initial transients.

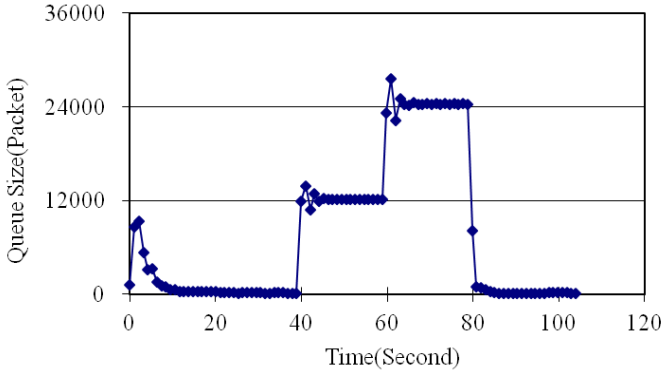


Fig. 4.20: Instantaneous Queue Size under Bandwidth Change

Fig. 4.20 depicts the instantaneous queue size. When the controller first starts, the queue size shows a quick buildup of maximum 9300 packets in the first 2 seconds, and then drops

back to nearly zero (i.e., about 170 packets). When the bandwidth decreases from 8Gbps to 7.5Gbps at t=40s, the queue length sharply rises to 12000 packets in response to the bandwidth loss (i.e., 0.5Gbps), and stably stays there until t=60s. The second queue increase happens at t=60s due to the second bandwidth downsizing from 7.5Gbps to 7Gbps, and this time the queue length rises to 24000 packets to deal with a bandwidth loss of 1.0Gbps from the nominal bandwidth. As noticed, in both bandwidth decrease cases, the queue size increase is far below the buffer size of 192000 packets (which was designed according to the network BDP), and thus there is no any packet loss. When bandwidth increases to 8Gbps from 7Gbps at t=80s, the queue size sharply decreases to the original steady state again, i.e., about 170 packets. This is because the outgoing capacity of the router has gone back to its nominal bandwidth.

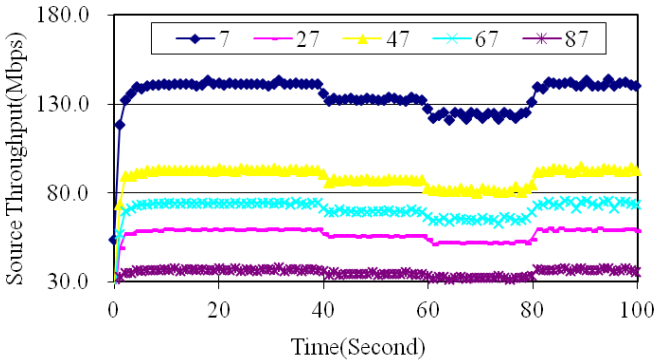


Fig. 4.21: Source Dynamics under Bandwidth Change

Fig. 4.21 shows the source dynamics upon bandwidth variations. As shown, after the controller reaches the steady state in the first 6.5 seconds, the sources adjust their sending rate as expected each time the link bandwidth changes.

Let us first examine flow 7 as a representative of the flows from ftp group 1. When the link bandwidth decreases from 8Gbps to 7.5Gbps at t=40s, the flows decrease their sending rates from 141Mbps to 133Mbps in 2 seconds and then stabilizes at that level. When responding to the second decrease of the bandwidth from 7.5Gbps to 7Gbps at t=60s, the flows decrease their sending rates from 133Mbps to about 122Mbps in 2 seconds. When the bandwidth increases to the nominal 8Gbps from 7Gbps at t=80s, the flows increase their sending rate to the original 141Mbps. The flows in other groups (represented by flows 27, 47, 67 and 87) demonstrate similar behaviors upon bandwidth variations. The only difference is that they operate at a lower throughput due to the fact that they pay a lower price to the router.

Their steady share of the bandwidth in each time segment of different link bandwidth can be explained by the proportional fairness as discussed in Section 4.6.2. However, the RM concept can be used to explain the changes at t=40, 60 and 80s as will be discussed with Fig. 4.22.

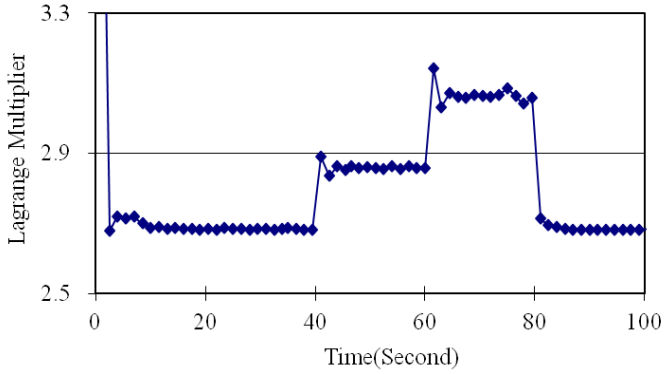


Fig. 4.22: Link Price Dynamics under Bandwidth Change

Fig. 4.22 shows how the unit link price (represented by the Lagrange multiplier value) varies with respect to the bandwidth change in the router. After the controller starts, the unit link price converges to 2.68 from the original value $\lambda_l^{(0)} = 10$ after 2 seconds and remains at this level. When the bandwidth decreases from 8Gbps to 7.5Gbps at t=40s, the unit link price goes up to 2.86 in 0.5s. Such a change in the unit link price reflects the RM principle, i.e., the price of a commodity will increase when its supply shrinks, just like here the available bandwidth shrinks to 7.5Gbps from 8Gbps. The same observation can be obtained under the second bandwidth shrinkage after t=60s, where the unit link price increases to 3.08 from 2.86, and stabilizes there until t=80s. Finally, the price decreases to 2.68 at t=80s when the bandwidth increases to the nominal value of 8Gbps from 7Gbps.

According to Eqn. (4-13) in Section 4.2, it is readily seen the changes of the source sending rate (as shown in Fig. 4.21) well reflects the above evolution of the unit link price because Eqn. (4-13) shows how these two variables are related.

4.6.5 Comparison with Other Controllers

Below, we shall make comparisons with the counterpart of the OFEX controller.

4.6.5.1 Performance Improvement over the Relatively Explicit Controller

The purpose of this experiment is to verify the superiority of the OFEX controller over the

relatively explicit controller in improving the throughput and convergence speed of the multi-bottlenecked users. Besides, we will compare the $q(t)$ of the two controllers as well.

The comparison of the two controllers is conducted with the multiple bottleneck network settings described in Section 4.6.1, where $\delta = 0.05$, $\lambda_l^{(0)} = 20$ in all the routers. We also use the same source weight vector ψ_s in the two controllers. For example, sources in subnet 0 pays a price of 7324.20 in both controllers, and sources in subnet 20 pays a price of 2441.40 in both controllers. For brevity, we skip enumerating the prices paid by other sources. In addition, upon light traffic, the routers request a minimum payment of 0.001, which is set close to zero. Two multi-bottlenecked flows will be used to compare the performances of the two controllers. They are called Flow-A, and Flow-B representing flows that pass through three bottlenecks and four bottlenecks respectively.

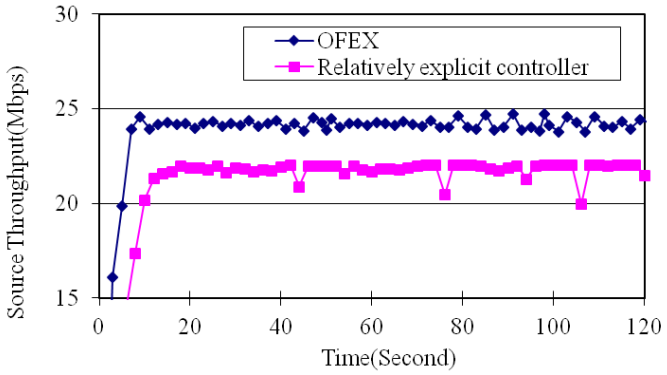


Fig. 4.23: The Source Sending Rate in Subnet 10

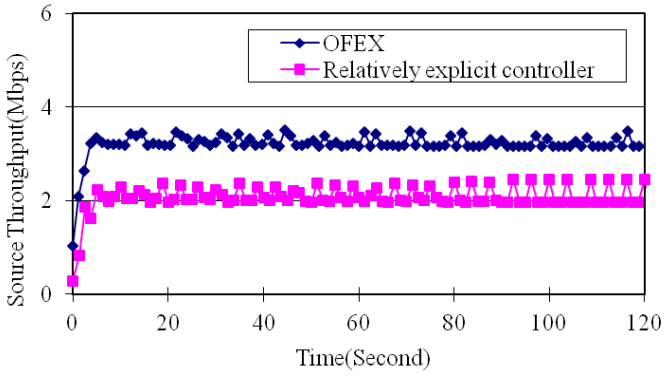


Fig. 4.24: The Source Sending Rate in Subnet 19

Fig. 4.24 shows the instantaneous source sending rate of Flow-B in subnet 19, where the flows have 4 bottlenecks (i.e., from Router 1 to Router 4). As with Fig. 4.23, the same trend

can be observed. The sources with the OFEX controller have higher sending rate and shorter convergence time than those with the relatively explicit controller. In particular, as shown in Fig. 4.24, the OFEX controller converges 1.7 seconds faster (i.e., an improvement of 35.42%) than the relatively explicit controller in that the former spends 3.1 seconds and the latter spends 4.8s converging to the steady state.

Upon comparing the Flow-B throughput between the two controllers in Fig. 4.24, the OFEX controller shows much better performance than the three-bottlenecked flows observed in Fig. 4.23. Specifically, Fig. 4.24 shows that the sending rate of the flows with the OFEX controller is 3.17Mpbs, and the sending rate with the relatively explicit controller is about 2.01Mbps. The difference between them is 1.16Mbps, which means the OFEX controller achieves a throughput improvement of 57.71%.

In summary, as shown in Fig. 4.23 and Fig. 4.24, the OFEX controller conspicuously improves the source throughput and convergence speed of the multi-bottlenecked users. In particular, the more bottlenecks a flow passes, the better throughput performance the OFEX controller shows (e.g., Flow-B illustrated above).

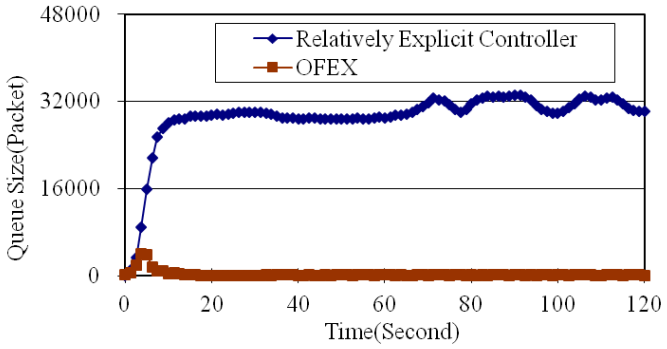


Fig. 4.25: $q(t)$ in Router 1

Fig. 4.25 shows the instantaneous queue size $q(t)$ of Router 1. The OFEX controller maintains a near-zero instantaneous queue size except the first 15s (i.e., the router starting stage). In contrast, the relatively explicit controller builds up a big queue size which is close to 32000 packets. Besides, its queue size oscillates occasionally. For example, beyond t=60s, the queue size of the relatively explicit controller appears to go into some kind of "sinusoidal" oscillation.

In Fig. 4.25, the effectiveness of introducing $q(t)$ into the system model is confirmed again by the multi-bottleneck network, i.e., it greatly improves the queueing performance.

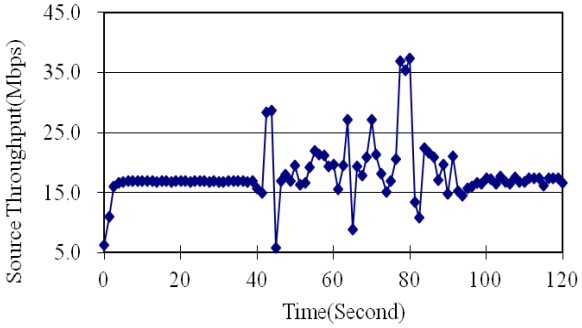
The $q(t)$ in other routers (i.e., Routers 2, 3 and 4) shows the similar trend, and we omit to discuss them here for brevity reason.

4.6.5.2 Comparisons in the Single Bottleneck Network

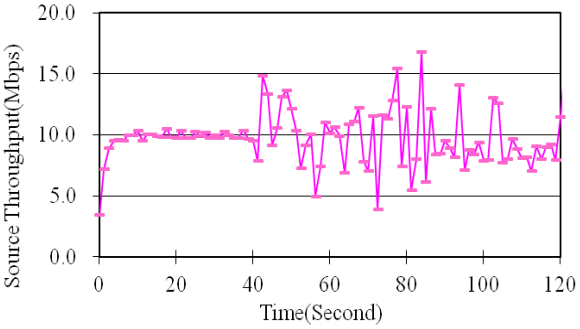
As discussed in Section 4.1, the existing NUM-based relatively explicit controller has no way to tackle the link bandwidth dynamics. The OFEX controller has introduced $q(t)$ in the system model as a remedial measure to counter such an impact.

The purpose of this experiment is to show the improved performance of the OFEX controller in dynamic link bandwidth conditions over the relatively explicit controller. In the meanwhile, we also compare with the XCP controller because it also has problems upon dynamic bandwidth.

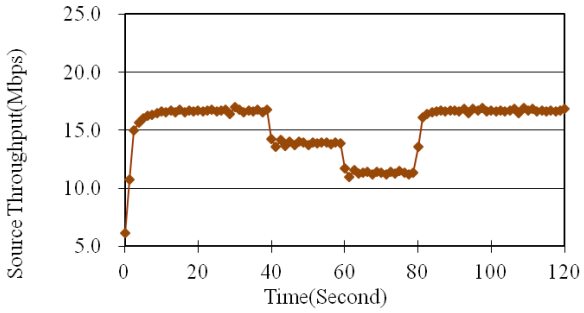
In this experiment, we set up a 1Gbps link in Router 1 of the single bottleneck network Fig. 4.7a. To produce the link bandwidth dynamics, the link bandwidth will be reduced to 880Mbps at t=40, and then to 675Mbps at t=60s. Again, this is one common scenario in contention-based networks. Finally, the bandwidth will recover to 1Gbps at t=80s.



(a) The Relatively Explicit Controller



(b) The XCP Controller



(c) The OFEX Controller

Fig. 4.26: Source Dynamics Comparison

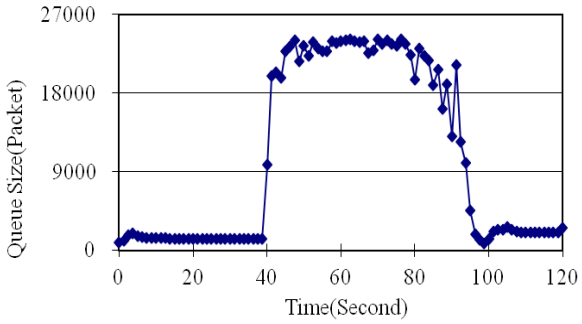
The parameters of the OFEX controller are $\delta = 0.05$, $\lambda_l^{(0)} = 20$, $\gamma = 10$, and B=24000 packets. The weights ψ_s in the flows from ftp groups 1-5 are 12207.03, 4882.81, 9765.62, 7324.22 and 2441.4, respectively. The relatively explicit controller uses the same parameters and configuration as with the OFEX controller except the parameter γ which it does not have. The parameters of the XCP controller are $\alpha = 0.4$, and $\beta = 0.226$ as set in [KaHa02]. The three controllers have the same buffer size B=24000 packets and the same configuration as given in Table 4.1.

Fig. 4.26 demonstrated the ftp source sending rate dynamics of the three controllers upon changes of the link bandwidth. Fig. 4.26a is the source behavior of the relatively explicit controller. After the transient in the first few seconds, its source sending rate stabilizes at 16.73Mbps. When the first bandwidth reduction happens at t=40s, its sending rate renders oscillation (i.e., unstable behavior) which fluctuates severely in the first cycle and then subsides until the second bandwidth reduction at t=60s. From Fig. 4.26a, one can see the source fluctuates more severely in the second bandwidth reduction (i.e., between t=60s and t=80s) than that in the first bandwidth reduction as it presents more severe oscillations. When the link bandwidth goes back to the original value at t=80s, its sending rate gradually recovers to the previous smooth behavior. Note that the controller cannot stabilize during the bandwidth reduction periods (i.e., between t=40 and t=80s). The reason could be due to the queue built up in the relatively explicit controller. To be seen in Fig. 4.27a, the relatively explicit controller has no queue control measure, so its queue size builds up quickly to take up the whole buffer size after the bandwidth reduction at t=40s, and packet loss happens. Its queue size has become unstable, so does the source sending rate.

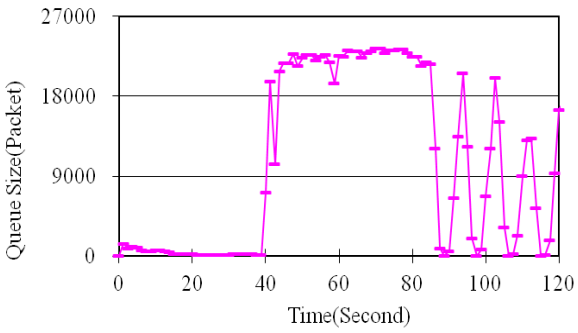
The source behavior in the XCP controller depicted in Fig. 4.26b shows the same unstable behavior as with the relatively explicit controller during the bandwidth reduction periods. The difference is that the sending rate of XCP cannot recover to stability after t=80s, instead it oscillates throughout the rest of the simulation.

In contrast, the OFEX controller as shown in Fig. 4.26c demonstrates stable performance all the time irrespective of bandwidth dynamics. That is, when the bandwidth shrinks at t=40s and 60s, or when the bandwidth recovers at t=80s, there are hardly any fluctuations. In summary, the introduction of $q(t)$ in the OFEX controller model has greatly

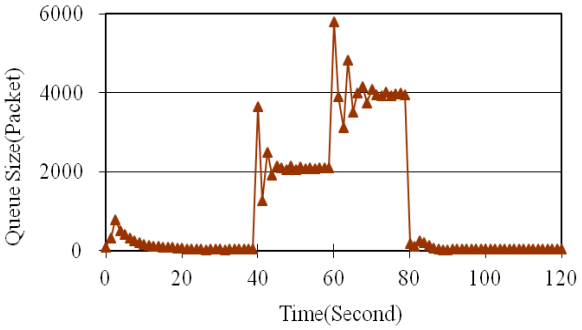
improves its queueing stability against the link bandwidth dynamics, and makes it outperform its counterparts.



(a) The Relatively Explicit Controller



(b) The XCP Controller



(c) The OFEX Controller

Fig. 4.27: Instantaneous Queue Size Comparison

Fig. 4.27 shows the instantaneous queue size $q(t)$ of the three controllers. Fig. 4.27a shows that the relatively explicit controller has a queue length of 1300 packets after the router is started. When the bandwidth undergoes the first reduction at t=40s, its queue size sharply increases to the full capacity of the buffer, i.e., 24000packets, and then stays there with some oscillations. The packet loss becomes unavoidable thereafter. The buffer is kept overflowed through the second bandwidth reduction from t=60s. Only after the bandwidth goes back to the original value at t=80s, the queue size of the relatively explicit controller begins to decrease and eventually stays at 2300 packets.

Fig. 4.27b depicts the queue size of the XCP controller. It keeps a low occupancy queue after the transient in the first 12 seconds. Like the relatively explicit controller, the queue size of the XCP controller sharply rises to 24000 packets after the first bandwidth shrink until t=80s when the link bandwidth begins to recover. But unlike the relatively explicit controller, the XCP controller cannot settle down to its original smooth queue after t=80s; instead it

keeps on fluctuating.

In contrast, the queue size of the OFEX controller depicted in Fig. 4.27c shows better performance than both that of the relatively explicit controller and the XCP controller. The OFEX controller renders a near zero queue size (around 30 packets) after the router is started in the first 8 seconds. Upon the first bandwidth shrink at t=40s, the queue size of the OFEX controller rises to a level of 2000 packets. The second bandwidth shrink at t=60s elevates the queue size to around 4000 packets, which is only 1/6 of the buffer size, so there is no packet loss at all. When the bandwidth recovers to its original capacity, the queue size of the OFEX controller goes back to its smooth near zero level accordingly.

We also make the observation that the XCP controller has also considered the queue size in its controller model where the queue size parameter is set to 0.226 (which was adopted in [KaHa02]). We do not think this is big enough to vacate the queue upon link dynamics because it still has big queue size as shown in Fig. 4.27b. However, to increase this parameter (i.e., a value bigger than 0.226) does not solve the problem because instability can occur as analyzed and proved in [KaHa02].

In summary, the OFEX controller shows a much better performance in its queue size due to the introduction of the instantaneous queue size $q(t)$ as a controlled performance in its model, and thus significantly improving the source sending rate performance in terms of stability.

Discussion: Please note the conclusion in [KeRa08] and [VoRa09] that the queue size feedback may cause the queue to be less accurately controlled is based on their small buffers design. The buffer in their controller is "small enough so that it is no longer possible to explicitly model the queue [VoRa09] ". However, the buffer in our OFEX controller is designed according to the BDP. That is, the buffer is large enough to hold the traffic of one round trip time which is what router manufacturers typically use nowadays as a rule of thumb. The concern that big buffers can cause big queueing delay and jitter in [DaMc05] is because they are handling the *implicit* congestion control protocol---TCP, which is well known in its design to fill any buffer and cause the buffer to overflow and fluctuate (and therefore jitter) [DaMc05]. Our OFEX controller is an *explicit* congestion control protocol and does not share the TCP "saw tooth" problem with its precise rate control where each router just allows certain amount of rate that sources can send.

4.6.6 Summary

Under the single bottleneck and the multiple bottleneck scenarios, the effectiveness and the superiority of the OFEX controller have been demonstrated and verified via simulation.

The optimal rate allocation test demonstrates how the OFEX controller can allocate link bandwidth in a proportionally fair manner. We also illustrate how the parameter γ affects the controller performance. The tests using sudden traffic changes and bandwidth variation verify that the new scheme is able to effectively work in the time-varying dynamic networks. Finally, the comparison with the other controllers demonstrates the superiority of the OFEX controller in terms of source performance and the ability against link bandwidth dynamics. How the convergence speed is affected by some parameters such as link bandwidth, the initial value of Lagrange multiplier or step size is experimentally demonstrated in Appendix J.

4.7 Concluding Remarks

Using the convex optimization technique, we have formulated and investigated an NUM-based controller, called the OFEX (Optimal and Fully Explicit) congestion controller.

In contrast to the existing relatively explicit controllers, the OFEX controller provides fully explicit congestion signals by exercising the link-wise proportional fairness and network-wise max-min fairness. The distinctiveness of the OFEX controller is that it overcomes the drawback of the existing relatively explicit congestion controllers that bias the multiple-bottlenecked users. Furthermore, a time-varying feature was considered by incorporating the only state variable $q(t)$ in the algorithm, which makes the OFEX controller more adaptable against link bandwidth variations and traffic dynamics.

The OFEX controller has been proved being able to converge to its equilibrium at least with a rate of a geometric series. Also, the control system has been proved locally stable under certain conditions before its robustness analysis is conducted. A series of systematic OPNET simulations have verified the effectiveness and performance improvement of the OFEX controller. For brevity, the computational complexity of the controller can be found in Appendix I where the complexity comparison with the relatively explicit controller shows the feasibility of the real-time implementation of the OFEX controller.

Chapter 5
Wireless Local Area Network Application

The channel capacity of a Wireless Local Area Network (also called Wi-Fi), abbreviated as WLAN, is usually much less than the wired network it is connected to. This can present a significant bottleneck for traffic flowing from the wired network to the wireless network [NyDa07]. Therefore, it is a good application of our controllers in order to test their capabilities. Please note that WLAN and the backbone network connecting to it (e.g., the one to see in Fig. 5.1) can be regarded as a special case of multi-bottlenecked networks because the last router here is always an AP (Access Point), which is used to bind the wired network and the wireless network.

After summarizing the operation and bandwidth characteristic aspects of a WLAN in Section 5.1, we shall provide the details of OPNET simulation setting in Section 5.2 in order to test the performance of the IntelRate controller in Section 5.3 and the OFEX controller in Section 5.4.

5.1 WLAN Application and Bandwidth Characteristics

We consider an IEEE 802.11 WLAN [KrRo12] with an AP interfacing the wired and wireless parts of the network. In addition to the big disparity in channel capacity between the wired and wireless parts, the bandwidth variations will be a good test for the practicality of our congestion controllers. The wireless channel is more interference-prone as well as contention-prone, and its bandwidth can vary more unpredictably [NyDa07]. Interference refers to unwanted signals which may alter or disrupts a useful signal as it travels in a channel between a source and a receiver, and can be measured by SNR (Signal-Noise-Ratio). On the other hand, contention arises as the result of different stations trying to access a broadcast medium randomly.

According to the IEEE 802.11 wireless standards, the router (or an access point) can have several nominal bandwidths, e.g., the IEEE 802.11b [HaKe99] specifies 4 nominal bandwidths: 1Mbps, 2Mbps, 5.5Mbps and 11Mbps. Called link adaptation, the router can dynamically switch its bandwidth among these different data rates according to the SNR,

RSSI (Received Signal Strength Indicator), missing ACKs or the data error rate [WuLi01, QiCh02, PaCh03, ZhMa07]. Such kinds of bandwidth switching between nominal bandwidths are considered to be major bandwidth variations.

Bandwidth seen by the upper layers (e.g., transport layer) in wireless networks also varies/adapts in response to different factors arising from multiple access contention (e.g., scheduling, back off, transmission probabilities) in the MAC layer or link sharing (e.g., half-duplex) in the physical layer [JeBa03, AbRi06]. Consequently, the effective bandwidth available to the upper layers can fluctuate below the nominal bandwidth values. Usually the extent/amplitude of these oscillations is smaller than that caused by link adaptation, and they are referred to as minor bandwidth variations.

The above two bandwidth variations are important considerations in the design of a congestion control protocol in the transport layer if the control protocols have to rely on the accurate bandwidth values; otherwise it will suffer from performance degradation problems due to bandwidth mis-estimations. In the following, we shall investigate the performance of the IntelRate controller and the OFEX controller at the transport layer of a WLAN subject to the above bandwidth characteristics.

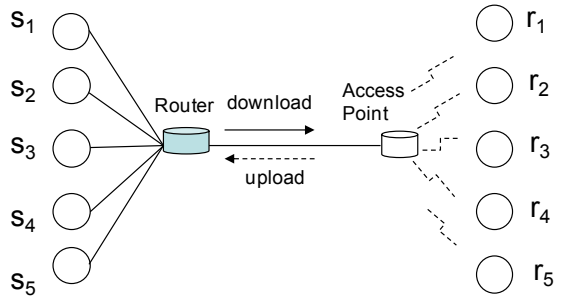


Fig. 5.1: WLAN Simulation Network Topology

5.2 WLAN Simulation Setup

The network topology used in our performance evaluation is depicted in Fig. 5.1, in which the IEEE 802.11b WLAN is used to be connected to a wired high-speed network via an AP. The network consists of 5 source-destination pairs, where s_i is the wired node and r_i is the wireless node. The backhaul (i.e., the connection between the router and the AP) has a propagation delay of 100ms and bandwidth of 1Gbps. The bandwidth between s_i and the router is 100Mbps. The nominal data rate of the AP is 11Mbps, and may be varied. Such a

WLAN topology forms a bottleneck at AP because of the bandwidth disparity. The IntelRate controller or the OFEX controller is located in the AP to exercise the congestion control.

We use the Application and Profile modules of OPNET to generate traffic in the sources s_i. In the Application module, we choose video (this module also has other types of traffic available such as ftp, http, audio or email) as our network traffic because, like ftp, video (such as IPTV, movies, online gaming) is a very important congestion maker in wireless networks nowadays. The IntelRate controller or the OFEX controller adjusts the video generation rate of the Application module so that the traffic that the sources produce will not overwhelm the AP (i.e., the wireless LAN). The average packet/frame size of the video is 1300 bytes and distributes between [360, 1500] bytes.

5.3 The IntelRate Controller Performance

To evaluate the IntelRate controller performance in Fig. 5.1, we set the TBO of the IntelRate controller to 60 packets. To make the experiment more stringent, we make the source of each flow of the IntelRate controller greedy by setting its desired sending rate to 3Mbps. Obviously, the nominal 11Mbps of the wireless bandwidth cannot satisfy such requirements from the 5 sources which have to fairly compete according to max-min fairness. In the sequel, we shall demonstrate the response performance of the controller to traffic changes, and make comparison to other most recent controllers.

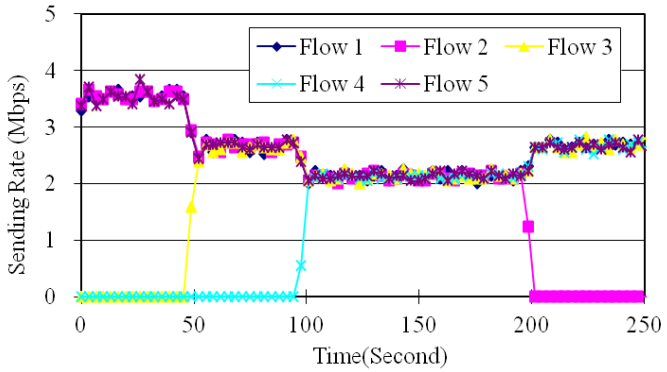


Fig. 5.2: Source Dynamics upon Traffic Change

5.3.1 Response to WLAN Traffic Changes

Network traffic change can happen due to joining, finishing transmission or connection broken in wireless networks. This experiment uses the following scenario to simulate the traffic changes. In the first 50 seconds, only video flows from s_1, s_2 and s_5 are in operation.

The flow from s_3 and s_4 will join in the traffic at t=50s and t=100s, respectively. The flow from s_2 finishes its transmission at t=200s while other flows remain in operation.

Fig. 5.2 shows the sending rate dynamics of each source during the traffic change process. In the first 50s, the flows 1, 2 and 5 share the 11Mbps bandwidth each about 3.3Mbps. At t=50s, after the flow 3 joins in, the sending rate of the flow 1, 2 and 5 decreases and the 4 flows each now shares 2.75Mbps. At t=100s, after the flow 4 joins in, each flow shares 2.2Mbps. After the flow 2 withdraws at t=200s, the remaining 4 flows each increases to share 2.75Mbps again. As shown in Fig. 5.2, the sending rate of sources maintains stable upon each traffic change and shows smooth performance.

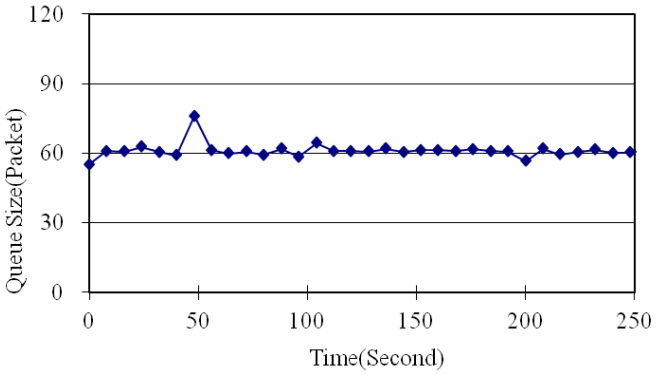


Fig. 5.3: IQSize in Access Point

Fig. 5.3 shows that the IQSize is well controlled and operating around the TBO of 60 packets. The traffic changes cause IQSize fluctuations at t=50s, 100s and 200s, but the influence is small and they can quickly settle back to the TBO.

Overall, Fig. 5.2 and Fig. 5.3 demonstrate good stability and fast response of the IntelRate controller upon traffic changes in wireless network as done in the wired network.

5.3.2 Comparison with Other Controllers in WLAN

We compare the IntelRate controller with the most recent WLAN-friendly congestion controllers, i.e., QFCP [PuHa08] and Blind [AbAr11] (we didn't choose ErrorS or MAC from the same paper to do the comparison because ErrorS is an interchangeable algorithm of Blind and faces the same problem while MAC is too complicated to be put into practice so far), and they are the enhancements of XCP in reducing the bandwidth estimation errors as discussed before. QFCP estimates the bandwidth by observing the average link output and the average queue size, while Blind infers the bandwidth using queue speed and a

dynamically heuristic calculation of the queue threshold. The parameters of QFCP and Blind controllers in the simulation are the same as those used in [PuHa08] and [AbAr11]. Specifically, for QFCP, α=0.1, β=0.5, w=0.2; for Blind, α=0.4, β=0.226, ρ=0.22, Q_x=0.54·B, τ=0.225. The buffer size B=600 packets in all the controllers.

