A RATE-BASED CONGESTION CONTROL OVERLAY SYSTEM

by

Lianjie Cao
A Thesis
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
The Requirements for the Degree
of
Master of Science
Computer Engineering

Committee:

_____	Dr. Brian L. Mark, Dissertation Director
_____	Dr. Qiliang Li, Committee Member
_____	Dr. Andre Manitus, Committee Member
_____	Dr. Andre Manitus, Department Chair
_____	Dr. Lloyd J. Griffiths, Dean, Volgenau School of Engineering
Date: 7/29/11	Summer Semester 2011 George Mason University Fairfax, VA

A Rate-based Congestion Control Overlay System

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at George Mason University

By

Lianjie Cao
Bachelor of Engineering
Huazhong University of Science and Technology, 2008

Director: Dr. Brian L. Mark, Professor
Department of Electrical and Computer Engineering

Summer Semester 2011
George Mason University
Fairfax, VA

Dedication

This is dedicated to my parents and my girlfriend.

Acknowledgement

I would like to express my sincere gratitude to my advisor Dr. Brian L. Mark. Without his patient guidance and support, none of this would have been achieved. I also would like to thank my parents for their encouragement and trust. This work was support in part by DARPA Contract N6600-05-9-8904 (Internet Control Plane).

Table of Contents

List of Tables

List of Figures

Abstract

A RATE-BASED CONGESTION CONTROL OVERLAY SYSTEM

Lianjie Cao, M.S.

George Mason University, 2011

Thesis Director: Dr. Brian L. Mark

Traditional TCP/IP networks provide a best effort delivery service which places the complexity of congestion control in the end hosts, leaving the network relatively simple. It is well-known that the performance of the TCP congestion control scheme degrades severely under conditions of large bandwidth-delay products and/or high loss rates. As the traffic load on the Internet increases, overall network performance and the quality-of-service (QoS) experienced by individual users will degrade. To address this problem, a rate-based congestion control system operating as an overlay network is studied. The nodes in the overlay network provide congestion control for the overlay links, which correspond to physical network paths. By implementing congestion control as an overlay, network congestion can be alleviated without significantly increasing the complexity of the network.

In this thesis, a bandwidth probe control (BPC) system is designed for an overlay link, which estimates the delay and loss characteristics of the overlay link by probing and then

uses this information dynamically to determine an appropriate transmission rate for the link. A rate control algorithm is proposed for this purpose, based on a simplified fuzzy logic controller. In contrast, the legacy TCP/IP congestion control is based on an additive-increase, multiplicative-decrease (AIMD) control scheme triggered by packet loss timeouts. The proposed rate control scheme can achieve much higher utilization and more stable performance than TCP/IP congestion control, which is especially crucial for the rate-based congestion control overlay. The BPC system also provides reasonable fairness to cross-traffic which may traverse a portion of the overlay link, thus sharing the link capacity.

The BPC system was originally developed as an overlay version of the Software Adaptive Flow-Intelligent RoutEr (SAFIRE) which was developed as part of the Control for High-throughput Adaptive Resilient Transport (CHART) project, led by Hewlett-Packard (HP) Laboratories and sponsored by the DARPA Internet Control Plane program.

Chapter 1: Introduction

1.1 Background

The TCP/IP network protocol suite, which is the foundation of current Internet, was originally designed to provide a single level of service (i.e., best-effort). However, as the Internet continues to thrive, the TCP/IP protocol suite is facing the challenge of quality-of-service (QoS) and performance problems. Since the intermediate nodes merely provide “best effort” service to transfer packets from one node to another independently, they are blind to any traffic control information. The current protocols are sufficient for most applications which transfer simple files or messages. However, more and more applications for which “best-effort” service is simply insufficient are emerging. For such applications, quality-of-service is necessary to guarantee a certain level of performance to data flow.

To address this problem, the Control for High-throughput Adaptive Resilient Transport (CHART) [1] project, which was a 42-month effort of the team led by HP Labs, designed an intelligent control plane to the current Internet. The CHART system achieved performance improvements by redesign the part of Internet Layer 3 and Layer 4 protocols. The Software Adaptive Flow-Intelligent RoutEr (SAFIRE) [2] is a flow-aware software router designed at George Mason University within the Network Architecture

and Performance Laboratory as part of CHART system. The focus of SAFIRE is on dynamic flow routing of traffic in conjunction with a QoS signaling protocol. SAFIRE supports TCP explicit rate feedback (TCP-ER) defined in TIA 1039 [3], a QoS signaling protocol proposed by Dr. Lawrence G. Roberts of Anagran. SAFIRE can store the traffic information in terms of flows rather than packets and adaptively forward packets based on the stored flow information. This type of traffic is called flow routing.

1.2 Problem statement and main contributions

In order to realize a fully operational CHART system, SAFIRE or the hardware flow routers designed by Anagran need to be deployed on each node of the network, which is costly and time-consuming. An alternative approach to avoid this huge expense is to deploy SAFIRE as an overlay application for which the system only need to be installed on several major intermediate nodes to establish an overlay network on top of the legacy IP network. Nevertheless, the QoS signaling protocol needs to be aware of the link characteristics in order to allocate network resource to provide the desired QoS, which is a difficult task for current network overlay systems. Moreover, the actual available link capacity varies when part of the overlay link is traversed by cross-traffic. Also, a certain level of fairness should be provided to non-QoS traffic such as legacy TCP and UDP flows.

To address those problems, this thesis proposes a new overlay system based on the SAFIRE implementation. The major functions of the new overlay system are to estimate

the link characteristics such as available bandwidth, delay and loss rate and to control the congestion of the traffic traveling through the overlay link at an optimal level. The goal of this work is to develop a rate-based congestion control algorithm for a network overlay system that can provide satisfactory and stable link utilization, robustness to random bit corruption, convergence to optimal efficiency and backward compatibility without introducing huge upgrading expense.

In this thesis, we developed the Bandwidth Probe Control (BPC) system to achieve this goal. The BPC system is a rate-based congestion control overlay system which estimates the delay and loss characteristics of the overlay link by a bandwidth probing technique and then determines an appropriate rate using the information collected. Unlike the additive-increase, multiplicative-decrease (AIMD) congestion control algorithm of TCP/IP, the BPC system uses single-trip delay measurements instead of packet loss timeouts to evaluate the congestion level. A simplified fuzzy logic controller is developed to compute a proper sending rate. Hence, the BPC system is able to achieve a higher utilization of link capacity and much better stability than current TCP/IP protocols. By using delay information as a multi-bit feedback, the system can avoid drops in utilization by random bit errors.

Although the BPC system was originally designed as an overlay extension of SAFIRE for the CHART system, it turns out that the BPC system can be applied independently of SAFIRE and/or CHART. Since it does not distinguish the nature of incoming traffic, the BPC system can improve the performance of all types of traffic.

1.3 Organization

The remainder of this thesis is organized as follow. Chapter 2 provides a survey of current mainstream congestion control algorithms. Chapter 3 discusses the implementation issues and experimental setup of the BPC system. In Chapter 4, we explore the feasibility of using single trip delay as the indication of network congestion. Chapter 5 discusses the rationale behind BPC system in details and analyses the experimental results. The thesis is concluded in Chapter 6.

Chapter 2: Review of Congestion Control Techniques

The first network congestion collapse of the Internet occurred in 1986 when the throughput of a communication link between Lawrence Livermore Laboratory and University of California, Berkeley dropped from 32 kbps to 40bps. Since then, the network congestion problem has been studied extensively. Network congestion occurs when a link or node carries so much data traffic that the quality-of-service deteriorates. Congestion may result in queueing delay, packet loss or even network collapse. Although the capacity of today's Internet is continuously increasing, network congestion is still one of the major concerns because of the faster increasing demand from newer network applications such as real-time media streaming and voice over IP.

The world's first congestion control algorithm, TCP Tahoe, was proposed in 1988 by Van Jacobson and quickly evolved into TCP Reno in 1990s, which was later extended to TCP New Reno. Since then, a great number of congestion control/avoidance algorithms have been proposed and evaluated by researchers.

In this Chapter, we investigate different classification of congestion control/avoidance algorithms. We also discuss the advantages and disadvantages of certain algorithms and the motivation of BPC system proposed in this thesis.

2.1 Goals of Congestion Control/Avoidance Algorithm

The goals of congestion control algorithm are not only to avoid congestion in network but also to take full advantage of limited network resources, as listed below.

- to achieve a high bandwidth utilization,
- to provide fairness to other data flows,
- to minimize oscillations in throughput and delay,
- to response quickly to the network dynamics.

The four items are the basic goals of a congestion control/avoidance algorithm. In [4], several performance metrics are also proposed to evaluate a given congestion control/avoidance algorithm. They are:

- throughput, delay and packet loss rates,
- response time to sudden changes or to transient events,
- oscillations in throughput or in delay,
- fairness and convergence times,
- robustness to challenging, dynamic environments,
- robustness to network equipment failures and to misbehaving users.

Analysis of a congestion control/avoidance algorithm should be evaluated by the metrics above.

2.2 Classification of Congestion Control/Avoidance Algorithms

Congestion control and congestion avoidance are two different approaches to handle network congestion. Congestion control is usually reactive after the link is overloaded while congestion avoidance prevents proactively the link from being overwhelmed by network traffic. In this thesis, we only discuss congestion control.

There are many different criteria to classify congestion control algorithms.

- type of different control strategies:

  Examples include AIMD-based algorithms, equation-base algorithms, controller-based algorithms and so forth.

- type of control variable:

  Congestion control algorithms can be divided into window-base algorithm and rate-base algorithm.

- type of feedback:

  Examples include binary feedback algorithms and multi-bit feedback algorithms.

- type of deployment

  Congestion control algorithms can be deployed on end systems or intermediate nodes.

In the remainder of this Chapter, we will discuss typical examples of congestion control algorithms.

2.3 TCP Tahoe

The first congestion control algorithm, TCP Tahoe, and its enhancements, such as TCP Reno, TCP SACK and TCP New Reno dominate the current computer networks. These

algorithms are AIMD-based. The control strategy of AIMD-base algorithms is simple. Basically, if the feedback does not indicate the onset of congestion, the congestion window size, *cwnd*, is increase by one packet. If congestion occurs, the *cwnd* is halved.

$$\text{Ack:} \quad \text{cwnd} = \text{cwnd} + \alpha$$
$$\text{Loss:} \quad \text{cwnd} = \text{cwnd} - \beta \cdot \text{cwnd}$$

In this case, $\alpha = 1,$ and $\beta = 1/2$. Advantages of AIMD algorithm are obvious: it can be easily implemented and it is very robust. However, there are also several drawbacks. Firstly, increasing the congestion window size by one packet is conservative, while decreasing it by a half is too aggressive, particularly for large bandwidth-delay networks. It may take a long time for *cwnd* to achieve the maximum bandwidth in large bandwidth-delay, which results in low utilization. Secondly, AIMD algorithm does not converge to a stable state. The throughput fluctuates roughly between 50% to 100% of the maximum throughput.

TCP Tahoe and its enhancements rely on packet loss timeouts, which are a typical binary feedback to indicate network congestion. More specifically, if the sender receives triple duplicate ACKs or timeouts, the network is considered to be experiencing congestion. However, random bit errors could be the cause of the packet loss and this may occur even when sufficient bandwidth is available. Cross-traffic on the reverse path is one of the possible reasons as well. Hence, packet loss is not a very reliable indication of network congestion. In addition, binary feedback can only provide “true” or “false” of congestion occurrence. It is not able to indicate the level of congestion. On the contrary,

multi-bit feedback can help the sender to understand the network situation much better and make congestion control decisions that are more appropriate.

2.4 FAST TCP and TFRC

Two examples of equation-based algorithms are TFRC [5] and FAST TCP [6]. The former is also rate-based, while the latter is window-based. Rate-based and window-based algorithms are mutually convertible according to specific network information. However, the main difference between these two algorithms is that most of the window-base algorithms use stop-and-wait ARQ (Automatic Repeat reQuest) sliding window protocol. The sender does not transmit the next window of packets until the acknowledgements (ACKs) of previously sent packets are received. However, rate-based algorithms usually send packets continuously. Generally, rate-based algorithms can achieve better utilization, but window-based algorithms tend to provide better reliability.

FAST TCP and TFRC share the same empirical TCP model from [7] based on round-trip time (RTT) and loss rate. The equation derived for rate-based TFRC is

$$\mathrm{T} = \frac{s}{R\sqrt{\frac{2p}{3}} + t_{RTO}(3\sqrt{\frac{3p}{8}})p(1+32p^2)}.$$

This equation gives an upper bound on the sending rate T in bytes/sec, as a function of packet size s, round-trip time R, steady state loss event rate p and the TCP retransmit timeout value t_{RTO}.

The equation used by FAST TCP to update congestion window size is

$$W \leftarrow min\left\{2W, (1-\gamma)W + \gamma\left(\frac{baseRTT}{RTT}W + \alpha\right)\right\},$$

where $\gamma \in (0,1]$, baseRTT is the minimum RTT observed and α is a positive protocol parameter.

Equation-based algorithms can reach equilibrium state and stability with a stationary window size or rate leading to high utilization and minimal oscillations in throughput even for high-speed long-latency networks. Both FAST TCP and TFRC adopt RTT and loss rate as multi-bit feedback indication. Therefore, they are capable of detecting a much more precise congestion level of the network based on which they can make more reasonable decisions than the one uses binary feedback.

However, the performance of an equation-based algorithm largely depends on the accuracy of the model. The TCP model provided by [7] is just an empirical model and was verified in few real-world scenarios. The assumptions made during the formula derivation may not be satisfied in a real-world network environment. Examples include the assumption that packet losses are correlated within a round, which means if a packet is lost, so are all packets that follow, till the end of the round. If the packet loss is caused by random bit errors, which may happen, then this assumption is false. On the other hand, the equations in FAST TCP and TFRC can only compute an upper bound of the next congestion window size. In our experiments, the upper bound computed by TFRC is actually too large to be of practical use. Furthermore, the accurate loss event rate p in the equation is also difficult to estimate.

2.5 XCP and Other Controller-based Algorithms

From the viewpoint of control theory, the congestion control issue can be seen as a complex nonlinear control system. As a result, a potential solution to the issue is to use a control theory framework to analyze and control the network congestion. However, classic controllers such as proportional-integral-derivative controller (PID controller) require an accurate model of the process to be controlled, which is almost impossible for today's Internet. Network traffic is simply too complicated to be modeled precisely. However, this is the case if we treat the entire intermediate networks as a black box and attempt to control it. If we can control the intermediate nodes and collect link characteristics from each of them, the congestion control issue would be much simpler.

