Computer Communications xxx (2010) xxx–xxx

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Preventing TCP performance interference on asymmetric links using ACKs-first variable-size queuing

Jiyong Park [a], Daedong Park [a], Seongsoo Hong [a,b,*], Jungkeun Park [c]

[a] School of Electrical Engineering and Computer Science, Seoul National University, Republic of Korea
[b] Department of Intelligent Convergence Systems, The Graduate School of Convergence Science and Technology, Seoul National University, Republic of Korea
[c] Department of Aerospace Information Engineering, Konkuk University, Republic of Korea

ARTICLE INFO

Article history:
Received 18 February 2009
Received in revised form 5 April 2010
Accepted 23 September 2010
Available online xxxx

Keywords:
Asymmetric link
TCP
Performance interference
Queue scheduling

ABSTRACT

In developing network-enabled embedded systems, developers are often forced to spend a great deal of time and effort analyzing and solving network performance problems. In this paper, we address one such problem: TCP performance interference on an asymmetric link. The upload or download throughput abruptly degrades if there is simultaneously upload and download TCP traffic on the link. While the problem has been addressed by many researchers, their solutions are incomplete as they only improve throughput in one direction, require TCP protocol modifications in end-user devices or are effective for a limited range of network configurations.

In order to overcome such limitations, we propose ACKs-first variable-size queuing (AFVQ) for a gateway. In doing so, we have derived an analytic model of the steady-state TCP performance with bidirectional traffic to clearly identify the two sources of the problem: the excessive queuing delay of ACK packets and the excessive number of ACK packets in the queue. Our AFVQ mechanism is designed to directly eliminate the two causes. Specifically, we have based AFVQ on two policies. First, ACKs-first scheduling is used to shorten the queuing delay of ACK packets. Second, the queue size for ACK packets is dynamically adjusted depending on the number of data packets queued in the gateway so that the number of ACK packets is reduced when packets are congested in the gateway. By applying the two policies simultaneously at the uplink and downlink output queue in the gateway, AFVQ achieves balanced TCP throughput improvements in both directions. In this way, it breaks circular dependencies between upload and download traffic.

We have implemented AFVQ in our ADSL-based residential gateway using the traffic control module of the Linux kernel. Our gateway yields 95.2% and 93.8% of the maximum download and upload bandwidth, respectively. We have also evaluated the proposed mechanism using the ns-2 simulator over a wide range of network configurations and have shown that AFVQ achieves better upload and download throughput than other representative gateway-based mechanisms such as ACQ, ACKs-first scheduling and ACK Filtering.

1. Introduction

As embedded systems are required to provide diverse network connectivity, a network subsystem has become an indispensable component in embedded systems design. Unfortunately, it is very challenging to realize network protocols in embedded systems and analyze their network performance since embedded systems usually operate in unconventional network configurations and under tight resource constraints. This causes unexpected interrelationships among network devices and protocols which in turn have a non-linear effect on network performance. Moreover, network performance problems should be carefully addressed since a straightforward solution approach may require modifications of the protocol standards or the end-user devices which developers cannot amend in most cases. As a result, developers often have to put a large amount of additional time and effort into getting the required network performance, even after they have completed development.

In this paper, we address an instance of network performance problems in a residential gateway when it is connected to an asymmetric link. A residential gateway is an embedded device that connects a home network to the Internet [1]. It provides various additional services such as security, authentication, content management, remote control and network resource management. As such, a residential gateway is a highly complex device that

* Corresponding author at: School of Electrical Engineering and Computer Science, Seoul National University, Republic of Korea. Tel.: +82 2 880 8370; fax: +82 2 882 4656.
E-mail address: sshong@redwood.snu.ac.kr (S. Hong).

0140-3664/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2010.09.010

provides composite features. For simplicity, we hereon refer to a residential gateway as a gateway.

Usually, a gateway is connected to the Internet using an asymmetric link technology such as Asymmetric Digital Subscriber Line (ADSL) [2], Data Over Cable Service Interface Specifications (DOCSIS) [3] and Passive Optical Networks (PONs) [4]. They are called asymmetric since more bandwidth is allocated to the downlink than the uplink. This decision is made based on the observation that most Internet traffic patterns favor download traffic. This efficient use of the limited total bandwidth makes the technology cost effective. Consequently, most of the residential Internet accesses are serviced via the asymmetric link technology.

However, as new types of devices and application domains emerge, upload traffic in home networks is also rapidly increasing. For example, devices such as interactive TVs, IP phones and web cameras frequently upload video and audio data across the Internet. Home PCs also incur a significant amount of upload traffic as new communication applications such as Peer-to-Peer (P2P) file sharing and webcasting become popular. As a result, upload speed as well as download speed is becoming an important factor in determining network performance.

The increased upload traffic on an asymmetric link may cause an unexpected interference in the presence of congestion since existing transport protocols such as TCP [5] are not designed to handle asymmetric traffic patterns. We call this the *TCP performance interference problem on an asymmetric link* where the traffic speed in one direction is interfered with by traffic in another direction. For simplicity, from now on we will refer to this as the *performance interference problem*.

1.1. Related work

Similar performance problems have been anticipated in the literature since early 1990s [6–14]. They examined the upload or download throughput degradation of TCP traffic over asymmetric links such as ADSL, cable and satellite link. Authors in [8,11] pointed out that the bandwidth asymmetry causes a phenomenon called *ACK starvation* in that ACK packets are delayed by data packets on a slower link. TCP performance degradation is also observed on a wireless link with a high bit-error rate such as satellite links [13,14].

Followed by such observations and analyzes, solution mechanisms have been proposed. Widely known solutions are ACC [8], ACE [15], ACK Filtering [8], SAD/AR [16], SACK Filtering [17], ACKs-first scheduling [8,18], REFWA [19], TCP sender adaptation [8], TCP New Jersey [20,21], ACQ [22,23], VAQ [22], DBCQ [24] and DQM [25].

ACC, ACE, ACK Filtering, SAD/AR and SACK Filtering attempt to reduce bandwidth consumed for transmitting ACK packets so that more bandwidth is provided for data packets. Particularly, they try to do so by limiting an ACK generation rate at an end-user device or selectively dropping ACK packets at a gateway. However, this decreases the download throughput since the queuing delay of ACK packets in the slow uplink is increased.

ACKs-first scheduling tries to shorten the queuing delay of ACK packets in the slow uplink by transmitting them prior to data packets in a gateway. Although this improves the download throughput, it has a downside as well. If the asymmetry between the uplink and downlink bandwidth is large, the upload output queue is flooded by ACK packets and thus the upload throughput is seriously compromised.

TCP New Jersey, TCP sender adaptation and REFWA are congestion control mechanisms that can also be used to alleviate the ACK starvation problem. They dynamically limit the transmission rate of data packets towards the slow uplink depending on the congestion level or the available bandwidth of the link. As a result, ACK packets for download traffic can avoid starvation but only at the cost of reduced upload throughput.

To avoid this problem, ACQ, VAQ, DBCQ and DQM store ACK and data packets in separate queues and schedule them using scheduling algorithms. While this prevents both ACK and data packets from being starved, it increases the average queuing delays of both ACK and data packets. This, in turn, degrades the throughput in both directions. RFC 3449 [26] covers a complete list of existing solutions and detailed comparisons among them.

None of the existing solutions are complete in the sense that they can improve the TCP performance only in one direction or are effective for a limited range of network configurations with different asymmetry ratios and Internet delays. Moreover, many of the solutions make use of per-connection data structures, weighted fair queuing and rate control mechanisms. Since these incur non-trivial run-time memory and performance overheads, they are not suitable for resource-constrained residential gateways. More importantly, some of the existing solutions cannot be applied to gateways since they require modifications to the TCP protocol stack implementations on end-user devices.

In this paper, we propose the ACKs-first variable-size queuing (AFVQ) mechanism that overcomes the above limitations of the existing solutions. In doing so, we have derived an analytic model for the steady-state TCP performance with bidirectional traffic to clearly identify the sources of the performance interference problem, namely the excessive queuing delay of ACK packets and the excessive number of ACK packets in the queue. Our AFVQ mechanism is designed to directly eliminate these two causes. Specifically, we base AFVQ on two policies. First, ACKs-first scheduling is used to shorten the queuing delay of ACK packets. Second, the queue size for ACK packets is dynamically adjusted depending on the number of data packets queued in the gateway so that the number of ACK packets is reduced when packets are congested in the gateway. The two policies are designed such that they change only the queuing policy in the gateway and do not rely on any per-connection data structures or rate control algorithms. Clearly, AFVQ is a gateway-only solution that improves TCP performance in both directions on an asymmetric link. It is also effective for a wide range of asymmetry ratios and Internet delays and does not incur significant run-time overheads since it does not rely on costly run-time mechanisms such as those mentioned above.

We have implemented the AFVQ mechanism in our ADSL-based residential gateway using the traffic control module of the Linux kernel. In order to show that AFVQ improves the TCP performance in both directions and is effective for a wide range of networks, we have conducted a series of experiments both in a real-world and simulated networks. The real-world experiments were performed across a home network and the Internet that were interconnected via a commercial ADSL link. Our gateway yields 95.2% and 93.8% of the maximum download and upload bandwidth, respectively. We have also evaluated the proposed mechanism using the ns-2 simulator over a number of different network configurations and shown that AFVQ achieves better upload and download throughput than other representative gateway-based mechanisms such as ACQ, ACKs-first scheduling and ACK Filtering.

1.2. Organization of the paper

The rest of the paper is organized as follows. In Section 2, we give the implementation of the residential gateway and its operating environment. Then the network performance interference problem is described in detail. Section 3 provides the network model of the gateway using an asymmetric link. In order to aid in the understanding of the rest of the paper, we also give an overview of the TCP congestion control mechanism. In Section 4, we derive an analytic model of the steady-state TCP throughput in

Please cite this article in press as: J. Park et al., Preventing TCP performance interference on asymmetric links using ACKs-first variable-size queuing, Comput. Commun. (2010), doi:10.1016/j.comcom.2010.09.010

our network configuration and provide our solution strategy. In Section 5, we present the AFVQ mechanism as a solution mechanism. Section 6 provides experimental results. Finally, Section 7 serves as our conclusion.