5.3.2.1 Performance under Fixed Wireless Bandwidth

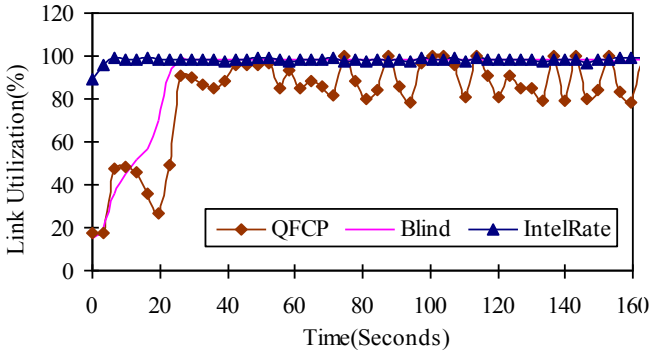


Fig. 5.4: Link Utilization

Fig. 5.4 shows the bottleneck utilization which is the ratio between the actual bottleneck throughput and 11Mbps in this case. Since the controllers do not know the bottleneck bandwidth at the beginning, in order to fully utilize the available bandwidth, they need to probe (or say, to estimate) it. As seen, QFCP spends about 20s approaching the 100% utilization (i.e., fully utilizing 11Mbps). There are oscillations in both its probing stage (when t<20s) and steady state (when t>20s). The fact that there are still oscillations in its steady state between 80% and 100% shows that the QFCP has a bandwidth under-estimation problem even though its over-estimation issue is addressed by using the router output as the link bandwidth. Blind spends a similar amount of time on the probing but shows smoother steady-state utilization (overlapping with the IntelRate) than QFCP. In contrast, the IntelRate controller only spends about 3s reaching 100% link bandwidth utilization. Furthermore, it shows the same stable performance as Blind in the steady state.

The reason that QFCP and Blind take a longer probing time is due to the probing process (see Equation (6) in [PuHa08] and Equation (18) or (19) in [AbAr11]) they use to explore the available bandwidth. In comparison, the IntelRate controller aims at building the queue up to the TBO of 60 packets as soon as possible. Once the queue size is built up, the bandwidth is fully utilized. The ability the IntelRate controller stably controls the queue size to TBO of 60

packets indicates that the incoming traffic and the link bandwidth of 11Mbps have struck a balance. This is why the IntelRate controller shows much shorter time reaching the 100% utilization.

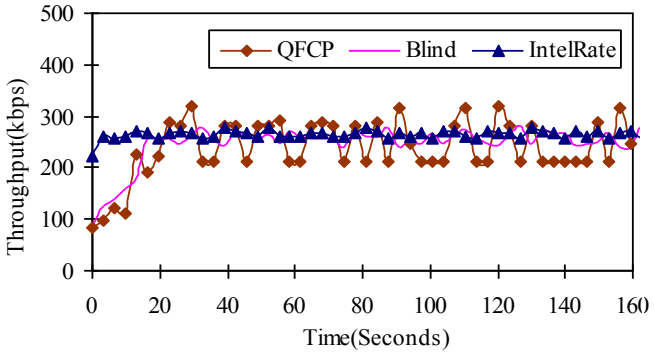


Fig. 5.5: Source Throughput Performance

The throughput of one of the sources in the three controllers is shown in Fig. 5.5. In the first 20s, both the QFCP and Blind gradually increase their sending rates in order to probe the available bandwidth. In their steady state (i.e., t>20s), QFCP has more oscillations than Blind due to two reasons. 1) QFCP calculates the sending rate based on the estimated link bandwidth. If the bandwidth is under-estimated from time to time (as shown in Fig. 5.4), the sending rate of the source will be decreased accordingly. 2). QFCP has unstable queue size to be illustrated next. The second reason can also explain the slight sinusoidal oscillations in the sending rate of Blind as perceived in Fig. 5.5. As with Fig. 5.4, the sending rate of the IntelRate controller shows much faster convergence speed and smoother steady state than QFCP and Blind.

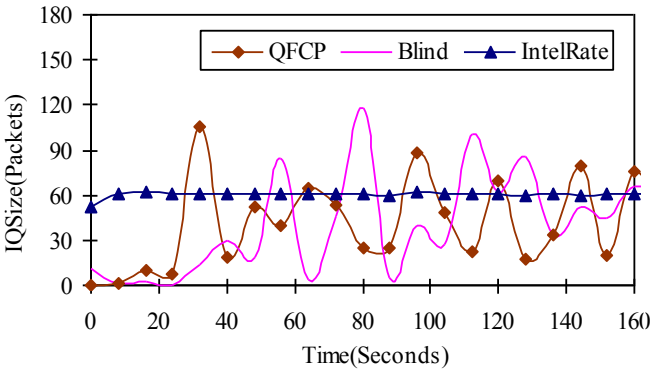


Fig. 5.6: IQSize Performance

Fig. 5.6 depicts the instantaneous queue size of the three controllers. Similar to the transients in Fig. 5.4 and Fig. 5.5, both QFCP and Blind take about 20s to build up their

queues due to their slow bandwidth probing process. In the steady state, their queue sizes are unstable and oscillating wildly all the time. This in turn results in an oscillating RTT which affects the smoothness of their source sending rates (as shown in Fig. 5.5). Unlike QFCP and Blind, the IntelRate controller has a closed-loop system dedicated to controlling the queue size (as seen in Fig. 3.1). Therefore, the IntelRate controller presents a much more stable queue size after it quickly reaches the TBO of 60 packets in Fig. 5.6.

5.3.2.2 Performance under Variable Wireless Bandwidth

In this experiment, we compare the IntelRate controller with the recent QFCP and Blind upon bandwidth dynamics. To simulate major bandwidth variations (i.e., the link adaptation), we switch the nominal bandwidths of the IEEE 802.11b WLAN as listed in Table 5.1.

Table 5.1: Link Adaptation

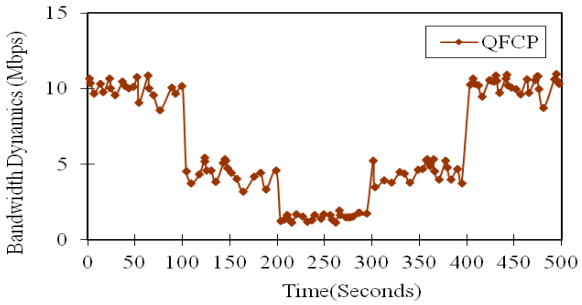
Time	Nominal Bandwidth
0~100s	11Mbps
100s-200s	5.5Mbps
200s-300s	2Mbps
300s-400s	5.5Mbps
400s-500s	11Mbps

To simulate the minor bandwidth variations due to contention of the MAC layer, we use a timer in the OPNET code block to generate random numbers at random intervals. The random number is used to generate the minor bandwidth change according to Raleigh distribution[7]. To make the bandwidth fluctuate stochastically, the random intervals generated by the timer are uniformly distributed within [0.5s, 15.0 s]. However, to be seen in Section 5.3.2.3, the IntelRate controller cannot respond well if the bandwidth randomly varies too fast.

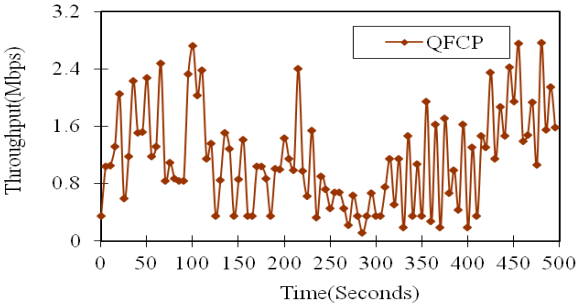
Finally, the bandwidth available to the transport layer is obtained by subtracting the minor bandwidth variations from the nominal bandwidth (which also varies due to link adaptation as a major bandwidth variation). For example, between 100s-200s in Table 5.1, the nominal bandwidth is 5.5Mbps. If the contention at t=125s takes 0.3Mbps bandwidth away, the actual bandwidth is just 5.5-0.3=5.2Mbps.

[7]A Rayleigh distribution has been widely applied in communication theory. It was first considered by Lord Rayleigh in 1880 as the distribution of the oscillated amplitude of wave light [PaPi02].

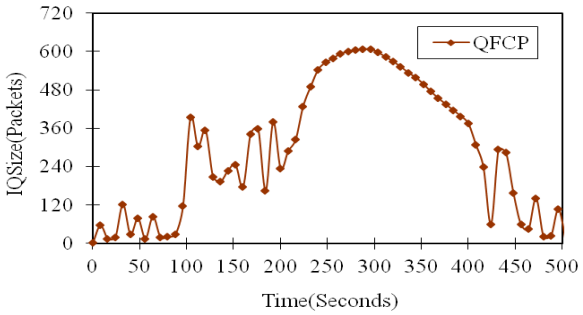
Below we shall separately test QFCP, Blind and the IntelRate controller under dynamic bandwidth variations, and compare their performances. Via the performance analysis for each controller, we show how the IntelRate controller outperforms these two schemes.



(a) Bandwidth Variation



(b) Source Throughput



(c) IQSize

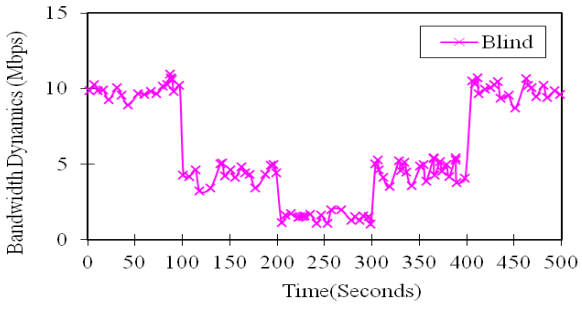
Fig. 5.7: QFCP

Fig. 5.7a depicts the dynamic bandwidth generated with the above-mentioned method to test QFCP. One can see that the wireless bandwidth varies among the nominal data rates with oscillations added to each nominal value. For example, in the first 100s, the wireless network operates on the nominal bandwidth of 11Mbps (see Table 5.1). The stochastic contentions in the MAC layer takes some bandwidth away from 11Mbps, and thus making the bandwidth oscillated and less than 11Mbps, as shown in [0,100s] in Fig. 5.7a.

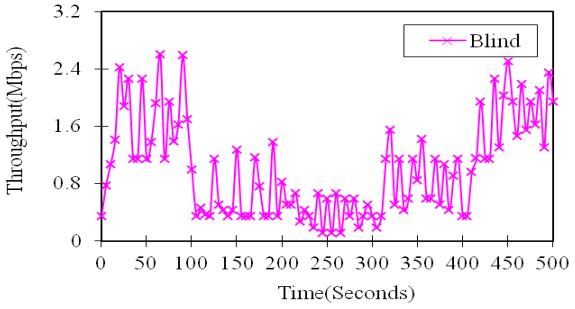
Fig. 5.7b shows that the source sending rate of QFCP is oscillating all the time under the dynamic bandwidth depicted in Fig. 5.7a. In Fig. 5.7b, the severe data rate oscillations show that the source sending rate just very roughly follows the nominal bandwidth changes, needless to say following the minor changes superimposed on the nominal bandwidth. For instance, when the bandwidth decreases to around 5Mbps at t=100s, the source sending rate decreases to around 100kbps but still oscillates severely. At t=200s, when the bandwidth

decreases to 2Mbps, the sending rate responds not fast, instead it spends more than 30s decreasing to a lower level while oscillating. The similar results are also for the time interval [300s, 400s] and [400s, 500s]. A general conclusion from Fig. 5.7b is that QFCP cannot well adjust its source sending rate to follow the bandwidth dynamics.

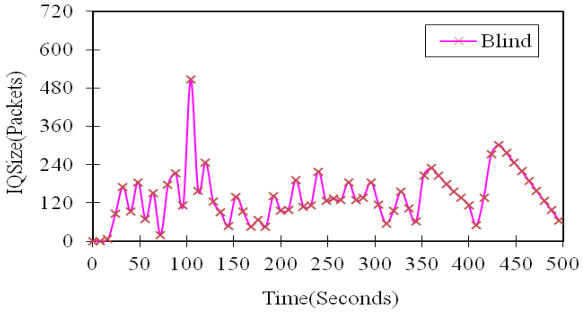
Fig. 5.7c depicts the IQSize of the QFCP under dynamic bandwidth. In the first 100s, the queue size is operating below 130 packets. When the bandwidth decreases to oscillate around 5Mbps at t=100s, the queue size sharply rises to 420 packets and then fluctuates around 200-400 packets until t=200s. The monotonic increase of the queue size occurs after the second time bandwidth decrease at t=200s. It can be seen that the queue size monotonically increases to the buffer capacity, i.e., 600 packets, which afterwards brings about packet losses. At t=300s, when the bandwidth increases to around 5Mbps, a monotonic decrease of the queue size from the buffer capacity can be observed because of the largely increased link capacity. Such a monotonic decrease lasts until t=400s when the bandwidth further increases to around 10Mbps, where the queue size presents some oscillations.



(a) Bandwidth Variation



(b) Source Throughput



(c) IQSize

Fig. 5.8: Blind

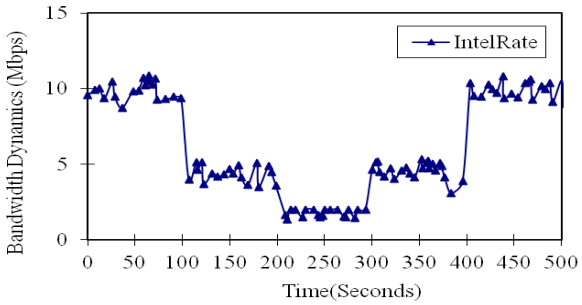
Fig. 5.8a is the bandwidth dynamics to test Blind. Fig. 5.8b shows the source sending rate of Blind under Fig. 5.8a. One sees that the source generally follows the nominal bandwidth variations well in the sense that the source throughput is oscillating on a higher/lower rate when the bandwidth is higher/lower. Such a performance of the source sending rate is better than that of QFCP in Fig. 5.7b because of the less chaos of the sending rate during the nominal bandwidth changes. For example, when the nominal bandwidth decreases to be around 5Mbps at t=100s, the source decreases its sending rate as well and oscillates at a lower rate around 60kbps. The same observations are also applicable to other nominal bandwidth changes, e.g., at t=200s when the nominal bandwidth decreases to 2Mbps from 5Mbps. That is to say, Blind can generally well follow the nominal bandwidth changes, but not the minor bandwidth changes superimposed on each nominal bandwidth because the oscillations are still not repressed well in Blind.

Fig. 5.8c demonstrates the queue size of Blind. Like QFCP in Fig. 5.7c, the queue size still shows oscillations, but is better than QFCP in the sense that it does not incur packet loss due to buffer overflow which happens when the queue size reaches the buffer capacity, i.e., 600 packets. Specifically, the queue size fluctuates around 120 packets after t=20s. When the bandwidth decreases to around 5Mbps at t=100s, the queue size sharply rises to 500 packets and then decreases back to a similar level to before with some oscillations. After the second bandwidth decrease at t=200s, the queue size oscillates at around 180 packets. At t=300s when the bandwidth increases to around 5Mbps, the queue size oscillates but less frequently in that just two peak values are present. This is probably due to the queue size tending to accumulate after more data comes in following the bandwidth increase, but the increased bandwidth vacates the accumulation. The similar trend of the queue size between t=400s and t=500s can be observed, too, in that one big peak value is present after the bandwidth increase to around 10Mbps.

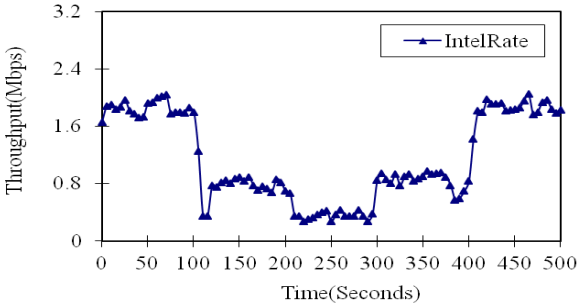
Fig. 5.9a is the bandwidth changes to test the IntelRate controller. The source sending rate of the IntelRate controller is plotted in Fig. 5.9b. Compared with QFCP and Blind, Fig. 5.9b essentially shows clearly that the IntelRate controller suppresses the oscillations and follows the bandwidth variations better.

It is also interesting to observe from Fig. 5.9b that the source can behave well when the nominal bandwidth jumps from one level to another, i.e., major bandwidth variations. For

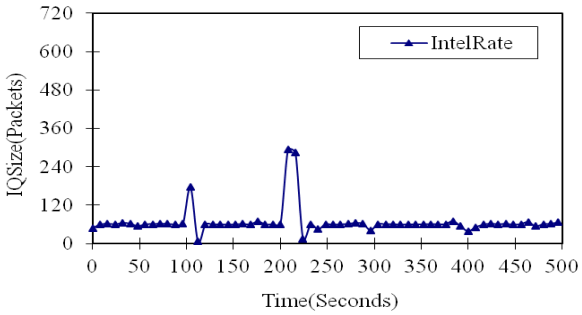
example, when the nominal bandwidth decreases to around 5Mbps at t =100s from about 10Mbps, the source sending rate decreases as well from around 230kbps with a big drop at the beginning and then settles at around 100kbps. The second decrease shows the same



(a) Bandwidth Variation



(b) Source Throughput



(c) IQSize

Fig. 5.9: IntelRate

trend, and the difference is that the sending rate decreases to around 55kbps from about 100kbps. When the bandwidth increases at t=300s from 2Mbps to 5Mbps, the source sending rate increases as well and again shows much smoother performance. The same observation is also for the second increase at t=400s.

On the other hand, the source sending rate of the IntelRate controller in Fig. 5.9b can respond to minor bandwidth variations (with respect to the nominal bandwidth) most of the time. For example, although not matched nicely within the interval [0,100s], one can inspect to see the minor increase or decrease of the date rate in Fig. 5.9b do follow more or less the slight increase or decrease of the bandwidth in Fig. 5.9a. The same observations can be made in the interval [100s, 200s] except for couple places (i.e., around t=120s) where the source sending rate does not show oscillations while the bandwidth does. The reason could be that the queue has not completely finished its adjustment after the major bandwidth change at t=100s (to be seen in Fig. 5.9c). The similar observations again can be made for

the intervals [300s, 400s] and [400s, 500s].

As a summary to Fig. 5.9b, the source sending rate of the IntelRate controller can well respond to the trend of bandwidth dynamics. By much suppressing the oscillations, it has greatly outperformed QFCP and Blind discussed in Fig. 5.7b and Fig. 5.8b.

Fig. 5.9c depicts the queue size of the IntelRate controller. The queue size is well controlled around the TBO=60 packets except when the link adaptation occurs. The minor bandwidth variations within any time interval, e.g., [100s, 200s], does not affect the smoothness of the queue size of the IntelRate controller. Specifically, after the router starts, the queue size quickly reaches the TBO and stabilizes there until t=100s. At the first bandwidth decrease at t=100s, the queue size undergoes a big oscillation in which the queue size reaches around 200 packets, but settles down after just one cycle of oscillation. The same oscillation can also be seen at t=200s where the second bandwidth decrease happens. The difference this time is the glitch has a peak of about 320 packets which is still far from the buffer capacity of 600 packets. When the bandwidth increases at t=300s or t=400s, the queue size just slightly drops and then settles back at TBO=60 packets. In conclusion, the queue size performance of the IntelRate controller has greatly overcome the fluctuating behavior of QFCP and Blind, and hence improves the system stability. Such a success is attributed to the closed-loop queue size control of the IntelRate controller.

In a nutshell, the IntelRate controller shows better performance than QFCP and Blind in terms of the source throughput and queueing stability. As shown in Fig. 5.9, the major bandwidth changes are less frequent with big amplitude among different nominal bandwidths, and it is not a challenge to the IntelRate controller. In contrast, the minor bandwidth changes are more frequent, and the amplitude variation is small, and thus it may pose challenges on the response capability of the IntelRate controller. A designer should pay attention to the response limitations that a congestion controller may encounter under fast bandwidth variations. As shown in this section, the IntelRate controller shows better responses upon both major bandwidth variations and minor bandwidth variations.

5.3.3 Response Limit to WLAN Bandwidth Changes

In Section 5.3.2.2, the IntelRate controller shows that it can respond to follow the bandwidth changes for both major and minor bandwidth changes, compared with other controllers.

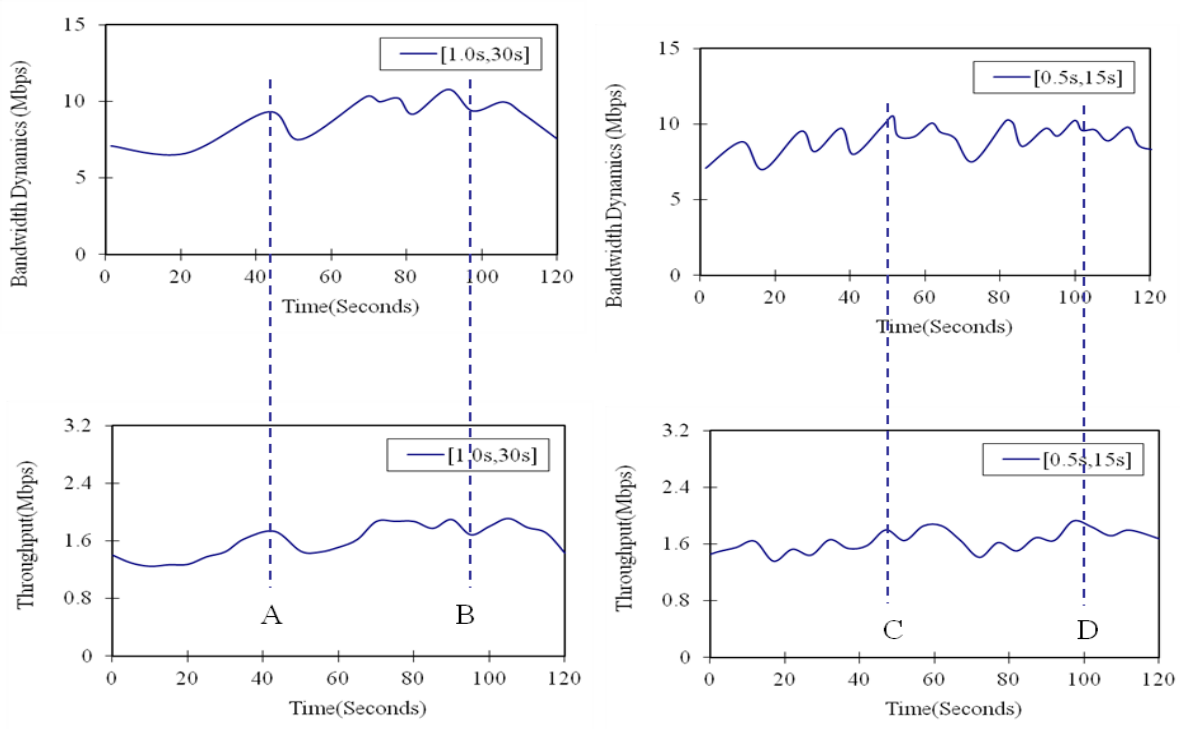
However, does the IntelRate controller can respond to any fast bandwidth changes? As shown in Section 5.3.2.2, the major bandwidth changes are less frequent with big amplitude among different nominal bandwidths, and it is not a challenge to the IntelRate controller. In contrast, the minor bandwidth changes are more frequent and unpredictable, and the amplitude variation is small, and thus it may pose challenges on the response capability of the IntelRate controller. In this regard, we would like to investigate the response limit of the IntelRate controller to the frequent minor bandwidth changes as done in the following tests. The timer (which controls the frequency of the minor bandwidth changes as discussed in Section 5.3.2.2) for minor bandwidth changes will be separately set in [1.0s, 30s], [0.5s, 15s] and [0.05s, 1.5s] so that we can observe the response ability of the IntelRate controller under the different bandwidth variation frequencies. The nominal bandwidth is maintained the same, i.e., 11Mbps.

Fig. 5.10 demonstrates the IntelRate controller responses under different bandwdith variation frequencies. In each figure, the upper diagram is the bandwidth variation, and the lower diagram is a source response which shows how the source sending rate behaves as the bandwidth is changing.

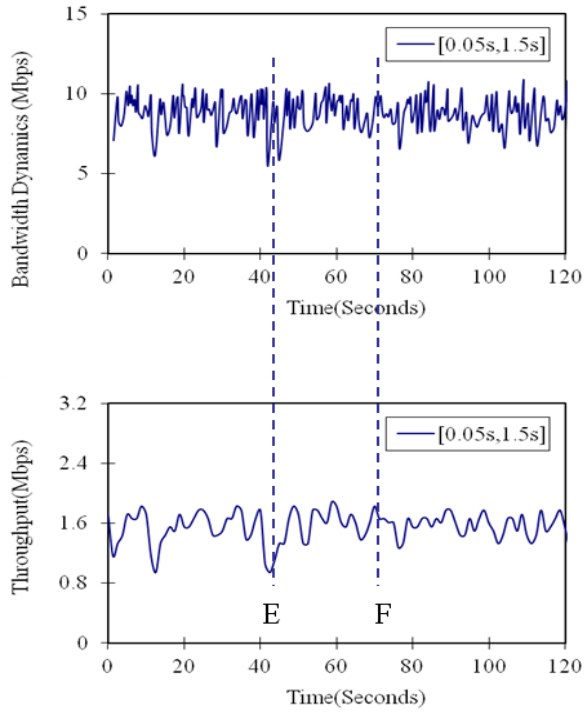
Fig. 5.10a is the source response under the bandwidth random variation periods [1.0s, 30s]. It shows that basically the source follows the bandwidth dynamics very well. For examples, at the dashed line A, the source increases its sending rate as a response to the bandwidth increase observed in the upper diagram of Fig. 5.10a. At the dashed line B, the sources decreases its sending rate as a response to the banddiwdth decrease. The same observations are also applicable to other time points that we do not designate, i.e., the source can respond to any bandwidth changes in Fig. 5.10a.

Fig. 5.10b shows the source response under higher bandwidth variation frequencies which has random variation periods [0.5s, 15s]. One can see the source is basically able to track the bandwidth variations, e.g., at the dashed line C, the source increases its throughput as the bandwidth becomes larger as shown in the upper diagram in Fig. 5.10b. However, one may also notice that the source behavior does not reflect the bandwidth variations everywhere. For instance, at the dashed line D, the bandwidth shows a small decrease, but the expected small decrease does not happen in the source sending rate shown in the lower diagram in Fig. 5.10b. This implies that the IntelRate controller has reached its response

limit with a limit variation period of about 0.5s which is the smallest value in [0.5s, 15s].



(a) Variation under [1.0s, 30s] **(b) Variation under [0.5s, 15s]**



(c) Variation under [1.0s, 30s]

Fig. 5.10: Response under Different Bandwdith Variation

How does the source respond if the bandwidth variation frequency is continuted to increase? Fig. 5.10c shows such a scenario under a much higher variation frequency with the timer set to [0.05s, 1.5s]. Essentially, Fig. 5.10c shows that the source cannot keep the variation pace with the bandwidth dynamics any more in such high variation frequencies in that the variation frequencies of the source are apparently lower than those of the bandwidth, instead it can only track the envolope of bandwidth changes. For examples, around the dashed line E, the bandwidth shows lower values with oscillates, but the source just has a big drop without any oscillation. Around the dashed line F, the bandwidth increases with oscillations, but the source does not oscillate, instead it just shows an increase, as seen in the lower diagram of Fig. 5.10c. In closing, Fig. 5.10c shows that the source can only track the envolope of the bandwidth changes, but cannot respond to the specific changes which are too frequent.

As a summary to Fig. 5.10, even though the IntelRate controller overcomes the bandwidth variation issues of other controllers, it has response limit to the bandwidth variations. If the bandwidth changes too fast (e.g., the scenario Fig. 5.10c), the IntelRate controller cannot track them, instead it can only follow the envolope of the bandwidth variations in that case.

5.4 The OFEX Controller Performance

In this section, we show the optimal rate allocation performance of the OFEX controller when it is applied in WLAN with $\lambda_l^{(0)} = 50$, $\delta = 0.015$ and γ=5. The source payment of s_i (i=1,2,…,5) are ψ_s=[1100, 900, 725, 1240, 570].

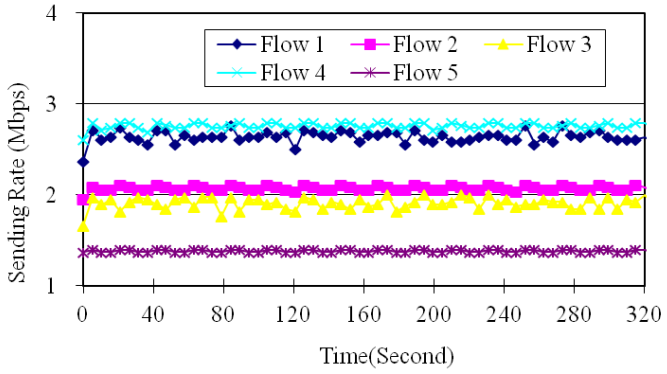


Fig. 5.11: Source Sending Rate

5.4.1 Optimal Rate Allocation in WLAN

Through this experiment, we will see how the wireless bandwidth of 11Mpbs in AP is proportionally shared among the different 5 video flows with the OFEX controller.

Fig. 5.11 shows the portion of wireless bandwidth shared among the 5 video flows. As seen, they separately obtain a sending rate of about 2.65 Mbps, 2.08 Mbps, 1.92 Mbps, 2.76 Mbps, 1.38Mbps, which corresponds the price in ψ_s above. The more they pay, the more portion of bandwidth they achieve. For example, the flow 2 pays 900 which is a price higher than what paid by the flow 3 and 5 while lower than what paid by the flow 1 and 4. Accordingly, the flow 2 obtains a portion of 2.08Mpbs bandwidth which is between those two higher ones and two lower ones. Therefore, the proportional fairness of the OFEX controller is verified in its application to the wireless network.

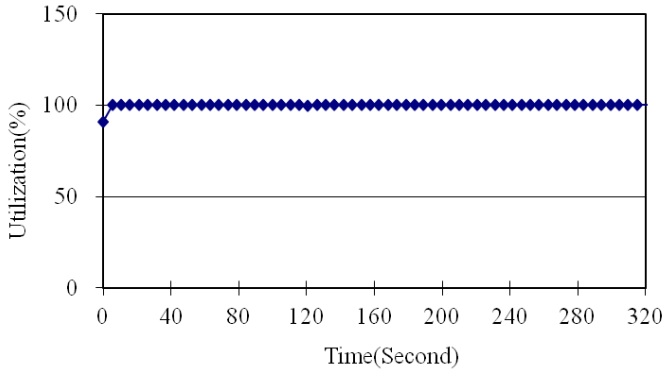


Fig. 5.12: Wireless Bandwidth Utilization

Fig. 5.12 shows that except the first 5 seconds (in which the controller is in the starting stage) the wireless bandwidth maintains 100% utilization with the optimal rate allocation, which shows good performance of the OFEX controller in fully utilizing the wireless bandwidth.

5.5 Concluding Remarks

By setting up a WLAN network in OPNET modeler, we have applied the IntelRate controller and the OFEX controller to a wireless local area network, and the experimental results show that both controllers can work well in the WLAN. In particular, via the application, we have investigated the performances of the IntelRate controller under different scenarios.

Chapter 6
Design Guideline and Deployment

This chapter will investigate some pertinent design parameters and issues of the IntelRate controller and the OFEX controller in more details. Our experience will be discussed in the sequel, especially the limits in the design parameters. Finally, a TCP-friendly implementation solution is proposed to address the concerns of the network industry in the deployment of these two controllers.

6.1 The IntelRate Controller

The good performances of the IntelRate Controller have been demonstrated in Chapter 3. The design parameters of the IntelRate controller have been experimentally investigated, as seen in Section 3.3 and Appendix F. Below our experiences of selecting those design parameters will be first discussed, and then one issue of the IntelRate controller that needs attention will be presented.

6.1.1 Value Selection of Different Design Parameters

This section suggests some guideline to choose the design parameter values, most of which need to consider their tradeoffs.