Several algorithms using a PID controller or other controllers have been proposed. The eXplicit Control Protocol (XCP) is one such algorithm which has received much attention [8]. XCP involves a joint design of end-systems and routers. The XCP sender uses a field called *H_feedback* to update the congestion window size. Whenever a new acknowledgement arrives, the *cwnd* is updated as follow:

$$cwnd = max\left(cwnd + H_{feedback}, s\right),$$

where *s* is the packet size and *H_feedback* could be positive or negative. The XCP router is responsible to compute the feedback value to cause the system to converge to optimal efficiency and min-max fairness. Two controllers, the efficiency controller and the fairness controller, are used to compute *H_feedback*.

Another controller-based congestion control algorithm is proposed in [9] which adopts a modified Smith Predictor to mitigate the effect of feedback delay. This rate controller is shown in Fig. 2.1.

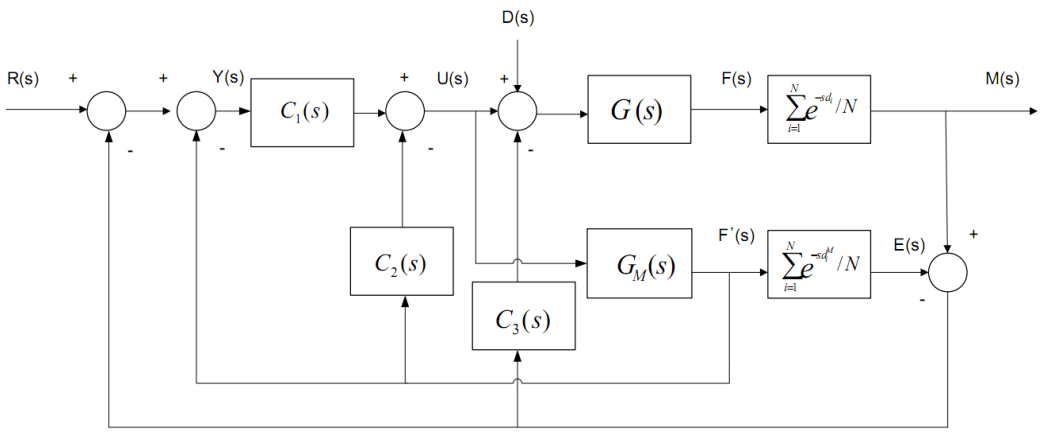

Figure 2.1 A modified rate controller with Smith predictor

With the modified Smith Predictor, this rate controller is able to cope with feedback delay and to provide better performance in convergence time and overshoot magnitude.

Most of the proposed controller-based congestion algorithms involve both end-system and the intermediate node system. The drawbacks are just like the SAFIRE system we discussed in Chapter 1. Other congestion control algorithms such as TCP Tahoe, FAST TCP and TFRC discussed earlier are end-to-end systems, which leave the network relatively simple in terms of congestion control functionality. The dilemma is that an end-to-end system is easier to upgrade but harder to gather accurate link

characteristics, while joint system is costly to upgrade but may be able to acquire more useful link information.

The solution we proposed to solve this issue is to design a network overlay system to provide congestion control. The BPC system proposed in this thesis only needs to be installed on some selected intermediate nodes to create an additional overlay network. Thus, the expense of upgrading the network can be kept to a minimum. Meanwhile, overlay nodes can acquire link information directly from their underlying legacy routers.

2.7 Conclusion

In this Chapter we presented an discussion of current congestion control algorithms. The advantages and limitations of various algorithms were considered. Based on the discussion given, none of those algorithms perfectly fits our requirements, as presented in Chapter 1. This motivates us to develop the BPC system to be discussed in Chapter 4 and 5.

Chapter 3: Implementation Issues and Experimental Setup

The BPC system was implemented as part of the CHART system described in Chapter 1, as a set of modules within the SAFIRE software flow router. SAFIRE in turn is based on the Click modular software router [10]. The BPC system was developed and tested using the Emulab network emulation testbed. This Chapter describes the building blocks of the BPC system based on the CHART/SAFIRE framework. Implementation issues and the experimental setup for investigating and developing the BPC system are also discussed.

3.1 Click modular router

Both SAFIRE and the BPC system use the basic data forwarding plane of Click modular router, an open source software project. The Click modular router, which was developed by MIT and maintained by University of California, Los Angeles and other organizations, is a scalable software architecture for building new configurable software routers. A software router is an application that runs on a general-purpose computer, which performs the functions of a router in software. The chief advantage of a software router is that it can be reconfigured easily and new functionality can be added readily to the router. This makes a software router ideal for experimenting with new networking

protocols in a realistic network environment. On the other hand, software routers are much slower than specialized hardware routers.

The Click software router consists of different modules which are called elements. As Click elements are programmed in C++, they can be easily inherited and modified by users to implement various functionalities. Each element is designed as a C++ class and connected to other elements through one or more inputs and outputs. A single element is a software component representing a unit of router processing, such as queueing, packet classification and so forth.

The BPC system is based on SAFIRE, which was also developed using Click modular router. Several universal packet processing elements from the original Click library were adopted to implement the BPC system. The Click elements are integrated into a functioning router using Click configuration files.

A simple example of a Click configuration file is shown below.

```
// R3.conf:
//      Click configuration file for "R3" router in "Simple-Topology"
//      with IP tunneling.
//

AddressInfo (
        source1      10.1.3.2 10.1.3.0/24,
        sink1        10.1.1.3 10.1.1.0/24,
        R1           10.1.5.2,
        R2           10.1.4.3,
);

// Routing table lookup
rt_flow :: LinearIPLookup (
        source1                 1,
        sink1                   2,
        255.255.255.255/32      0.0.0.0         0,
        0.0.0.0/0               0,
        0.0.0.0/32              0);

rt_flow[0] -> EtherEncap(0x0800, 1:1:1:1:1:1, 2:2:2:2:2:2) -> Discard;
```

```
rt_flow[1] -> IPPrint("to R1", TTL true)
           -> Queue
           -> Socket(UDP,R1,10002);

rt_flow[2] -> Queue
           -> Socket(UDP,R2,10002);

// Inbound from Click overlay
Socket(UDP, 0.0.0.0, 10002)
        -> CheckIPHeader
        -> rt_flow;
```

Fig. 3.1 shows the connection of different elements and packet processing procedures of this Click configuration file. In this example, packets arrive at the router from a socket tunnel. Then the IP header of incoming packets is checked by `CheckIPHeader` element. The `LinearIPLookup` element routes packets passing from the preceding element to the corresponding output. Output port 1 of `LinearIPLookup` is connected to the element `EtherEncap` which encapsulates packets in an Ethernet header and then the packets are dropped. Packets routed to output port 2 and 3 are enqueued by the `Queue` element and then sent to another socket.

3.2 Features of SAFIRE and implementation

SAFIRE integrates a control plane and a forwarding plane. The function of the control plane is to adaptively select the routing path based on a flow routing table. The forwarding plane simply forwards the packets according to the flow routing table. The two fundamental features of SAFIRE we are about to discuss are flow routing and in-band explicit rate-based signaling.

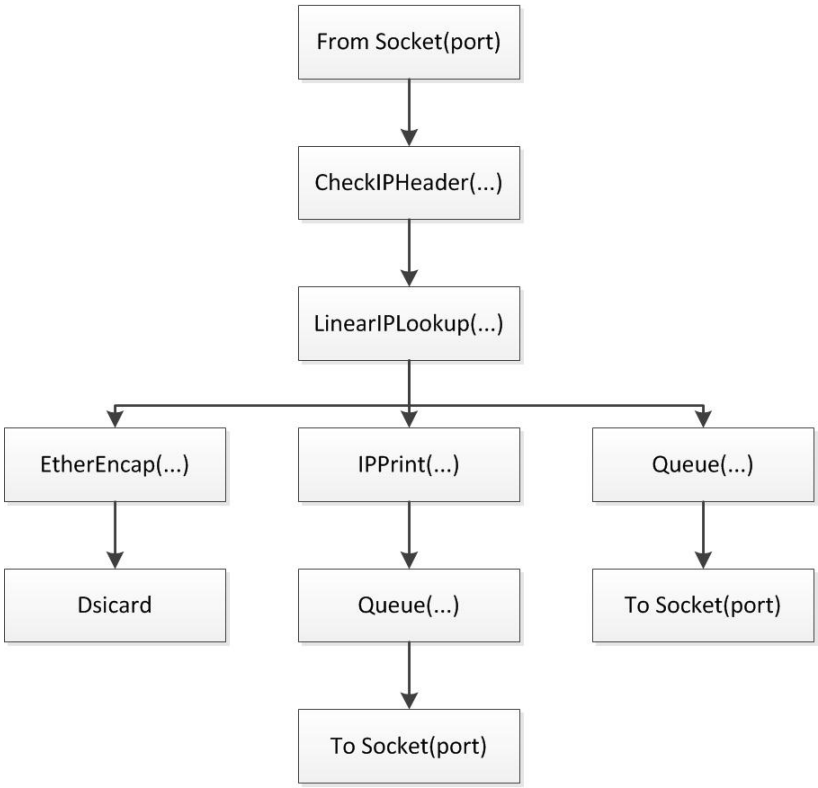

Figure 3.1: Simple example of Click configuration file

3.2.1 Flow routing

One of the main contributions of SAFIRE is that data traffic is processed in terms of flows rather than packets. A flow router is defined as a router can that recognizes IP flows, stores flow information, and makes forwarding decisions based on flow information. Benefits of flow routing include QoS guarantees, improved throughout performance and fast failure recovery. The FR-1000 flow router developed by Anagran

as part of the CHART project is a hardware flow router, in contrast to SAFIRE, which is a software flow router.

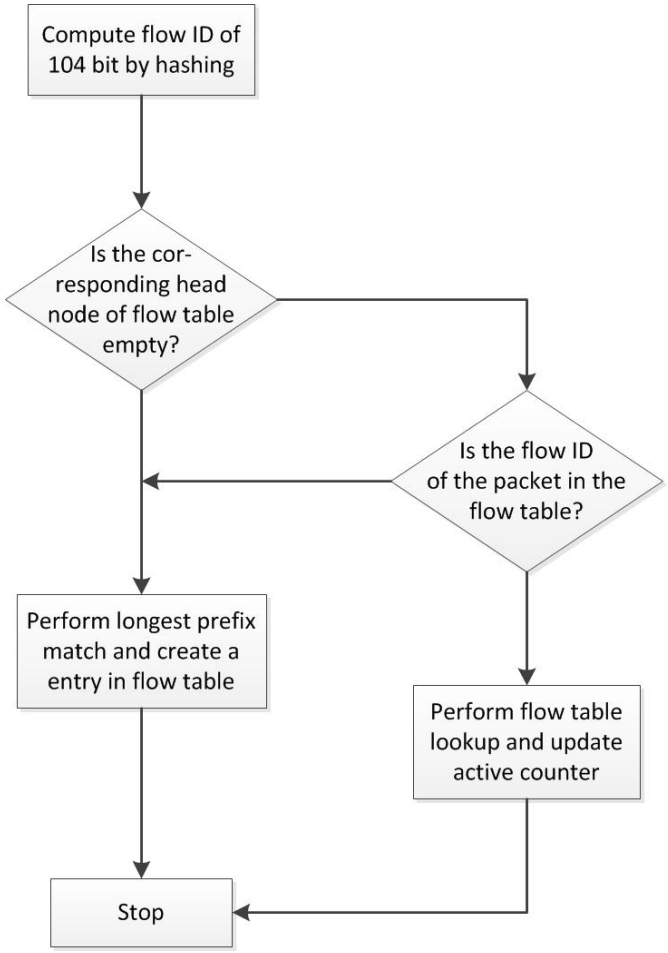

Figure 3.2 Flowchart of FlowLookup element

To realize the flow routing feature, SAFIRE includes a Click element called `FlowLookup`. Fig. 3.2 shows the flowchart of `FlowLookup`.

When the first packet arrives at SAFIRE, a unique flow ID is computed by hashing a 5-tuple (source IP address, destination IP address, source port number, destination port number, and protocol number) to identify a data flow. SAFIRE keeps track of the data flows traversing it and records the total number of flows. Whenever a packet arrives, the flow table where the flow information is stored is searched first. If the flow ID computed from the packet header is found in flow table and being active, the packet will be forwarded accordingly. If not, the packet will be forwarded based on a longest prefix match search and a new flow entry will be created and stored.

3.2.2 In-band rate-base signaling

TIA 1039 is an in-band QoS signaling protocol, designed by Anagran and standardized by TIA. The protocol can allocate network resource for flows via in-band signaling as they traverse the network. SAFIRE and BPC system were originally designed to support the TIA 1039 protocol.

TIA 1039 supports the following QoS parameters:

```
GR- Guaranteed Rate
MR- Maximum Rate
VR- Variable Rate
AR- Available Rate
PP- Preemption Priority
BT- Burst Tolerance
DP- Delay Priority
CH- Charge Direction
```

The GR parameter specifies a reserved bandwidth for a flow, which should be guaranteed by intermediate routers. Once the GR is confirmed for a flow, the allocated

bandwidth resource will not be shared with other flows. The AR parameter is the available rate for the flow based on the current loading of the network. The AR value changes from time to time according to the current network load. Detailed explanations of all of the TIA 1039 parameters can be found in [3].

A QoS header is inserted to the first packet and subsequently, a QoS header is inserted into once every 128 packets. The interval between QoS headers is by default 128 packets, but the sender may choose a different value. When QoS packet arrives at routers supporting TIA 1039 protocol, the resource is reserved and specific fields of the QoS header is updated by the routers. After the first packet arrives at the destination, a feedback QoS packet is sent to the source. Then the actual QoS flow is established. Fig. 3.3 shows the structure of TIA 1039 QoS header.