2. Problem description

Linux is one of the most suitable operating systems for implementing network devices due to the proliferation of supported protocols and versatile traffic control mechanisms. Unfortunately, the straightforward adoption of the native Linux network subsystem may incur severe performance problems depending on traffic patterns, workloads and network configurations. Our ADSL gateway also experiences a performance interference problem in that the upload or download speed is abruptly reduced when there is simultaneous upload and download TCP traffic on an ADSL link. In this section, we first explain the implementation of our Linux-based residential gateway and identify the performance interference problem. We then formulate the problem in terms of packet scheduling policies over the Linux traffic control mechanisms.

Table 1
Implementation of the residential gateway.

Hardware	Main processor	SoC based on ARM architecture (No MMU, clock speed: 40 MHz)
	Network processor	Proprietary ADSL DSP
	Memory	SDRAM: 8 Mbytes Flash memory: 2 Mbytes
	Network interfaces	ADSL and Ethernet
Software	Operating system	uClinux version 2.4.17 (Linux for MMU-less processor)
	Provided functionalities	Web server, NAT, SNMP, Firewall, IP-filtering, DHCP and etc.
	Protocol stacks	• RFC 2684 (bridged Ethernet) • RFC 2364 (PPP over AAL5) • RFC 1577 (IPv4 PDU)

2.1. Analyzing gateway implementation and identifying performance interference

Our gateway was developed using uClinux, a version of Linux specialized for MMU-less CPUs [27]. Table 1 summarizes the implementation of the gateway. It has one ADSL and one Ethernet interface. It is connected to the Internet through the ADSL interface and to the PC as well as various consumer devices through the Ethernet interface. Inside the gateway, there are three different protocol stacks, RFC 2684, RFC 2364 and RFC 1577. One of them is selected and used according to the gateway configuration. Though all three protocol stacks carry out the task of transferring a packet received from one network interface to another, the difference lies in how they analyze and handle packets.

The conventional structure and packet processing mechanism of the Linux network subsystem in the residential gateway is depicted in Fig. 1. It consists of three elements: (1) network device drivers, (2) the TCP network protocol stack and (3) an input queue and two output queues. There are two network device drivers, used for interfacing with the home network and the Internet, respectively. They perform two tasks: copying incoming packets into memory and copying outgoing packets stored in memory into the network interface. The network protocol stack does the actual packet processing. Specifically, it analyzes a packet header, discards a packet if its header matches one of the firewall rules, looks up the routing table, determines the network interface where the packet will be sent and modifies the packet header when Network Address Translation (NAT) is enabled. After being processed, a packet is stored in a specific memory location for transmission. The input and output queues are in-memory data structures used for storing packets in transit between the device drivers and the network protocol stack. Packets are queued and dequeued according to the FIFO policy. It is important to note that a separate output queue exists for each device driver whereas there is only one input queue for all of them. Thus, packets arriving at the gateway are processed one at a time in the FIFO order.

Fig. 2 shows the operating environment of the residential gateway. It consists of a local host, a residential gateway, the Internet and a remote host. The local host and the gateway are connected

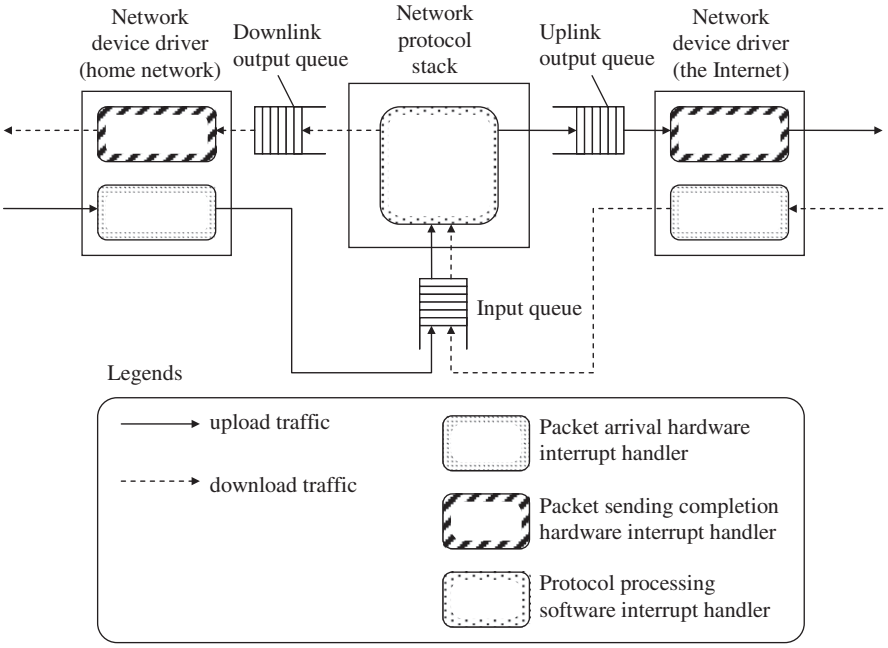

Fig. 1. The structure and packet processing mechanism of the conventional network subsystem in a residential gateway.

Please cite this article in press as: J. Park et al., Preventing TCP performance interference on asymmetric links using ACKs-first variable-size queuing, Comput. Commun. (2010), doi:10.1016/j.comcom.2010.09.010

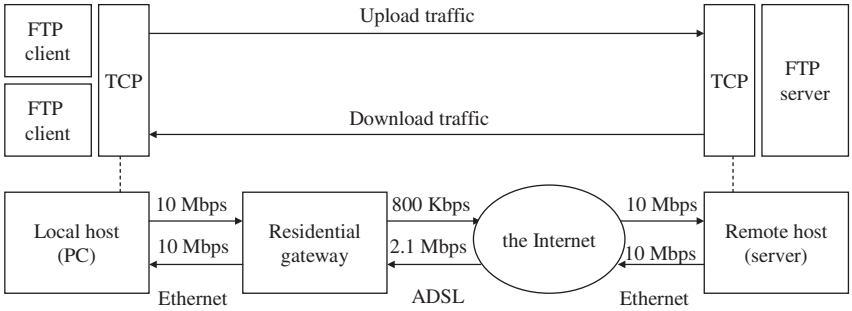

Fig. 2. Operating environment of the residential gateway for demonstrating performance interference problem. The lower part shows hardware configuration and the upper part shows software configuration.

via an Ethernet link and the gateway is connected to the Internet through an ADSL link. While the Ethernet is a symmetric link that offers the same speed in both directions, the ADSL is an asymmetric link that has different upload and download speeds. The upload speed from the gateway to the Internet is faster than the download speed in the opposite direction. For example, the ADSL2, which is one of the most widely used ADSL specifications, offers 12 Mbps for download traffic and 3.5 Mbps for upload speed.

In order to observe network performance degradation, we built an operating environment with the ADSL Lite [28] service that many Internet Service Providers (ISPs) offer. It offers 2.1 Mbps for download traffic and 800 Kbps for upload traffic. We used FTP programs to simultaneously generate continuous upload and download traffic. We executed a multi-threaded FTP server program in the remote host and two FTP client programs in the local host. They exchanged data using TCP. One FTP client sent a file to the server and the other got a file from the server. The file used in each transfer was an arbitrary file that was bigger than 100 Mbytes. Each client constantly re-transferred the same file. This configuration was created to reflect the characteristics of network applications such as P2P.We then measured the file transfer throughputs for upload and download directions in three cases: (1) only upload transfer is activated, (2) only download transfer is activated and (3) both transfers are activated. The results are given in Fig. 3. We can see that the upload speed is reduced abruptly from 750 Kbps to 250 Kbps when we activate transfers in both directions. This corresponds to a 67% decrease in speed. Contrary to our intuition, download speed is not affected.

The performance interference problem was observed when TCP was used regardless of other traffic patterns. We confirmed this by replacing the FTP programs by the TFTP [29] programs that use UDP and performing the same experiments. This revealed that UDP traffic alone did not incur any performance interference. However, we could observe the similar performance interference when we ran the FTP and TFTP programs together. In this case, the TCP throughput in both directions was reduced by the amount of bandwidth consumed for the UDP traffic. This was further confirmed by another experiment where UDP traffic generated by video streaming applications co-existed with TCP traffic.

In order to reveal the diverse behaviors of the problem, we also performed a number of experiments with different configuration parameters. Increasing the uplink output queue size from 100 to 200, we observed an opposite phenomenon: the upload speed remained the same while the download speed decreased by 56.1% from 1.8 Mbps to 790 kbps. This indicates that the performance interference problem could degrade the TCP performance in both directions depending on network configurations.

In line with the experimental results, we define the performance interference problem as below.

The performance interference problem is a phenomenon where the upload or download speed is abruptly reduced when there is simultaneous upload and download TCP traffic on an asymmetric link.

Clearly, it is very important to avoid the problem to smoothly support new applications in home networks.