6.1.1.1 TBO q_0

From the perspective of the queueing delay, the TBO value should be as small as possible. This is especially true under the heavy traffic conditions when the queue is to be stabilized at TBO. Therefore, a bigger TBO will result in longer steady state queueing delay, which is not desirable to some Internet applications such as the real-time video.

On the other hand, from the design of a FLC-based controller, too small TBOs are not preferred because it may cause the IntelRate controller to restrict its throughput severely or to under-utilize the bandwidth. For easy illustration, we choose the extreme case of q_0=0 which requires the controller to vacate the queue under whatever traffic conditions. In such a case, the LVs of $g(t)$ can only make transitions among ZR, NS, NM, NL and NV because the queue size cannot be greater than zero, i.e., $q(t)$=0. Accordingly the controller output (i.e.,

the allowed source sending rate) can at most reach BG, and never goes to the MX (as seen in the dashed part of Table 3.1). Therefore, the source throughput becomes quite restricted. Furthermore, too small TBOs may cause the link to become under-utilized momentarily after some flows finish their transmissions. This is because the remaining packets in a small queue can be easily emptied before the sending rates of other flows are increased to fully utilize the available bandwidth. The above extreme case analysis reveals that q_0=0 would not cause the instability or failure of the controller. Rather, it would just cause the controller to operate improperly, i.e., restricted source throughput and/or under-utilized link bandwidth.

In short, the TBO should be chosen such that the network can have a reasonable queueing delay while maintaining good throughput and link utilization. For the IntelRate controller, we would choose a queue size (and therefore the worst node queueing delay, e.g. less than 10 ms) in order to give a network delay acceptable to most of the real-time traffic while maintaining good performance.

Please note that once q_0 is determined in the design stage, it remains fixed (a constant) during the operation of the controller and is independent of whatever the traffic levels are.

6.1.1.2 The number *N* of LVs

The choice of *N* has to consider the trade-off between the throughput performance and the computation complexity of the controller. We observe that a big *N* complicates the controller in the sense that it has to do more logic computations on choosing the allowed sending rate according to the rules. Such a computation complexity can affect the rise time in the transient response of the controller as well as the control performance in the case of network parameter changes (e.g. the settling time during large bandwidth variation). On the other hand, a small *N* may lead the controller output to oscillate due to the big partitions of the LVs.

6.1.1.3 The Output Edge Value *D*

With reference to Fig. 3.3, the outermost edge value *D* in the output MFs corresponds to the maximum sending rate that the controller can output. This parameter is chosen to be the maximum value of the *Req_rate* field among the active incoming flows, i.e. *D*= $\max(\hat{u}_1, \hat{u}_2, \ldots, \hat{u}_R)$, where $\hat{u}_i$, *i=1,2, ...,R*, is the value recorded in *Req_rate* of each packet.

To determine the parameter D, we define an operation cycle of d within which the incoming packets are collected to reflect the dynamics of the network traffic. One would like to have a small d to guarantee the D value updated often enough to reflect the most current biggest *Req_rate* rate of all the passing flows. Note that too small of d would mean the packets of some flows may not be captured within one cycle. This is all right because the controller need not impose a new rate to those flows in that cycle, and those flows just operate with their current sending rates until their packets are captured. A general practice is to choose a d value smaller than the network RTT (which is generally less than 500ms [Floy12, JiDo02]).

6.1.1.4 The Width Limit *m*

The parameter m defines the base width of each membership function in the FS. Since it also affects the extent of overlapping between the adjacent MFs, the basic consideration to choose an appropriate parameter m is to have a smaller TBO while remaining the controller output smooth. An inappropriate m may have similar side effects like the parameter N. Too big m value leads to small partitions along $e(t)$ or $g(t)$, and thus may affect the response time of the controller. On the other hand, too small m may cause fluctuations in the controller output due to the too big partitions along $e(t)$ and $g(t)$. As seen in Fig. 3.3, it also decides the upper limit of $g(t)$ when there is no congestion in the routers.

6.1.1.5 The Buffer Size *B*

The determination of buffer size B is closely related to the chosen value of TBO. Although one can choose $B=q_0$, the smallest possible value, this is usually not desirable for the following two basic reasons: 1) a controller usually has various steady state error issues [DoBi08], and it is impossible that the queue size can be exactly pegged at TBO; 2) the dynamic Internet traffic can sometimes cause a surge on the queue size, e.g., due to a sudden traffic swarm-in or an unexpected bandwidth reduction. Therefore, the B should be greater than the TBO.

Previous controllers such as the XCP, RCP and API-RCP have designed their buffer size according to the BDP, which means they require big buffers in the high BDP networks. For example, the buffer size would be 183100 packets (assume the packet has a size of 1024

bytes) for a link bandwidth of 5Gbps with an average RTT of 0.3s in the network. Since the IntelRate controller aims to control the queue size operating around the TBO, our experiments show that B=10*q_0 is a good choice which is much smaller than the value determined by BDP while not causing buffer overflow. For example, with the same 5Gbps network, when we set q_0=6000packets (which means a queueing delay of 9.83ms upon heavy traffic at the steady state), the B is 60000 packets. As seen, it is less than1/3 of the above controllers.

6.1.2 Queueing Delay Issue

Queueing delay is part of the end to end delay performance that everybody likes to minimize or even eliminate; it is associated with the queue size performance. Although the IntelRate controller can operate the queue size around a target, i.e., TBO, the queueing delay cannot be eliminated. Setting it too low may bring adversary effects of restricted throughput and bandwidth under-utilization, as discussed in Section 6.1.1.1. The advantages using the IQSize are the measurement simplicity and accuracy as well as maintaining queueing delays steady. From our analysis and experiments, the compromise is to set it causing queueing delay less than 10ms so that the total queueing delays accumulated in multiple routers would not be significant.

Actually, the original TBO design of the IntelRate controller was even worse (i.e., 30% of the buffer capacity B. For brevity, we call it "30% design"), and the related results have been presented in [LiYa10]. The "30% design" idea was originally borrowed from API-RCP [HoYa07]. Due to time limit to publish [LiYa10], we used the same TBO setting for the IntelRate controller. "30% design" means the IntelRate controller may maintain a big queue size, which resulted in long queueing delay in routers. For example, for a router which has a buffer capacity B=120000 packets and bandwidth 5Gbps, the TBO will be 40000 packets and the queueing delay can be 81.92ms (assuming the packet size is 1024 bytes). If a real-time video needs to pass through 10 bottlenecked routers to reach the receivers, the total queueing delay en route will be 819.2ms. Plus the propagation delay, the data transmission time from the source to the audiences can be more than 1.0s.

6.2 The OFEX Controller

The OFEX controller also has couple design parameters that can affect the controller performance. In addition to the discussion of how those parameters should be chosen, we also presented one issue we have discovered with the OFEX controller.

6.2.1 Value Selection of Different Design Parameters

This section discusses the tradeoffs and considerations for the choice of various design parameter values of the OFEX controller.

6.2.1.1 The Step Size δ

The bigger the step size, the faster the convergence of the controller because theoretically it makes $[2\tilde{m}\delta(\frac{L_c}{2}\delta-1)+1]^{k+1}((\Omega(\lambda_l^{(0)})-\Omega^*)$ in (4-27) go to zero faster. However, caution must be exercised to choose too large of a step size which can give rise to divergence or oscillation [P31, Bert99]. This observation has been verified by the experimental results demonstrated in Section J.3 of Appendix J. For example, Fig. J.5 shows the source throughput begins to render some fluctuations when the step size increases to 0.5.

From most of the experiments such as those conducted in Section 4.6, we found that a step size δ=0.01 is feasible regardless of the link bandwidth in that both the convergence speed and the stability of the controller are acceptable. Actually for a specific router, once its nominal bandwidth is known, some trial-and-error experiments like those conducted in Section J.3 can be used to help choose an appropriate step size δ.

6.2.1.2 The Initial Lagrange Multiplier $\lambda_l^{(0)}$

If the initial value of $\lambda_l^{(0)}$ is so chosen such that $\Omega(\lambda_l^{(0)})$ is closer to Ω^* in (4-27), the convergence time is going to be short. This observation has been verified by the experimental results demonstrated in Section L.2 of Appendix J. The only difficulty is in a practical implementation, one usually does not know what Ω^* is in advance. Fortunately, routers usually run forever once started, and do not need frequent re-starting and initialization. So the convergence time due to the choice of $\lambda_l^{(0)}$ at the starting stage should not be a concern. This indicates that one may choose $\lambda_l^{(0)}$ as they want.

6.2.1.3 The Parameter γ

As demonstrated in Section 4.6.3, the parameter γ stipulates the response speed to link dynamics as well as the queue size level of $q(t)$. The bigger the constant γ, the faster the response speed as well as the lower $q(t)$. However, a bigger γ may lower the link utilization. As summarized in Section 4.6.3, one would try not to choose too large of the γ value to cause the system performance to drop below an acceptable level of link utilization and neither to choose too small of the γ value to make the response time worse for a router.

6.2.2 Computational Complexity Issue

An algorithm with low computational complexity is usually preferred to meet the real time requirement of congestion control. From the computational analysis conducted in Appendix I, one sees that the OFEX controller requires 20 clock cycles more for every packet, compared with the relatively explicit controller. Such an extra computational resource requirement could be significant for high-speed routers when dealing with thousands of packets per second. For example, for a 5Gbps router, it can process about 600 thousand packets per second for a packet size of 1024 bytes. As such, every second the OFEX controller consumes 12 million clock cycles more. Unless we can find a simpler implementation than that discussed in Appendix I, the OFEX controller requires a faster CPU to compete with the relatively explicit controllers. Finally, since the OFEX controller needs to determine the packet size upon packet arrival, this may take time and incur a bit delay which can be regarded as part of the packet processing time.

6.3 TCP-Friendly Deployment

The deployment of any new congestion control scheme usually has to consider its backward compatibility to TCP because of the absolute dominance of TCP in networks nowadays.

In view of the successful deployment of RED and ECN in routers nowadays, the deployment of the IntelRate controller and the OFEX controller is hopeful in the sense that they operate on the same objects (e.g., queue and where the controllers are located at) of a router. Therefore, the architecture (or hardware) of the router needs no modification in order to deploy our controllers. In this section, we conjecture how our controllers should share the networks with TCP.

6.3.1 The IntelRate Controller Deployment

To get the IntelRate controller deployed, three basic issues need to be addressed. One is queue sharing strategy with TCP because the queue size is the only parameter that the IntelRate controller uses to control the congestion. The second one is where the congestion header (i.e., the extra field "*req_rate*") should be placed in a packet used to carry the congestion signal. Another one is how to split tasks between the IntelRate controller and TCP.

6.3.1.1 Separate Queues for TCP and the IntelRate Controller

To co-exist with TCP, two separate queues are required in a router. One is for TCP, and the other one is for the IntelRate controller. The queue to be controlled here is the one located in the egress (or say, output) interface of a practical router. TCP and the IntelRate controller cannot share the same queue because: if TCP and the IntelRate controller share the same queue, the IntelRate controller will be treated unfairly and must be starved from sending data. To clarify this, we below analyze how these two protocols operate the queue.

TCP always tends to top up a buffer and overflow it so that the congestion signal, i.e., packet loss, can be generated. However, the IntelRate controller tries to control the queue size to the TBO of a buffer (note TBO is far less than the buffer capacity *B*). When TCP and the IntelRate controller share the same queue, whenever TCP pulls up the queue size to high, i.e., beyond the TBO, it triggers the IntelRate controller to decrease the source sending rate. Such restless pull-ups from TCP severely disturb the correct operation of the IntelRate controller which has to decrease its source sending rate all the time. Therefore, if the IntelRate controller would like to get fair bandwidth for sending data, it has to have a separate queue for its closed-loop queue control.

6.3.1.2 Deployment in IPv4 or IPv6 for the Congestion Header

Since the current TCP packet header may not have enough unsued space to accommodate the *Req_rate* field to support the IntelRate controller, two approaches have been [LiWa09] for the deployment in IPv4. One is override the "advertised window size" field in the TCP packet header. The other approach is to introduce an adaptation sublayer can be introduced between

the TCP and the IP layer.

Deployment in IPv6 is easier in comparison because there are quite a few reserved extension headers in IPv6 [KrWo12] to carry the "*req_rate*" field.

6.3.1.3 Task Split with TCP

Usually a router has a few ingress and egress interfaces (or say, ports), some of which can be used by TCP and the rest by the IntelRate controller. From the perspective of a gradual deployment, at the beginning, the IntelRate controller may just use one ingress interface and one egress interface, in which the traffic from the ingress interface goes into the queue first and then is sent to the egress interface after processed by router. The IntelRate controller is implemented in the router to monitor the queue size and takes actions to manage congestions in each egress interface of the router. If the control service from the IntelRate controller proves to be more superior practically, more ingress and egress interfaces may be handed over to the IntelRate controller. This is can be done when deploying new routers in the networks.

The above deployment conjecture assumes the IntelRate controller can be integrated into routers, which needs much encourage for the industries because cautiousness is always exercised when new technologies are to be adopted. As an alternative, the IntelRate controller in the beginning may not be integrated into routers, instead the IntelRate controller can be put into an external auxiliary device which can communicate with a router. By this means, the IntelRate controller collects the queue information from the router and in return sends the control signals back to the router after completing the calculation. However, it may have real-time issue if the communication speed between the router and the external device is not fast enough.

6.3.2 The OFEX Controller Deployment

The OFEX controller can also co-exist with TCP networks as proposed for the IntelRate controller**.** This is because 1) the OFEX controller deployment also has to consider the backward compatibility with the dominant TCP in networks nowadays; 2) the OFEX controller also involves queue, congestion header and task split with TCP. Therefore, the OFEX would share the same solution as proposed for the IntelRate controller above from

Section 6.3.1.1-Section 6.3.1.3. in which the details are omitted here for brevity reason.

One minor difference from the IntelRate controller lies in Section 6.3.1.1, i.e., the reason that the OFEX controller also needs a separate queue is because the OFEX controller uses the changes of the queue size to signal traffic changes and/or bandwidth changes. The restless pulling up of the queue size by TCP would thus affect the correct operation of the OFEX controller.

6.4 Concluding Remarks

In this section, we have shared our experiences on parameters designs of the IntelRate controller and the OFEX controller, and pointed out the tradeoffs that should be considered for the choice of parameters values. We also discussed some issues that would affect the performances of the controllers. Finally, our controllers can co-exist with TCP which should facilitate their deployment.

Chapter 7
Conclusion

We have put forward two explicit congestion controllers based on modern control approaches in order to address the issues of existing congestion control schemes in handling the massive amount of data transmitted such as file transfer and video streaming in the network nowadays.

We have designed an intelligent explicit rate controller using FLC. Called the IntelRate controller, it addresses the system instability issue due to parameter mis-estimations, and greatly reduces the computational complexity of an explicit congestion controller. The IntelRate controller does not need to estimate the link bandwidth, packet loss, latency or the number of incoming flows while providing the explicit congestion signals which effectively throttle the source sending rate with max-min fairness. In fact, it can avoid network congestion by only relying on IQSize. Our congestion control system embedding the IntelRate controller has been proved and verified to be globally asymptotically stable in whatever traffic conditions.

Based on the NUM, we have classified the explicit congestion control protocols into relatively explicit protocols and fully explicit protocols. This allows us to propose an optimal and fully explicit scheme, called the OFEX controller. The OFEX controller uses the link-wise proportional fairness to achieve the network-wise max-min fairness, and has addressed the shortcomings of the relatively explicit controllers which bias the multi-bottlenecked flows and cannot respond to bandwidth dynamics. Furthermore, the OFEX controller reduces the queue size and thus improving queueing delay and packet loss performance. The properties of the OFEX controller such as the convergence, system stability, computational complexity and robustness have also been analyzed.

Extensive simulations have been employed to evaluate the effectiveness of both controllers. Their applications to a WLAN are demonstrated. Some design guidelines of the two controllers are also discussed.

As a closing statement, even though we have not directly made a comparison between these two controllers, experimental results separately presented in Chapter 3 and Chapter 4 indicate that they have similar performances in terms of the network parameters dynamics

such as bandwidth changes or the number of flows changes. One primary difference between them from the queueing perspective is that the IntelRate controller can control the queue size operating as desired, whereas the OFEX controller cannot because it does not have a dedicated closed-loop control for the queue, which is listed for our future work right after.

7.1 Future Work

The following are some potential and interesting aspects for extending our present work.

1) Implementation of our controllers on real networks: instead of software simulations performed here, testing the controllers in real networks would be more practical to improve their deployment.

2) The precondition of the IntelRate controller to adjust the source sending rate is that the source can receive the ACK packet after one RTT. However, the ACK packet can be lost or broken during the transmission. Therefore, some ways that can be embedded into our controllers need to be figured out in order to counter the impact of the ACK packet failure.

3) Feasibility research on applying genetic algorithm to the IntelRate controllers: instead of fixing the rule base, use the genetic algorithm to tune fuzzy control rules and see if any performance improvement can be obtained.

4) Feasibility research on applying type-2 FLC [MeJo02] which is a fuzzier and more complicated approach than the IntelRate controller: instead of type-1FLC [PaYu98] performed here, use type-2 FLC to deal with the uncertainties of the control system.

5) Fixing TBO in the OFEX controller: methods to make the queue size of the OFEX controller operate around a target position instead of leaving the queue size fluctuate randomly.

6) The theoretical stability proof of the congestion control in multi-bottlenecked network models or a general network topology for the IntelRate controller and the OFEX controller.

7) More detailed performance evaluation of the OFEX controller in WLAN to confirm its superiority over other controllers as well as the response limit to bandwidth variations.

As a closing comment, congestion control research has been around for more than two decades. The control approach has evolved from implicit to explicit. Quite a few of the explicit schemes (as those reviewed in this thesis) for TCP/IP networks have been proposed in the past ten years. However, due to the commercializing risks, their deployment in routers

and end users is hardly seen. Therefore, some systematic research work on minimizing the commercialization risk of the explicit congestion controllers is needed.

References

[AbRi06] F. Abrantes and M. Ricardo, "XCP for shared access multirate media", *ACM SIGCOMM Computer Communication Review*, Vol.36, No.11, pp. 29-38, 2006.

[AbAr11] F. Abrantes, J. Araujo and M. Ricardo, "Explicit congestion control algorithms for time varying capapcity media", *IEEE Transactions on Mobile Computing*, Vol.10, No.1, pp.81-93, Jan. 2011.

[AlBa02] T. Alpcan and T. Basar, "A game-theoretic framework for congestion control in general topology networks", *Proceedings of the IEEE Confererence on Decision and Control* (*CDC*), pp.1218–1224, Dec. 2002.

[AoNa04] Y. H. Aoul, A. Nafaa, D. Negru and A. Mehaoua, "FAFC: fast adaptive fuzzy AQM controller for TCP/IP networks", *Proceedings of the IEEE Global Communications Conference* (*GLOBECOM*), Vol. 3, pp.1319 - 1323, Nov.29-Dec.03, 2004.

[ArKa58] K. J. Arrow, S. Karlin and H. E. Scarf, Studies in the Mathematical Theory of Inventory and Production, *Stanford University Press*, 1958.

[AtLo00] S. Auhuraliya and S. H. Low, "Optimization flow control with Newton-like algorithm", *Telecommuncation Systems*, Vol.15, No.3-4, pp.345-358, 2000.

[BeBa09] M. Belleschi, L. Balucanti, P. Soldati, M. Johansson, and A. Abrardo, "Fast power control for cross-layer optimal resource allocation in DS-CDMA wireless networks", *Proceeding of the IEEE International Conference on Communications,* (*ICC*)*,* pp.1 – 6, Jun. 14-18, 2009.

[BeMe93] L. Benmohamed and S. M. Meerkov, "Feedback control of congestion in packet switching networks: the case of a single congested node", *IEEE/ACM Transactions on Networking*, Vol.I. No.6, pp.693-708, Dec.1993.

[Bert99] D. P. Bertsekas, Nonlinear Programming, *Athena Scientific*, 1999.

[Bhat08] U. N. Bhat, An Introduction to Queueing Thoery: Modeling and Analysis in Applications, *Birkhauser Boston*, 2008

[BoDi03] M. Bouhtou, M. Diallo and L. Wynter, "Capacitated network revenue management through shadow pricing", *Proceedings of the Networked Group Communication,* Sep. 16-19, pp.342-351, 2003.

[BoVa04] S. Boyd and L.Vandenberghe, Convex Optimization, *Cambridge University Press*, 2004.

[BrMa94] L. S. Brakmo, S. W. O'Malley, and L. Peterson. "TCP Vegas: new techniques for congestion detection and avoidance," *Proceedings of the ACM SIGCOMM*, pp 24-35, 1994.

[BrPe95] L. S. Brakmo, L. L. Peterson, "TCP Vegas: end to end congestion avoidance on a global Internet", *IEEE Journal on Selected Areas in Communication*, Vol. 13, pp. 1465- 1490, 1995.

[ChCh94] C. Chang and R. Cheng, "Traffic control in an ATM network using fuzzy set theory", *Proceedings of the IEEE International Conference on Computer Communications* (*INFOCOM*), Vol. 3. pp.1200-1207, Jun. 12-16, 1994.

[Chen06] Q. Chen, "Active queue management methods in computer communication networks based on pole placement and H_∞ optimal control", *PhD dissertation, University of Ottawa*, 2006.

[Chia05] M. Chiang, "Balancing transport and physical layers in wireless multihop

networks: jointly optimal congestion control and power control", *IEEE Journal of Selected Areas in Communications*, Vol. 23, No.1, pp. 104-116, Jan. 2005.

[ChLe01] S. Chong, S. Lee and S. Kang, "A simple, scalable, and stable explicit rate allocation algorithm for max–min flow control with minimum rate guarantee", *IEEE/ACM Transactions on Networking*, Vol.9, No.3, pp.322-335, Jun. 2001.

[ChPi03] C. Chrysostomou, A. Pitsillides, G. Hadjipollas, A. Sekercioglu and M. Polycarpou, "Fuzzy explicit marking for congestion control in differentiated services networks", *Proceedings of the Eighth IEEE International Symposium on Computers and Communication,* Vol.1, pp.312 – 319, 2003.

[ChPi06] C. Chrysostomou and A. Pitsillides, "Fuzzy logic congestion control in TCP/IP tandem networks", *Proceedings of the 11th IEEE Symposium on Computer and Communications*, pp.123-129, 2006.

[ChPi09] C. Chrysostomou and A. Pitsillides, "Fuzzy logic control in communication networks", Chapter of Foundations of Computational Intelligence, Vol.2, pp.197-236, 2009.

[ChWa99] C. Chen, S. Wang, C. Hsieh and F. Chang, "Theoretical analysis of a fuzzy-logic controller with unequally spaced triangular membership functions", *Fuzzy Sets and Systems*, Vol.101, pp.87-108.

[ChYa05] Q. Chen and O. W.W. Yang, "Design of AQM controller for IP routers based on H/S/U MSP", *Proceedings of the IEEE International Conference on Communications,* (*ICC*), Vol.1, pp.340 - 344, 2005.

[CrBe97] M. E. Crovella and A. Bestavros, "Self-similarity in world wide web traffic: evidence and possible causes", *IEEE/ACM Transactions on Networking*, Vol.5, No.6, pp.835-846, Dec. 1997.

[DaHa04] J. G. Dai, J. J. Hasenbein and J. H. Vande Vate, "Stability and instability of a two-station queueing network", Annals of Applied Probability, Vol. 14, No., 1, pp. 326-377, 2004.

[DaMc05] W. Damon, and N. McKeown. "Part I: Buffer sizes for core routers", *ACM Computer Communication Review*, Vol.35, No.3, pp.75-78, Jul. 2005.

[DoBi08] R. C. Dorf and R. H. Bishop, Modern Control System, *Pearson Prentice Hall*, 11th edition, 2008.

[DuMc06] N. Dukkipati, N. McKeown and A .G. Fraser, "RCP-AC congestion control to make flows complete quickly in any environment", *Proceedings of the IEEE International Conference on Computer Communications* (*INFOCOM*), pp.1 – 5, Apr.23-29, 2006.

[DuNi03] X. Duan, Z. Niu and J. Zheng, "Downlink optimization of radio resource allocation in DS-CDMA networks: an economic approach", *Proceedings of the 14th IEEE International Symposium on Persona1, lndoor and Mobile Radio Communication*, Vol.1, pp. 643-647, 2003.

[FeHu06] M. Felegyhazi and J. P. Hubaux, "Game theory in wireless networks: a tutorial", *EPFL Technical Report: LCA-REPORT-2006-002 Revised*, Jun. 21, 2006.

[FeSh02] W. C. Feng, K. G. Shin, D. D. Kandlur and D. Saha, "The blue active queue management algorithms", *IEEE Transactions on Networking*, Vol.10, No.4, Aug. 2002.

[FeVa03] W. Feng and S. Vanichpun, "Enabling compatibility between TCP Reno and TCP Vegas", *Proceedings of the Symposium on Applications and the Internet*, pp.301 – 308, 2003.

[FlHe04] S. Floyd and T. Henderson, “The new-Reno modification to TCP’s fast recovery algorithm”, *RFC 3782*, Apr. 2004.

[FlJa93] S. Floyd and V. Jacobson, “Random early detection gateways for congestion avoidance”, *IEEE/ACM Transactions on Networking,* Vol.1, No.4, pp.397–413, Aug. 1993.

[Floy03] S. Floyd, “High-speed TCP for large congestion windows”, *RFC 3649*, Dec. 2003.

[Floy08] S. Floyd, “Measurement studies of end-to-end congestion control in the Internet”, http://www.icir.org/floyd/ccmeasure.html, 2008.

[Floy12] http://www.icir.org/floyd/ccmeasure.html, as of Nov. 22, 2012.

[Floy91] S. Floyd, “Connections with multiple congested gateways in packet-switched networks part 1: one-way traffic”, *ACM Computer Communication Review*, Vol. 21, No. 5, pp. 30–47, Oct. 1991.

[FrAl09] G. Franks, T. Al-Omari, M. Woodside, O. Das and S. Derisavi, "Enhanced Modeling and Solution of Layered Queueing Networks", IEEE Transactions on Software Engineering, Vol. 35, No. 2, pp. 148-161, 2009.

[GrSh08] D. Gross, J. Shortle, J. Thompson, J. M. Thompson and C. M. Harris, Fundamentals of Queueing Theory, *John Wiley & Sons Inc.*, 4th Edition Inc., 2008.

[GuAd04] T. Gusmi, H. H. Adballa and A. Toumi, “Transient stability fuzzy control approach for power systems”, *Proceedings of IEEE International conference on Industrial Technology*, Vol.3, pp.1676-1681, Dec. 2004.

[GuYa07] X. Guan, B. Yang, B. Zhao, G. Feng and C. Chen, “Adaptive fuzzy sliding mode active queue management algorithms”, *Telecommunication Systems*, Vol.35, No.1-2, Jun. 2007.

[HaBa04] J. F. Hays, T. G. Babu, Modeling and Analysis of Telecommunications Network, *John Wiley & Sons Inc.*, 2004.

[HaBe07] M. M. Hassani and R. Berangi, “An analytical model for evaluating utilization of TCP Reno”, *Proceedings of the International conference on computer systems and technologies*, pp.IIIB. 14-1-7, 2007.

[HaKe99] V. Hayes, S. J. Kerry, A. Petrick, G. Fishel, R. O'Hara and A. Heberling, “IEEE 802.11b-1999: supplement to ANSI/IEEE Std 802.11, 1999 Edition”, *IEEE Network Working Group*, 1999.

[HaKi92] T. Haruki and K. Kikuchi, “Video camera system using fuzzy logic”, *IEEE Transactions on Consumer Electronics,* Vol. 38, No.3, pp.624–634, Aug 1992.

[Hanh08] H. Han, “Fuzzy controller design with input saturation”, *International Journal of Innovative Computing, Information and Control*, Vol.4, No.10, pp.2507-2521, Oct. 2008.

[HaPu99] J. Harju and K. Pulakka, “Optimisation of the performance of a rate-based congestion control system by using fuzzy controllers”, *Proceedings of the IEEE International Performance, Computing and Communications Conference*, pp.192–198, Feb. 10-12, 1999.

[Hark05] T. Harks, “Utility proportional fair resource allocation: An optimization oriented approach”, *Proceedings of QoS in Multiservice IP Networks*, Feb. 2005.

[Harr98] J. M. Harrison, "Heavy traffic analysis of a system with paralel servers: asymptotic optimality of discrete-review policies", Annals of Applied

Probability, Vol.8, No.3, pp.822-848, 1998.

[HaSc03] R. E. Haber, G. Schmitt-Braess, R. H. Haber, A. Alique and S. Ros, "Using circle criteria for verifying asymptotic stability in PI-like fuzzy control systems: Application to the milling process", *IEE Proceedings of Control Theory and Applications*, Vol.150, No.6, pp.619-627, 2003.

[HaYi04] A. Haj-Ali and H. Ying, "Structural analysis of fuzzy controllers with nonlinear input fuzzy sets in relation to nonlinear PID control with variable gains", *Automatica*, Vol. 40, pp. 1551-1559, 2004.

[HeBo00] U. Hengartner, J. Bolliger, and T. Gross, "TCP Vegas revisited," *Proceedings of IEEE International Conference on Computer Communications* (*INFOCOM*), Vol. 3, pp.1546–1555, 2000.

[Hong06] Y. Hong, "Adaptive controller design for Internet traffic: a control theoretical approach", *PhD dissertation, University of Ottawa*, 2006.

[HoYa07] Y. Hong and O. W. W. Yang,"Design of adaptive PI rate controller for best-effort traffic in the Internet based on phase margin", *IEEE Transactions on Parallel and Distributed Systems,* Vol. 18, No. 4, pp.550 – 561, Apr. 2007.

[HoYa10] Y. Hong and O. W. W. Yang, "An API-RCP design using pole placement technique", *Proceedings of the IEEE International Conference on Communications,* (*ICC*), pp.1-5, 2010.

[Huit99] C. Huitema, Routing in the Internet, *Prentice Hall Inc.*, 1999.

[HuRe96] J. Hu, J. Ren, J. Thompson and T. Sheridan "Fuzzy sliding control of a force reflecting telerobotic system", *Proceedings of the Fifth IEEE International Conference on Fuzzy Systems,* Vol.3, pp.2162 – 2167, Sep. 8-11, 1996.

[HuXi09] W. Hu and G. Xiao, "Design of congestion control based on instantaneous queue size in the routers", *Proceedings of the IEEE Global Communications Conference* (*GLOBECOM*), pp.1-6, Nov.30-Dec.04, 2009.

[Jaco88] V. Jacobson, "Congestion avoidance and control", *Proceedings of the ACM SIGCOMM Computer Communication*, pp.314-329, Aug.1988.

[Jaco90] V. Jacobson, "Modified TCP congestion avoidance algorithm", *mailing list, end-to-end-interest*, April 30, 1990.

[Jain96] R. Jain, "Congestion control and traffic management in ATM networks: recent advances and a survey", *Computer Networks and ISDN Systems*, Vol. 28, No.13, pp. 1723-1738, Oct. 1996.

[JaFl09] E. Jammeh, M. Fleury, C. Wagner, H. Hagras and M. Ghanbari, "Interval type-2 fuzzy logic congestion control for video streaming across IP networks", *IEEE Transactions on Fuzzy Systems*, Vol.17, No.5, 2009.

[JeBa03] J. Jelitto, A. N. Barreto and H. L. Truong, "Power and rate adaptation in ieee802-11a wireless lans", *Proceedings of the IEEE VTC 2003-Spring*, Vol.1, pp.413-417, 2003.

[JiCh10] Y. Jing, Z. Chen and G. M. Dimirovski, "Robust fuzzy observer-based control for TCP/AQM network systems with state delay", *Proceedings of 2010 American Control Conference*, pp.1350-1355, Jun.30-Jul.02, 2010.

[JiDo02] H. Jiang and C. Dovrolis, "Passive estimation of round-trip times", *ACM SIGCOMM Computer Communication Review*, Vol.32, No.3, 2002.

[JiWe05] C. Jin , D. Wei, S. Low, J. Bunn , H. D. Choe , J. C. Doyle , H. Newman, S. Ravot and S. Singh, "FAST TCP: from theory to experiments", *IEEE Network,* Vol. 19, pp.4 – 11, 2005.

[KaHa02] D. Katabi, M. Handley and C. Rohrs, “Congestion control for high bandwidth-delay product networks”, *Proceedings of the IEEE SIGCOMM 2002*, pp.89-102, Aug. 19-23, 2002.