Fig. 3.4 gives an example showing how the AR value is computed and how a QoS flow is established. The flow source sends the first packet with AR = 100. When the first packet reaches the first router, its AR value will be reset to 35, because there are only 35 bandwidth units available at this router. When the packet is processed by the third router, its AR value reset to 30. So when the first packet arrive the receiver, its AR value is 30. Then the receiver sends a QoS feedback packet to the sender, informing it of the final AR setting of this flow. The sender then adjusts its sending speed to 30. Once a QoS flow is established, not all packets belonging to this flow need to carry the QoS header. Only after a certain, i.e. 128 packets mentioned earlier, interval of packet transmissions, a QoS structure needs to be resent by the sender to update the QoS information.

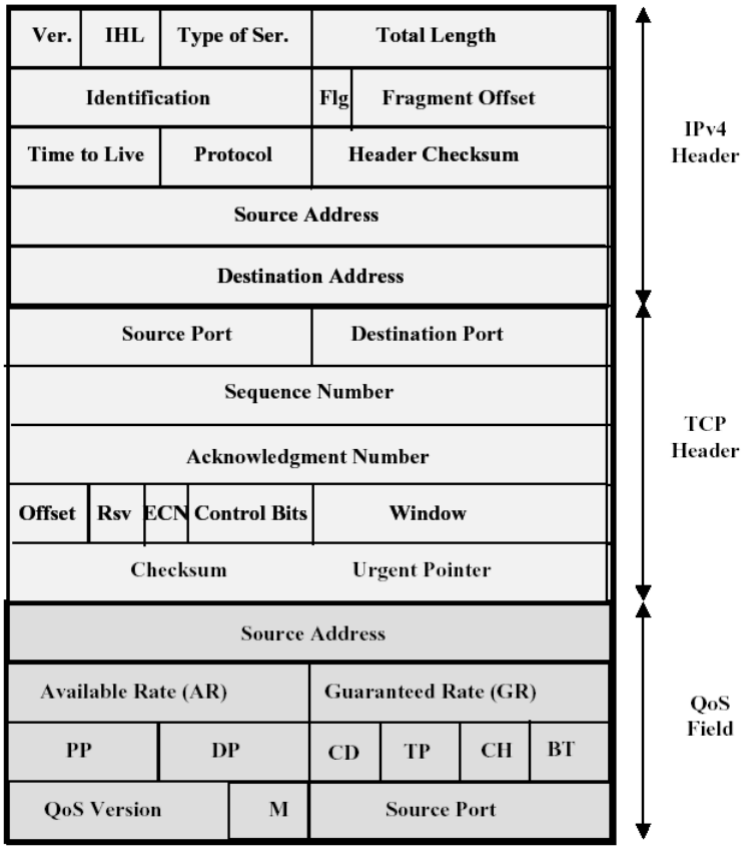

Figure 3.3 TIA 1039 QoS header structure [3]

A Click element called `RateUpdate` to implement the AR and MR provisions of TIA 1039. The AR parameter is the most important parameter for improving the current TCP performance. It can be used by end-system to figure out the maximum sending rate without causing severe network congestion along the path. A data flow, which follows the TIA 1039 QoS protocol and which is aware of the explicit sending rate is referred to

as TCP-ER flow. In overlay scenario, the BPC system provides the AR value of the overlay link to end-systems.

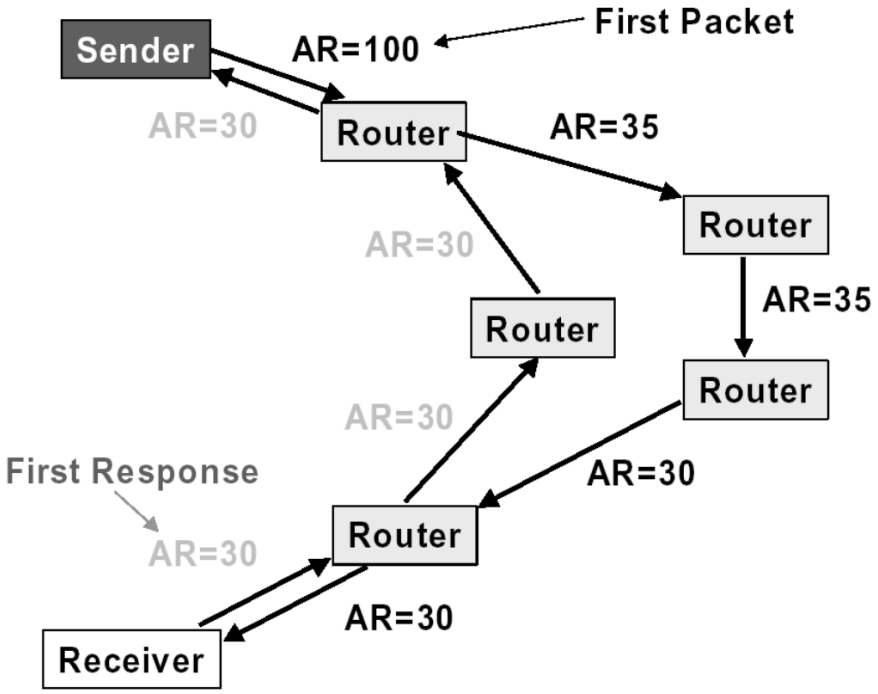

Figure 3.4 A simple example of QoS flow establishment [3]

3.3 The BPC system implementation

The routers with the BPC system are divided into three types according its functionality: BPC-IN, BPC-OUT or a combination of both. A BPC-IN router encapsulates the incoming packets of an overlay link in a new header shown in Fig. 3.5. Then the packet is

delivered to the underlying legacy router and then transmitted to the BPC-OUT router that is on the other end of the overlay link. The BPC-OUT router reads the inserted header and updates the related fields. A feedback packet is then generated and sent back to BPC-IN router in which the sending rate is computed and updated based on the link characteristics stored in the feedback packet. An AR value is also sent to the end-systems via the TIA 1039 signaling protocol.

Time Stamp	
Serial Number	Delay
Loss Counter	

Figure 3.5 The new header inserted by BPC-IN

If two overlay links share a common BPC router, this router performs the functionalities of both BPC-IN and BPC-OUT. Fig. 3.6 shows an example of a network with multiple BPC routers. The blue line and red line represent two different overlay links respectively. Green line represents the cross-traffic which traverses a portion of red overlay link but is not observed by any BPC overlay router.

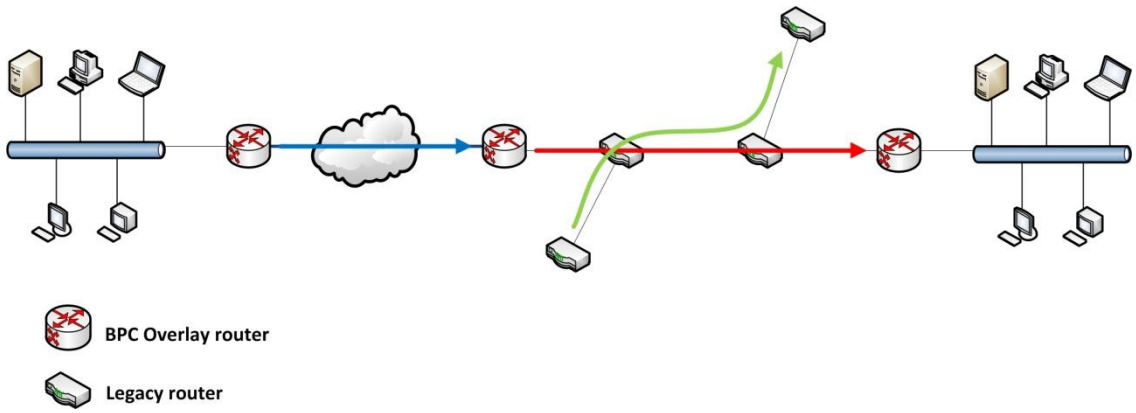

Figure 3.6 An example of a network with BPC routers

We developed three Click elements to realize this process. They are:

- `IPTimesender`

  The `IPTimesender` element is located in the BPC-IN router. It inserts a 20-byte header between the transport layer and data section of a packet. This header consists of a time stamp, a serial number, delay information and loss information. Then the encapsulated packet is sent to the underlying IP networks.

- `IPTimereceiver`

  `IPTimereceiver` element operates within the BPC-OUT router. This element analyses the header inserted by IPTimesender. It calculates the single-trip delay and the number of packet loss, if any. Then it creates a feedback packet which will be sent back to BPC-IN router to carry this information.

- `FeedbackControl`

`FeedbackControl` in the BPC-OUT router is the element that collects the link characteristics of the overlay link from feedback and computes the appropriate sending rate to control the congestion on the overlay link. Details about the implementation of the congestion control algorithm will be discussed in Chapter 5.

3.4 Emulab

Emulab [11] is a network testbed built by University of Utah, giving researchers a wide range of environments in which to develop, debug, and evaluate their systems. On Emulab, users are allowed to create their own network topologies with real network equipment and to configure the network nodes to test new protocols. Users can also modify the bandwidth, loss rate and latency of a given link dynamically.

To create an experiment, users need to submit a topology file written in the scripting language Tcl. The topology file specifies the network structure, link characteristics, hardware of the nodes and operating systems.

To evaluate BPC system, we created a network topology on Emulab. The topology file in Tcl is as follows. We chose pc850 as the node hardware and Fedora Core 4.0 Linux as the operating system. The delay of each link is set to 5 ms. Link capacity is set to 100 Mbps except that the link between *clicks* and *clickn* is set 1Mbps.

The network topology is shown in Fig. 3.7.

```
#Bandwidth Probe Control
#Create a simulator object
set ns [new Simulator]
source tb_compat.tcl
```

```
#Enable linux router
$ns rtproto Session

######################BUILD_TOPOLOGY############################

#Create east and west coast CONUS nodes
set westc [$ns node]
set eastc [$ns node]
set westc0 [$ns node]
set eastc0 [$ns node]
set crossn [$ns node]
set crosss [$ns node]

set clickw [$ns node]
set clicke [$ns node]
set clickn [$ns node]
set clicks [$ns node]

#Set special node hardware for QoS driver
tb-set-hardware $eastc pc850
tb-set-hardware $westc pc850
tb-set-hardware $clickw pc850
tb-set-hardware $clicke pc850
tb-set-hardware $clickn pc850
tb-set-hardware $clicks pc850
tb-set-hardware $crossn pc850
tb-set-hardware $crosss pc850

#Set of the node OS for sources and destination FC4-UPDATE
tb-set-node-os $eastc QOS19-850
tb-set-node-os $westc QOS19-850
tb-set-node-os $clickw FC4-UPDATE
tb-set-node-os $clicke FC4-UPDATE
tb-set-node-os $clickn FC4-UPDATE
tb-set-node-os $clicks FC4-UPDATE
tb-set-node-os $crossn FC4-UPDATE
tb-set-node-os $crosss FC4-UPDATE

############################Enable CSI################################
tb-set-node-startcmd clicke
"/proj/chart/groups/csi/csi/common/utils/initialize"
tb-set-node-startcmd clickw
"/proj/chart/groups/csi/csi/common/utils/initialize"
tb-set-node-startcmd clickn
"/proj/chart/groups/csi/csi/common/utils/initialize"
tb-set-node-startcmd clicks
"/proj/chart/groups/csi/csi/common/utils/initialize"

#Create initial link between east and west
set link1 [$ns duplex-link $clickw $clickn 100Mb 5ms DropTail]
set link2 [$ns duplex-link $clickn $clicks 1Mb 5ms DropTail]
set link3 [$ns duplex-link $clicks $clicke 100Mb 5ms DropTail]
set link4 [$ns duplex-link $crossn $clickn 100Mb 5ms DropTail]
```

```
set link5 [$ns duplex-link $clicks $crosss 100Mb 5ms DropTail]

#Set LAN
set lan0 [$ns make-lan "$westc $westc0 $clickw" 100Mb 5ms]
set lan1 [$ns make-lan "$eastc $eastc0 $clicke" 100Mb 5ms]

######################LINK_LOSS############################

tb-set-link-loss $link1 0.01
tb-set-link-loss $link2 0.01
tb-set-link-loss $link3 0.01
tb-set-link-loss $link4 0.01
tb-set-link-loss $link5 0.01

#Run the simulation
$ns run
```

3.5 *iperf*

Iperf [17] is a network testing tools written in C++ that can generate TCP and UDP data flows and measure the throughput, transmitted data size and loss rate of a network. It is an open source cross-platform software project supported by the National Laboratory for Applied Network Research. In the experiments in the remainder of this thesis, we invoke *iperf* to generate UDP and TCP data flows as main-traffic or cross-traffic.

Iperf includes two functionalities, which are client and server. The information of the data flow created is displayed on the server for users to monitor the network situation. *Iperf* allows the user to set various parameters for different types of data flows. For example, the sending rate, datagram size and duration time can be configured for UDP data flows.

In addition, another new implementation called *iperf3* [18] is being developed to provide a smaller, simpler code base, and library version of the *iperf*. However, it is not backwards compatible with *iperf*.