2.2. Formulating network performance interference problem into packet queuing policy selection

The performance interference problem forces us to carefully re-engineer the network subsystem of the Linux-based gateway implementation. Since it is often considered as one of the most complicated components of the Linux kernel, it is critical to first characterize the network congestion model and then map it into a set of packet queuing policies that collectively define the entire packet handling process of a network subsystem. This implies that the network performance interference problem becomes a

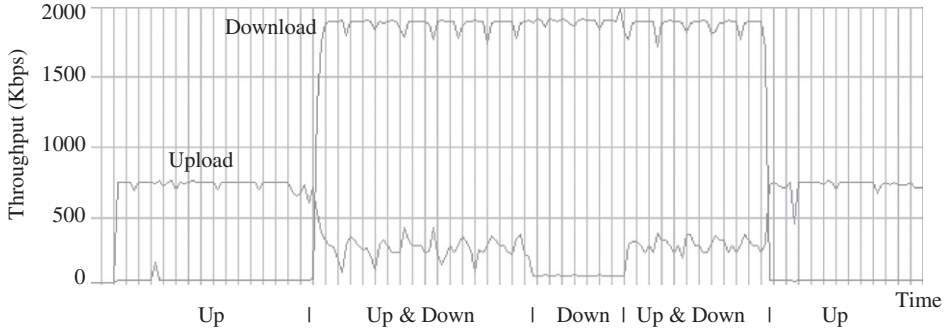

Fig. 3. File transfer throughput for the upload and download traffic according to the combinations of upload and download file transfer activities. "Up" and "Down" indicate upload and download file transfer is activated, respectively.

Please cite this article in press as: J. Park et al., Preventing TCP performance interference on asymmetric links using ACKs-first variable-size queuing, Comput. Commun. (2010), doi:10.1016/j.comcom.2010.09.010

problem of selecting packet queuing policies. We categorize the packet queuing policies in a top-down manner as listed below.

- Queue locations for applying new policies: We need to select where to apply our new queuing policies inside the network subsystem. As shown in Fig. 1, there are three candidate places, the input queue, the uplink output queue and the downlink output queue.
- Packet enqueing policy: After we determine the queue locations, we need to determine how packets are enqueued at each location. We may enqueue all packets into a single queue, or classify packets into several classes and store them into separate queues.
- Queue scheduling policy: In case of multiple queues, we need to conduct queue scheduling in order to pick a queue for packet dequeuing. Possible queue scheduling policies include round-robin scheduling, priority scheduling and weighted fair scheduling.

For each queue, following three policies should be further determined.

- Queuing policy: Queuing policy determines how packets are handled inside a queue after they are enqueued. We may use the default FIFO queuing or choose different queuing policies such as the token bucket queuing and the priority queuing.
- Queue size: Along with the queuing policy, we also have to determine the size of each queue. In Linux, the default size is 200 packets per queue. We can override the default value and even change the size dynamically.
- Drop policy: Each queue needs a drop policy. It determines which packets to drop when a queue is full. In Linux, the drop-tail policy is used by default. We may use other policies such as the drop-head, the Random Early Detection (RED) and so on.

After the above six policies are determined, we need to realize them into the Linux-based network subsystem. Fortunately, the Linux kernel provides for a clear separation of packet handling policies and mechanisms via the traffic control module [30]. It allows us to modify the configurations of the network input and output queues via a command-line tool at run-time. We can replace the default FIFO queue with others with different policies and construct multiple queues in a hierarchical manner.

3. Network model and TCP mechanism

Before analyzing the performance interference problem in detail, we first designed a network model in which the problem can be observed. In addition to the model, we also give an overview of the TCP congestion control and avoidance mechanisms in order to aid in the understanding of the performance analysis in the next section.

3.1. Network model

Our network model is a point-to-point network consisting of a local and a remote host. They are connected to each other via two logical links in opposite directions: a logical uplink and a downlink. A logical link is an end-to-end path covering the home network, the gateway and the Internet. Since we are looking for a gateway-only solution, we abstracted the home network and the Internet and simply model them as delays. We model the gateway in a logical link as a combination of a queuing system and a delay where the former models the output queue and the latter packet processing time in the gateway. We do not have to model the input queue of the gateway as a queuing system since its queuing delay is almost zero as explained in Subsection 2.1. We can simplify the entire model by adding the three different delays into a single propagation delay. The entire model is depicted in Fig. 4.

Our model is instantiated with four types of parameters: a transfer rate, a queue size, a propagation delay and a packet loss probability. The transfer rate denotes the speed of packet transmission from an output queue and the packet loss probability denotes the probability that a packet is lost on the path.

Table 2 summarizes notations used throughout the paper. In the table, upper case subscripts D and U denote a logical downlink and an uplink, respectively, and lower case subscripts d and u denote download and upload traffic, respectively. More notations are defined in following sections.

3.2. Overview of TCP congestion control and avoidance mechanisms

Generally, for reliable data transmission, a sender needs to wait for an acknowledgement (ACK) packet after transmitting a data packet. However, this stop-and-wait mechanism may well lead to low throughput. TCP uses a sliding window mechanism [31] that allows a sender to transmit multiple data packets without waiting for corresponding ACK packets. Since this may cause buffer overflows on the receiving side, a TCP receiver notifies the sender of its free buffer space via ACK packets. This is called a receiver window, denoted as *rwnd*, and is used to limit the number of unacknowledged data packets that the sender can send.

The sliding window mechanism was used in the early implementations of TCP but it caused the congestion collapse to the Internet gateways and routers due to excessive data packets. In order to rectify this problem, Jacobson [5] devised congestion control and avoidance mechanisms which are also known as TCP Tahoe. They were later enhanced and called TCP Reno [32,33] and TCP

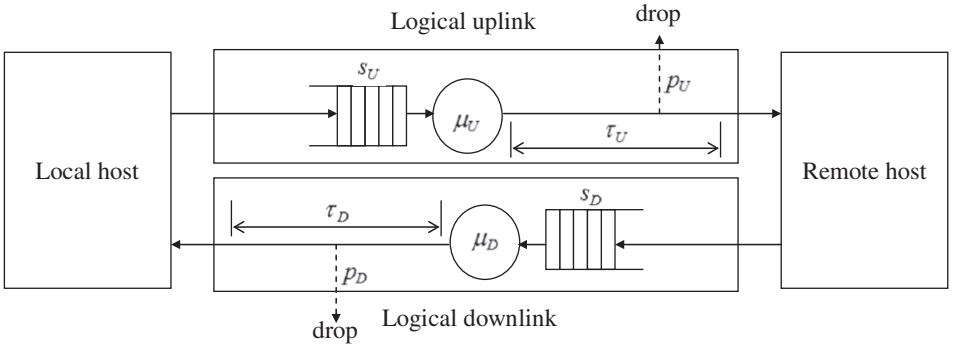

Fig. 4. The network model for the performance interference problem.

Please cite this article in press as: J. Park et al., Preventing TCP performance interference on asymmetric links using ACKs-first variable-size queuing, Comput. Commun. (2010), doi:10.1016/j.comcom.2010.09.010

Table 2
Symbols used in the paper. Upper case subscripts *D* and *U* indicate that a symbol is for logical downlink and uplink, respectively. Lower case subscripts *d* and *u* indicate that a symbol is for download and upload traffic, respectively.

Symbols	Meanings
μ_U, μ_D	Transfer rate of a logical link (in bytes per second)
s_U, s_D	Size of a queue (in number of packets)
τ_U, τ_D	Delay of a logical link (in seconds)
p_U, p_D	Packet loss probability of a logical link
$maxwnd_u$, $maxwnd_d$	Maximum windows size (in number of packets)
D	Size of a TCP data packet (in bytes)
A	Size of a TCP ACK packet (in bytes)
b	Number of data packets required for a TCP receiver to generate an ACK packet
λ_u, λ_d	Steady-state TCP throughput (in bytes per second)
RTT_u, RTT_d	Round-trip time of a TCP connection (in seconds)
$n_{q,u}^{data}$, $n_{q,d}^{data}$	Average number of data packets in the queue of the logical link
$n_{l,u}^{data}$, $n_{l,d}^{data}$	Number of data packets in the delay line of the logical link
$n_{q,u}^{ack}$, $n_{q,d}^{ack}$	Number of ACK packets in the queue of the logical link
$n_{l,u}^{ack}$, $n_{l,d}^{ack}$	Number of ACK packets in the delay line of the logical link
q_u^{data}, q_d^{data}	Queuing delay of data packets
q_u^{ack}, q_d^{ack}	Queuing delay of ACK packets

Vegas [34]. In the congestion control mechanism, the concept of a congestion window, denoted as *cwnd*, was introduced. It is the sender-side limit on the number of unacknowledged data packets. Therefore, a sender is allowed to send *wnd* = min(*cwnd*,*rwnd*) data packets without being acknowledged. The *cwnd* is a variable that is incremented each time a new ACK packet arrives. This allows a TCP sender to increase the packet transmission rate as it successfully receives ACK packets.

The limit that *wnd* cannot exceed is called 'maximum window size' and is denoted as constant *maxwnd*. In fact, it is equal to the size of the receive buffer in the TCP receiver since *wnd* is less than or equal to *rwnd*, which is at most equal to the size of the receive buffer. As in Table 2, we use $maxwnd_u$ and $maxwnd_d$ to denote the maximum window size of the upload and download traffic, respectively. The sizes of a data and an ACK packet are denoted as D and A, respectively.

As *cwnd* increases, the probability of losing data packets increases due to link errors or buffer overflows caused by congestion in a gateway or routers. A TCP sender detects the loss either via triple-duplicate ACK packets or timer expirations. When such a data packet loss is detected, the sender decreases *cwnd* in order to resolve the congestion. The exact amount of the increment and decrement of *cwnd* varies among different versions of TCP. Specifically, Additive-Increase/Multiplicative-Decrease (AIMD) [5] and its variants were used from the early versions of TCP, but they are being replaced by other policies such as Additive-Increase/Additive-Decrease (AIAD) [35]. However, since the difference does not affect our analysis, we do not provide any further details on the TCP congestion control mechanism.

4. Analytic modeling of the problem and deriving solution strategy

In this section, we analytically analyze the performance interference problem and derive our solution strategy. Since we are looking for a gateway-only solution, we derive relationships between the steady-state TCP throughput and queue parameters in the gateway. We derive two quadratic equations each of which is for the steady-state TCP throughput in upload and download direction, respectively. These equations reveal that the throughput can be controlled by two parameters in the gateway: the number of ACK packets in the output queue and the queuing delay of ACK packets. Specifically, the throughput can be increased close to the maximum link bandwidth by minimizing the two parameters. This result leads to our AFVQ mechanism.