[KaMe06] H. B. Kazemian and L. Meng, “An adaptive control for video transmission over Bluetooth”, *IEEE Transactions on Fuzzy System*s, Vol.14, No.2, pp. 263-274, Apr. 2006.

[Kata03] D. Katabi, “Decoupling congestion control and bandwidth allocation policy with application to high bandwidth-delay product networks”, *PhD dissertation, MIT*, 2003.

[Kelly97] F. P. Kelly, “Charging and rate control for elastic traffic”, *European Transactions on Telecommunications*, Vol.8, pp.33-37, 1997.

[KeMa98] F. P. Kelly, A. K. Maulloo and D. K. H. Tan, “Rate control for communication networks: shadow prices, proportional fairness and stability”, *Journal of the Operation Resource Society*, Vol.49, No.3 pp.237–252, Mar. 1998.

[KeRa08] F. Kelly, G. Raina and T. Voice, “Stability and fairness of explicit congestion control with small buffers”, *ACM SIGCOMM Computer Communication Review*, Vol.38, No.3, pp.51-62, Jul. 2008.

[KeRa10] F. P. Kelly and G. Raina, “Explicit congestion control: charging, fairness and admission management”, *Next-Generation Internet Architectures and Protocols*, Cambridge University Press, 2010.

[KiSa01] T. Kiryu, I. Sasaki, K. Shibai and K. Tanaka, “Providing appropriate exercise levels for the elderly”, *IEEE Engineering in Medicine and Biology Magazine*, Vol. 20, No.6, pp.116 – 124, 2001.

[Klei75] L. Kleinrock, Queue Systems, Volume I: Theory, John Wiley & Sons, Inc., 1975.

[Klei76] L. Kleinrock, Queue Systems, Volume II: Computer Applications, John Wiley & Sons, Inc., 1976.

[Kozi08] W. Kozinski, “Linear Matrix Inequalities (LMI) in Control”, http://www.isep.pw.edu.pl/~kozinski/skrypt/KSO_wykl_08.pdf, 2008.

[KrRo12] B. Kraemer, J. W. Rosdahl, A. P. Stephens, S. McCann, P. Ecclesine and et al., “IEEE 802.11: wireless LAN medium access control (MAC) and physical layer (PHY) specifications”, *Proceedings of the IEEE 802.11 Working Group*, 2012.

[KrWo12] S. Krishnan. J. Woodyatt, E. Kline, J. Hoagland and M. Bhatia, “A uniform format of IPv6 extension headers”, *Internet Engineering Task Force* (*IETF*), Apr.2012.

[LaAn02] R. J. La and V Anantharam, “Utility-based rate control in the Internet for elastic traffic”, *IEEE Transactions on Networking*, Vol. 10, No. 2, pp. 272-286, Apr. 2002.

[LaDo10] J. Lavaei, J. C. Doyle and S. H. Low, “Utility functionals associated with available congestion control algorithms”, *Proceedings of the IEEE International Conference on Computer Communications* (*INFOCOM*), pp.1-9, 2001.

[LaLe99] H. K. Lam, F. H. Leung and P. K. S. Tam, “An improved stability analysis and design of fuzzy control systems”, *Proceedings of the IEEE International Fuzzy Systems Conference*, Vol.1, pp.430-433, Aug. 1999.

[LaPh99] W. T. Lau, K. K. Phang, Y. Mashkuri and T. C. Ling, "Fuzzy logic control in ATM network", *Malaysian Journal of Computer Science*, Vol.12, No.2, pp. 47-56, 1999.

[LaZi06] I. D. Landau, and G. Zito, Digital Control Systems: Design Identification and Implementation, Springer-Verlag London Limited, 2006.

[LeHo00] S. J. Lee and C. L. Hou, "A neural-fuzzy system for congestion control in ATM networks", *IEEE Transactions On Systems, Man and Cybernetics---Part B Cybernetics*, Vol.30, No.1, pp.2-9, 2000.

[LeMo01] I. K. Leung, J. K. Muppala, "Packet marking strategies for explicit congestion notification (ECN)," *Proceedings of IEEE International Conference on Performance, Computing and Communications*, pp.17-23, 2001.

[LiEr10] R. Li, A. Eryilmaz, L. Ying and N. B. Shroff, "A unified approach to optimizing performance in network serving heterogeneous flows", *IEEE/ACM Trans. on Networking*, Vol.PP, No.99, pp.1-14, 2010.

[LiLi12] H. Li, H. Liu, H. Gao and P. Shi, "Reliable fuzzy control for active suspension systems with actuator delay and fault", *IEEE Transaction On Fuzzy Systems*, Vol.20, No.2, pp.342-357, Apr. 2012.

[LiSh06] X. Lin and N. B. Shroff, "Utility Maximization for communication networks with multi-path routing", *IEEE Transactions on Automatic Control*, Vol. 51, No. 5, pp.766 – 781, May 2006.

[LiWa09] Z. Liu, H. Wang, W. Hu, Y. Hong, O. Yang and Y. Shu, "An Implementation and Experimental Study of the Adaptive PI Rate Control Protocol", *Proceedings of the International Conference on High Performance Switching and Routing*, pp. 1-6, May 2009.

[LoLa99] S. H. Low and D. E. Lapsley, "Optimization flow control, I: basic algorithm and convergence", *IEEE/ACM Transactions on Networking*, Vol.7, pp. 861–874, 1999.

[LoPa02] S. H. Low, F. Paganini, J. Wang, S. Adlakha and J. C. Doyle, "Dynamics of tcp/red and a scalable control", *Proceedings of the IEEE International Conference on Computer Communications* (*INFOCOM*), Vol.1, pp.239-248, Jun. 23-27, 2002.

[MaMa08] K. Ma, R. Mazumdar and J. Luo, "On the performance of primal/dual schemes for congestion control in networks with dynamic flows", *Proceedings of the IEEE International Conference on Computer Communications* (*INFOCOM*), pp.326-330, Apr.15-17, 2008.

[MaPe06] C. Marcondes, A. Persson, M. Y. Sanadidi, M. Gerla, H. Shimonishi, T.Hama and T. Murase, "Inline path characteristic estimation to improve tcp performance in high bandwidth delay networks", *Proceedings of the 4th International Workshop on Protocols for Fast Long Distance Networks* (*PFLDnet*), 2006.

[Matl13] Fuzzy Logic Toolbox™ User's Guide, R2013b Version, MathWorks Inc, 2013.

[MeSa09] J. L. Meza, V. Santibanez, R. Soto and M. A. Llama, "Stable fuzzy self-tuning pid controller of robot manipulators", *Proceedings of the IEEE International Conference on System, Man, and Cybernetics*, pp.2624-2629, Oct. 2009.

[MiKl06] K. Michels, F. Klawonn, R. Kruse and A. Nürnberger, Fuzzy Control: Fundamentals, Stability and Design of Fuzzy Controllers, *Springer*, 1st edition, 2006.

[MoWa00] J. Mo and J. Walrand, "Fair end-to-end window based congestion control", *IEEE/ACM Transactions on Networking*, Vol. 8, No. 5, pp.556-567, Oct. 2000.

[MuYu07] S. A. Munir, W. Yu, B. Ren and J. Ma, "Fuzzy logic based congestion estimation for QoS in wireless sensor network", *Proceedings of the IEEE Wireless Communications and Networking Conference*, pp.4336 – 4341, 2007.

[NiCh06] B. Nick, C. Jin, S. H. Low and S. Hegde, "FAST TCP: motivation, architecture, algorithms, performance", *IEEE/ACM Transactions on Networking,* Vol.14, No.6, pp. 1246–1259, 2006.

[Nise11] N. S. Nise, Control System Engineering, *John Wiley&Sons Inc.*, 4th edition, 2011.

[NyDa06] C. N. Nyirenda and D. S. Dawoud, "Multi-objective particle swarm optimization for fuzzy logic based active queue management", *Proceedings of the IEEE International Conference on Fuzzy Systems*, pp.2231–2238, Jul.16-21, 2006.

[NyDa07] C. N. Nyirenda and D. S. Dawoud, "Fuzzy logic congestion control in IEEE 802.11 wireless local area networks: a performance evaluation", *Proceedings of the AFRICON 2007*, pp.1–7, Sep. 26-28, 2007.

[Ogat02] K. Ogata, Modern Control Engineering, *Prentice Hall*, 4th edition, 2002.

[Opnet05] OPNET Modeler Manuals, OPNET version 11.5, *OPNET Technologies Inc*, 2005.

[PaCh03] J. P. Pavon and S. Choi, "Link adaptation strategy for ieee 802.11 wlan via received signal strength measurement", *Proceedings of the IEEE International Conference on Communications,* (*ICC*), Vol.2, pp.1108-1113, May 2003.

[PaCh06] D. P. Palomar and M. Chiang, "A tutorial on decomposition methods for network utility maximization", *IEEE on Selected Areas in Communications*, Vol.24, No.8, pp.1439-1451, Aug.2006.

[PaDe08] J. Papandriopoulos, S. Dey and J. Evans, "Optimal and distributed protocols for cross-layer design of physical and transport layers in MANETs", *IEEE/ACM Transactions on Networking*,Vol. 16, No.6, pp. 1392-1405, 2008.

[PaPi02] A. Papoulis and S. Pillai, Probability, Random Variables and Stochastic Processes, 4th edition, *McGraw Hill*, 2002

[PaYu98] K. M. Passino and S. Yurkovich, Fuzzy Control, *Addison Wesley Longman, Inc.*, 1998.

[PiSe97] A. Pitsillides, Y. A. Sekercioglu and G. Ramamurthy, "Effective control of traffic flow in ATM networks using fuzzy explicit rate marking (FERM)", *IEEE Journal on Selected Areas in Communications, ,*Vol.15, No.2, pp.209–225, 1997.

[PrVa11] D. Pradas and M. A. Vázquez-Castro, "NUM-Based fair rate-delay balancing for layered video multicasting over adaptive satellite networks", *IEEE on Selected Areas in Communications*, Vol.29, No.5, pp.969-978, 2011.

[PuHa07] J. Pu and M. Hamdi, "Applying router-assisted congestion control to wireless networks: challenges and solutions", *High Performance Switching and Routing workshops*, pp.1-6, 2007.

[PuHa08] J. Pu and M. Hamdi, "Enhancements on router-assisted congestion control for wireless networks", *IEEE Transactions on Wireless Communications*, Vol.7, No.6, pp.2253-2260, Jun. 2008.

[QaZn09] I. A. Qazi, T. Znati, L. L. H Andrew,"Congestion control using efficient explicit feedback," *Proceeding of International Conference on Computer Communications* (*INFOCOM*), pp.10 – 18, 2009.

[QiCh02] D. Qiao, S. Choi and K. G. Shin, "Goodput analysis and link adaptation for IEEE 802.11a wireless LANs", *IEEE Transactions on Mobile Computing*, Vol. 1, No.4, pp.278-292, 2002.

[QiMa03] Y.Qiu and P.Marbach, "Bandwidth allocation in ad hoc networks: a price-based approach", *Proceeding of the IEEE International Conference on Computer Communications* (*INFOCOM*), pp.1-6, Mar.30-Apr.03, 2013.

[Qiu97] B. Qiu, "A predictive fuzzy logic congestion avoidance scheme", *Proceedings of the IEEE Global Communications Conference* (*GLOBECOM*), Vol. 2, pp.967 – 971, Nov.3-8, 1997.

[QuOz04] P. F. Quet and H. Özbay, "On the design of AQM supporting TCP flows using robust control theory", *IEEE Transactions on Automatic Control*, Vol.49, No.6, pp.1031-1036, Jun. 2004.

[Qwha14] http://www.qwhatis.com/what-is-opnet/, as of Mar. 17. 2014.

[RaFl99] K. K. Ramakrishnan and S. Floyd, "Proposals to add explicit congestion notification (ECN) to IP", *RFC 2481*, Jan.1999.

[ReJo95] S. Renganathan and J. S. Johnson, "A robust adaptive controller using fuzzy logic approach", *Proceedings of the IEEE/IAS International Conference on Industrial Automation and Control,* pp.95 – 99, Jan. 1995.

[RiYe08] B. Ribeiro, T. Ye and D. Towsley, "Resource-minimalist flow size histogram estimator", *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, pp.285-290, Oct. 20-22, 2008.

[Robe94] L. Roberts, "Enhanced PRCA proportional rate control algorithm", *ATM Forum*, Aug. 1994.

[RoHa77] N. Rouche, P. Habets and M. Laloy, Stability Theory by Liapunov's Direct Method, *Springer-Verlag New York Inc.*, 1977.

[SaKe04] V. Santibanez, R. Kelly and M. A. Lama, "Global asymptotic stability of a tracking sectorial fuzzy controller for robot manipulators", *IEEE Transactions On Systems, Man, and Cybernetics, Part B: Cybernetics*, Vol.34, No.1, pp.710-718, Feb.2004.

[SiTe01] T. Sijak, S. Tesnjak and O. Kuljaca, "Stability analysis of fuzzy control systems using describing function method", *Proceedings of the 9th Mediterranean Conference on Control and Automation*, Jun. 2001.

[Srik04] R. Srikant, The mathematics of Internet congestion control, *Birkhauser*, 2004.

[Stid04] J. S. Stidham, "Pricing and congestion management in a network with heterogeneous users", *IEEE Transactions on Automatic Control*, Vol.49, No.6, pp.976 – 981, 2004.

[SuZu07] J. Sun, M. Zukerman and M. Palaniswami, "Stabilizing RED using a fuzzy controller", *Proceedings of the IEEE International Conference on Communications,* (*ICC*), pp. 266 -271, Jun. 24-28, 2007.

[TeSz05] M. Tekala and R. Szbo, "Evaluation of scalable TCP", *Proceeding of the 3rd ACS/IEEE International Conference on Computer Systems and Applications*, pp.55-62, 2005.

[Trib13] M. Tribastone, "A Fluid Model for Layered Queueing Networks". IEEE Transactions on Software Engineering, Vol. 39, No. 6, pp.744–756, 2013.

[VoRa09] T. Voice and G. Raina "Stability analysis of a max-min fair rate control protocol (RCP) in a small buffer regime", *IEEE Transactions on Automatic Control*, Vol.54, No.8 pp.1908-1913, Aug. 2009.

[WaLi09] L. Wang and X. Liu, "New relaxed stabilization conditions for fuzzy control systems", *International Journal of Innovative Computing, Information and Control*, Vol.5, No.5, pp.1451-1460, May 2009.

[WaLt09] N. S. Walton, "Proportional fairness and its relationship with multi-class queueing networks", *Annals of Applied Probability*, Vol. 19, No. 6, 2301-2333, 2009.

[WeJi07] D. X. Wei, C. Jin, S. H. Low, et al, "Fast TCP: motivation, architecture, algorithms, performance," *IEEE/ACM Transaction on Networking*, Vol.14, No. 6, pp. 1249-1256, 2007.

[Welz02] M. Welzl, "Traceable congestion control", *Proceedings of the 3rd international conference on quality of future internet services and internet charging and QoS technologies. Springer-Verlag* Berlin, Oct. 2002.

[Welz05] M. Welzl, Network Congestion Control: Managing Internet Traffic, *John Wiley&Sons, Ltd.*, 2005.

[Wiki13] http://en.wikipedia.org/wiki/Social_welfare_function, as of Sep. 2013.

[WuLi01] J. C. Wu, H. H. Liu and Y. J. Lung, "An adaptive multirate IEEE802 wireless LAN", *Proceeding of 15th International Conference on Information Networking 2001*, pp.411-418, 2001.

[WuSh12] Z. G. Wu, P. Shi, H. Su and J. Chu, "Reliable H_∞ control for discrete-time fuzzy systems with infinite-distributed delay", *IEEE Transactions on Fuzzy Systems*, Vol.20, No.1, pp.22-31, Feb. 2012.

[WuSu11] L. Wu, X. Su, P. Shi and J. Qiu, "A new approach to stability analysis and stabilization of discrete-time t-s fuzzy time-varying delay systems", *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, Vol.41, No.1, pp.273-286, Feb.2011.

[WyAn03] B. Wydrowski, L. L. H. Andrew and M. Zukerman, "MaxNet: a congestion control architecture for scalable networks", *IEEE Communications Letters*, Vol. 7, No.10, pp.511 – 513, Oct. 2003.

[Vane97] T. W. Vaneck, "Fuzzy guidance controller for an autonomous boat", *IEEE Control Systems Magazine,* Vol.17, No.2, pp.43 – 51, Apr. 1997.

[Ying93] H. Ying, "The simplest fuzzy controllers using different inference methods are different nonlinear proportional-integral controllers with variable gains", *Automatica*, Nov.29, No.6, pp.1579-1589, 1993.

[YiSi90] H. Ying, W. Siler and J. J. Buckley, "Fuzzy control theory: a nonlinear case", *Automatica*, vol.26, No.3, pp. 513-520, 1990.

[YuGr08] G. R. Yu, "Robust fuzzy control of piezoelectric systems with input delays and disturbances based on piecewise Lyapunov functions", *International Journal of Innovative Computing, Information and Control*, Vol.4, No.10, pp.2721-2730, Oct. 2008.

[ZhAh05] Y. Zhang and M. Ahmed, "A control theoretic analysis of XCP", *Proceedings of the IEEE International Conference on Computer Communications* (*INFOCOM*), Vol. 4, pp. 2831 – 2835, Mar.13-17, 2005.

[ZhHe05] Y. Zhang and T. R. Henderson, "An implementation and experimental study of

the explicit control protocol (XCP)", *Proceeding of the IEEE International Conference on Computer Communications* (*INFOCOM*), Vol.2, pp.1037-1048, Mar.13-17, 2005.

[ZhLe06] Y. Zhang, D. Leonard and D. Loguinov, "JetMax: scalable max-min congestion control for high-Speed heterogeneous networks", *Proceedings of the IEEE International Conference on Computer Communications* (*INFOCOM*), pp.1-13, Apr. 23-29, 2006.

[ZhLu11] Q. Zhou, P. Shi, J. Lu and S. Xu, "Adaptive output-feedback fuzzy tracking control for a class of nonlinear systems", *IEEE Transactions on Fuzzy Systems*, Vol.19, No.5, pp.972-982, Oct. 2011.

[ZhMa07] R. Zhu and M. Ma, ""Adaptive rate selection scheme based on intelligent learning algorithm in wireless LANs", *Proceedings of the 2nd International Conference on Advances in Computation and Intelligence*, pp.529-538, 2007.

[ZhPh05] R. Zhang, Y. A. Phillis, and V. S. Kouikoglou, "Fuzzy control of queuing systems", *Springer-Verlab London Ltd*., 2005.

[ZhSh10] J. Zhang, P. Shi and Y. Xia, "Robust adaptive sliding-mode control for fuzzy systems with mismatched uncertainties", *IEEE Transactions on Fuzzy Systems*, Vol.18, No.4, pp.700-711, Aug. 2010.

[ZhWu05] G. Zhang, Y. Wu and Y. Liu, "Stability and sensitivity for congestion control in wireless networks with time varying link capacities", *Proceeding of the 13th IEEE International Conference on Network Protocols*, Nov. 2005.

APPENDIX A: A SUMMARY OF THE FUZZY LOGIC CONTROL

This appendix summarizes the concepts and components of a fuzzy logic controller along with the design example of the IntelRate controller described in Section 3.1 of Chapter 3. Most of its contents are cited from [PaYu98].

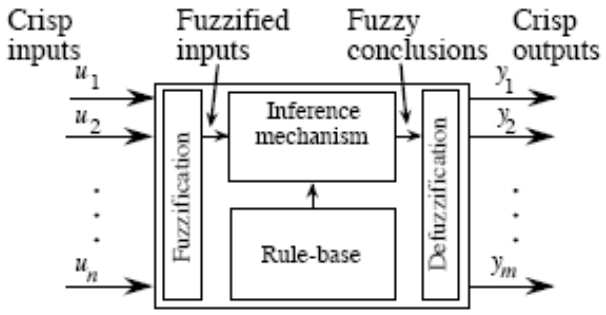


Fig. A.1: Fuzzy Controller Architecture

A typical fuzzy controller has the architecture depicted in Fig. A.1.

(1) Universes of Discourse (UOD)

Assume that the fuzzy system has inputs $u_s \in Us$, where $s = 1, 2, \ldots, n$ and outputs $y_s \in Ys$ where $s = 1, 2, \ldots, m$, as shown in Fig. A.1. The ordinary sets *Us* and *Ys* are called the "universes of discourse (UOD)" for u_s and y_s, respectively. In other words, *Us* and *Ys* are the domains of u_s and y_s.

(2) Crisp value

The terminology probably comes from multiple-valued concepts where continuous inputs values like $e(t)$ and $g(t)$ are all discretized to levels like NV, NL in Table 3.1 according to some reference values to indicate the levels of certainty. For example, $e(t)=a$, where a is a real number and is called crisp value.

(3) Linguistic variables

Linguistic variables are used to describe the inputs u_s and outputs y_s (see Fig. A.1). For example, we describe u_s and y_s with $\tilde{u}_s$ and $\tilde{y}_s$.

(4) Linguistic values

Linguistic values are used to describe characteristics of the linguistic variables. For example, in our IntelRate controller, linguistic values for input linguistic variables are defined by "Negative Very Large(NV)", "Negative Large (NL)", "Negative Medium(NM)", "Negative Small(NS)", "Zero(ZR)", "Positive Small(PS)", "Positive Medium(PM)", "Positive Big(PL)" and "Positive Very Big(PV)".

(5) Linguistic rules

Rules are the set of conditions mapping the inputs to the outputs, usually in *"If…Then…"* form. For example, in our IntelRate controller, when $e(t)$ is Zero(ZR) and $\tilde{g}(e(t))$ is Positive Small(PS), the rule is "If $\tilde{e}(t)$ is Zero(ZR) and $\tilde{g}(e(t))$ is Positive Small(PS), Then $\tilde{u}(t)$ is Big(BG)." "If…" is the premise, and "Then…" is the consequent. The sets of rules consist of a rule base.

(6) Membership function (MF)

Membership functions are used to describe the certainty of crisp input or output, designated by $\mu_{P_i^j}(p_i)$, where p_i is the crisp value. $\mu_{P_i^j}(p_i) \in [0,1]$. Next we introduce two types of MFs: triangular MFs and trapezoidal MFs.

A triangular membership function is depicted in Fig. A.2,

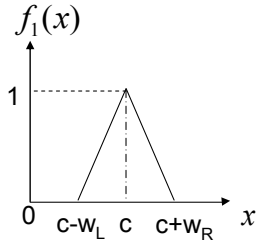


Fig. A.2: Triangular MF

Its equation is given as

$$f_1(x,c,w_L,w_R) = \begin{cases} \dfrac{x-c}{w_L}+1 & c-w_L < x \le c \\ \dfrac{c-x}{w_R}+1 & c < x \le c+w_R \\ 0 & \textit{Otherwise} \end{cases} \qquad \text{(A-1)}$$

where *c* is the centroid of triangular (i.e., the center point of the bottom of a triangle); w_L (w_R) is the left (right) width of the triangular function.

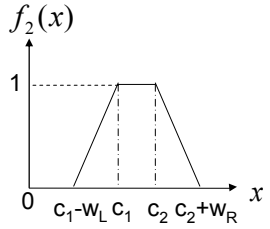


Fig. A.3: Trapezoidal MF

A trapezoidal MF is depicted in Fig. A.3. Its equation is given as

$$f_2(x, c_1, c_2, w_L, w_R) = \begin{cases} \dfrac{x - c_1}{w_L} + 1 & c_1 - w_L < x \le c_1 \\ 1 & c_1 < x \le c_2 \\ \dfrac{c_2 - x}{w_R} + 1 & c_2 < x \le c_2 + w_R \\ 0 & \textit{Otherwise} \end{cases} \tag{A-2}$$

where c_1 and c_2 are shown in Fig. A.3, w_L (w_R) is the left (right) width of the trapezoidal function.

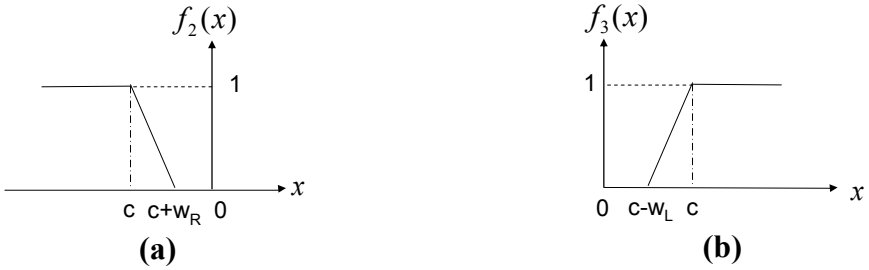


Fig. A.4: Trapezoid-like MFs

Since we just use trapezoid-like membership functions as shown above in Fig. A.4, we give their equations as follows.

$$f_2(x, c, w_R) = \begin{cases} \dfrac{c - x}{w_R} + 1 & c < x \le c + w_R \\ 1 & x \le c \\ 0 & \textit{Otherwise} \end{cases} \tag{A-3}$$

where c is shown in Fig. A.4a, and w_R is the distance between (c, $c+w_R$).

$$f_3(x,c,w_L)=\begin{cases}\dfrac{x-c_1}{w_L}+1 & c-w_L<x\le c\\ 1 & x\ge c\\ 0 & \textit{Otherwise}\end{cases} \tag{A-4}$$

where c is shown in Fig. A.4b, and w_L is the distance between (c-w_L, c).

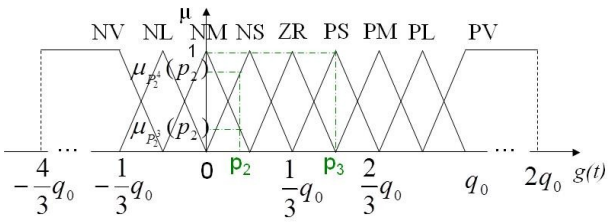


Fig. A.5: Membership Functions Example of *g(t)*

Fig. A.5 shows seven triangle and two trapezoid-like membership functions for the input $g(t)$. If q_0=300, the bottom of the triangle is 100. The membership values $\mu_{P_i^j}(p_i)$ can be obtained with equations (A-1) and (A-2) as shown below.

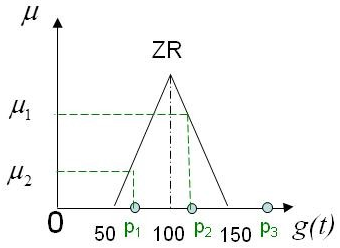


Fig. A.6: Triangular MF "ZR"

We first consider the triangle membership function designated by "ZR" in Fig. A.5 to show how to use (A-1). For clarity, we depict it separately in Fig. A6. For "NS", the triangle has a centroid c=100, bottom=100, w_L=50 and w_R=150. Thus certainty degree Eqn. (A-1) becomes

$$f_1(g(t),c,w_L,w_R)=\begin{cases}\dfrac{g(t)-100}{50}+1 & 50<g(t)\le 100\\ \dfrac{100-g(t)}{50}+1 & 100<g(t)\le 150\\ 0 & \textit{Otherwise}\end{cases}$$

For examples, when $g(t)=p_1=62.5$ $f_1(g(t),c,w_L,w_R)=0.25$; when $g(t)=p_2=112.5$, $f_1(g(t),c,w_L,w_R)=0.75$; and when $g(t)=p_3=180$, $f_1(g(t),c,w_L,w_R)=0$.

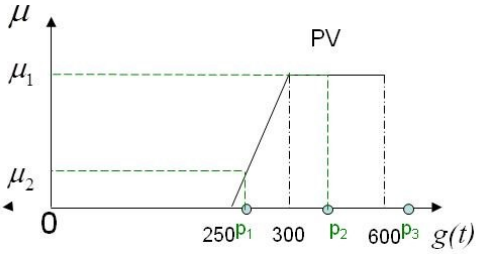


Fig. A.7: Trapezoid-like MF "PV"

Next we take the rightmost trapezoid-like function of Fig. A.5 as shown in Fig. A.7 to show how to use (A-2). When q_0=300, and the bottom of the triangle is 100, we have c_1=300, c_2=600 and w_L=50. For Fig. A.7, the (A-2) becomes

$$f_2(g(t),c_1,c_2,w_L)=\begin{cases}\dfrac{g(t)-300}{50}+1 & 250<g(t)\le 300\\ 1 & 300<g(t)\le 600\\ 0 & Otherwise\end{cases}.$$

Similarly, for a given value of $g(t)$, we thus can find its certainty degree $f_1(g(t),c,w_L,w_R)$, e.g. when $g(t)=p_1=260$ $f_1(g(t),c,w_L,w_R)=0.2$; when $g(t)=p_2=450$, $f_1(g(t),c,w_L,w_R)=1$; and when $g(t)=p_3=700$, $f_1(g(t),c,w_L,w_R)=0$.

(7) Fuzzy sets

A fuzzy set is a crisp set of pairing of elements of the UOD with their associated membership functions, e.g. in our IntelRate controller, a input fuzzy set is defined as

$$P_i^j=\{(p_i,\mu_{P_i^j}(p_i)):p_i\in P_i\}\quad i=1,2. \qquad \text{(A-5)}$$

(8) Fuzzification

Fuzzification refers to the process how the fuzzy system converts its crisp inputs into fuzzy sets. For example, in our IntelRate controller, we first use MFs functions (A-1) and (A-2) to get the certainty degree, then with (A-5) we get the fuzzy set. This process is called the "Singleton fuzzification".

(9) Firing level

Firing level is how certain a rule is to the current situation specified by the premises. Zadeh AND logic is usually used to determine this parameter with *minimum operation*. For example, in our IntelRate controller, given a rule "If $\tilde{e}(t)$ is Zero(ZR) and $\tilde{g}(t)$ is Positive Small(PS), Then $\tilde{u}(t)$ is Big(BG)." if the certainty level of the rule premises $\mu_{P_i^j}(p_1) = 0.25$ and $\mu_{P_i^j}(p_2) = 0.90$, then the final certainty of this rule to current situation is $\mu_{P_1^m \cap P_2^n} = \min\{0.25, 0.90\} = 0.25$, which is usually called firing level.

(10) Implied fuzzy sets

"Fuzzy set" is used so far to specify the certainty degree of input. Here, "implied fuzzy set" is introduced to specify the certainty level in that the output should be a specific crisp output *u*. Continuing on the example rule "If $\tilde{e}(t)$ is Zero(ZR) and $\tilde{g}(t)$ is Positive Small(PS), Then $\tilde{u}(t)$ is Big(BG)", $\mu(u) = \min\{\mu_{P_1^m \cap P_2^n}, \mu_{BG}(u)\} = \min\{0.25, \mu_{\overline{U}^6}(u)\}$. The justification for the use of the *minimum operation* to represent the implication is that we can be no more certain about our consequent than our premise in the rule [PaYu98].

(11) Inference mechanism

The above processes to obtain the "firing level" and "implied fuzzy set" are the mechanism to provide inferences on the inputs in order to find the rules and draw the conclusions for the output.

(12) Defuzzification

Defuzzification operates on the implied fuzzy sets produced by the inference mechanism and combines their effects to provide the "most certain" controller output. For example, our IntelRate controller applies COG (Center of Gravity) method to obtain the crisp output with the equation $u(t) = (\sum_{j=1}^{k} c_j S_j)/(\sum_{j=1}^{k} S_j)$ as mentioned in Section 3.1 of Chapter 3.

Above we introduced the elements involved in a typical fuzzy logic controller. Actually, the design of a fuzzy controller is the process to design all these fuzzy elements.

APPENDIX B: LYAPUNOV'S DIRECT METHOD

This appendix summarizes the basic ideas of LDM (Lyapunov's Direct Method) which is used to prove the stability of the IntelRate controller in Section 3.2 of Chapter 3. The interested readers can refer to the textbooks [MiKl06] and [PaYu98] for details of LDM .

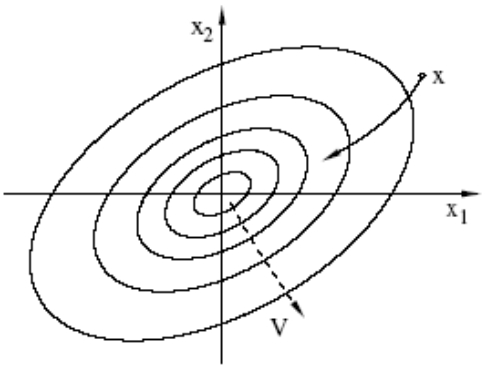


Fig. B.1: Illustration of Lyapunov's Direct Method

We define a positive definite Lyapunov function $V(\boldsymbol{x})$ (i.e. $V(\boldsymbol{x}) > 0$ for $\boldsymbol{x} \neq 0$ and $V(\boldsymbol{x}) = 0$ for $\boldsymbol{x} = 0$) where $\boldsymbol{x}$ is the system state vector and has the steady state at the origin of the state plane. Fig. B.1 shows the typical contour lines of $V(\boldsymbol{x})$, and $|V(\boldsymbol{x})|$ can actually be considered as a generalized distance from the position of rest (e.g., the length of the dashed line in Fig. B.1) [MiKl06].