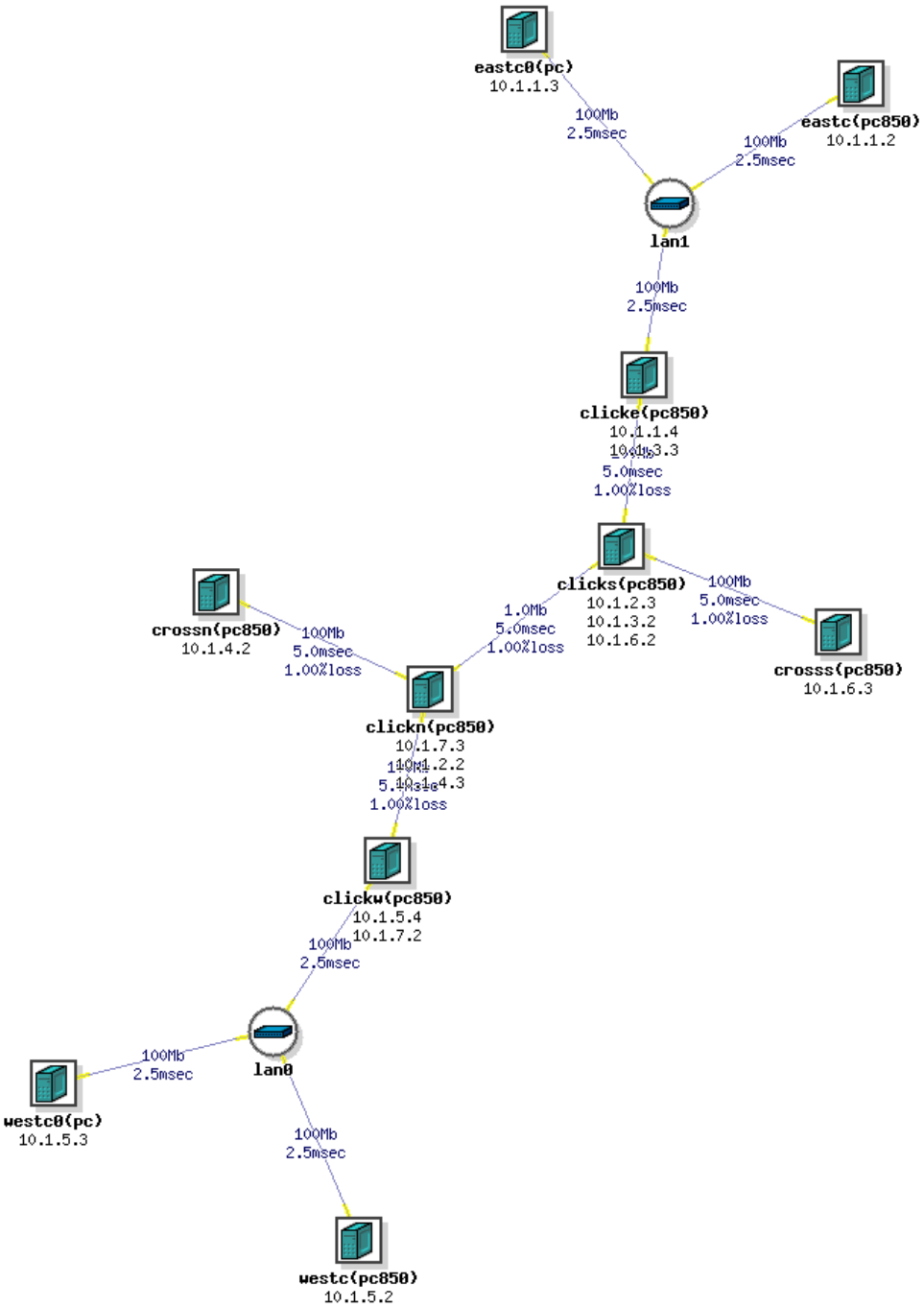

Figure 3.7: Experiment topology on Emulab

Chapter 4: Delay-based Congestion Control

In this Chapter, the use of delay measurements for controlling congestion on an overlay link is investigated. First, related work on delay-based congestion control algorithms is reviewed. Then, congestion control based on single-trip delay measurements through experiments on Emulab. Finally, the results of the experimental study are analyzed and conclusions are given.

4.1 Related work

Using delay as congestion signal can overcome two main drawbacks of packet loss timeouts. First, delay can provide much more accurate and reliable estimation of network congestion than packet loss timeouts. Indeed, the delay information is noisy, just as of the packet loss rate. The triple ACK timeouts used for congestion detection can only suggest if congestion occurs or not, due to the binary nature of the packet loss signal. Further, the oscillation of congestion window size is unavoidable when using packet loss timeouts as congestion signal, especially in a network with large bandwidth-delay product. A multi-bit congestion signal has to be employed to eliminate the two drawbacks.

A delay-based congestion avoidance algorithm (DCA) was proposed by Jain [12] in 1989 for the first time. He discussed the feasibility of using changes of RTT as an implicit feedback signal and derived an expression of optimal congestion window size as a function of RTT. A practical implementation of DCA is best exemplified by TCP Vegas [13] which was developed at the University of Arizona in 1994. The main strategy of TCP Vegas is to keep a small number of packets buffered in the routers along the path by adjusting the congestion window based on RTT. However, the performance of TCP Vegas largely depends on the accuracy of the estimation of the *baseRTT*, which may be affected by rerouting or other network dynamics. Another problem is the fairness to legacy TCP flows. It was reported in [14] that TCP Vegas flows receive a smaller fair share of the available link capacity when competing with other types of TCP flows, i.e., TCP Reno.

Further, [15] suggested that the correlation between RTT and packet loss is not strong enough to help the TCP sender to improve throughput reliably. This conclusion was established both by real-world experiments on seven high-speed Internet paths and by simulation experiments. However, a weak correlation between RTT and packet loss does not mean that the fluctuation of delay cannot reflect the level of network congestion. It is probably a wrong approach to enhance the AIMD algorithm by using the correlation between RTT and packet loss in TCP protocols. However, delay can be combined with other non-AIMD congestion control algorithms to improve the performance of congestion control. In the BPC system to be discussed in Chapter 5, single-trip delay is used by a simplified fuzzy logic controller and the binary search method to control the

network congestion. Other congestion algorithms, such as FAST TCP and TFRC, which are discussed in Chapter 2, also achieve better performance with RTT as feedback signal.

Therefore, the key issue in the design of delay-based congestion control algorithms is how to take advantage of delay information.

4.2 Study on single-trip delay behavior

The single-trip delay can be on an overlay link represented as

$$Delay = T_{trans.} + T_{prop.} + T_{Proc.} + T_Q,$$

where $T_{trans.}$ is the transmission delay, $T_{prop.}$ is the propagation delay, $T_{Proc.}$ is the packets processing delay and T_Q is the queueing delay.

Generally, $T_{trans.}$, $T_{prop.}$ and $T_{Proc.}$ are almost invariable, if the path is fixed. Hence, the variation in the single-trip delay is usually caused by queueing delay. According to Little's law, which is given by

$$Q_W = \lambda T_W,$$

the long-term average number of packets Q_W is equal to the product of the long-term arrival rate λ and the waiting time T_W which is queueing delay. The arrival rate is also related to the sending rate. For example, in an overlay network system, if we treat an overlay link as a single virtual hop, then the arrival rate is the sending rate of one end of the overlay link and all packets traveling on the overlay link are packets waiting in a virtual queue.

4.2.1 Experimental setup

Several experiments corresponding to different network situations were set up to investigate the feasibility of using single-trip delay as network congestion measurement. Instead of conducting experiments in the real Internet environment, we chose Emulab as experimental environment. The reason behind this is that Emulab provides a purer environment without other noise traffic compared to the live Internet, which is better for analyzing the behavior of delay under different congestion control strategies.

The experiments used the same network topology described in Chapter 3.The links between *clicke* and *clickw* constitute an overlay link. The bottleneck of this overlay link lies on the link between *clicks* and *clickn*, which is set to 1 Mbps. We use a network tool called *iperf* to generate UDP and TCP traffic from *westc* to *eastc* and to measure the throughput. The BPC system computes the forwarding delay between the BPC-IN in *clickw* and the BPC-OUT in *clicke*. Traffic traveling through the entire overlay link is called main-traffic in the remainder of this thesis. Main-traffic is a mixture of different types of traffics, i.e., UDP and TCP Reno. Cross-traffic traverses the link between *clicks* and *clickn*. For simplicity, we used UDP flow as the main-traffic in our experiments. Two types of cross-traffic, TCP and UDP, are introduced into the experiments separately.

4.2.2 Experimental results

4.2.2.1 Main-traffic without cross-traffic

In this experiment, the main-traffic represented by UDP flow is the only traffic in the entire network. The sending rate of the main-traffic is increased from 150 Kbps to 1 Mbps in increments of around 500 Kbps. Each of the sending rate is kept for at least 10 seconds to capture the dynamics of the delay.

Fig. 4.1 shows the results of the experiment. The real-time single-trip delay does not change much before the sending rate reaches 950 Kbps which is 95% of the bottleneck capacity. At 950 Kbps of the sending rate, the delay starts to fluctuate up to 50% more than the value before. This is caused by the packets being queued in the buffer of the bottleneck link. However, after the sending rate climbs up to 1 Mbps which is exactly the bottleneck capacity, the delay starts to increase linearly, from a average of 37 ms to around 450 ms and is finally stabilized around 400 ms. The remarkable increment of delay from 26 to 35 seconds is due to packets piling up quickly at the buffer of the bottleneck link. After the buffer is filled up, the delay remains roughly stable. In addition, the throughput computed by *iperf* drops soon after the sending rate is set to 1 Mbps, since more packet are lost at this stage.

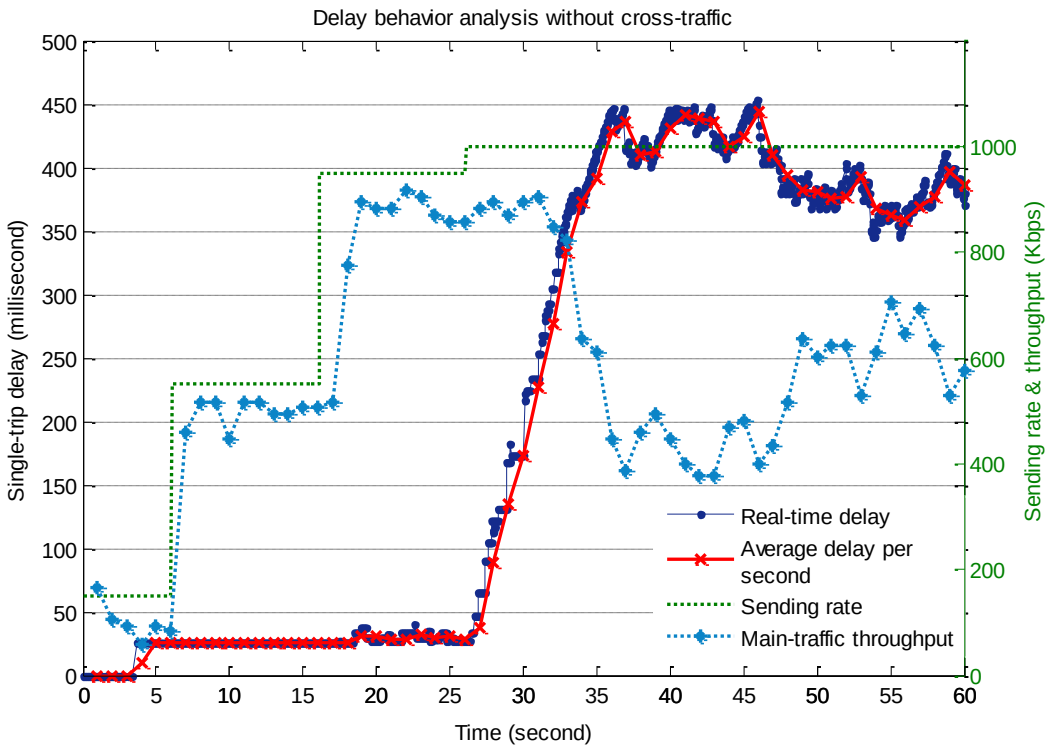

Figure 4.1: Delay and throughput behavior without cross-traffic

4.2.2.2 Main-traffic with TCP cross-traffic

A TCP flow is introduced as cross-traffic in this experiment. The sending rate of main-traffic is set to a constant of 500 Kbps, which is half of the bottleneck capacity. Since TCP uses the AIMD congestion control algorithm (see Section 2.3), it keeps trying to find the maximum available bandwidth along the path.

Fig. 4.2 presents the experiment results. Delay fluctuates drastically up to 200 ms after the TCP cross-traffic is introduced. The reason why delay cannot maintained at a stable value is that the AIMD adjusts the congestion window size all the time and never

converges to a steady-state value. On the other hand, the throughputs of both main-traffic and TCP cross-traffic are around 500 Kbps.

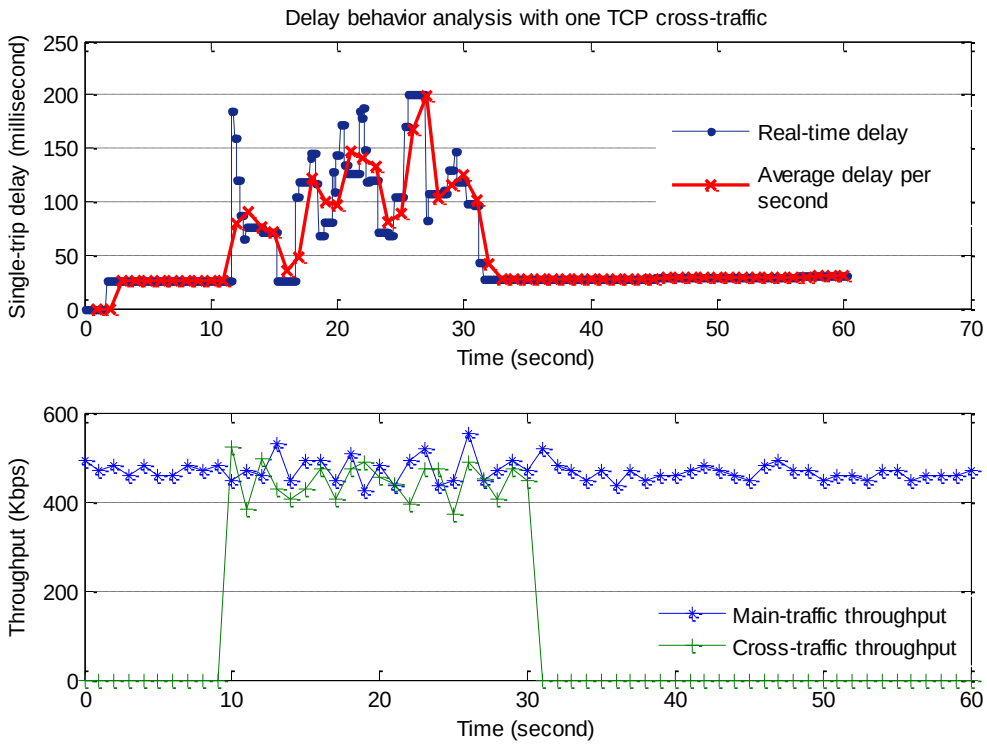

Figure 4.2 Delay and throughput behavior with TCP cross-traffic

4.2.2.3 Main-traffic with UDP cross-traffic

A UDP flow at the rate of 400 Kbps replaces TCP flow served as the cross-traffic in this experiment, while the main-traffic keeps the rate of 500 Kbps. The rate 400 Kbps is

chosen to be the rate of the cross-traffic because a higher rate would overwhelm the bottleneck link and causes severe congestion, just as seen in the first experiment.

From the results shown in Fig. 4.3, we see that after introducing UDP cross-traffic at 10 seconds, the delay of the main-traffic does become less stable than before. However, the magnitude of the fluctuations is much more moderate than in the scenario with TCP cross-traffic. Since UDP does not include any congestion control scheme, the sending rate of the cross-traffic is basically a constant, which could be treated as a steady-state value.