We first derive an equation for the steady-state TCP throughput of the download traffic. There has been effort to analytically derive the steady-state TCP throughput [11,36–38]. We adopt the stochastic model presented in [38]. The steady-state download throughput λ_d is derived as

$$\lambda_d = \begin{cases} \frac{maxwnd_d D}{RTT_d}, & \text{if } maxwnd_d \leqslant E[wnd_d^u] \\ \frac{D}{RTT_d\sqrt{\frac{2bp_D}{3}} + T_0 \min\left(1, 3\sqrt{\frac{3bp_D}{8}}\right) p_D\left(1+32p_D^2\right)}, & \text{otherwise.} \end{cases} \tag{1}$$

As defined in the previous section, D, $maxwnd_d$ and p_D are the data packet size, the maximum window size and the packet loss probability, respectively. T_0, b and RTT_d are the initial timeout value, the number of data packets required for a TCP receiver to generate an ACK packet and the round-trip time for the download traffic, respectively. $E[wnd_d^u]$ is the mean value of the unconstrained window size that is given as below.

$$E[wnd_d^u] = \frac{2+b}{3b} + \sqrt{\frac{8(1-p_D)}{3p_D} + \left(\frac{2+p_D}{3p_D}\right)^2}. \tag{2}$$

We simplify Eq. (1) by introducing constants C_1 and C_2 below. Thus, we have

$$\lambda_d = \begin{cases} \frac{maxwnd_d D}{RTT_d}, & \text{if } maxwnd_d \leqslant E[wnd_d^u] \\ \frac{D}{RTT_d C_1 + C_2}, & \text{otherwise,} \end{cases} \tag{3}$$

where

$$C_1 = \sqrt{\frac{2bp_D}{3}} \text{ and } C_2 = T_0 \min\left(1, 3\sqrt{\frac{3bp_D}{8}}\right) p_D\left(1+32p_D^2\right). \tag{4}$$

In Eq. (3), RTT_d is the only unknown variable since D, $maxwnd_d$, b, p_D, T_0 and $E[wnd_d^u]$ are constants. Recall that RTT_d denotes the time duration from when the remote host sends a data packet to when the host gets an ACK packet acknowledging the data packet. As shown in Fig. 5, it can be decomposed into t_d^{data} and t_d^{ack}. The former is the delay of a data packet along the logical downlink and the latter is the delay of an ACK packet along the logical uplink. The delay along a logical link is further decomposed into a queuing delay, a transmission delay and a propagation delay. The round-trip time is thus

$$RTT_d = t_d^{data} + t_d^{ack} = \left(q_d^{data} + \frac{D}{\mu_D} + \tau_D\right) + \left(q_d^{ack} + \frac{A}{\mu_U} + \tau_U\right), \tag{5}$$

where q_d^{data} and q_d^{ack} are the queuing delay experienced by data and ACK packets for the download traffic, respectively. If we repeat the above formulation for the upload traffic, we can similarly derive equations for the upload throughput.

From Eqs. (3) and (5), it is easily inferred that the download throughput increases as q_d^{data} and q_d^{ack} decrease. Similarly, the upload throughput increases as q_u^{data} and q_u^{ack} decrease. However, since ACK and data packets share a common queue, increasing the upload throughput by decreasing q_u^{ack} and q_u^{data} may increase q_d^{data} and q_d^{ack}. This in turn decreases the download throughput. Therefore, it is necessary to clearly model circular dependencies between upload and download traffic in order to come up with a way to simultaneously increase both upload and download throughput.

We model the interference of the upload traffic on the download traffic. We first examine the downlink output queue where

Please cite this article in press as: J. Park et al., Preventing TCP performance interference on asymmetric links using ACKs-first variable-size queuing, Comput. Commun. (2010), doi:10.1016/j.comcom.2010.09.010

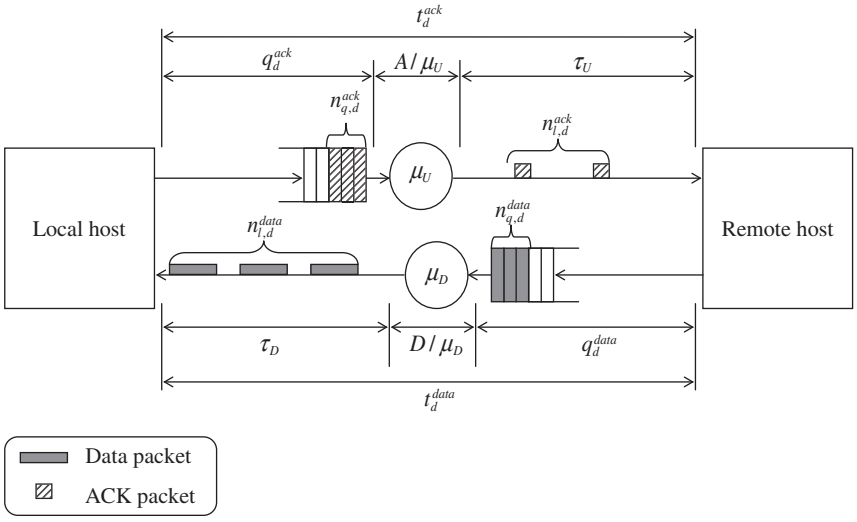

Fig. 5. The decomposed round-trip time and the number of packets for the download traffic.

download packets experience delay due to upload ACK packets. We denote the average numbers of data and ACK packets in the queue as $n_{q,d}^{data}$ and $n_{q,u}^{ack}$, respectively. The total amount of information in bytes in the queue is $D \cdot n_{q,d}^{data} + A \cdot n_{q,u}^{ack}$. It is transmitted at the speed of μ_D bytes per second. Thus, an incoming data packet experiences the following queuing delay on the average.

$$q_d^{data} = \frac{D \cdot n_{q,d}^{data} + A \cdot n_{q,u}^{ack}}{\mu_D}. \tag{6}$$

We elaborate on $n_{q,d}^{data}$ in Eq. (6). As shown in Fig. 5, the packets for the download traffic can be categorized into four groups: 1) data packets in the downlink output queue, 2) in-transit data packets in the delay line, 3) ACK packets in the uplink output queue and 4) in-transit ACK packets in the delay line. The number of packets in each group is denoted as $n_{q,d}^{data}$, $n_{l,d}^{data}$, $n_{q,d}^{ack}$ and $n_{l,d}^{ack}$. Subscripts q and l indicate a queue and a delay line, respectively. The packets that are currently being processed by the gateway are considered as still being queued. The first two groups contain data packets that have been transmitted but not acknowledged by the TCP receiver. Since a TCP receiver transmits an ACK packet for every b data packets received, $n_{q,d}^{ack} + n_{l,d}^{ack}$ ACK packets acknowledge $b\left(n_{q,d}^{ack} + n_{l,d}^{ack}\right)$ data packets. In most TCP implementations, b is greater than or equal to 2. As a result, the number of data packets that have been transmitted but whose acknowledgements have not been received is $n_{q,d}^{data} + n_{l,d}^{data} + b\left(n_{q,d}^{ack} + n_{l,d}^{ack}\right)$. By the definition of a window size in Section 3.2, this number is equal to current window size wnd_d for download traffic. Hence, we get

$$n_{q,d}^{data} + n_{l,d}^{data} + b\left(n_{q,d}^{ack} + n_{l,d}^{ack}\right) = wnd_d. \tag{7}$$

However, since we deal with the steady-state TCP throughput, we use average window size $avgwnd_d$ instead of wnd_d. Padhye et al. [38] show that the average window size is

$$avgwnd_d = \begin{cases} E[wnd_d^u], & \text{if } maxwnd_d \leqslant E[wnd_d^u] \\ maxwnd_d, & \text{otherwise.} \end{cases} \tag{8}$$

Therefore, the number of data packets in the downlink output queue is

$$n_{q,d}^{data} = avgwnd_d - n_{l,d}^{data} - b\left(n_{q,d}^{ack} + n_{l,d}^{ack}\right). \tag{9}$$

We analyze $n_{l,d}^{data}$ and $\left(n_{q,d}^{ack} + n_{l,d}^{ack}\right)$ in Eq. (9). First, we start with $n_{l,d}^{data}$. Since the download throughput is λ_d bytes per second and the size of a data packet is D bytes, data packets are pushed into the downlink delay line at the rate of λ_d/D packets per second. The number of data packets in the delay line is determined by using the bandwidth-delay product as follows

$$n_{l,d}^{data} = \frac{\lambda_d \tau_D}{D}. \tag{10}$$

We examine $\left(n_{q,d}^{ack} + n_{l,d}^{ack}\right)$. Since the ACK packets are on the logical uplink, we consider the entire link as a queue-less delay line whose propagation delay is t_d^{ack}. As mentioned above, the local host sends an ACK packet at every b data packets received. Therefore, the rate of ACK packets toward the uplink is λ_d/bD. Again, the bandwidth-delay product gives us the total number of ACK packets on the logical uplink as below.

$$\left(n_{q,d}^{ack} + n_{l,d}^{ack}\right) = \frac{\lambda_d t_d^{ack}}{bD}. \tag{11}$$

From Eqs. (5), (9), (10) and (11),

$$\begin{aligned} n_{q,d}^{data} &= avgwnd_d - \frac{\lambda_d \tau_D}{D} - \frac{\lambda_d t_d^{ack}}{D} \\ &= avgwnd_d - \frac{\lambda_d}{D}\left(\tau_D + \tau_U + q_d^{ack} + \frac{A}{\mu_U}\right). \end{aligned} \tag{12}$$