The system state vector $\boldsymbol{x}$ may follow a certain curve in the course of time, which is determined by the system state equation, e.g., $\dot{\boldsymbol{x}} = f(\boldsymbol{x})$. If we can show that the derivative of the function $V(\boldsymbol{x})$ with respect to time is negative (i.e. $\dot{V}(\boldsymbol{x}) < 0$) for any state vector $\boldsymbol{x}$, then starting at any initial state $\boldsymbol{x}$, the state trajectory (e.g., the solid motion curve in Fig. B.1) would pass through all contour lines of $V(\boldsymbol{x})$ (such as the ellipsoids depicted in Fig. B.1) from outside to inside, and therefore the state vector would converge towards zero, thus implying the system is asymptotically stable.

Following the above basic idea, the theorem of LDM is stated as follows, which includes two parts. (The phrase "the position of rest" in the theorem means "the steady state".)

The LDM Theorem [MiKl06]**:**

(1) Asymptotical stability condition

Let $\boldsymbol{x} = 0$ be the position of rest of the dynamic system $\dot{\boldsymbol{x}} = f(\boldsymbol{x})$ (e.g., Eqn. (3-9)). Let $V(\boldsymbol{x})$ be a positive definite function, where $V(\boldsymbol{x})$ and its first partial derivatives are continuous functions within a certain region around the position of rest. Let furthermore its derivative with respect to time, i.e., $\dot{V} = \sum_{i=1}^{n} \frac{\partial V}{\partial x_i} \dot{x}_i = \sum_{i=1}^{n} \frac{\partial V}{\partial x_i} f_i$, be negative definite for this area. Then the position of rest is asymptotically stable, and the region forms its domain of attraction.

If $\dot{V}(\boldsymbol{x})$ is negative semi-definite ($\dot{V}(\boldsymbol{x}) \le 0$), then only the simple stability can be guaranteed. However, if the set of points with $\dot{V}(\boldsymbol{x}) = 0$ comprises no other trajectory except $\boldsymbol{x} = 0$, then we have asymptotic stability here, too.

(2) Globally Asymptotical Stability Condition

If the domain of attraction comprises the entire state space and if furthermore $V(\boldsymbol{x}) \to \infty$ follows from increasing distance from the position of rest, i.e., $|\boldsymbol{x}| = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2} \to \infty$, then the position of rest is globally asymptotically stable.

With the understanding of LDM, we can now discuss the rationale behind its usage to "indirectly" analyze the stability of the IntelRate congestion control system as depicted in Fig. 3.1 of Section 3.1.1. This is because it is very difficult or even impossible to express a fuzzy logic controller in an analytical form [MiKl06], not to mention that the system state equation is also difficult or even impossible. As suggested in [PaYu98], the idea to do the stability analysis for such a system is to reversely use the LDM. The approach is to let $\dot{V}(\boldsymbol{x}) < 0$, and to see how the fuzzy logic controller should behave to guarantee $\dot{V}(\boldsymbol{x}) < 0$ so that the system can meet the **Asymptotical stability condition**. Furthermore, if $|\boldsymbol{x}| \to \infty, V(\boldsymbol{x}) \to \infty$, the system should be globally stable as stated in the **Globally Asymptotical stability condition**. In what follows, we shall take IntelRate congestion control system as an example to see how to do the stability analysis for a fuzzy control system with LDM.

As we discussed in Section 3.2.2, we first get the system equation (3-9) in the form of

$$\begin{cases} \dot{x}_1 = h(x_1, x_2) \\ \dot{x}_2 = \begin{cases} x_1 + y_s + v(t) - c(t) & q(t) > 0 \\ [x_1 + y_s + v(t) - c(t)]^+ & q(t) = 0 \end{cases} \end{cases}$$, which has the steady state at $\boldsymbol{x}$=(x_1, x_2) = 0. Then we define the Lyapunov function $V(\boldsymbol{x}) = \frac{1}{2}(x_1^{\,2}(t) + x_2^{\,2}(t))$, and differentiate it to obtain the equation $\dot{V}(\boldsymbol{x}) = x_1(t)\dot{x}_1(t) + x_2(t)\dot{x}_2(t)$.

The **Asymptotical stability condition** requires that $\dot{V}(\boldsymbol{x}) = x_1(t)\dot{x}_1(t) + x_2(t)\dot{x}_2(t) < 0$, i.e. $x_1(t)h(x_2, \dot{x}_2) + x_2(t)(x_1 + y_s + v(t) - c(t)) < 0$ where $h(x_2, \dot{x}_2)$ is the change rate of the controller output. From this inequality, we obtain the condition in (3-12). That is

$$\begin{cases} \dot{y}(t) < \frac{1}{x_1(t)}[x_2(t)(c(t) - x_1(t) - v(t) - y_s) - \varepsilon] & if\ x_1(t) > 0 \\ \dot{y}(t) > \frac{1}{x_1(t)}[x_2(t)(c(t) - x_1(t) - v(t) - y_s) - \varepsilon] & if\ x_1(t) < 0 \end{cases}$$, where the two inequalities indicate how the fuzzy logic controller (determined by the variable $e(t)$ and $g(t)$ should behave. As analyzed in Section 3.2, to guarantee the system stable, the controller should be able to decrease the allowed sending rate under the heavy traffic (i.e., when $x_1(t) > 0$) while increasing the allowed sending rate under light traffic(when $x_1(t) < 0$). The subsequent analysis conducted in Theorem 1 and Theorem 2 in Section 3.2.3 and Section 3.2.4 shows the IntelRate controller indeed satisfies these requirements. Furthermore, since it also satisfies $|\boldsymbol{x}| \to \infty, V(\boldsymbol{x}) \to \infty$, the IntelRate congestion control system is globally stable.

APPENDIX C: THE INTELRATE STABILITY PROOF

This appendix provides the proof to Theorem 1 of Section 3.2.3 and Theorem 2 of Section 3.2.4 of Chapter 3. We conclude that the IntelRate controller can achieve globally asymptotical stability.

C.1 Stability Analysis under Light Traffic ($x_1(t) < 0$)

Theorem 1: The inequality $\dot{y}(t) > \frac{1}{x_1(t)}[x_2(t)(c(t) - x_1(t) - y_s - v(t)) - \varepsilon]$ *if* $x_1(t) < 0$ in (3-12) corresponds to the light traffic scenario in a router where the IntelRate controller allows each source to increase its sending rate until the desirable rate is reached while maintaining the system stable.

Proof: According to the upper inequality in (3-12) and to the definition of $x_1(t)$, that case of $x_1(t) < 0$ implies the incoming traffic $y(t)$ is arriving at a rate less than y_s. From Proposition 1 in Section 3.2.1, it means $y(t) < c(t)$, or $c(t) - x_1(t) - y_s - v(t) > 0$ (where $v(t)$=0, as assumed in Assumption 1 in Section 3.2.1). Therefore, the aggregate incoming rate $y(t)$ into the queue is less than the router service rate $c(t)$, and the system is working in a light traffic scenario. In this context, if $x_1(t)$ remains less than zero, $q(t)$ is working in empty state, i.e., $x_2(t) < 0$.

Table C.1: Rules for Light Traffic (gray)

Allowed Throughput *u(t)*		*e(t)*								
		NV	NL	NM	NS	ZR	PS	PM	PL	PV
g(t)	NV	ZR	ZR	ZR	ZR	ZR	ES	VS	SM	MD
	NL	ZR	ZR	ZR	ZR	ES	VS	SM	MD	BG
	NM	ZR	ZR	ZR	ES	VS	SM	MD	BG	VB
	NS	ZR	ZR	ES	VS	SM	MD	BG	VB	EB
	ZR	ZR	ES	VS	SM	MD	BG	VB	EB	MX
	PS	ES	VS	SM	MD	BG	VB	EB	MX	MX
	PM	VS	SM	MD	BG	VB	EB	MX	MX	MX
	PL	SM	MD	BG	VB	EB	MX	MX	MX	MX
	PV	MD	BG	VB	EB	MX	MX	MX	MX	MX

Note that ε in (3-12) is any small positive number to ascertain the stability condition in (3-10). Therefore, as ε approaches zero from the positive side, the right side of the upper

inequality in (3-12) would be positive, which in turn means the output change $\dot{y}(t)$ has to be positive. This requires the controller to increase its output $y(t)$ to make $q(t)$ reach q_s, provided sources have enough data to send. The scenarios of this for the controller to achieve stability are enumerated and exemplified below.

The rule region that is shaded gray in Table C.1 regulates the controller behaviors under the light traffic scenario where $e(t)=q_0-q(t)>0$ (since $x_2(t) < 0$).

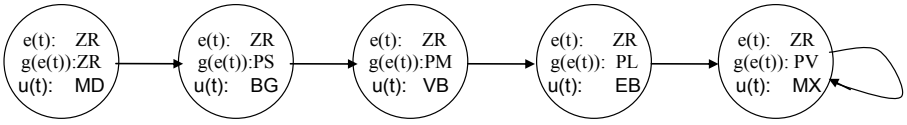


Fig. C.1: Controller Behavior When *e*(*t*)= "ZR" (Light Traffic Case)

With reference to Section 3.1.1, Fig. C.1 shows the controller behavior under one of the light traffic scenarios, i.e., when $e(t)$= "ZR" (which corresponds to the left column of the shaded area in Table C.1). Since $g(t) = \int_0^t e(\phi)d\phi$ and $e(t)>0$, $g(t)$ would keep on increasing from ZR to PS, PM, PL until PV, and eventually the controller output reaches the linguistic value "MX", as seen in Fig. C.1, where all the sources can send data with their maximum desired rates.

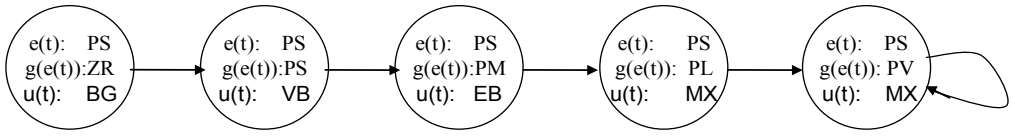


Fig. C.2: Controller Behavior When *e*(*t*)= "PS" (Light Traffic Case)

Fig. C.2 shows the controller behavior under the other light traffic scenario, i.e., when $e(t)$= "PS" (which corresponds to the right column in the shadowed area in Table C.1). Like the above $e(t)$= "ZR" case, the controller also increases its output continuously until the "MX" is reached.

Note that in the light traffic condition, the bandwidth of a link is enough to deal with all the incoming flows, and there is no congestion. The queue is often close to empty (Note this situation corresponds to the system model with the boundary condition $q(t)=0$ in equation (3-9)), or occasionally there can be a low buffer occupancy due to some arrival bursts. In the rare situation where there is occasional large data burst for the IntelRate controller (which

causes the queue size $q(t)$ over q_0, i.e. $e(t)<0$), the variable $g(t)$ would decrease its value for a while until the burst is subsided and $q(t)$ drops below q_0 . During the decrease, the controller output would return to the previous states for a while. For example, the LVs may go from "MX" back to "EB" or even "VB". After the burst dissipates, $g(t)$ will increase again. Therefore, the general practice that $e(t)>0$ under light traffic scenario is justifiable.

In summary, the controller behaviors described in the above two cases required by the upper inequality in (3-12) is met for the light traffic condition, and the proof is complete. ■

Theorem 1 establishes that the IntelRate controller can follow the prescribed behaviors to guarantee the system stability in the presence of light traffic. It is noteworthy that the queue size $q(t)$ mostly operates below q_0 under light traffic scenarios. However, this does not affect the system stability because the router is not congested at all in such scenarios. This has been verified by our simulations in Section 3.3.2.

Note also that the evolutions among NV,NL, NM and NS for $g(t)$ are also possible, but they correspond to the cases of $g(t)<0$ and $e(t)<0$ which will be discussed in the next section.

C.2 Stability Analysis under Heavy Traffic ($x_1(t) \geq 0$)

Theorem 2: The inequality $\dot{y}(t) < \frac{1}{x_1(t)}[x_2(t)(c(t) - x_1(t) - y_s - v(t)) - \varepsilon]$ *if* $x_1(t) > 0$ in (3-12) corresponds to the heavy traffic scenario in a router where the IntelRate controller decreases the source sending rate according to max-min fairness until the queue size reaches the TBO q_0 so that the asymptotical stability of the IntelRate control system is guaranteed.

***Proof*:** The condition $x_1(t) > 0$ in the lower inequality of (3-12) implies the incoming traffic $y(t)$ is arriving at a rate higher than y_s. From Proposition 1 in Section 3.2.1, it means $y(t) > c(t)$, or $c(t) - x_1(t) - y_S - v(t) < 0$. Since the aggregate incoming rate $y(t)$ to the queue is higher than the router service rate $c(t)$, and when $x_1(t)$ is to remain bigger than zero, $q(t)$ would build up quickly and pass over q_s. Then the system is now working under heavy traffic with $x_2(t) > 0$. Thus the right side of the lower inequality in (3-12) becomes negative. Note that ε in (3-12) is any small positive number to ascertain the stability condition in (3-10), and we can approximate it as zero. Now $\dot{y}(t) < 0$ means the controller should decrease its output $y(t)$ so that the IQSize $q(t)$ can be reduced back to q_0

(which is the steady state q_s) and then be stabilized there. To analyze whether the controller can behave this way, below we do the analysis based on the shaded rule region of Table C.2 which regulates the controller behaviors in the presence of heavy traffic.

Table C.2: Rules for Heavy Traffic (gray)

Allowed Throughput *u(t)*		*e(t)*								
		NV	NL	NM	NS	ZR	PS	PM	PL	PV
g(t)	NV	ZR	ZR	ZR	ZR	ZR	ES	VS	SM	MD
	NL	ZR	ZR	ZR	ZR	ES	VS	SM	MD	BG
	NM	ZR	ZR	ZR	ES	VS	SM	MD	BG	VB
	NS	ZR	ZR	ES	VS	SM	MD	BG	VB	EB
	ZR	ZR	ES	VS	SM	MD	BG	VB	EB	MX
	PS	ES	VS	SM	MD	BG	VB	EB	MX	MX
	PM	VS	SM	MD	BG	VB	EB	MX	MX	MX
	PL	SM	MD	BG	VB	EB	MX	MX	MX	MX
	PV	MD	BG	VB	EB	MX	MX	MX	MX	MX

For simplicity, assume the router is originally working in a non-congested condition where the controller allows the flows to send data with the LV "MX" (e.g. the scenario described in the proof of Theorem 1). We then assume an infinite number of ftp flows swarm into the router, each greedily demanding a desired sending rate. Obviously, this is the worst case assumption for a link to become severely congested. In such a scenario, when the link bandwidth is fully utilized and the congestion becomes imminent, the queue begins to build up (Note this situation corresponds to the system model under the boundary condition $q(t)>0$ in Equation (3-9)), and that $e(t) = q_0 - q(t)$ would become smaller and smaller. Eventually, $e(t)$ would become negative after $q(t)>q_0$. With reference to the shaded area of Table C.2, the negative $e(t)$, i.e., $e(t)<0$, corresponds to the state $e(t)$="ZR" and $e(t)$= "NS". As a result, $g(t)$ starts to decrease along the direction from "PV" to "NV" until $q(t)$ is working around the TBO q_0. When the controller input $g(t)$ decreases to "NV", the output would reach the linguistic value of "ZR" independent of the input $e(t)$, which means the controller is trying to assign an infinitesimal portion of bandwidth to each of the flows. As such, there are the following possible cases of the IntelRate controller behavior.

a) case of $e(t)$= "ZR" as shown in Fig. C.3 (which corresponds to the right column of the shaded area in Table C.2).

b) case of $e(t)$= "NS" as shown in Fig. C.4 (which corresponds to the left column of the shaded area in Table C.2).

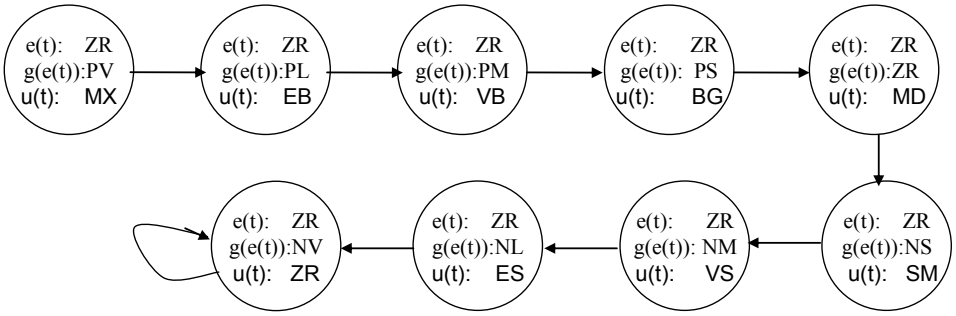


Fig. C.3: Controller Behavior under the Worst Traffic Condition for *e*(*t*)= "ZR"

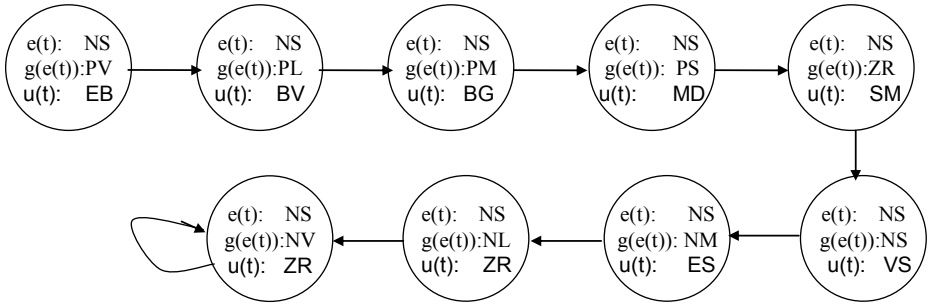


Fig. C.4: Controller Behavior under the Worst Traffic Condition for *e*(*t*)= "NS"

In both cases, one can see the controller output eventually decreases to "ZR" when $g(t)$ remains negative in order to maintain $q(t)$ to q_0.

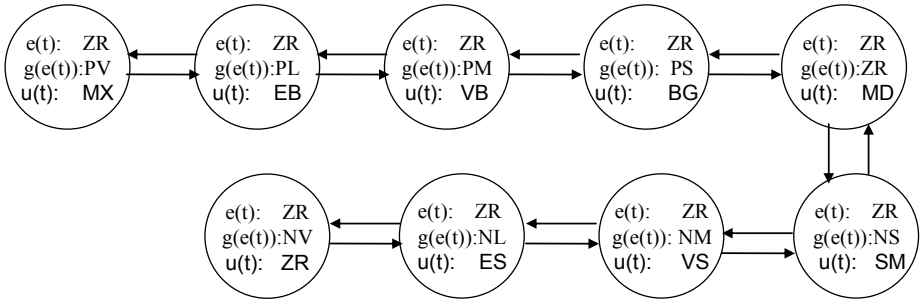


Fig. C.5: Controller Behavior under a Heavy Traffic Condition for *e*(*t*)= "ZR"

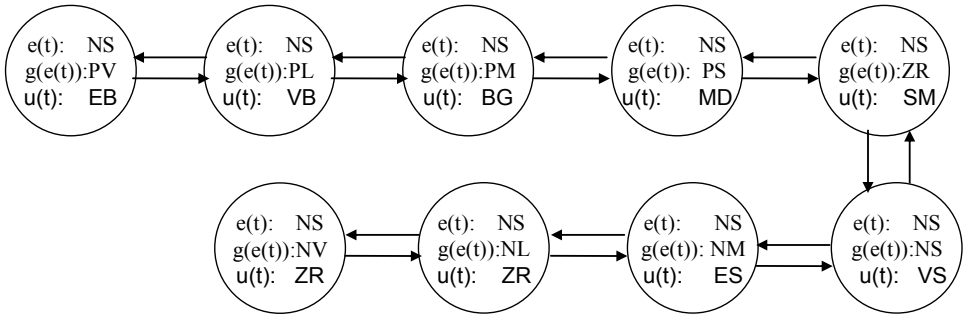


Fig. C.6: Controller Behavior under a Heavy Traffic Condition for *e*(*t*)= "NS"

Beside the above worst cases, the IntelRate controller output need not decrease all the way to "ZR". This leads to the other two cases:

c) case of $e(t)$= "ZR" with bi-directional transition as shown in Fig. C.5.

d) case of $e(t)$= "NS" as with bi-directional transition as shown in Fig. C.6.

System stability can still be accomplished in the cases c) and d) above. This is because, as shown in Fig. C.5 or Fig. C.6, the IntelRate controller can transit bi-directionally among the neighboring rules. Take the controller rule (ZR, NS, SM) in Fig. C.5 as an example. If this controller rule cannot decrease the queue size $q(t)$ to q_0, $g(t)$ will become more negative (because $e(t)$ is still less than 0), and consequently the controller will make a transition to the next rule (ZR, NM,VS) to further decrease its output. If $q(t)$ is still greater than q_0, the controller will continue to decrease the output by going to the next rule (ZR, NL, ES) for the same reason. This process is repeated until one output can decease the queue size $q(t)$ to q_0 and stabilize it there. Simulations in Section 3.3.3 shows how the IQSize is stabilized around q_0. In fact, under such a process, the system is working in max-min fairness. Similarly, for any given rule that the controller initially resides in, it can be shown that the IntelRate controller can always adjust the sending rates to shift $q(t)$ back to q_0 and make the system stable. In summary, the controller behavior in the heavy traffic conditions required by the lower inequality in (3-12) is met. ■

APPENDIX D: THE INTELRATE CHARACTERIZATION

This appendix explores further the characteristics of the IntelRate controller designed in Chapter 3. We characterize the IntelRate controller in a form similar to the traditional PI controller and show its unique property, which would help us to understand why the IntelRate controller does not need to evaluate network parameters but still works well and even better.

The IntelRate controller outputs are determined by the heuristic rules (e.g., Table 3.1), so it would be appropriate to see how they would affect the performance of our controller. Even though it may be impossible to express a fuzzy-logic-based controller in an analytical form [MiKl06], we would like to characterize/model it mathematically as much as we can in order to reflect the mathematical principle of the controller under the heuristic rules. We first establish a quasi-PI model for our IntelRate controller, and then compare it with the classical PI controllers.

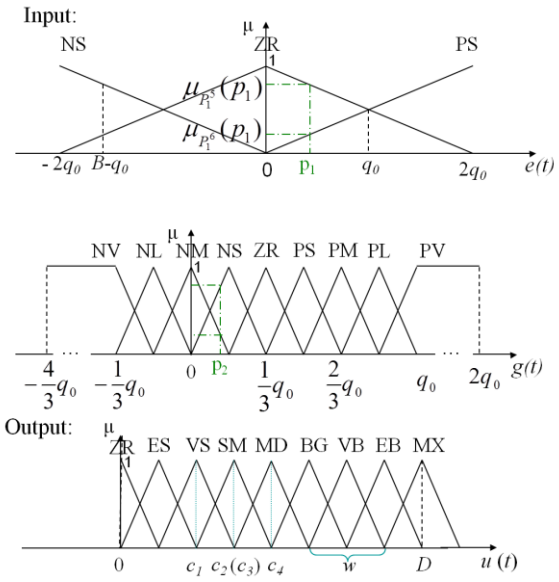


Fig. D.1: Membership Functions

D.1 Summary of Computational Steps

We below summarize the computational steps of the IntelRate controller discussed in Section 3.1.1. This would allow us to derive a general equation for different instances of the IntelRate controller in Section D.3.

Input variables:

$$e(t) = q_0 - q(t) \in [q_0 - B, q_0], \tag{D-1}$$

$$g(t) = \int_{t_0}^{t} e(\phi) d\phi \,. \tag{D-2}$$

Fuzzifications:

$$\mu_{P_1^j}(e(t)) = F(e(t), \overline{P}_1^{\,j}) \;\; j = 1,2,...,N \,, \tag{D-3}$$

$$\mu_{P_2^l}(g(t)) = F(g(t), \overline{P}_2^{\,l}) \;\; l = 1,2,...,N \,, \tag{D-4}$$

where N is the number of LVs (i.e., N=9 as shown in Table 1), $\mu_{P_1^j}(e(t))$ and $\mu_{P_2^l}(g(t))$ are the certainty degree of the crisp inputs, and operator F denotes the "singleton fuzzification" [PaYu98].

Inference:

$$\mu_{P_1^j \cap P_2^l} = I(\mu_{P_1^j}(e(t)), \mu_{P_2^l}(g(t))) \,, \tag{D-5}$$

$$\mu'_{\min(\max(j+l-\frac{N+1}{2},1),N)}(u) = I(\mu_{P_1^j \cap P_2^l}, \mu_{\min(\max(j+l-\frac{N+1}{2},1),N)}(u)) \,, \tag{D-6}$$

where the operator I decides the *firing level* of a rule with Zadeh AND logic, and $\mu'_{\min\left(\max\ (j+l-\frac{N+1}{2},\ 1),\ N\right)}(u)$ is the output fuzzy set of a rule.

Defuzzification and output:

$$\mu(u) = A(\mu'_{\min\left(\max\ (j+l-\frac{N+1}{2},\ 1),\ N\right)}(u)) \,, \tag{D-7}$$

$$u(t) = D(\mu(u)) \,. \tag{D-8}$$

where the operator A is used to aggregate the individual fuzzy outputs to form the overall output fuzzy set $\mu(u)$, and the operator D denotes the nonlinear defuzzification COG method [PaYu98].

D.2 Fundamental Properties

We have a total of 27 linguistic rules in the dashed part of Table 3.1 and Fig. D.1 shows

another MFs example of these rules [LiYa10]. Each rule corresponds to an MF graph. Fig. D.2 shows an example of the MF graph for the rule (ZR, PS, BG), where w_1 , w_2 and w are respectively the bottom widths of the triangles arising from the $e(t)$, $g(e(t)$ and $u(t)$. The enumerating variables m and n have a range of $\{1,2,...,N\}$, where N is the number of LVs.

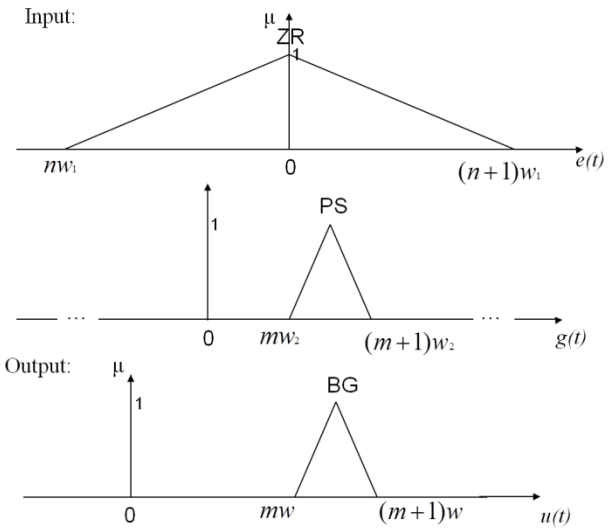


Fig. D.2: MFs for Rule (ZR, PS, BG)

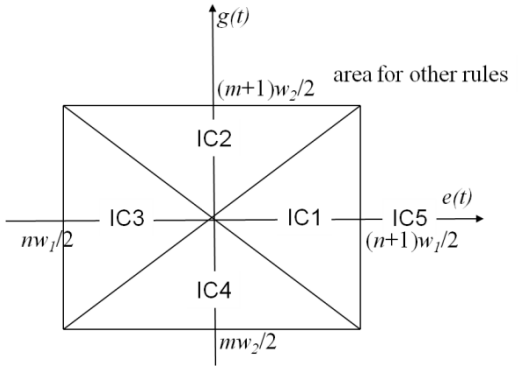


Fig. D.3: Possible ICs for Each Rule

According to the Zadeh AND logic between the inputs of each rule, two different conditions arise [Ying93]: (1) $\mu_{P_1'}(e(t))$ is greater than or equal to $\mu_{P_2'}(g(t))$, and (2) $\mu_{P_1'}(e(t))$ is less than or equal to $\mu_{P_2'}(g(t))$. From these conditions, one can obtain different possible Input Combinations(IC). For example, for the triangular MFs depicted in Fig. 3.3, there are four possible combinations IC1, IC2, IC3 and IC4 as shown in Fig. D.2. The reason is that each of $\mu_{P_1'}(e(t))$ and $\mu_{P_2'}(g(t))$ has two possible values in each rule.

In the MFs for the input $e(t)$ and $g(t)$ in Fig. D.1, we have $w_1=4q_0$ and $w_2=1/3q_0$. Therefore, each region $\mu_{P_1^j}(e(t))$ in IC1 or IC3 in Fig. D.2 is always less than or equal to $\mu_{P_2^l}(g(t))$, while each region $\mu_{P_1^j}(e(t))$ in IC2 or IC4 is always greater than or equal to $\mu_{P_2^l}(g(t))$. Regions other than IC1, IC2, IC3, IC4 (e.g., IC5) are due to the ICs of other rules, and the same graphical analysis can be applied.

We next state two properties [ChWa99] of the IntelRate controller which will help us to characterize our controller via Theorem 1 later on.

Property 1: Consider an IntelRate controller with crisp inputs $e(t)\in[\overline{P}_1^j,\overline{P}_1^{j+1}]$ and $g(e(t))\in[\overline{P}_2^{\hat{l}},\overline{P}_2^{\hat{l}+1}]$. If their corresponding LVs are not absolutely certain, there exist only two non-zero membership values in MFs for each of the two input variables, that is,

$$\mu_{P_1^j}(e(t))\begin{cases}\neq 0 \ if \ \ j=\hat{j},\hat{j}+1\\ =0 \ otherwise\end{cases}, \tag{D-9}$$

and

$$\mu_{P_2^l}(g(t))\begin{cases}\neq 0 \ if \ l=\hat{l},\hat{l}+1\\ =0 \ otherwise\end{cases}, \tag{D-10}$$

where $\hat{j},\hat{l}$ =*1,2, ..., N-1*. Otherwise, each input variable has only one membership value at 1, i.e. $\mu_{P_1^j}(e(t))=1$ and $\mu_{P_2^l}(g(t))=1$ which are absolutely certain. ■

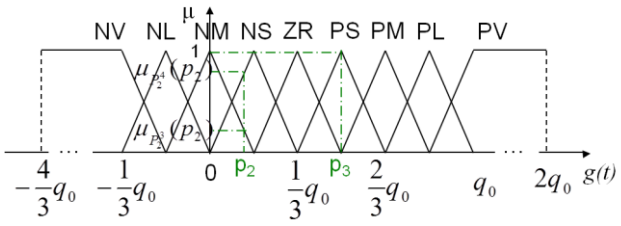


Fig. D.4: Membership Value of Input *g*(*t*)

***Proof*:** Given an arbitrary crisp input, if it is not located at the centroid (see Appendix A for definition) of a triangle nor outside the interval [$-1/3q_0$, q_0], its membership values always corresponds to only two non-zero MFs. There are two reasons: (1) a triangle is designed not to overlap with any triangles beyond its left or right adjacent neighbor; (2) the two adjacent neighbors of a triangle have no non-zero overlap. If a crisp input is located at the centroid of

a triangle or outside [-1/3q_0, q_0], its corresponding LV is absolutely certain, i.e. its membership value is 1. ■

Example

In Fig. D.3, $g(t)= p_2$ is an arbitrary crisp input which is inside the interval [-1/3q_0, q_0]. One sees that the two membership values of p_2 are non-zero values of $\mu_{P_2^3}(p_2)$ and $\mu_{P_2^4}(p_2)$ which correspond to two MFs of "NM" and "NS" respectively. On the other hand, $g(t)= p_3$ is a centroid and one sees that its corresponding LV (e.g. the membership values of p_3) is "PS", which is absolutely certain with a membership value of 1. Similar argument goes to the case of a crisp input located outside [-1/3q_0, q_0].