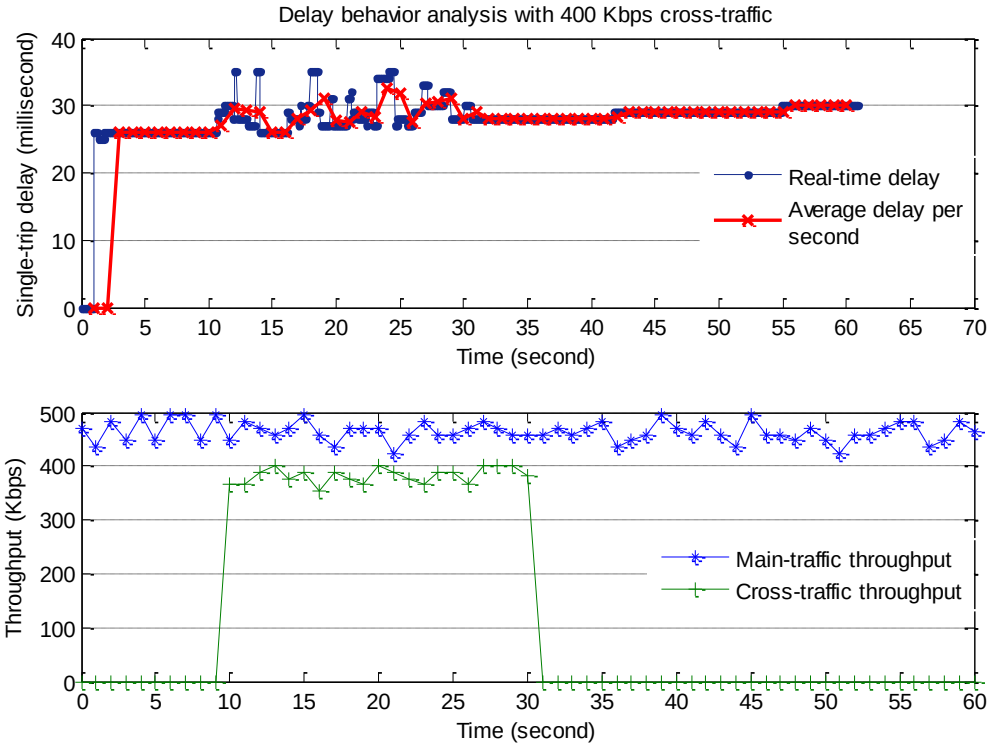

Figure 4.3 Delay and throughput behavior with UDP cross-traffic

4.3 Analysis and conclusions

According to our experiment results shown in Fig. 4.1, the buffer of the bottleneck link starts to enqueue packets when the sending rate is increased to around 95% of the bottleneck capacity. The higher the rate is, the faster and more severe the network becomes congested. Our experiments also indicate that if the sending rate is twice that of the bottleneck capacity, the buffer of the bottleneck link can be filled up within 1 second. Therefore, an indication of the sending rate reaching the maximum value is a moderate fluctuation of delay. In this case, the range of sending rate in which this indication emerges is around 940 Kbps to 980 Kbps. This is the range in which the BPC system, to be discussed in the next chapter, attempts to control the sending rate.

In [14], it was claimed that when a TCP Vegas flow is competing with a TCP Reno flow, the TCP Vegas flow continuously backs off, losing buffer space to TCP Reno. This problem can be explained by the results shown in Fig. 4.2. Since the AIMD algorithm of TCP Reno keeps being greedy for more bandwidth, the bottleneck link repeatedly experiences a process of being overloaded and then being relieved. As a result, the delay experienced by the main-traffic changes continuously. If TCP Reno detects a packet loss when the buffer of bottleneck link is almost full, the delay of the main-traffic is very likely to be a large value. In the second experiment, we see a maximum of 200 ms delay, which is almost 10 times the normal value. Since the delay is greater than *baseRTT* most of the time, TCP Vegas will continuously decide to back off its transmission rate , which results in its unfair treatment compared to TCP Reno. To avoid this situation, the BPC

system chooses a minimum value between the rate of incoming main-traffic and half of the detected bottleneck capacity to be the sending rate. In the worst case, the main-traffic is guaranteed for at least half of the bottleneck capacity, if needed.

In this Chapter, we analyzed the delay behavior in various scenarios. It was established that changes in the single-trip delay are able to reflect the network congestion level. If delay information can be used in a proper way, it can be a very good measurement signal for congestion control. The experimental results discussed in this Chapter for the basis for the BPC system developed in the next Chapter.

Chapter 5: Bandwidth Probe Control Algorithm

In this Chapter, we discuss the details of the congestion control algorithm of the BPC system. Based on the experimental results discussed in Chapter 4, the BPC system uses the single-trip delay between the BPC-IN node and the BPC-OUT node to evaluate the congestion level of the overlay link. Unlike other delay-based congestion control algorithms, which use round-trip time, single-trip delay is able to provide more accurate congestion information without being disturbed by the network state on the reverse path.

The BPC system collects single-trip delay for each packet delivered successfully and then computes an average delay by applying the exponentially weighted moving average (EWMA) method for every sample period. The EWMA delays are stored in a C++ vector until the next adjustment period arrives. The mean value and two types of standard deviation of the EWMA delay values in the vector are computed and evaluated by a function called *delay_analysis* at the beginning of the next adjustment period. The BPC system computes a new sending rate based on the value returned by *delay_analysis*.

5.1 Fuzzy logic algorithm

The BPC system applies a fuzzy logic algorithm to define the network state based on the parameters described above. The concept of fuzzy logic was proposed by Lotfi Zadeh in

1965 [19]. At first, the fuzzy logic was designed to deal with data-implicit states, different binary values 0 and 1. The fuzzy set theory in [19] provides a robust mathematical framework to process the uncertainty and imprecision in the real-world. Although the concept of fuzzy logic is now well-known in the form of fuzzy logic controllers, it was not applied to control system until the 1970s. The fuzzy logic was introduced to control congestion in ATM networks by [20]. A number of protocols and algorithms that apply fuzzy logic in telecommunication networks were discussed in [21]. In the *delay_analysis* function, we define a set of fuzzy rules for the statistical parameters of the single-trip delay to determine the network state.

There are two types of periods in the BPC system: the sample period and the adjustment period. The adjustment period is longer than sample period, because the system needs to wait until the new sending rate takes effect on the system. A sufficient number of delay samples is also required to evaluate the effect of the new sending rate.

At the beginning of a new adjustment period, the *delay_analysis* function is triggered to evaluate the effect of the previous sending rate on the overlay link to the network. This function computes four parameter values:

- the average delay of a adjustment period ($delay\_avg$),
- the normal standard deviation of delay collected during a adjustment period ($delay\_std$),
- the standard deviation based on *baseDelay* ($delay\_std\_base$),
- the slope of delay variation ($delay\_slope$).

The *baseDelay* is the standard single-trip delay measured during the system startup time when the network is not fully utilized.

From the experiments performed in Chapter 4, we found that there are three possible states of the network.

- network is underutilized:

  In this case, the network capacity is not fully utilized. Sender can increase the sending rate, if more bandwidth is required.

- network is on the "edge" :

  When the network is on the edge, it means that the sending rate is already in the range that packets start to be enqueued in the bottleneck buffer along the path. The sending rate should not increase any further in this state.

- network is congested:

  In this state, the delay is much larger than normal and it fluctuates in a certain range. The sending rate needs to decrease in this state.

Network situation	***delay_avg***	***delay_std***	***delay_std_base***	***delay_slope***
Underutilization (1)	$< 1.5 \times baseDelay$	< 1.325	N/A	N/A
On the edge (2)	$> baseDelay$	1.325~13.25	$< baseDelay$	
Severe congestion (3)	$>5 \times baseDelay$	≥ 1.325		> -2
To be observed (0)	else	else	else	else

Table 5.1 Criteria to distinguish different network states

The *delay_analysis* function uses the four parameters to determine in which of these three states the network lies in. We choose two basic criteria to distinguish the three states. They are based on the values $1.5 \times delay\_avg$ and $3 \times delay\_avg$. The values of $delay\_std$ and $delay\_std\_base$ are computed based on the two criteria and *baseDelay* using the relationship $\sigma = \mathrm{E}[(\mathrm{x} - \mu)^2] = \mathrm{E}[x^2] - \mu^2$. Table 5.1 shows the combination of the four measurements for different network states.

The parameters used in this table are based on the two rules of $1.5 \times delay\_avg$ and $3 \times delay\_avg$ and the specific experimental situation.

The evaluation result of the *delay_analysis* function is returned to the controller to determine the appropriate course of action in next step of the congestion control algorithm..

5.2 Congestion control algorithm

The congestion control algorithm of the BPC system is divided into three exclusive phases: the increasing phase, the searching phase and the controlling phase. Only one of the three phases can be active at any given time. The increasing phase and the searching phase are used to probe for the maximum available bandwidth (i.e., the bottleneck link capacity), during which we assume there is no cross-traffic throughout the network, i.e., in the system startup phase. After the bottleneck capacity is acquired by the BPC system, it steps into the controlling phase to cope with cross-traffic and maintain the congestion level of the overlay link. If the function *delay_analysis* determines that the network is

underutilized, the corresponding sending rate is called a “safe” value. If the network is overloaded, it is called an “aggressive” value.

In the remainder of the thesis, we use $R_{(n)}$ to represent the new sending rate, B to represent the bottleneck capacity, R_t to represent the most recent safe value, R_{agg} to represent the most recent aggressive value and R_{income} to represent the incoming rate.

5.2.1 Increasing phase

In the increasing phase, the sending rate of BPC-IN node is initialized to a relatively small value. This initialization value can be customized by users based on their experience and this value may affect the convergence time of the BPC system. During the initializing phase, the control strategies for different network states are as follow:

- network is underutilized:

  If the incoming rate is greater than preceding sending rate, increase the sending rate exponentially, according to $R_{(n)} = 2 \times R_{(n-1)}$. Otherwise, switch to the controlling phase.

- network is on the “edge”:

  Switch to the controlling phase. The sending rate is recorded as the bottleneck capacity, $B = R_{(n-1)}$.

- network is congested:

  Switch to searching phase.

The worst possible scenario when exiting the increasing phase is that the sending rate is twice the bottleneck capacity, while the best scenario is that the sending rate just falls into the edge region.

5.2.2 Searching phase

The BPC system jumps into the searching phase because the bottleneck link is overloaded during the increasing phase. In the searching phase, a binary search method is performed to adjust the sending rate and finally reach the bottleneck capacity.

The different control strategies employed in this phase are shown below.

- network is underutilized:

  This means that the sending rate was decreased too much in the previous period. The new sending rate will be $R_{(n)} = (R_{(n-1)} + R_{agg})/2$.

- network is on the "edge":

  Switch to controlling phase. The sending rate is recorded as the bottleneck capacity, $B = R_{(n-1)}$.

- network is congested:

  In this case, the sending rate should be increased to $R_{(n)} = (R_{(n-1)} + R_t)/2$.

Another possible scenario of the searching phase is that the BPC system may never reach the edge region. This may happen especially when the increasing phase ends up in the worst-case scenario. It may require more than one adjustment period for the BPC system to reduce the queue size along the path. As a result, the sending rate may still be

considered too large even it is already below the bottleneck link capacity and none of the three states above is met. The BPC system can tolerate this problem by putting the system into the “to be observed” state in Table 5.1. In this state, the controller does not change the sending rate until any of the conditions for the other three states is satisfied. The BPC system enters controlling phase, if the difference between $R_{(n)}$ and $R_{(n-1)}$ is smaller than 3% of $R_{(n-1)}$.

5.2.3 Controlling phase

The controlling phase has multiple entrances from different states of the increasing phase and searching phase. The goal of the controlling phase is to control the sending rate around the bottleneck capacity and cope with possible cross-traffic.

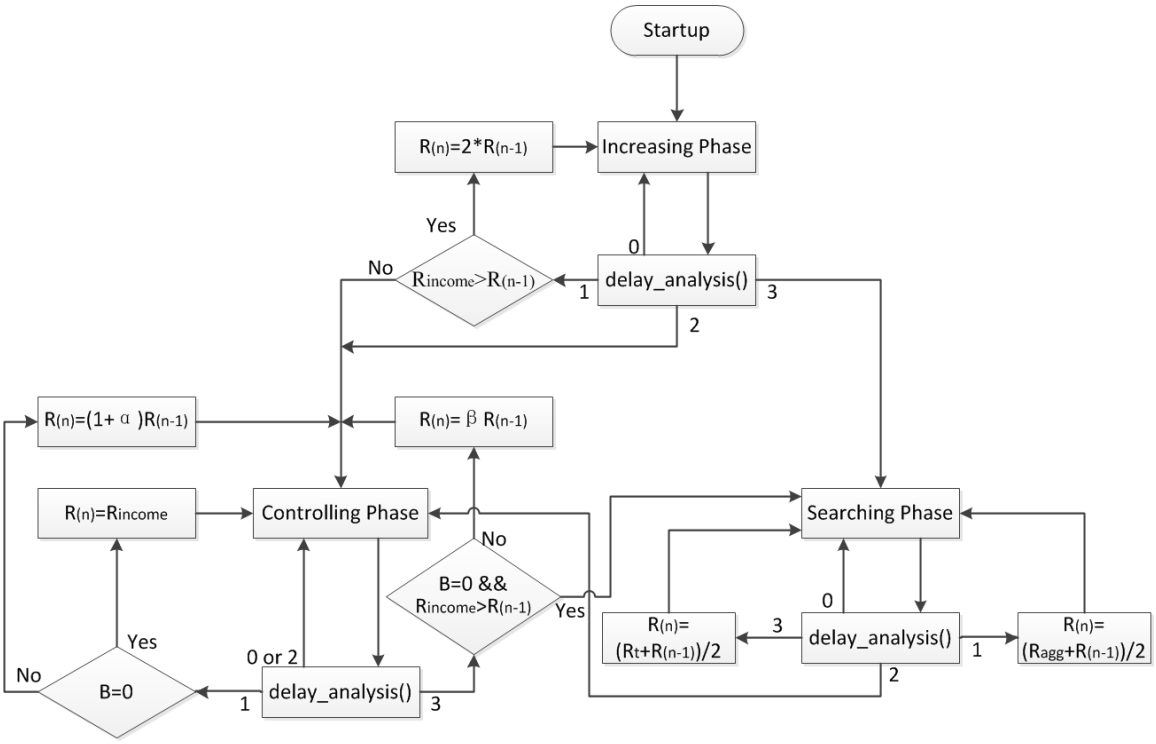

Figure 5.1 Transition among different network states

The noise introduced by cross-traffic as discussed in Chapter 4 may cause a different level of delay fluctuation depending on the nature and the rate of the cross-traffic. If the delay starts to fluctuate in the controlling phase, the controller adjusts the sending rate based an AIMD algorithm. However, different from the AIMD algorithm in TCP Reno, the BPC-AIMD algorithm uses $\alpha = 0.31$ and $\beta = 7/8$. This parameter setting is recommended in [15], which reports better performance and fairness to TCP Reno with this setting. The upper and lower bounds of the sending rate during the controlling phase are the bottleneck capacity B and the minimum value between $B/2$ and the incoming rate. This strategy is applied after the bottleneck capacity B is acquired in the searching phase.