By plugging this result into Eqs. (8), (6), (5) and (3), we obtain the following equation for the download throughput λ_d:

$$\lambda_d = \begin{cases} \dfrac{maxwnd_d D}{\frac{D}{\mu_D}(maxwnd_d+1) + \frac{A \cdot n_{q,u}^{ack}}{\mu_D} + \left(\tau_D + \tau_U + q_d^{ack} + \frac{A}{\mu_U}\right)\left(1 - \frac{\lambda_d}{\mu_D}\right)}, \\ \quad \text{if } maxwnd_d \leqslant E\left[wnd_d^u\right] \\ \dfrac{D}{C_1\left\{\frac{D}{\mu_D}\left(E\left[wnd_d^u\right]+1\right) + \frac{A \cdot n_{q,u}^{ack}}{\mu_D} + \left(\tau_D + \tau_U + q_d^{ack} + \frac{A}{\mu_U}\right)\left(1 - \frac{\lambda_d}{\mu_D}\right)\right\} + C_2}, \\ \quad \text{otherwise.} \end{cases} \tag{13}$$

This equation is equivalent to quadratic equation $f_d(\lambda_d) = 0$ where

$$f_d(\lambda_d) = \begin{cases} X_1\lambda_d^2 - (D(maxwnd_d + 1) + X_2)\lambda_d + \mu_D maxwnd_d D, \\ \quad \text{if } maxwnd_d \leqslant E\left[wnd_d^u\right] \\ C_1 X_1 \lambda_d^2 - \{C_1(D(E\left[wnd_d^u\right] + 1) + X_2) + \mu_D C_2\}\lambda_d + \mu_D D, \\ \quad \text{otherwise} \end{cases} \tag{14}$$

Please cite this article in press as: J. Park et al., Preventing TCP performance interference on asymmetric links using ACKs-first variable-size queuing, Comput. Commun. (2010), doi:10.1016/j.comcom.2010.09.010

where

$$X_1 = \tau_D + \tau_U + q_d^{ack} + \frac{A}{\mu_U} \text{ and}$$
$$X_2 = \mu_D X_1 + A \cdot n_{q,u}^{ack}. \quad (15)$$

Download throughput λ_d is determined as one of the roots of Eq. (14). Upload throughput λ_u can be derived by following the same formulation steps as above. The equations for the upload throughput are the same as Eqs. (14) and (15) only with subscripts D and d being respectively changed to U and u, and vice versa. Therefore, we do not present the equations for the sake of simplicity.

Having derived the quadratic equations for the upload and download throughput, we present a solution strategy to maximize the throughputs simultaneously. Our strategy is based on the following theorem.

Theorem 1. *Root* $\lambda \in (0, \mu_D)$ *of* Eq. (14) *monotonically increases as* q_d^{ack} *and* $n_{q,u}^{ack}$ *decrease.*

Proof. The proof is given in Appendix. □

For the upload throughput, we can establish a similar theorem via the same proof steps; that is, the upload throughput monotonically increases as q_u^{ack} and $n_{q,d}^{ack}$ decrease.

According to these theorems, we can maximize the download and the upload throughput simultaneously by minimizing q_d^{ack} and $n_{q,d}^{ack}$ at the uplink output queue and q_u^{ack} and $n_{q,d}^{ack}$ at the downlink output queue. This requires us to minimize the queuing delay experienced by ACK packets and the number of ACK packets in each of the two output queues.

This result is consistent with our intuition. If the number of ACK packets in a queue decreases, the spare bandwidth can be used to transfer more data packets and this in turn increases the throughput for the opposite direction traffic. If the queuing delay of ACK packets decreases, so does the round-trip time. As discussed earlier, the throughput increases as the round-trip time decreases.

5. Realizing the AFVQ Mechanism into the Gateway

Based on our previous analysis, we devise the ACKs-First Variable-size Queuing (AFVQ) mechanism. We will first give an overview of the mechanism and then present its implementation in our gateway running Linux.

5.1. The AFVQ mechanism

AFVQ solves the performance interference problem by using two major policies: ACKs-first scheduling and variable-size queuing. The former reduces the queuing delay of ACK packets by separating ACK and data packets and transmitting ACK packets prior to data packets. The latter reduces the number of queued ACK packets by limiting the number of ACK packets in an output queue. This causes the dropping of excessive ACK packets and as a result more bandwidth is provided for transmitting data packets. Dropping ACK packets is allowed since ACK packets are cumulative. The newest ACK packet informs the TCP sender that all data packets were successfully transmitted even though some prior ACK packets may have been lost and not received. Unless all ACK packets are lost, the TCP sender does not reduce transmission speed. These policies are applied both at the uplink and the downlink output queue in the gateway. The input queue remains unmodified since the queue is almost empty and thus applying any queuing policy can hardly affect the entire packet handling process. Queuing seldom occurs at the input queue because packets inside the queue is dequeued by the CPU which processes the packets much faster than the packet arrival rate.

Fig. 6 depicts how our policies are realized in AFVQ. It shows the packet handling process in one of the two output queues. It consists of three separate FIFO queues, a packet classifier, a priority scheduler, a round-robin scheduler and a queue size modifier. The three separate queues and the packet classifier are used to isolate TCP ACK, TCP data and non-TCP packets from each other. When a packet is delivered from the network protocol stack, the packet classifier inspects the packet header and stores the packet into the corresponding queue according to the header type. Since one of our goals is to avoid the use of per-connection data structures that cause run-time memory overheads, the classifier does not distinguish packets for different TCP connections.

The priority scheduler realizes the ACKs-first scheduling policy. The ACK queue gets the high priority while the other two queues get the low priority. Packets in the two queues can be selected for transmission only when the ACK queue is empty. The round-robin scheduler is used to provide the equal amount of bandwidth to the TCP data queue and the non-TCP queue. This is to prevent the TCP data packets from being starved by non-TCP packets. Since non-TCP protocols such as UDP do not have a rate control mechanism, non-TCP packets tend to consume the bandwidth as much as possible and thus remain no bandwidth for TCP data packets.

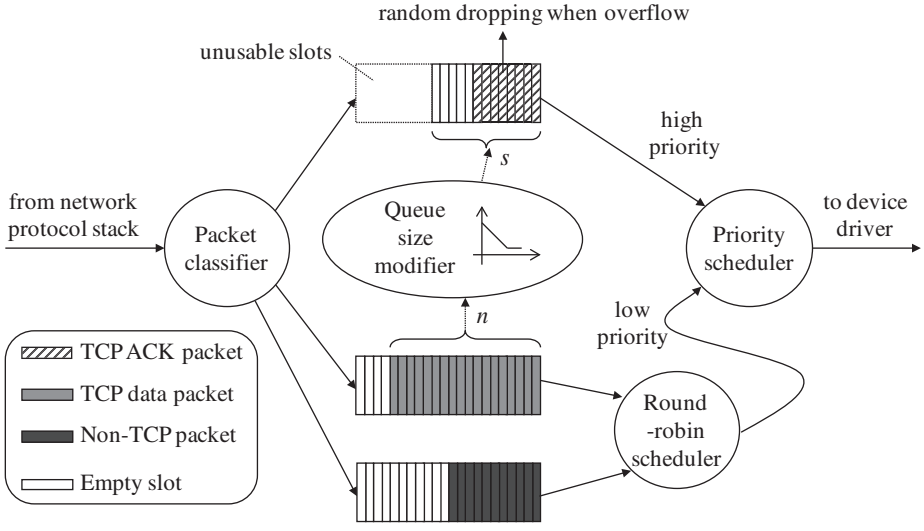

Fig. 6. The structure and the packet handling process of the AFVQ mechanism.

Please cite this article in press as: J. Park et al., Preventing TCP performance interference on asymmetric links using ACKs-first variable-size queuing, Comput. Commun. (2010), doi:10.1016/j.comcom.2010.09.010

The queue size modifier realizes the variable-size queuing policy. It adjusts the size of the ACK queue depending on the number of packets stored in the TCP data queue. As the number of queued TCP data packets increases, the modifier gradually shrinks the ACK queue. When the number exceeds a threshold, the ACK queue size is set to its minimum 1. At least one ACK packet needs to be queued not to lose all ACK packets. The behavior of the modifier is expressed below.

$$s = \begin{cases} \lceil -\frac{s_{max}}{n_{thr}} n + s_{max} \rceil & \text{if } 0 \leqslant n < n_{thr} \\ 1 & \text{otherwise} \end{cases} \quad (16)$$

Here, n is the number of packets in the TCP data queue and s is the size of the ACK queue. s_{max} and n_{thr} are the maximum size of the ACK queue and the threshold, respectively.

When the ACK queue overflows, we use the drop-random policy in which a randomly selected packet is dropped from the queue. This policy is preferred to the ordinary drop-head and drop-tail policies where the oldest and the newest packets are dropped, respectively. This is because the drop-random policy prevents an acknowledgement number seen by the TCP sender from being increased abruptly whereas the other two policies do not. Such an abrupt increase in an acknowledgement number is undesirable since it leads to a sudden fluctuation of the number of transmitted data packets and may cause buffer overflows at routers [39]. Table 3 summarizes the complete list of policies explained so far.

5.2. Implementing AVFQ in the residential gateway

As described in Section 2.1, since AFVQ is a gateway-only solution, the mechanism could be implemented by only changing the queuing policy at the two output queues in the residential gateway. Specifically, we modified each output queue using the traffic control module in the Linux kernel as shown in Fig. 7. Our mechanism is implemented across the kernel-space and the user-space. In the kernel-space, we add a CBQ queuing discipline at the top level. The top-level queuing discipline provides a packet enqueing and dequeing interface for the protocol stack. Inside the queuing discipline, we further add three pFIFO queuing disciplines to the CBQ for TCP ACK packets, TCP data packets and non-TCP packets. In the traffic control module, a pFIFO queuing discipline implements a FIFO queue with the drop-tail policy. Since we use the drop-random policy for ACK packets, we modify the existing pFIFO to support both policies. For separating the three types of packets, we add a filter object and configure it to forward each incoming packet to the corresponding pFIFO queuing discipline. We implement the priority scheduler and the round-robin scheduler by associating each pFIFO queuing discipline with a CBQ class. Specifically, we assign the highest priority to the CBQ class for ACK packets than the CBQ class for the other two. The CBQ classes for TCP data packets and non-TCP packets get the equal weight of 0.5 to share the bandwidth equally.