Property *2*: For the IntelRate controller, the two non-zero membership values of Property 1 have a sum of 1 and hence the summation of all membership values for $e(t)$ or $g(t)$ is also 1. That is,

$$\sum_{j=1}^{N} \mu_{P_1^j}(e(t)) = 1 \text{ or } \sum_{l=1}^{N} \mu_{P_2^l}(g(e(t))) = 1 \ . \qquad \text{(D-11)}$$

***Proof*:** Note that all the triangles are assumed to be isosceles with the same unit height and the same fixed bottom width, and they overlap in a form shown in Fig. D.3. From the geometry of the two adjacent isosceles, any input $e(t)$ or $g(t)$ not located at the centroid of a triangle would have its membership value $\mu_{P_1^j}(e(t))$ (or $\mu_{P_2^l}(g(t))$) equal to 1 minus the membership value of the other input, i.e., 1- $\mu_{P_1^{j+1}}(e(t))$ or 1- $\mu_{P_2^{l+1}}(g(t))$). Moreover, the membership values of this crisp input for other LVs are zero because by design, a triangle never overlaps with any other triangles beyond its non-overlapping adjacent neighbors. In the special case where a crisp input is located right at the centroid of a triangle, its membership value in the corresponding LV is 1, and zeros for other LVs according to Property 1. Summarizing all these cases, we have $\sum_{j=1}^{N} \mu_{P_1^j}(e(t)) = 1$ or $\sum_{l=1}^{N} \mu_{P_2^l}(g(t)) = 1$. ■

Example

Consider the crisp input $g(t) = p_2$ in Fig. D.3, which is a point not located in the centroid of

a triangle, it corresponds two non-zero membership values, i.e., $\mu_{P_2^3}(p_2)$ for "NM" and $\mu_{P_2^4}(p_2)$ for "NS". Since the triangles are isosceles, we must have $\mu_{P_2^3}(p_2) = 1 - \mu_{P_2^4}(p_2)$. That is $\mu_{P_2^3}(p_2) + \mu_{P_2^4}(p_2) = 1$. Since the membership values of p_2 for other LVs are zero, $\sum_{j=1}^{N} \mu_{P_2^j}(p_2) = 1$. For the crisp input $g(t)=p_3$ in Fig. D.3, it is a point located in the centroid of the triangle "PS", and the right (left) bottom vertex of the triangle "ZR" ("PM") at the same time. Obviously, the membership value for "PS" is 1, and zeros for "ZR" or "PM". Thus the total sum would be also 1.

D.3 Quasi-PI Controller Model

Since our IntelRate controller uses fuzzy logic to heuristically partition the inputs into different regions and produce a heuristic output for each region, it is difficult to describe all regions with exactly the same mathematical model. Fortunately, due to the similarities of the ICs and two properties presented above, we can extend the ideas in [Ying93] by generalizing the mathematical models of different regions to do the analysis. The generalized equation will be formulated in the theorem and its proof below.

Theorem 1: The IntelRate controller that uses crisp inputs $e(t)$ and $g(t)$, the singleton fuzzification, the isosceles triangular and trapezoid-like MFs, the Zadeh AND operator, the rules listed in Table 3.1, and the nonlinear COG defuzzifier is structurally equivalent to a nonlinear quasi-PI controller in a form of $u(t) = K_{fP}(t)e(t) + K_{fI}(t)g(t) + K_c$ with time-varying gains $K_{fp}(t)$ and $K_{fI}(t)$, both of which are the functions of $e(t)$ and $g(t)$.

***Proof*:** For an arbitrary input pair ($e(t)$, $g(t)$), we use the fuzzification equations (D-3) and (D-4) to obtain their membership values $\mu_{P_1^j}(e(t))$ and $\mu_{P_2^l}(g(t))$. By analyzing the relationship between $\mu_{P_1^j}(e(t))$ and $\mu_{P_2^l}(g(t))$, we can find their possible ICs. The firing levels can then be determined with equations (D-5) and (D-6), and their corresponding IC regions thus can be found. By applying the *COG* defuzzification equations (D-7) and (D-8), the controller $u(t) = K_{fP}(t)e(t) + K_{fI}(t)g(t) + K_c$ in a quasi-PI form can thus be obtained with the time-varying gains $K_{fp}(t)$ and $K_{fI}(t)$. ■

Example:

We choose an input pair (p_1, p_2) as shown in Fig. D.4, where $e(t) \in (0, q_0)$ and $g(e(t)) \in (\frac{1}{12} q_0, \frac{1}{6} q_0)$. According to Property 1, the pair of input (p_1, p_2) corresponds to (ZR, NM, VS), (ZR, NS, SM), (PS, NM, SM) and (PS, NS, MD). These rules are positioned under regions IC1, IC2 or IC3 after analyzing whether $\mu_{P_1^j}(t)$ is greater than or equal to $\mu_{P_2^l}(g(t))$. For this particular instance, no rules are located in IC4 region. Computed with Zadeh AND logic, their firing levels are $\mu_{P_1^5}(e(t))$, $\mu_{P_2^3}(g(t))$, $\mu_{P_2^3}(g(t))$ and $\mu_{P_1^6}(e(t))$, respectively. According to Property 2, $\mu_{P_1^5}(e(t)) + \mu_{P_1^6}(e(t)) = 1$.

Using the firing levels discussed previously, and by applying COG defuzzification discussed in Section D.1, we obtain the output

$$u(t) = \frac{\sum_{j=1}^{k} c_j S_j(t)}{\sum_{j=1}^{k} S_j(t)} = \frac{c_1 S_1(t) + c_2 S_2(t) + c_3 S_3(t) + c_4 S_4(t)}{S_1(t) + S_2(t) + S_3(t) + S_4(t)}, \tag{D-12}$$

where

$$S_1(t) = w \times [\mu_{P_1^5}(e(t)) - \mu_{P_1^5}(e(t)) \times \mu_{P_1^5}(e(t)))/2], \tag{D-13}$$

$$S_2(t) = S_3(t) = w \times [\mu_{P_2^3}(g(t)) - \mu_{P_2^3}(g(t)) \times \mu_{P_2^3}(g(t))/2], \tag{D-14}$$

$$S_4(t) = w \times [(1 - \mu_{P_1^5}(e(t))) - (1 - \mu_{P_1^5}(e(t))) \times (1 - \mu_{P_1^5}(e(t))))/2] \tag{D-15}$$

$$c_2 = c_3 = c_1 + 0.5w, \tag{D-16}$$

$$c_4 = c_1 + w. \tag{D-17}$$

For the triangle MFs,

$$\mu_{P_1^5}(e(t)) = 1 + [c_{I1} - e(t)]/(0.5w_1), \tag{D-18}$$

$$\mu_{P_2^3}(g(t)) = 1 + [g(t) - c_{I2}]/(0.5w_2). \tag{D-19}$$

Substituting (D-18) and (D-19) into (D-13) to (D-15) and simplifying, we obtain

$$S_1(t) = w \times \{1 + [c_{I1} - e(t)]/(0.5w_1)\} \times \{0.5 - [c_{I1} - e(t)]/w_1\}, \tag{D-20}$$

$$S_2(t) = S_3(t) = w \times \{1 + [g(t) - c_{I2}]/(0.5w_2)\} \times [0.5 - [g(t) - c_{I2}]/w_2], \tag{D-21}$$

$$S_4(t) = w \times \{-[c_{I1} - e(t)]/(0.5w_1)\} \times \{1 + [c_{I1} - e(t)]/(2w_1)\}. \quad \text{(D-22)}$$

Plugging (D-16), (D-17) into the numerator of equation (D-12), $u(t)$ becomes

$$u(t) = \frac{c_1 S_1(t) + (c_1 + 0.5w)S_2(t) + (c_1 + 0.5w)S_3(t) + (c_1 + w)S_4(t)}{S_1(t) + S_2(t) + S_3(t) + S_4(t)}$$

$$= \frac{w \times (S_2(t) + S_4(t))}{S_1(t) + S_2(t) + S_3(t) + S_4(t)} + c_1, \quad \text{(D-23)}$$

After substituting the expressions of S_2 and S_4 into the numerator of $u(t)$ and simplify, we obtain

$$u(t) = \frac{w^2 \times (-\frac{c_{I1} - e(t)}{0.5w_1}) \times \left[1 + \frac{c_{I1} - e(t)}{2w_1}\right]}{S_1(t) + S_2(t) + S_3(t) + S_4(t)} + \frac{w^2 \times \left[1 + \frac{g(t) - c_{I2}}{0.5w_2}\right] \times \left[0.5 - \frac{g(t) - c_{I2}}{w_2}\right]}{S_1(t) + S_2(t) + S_3(t) + S_4(t)} + c_1. \quad \text{(D-24)}$$

Thus the IntelRate controller equation can be written in the following compact form:

$$u(t) = K_{fP}(t)e(t) + K_{fI}(t)g(t) + K_c, \quad \text{(D-25)}$$

where

$$K_{fP}(t) = [w^2 \times (-c_{I1}/e(t) + 1) \times (2w_1 + c_{I1} - e(t))]/[w_1^2(S_1 + S_2 + S_3 + S_4)], \quad \text{(D-26)}$$

$$K_{fI}(t) = \{w^2 \times [(0.5w_2 - c_{I2})/g(t) + 1] \times (0.5w_2 - g(t) + c_{I2})\}/[0.5w_2^2(S_1 + S_2 + S_3 + S_4)], \quad \text{(D-27)}$$

$$K_c = c_1. \quad \text{(D-28)}$$

Eqn. (D-25) is very similar to the standard PI controller $u(t) = K_P e(t) + K_I g(t)$ except for an extra term K_c. This extra term originates from the non-zero output with the linguistic value "MD" when $e(t)$ and $g(t)$ are both "ZR", according to the controller rule (ZR, ZR, MD) in Table 3.1. For a given instance of ($e(t)$, $g(t)$), Eqn. (D-26) and (D-27) show that both $K_{fp}(t)$and $K_{fI}(t)$ are time-varying functions with respect to the IQSize dynamics where S_1, S_2, S_3 and S_4 are functions over time as well. For different input instances ($e(t)$, $g(t)$), the expressions for the gain $K_{fp}(t)$and $K_{fI}(t)$ may be different because the different input pairs ($e(t)$, g(t)) would generate different ICs and firing levels, and the $S_1(t)$, $S_2(t)$, $S_3(t)$ and $S_4(t)$ functions are different according to the equations (D-13) to (D-27). Therefore, K_P and K_I are different functions over time for different ($e(t)$, $g(t)$) pairs. To this end, we rewrite $K_{fp}(t)$ as

$K_{fp}(t, LV1,LV2)$ to emphasize that $K_{fp}(t)$ are different for different inputs ($e(t)$, $g(t)$) according to the input LVs used. Likewise we use $K_{fI}(t, LV1,LV2)$ instead of $K_{fI}(t)$. For example, the expressions for (D-26) and (D-27) are those of $K_{fp}(t, \text{“}ZR(PS)\text{”}, \text{“}NM(NS)\text{”})$ and $K_{fI}(t, \text{“}ZR(PS)\text{”}, \text{“}NM(NS)\text{”})$, respectively.

As a summary, the above derivations and discussions provide a theoretical ground to explain why the IntelRate controller can adapt and perform better under dynamic environment than other existing controllers such as XCP, RCP or API-RCP (which were designed with the classical PI principle and their gains are fixed).

Appendix E: The IntelRate Computational Complexity

As a cooperation-based protocol, the IntelRate controller designed in Chapter 3 needs to reside in routers to exercise the congestion control. As stated in Chapter 1 and Chapter 3, the IntelRate controller saves much computation resources by not relying on network parameter estimations. In this appendix, we analyze the computational complexity of the IntelRate controller in a router and make comparisons with its counterparts to show how the IntelRate controller saves the computation resources.

E.1 Complexity Analysis of Parameters Estimation

The explicit congestion controller in a router has to rely on some network parameters to provide explicit congestion control signals. These parameters include the link bandwidth, the number of flows and the queue size to calculate the source sending rate, all of which require computational and/or memory resources. Therefore, we shall analyze the computational complexity and memory requirement of measuring each of these parameters in a router in order to provide a comparison. For convenience, we use the word "estimate" and "measure" interchangeably in the text.

E.1.1 Link Bandwidth

So far there has not been much investigation about measuring the link bandwidth of a router. Note that we are not talking about the end-to-end bandwidth measurement that can be found a good number of research papers or practical tools. The measurement of link bandwidth of a router is tricky [PuHa08] due to the dependency on technology/scenario and the maintenance of network states [AbAr11], as a couple of examples. Furthermore, the accuracy is hard to obtain. In the following, we take the approach proposed in [PuHa08] to analyze the complexity of bandwidth estimation. It is an adaptive method by employing the following equations. The interested readers can refer to [PuHa08] for details.

Let q and $output$ be the IQSize and router throughput (equivalently the output traffic rate), respectively. Also let q_{avg} and $output_{avg}$ be their weighted moving average. Then these averages can be updated as follows [PuHa08]:

$$C' = \begin{cases} output_{avg} & if\ q_{avg} \geq 1 \\ (1+\alpha)\cdot C' & else \end{cases} \quad \text{(E-1)}$$

$$q_{avg} = w \cdot q + (1-w) \cdot q_{avg} \quad \text{(E-2)}$$

$$output_{avg} = w \cdot output + (1-w) \cdot output_{avg} \quad \text{(E-3)}$$

where $\alpha = 0.1$, and w=0.2 as adopted in [PuHa08].

Eqn. (E-1) shows two basic ideas measuring link bandwidth [PuHa08]:

1) If q_{avg} is greater than or equal to one packet, the output interface is busy and the router is encountering heavy traffic in last control interval. Therefore, the output traffic rate can be a good estimation of the current link capacity.

2) If q_{avg} is less than one packet, the output link is idle and the router is under light traffic during the last control interval. In this situation, the link capacity estimation would be increased by a factor $(1+\alpha)$ and wait a control interval to see whether the queue is building up.

To implement the bandwidth estimation, one can see from the above that the congestion controller must require one timer to update (E-1)-(E-3) periodically (i.e., with a defined control interval T). The detailed operations to accomplish the computation of (E-1)-(E-3) are analyzed below.

1) To obtain $output_{avg}$ in (E-1) in one interval T under heavy traffic, the congestion controller needs to compute *output* first and then use (E-3) to compute $output_{avg}$. To compute *output*, the controller needs to add the size of all outgoing packets up in one interval T, and then use the summation to divide T. For instance, if there are Θ packets passing through the router in one interval T and each packet has R bits, then $output = \Theta R/T$. (Note that the bandwidth estimation is more complicated for variable packet size because the router has to measure the packet size every time it receives a packet.) To compute *output*, the congestion controller hence needs to perform Θ-1 additions (to count the packets), 1 multiplication and 1 division in T. To find $output_{avg}$ at the end of interval T, the controller needs to put $output = \Theta R/T$ into (E-3) which involves 1 addition and 2 multiplications (Note, 1-w=0.75 is a constant; thus we removed the subtraction operation therein). As a summary, the total operations to obtain $output_{avg}$ in one interval T are Θ additions, 2 multiplications and 1 division.

2) To obtain $(1+\alpha)\cdot C'$ in (E-1) under light traffic, the controller needs one multiplication. (Also note, $(1+\alpha)$ =1.1 is a constant as α = 0.1. Consequently, we get rid of one addition

operation.)

3) To compute q_{avg} in (E-2) in one control interval T, the controller needs to find the q first. Since the computation of q is only to increase q by one (i.e., one addition) when a packet arrives in a queue and decrease q by one (i.e., one subtraction) when a packet departs from the queue, the total operational steps are TY_{in} and TY_{out}, respectively, where Y_{in} is the incoming rate and Y_{out} is the departure rate to a router in packets/sec in one control interval. Finally, there are 1 addition and 2 multiplications (like the computations in (E-3)) to determine q_{avg}. Altogether, the total number of operations required to obtain q_{avg} in one interval T are TY_{in}+1 additions, TY_{out} subtractions and 2 multiplications.

4) In one interval T, the controller needs to do one comparison as shown in (E-1), i.e., $q_{avg} \geq 1$.

Considering 1)-4) above, we see all the operations to measure link bandwidth in one control interval T require (Θ+*TYin*) additions, *TYout* subtractions, 4 multiplications, 1 division and 1 comparison. If the router uses Intel 64 and IA-32 type CPU, the time for it to implement a floating point addition, subtraction, multiplication, division, and a comparison instruction would be about 3, 3, 5, 20 and 3 clock cycles, respectively [Intel12]. Therefore, considering all the above operations, the total clock cycles required to estimate the bandwidth in an interval T is Π=3(Θ+TY_{in}+TY_{out})+43. For simplicity, we assume Y_{in}=Y_{out}=Cp under heavy traffic, and thus Π=3(Θ+2CpT)+43.

To assist the computations in (E-1)-(E-3), they need to store/maintain the variables C, q, $q_{avg,}$ *output*, $output_{avg}$ in memory. We shall compare the memory requirement associated with these variables in Section E.2.5.

E.1.2 The Number of Flows

The API-RCP needs to estimate the number of flows to maintain its congestion control system stable. As discussed in [HoYa10], originally API-RCP uses an address queue (or a hash table) to estimate the number of flows [HoYa07], but this approach becomes impractical due to extensive CPU computation required for millions of flows. So an improved approach was then proposed in [HoYa10], where a CPU cost-effective method, called zombie list, is used to estimate the number of flows. Each zombie in the list is used to store the flow identifier.

Let *Hit* be the result of comparing the arriving packet with the packets already in the zombie list. If they match, it means they come from the same flow and the controller declares a hit. Define P to be an estimate of the hit frequency, N' be the estimated number of flows and w a constant. The zombie list method is as follows, and the interested readers can refer to [OtLa99] for details.

$$Hit = \begin{cases} 0 & if\ no\ hit \\ 1 & if\ hit \end{cases} \quad \text{(E-4)}$$

$$P=(1-w) \cdot P+w \cdot Hit \quad \text{(E-5)}$$

$$N'=1/P \quad \text{(E-6)}$$

Equations (E-4)-(E-6) are computed sequentially whenever a packet arrives. (E-5) requires one comparison and no other operations. Like (E-2), (E-5) requires one addition and 2 multiplications. To find N', one division in (E-6) is needed. So totally the computations per packet to evaluate the number of flows are 1 addition, 2 multiplications, 1 division, and 1 comparison. With the same Intel 64 and IA-32 type CPU mentioned in Section E.1.1, the total clock cycles of these operations are 36.

To assist the computations in (E-4)-(E-6), the variables required to be stored/maintained in memory for (E-4)-(E-6) are *Hit*, P, N'. Note that a maximum memory of 1000 zombies is required in [OtLa99] because the number of flows is 1000 therein for a bottleneck bandwidth of 45Mbps. Actually, for a practical router with bandwidth more than 10Gbps, it is possible to have 100,000 flows passing through it. Since there is at least one zombie per flow, the zombie list requires at least 100,000 zombies. Assuming 12 bytes per zombie to record flow identifiers such as source address, destination address, count, timestamp and etc [OtLa99], the zombie list will need a memory of 1.2 million bytes.

Table E.1: Complexity of Estimating Different Parameters (in clock cycles)

Link Bandwidth	The No. of Flows	IQSize
$3(\Theta+2CpT)+43$	36	6

E.1.3 IQSize

As discussed in Section E.1.1 for computing q, to obtain the instantaneous queue size, the controller just needs one addition when a packet arrives and one subtraction when a packet departs. So totally the computation for each packet is just 6 clock cycles [Intel12]. The

variable required to store is just q.

E.1.4 Computation Complexity Comparison

Table E-1 summarizes the above discussion in the complexity in obtaining different parameter estimations. The complexity is measured in terms of clock cycles. We can see that measuring IQSize is much more economic than the other two methods.

E.2 Computation Complexity of Different Controllers

In this section, we will do the complexity and memory usage analysis for three explicit congestions controllers, i.e., XCP, API-RCP and the IntelRate controller. We choose XCP and API-RCP to compare because both of them are known for their simple implementations. The interest readers can refer to [KaHa02] and [HoYa07] for their implementation details.

E.2.1 XCP

The complexity of XCP processing each packet involves about 4 additions, 3 subtractions, 3 multiplications, 3 divisions, and 5 comparisons [KaHa02]. Assuming the router uses the Intel 64 and IA-32 type CPU mentioned in Section E.1.1, the processing time involved in each packet in XCP would be $X1$=111 clock cycles. However, in one control Interval T, in addition to the per packet manipulations, XCP needs some manipulations to estimate the bandwidth and shuffle the bandwidth. The number of manipulations to shuffle bandwidth has 4 additions, 2 subtractions, 3 multiplications, 6 divisions and 5 comparisons [KaHa02], and sum them up as $X2$=168 clock cycles. As discussed in Section E.1.1, the operations to estimate the link bandwidth in one control interval is $\Pi=3(\Theta+2CpT)+43$.

We can now get the total complexity X for XCP controller in one control interval T using $X=CpTX1+X2+\Pi$. Substituting in $X1$, $X2$, Π from above, we have $X=111CpT+3(\Theta+2CpT)+211=117CpT+3\Theta+211$. For CpT which there are hundreds of thousands of packets (even more for high-speed routers) arriving every T, $X\approx117CpT+3\Theta$. From the definition of Θ in Section E.1.1, $\Theta\approx$CpT. So X≈120CpT.

Finally, at least two timers are required to implement XCP [KaHa02].

E.2.2 API-RCP

Upon one packet arrival, API-RCP needs 5 additions, 1 subtraction, 7 multiplications, 1

division and 3 comparisons [HoYa07]. So the total clock cycles so far for an Intel 64 and IA-32 type CPU are $X1$=82. From Section E.1.2, the operations of estimating the number of flows in API-RCP are $X2$=36 per packet. Hence, the total clock cycles per packet are 82+36=118. For the sake of comparison, in an interval T, the total clock cycles that API-RCP requires will be X=118CpT.

E.2.3 The IntelRate Controller

As with XCP, the code of the IntelRate controller includes two parts (see Section 3.1.3). One part is implemented per packet, and the other part is implemented every control interval T. Below the computation complexity of the IntelRate controller will be analyzed via the two parts, respectively. For convenience, we do the analysis based on the pseudo-code of the IntelRate controller.

/*Sample $e(t)$ and $g(t)$ */

(1) $e(t)=q_0-q(t)$;

(2) ★ $g(t)=\int_0^t e(\phi)d\phi$.

(3) if ($u(t)$< Req_rate)

(4) ★★ $Req\_rate=u(t)$;

Fig. E.1: Pseudo-code Implemented Per Packet

★ In the coding, the integration is done via summation (which is an example of the discretization for a continuous-time model to be implemented in digital computers [LaZi06]), i.e., $g(k)= g(k-1)+step*e(k)$, where we choose the packet inter-arrival time as the integration "step", and k is a positive integer as the departure of the kth packet (k=1, 2, 3, ...). Our way to decide the "step" is to dynamically compute the reciprocal of the total number of the packets arrived in every second. This can be simply implemented using a timer with a cycle of one second, during which counting how many packets enter the router.

★★We omit the "=" operation in the complexity analysis for all the controllers.

Table E.2: Computation Complexity Per Packet

Code	Types of manipulations	Total manipulations
(1)	1 subtraction	1 subtraction
(2)	2 addition, 1 subtraction and 1 multiplication	1 addition, 1 subtraction and 1 multiplication
(3)	1 comparison	1 comparison

Fig. E.1 is the pseudo-code implemented upon each packet departure. Table E.2 lists the manipulations involved in each statement of the pseudo-code in Fig. E.1. Assuming the Intel 64 and IA-32 type CPU to be used, the total manipulations per packet cost can thus be $X1$=17 clock cycles.

```
● /*Fuzzify e(t) */
   (1) for(i=0;i<N;i++)
       {
   (2) μ1[i] =  the values of membership functions given the values of e(t).
   (3) if( μ1[i] =!0 )                    //The code line (3) and (4) are to do the match;
    {
   (4) match_ e(t) =i;                    //match_e(t) stores the index of the first non-zero  μ1[i] ;
    }
    }
● /*Fuzzify g(t) */
   (5) for(j=0;j<N;j++)
    {
   (6) μ2[ j] =  the values of membership functions given the values of g(t).
   (7) if ( μ2[ j] =!0 )                  //The code line (7) and (8) are to do the match
   {
   (8) match_g(t)=j;              //match_g(t) stores the index of the first non-zero  μ2[ j] ;
   }
     }
● /*Inference and defuzzification*/
   (9) for(i=0;i<2;i++)
     (10)    for(j=0;j<2;j++)
               {
     (11)      inputs_certainty=min( μ1 (match_e(t)+i),  μ1 ( match_g(t)+j);
     (12)    area_S=2*width_output_MF*( inputs_certainty-inputs_certainty*
          inputs_certainty/2);
     (13)        den_area_total+=area_S;
     (14)      ★★★num_area_total+= area_S*center_output_MF;
    }
● /*compute fuzzy logic crisp output u(t)*/
   (15) u(t)= num_area_total/ den_area_total;
```

Fig. E.2: Pseudo-code Implemented Every Control Interval

★★★With the way designed to decide the outmost edge value D in [LiYa10], "width_output_MF" and "center_output_MF" can be dynamically updated every control interval.

Table E.3: Complexity Analysis for a Control Interval

Code	Types of manipulations	Total manipulations
(1)	1 addition and 1 comparison	N+1 addition and N+1 comparison
(2)	2 subtractions, 1 division and 1 comparison	2N subtractions, N divisions and N comparisons
(3)	1 comparison	N comparisons
(4)	no manipulation (just assign value j to the variable match_g(t)	-
(5)	1 addition and 1 comparison	(N +1) additions and (N +1) comparisons
(6)	2 subtractions, 1 division and one 1 comparison	2 N subtractions, N divisions and N comparisons
(7)	1 comparison	N comparisons
(8)	no manipulation	-
(9)	1 addition and 1 comparison	3 additions and 3 comparisons
(10)	1 addition and 1 comparison	6 additions and 6 comparisons
(11)	2 addition and 1 comparison	8 additions and 4 comparisons
(12)	1 subtraction, 3 multiplications and 1 division	4 subtractions, 12 multiplications and 4 divisions
(13)	1 addition	4 additions
(14)	1 multiplication and 1 addition	4 additions and 4 multiplications
(15)	1 division	1 division

Fig. E.3 is the pseudo-code implemented per control interval T, and Table E.3 is the enumerations of different manipulations for each statement in Fig. E.2, where N is the number of LVs in the IntelRate controller. The total number of operations in Table E.3 in one control interval T consists of 2N+27 additions, 4N+4 subtractions, 16 multiplications, 2N+6 divisions and 6N+15 comparisons. For an Intel 64 and IA-32 type CPU [19], these operation requires $X2$=76N+338 clock cycles. For N=9 in our controller, $X2$=1022 clock cycles. From Section E.1.3, the clock cycles per packet to measure the queue size is $X3$=6.

Summing $X1$, $X2$ and $X3$ up, we can now obtain the total complexity X for the IntelRate controller in one control interval T, we have X=$X1CpT$+$X2$+$X3CpT$=23CpT+1022.

As with XCP in Section E.2.1, for CpT with hundreds of thousands of packets in a router every T, X≈23CpT.

Table E.4: Implementation Complexity of Different Controllers (in clock cycles)

Cycle	IntelRate controller	XCP	API-RCP
One interval T	23 CpT	120 CpT	118 CpT
Per Packet (Normalized by CpT)	23	120	118

Quantitatively, for a 3.40GHz CPU in a router, the execution time of the IntelRate controller per packet as shown in Table E.4 is about 6.76ns (i.e., 23/3.4G=6.76ns).

E.2.4 Summary of Complexity Comparison

Based on the above analysis, the computation complexity comparison among the IntelRate controller, XCP and API-RCP is summarized in Table E.4.

Table E.4 shows that the IntelRate controller is more than FIVE TIMES less in the computation complexity per packet than the other two typical explicit congestion controllers, i.e., XCP and API-RCP. Since there is millions of packets into the router every second, the cost saved for CPU with the IntelRate controller is practically more significant.

E.2.5 Memory/Buffer Requirement Comparison

For simplicity in determining the memory requirement, we will not distinguish of variables (e.g., an integer type vs. a double type) used in the XCP, the API-RCP or the IntelRate controller in their implementation. Instead we assume all controller variables have the same type, i.e., a *double* type (assume one *double* type variable has 8 bytes). This is because the distinction of such will not affect the performance (e.g. order of complexity) for big memory users.

Starting with the IntelRate controller, we need N variables to store the certainties of crisp values or center positions of linguistic variables. Together with other variables, the IntelRate controller would require a total of $(5N+8)$ variables. For example, using N=9 in the IntelRate controller would require 53 variable, or 424 bytes.

Next, XCP needs at least 32 variables [KaHa02]. Considering the 4 variables required by link bandwidth estimation in Section E.1.1, XCP totally requires 36 variables, or 288 bytes.

Finally, API-RCP needs 17 variables [HoYa07], or 136 bytes. However, this becomes insignificant when considering the giant memory consumption in its zombie list. As discussed in Section E.1.2, for a practical router, the zombie list can require a memory of 1.2 million bytes. For a high-speed router, the zombie list will be bigger than 1.2 million bytes.

Table E.5: Memory Requirement of Different Controllers (in bytes)

Storage	IntelRate controller	XCP	API-RCP
Memory for controller variables	424	288	>1.2 million
Normalized buffer requirement	<1/3	1	1

As a summary, Table E.5 summarizes the memory requirement of different solutions. As

seen in Row 1, although the IntelRate controller requires a bit more memory than XCP in terms of the controller implementation, it is much less than the API-RCP because of its giant zombie list. Further, Row 2 compares the buffer requirement by recalling that the design of the IntelRate controller requires less than 1/3 of the buffer capacity (as discussed in Section 6.1.1.5) compared with XCP and API-RCP. The normalization constant for Row 2 is the BDP. Overall, the IntelRate controller significantly saves a lot of memory usage.

Regarding the implementation of the IntelRate controller, the OPNET documentation is extensive and the non-proprietary code (some codes from collaborating industry are usually proprietary) may be requested from Prof. Yang where appropriate and when available.

APPENDIX F: INVESTIGATION OF DIFFERENT INTELRATE DESIGN ISSUES

As a complement to Section 3.3 of Chapter 3, we experimentally investigate some design issues of the IntelRate controller in this appendix, e.g., fuzzy smoother and choices of some design parameters. Besides, how the controller can deal with the short-lived flows is also illustrated. The simulation settings used for the following experiments are the same as described in Section 3.3.1.1.

F.1 Fuzzy Smoother (FS)

Section 3.1.1 in Chapter 3 mentioned the use of a FS mechanism to make controller output smoother. In this section, we shall show how the performance of the IntelRate control system with FS outperforms that without FS.

We conduct the experiments in a bottleneck bandwidth of 1Gbps with B=24000 packets and TBO=8000 packets, and set the number of LVs to 9. For the experiment without FS, we call it "experiment 1"; for the experiment with FS, we call it "experiment 2". Both experiments are conducted under the same heavy traffic scenario where each flow's desired rate is 20Mbps.

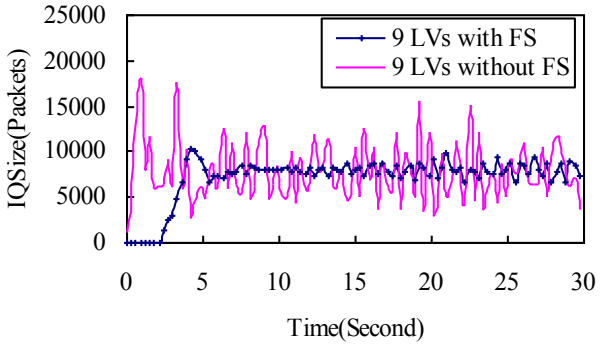


Fig. F.1: IQSize Dynamics with and without FS Mechanism

Fig. F.1 shows the router IQSize dynamics for the two experiments. The IQSize generated by the control system with the FS is much smoother than the one by the system without FS. Specifically, when the system starts, The IQSize in "experiment 1" has a sharp increase, and even though the fluctuations become smaller after the first 5 seconds, but they are still bigger than its counterpart in "experiment 2". In contrast, the IQSize in "experiment 2" first shows a zero IQSize, and then increases gradually. Even though a small overshoot

happens after it reaches the TBO position, the IQSize behaves smoother than the one in "experiment 1" afterwards. The reason will be analyzed later on.