However, an issue may occur if the BPC system enters the controlling phase from the underutilization state of the increasing phase when the incoming rate is small. Thus,

the system is unaware of the bottleneck capacity (i.e., $B = 0$), because the incoming rate is smaller than the bottleneck link capacity and the network was never overloaded. As a result, the strategy will be different from BPC-AIMD algorithm. The sending rate remains the same as the incoming rate, no matter whether the delay fluctuates or not. This is not completely fair to cross-traffic, since the sending rate may be higher than $B/2$ and cross-traffic shares less bottleneck link capacity than main-traffic. However, there is a compromise that has to be made. In the BPC system, the main-traffic has a higher priority when the bottleneck capacity is unkown. On the other hand, if the incoming rate changes, either by increasing or by decreasing, the sending rate is set directly to match the incoming rate. Then, in the next adjustment period, the network either remains in the same underutilization state (i.e., incoming rate is still smaller than the bottleneck capacity) or enters the other two states. If it is in the congestion state, it means the incoming rate now exceeds the actual bottleneck link capacity. The BPC system also switches from the controlling phase to the searching phase to figure out the bottleneck link capacity and finally returns to controlling phase.

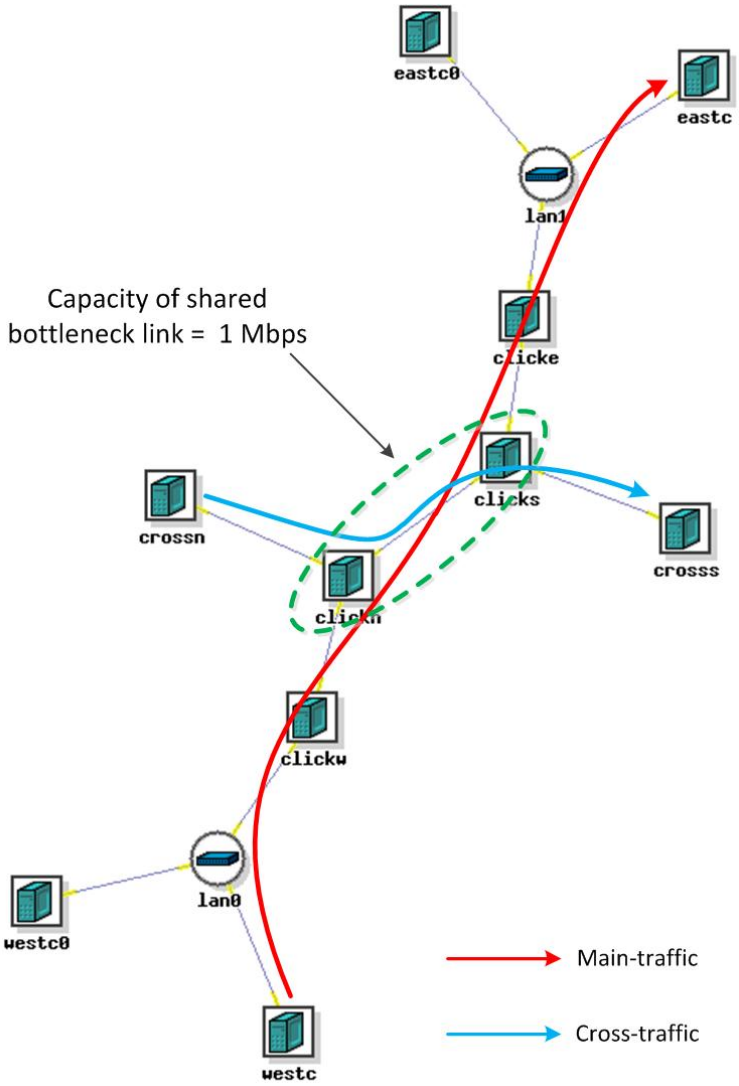

Figure 5.2 Experiments for the BPC system

Figure 5.1 shows the transitions among the different network states. The numbers from 0 to 3 indicate the different network states as listed in Table 5.1.

5.3 Experiments and results

The BPC system was tested under the same experimental environment on Emulab as described in Chapter 4. The main-traffic and the cross traffic are annotated in Figure 5.2. The network parameters also remain the same. The single-trip delay of the overlay link from *clickw* to *clicke* is set to 25 ms and the loss rate is set to 5% (without any network congestion and cross-traffic) in Emulab. We chose 160 Kbps as the initial value for the sending rate in the increasing phase.

To validate the ability of the BPC system to handle different network scenarios, seven experiments, which are representative of the range of possible network situations, are presented in this section.

5.3.1 Experiment 1: Single flow of main-traffic without cross-traffic

In the first experiment, the simplest scenario is that the main-traffic contains one flow with rate greater than the bottleneck capacity. The result of this experiment is shown in Fig. 5.3.

The sending rate starts to increase from 160 Kbps and exits the increasing phase at the value of 1280 Kbps. After this, the BPC system enters the searching phase to find out the bottleneck link capacity, and then switches to the controlling phase. Finally, the sending rate remains stable at 940 Kbps and recorded as the detected bottleneck link capacity B which is 94% of the real bottleneck link capacity. The single-trip delay in this experiment changes from around 30 ms to 400 ms and back to 30ms in the controlling phase. The average throughput of main-traffic computed by *iperf* in the controlling phase is around 900 Kbps.

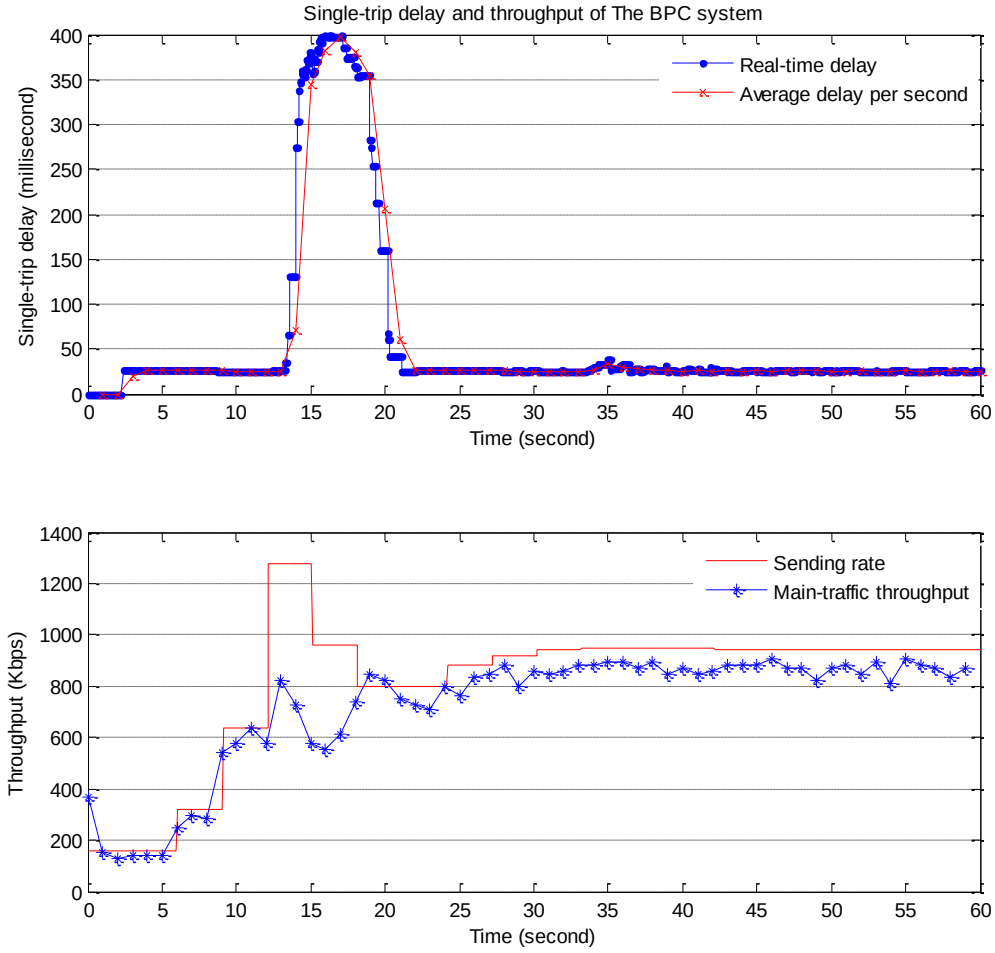

Figure 5.3 Result of experiment 1 with one flow main-traffic, none cross-traffic

5.3.2 Experiment 2: Multiple flows as main-traffic without cross traffic

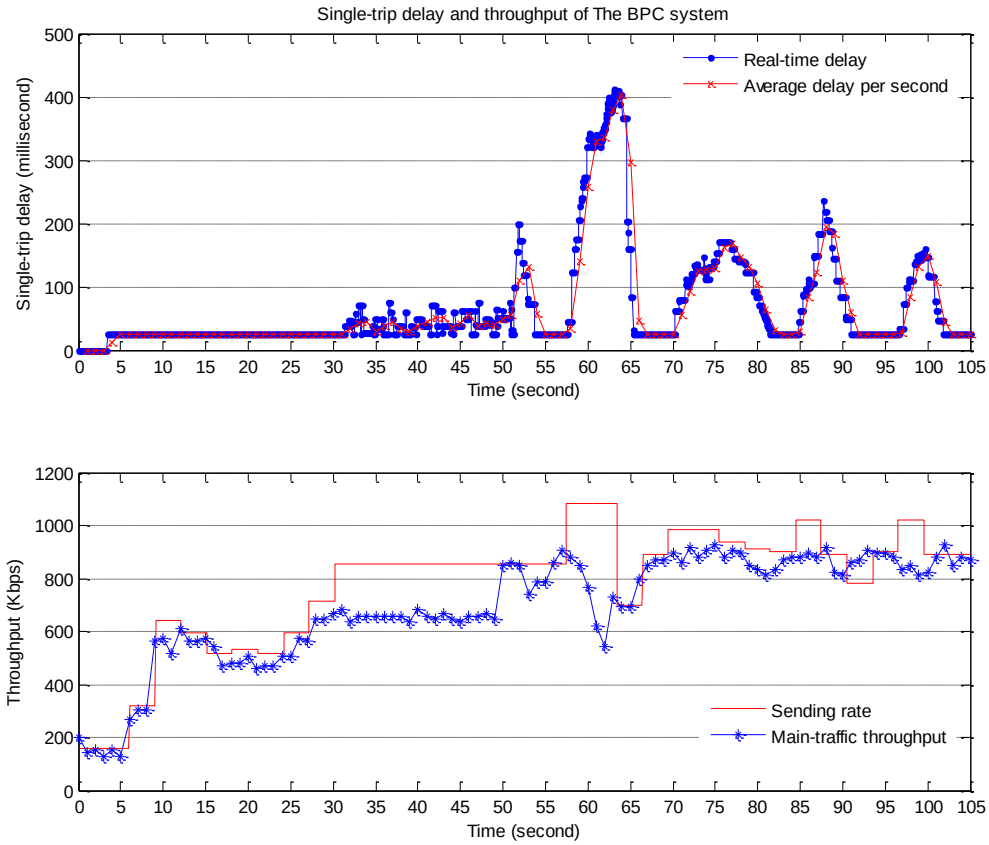

Figure 5.4 Result of experiment 2 with multiple flows of main-traffic, none cross-traffic

Another two flows are introduced as main-traffic in this experiment. Flow 1 is a UDP flow with sending rate 500 Kbps and starts at 0. Flow 2 is a UDP flow with rate 200 Kbps, which starts at time 30 seconds. Finally, flow 3 is another UDP flow with rate 600 kbps, starting at time 50 seconds. Results are shown in Figure 5.4.

Since the rate of the first flow is 500 Kbps, the BPC system enters the controlling phase directly from the increasing phase at around 12 seconds. At this time, the system is

unaware of the bottleneck capacity, which means $\mathrm{B} = 0$. From 12 to 20 seconds, the sending rate of the BPC-IN router is set to $1.2 \times R_{income}$. At the 20 second mark, flow 2, of rate 200 Kbps, is introduced. The introduction of this flow does not affect the system much, since the incoming rate of the two flows is still smaller than the bottleneck capacity. The delay monitored by the BPC system starts to fluctuate within a small range. Finally, the third flow of 600 Kbps comes in at the 50 second mark. We can see that the delay fluctuates much more drastically after the third flow is introduced. This indicates that the bottleneck is overloaded. Then the BPC system enters the searching phase from the controlling phase to determine what the bottleneck link capacity is.

After adjusting the sending rate for several adjust periods, at 78.5 seconds, the BPC system finally determines that the bottleneck link capacity is 913 Kbps. Then it enters the controlling phase again to maintain this value as the sending rate. As shown in Fig. 5.4, the throughputs of main-traffic during the two controlling periods are both very close to the theoretical values.