In the user-space, we implement the queue size modifier as a shell script. It uses the tc command-line program [30] to monitor and control the two pFIFO queuing disciplines for TCP ACK and data

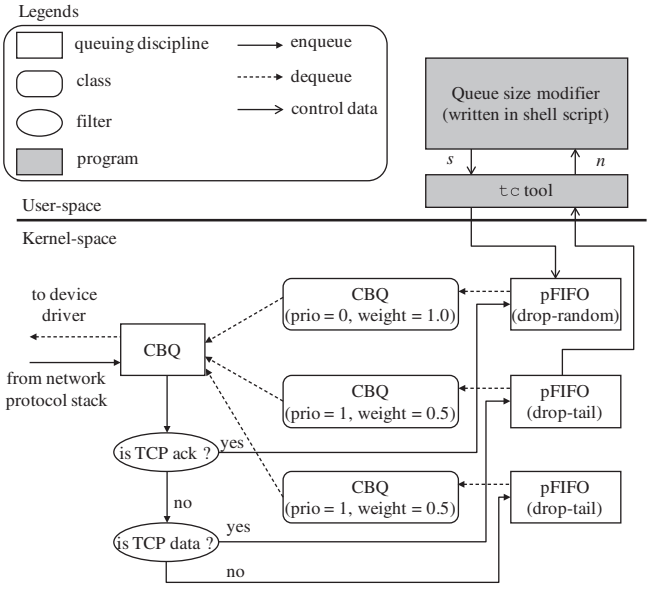

Fig. 7. Implementation of the AFVQ mechanism in the residential gateway.

packets. It reads the number of queued TCP data packets and configures the size of the queue for ACK packets. Since the tc tool uses system calls to enter the kernel, the monitoring and controlling are expensive operations. However, the overall overheads are insignificant since the queue size modifier runs slowly at the rate of 1 Hz. The rate can also be adjusted depending on the load of the gateway.

6. Experimental results

As clearly explained in previous sections, AFVQ is a gateway-only solution since it does not attempt to modify protocol stack implementations on end-user devices and is entirely implemented within a gateway. Moreover, it runs very efficiently in a gateway since it does not rely on costly per-connection data structures or any expensive run-time mechanisms. In this section, we attempt to show the most important property of AFVQ. That is, we demonstrate that it actually solves the performance interference problem or significantly improves the TCP performance in both directions on an asymmetric link. Also, we show that the AFVQ is effective over a wide range of asymmetry ratios and Internet delays while achieving better upload and download throughput than other representative gateway-based mechanisms, such as ACQ, ACKs-first scheduling and ACK Filtering. To do so, we have conducted experiments with the AFVQ implementation in our ADSL-based residential gateway and performed a series of simulations.

6.1. Real-world experiments

Real-world experiments were performed in a network configuration similar to Fig. 2 in Section 2. It consisted of a local host, a home network, a residential gateway, the Internet and a remote

Table 3
List of packet queuing policies used in AFVQ.

Queue locations	Uplink output queue and downlink output queue
Packet enqueing policy	Packets are classified into TCP ACK, TCP data and non-TCP packets and stored into separate queues depending on their type
Queue scheduling policy	ACKs-first scheduling among TCP ACK queue and the other two queues TCP data and non-TCP queues are scheduled using the round-robin policy
Queuing policy	FIFO (for all queues)
Queue size	TCP ACK queue: dynamically decreases as the number of packets in the TCP data queue increases TCP data and non-TCP queues: the system default value is used
Drop policy	TCP ACK queue: random dropping TCP data and non-TCP queues: tail dropping

Please cite this article in press as: J. Park et al., Preventing TCP performance interference on asymmetric links using ACKs-first variable-size queuing, Comput. Commun. (2010), doi:10.1016/j.comcom.2010.09.010

host. The local host was connected to the gateway via the home network which was realized with a 100 Mbps Ethernet link and the gateway was again connected to the Internet through an ADSL link whose service was offered by a commercial Internet Service Provider (ISP). The raw upload and download bandwidth of the ADSL link were measured to be 2.1 Mbps and 800 Kbps, respectively.

The local host was implemented with a lap top PC running Windows XP and the remote host with a PC running Linux 2.4. The remote host was a public FTP server located about 200 km away from the local host. On the local host, a multithreaded FTP client program ran to continuously generate upload and download traffic to and from the FTP server simultaneously. The network configuration parameters of the two operating systems were set to their default values and remained unchanged during the experiments. Two parameters s_{max} and n_{thr} introduced by AFVQ were set to 5 and 36, respectively. The values were determined by measuring the number of packets in the ACK and data queues while activating the traffic in one direction. Specifically, the former was chosen as the maximum number of packets in the ACK queue. This is to prevent ACK packets from being dropped when the performance interference problem does not occur. The latter was chosen as the average number of packets in the data queue. This is to allocate the maximum allowable bandwidth for data packets by minimizing the ACK queue size when the number of queued data packets exceeds the average.

We measured the file transfer rates of upload and download traffic at the local host with and without AFVQ. Initially, we ran the gateway with the original FIFO queuing for a predetermined period of time and then we activated the AFVQ in the gateway. Note that it is possible to dynamically activate AFVQ since the traffic controller supports the run-time reconfiguration of queuing policies. Fig. 8 shows the results of our experiments. With AFVQ, download and upload speed yielded 2.0 Mbps and 750 Kbps, respectively, which are 95.2% and 93.8% of the maximum download and upload bandwidth. Without AFVQ, the upload speed was seriously degraded to 300 Kbps. This clearly suggests that the AFVQ mechanism and its implementation effectively solve the performance interference problem.

Observe that both upload and download speed with AFVQ are slightly less than the maximum attainable bandwidths of ADSL. They are reduced by 6.3% and 4.8%, respectively.This is due to the extra bandwidth required for transferring ACK packets.

6.2. Simulation experiments

In order to show that AFVQ outperforms other existing approaches in diverse network configurations, we performed extensive simulation analyzes with different network parameter settings. We particularly focused on the effect of the asymmetry ratio of a link and the Internet delay since there exist a wide variety of link technologies with different asymmetry ratios and the Internet delay often has a significant effect on the TCP throughput. We thus chose the following two test variables.

AsymmetryRatio × *InternetDelay*

The asymmetry ratio is defined as a ratio of the downlink bandwidth to the uplink bandwidth.

To effectively quantify the performance of AFVQ and other approaches, we defined two metrics, an aggregated utilization and a utilization variance. The former is the sum of the upload and the download utilization of a link and the latter is the variance between the two. The upload utilization is defined as the average TCP throughput over a raw uplink transfer rate and the download utilization is similarly defined. It measures how much of the raw bandwidth is used for transferring TCP data packets. AFVQ attempts to achieve a large aggregate utilization by preventing the performance interference problem. Clearly, the aggregated utilization of 2 is the ideal value. On the other hand, it is equally important to achieve balanced upload and download utilizations; otherwise, a high utilization in one direction may be achieved by sacrificing the utilization in the other. Thus, a utilization variance close to 0 is preferred.

We implemented our simulation environment using the ns-2 network simulator [40]. It consisted of a local and a remote host, a residential gateway, a home network, an asymmetric link and the Internet, as did our experimental environment. We configured the home network to have 100 Mbps transfer rate and 0.1 ms propagation delay. This was to model the popular Fast Ethernet. The propagation delay and drop probability of the asymmetric link were respectively set to 10 ms and 1% in both directions. These values are higher than those values that can be observed in about 90% of Internet accesses via asymmetric links [41]. We chose such conservative values for our simulation since we wanted to observe the performance behaviors in the worst case scenarios. We also set the transfer rate of the Internet to a value greater than that of the asymmetric link so as to prevent the exact value of the Internet transfer rate from affecting the TCP throughput. Note that the end-to-end throughput is limited by the slowest link, which is the asymmetric link in our simulation.

We configured the protocol stack implementation as well. The maximum TCP window size was set to 200 data packets. The queue size of each node was set to an arbitrarily large value to model the large buffer size inside routers. In the local and remote host, we used the most widely deployed version of TCP, TCP Reno, with the delayed ACK feature.

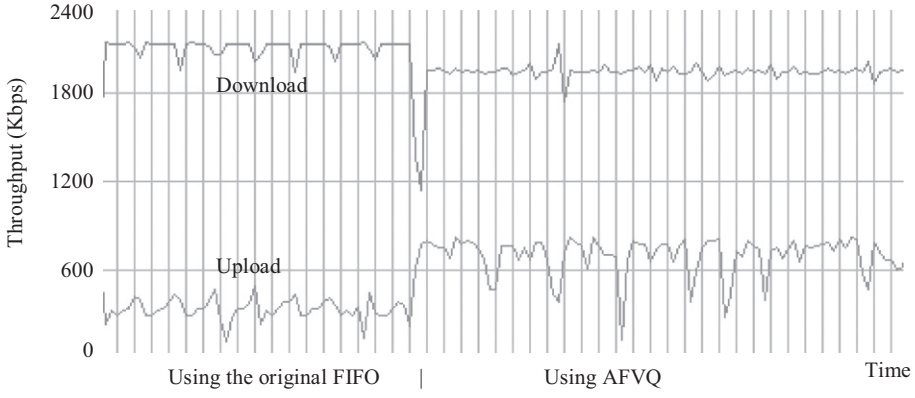

Fig. 8. Changes in upload and download throughputs when AFVQ is used or not.

Please cite this article in press as: J. Park et al., Preventing TCP performance interference on asymmetric links using ACKs-first variable-size queuing, Comput. Commun. (2010), doi:10.1016/j.comcom.2010.09.010

To design test cases for our simulation, we varied the asymmetry ratio of the link 1.0 to 200.0 by changing the uplink transfer rate from 20 Mbps to 100 Kbps while maintaining the downlink transfer rate to 20 Mbps. This covers the wide variety of existing asymmetric link technologies as discussed in [41]. We also varied the Internet delay from 1 ms to 100 ms to emulate diverse distances between end hosts and various Internet traffic congestion situations.