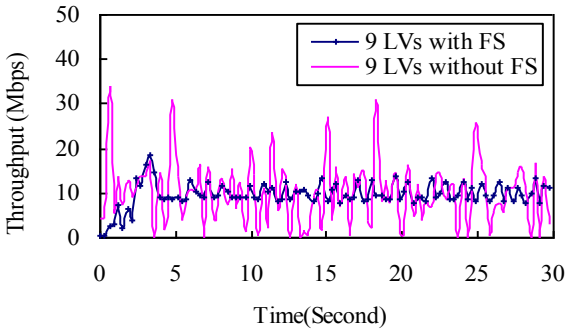


Fig. F.2: Throughput Dynamics with and without FS Mechanism

Fig. F.2 shows the source sending rate for the same two experiments. It shows the similar trend to Fig. F.1, i.e. the source throughput generated by the system with FS behaves much smoother than that without FS. In the beginning, the source in "experiment 1"gives sharp throughput, while the one in "experiment 2" shows more suppressed. Moreover, during the whole remaining simulated time, the throughput given by "experiment 2" behaves much smoother than that in "experiment 1". Both experimental results can be analyzed with Fig. 3.2, Fig. 3.3 and Table 3.1 in Chapter 3 as follows.

The evaluation called "experiment 1" uses the above settings in Fig. 3.2. When the router starts, the variable $q(t)$=0, and the crisp inputs of $e(t)$ *and* $g(t)$ are q_0 and 0, respectively. From Fig. 3.2, the LVs of these two crisp inputs correspond to "PV" and "ZR". Mapping these two inputs to output according to Table 3.1, the IntelRate controller gives the crisp output with a LV of "MX". This means the controller allows the source to send data with "Maximum" sending rate at the beginning. The heuristic rule at this moment is (PV, ZR, MX). The results can be seen in Fig. F.2 where the source throughput in "experiment 1" has a sharp increase at the beginning, and results in the IQSize peaks as shown in Fig. F.1.

On the other hand, the evaluation called "experiment 2" uses the settings in Fig. 3.3. When the router starts, the crisp inputs of $e(t)$ *and* $g(t)$ are also q_0 and 0, but they correspond to the LVs of "ZR(or PS)" and "NM" respectively. According to the rules in Table 3.1, they generate the outputs with a LV of "VS" or "SM", i.e. the controller only allows the sources to send data with a "Very Small" or "Small" rate when the router starts. Consequently, the queue size and source throughput in "experiment 2" increases mildly, as one can see in Fig.

F.1 and Fig. F.2.

Summarizing the comparisons in both figures, IQSize and throughput without FS are more oscillating than those with FS, and the overshoot in the former one is sometimes more than 100 [Nise11] (i.e. *%OS*>100) than that in the latter one. We also notice that the cost is the starting time of a router becomes longer when we use FS because the IQSize or source sending rate increases in a milder pace.

F.2 Parameter *m*

We have conducted various experiments to study the effect of the width parameter *m*. In the first experiment, we use a link bandwidth of 4Gbps, and set TBO=4000 packets. The desired rates of the sources from ftp group 1 to group 5 are 13.11Mbps, 22.94Mbps, 32.77Mbps, 65.44Mbps and 72.09Mbps, respectively.

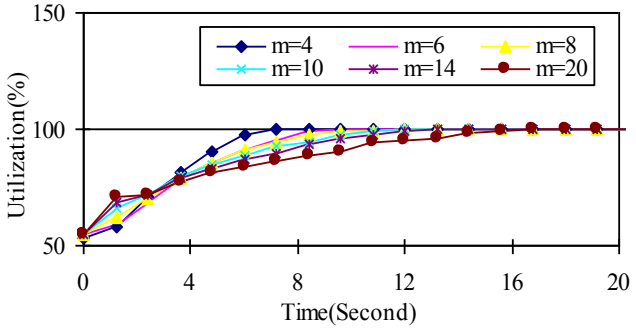


Fig. F.3: Link Utilization

Fig. F.3 shows the time evolution of the link utilizations in the router for different *m* values. For example, when *m*=4, the utilization appears to approach the steady state value of 100% exponentially. The 2% settling time[8] *Ts* is measured to be 6.0s while the rise time[9] *Tp* is measured to be 4.3s for *m*=4. The behavior is similar as *m* increases but both the settling time and rise time become bigger. This is because too big of an *m* value leads to small partitions along $e(t)$ or $g(t)$ in the IntelRate controller, which brings about the long inference process of the fuzzy logic. Specifically, the settling time *Ts* with *m*=6, 8, 10, 14 and 20 are separately about 6.5s, 6.9s, 7.4s, 9.0s and 11.4s, respectively. In this regard, the relationship between *Ts* and *m* can be linearly approximated as $Ts=0.25m+5.3$. The rise time with *m*=6, 8, 10, 14 and 20 are about 4.8s, 5.2s, 5.6s, 6.5s and 8.2s. Similarly, the relationship between *Tp*

[8] Settling time is the time required for the transient to reach and stay within ±2% of the steady state value [DoBi08].
[9] Rise time is the time required for the waveform to go from 10% to 90% of the final value [DoBi08].

and m can be linearly approximated as $Tp = 0.2m + 3.5$. Therefore, from the perspective of having a short rise time or settling time, smaller m is preferred.

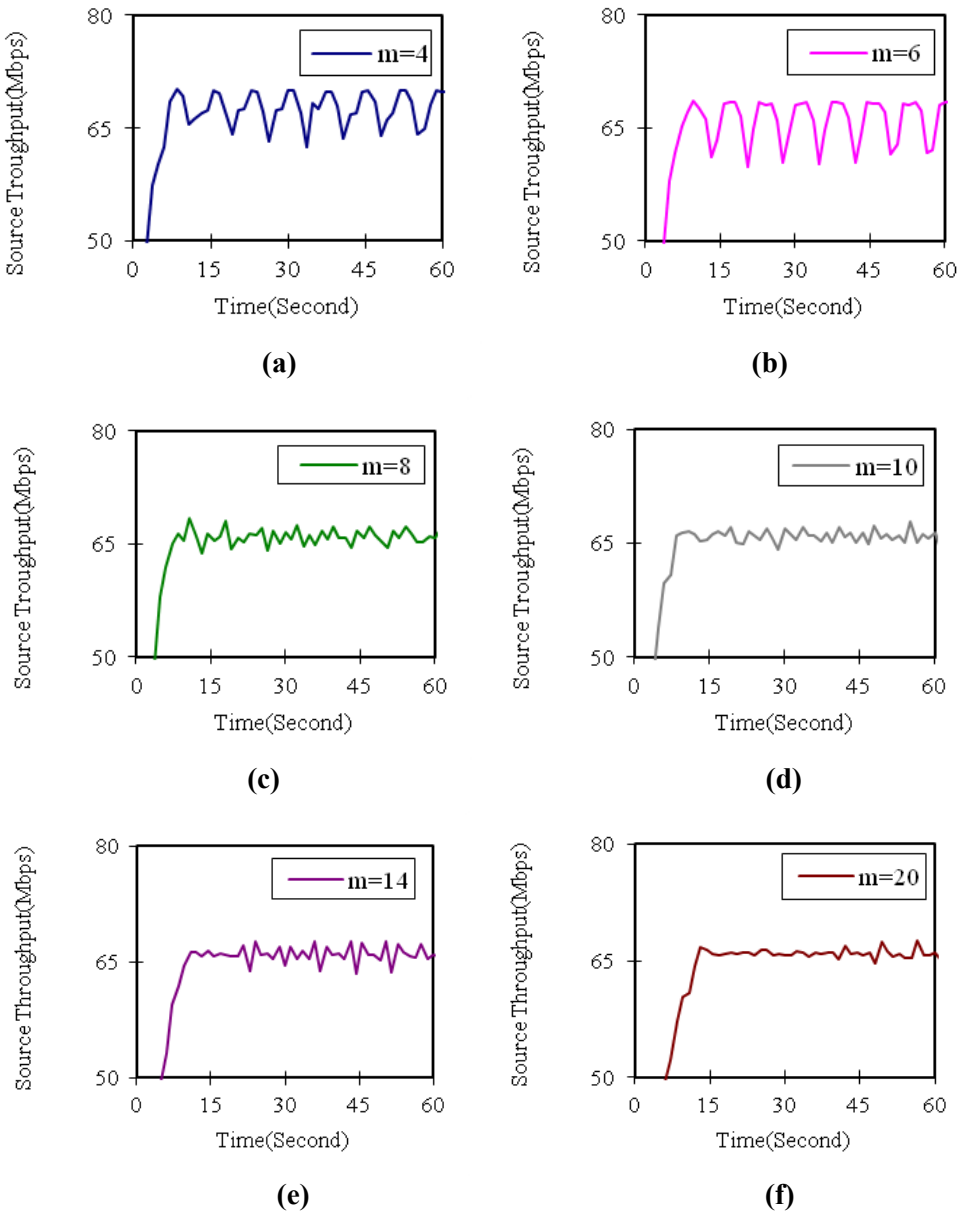


Fig. F.4: The Source Throughput Dynamics

Fig. F.4 shows the source throughput dynamics under different m. From Fig. F.4a, the source rate increases approximately in exponential from its initial value, overshoots the steady state value and oscillates with an amplitude (trough or crest to the average) of about 3.5Mbps after the transients. Fig. F.4b shows that the similar increase trend to Fig. F.4a, and oscillates with an amplitude of about 3.8Mbps. From Fig. F.4c to Fig. F.4f (i.e., m=8, 10, 14,

20), the oscillations are about 1.8Mbps, 1.2Mbps, 1.6Mbps and 0.72Mbps, respectively (Note in Fig. F.4f, big oscillation just happens occasionally). One can see the general trend is the oscillations become smaller with increasing m, but the oscillations does not decrease monotonically or linearly, instead the oscillations can become bigger when m increases, e.g., the oscillations when m=14 is bigger than some of those when m=10. Therefore, from the perspective of the smoothness/oscillations of the source sending rate, a bigger m is preferred.

We can also inspect the transient response from Fig. F.4a. The settling time Ts and the rise time Tp of the source are 6.6s and 4.8s respectively for m=4. Fig. F.4b to F4f show that the settling time and the rising time, like the link utilization in Fig. F.3, becomes longer when m increases. The settling time we have determined from Fig. F.4b to Fig. 4f are about 7.8s, 8.2s, 8,4s, 9.6s and 12.0s, respectively, which can be roughly approximated as Ts =0.3m+5.4. On the other hand, the rise times from Fig. F.4b to Fig. 4f are around 5.8s, 6.4s, 7.2s, 8.0s and 9.6s, respectively, which can be approximated as Tp=0.35m+3.5. Therefore, like the utilization in Fig. F.3, from the perspective of a shorter settling time and rise time, a smaller m is preferred.

As a summary from above observations, either a too small m or a too big m would not make the IntelRate controller work well if we would like to have both good transient response (i.e., setting time or rise time) and smooth source throughput. Instead, it should be a trade-off of the two performances. Considering such a trade-off, we would like to choose the middle, i.e., m=8.

Finally, recall that the IQSize is the only variable that the IntelRate controller relies on to do its control. It would be of interest to find out if IQSize performance gives any information about the determination of m. Note that we have shown from numerous previous experiments that the IQSize can always be controlled around the TBO. So the interests would be to see how the changes of m may affect the transient response and steady state of the queue size as shown below.

Figures F.5a to F.5f depict the IQSize performances under different m values. All of them show that after the transients, the IntelRate controller is able to well stabilize the queue size around TBO=4000 packets despite the value of m. However, the steady state error (i.e., the variations of queue size around the TBO) may vary. For example, among all figures from Figures F.5a to F.5f, the queue size with m=4 has the smallest steady state error (i.e., the

oscillations above or below the TBO), which are averagely about 400 packets. While the ones with m=20 have the biggest steady state errors, which are averagely 960 packets. The steady state errors of the queue size with m=6, m=8, m=10 and m=14 are in between, i.e., 430 packets, 500 packets, 620 packets and 780 packets. Similar to the transients (i.e., the settling time or rise time), the relationship between the steady state error and m may be approximated with 30m+280. With such an observation, it may allow us to draw a general conclusion that small m tends to give small steady state error.

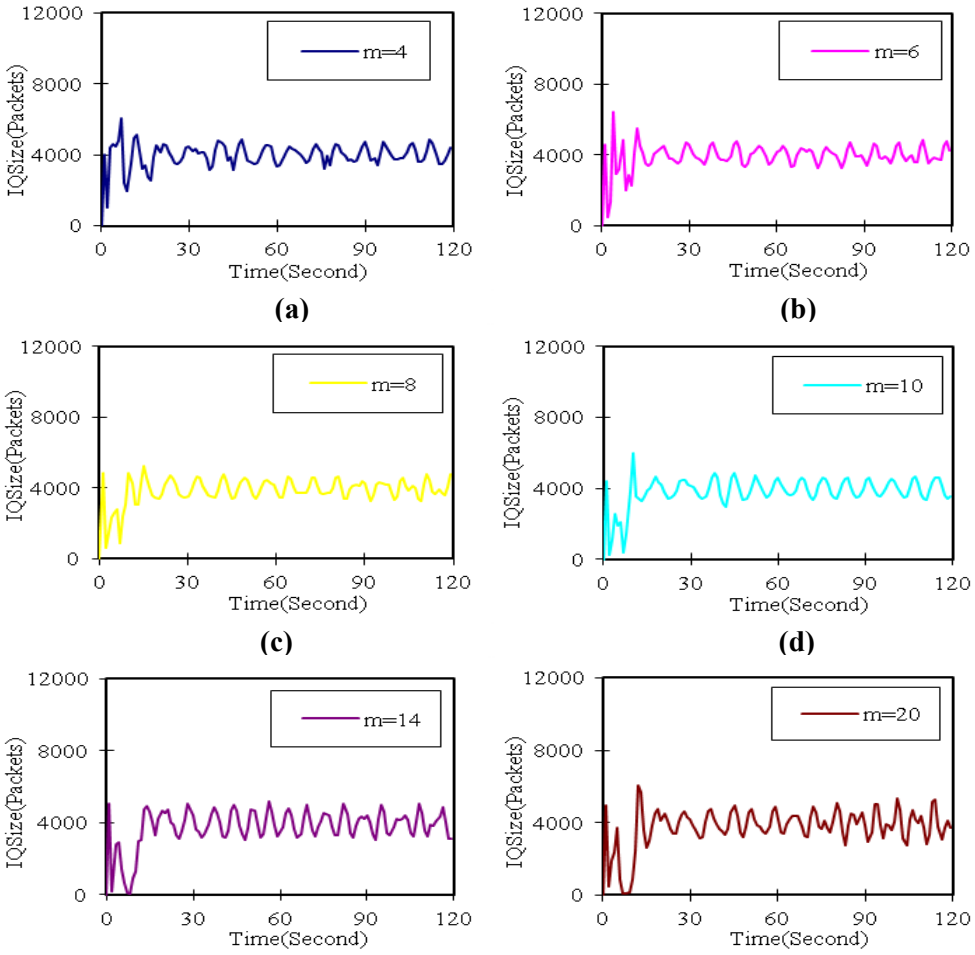


Fig. F.5: The IQSize Dynamics

Furthermore, qualitatively, the settling time or the rise time is also shorter with a small m than a big m in that the IntelRate controller spends more time in transient before the queue size reaches the TBO, as seen in Figures F.5a to Fig. F.5f.

We have also conducted similar experiments under different link bandwidth scenarios (e.g., 1Gbps, 1.62Gbps, 3Gbps and 4Gbps), and obtained very similar trends and therefore

the same conclusion. They are omitted here for brevity reason.

To summarize, a good queue size performance requires a small *m*. However, as discussed above, such a requirement also has to be compromised with the performance requirement of the source sending rate (which requires a big *m* for smoothness). The above decision that *m*=8 is also acceptable to the queue size performance, as shown in Fig. F.5c.

F.3 Parameter *N*

The parameter *N* is the number of LVs employed by the IntelRate controller. To determine this parameter, couple of experiments are conducted under different link bandwidth. In the first experiment, with the 1Gbps link, TBO is set to be 1000 packets. The desired rates of the sources from ftp group 1 to group 5 are 3.28Mbps, 5.74Mbps, 8.19Mbps, 16.36Mbps and 18.02Mbps, respectively. We test the IntelRate controller with *N*=5, 7, 9, 11, 13 and 15 and compare their transient response and steady state performances.

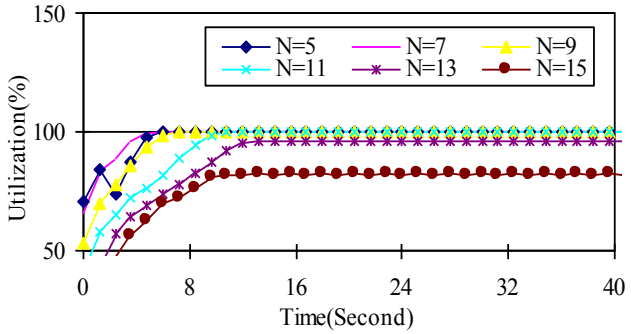


Fig. F.6: Link Utilization

The link utilization under different *N* values is shown in Fig. F.6. When *N*=5, the settling time *Ts* and the rise time *Tp* are 7.2s and 5.7s, respectively. We also notice that there is a "kink" during the rise time, which we suspect to be the queue size increasing too fast (also observed in Fig. F.8a to be discussed later) at the beginning and the controller therefore decreases the source sending rate a bit aggressively. Fig. F.6 shows that when *N* increases from 5 to 15, the rise time of the link utilization becomes longer because the controller spends more time before reaching the steady state. Specifically, the settling time from *N*=7 to 15 is separately 6.8s, 7.2s, 9.6s, 11.0s and 9.6s, Furthermore, when *N* increases to 13 or 15, the link can only reach a utilization of 95% or 82%, instead of 100%, which means the big *N* can cause link under-utilization problem. Therefore, the settling time for the last one may not

be good result since it is not based on 100% utilization or close to 100% utilization. Besides, as one can see from the settling times, the relationship between *Ts* and *N* cannot simply be linearly approximated. As for the rise time, when *N* increases from 7 to 15, they are separately 5.6s, 5.8s, 8.0s, 9.3s and 7.6s. The above comment for *Ts* can be similarly applied to *Tp* here, too, i.e., it seems a linear relationship cannot be used to describe *Tp* and *N*. However, the general trend of the transients shows that in order to have fast transient response and full link utilization, one would like to choose a small *N*.

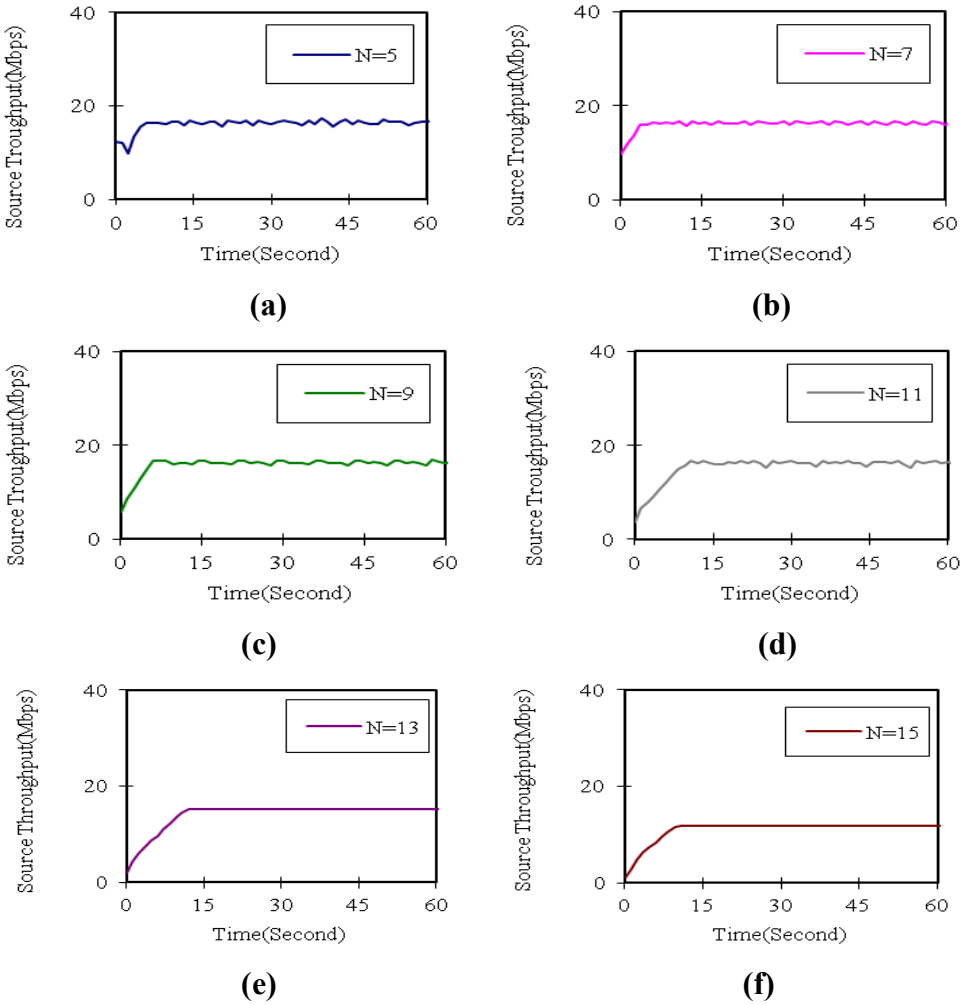


Fig. F.7: The Source Throughput Dynamics

The source throughput curves with different *N* values are depicted in Fig. F.7. On one hand, like the link utilization performance in Fig. F.6, the bigger *N* leads to longer settling time or rise time, as seen from Fig. F.7a-Fig. F.7f. Specifically, the settling times from *N*=5 to 15 are 7.2s, 8.4s, 9.0s, 10.2s, 11.5s and 9.6s, respectively. For Fig. F.7f, since the link is

under-utilized (as shown in Fig. F.7f), the sending rate is lower, too, as expected than others in Fig. F.7a-Fig. F.7e. On the other hand, although the smoothness shown from Fig. F.7a-Fig. F.7f is all acceptable in terms of the slight difference in their oscillations, it still indicates a big *N* gives better smoothness, which is evident when we compare Fig. F.7e or Fig. F.7f with Fig. F.7a to Fig. F.7d. However, the caution must be exercised when we prefer big *N* because big *N* complicates the controller in the sense that the controller has to perform more logic computations to produce a control command. Therefore, from the viewpoint of the source performance shown in Fig. F.7, *N* can be a value to trade off the transients and the smoothness. In this regards, we would like to choose *N*=9. Furthermore, the following queue size performance also renders such a requirement for the choice of *N*.

The queue size performance under different *N* is depicted in Fig. F.8a to Fig. F.8f. In Fig. F.8a, in the first 3.6s of transient stage, the queue size oscillations can be as high as 1600 packets. When the controller reaches the steady state, the steady state error is 980 packets on the average. When *N* becomes bigger, the oscillations become smaller as seen from Fig. F.8b to Fig. F.8f. In detail, the steady state error from Fig. F.8b- Fig. F.8d (i.e., *N*=7 to *N*=11) is separately 320 packets, 78 packets and 43 packets. For Fig. F.8e (*N*=13) and Fig. F.8f (*N*=15), the queue does not reach TBO (i.e., 1000 packets), instead it can only stay close to the empty position, which means a big *N* can affect the queue performance in terms of the ability to control IQSize to TBO. The reason is that the source sending rate decreases (e.g., as seen in Fig. F.7f) with a big *N*, so there is not enough traffic to support the queue level. In a nutshell, as chosen for *N* above, *N*=9 can also a good trade-off between the queue steady state error and the ability to control queue size to TBO.

The similar experiments under different link bandwidth scenarios (e.g., 5Gbps) show the same trend in their results as shown in Fig. F.6, Fig. F.7 and Fig. F.8. For brevity, they are not shown here.

As a summary to the controller performances, the choice of *N* (i.e., we choose *N*=9) is made as a trade-off by considering the link utilization performance, the source throughput performance, the queue size performance as well as the computation complexity.

F.4 Different Defuzzifications

To determine the defuzzfication method, we test the IntelRate controller under three most

commonly used defuzzification techniques, i.e., COG (Centre of Gravity), CA (Center Average) and MC (Max Criterion).

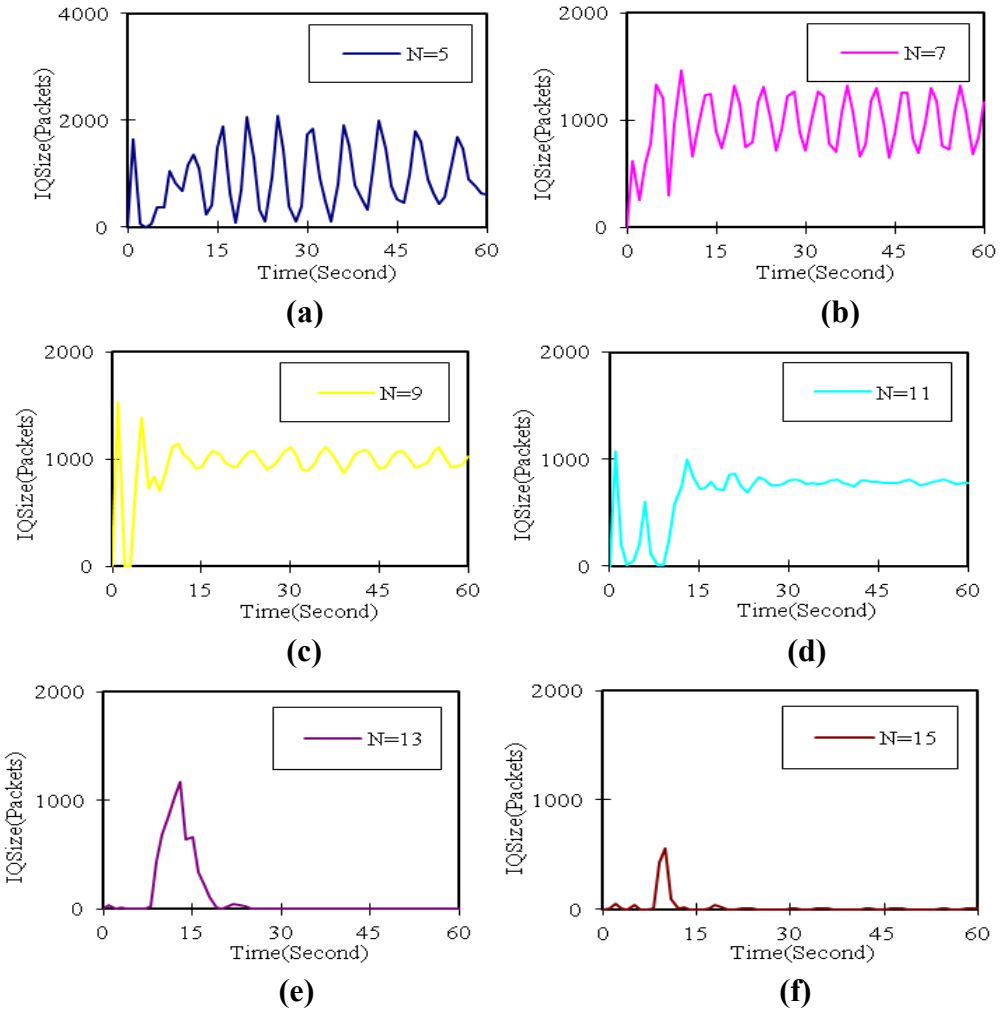


Fig. F.8: The IQSize Dynamics

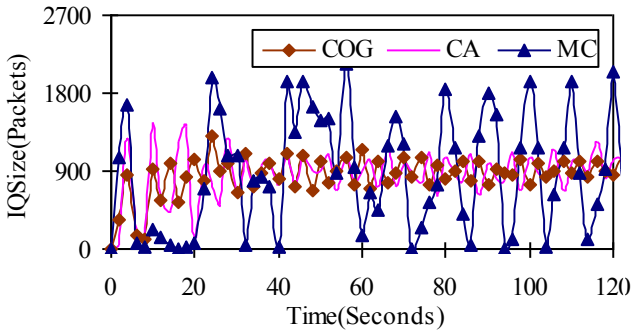


Fig. F.9: IQSize under Different Defuzzifications

In this experiment, the bottleneck has a bandwidth of 1Gbps, and TBO is set to 900

packets (as you may see throughout this document, TBO can even be set a bit differently for a same link bandwidth), m=8 and N=9.

Fig. F.9 depicts the dynamics of the IQSize under different defuzzification techniques, i.e., COG, CA and MC. For COG, the queue size rises close to 900 packets at the beginning when the router starts. After t=10s, the oscillation amplitude of the queue size decreases and the queue size is now operating around 900 packets. CA achieves similar queue size performance overall to COG, and the slight difference is that CA oscillates a bit harder in the first 20s than COG, i.e., its queue size rises to about 1500 packets. However, MC renders much worse performance in that its queue size cannot settle closely to the TBO of 900 packets, instead it oscillates between the bottom of the queue and about 2000 packets.

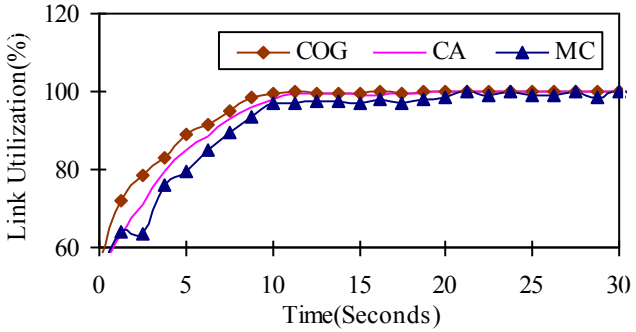


Fig. F.10: Link Utilization under Different Defuzzifications

Link utilization under different defuzzification techniques is illustrated in Fig. F.10. As seen, COG utilizes the link bandwidth the highest during the transient period, i.e., the first few seconds, and shows fastest speed (about 8s) reaching100%. Although CA has a slightly lower speed (about 12s) than COG, it also obtains 100% utilization. MC is the slowest one. Specifically, MC spends more than 20s reaching 100%. Besides, Fig. F.10 shows that MC is unable to stay well at 100% utilization level because of the occasional deviations from 100%.

From the above performance, obviously COG and CA are the better candidates for the IntelRate controller. To choose from these two, COG may be a better one in view of the weaker performance of CA in the transient period (as shown in Fig. F.9). This is why our IntelRate controller chose COG as the difuzzification method.

Discussion: The reason that COG and CA present similar performance while MC is worse may be attributed to the different ways they interpret the fuzzy set which affects the controller output. Specifically, COG or CA calculates their output based on average of all the

implied fuzzy sets, while MC does this based on the OVERALL implied fuzzy set. In other words, MC just considers a biggest point in the overall implied fuzzy set, instead of the average. Therefore, the output of MC may be too far from the optimized point and hence leading to oscillations. In contrast, the compromised output from COG or CA by averaging is not so fluctuating. As for the slight performance difference between COG and CA, COG considers the AVERAGED AREAS of the implied fuzzy sets while CA cares about the averaged BIGGEST CERTAINTIES of the implied fuzzy sets. So COG has a better performance because it involves more information than CA in the calculation of the controller output.

F.5 Effect of Short-lived Traffic

So far, our discussion above mainly focuses on the long-lived ftp flows. As shown in Fig. 3.6 in Chapter 3, short-lived http flows usually have sporadic arrivals (following a think-time) with a small transfer rate. Their existence is usually negligible compared to the rate of ftp flows which is in the order of Mbps in our experiments. So even in the worst case when 100 http flows are transferring data at the same time at a rate of 200kbps and for a duration time of 1ms, they only consume 2Mbps of the bandwidth in a split second. The consumption makes up only a 4.4% portion of a 45Mbps bottleneck or a 0.2% portion of a 1Gbps bottleneck. Hence they hardly influence the performance of the large ftp flows. Therefore, leaving them as the uncontrolled flows is usually justifiable for the analysis and operation of the IntelRate controller. However, it would be of interest to investigate the possibility of making the http flows controllable in case these short-lived flows would also transmit at a high data rate in a short time. So we would like to see how the IntelRate controller reacts to such kinds of sporadic arrivals.

We conduct an experiment under the heavy traffic condition, where 40 ftp flows and 160 http flows share a 600Mbps bottleneck. The 160 http flows will enter the traffic at t=20s, and exit at t=50s. Each flow arrives after a think-time uniformly distributed within the interval [0.1s, 20s], and the duration of each http flow is uniformly distributed in [0.1s, 10s]. In order to show the effect, we allow http flows in groups 2 and 3 to desire big sending rates. The source desired rates in ftp group 1 and 2 are 12.95Mbps and 26Mbps, respectively. The desired rates of the http groups 1 to 8 are 2.5Mbps, 12.95Mbps, 20Mbps, 5Mbps, 7.5Mbps,

1.3Mbps, 2.5Mbps and 3.25Mbps, respectively.