5.3.3 Experiment 3: Decreasing main-traffic without cross-traffic

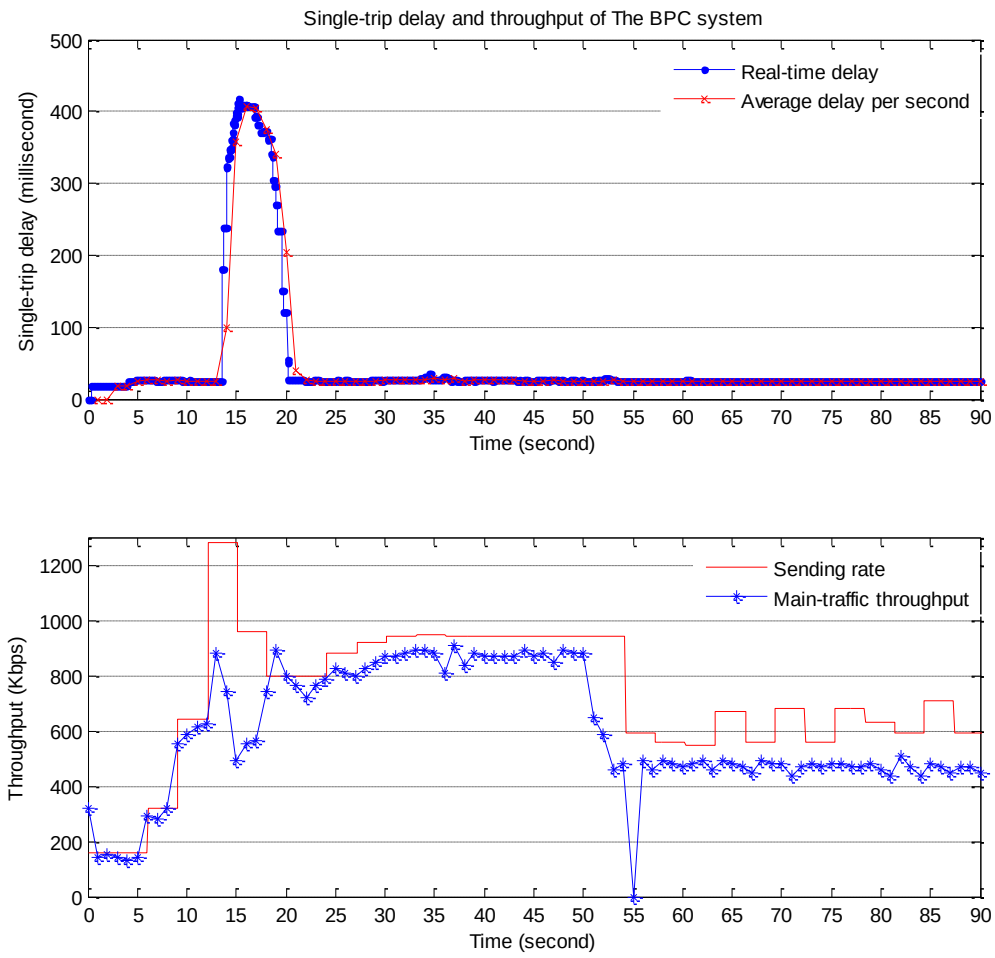

Figure 5.5 Result of experiment 3 of main-traffic with decreasing rate, none cross-traffic

Main-traffic contains two flows, each of which has a rate of 500 Kbps, at the beginning of experiment 3. Then one flow stops at 50 seconds, while remain flow continues. Fig. 5.5 shows the result of this experiment.

Before 50 seconds, the behavior of the system is basically the same as that in the first experiment. The bottleneck is detected to be 940 Kbps at around 36.1 seconds. After 50 seconds when one flow stops, the sending is reduced to match the current incoming rate. In addition, the throughput and delay follow their theoretical values.

5.3.4 Experiment 4: 1 Mbps main-traffic with 300 Kbps UDP cross-traffic

Cross-traffic is included in all of the next four experiments. In this experiment, cross-traffic of a 300 Kbps UDP flow is introduced at 50 second.

As shown in Fig. 5.6, bottleneck is detected as 940 Kbps at 30.1 seconds. The throughput stays around 900 Kbps until the cross-traffic starts at 50 seconds. At 50 seconds, because of the 300 Kbps cross-traffic, the single-trip delay is increased from 30 ms to 500 ms within 5 seconds and the throughput also drops to 300 Kbps. After the BPC system detects the significant increase of delay, the controller starts to decrease the sending rate based on the BPC-AIMD algorithm. Four periods later, the BPC system finds that the delay is back to *baseDelay* when the sending rate is reduced to 550 Kbps. The controller actually attempts to increase the sending rate later on to achieve better throughput performance without causing severe network congestion. The cross-traffic lasts for 30 second with a very stable throughput around 270 Kbps. After the cross-traffic disappears, the controller increases the sending rate gradually to the detected bottleneck link capacity. The throughput of main-traffic also comes back to 900 Kbps.

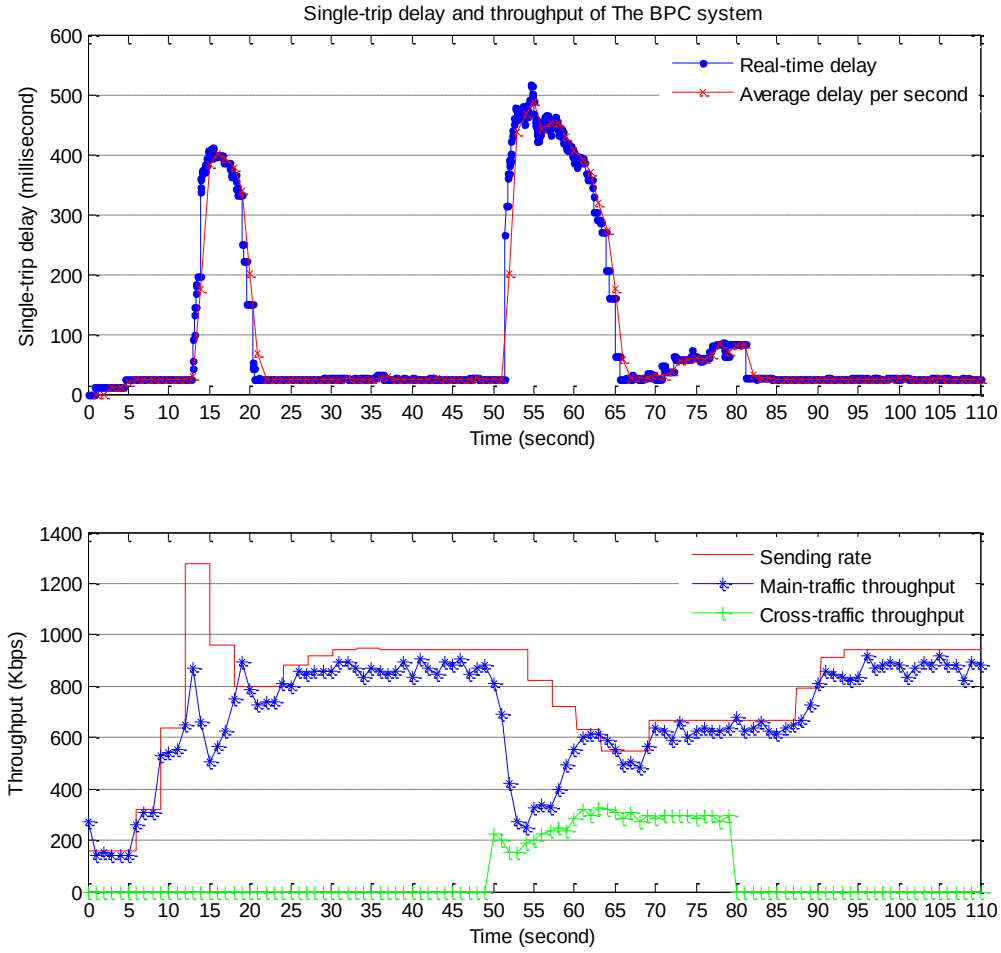

Figure 5.6 Result of experiment 4 with UDP cross-traffic

5.3.5 Experiment 5: 1 Mbps main-traffic with TCP cross-traffic

TCP cross-traffic is introduced in this experiment to test the fairness to legacy TCP flows of the BPC system. The main-traffic remains the same as in the fourth experiment.

From the result shown in Fig. 5.7, we can see that the BPC system acquires the bottleneck link capacity of 940 Kbps as usual at 20 seconds. After 50 seconds, the TCP cross-traffic enters the network. Although the delay does start to fluctuate 4 seconds later, the TCP flow does not obtain much network bandwidth until 60 seconds when the sending rate of the BPC system is reduced by more than 200 Kbps. This is because the window-based mechanism makes the TCP flow less aggressive than other rate-base data flows. However, the main-traffic controlled by the BPC system and the legacy TCP flow share the overlay link capacity equally.

5.3.6 Experiment 6: 500 Kbps main-traffic with 300 Kbps UDP/TCP cross-traffic

In the next two experiments, we explore the performance of BPC system with cross-traffic when the network is not overload by main-traffic. In this experiment, the rate of main-traffic is set to 500 Kbps throughout the experiment, while UDP cross-traffic is set to 300 Kbps.

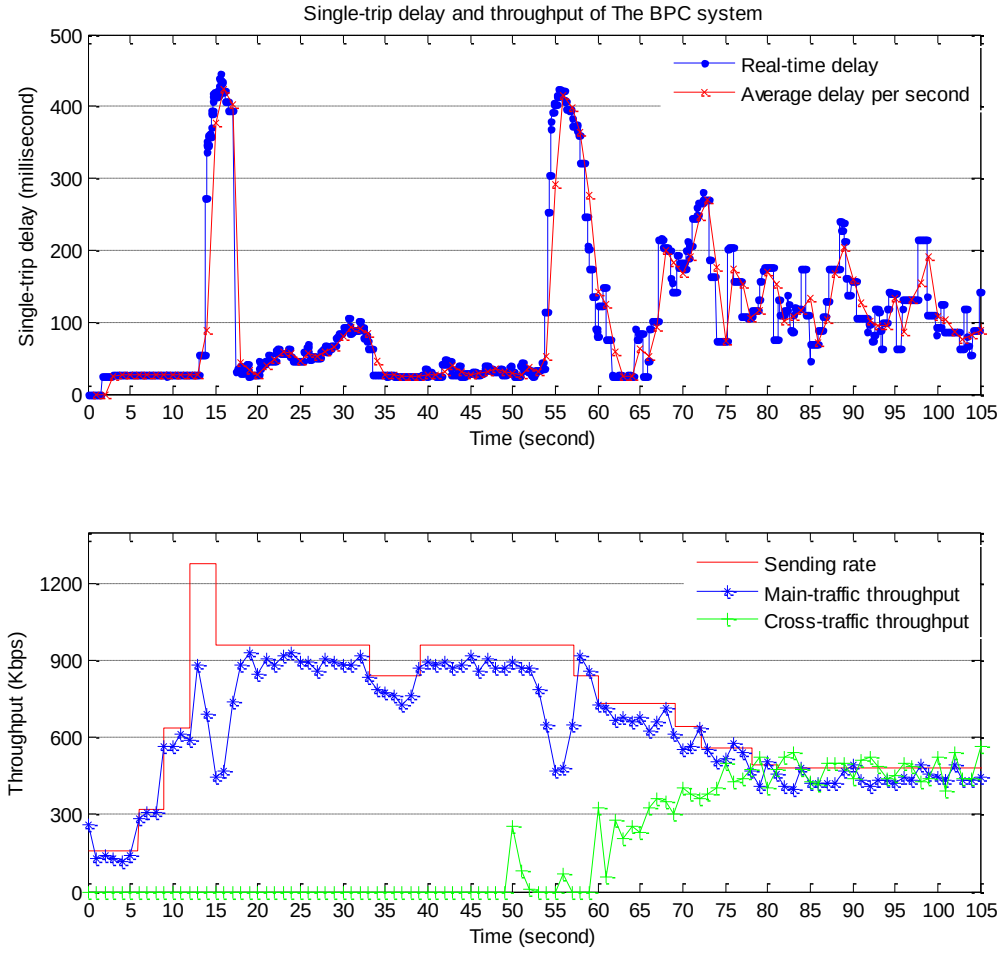

Figure 5.7 Result of experiment 5 with TCP cross-traffic

Fig. 5.8 shows the results for this experiment. Since the network is never overloaded by the main-traffic, the BPC system does not know about the bottleneck capacity. Thus, the sending rate follows the incoming rate. The reason the sending rate is higher than 500 Kbps in practice is that the BPC system need to leave some extra capacity for the possible

new flows. This does not affect the throughput of the main-traffic, which is 500 Kbps or above. The UDP cross-traffic is introduced at 30 seconds. Nothing is done by the BPC system at this point, although it indeed detects the fluctuation of delay. As we discussed

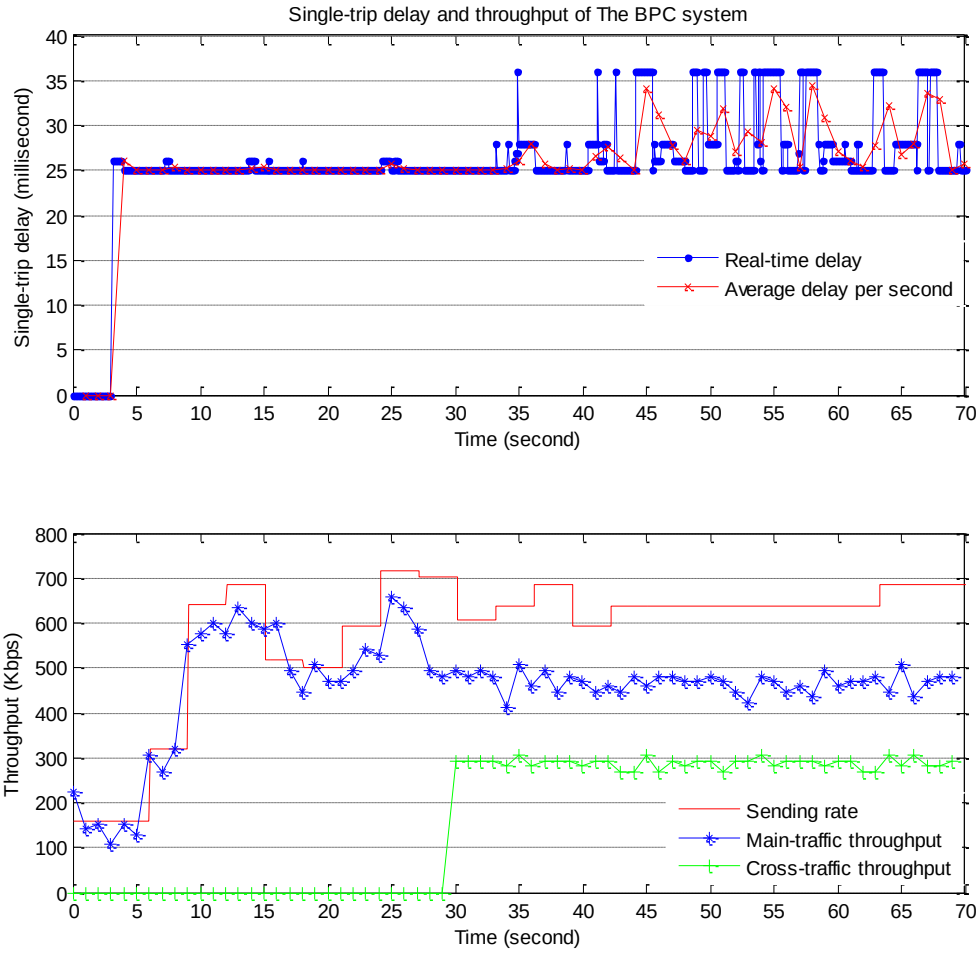

Figure 5.8 Result of experiment 6 with UDP cross-traffic

before, when the BPC system is not aware of the bottleneck capacity, it will maintain the incoming rate as its sending rate regardless of whether there is cross-traffic or not. The same result is observed in Fig. 5.9, where the UDP flow is replace by TCP flow.