For each test case, we ran a simulation and evaluated AFVQ and three representative gateway-based approaches, ACQ [22,23], ACKs-first [8,18] and ACK Filtering (AF) [8]. We chose them because they could be used for our gateway implementation and their performance behaviors subsumed other minor variations. As the ACQ mechanism required to define two tunable parameters, we used the same values provided in [23]. The ACKs-first and the AF mechanism did not have any tunable parameter. For our AFVQ mechanism, we set the parameter values in the same way that we used in the previous experiment. During the simulation, we ran TCP applications on the local and the remote host that generate continuous data streams in both directions.

Fig. 9 shows the results of the simulation with different asymmetry ratios and the fixed Internet delay of 20 ms. In Fig. 9 (A), we see that AFVQ achieves a higher aggregated utilization than the others for all asymmetry ratios. ACQ and ACKs-first perform closely to AFVQ only when the asymmetry ratio is smaller than 6 or higher than 57. AF yields the lowest aggregated utilization among the four mechanisms. In Fig. 9 (B), we observe that AFVQ achieves the second lowest utilization variance after ACQ. However, when asymmetry ratio is in between 0 and 25, AFVQ shows the lowest utilization variance among the four. For all asymmetry ratios, the utilization variance of AFVQ does not exceed 0.3, which indicates that AFVQ fairly improves both upload and download utilization.

Fig. 10 shows the simulation result when we varied the Internet delay while the uplink and the downlink transmission rates were fixed to 100 Kbps and 3000 Kbps, respectively. In Fig. 10(A), we

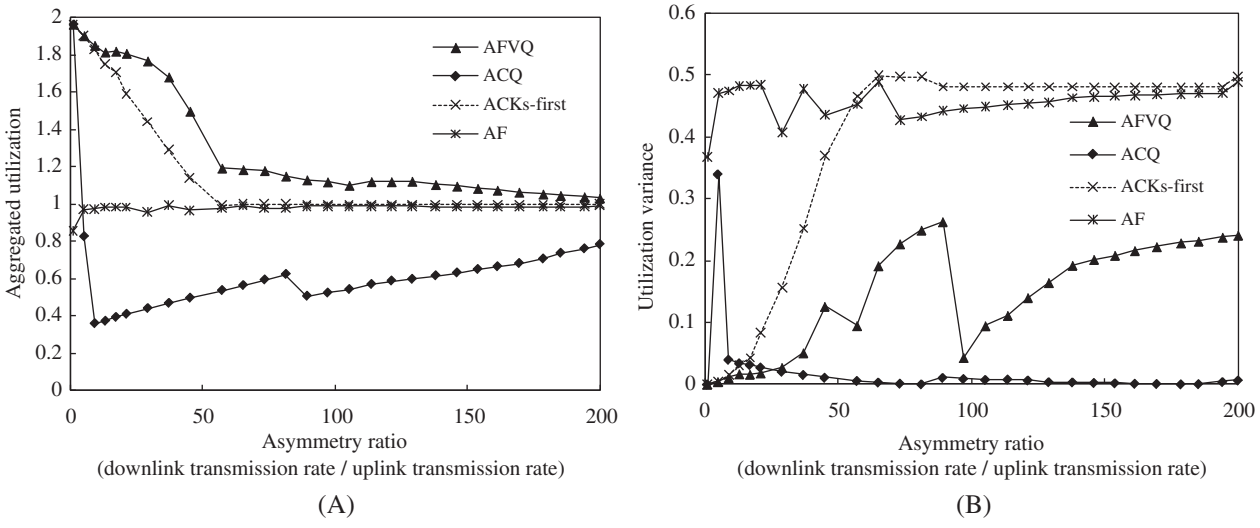

Fig. 9. The influence of the asymmetry ratio to the TCP performance: (A) the aggregated utilization versus asymmetry ratio; and (B) the utilization variance versus asymmetry ratio.

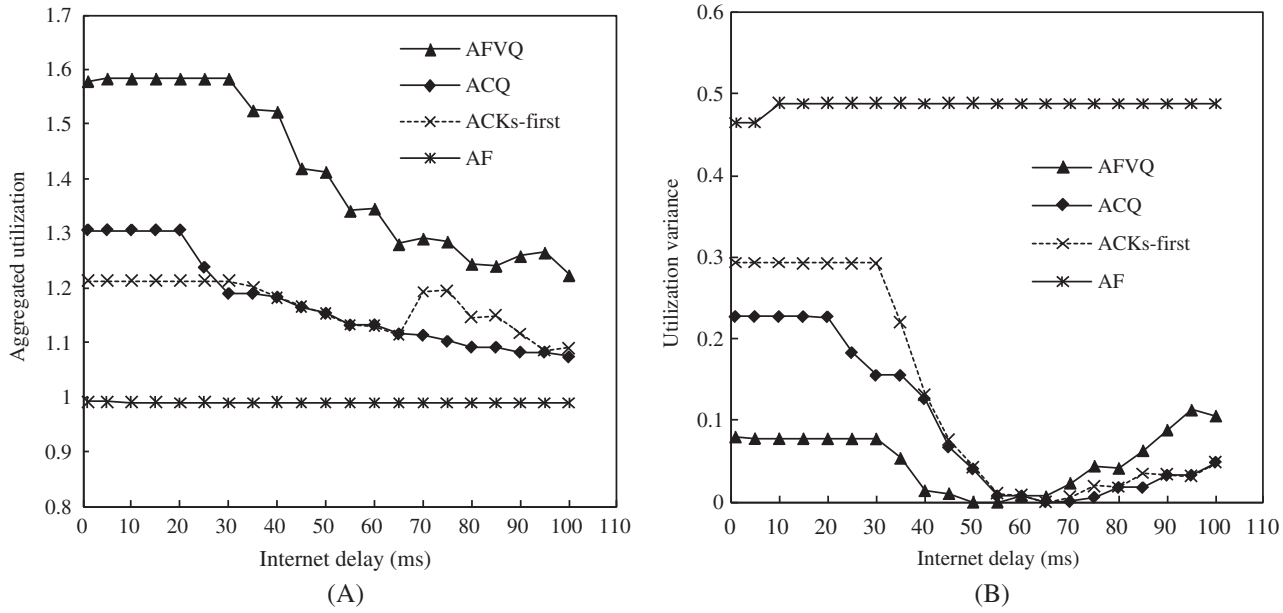

Fig. 10. The influence of the Internet delay to the TCP performance: (A) the aggregated utilization versus Internet delay; and (B) the utilization variance versus Internet delay.

Please cite this article in press as: J. Park et al., Preventing TCP performance interference on asymmetric links using ACKs-first variable-size queuing, Comput. Commun. (2010), doi:10.1016/j.comcom.2010.09.010

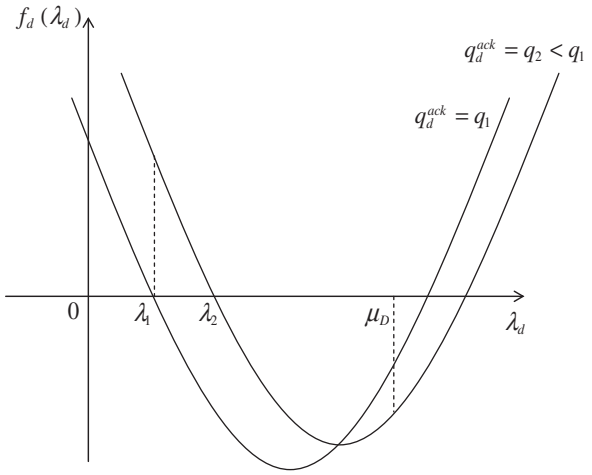

Fig. 11. Plots of the function f_d when q_d^{ack} is q_1 and $q_2 < q_1$.

see that AFVQ achieves the highest aggregate utilization for all Internet delays. Similar to the previous experiments, ACQ and ACKs-first perform closely to AFVQ only when the Internet delay is longer than 100 ms. In Fig. 10(B), we observe that AFVQ yields the variance under 0.03 for all the Internet delays.

7. Conclusions

In this paper, we have addressed the TCP performance interference problem on an asymmetric link in which upload or download throughput abruptly degrades in the presence of bidirectional TCP traffic on the link. In order to solve the problem, we derived an analytic model for the steady-state TCP performance with bidirectional traffic to clearly identify the sources of the problem. The sources we identified are the excessive queuing delay of ACK packets and the excessive number of ACK packets in the queue. We designed the AFVQ mechanism to directly eliminate the two causes. Specifically, we based AFVQ on two policies. First, ACKs-first scheduling was used to shorten the queuing delay of ACK packets. Second, the queue size for ACK packets was dynamically adjusted depending on the number of queued data packets so that the number of ACK packets was reduced when packets were congested.

We have implemented the AFVQ mechanism in our ADSL-based residential gateway using the traffic control module of the Linux kernel. In order to show that AFVQ improves the TCP performance in both directions and is effective for a wide range of networks, we have conducted a series of experiments both in real-world and simulated networks. The real-world experiments have been performed across a home network and the Internet that were interconnected via a commercial ADSL link. Our gateway yielded 95.2% and 93.8% of the maximum download and upload bandwidth, respectively. We have also evaluated the proposed mechanism using the ns-2 simulator over a number of network configurations with different asymmetry ratios and Internet delays and shown that AFVQ achieves better upload and download throughput than other representative gateway-based mechanisms, such as ACQ, ACKs-first scheduling and ACK Filtering.

There are two research directions along which our solution can be extended. First, we are extending the proposed mechanism for latency-asymmetric links such as satellite links and hybrid coaxial cables in which different paths are used for upload and download traffic. Second, we are also looking to enhance AFVQ so that it can adopt itself in dynamically changing networks such as the *ad hoc* wireless networks. Using a close loop control mechanism is currently being considered a candidate solution. The results look promising.