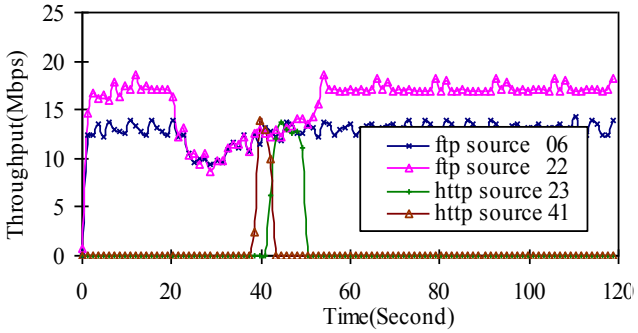


Fig. F.11: Flow Throughput Dynamics

Fig. F.11 shows the throughput of 4 sources, i.e. 2 ftp sources from each of the ftp groups 1 and 2, and 2 http sources from each of the http groups 2 and 3. In the first 20 seconds, the ftp flows are operating under max-min fairness. After the http flows join in the traffic from t=20s, the ftp flows begin to decrease their throughput to accommodate the big http flows. After these http flows finish their transmission, the ftp flows recover their original throughput. During this process, we notice that the http flows are also operating with the max-min fairness, e.g. the http source 41 which desires a higher sending rate to transmit its data with a speed lower than 20.00Mbps. However, the http source 23 which desires a lower sending rate is allowed to transmit its data with the desired rate of 12.95Mbps.

In summary, this experiment shows that the long-lived flows can accommodate the large short-lived http flows upon their arrivals. They simply regard the http flows as the long-lived ftp flows.

APPENDIX G: CONVEX OPTIMIZATION TERMINOLOGY

This appendix attempts to bring more information/explanation/clarifications needed for Chapter 4. A good portion of the content here is abstracted from [BoVa04] and [BeBa09]. For easy reading, here we use the same notations as introduced in Chapter 4.

G.1 Convex Function

Let **dom** f be the domain of a function f. Then the function f is convex if for all $x, y \in$ **dom** f, and θ with $0 \le \theta \le 1$, we have

$$f(\theta x + (1-\theta)y) \leqslant \theta f(x) + (1-\theta)f(y). \qquad \text{(G-1)}$$

Geometrically, this inequality means that the line segment between $(x, f(x))$ and $(y, f(y))$, which is the chord from x to y, lies above the graph of f (Fig. G.1). A function f is strictly convex if strict inequality holds in Eqn. (G-1) whenever $x \neq y$ and $0 < \theta < 1$. We say f is concave if $-f$ is convex, and strictly concave if $-f$ is strictly convex. Therefore a concave function can also yield to convex optimization.

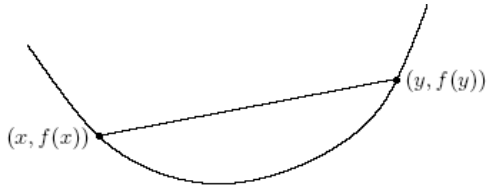


Fig. G.1: Graph of a convex function

G.2 Affine Functions

Affine functions represent the functions of the form

$$h_i(x) = a_i^T x - b_i \qquad \text{(G-2)}$$

The coefficient a_i^T and b_i can be a scalar or a vector as per whether x is a scalar or a vector.

G.3 Convex Optimization Problem

A convex optimization problem is one of the forms:

$$\begin{aligned} &\text{min} && f_0(x) \\ &\text{subject to} && f_i(x) \le 0,\ i = 1, \ldots, m \\ & && a_i^T x = b_i,\ i = 1, \ldots, p, \end{aligned} \qquad \text{(G-3)}$$

where $f_0, \ldots, f_m$ are convex functions. The convex problem has three additional

requirements:

• the objective function must be convex,

• the inequality constraint functions must be convex,

• if the equality constraint functions $h_i(x) = a_i^T x - b_i$ must be affine.

Note the wording "…one of the form…" in the definition (G-3), which can mean: (1) the notation "min" before the objective function $f_0(x)$ can be replaced by "max" as per the problem one is dealing with. The following definitions of Lagrangian dual function or Lagrangian dual problem is defined with respect to the "min" form in (G-3); (2) not all the problems require both of the two forms of constraint functions. Some problems even do not require any constraint, which case is called an unconstraint convex optimization problem.

In contrast to the dual problem to be defined later on, the convex problem (G-3) is called the primal problem.

G.4 The Lagrangian

The Lagrangian η associated with the problem (G-3) is defined as

$$\eta(x,\lambda,\nu) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x), \qquad \text{(G-4)}$$

λ_i is referred to as the Lagrange multiplier associated with the ith inequality constraint $f_i(x) \leq 0$; similarly ν_i is referred to as the Lagrange multiplier associated with the ith equality constraint $h_i(x)=0$. The vector form of the Lagrange multiplier is separately designated with λ and ν , which also called the dual variables.

G.5 The Lagrangian Dual Function

As per the "min" form of (G-3), the Lagrange dual function (or just dual function) is defined associated with (G-4) as,

$$\Omega(\lambda,\nu) = \min_x \eta(x,\lambda,\nu) = \min_x \left(f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x) \right) \qquad \text{(G-5)}$$

Note: if (G-3) has a "max" form, (G-5) will be in the "max" form, too.

G.6 The Lagrangian Dual Problem

As per the "min" form of (G-3), the Lagrange dual problem is defined associated with (G-5)

as,

$$\begin{aligned} &\max && \Omega(\lambda,\nu) \\ &\text{subject to} && \lambda \geq 0 \end{aligned} \qquad \text{(G-6)}$$

Note: if (G-3) has a "max" form, (G-6) is to "min" the objective function $\Omega(\lambda,\nu)$.

G.7 The Duality Gap

The gap between primal and dual objectives, i.e., $f_0(x) - \Omega(\lambda,\nu)$, is defined as the duality gap.

G.8 The Optimal Duality Gap

Assume the optimal value of the primal problem in (G-3) is f_0^*, and the optimal value of the dual problem in (G-5) is Ω^*, then the difference $f_0^* - \Omega^*$ is the optimal duality gap.

G.9 Strong Duality

If the equality

$$f_0^* = \Omega^* \qquad \text{(G-6)}$$

holds, i.e., the optimal duality gap is zero, then we say that strong duality holds.

When the strong duality holds, since the optimal values of the primal problem (e.g., **PP** in Chapter 4**)** and the dual problem (e.g., **DP** in Chapter 4) are the same, solving the dual problem also solves the primal problem.

G.10 Slater's Condition

If there exists an x in the domain of the defined convex optimization problem such that $f_i(x) \leq 0,\ i = 1, \ldots, m,\ a_i^T x = b_i,\ i = 1, \ldots, p,$ then we say the Slater's condition is satisfied.

G.11 The KKT Optimality Conditions for Convex Problems

KKT (Karsh-Kuhn-Tucker) optimality conditions refer to the series of constraints attached to the primal and dual problems of the convex optimization, and they are

$$f_i(x) \leq 0, \quad i = 1, \ldots, m \qquad \text{(G-7.1)}$$

$$h_i(x) = 0, \quad i = 1, \ldots, p \qquad \text{(G-7.2)}$$

$$\lambda_i \geq 0, \quad i = 1, \ldots, m \qquad \text{(G-7.3)}$$

$$\lambda_i f_i(x) = 0, \ i = 1, \ldots, m \qquad \text{(G-7.4)}$$

$$\nabla f_0(x) + \sum_{i=1}^{m} \lambda_i \nabla f_i(x) + \sum_{i=1}^{p} v_i \nabla h_i(x)) = 0 \quad . \tag{G-7.5}$$

where λ_i and v_i have been introduced in Section G.4.

G.12 The Lagrange Mean Value Theorem

If f is continuously differentiable over an open interval I, then for every $x, y \in I$, there exists some $\xi \in [x,y]$ such that

$$f(y) - f(x) = \nabla f(\xi)(y - x) \tag{G-8}$$

G.13 The Lipschitz Continuity Condition

A condition of the form

$$\|\nabla f(x) - \nabla f(y)\| \le L\|x - y\|, \qquad \forall x, y \tag{G-8}$$

is called a Lipschitz continuity condition on $\nabla f(x)$ or on the curvature of $f(x)$, and L is a positive constant (i.e., $L>0$).

G.14 The Descent Lemma

Let f be continuously differentiable, and let x and y be two vectors. Suppose that

$$\|\nabla f(x + ty) - \nabla f(x)\| \le Lt\|y\| \tag{G-8}$$

where L is some scalar. Then

$$f(x + y) \le f(x) + y'\nabla f(s) + L\|y\|^2 / 2 \tag{G-8}$$

APPENDIX H: INTRODUCTION TO REVENUE MANAGEMENT

The operation of the OFEX controller can be economically interpreted with the RM (Revenue Management), as discussed in Section 4.2 of Chapter 4. This appendix introduces the basic principle of the RM, which is about the strategic and tactical price some firms take in order to optimize revenues and profits [YeMc11].

RM has been a large success in airlines, hotels and other companies and is nowadays considered to be a key component of capacity management in many industries. For better understanding, we can look at the airlines below [Mich07] as an example.

People will have noticed that the same seat on the same aircraft is sold for different prices. These differences can be quite large. For example, Lufthansa German airlines sells flights between Dresden and Frankfurt/Main for €109 (return) – a special discount of very limited availability. There is no such thing like a single "regular" price for that route to compare with, but the general agreement is that a "usual" fare (i. e. a fare that is not part of a special discount offer) is well above €200. Fares between €300 and €400 are also not extraordinary because passengers have to pay more than €450 for some travel dates. For this (arbitrarily chosen) example the premium for "regular" tickets compared to the discount is in the order of 100 to 400 %. It is important to stress that we are not talking about different prices for economy, business and first class – all the prices mentioned above are for a single seat in the economy class compartment.

The fact that the same seat on the same aircraft is sold for various prices at the same time implies some challenging decision problems: On one hand, it is obviously reasonable to sell seats at the highest possible prices. Demand is stochastic, though, and the bulk of passengers with a high willingness to pay (e. g. business travelers) will typically book close to departure, while other consumers who cannot afford the highest prices will submit reservation requests very early. On the other hand, a seat that is empty at the time of departure represents opportunity costs, because it may have been sold to a paying customer; and even if the fare received was low, the contribution margin would have been positive because the marginal costs of carrying an additional passenger are negligible. Given a request of a passenger with a low yield the airline thus has to decide whether to accept it (running the risk of displacing subsequently arriving demand with higher revenue) or to

reject it – which is a bad decision if not enough high yield requests arrive in the future. In general, the question arises how the given capacity should be assigned to products (i. e. fares and passengers) such that the total revenue (profit, contribution margin, . . .) is maximized. Aspects related to that general question are subsumed under the term RM.

Going back to our OFEX controller, the relationship of NUM to RM is that a Lagrange multiplier (i.e., λ_l , see table of notations) is born after the primal problem formulated with the NUM theory is converted to a dual problem. Accidentally, the Lagrange multiplier can be regarded as a unit bandwidth price of routers, which can go up and down according to the traffic intensity, just like the price of plane tickets where RM applies.

Appendix I: The OFEX Computational Complexity

Similar to the IntelRate controller, to relieve the concern on the computational complexity of the OFEX controller in a router designed in Chapter 4, this appendix analyzes and compares the implementation complexity of the OFEX controller and its counterpart, i.e., the relatively explicit controller. We point out that they have the same complexity order. Examples can be found from simulation results in Section 4.6 and discussions in the design guideline of Sections 6.2.

I.1 Computation Complexity of the OFEX Controller

As demonstrated in Fig. 4.2, the algorithm of the OFEX controller resides in each router, and eventually a source adjusts its sending rate according to the most congested router along its path. A careful inspection of the algorithm in Fig. 4.2 shows that there are two levels of operations: 1) packet-level when Step (3) is executed upon each packet arrival or departure to record the allowed sending rate to the passing flows; 2) cycle-level when Step (4) is executed periodically to reflect the current traffic level in a router.

On packet level, Step 3A involves 1 addition, and Step 3B involves 1 division and 1 comparison. Therefore, there are three operations on the packet level in order to receive and deliver a packet out of a router. On cycle-level, Step 4.1 involves 1 division, and Step 4.3 involves 2 additions, 1 subtraction and 2 multiplications. Hence, the total number of operations on the cycle level consists of 2 additions, 1 subtraction, 2 multiplications and 1 division. Finally, since the initialization step is executed once (i.e., when the router starts), this step can be ignored in the complexity analysis.

In the following, we convert all the operations into machine clock cycles. If the algorithm is implemented in a router with the Intel 64 and IA-32 type CPU [Intel12], the time to implement a floating point addition, subtraction, multiplication, division, and a comparison instruction would be about 3, 3, 5, 20 and 3 clock cycles, respectively [Intel12]. Therefore, totally the OFEX controller requires 39 clock cycles in the cycle level and 26 clock cycles in the packet level for one implementation.

To facilitate the comparison later on, we map the complexity into the cycle level and represent it with the big O notation. In one cycle, the complexity is $39+T*R*26$, where T is the operation interval mentioned in Fig. 4.2 and R is the link bandwidth with the unit of

packets/second. For $T*R$ with hundreds of thousands of packets in a router every T, $39+T*R*26 \approx T*R*26$. Therefore, the computation complexity of the OFEX controller is in an order of $O(T*R)$.

I.2 Computational Complexity of a Relatively Explicit Controller

Unlike our OFEX algorithm which is executed in the routers only, a relatively explicit controller has to operate its algorithm in both its source and the routers. Its router-algorithm computes the traffic level, just like the cycle-level operation in the OFEX controller. Its source-algorithm derives the allowed sending rate according to the cumulative link price and a demand function chosen by the source. A typical router-algorithm and a source-side algorithm (e.g., those from [AtLo00]) can be respectively summarized in Fig. I.1 and Fig. I.2 below.

---

(1) Initialize the parameter γ, the step size δ, the Lagrangian multiplier $\lambda_l^{(0)}$ in link $l, \forall l \in L$, and set $W^{(k)}=0$.

(2) Do forever the following steps (3) to (4)

(3) On a packet arrival, e.g., for the *k*th packet arrival, *k=1, 2, 3* ….

3A1) Obtain the packet size *r* of the arrived packet.

3A2) Update $W^{(k)}$ via $W^{(k+1)} = W^{(k)} + r$

(4) At fixed intervals T, do the following

4.1) Compute the aggregate incoming rate $W^{(k)} / T$.

4.2) Set $W^{(k)} = 0$

4.3) Update the Lagrangian multiplier λ_l with the following equation except λ_l is now $\lambda_l^{(k+1)}$.

$$\lambda_l^{(k+1)} = \left[\lambda_l^{(k)} + \delta \cdot H_{ll}^{-1} \left(\sum_{s \in S_l(t)} x_s^l - c_l \right) \right]^+ \qquad \text{(I-1)}$$

where H is a strictly positive definite diagonal matrix in which the diagonal element $H_{ll} = \max\{\varepsilon > 0, -\frac{\sum x_s^{l^{(k)}} - \sum x_s^{l^{(k-1)}}}{\lambda_l^{(k)} - \lambda_l^{(k-1)}}\}$, and other parameters have the same notions as (4-9) of the OFEX controller.

4.4) Communicate the new price $\lambda_l^{(k+1)}$ to all sources that use link *l*.

Fig. I.1: The Router-side Algorithm for Link *l*

With iterations *k=1,2,3, …..,* the algorithm in source *s* periodically repeats the following two steps every *T* seconds:

(1) Compute a new source rate

$$u_s(k+1) = [U_s'^{-1}(\lambda^s)]_-^+ \qquad \text{(I-2)}$$

where $U_s(.) = \omega_s \log(1+u_s)$ is the demand function, λ^s is the cumulative link price in the path of source *s*, and $[z]_-^+$ refers to the function z that is truncated by its upper bound and lower bound.

(2) Communicate the new rate x_s to all links in its path.

Fig. I.2: The Source-side Algorithm in Source *s*

Since the authors [AtLo00] did not mention how to communicate the new link price $\lambda_l^{(t+1)}$ to its sources, we assume there is a dedicated field for the cumulative link price in the header of each packet. So when a packet passes a link, the value of this cumulative price is incremented by the link price. When the packet eventually arrives at the destination, the destination could send the cumulative price, which reflects the total congestion along the path, back to the source in an ACK packet. The updated router-side algorithm for the relatively explicit controller is shown in Fig. I.3.

(1) Initialize the parameter γ, the step size δ, the Lagrangian multiplier $\lambda_l^{(0)}$ in link *l*, $\forall l \in L$, and set $W^{(k)}=0$.

(2) Do forever the following steps (3) to (4)

(3) Perform A or B depending on a packet arrival or a departure.

A) For the *kth* packet arrival, *k*=1, 2, 3 ….

3A1) Obtain the packet size *r* of the arrived packet.

3A2) Update $W^{(k)}$ via $W^{(k+1)} = W^{(k)} + r$

B) For the *jth* Packet Departure, add λ_l up to the existing cumulative price of the packet header except λ_l is now $\lambda_l^{(J)}$, and send it out.

(4) At fixed intervals T, do the following

4.1) Compute the aggregate incoming rate $W^{(k)} / T$.

4.2) Set $W^{(k)} = 0$

4.3) Update the Lagrangian multiplier λ_l with the following equation except λ_l is now $\lambda_l^{(k+1)}$.

$$\lambda_l^{(k+1)} = \left[\lambda_l^{(k)} + \delta \cdot H_{ll}^{-1} (\sum_{s \in S_l(t)} u_s^l - c_l) \right]^+ \quad \text{(I-3)}$$

where H is a strictly positive definite diagonal matrix in which the diagonal element $H_{ll} = \max\{\varepsilon > 0, -\frac{\sum u_s^{l^{(k)}} - \sum u_s^{l^{(k-1)}}}{\lambda_l^{(k)} - \lambda_l^{(k-1)}}\}$, and other parameters have the same notions as (3-13) of the OFEX controller.

Fig. I.3: The Updated Router-side Algorithm in Link *l*

From Fig. I.3, the relatively explicit controller also involves two levels of operations: packet level in Step 3 and cycle level in Step 4. Step 3A or Step 3B requires 1 addition, respectively, and thus the packet level operation has 2 additions in total. Step 4.1 needs 1 division, and Step 4.3 needs 1 addition, 3 subtractions, and 2 multiplications. So the cycle level operations in each period T require 1 addition, 3 subtractions, 2 multiplications and 1 division.

With the same Intel 64 and IA-32 type CPU, the complexity of the relatively explicit controller is 42 clock cycles in the cycle level and 6 clock cycles in the packet level. Similar to the OFEX controller, in one cycle, the complexity of the relatively explicit controller can be written as 42+T*R*6 (can be approximated as T*R*6 for hundreds of thousands of packets in a router every T), which is in an order of O(T*R).

This is our best knowledge to implement a relatively explicit algorithm in a simplest way. An alternative way may be to broadcast the link price to the millions of sources in a network, which we believe would produce too much overhead (or say, traffic) and worsen the network congestion.

I.3 Complexity Comparison

Since the computational complexity on the router side is the concern for the explicit congestion controllers, in the following comparison the discussion would focus on the

algorithms illustrated in Fig. 4.2 and Fig. I.3.

As for the algorithm complexity on the source side, it is usually assumed that the sources have enough computation resources because the end users usually have much less tasks than a router which also at least deals with routing. If we do want to make the source-side computation comparison, as obviously illustrated in Fig. I.2, the relatively explicit controller puts more computational burden on the sources than the OFEX controller because the latter has no computational requirement on the sources at all.

By comparing the operations we summarized in Section I.2, the OFEX controller requires 3 computer clock cycles (or equivalently say, 1 addition) less in the cycle level while 20 clock cycles (or equivalently say, 1 division) more in the packet level when the router uses an Intel 64 and IA-32 type CPU. The quantitative comparison in the cycle level is shown in Table I.1. However, they have the same order of complexity, i.e., O(T*R).

Table I.1: Cycle Level Complexity Comparison (in clock cycles)

Cycle	OFEX controller	Relatively explicit controller
One interval T	T*R*26	T*R*6
Per Packet (Normalized by T*R)	26	6

Quantitatively, for a 3.40GHz CPU in a router, the execution time of the OFEX controller per packet as shown in Table I.1 is about 7.65ns (i.e., 26/3.4G=7.65ns).

I.4 Concluding Remarks

The computational complexity comparison of the two controllers in Section I.3 shows that the OFEX controller has the same order of computation complexity as the relatively explicit controller. In the packet level, the OFEX controller just equivalently requires one division more computation. Regarding the implementation of the OFEX controller in OPNET, it will be provided upon request to Prof Yang.

APPENDIX J: THE OFEX CONVERGENCE EXPERIMENTS

As a complement to the experiments done in Section 4.6 of Chapter 4, this appendix shows: 1) the convergence speed of the OFEX controller under different high bottleneck bandwidth; 2) how the initial Lagrange multiplier $\lambda_l^{(0)}$ or the step size δ affects the convergence speed of the OFEX controller.

For easy interpretation later on, we examine the starting transient of the OFEX controller in the first x seconds (i.e. the interval [0, x]). Our numerous experiments show that if convergence is fast under the interval [0, x], it is also fast when reacting to the network parameter changes in the "hot state", and vice versa. In another word, the convergence rate in the "hot state" can be reflected in the convergence analysis from the "cold start".

Table J.1: Source Payment, Link Bandwidth and Buffer Size

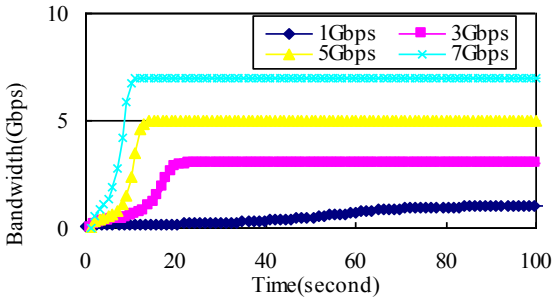
Exp. No.	ψ_s (from ftp group 1 to 5)	Bandwidth (Gbps)	Buffer (packets)
1	12207.0, 48828.8, 9765.6, 7324.2, 2441.4	1	24000
2	24414.1, 9765.6, 19531.24, 14648.4, 48828.8	3	72000
3	36621.1, 14648.4, 24414.1, 19531.24, 9765.6	5	120000
4	61035.2, 36621.1, 48828.1, 42724.6, 24414.1	7	182000

J.1 Convergence Speed under Different Bottleneck Bandwidth

To show the convergence speed of the OFEX controller under different high bottleneck bandwidths, we do a set of experiments with $\lambda_l^{(0)} = 50$ and $\delta = 0.01$. Each experiment differs in the bandwidth of the outgoing link, and accordingly the assigned flow weights and the buffer space.

The source payments ψ_s in each experiment are such that they can produce congestion in the link (i.e., the link can be fully utilized at 100%) after some trial-and-error experiments. As seen in the second column of Table J.1, the price that the flows in each group pay increases as the link bandwidth increases (see the third column of Table J.1) from Exp.1 to Exp.4. For example, the price that the flows in group 1 pay is separately 12207.0, 24414.1, 36621.1 and 61035.2 from Exp.1 to Exp.4. Thus, from Exp.1 to Exp.4, the flows are going to

obtain more and more share of the bottleneck bandwidth at congestion from which we investigate the convergence.



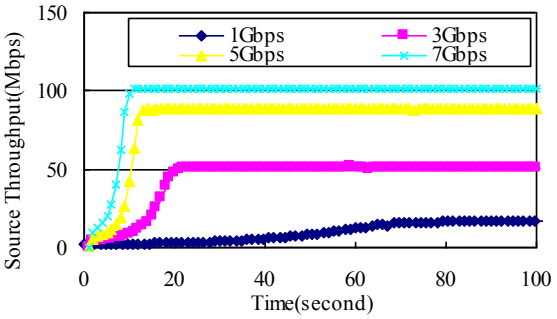


Fig. J.1: Link Bandwidth Convergence

Fig. J.2: Source Convergence (ftp flow 5)

Fig. J.1 shows the bandwidth convergence speed of different links. The link with the bandwidth of 7Gbps has its throughput increasing exponentially and achieves full utilization after an initial transient of about 12 seconds. Links with lower bandwidth have similar characteristics but have longer convergence times. Specifically, a convergence time of about 80s, 20s and 14s can be separately observed for the link of 1Gbps, 3Gbps and 5Gbps.

The source rate convergence speed in Fig. J.2 shows a trend very similar to the bandwidth convergence in Fig. J.1. That is, the convergence time of the 1Gbps, 3Gbps, 5Gbps and 7Gbps are 80s, 20s, 14s and 12s, respectively. In a matter of fact, the source convergence speed can also be used to explain the bottleneck convergence speed, because the bottleneck capacity must be used up slowly if the sources increase their sending rate with a lower convergence speed.

In summary, we can make two observations via the performance illustrated in Fig. J.1 and Fig. J.2). The OFEX controller may potentially converge faster at congestion under high-speed networks. 2) One fixed step size may already work for a range of bottleneck bandwidth. For example, a step size of $\delta = 0.01$ gives similar convergence time when the router bandwidth increases from 3Gbps to 7Gbps. This is a potential advantage of the OFEX controller because once a router is in place and its nominal link bandwidth is known, a step size may work well independent of the variations of link bandwidth around the nominal value. Of course, one fixed value of step size may not be suitable to any bandwidth, e.g., the convergence time of $\delta = 0.01$ with 1Gbps seems too long. More discussion will be provided later in Section J.3 when we study the effect of varying δ.

J.2 Convergence under Different Initial Value $\lambda_l^{(0)}$

Since we need to give an initial value of $\lambda_l^{(0)}$ to make our algorithm start to work (please see algorithm Fig. 4.2), we are interested to investigate the impact of $\lambda_l^{(0)}$ on the convergence time/speed of the OFEX controller. To do so, we choose different $\lambda_l^{(0)}$ values of 0.1, 1, 50 and 160 in our following experiments while keeping other settings the same as the 1Gbps experiment in Section J.1.

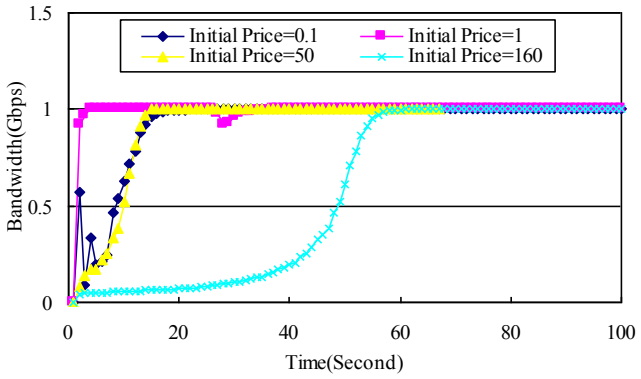


Fig J.3: Link Convergence

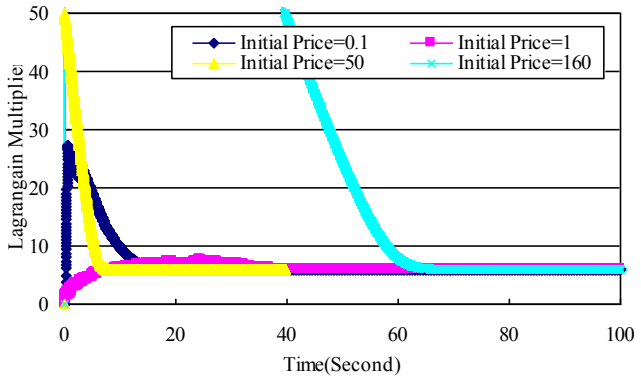


Fig. J.4: Link Price Convergence

Fig. J.3 depicts the link utilization convergence performance under different initial Lagrangian multipliers $\lambda_l^{(0)}$. As seen, the controller with $\lambda_l^{(0)}$=160 has the longest convergence time (~57 s) among the 4 traces. The utilized bandwidth is increasing linearly in the first 38s followed by an exponential increase to the full bandwidth, i.e., 1Gbps. The trace with $\lambda_l^{(0)}$=1 has the shortest convergence time, as seen. It sharply reaches the full link

capacity in 3s. In between we find the traces of $\lambda_l^{(0)}$=0.1 and $\lambda_l^{(0)}$=50 have almost the same convergence time. Careful inspection shows that $\lambda_l^{(0)}$=50 converges slightly faster with no oscillation in the first 5s.

In order to understand the mechanism contributing to different convergence time, we show the time evolution of the Lagrange multipliers in Fig. J.4. It is interesting to see that the Lagrange multiplier always converges to the same steady-state (i.e., the optimal value $\lambda_l^* = 6$) independent of the initial value $\lambda_l^{(0)}$. However, they all have different convergence time. Furthermore, the Lagrangian multiplier with $\lambda_l^{(0)}$=160 takes the longest, which explains why its link utilization takes the longest to converge in Fig. J.3. Similar explanation can be made to the convergence time of link utilization under different initial values although their transient shapes may be different. In general, if the initial value of step size is bigger than the steady state value, the curve is decreasing to the steady state (e.g., the curve with initial value=50 in Fig. J.4). Otherwise, the curve is increasing to the steady state (e.g., the curve with initial value=1 in Fig. J.4).

J.3 Convergence Speed under Different Step Size δ

The experiments are conducted under the same bottleneck as Section J.2 but with different step sizes of δ=0.50, 0.10, 0.07, 0.03 and 0.01, respectively.

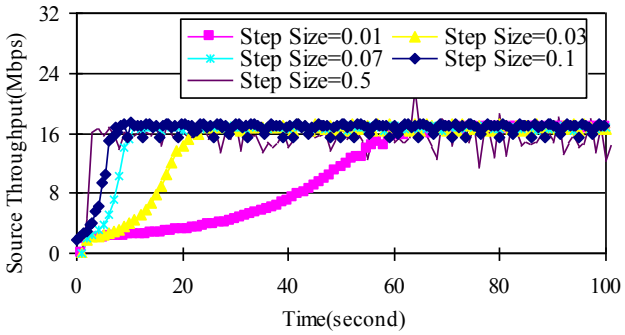


Fig. J.5: Link Convergence

Fig. J.5 shows that all the traces have a geometric convergence rate in the [0.x] interval as depicted by the exponential growth curve towards the equilibrium (or say, steady state). The basic observation is that convergence time increases as δ decreases. However, the source throughput (sending rate) becomes smoother at steady state as δ decreases. For example,

the trace with the biggest step size of δ=0.50 has a fluctuation of ±6 Mbps around the mean value of 16.8 Mbps. This fluctuation decreases to ±1 Mbps for the smaller step size of δ=0.1. The trace under δ=0.01 has the smoothest steady state after convergence. Such an observation indicates that the choice of the step size should be a performance trade-off between convergence speed and source sending rate smoothness because fast convergence speed gives rise to larger fluctuations after convergence.

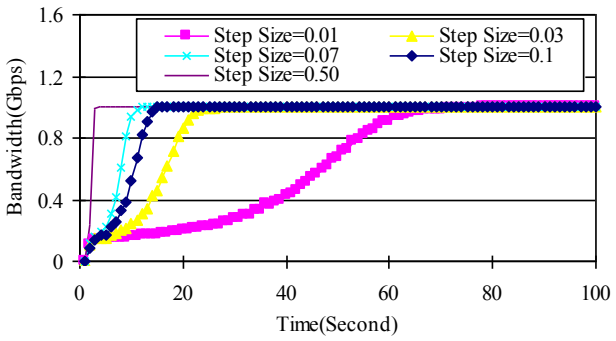


Fig. J.6: Source Convergence

Fig. J.6 shows similar trend as in Fig. J.5, i.e., all the traces have an exponential convergence but with different convergence time. The trace of δ=0.5 has the fastest convergence speed because it sharply rises to 1Gbps in 3s after the controller starts. In general, the convergence time is increasing with decreasing δ. As shown, a convergence time of about 6s, 9s, 22s and 60s can be separately observed as δ decreases from 0.10, 0.07, 0.03 to 0.01, hence confirming our intuition.

Based on the results from Fig. J.5 and Fig. J.6, a step size of δ=0.1 or 0.07 can be a good choice for a 1Gbps of bottleneck because its convergence time and smoothness are both reasonable.