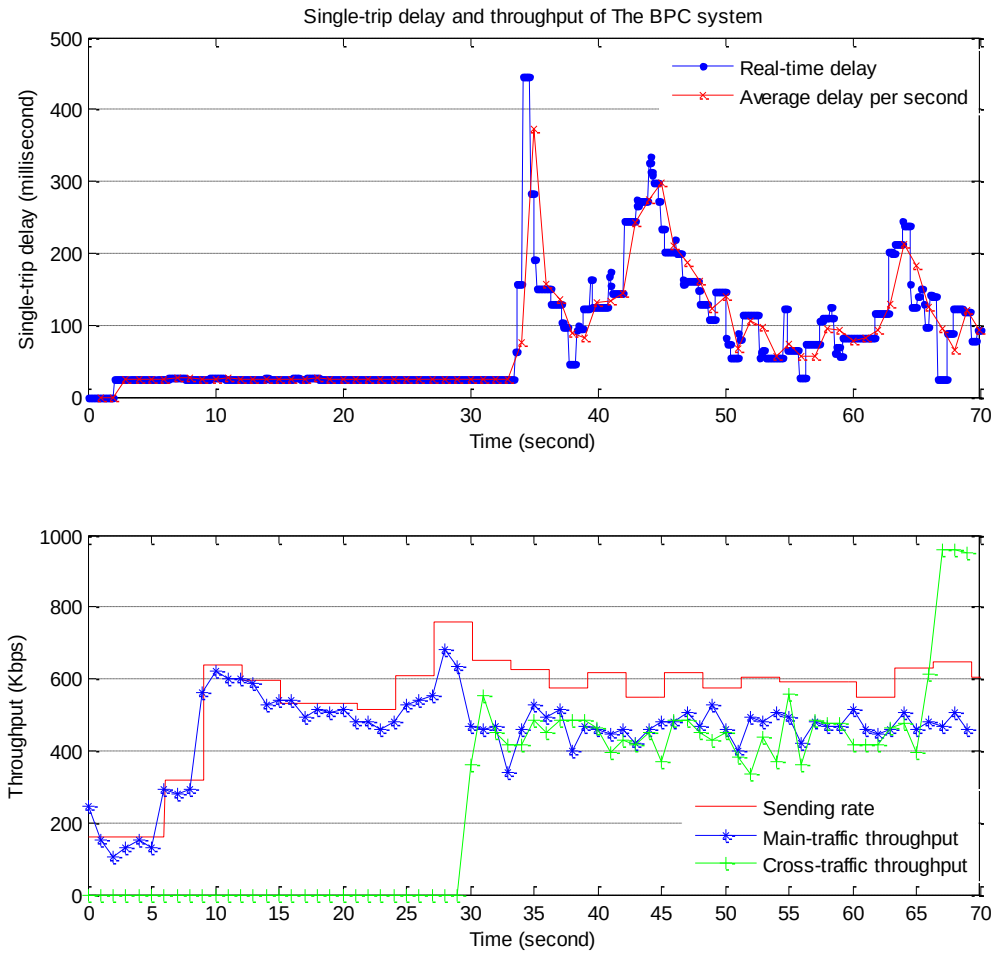

Figure 5.9 Result of experiment 6 withTCP cross-traffic

To provide a comparison with the performance of the legacy TCP flows, we conducted another experiment with only one TCP flow as main-traffic without any cross-traffic. Fig. 5.10 shows the results of this experiment. We note that the throughput of the TCP flow oscillates drastically from 220 Kbps to 1000 Kbps during the transmission. Therefore, the BPC system is able to provide a much better performance than legacy TCP in terms of both average and instantaneous throughput.

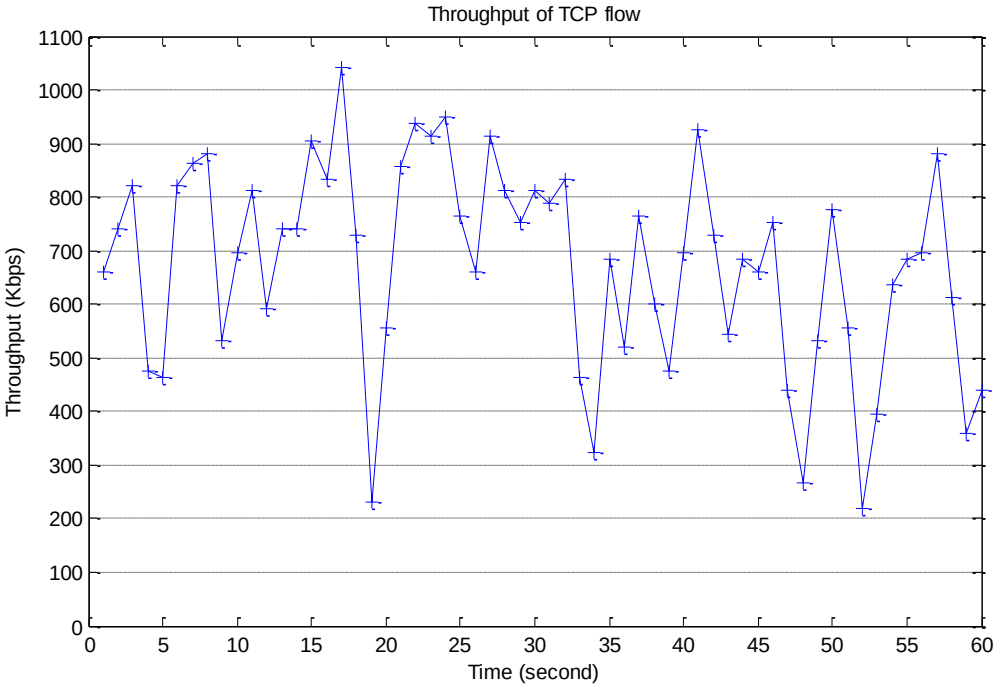

Figure 5.10 Throughput of TCP flow as main-traffic

5.4 Conclusions

Network utilization rate and fairness to legacy TCP flows are the two key issues for our new rate-based overlay congestion control algorithm. In this chapter, we explained in details how these two issues are addressed by the BPC system.

As shown in the results of the experiments above, the BPC system can achieve up to 94% of the network utilization rate. This value varies depending on the network environment and the setting of the parameters.

The BPC system basically divides data flows on the overlay link into two types. One is the data flows travels throughout the entire overlay link from the BPC-IN node to the BPC-OUT node. The other one is any data flows traverse a portion of the overlay link. The BPC system does not distinguish the constitution of these two types of traffic. The fairness the BPC system can achieve is that of main-traffic and cross-traffic. Thus, it is possible that different types of data flows within main-traffic or cross-traffic are not fair to each other. However, this is not the responsibility of the BPC system.

Chapter 6: Conclusions

Using delay information as a congestion signal can improve the ability of congestion control/avoidance algorithms to understand the network congestion significantly. Many approaches have been proposed to explore the possibility to take advantage of delay information. TCP Vegas simply compares the real-time RTT samples with the *baseRTT* to detect network congestion. FAST TCP and TFRC established a model of the network with loss event rate and RTT to compute a proper congestion window size or sending rate. Our approach includes several features that can utilize the delay information to improve the network performance.

The first feature is that we choose single-trip delay instead of round-trip time as the congestion signal. The delay on the reverse path, which contributes to the RTT, is not useful for evaluating the network congestion of forward path. Worse still, the reverse path delay may introduce destructive noise, if the reverse path is congested. This noise can lead to an inaccurate perception of the network status, which may guide the congestion control/avoidance algorithm to make incorrect decisions. In contrast, the single-trip delay excludes this possibility.

The second feature is we introduce a new approach evaluating delay information. We assess the correlation among delay samples over a certain period by average value, standard deviation and variation slope. Those parameters can reveal not only the absolute value but also the fluctuation level and the variation trend of the delay measurements. More effective information can be extracted from the delay by measurements using this feature.

The next feature is the introduction of the principle behind fuzzy logic controller. The fuzzy logic controller is widely used in many control scenarios, especially when it is difficult to create an accurate model. This is also the situation of today's Internet. The complexity of the Internet makes it too challenging to create an appropriate model. Our approach applies several fuzzy rules to determine the status of current network and make proper decisions.

The proposed BPC system is still not developed enough to be applied to a complex real-world network. Firstly, the accuracy of the single-trip delay relies on the resolution of the network synchronization protocols such as Network Time Protocol (NTP) and Simple Network Time Protocol (SNTP), particularly in a network with large physical distances. Secondly, parameter tuning is another concern. The parameters we choose to determine the network states are not universal to all other networks. A basic threshold has to be defined to derive the other parameters. However, this threshold currently relies on human experience. When applying our approach to a different network, a learning process is necessary to help the system operates effectively. Thirdly, when the bottleneck link capacity is unknown, the fairness our system provides to legacy TCP flows is not

perfect. The main-traffic always has priority to pass through the overlay link. An adaptive learning method can be developed to address those issues. This learning method should be able to study a new network without human interaction by monitoring the delay and throughput performance under different network loads.

Bibliography

Bibliography

[1] A. Bavier et al., “Increasing TCP Throughput with an Enhanced Internet Control Plane," in *Proc. IEEE Military Communications Conference (Milcom '06)*, Washington DC, Oct. 2006.

[2] B.L. Mark and S. Zhang, "A Multipath Flow Routing Approach for Increasing Throughput in the Internet," in *Proc. IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PacRim 2007)*, Victoria, Canada, Aug. 2007.

[3] L. G. Roberts, "QoS Signaling for IP QoS Support, version 2," Telecommunication Industry Association (TIA), Tech. Rep., April 2007.

[4] S. Floyd , “Metrics for the Evaluation of Congestion Control Mechanisms,” RFC 5166 (Informational), March 2008.

[5] S. Floyd,M. Handley, J. Padhye, and J.Widmer, “Equation-based congestion control for unicast applications,” in *Proc. ACM SIGCOMM*, Stockholm, Sweden, Sept. 2000.

[6] D. X. Wei, C. Jin, S. H. Low , S. Hegde, "FAST TCP: motivation, architecture, algorithms, performance," *IEEE/ACM Transactions on Networking (TON)*, vol.14, no.6, pp.1246-1259, Dec. 2006.

[7] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, “Modeling TCP throughput: A simple model and its empirical validation,” in *Proc. ACM SIGCOMM*, Vancouver, Canada, Aug. 1998.

[8] D. Katabi, M. Handley, and C. Rohrs, “Internet Congestion Control for High Bandwidth-Delay Product Networks,” in *Proc. ACM SIGCOMM*, Pittsburg, USA, Aug. 2002.

[9] S. Sohn, “Multipath and Explicit Rate Congestion Control on Data Networks,” Ph.D. Thesis, University of George Mason, May 2010.

[10] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click modular router," *ACM Trans. on Computer Systems*, vol. 19, no. 3, pp. 263-297, Aug. 2000.

[11] B. White et al., "An Integrated Experimental Environment for Distributed Systems and Networks," in OSDI02. Boston, MA: USENIX Association, pp. 255-270, Dec. 2002.

[12] R. Jain, “A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks,” *Computer Communication Review*, vol.19, no.5, p.56-71, Oct. 1989.

[13] L.S. Brakmo, S. O'Malley, and L.L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," *Computer Communication Review*, vol. 24, no. 4, pp. 24-35, Oct. 1994.

[14] J. Ahn, P. Danzig, Z. Liu, and L. Yan, "Evaluation of TCP Vegas: emulation and experiment," *Computer Communication Review*, vol. 25, no. 4, pp. 185-95, Oct. 1995.

[15] J. Martin, A. Nilsson, and I. Rhee, “Delay-based congestion avoidance for TCP,” *IEEE/ACM Trans. on Networking*, vol. 11, no. 3, pp. 356-369, June 2003.

[16] Y. R. Yang and S. S. Lam, “General AIMD congestion control,” In *Proc. International Conference on Network Protocols*, Osaka, Japan, Nov. 2000.

[17] http://iperf.sourceforge.net/

[18] http://code.google.com/p/iperf/

[19] L. A. Zadeh, “Fuzzy sets,” Inform. Contr., vol. 8, pp. 338-353, 1965.

[20] C. Douligeris and G. Develekos, “A fuzzy logic approach to congestion control in ATM networks," Proc. IEEE Int. Conf. Commun., vol. 3, pp. 1969-1973, 1995.

[21] S. Ghosh, Q. Razouqi, H.J. Schumacher and A. Celmins, “A survey of recent advances in fuzzy logic in telecommunications networks and new challenges,” IEEE Tran. on Fuzzy Systems, vol. 6, no. 3, pp. 443-447, Aug. 1998.

Curriculum Vitae

Lianjie Cao received the B.E. (Bachelor of Engineering) in Measuring and Control Technology and Instrumentations in 2008 from Huazhong University of Science and Technology. He then pursued the M.S. (Master of Science) in Computer Engineering in the Department of Electrical and Computer Engineering at George Mason University. He worked as a graduate research assistant at the Network Architecture and Performance Lab. His research interests lie in design, modeling and performance evaluation of broadband wired and wireless network architectures and protocols.