Appendix A

In order to prove Theorem 1, we first show that Eq. (14) has only one root in internal $(0, \mu_D)$.

Lemma 1. Eq. (14) *always has only one root in interval* $(0,\mu_D)$.

Proof. Since function f_d in Eq. (14) is a continuous quadratic function, we prove the lemma by showing $f_d(0)f_d(\mu_D) < 0$. We proceed with two cases.

Case 1: $maxwnd_d \leqslant E[wnd_d^u]$. In this case,

$$f_d(0) = \mu_D maxwnd_d D$$
$$f_d(\mu_D) = -\mu_D\left(D + A \cdot n_{q,u}^{ack}\right).$$

Clearly, $\mu_D > 0$, $maxwnd_d > 0$, $D > 0$, $A > 0$ and $n_{q,u}^{ack} \geqslant 0$ since they are physical quantities. Thus, $f_d(0) > 0$ and $f_d(\mu_D) < 0$, and finally, $f_d(0)f_d(\mu_D) < 0$.

Case 2: $maxwnd_d > E[wnd_d^u]$. Calculating $f_d(0)$ and $f_d(\mu_D)$ gives

$$\begin{aligned} f_d(0) &= \mu_D D \\ f_d(\mu_D) &= \mu_D\{C_1X_1 - C_1DE[wnd_d^u] - C_1D - C_1X_2 - \mu_D C_2 + D\} \\ &= \mu_D\left\{-C_1DE[wnd_d^u] - C_1D - C_1A\cdot n_{q,u}^{ack} - \mu_D C_2 + D\right\} \text{by Eq.(15)} \\ &= \mu_D\left\{D(1 - C_1 - C_1E[wnd_d^u]) - C_1A\cdot n_{q,u}^{ack} - \mu_D C_2\right\} \\ &< \mu_D\{D(1 - C_1 - C_1E[wnd_d^u])\} \end{aligned}$$

Since all variables in the above equations are positive, $f_d(0) > 0$. We now show $(1 - C_1 - C_1E[wnd_d^u]) < 0$. Using Eqs. (2) and (4), we get

$$\begin{aligned} &1 - C_1 - C_1E[wnd_d^u] \\ &= 1 - \left(\sqrt{\frac{2bp_D}{3}}\frac{2+4b}{3b} + \sqrt{\frac{2bp_D}{3}}\sqrt{\frac{8(1-p_D)}{3p_D} + \left(\frac{2+p_D}{3p_D}\right)^2}\right) < 1 \\ &- \sqrt{\frac{2bp_D}{3}}\sqrt{\left(\frac{2+p_D}{3p_D}\right)^2} \text{ since } 0 < p_D < 1 = 1 - \sqrt{\frac{2b(p_D^2+4p_D+4)}{27p_D}}. \end{aligned}$$

As a result, showing $(1 - C_1 - C_1E[wnd_d^u]) < 0$ is equivalent to showing

$$\begin{aligned} &1 - \sqrt{\frac{2b(p_D^2+4p_D+4)}{27p_D}} < 0 \\ &\iff \frac{2b(p_D^2+4p_D+4)}{27p_D} > 1 \\ &\iff p_D^2 + \left(4 - \frac{27}{2b}\right)p_D + 4 > 0. \end{aligned}$$

The last inequality holds since

$$\begin{aligned} p_D^2 + \left(4 - \frac{27}{2b}\right)p_D + 4 &= \left(p_D - \left(\frac{27}{4b} - 2\right)\right)^2 + \frac{27}{b}\left(1 - \frac{27}{16b}\right) \\ &> \frac{27}{b}\left(1 - \frac{27}{16b}\right) > 0 \text{ since } b \geqslant 2. \end{aligned}$$

This concludes the proof. □

Proof of Theorem 1. Function f_d is a multivariable function of λ_d, q_d^{ack} and $n_{q,u}^{ack}$. We show that f_d is a monotonically decreasing function of q_d^{ack} and $n_{q,u}^{ack}$ and that its root monotonically increases as q_d^{ack} and $n_{q,u}^{ack}$ decrease. We consider two cases.

Case 1: $maxwnd_d \leqslant E[wnd_d^u]$. To show that f_d is a monotonically decreasing function of q_d^{ack} and $n_{q,u}^{ack}$, we calculate its partial derivatives with respect to the two variables.

$$\frac{\partial f_d}{\partial q_d^{ack}} = \lambda_d^2 - \mu_D\lambda_d = \lambda_d(\lambda_d - \mu_D)\frac{\partial f_d}{\partial n_{q,u}^{ack}} = -A\lambda_d$$

Please cite this article in press as: J. Park et al., Preventing TCP performance interference on asymmetric links using ACKs-first variable-size queuing, Comput. Commun. (2010), doi:10.1016/j.comcom.2010.09.010

Since $\lambda_d > 0$, $\lambda_d < \mu_D$ and $A > 0$ as shown in Lemma 1, $\partial f_d/\partial q_d^{ack} < 0$ and $\partial f_d/\partial n_{q,u}^{ack} < 0$. As q_d^{ack} and $n_{q,u}^{ack}$ are mutually independent, we have

$$f_d\left(\lambda_d, q_1, n_{q,u}^{ack}\right) < f_d\left(\lambda_d, q_2, n_{q,u}^{ack}\right), \quad \text{if } 0 < q_2 < q_1$$
$$f_d\left(\lambda_d, q_d^{ack}, n_1\right) < f_d\left(\lambda_d, q_d^{ack}, n_2\right) \quad \text{if } 0 < n_2 < n_1.$$

Let λ_1 be the root of Eq. (14) when $q_d^{ack} = q_1$. By substituting λ_d with λ_1 in the first inequality just above, we have

$$f_d\left(\lambda_1, q_2, n_{q,u}^{ack}\right) > f_d\left(\lambda_1, q_1, n_{q,u}^{ack}\right), \quad \text{if } 0 < q_2 < q_1.$$

Since $f_d\left(\lambda_1, q_1, n_{q,u}^{ack}\right) = 0$, we have $f_d\left(\lambda_1, q_2, n_{q,u}^{ack}\right) > 0$ for $0 < q_2 < q_1$. On the other hand, as shown in the proof of Lemma 1, $f_d\left(\mu_d, q_2, n_{q,u}^{ack}\right) < 0$. Thus, we have

$$f_d\left(\lambda_1, q_2, n_{q,u}^{ack}\right) f_d\left(\mu_d, q_2, n_{q,u}^{ack}\right) < 0.$$

This implies that $f_d\left(\lambda_d, q_2, n_{q,u}^{ack}\right) = 0$ has root λ_2 in interval (λ_1, μ_D). Since the interval does not include λ_1, we conclude that $\lambda_2 > \lambda_1$. In other words, the download throughput increases as we decrease the queuing delay of ACK packets for download traffic. Fig. 11 visualizes the function when $q_d^{ack} = q_1$ and $q_d^{ack} = q_2$.

Similarly, via the same process, we can show that $\lambda_4 > \lambda_3$ where λ_3 and λ_4 are respectively the roots of $f_d(\lambda_d, q_d^{ack}, n_1) = 0$ and $f_d(\lambda_d, q_d^{ack}, n_2) = 0$ where $n_2 < n_1$. This implies that the download throughput increases as we decreases the number of queued ACK packets for the opposite side traffic.

Case 2: $maxwnd_d > E[wnd_d^u]$. We show that $\partial f_d/\partial q_d^{ack} < 0$ and $\partial f_d/\partial n_{q,u}^{ack} < 0$. From Eq. (14), we get

$$pc\frac{\partial f_d}{\partial q_d^{ack}} = C_1\lambda_d^2\frac{\partial X_1}{\partial q_d^{ack}} - C_1\lambda_d\frac{\partial X_2}{\partial q_d^{ack}}$$
$$= C_1\lambda_d(\lambda_d - \mu_D) \text{ by Eq.(15)},$$
$$\frac{\partial f_d}{\partial n_{q,u}^{ack}} = -C_1\lambda_d\frac{\partial X_2}{\partial n_{q,u}^{ack}} = -C_1\lambda_d A \text{ by Eq.(15)}.$$

Since $\lambda_d < \mu_D$ and all the variables in the two equations are positive, $\partial f_d/\partial q_d^{ack} < 0$ and $\partial f_d/\partial n_{q,u}^{ack} < 0$. The remainder of the proof is the same as in the former case. □

Acknowledgment

The work reported in this paper was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MEST) (No. 2010-0027809 and No. 2010-0001201).

References

[1] HGI. What is a residential gateway for?, Home Gateway Initiative.
[2] ITU, ITU-T Rec. G.992.1 (07-1999) Asymmetrical digital subscriber line (ADSL) transceivers, 1999.
[3] ITU, ITU-T Rec. J.112 (03/98) Transmission systems for interactive cable television services, 1998.
[4] ITU. ITU-T Rec. G.983.1-4 (01/2005) Broadband optical access systems based on Passive Optical Networks (PON), 2008.
[5] V. Jacobson, M.J. Karels, Congestion Avoidance and Control, 1988.
[6] S. Shenker, L. Zhang, D.D. Clark, Some observations on the dynamics of a congestion control algorithm, ACM Computer Communications Review 20 (1990) 30–39.
[7] L. Zhang, S. Shenker, D.D. Clark, Observations on the dynamics of a congestion control algorithm: the effects of two-way traffic, ACM Computer Communications Review 21 (1991) 133–147.
[8] H. Balakrishnan, V.N. Padmanabhan, R.H. Katz, The Effects of Asymmetry on TCP Performance International Conference on Mobile Computing and Networking, Budapest, Hungary, 1997.
[9] T.R. Henderson, R.H. Katz, TCP Performance over Satellite Channels, University of California, Berkeley, 1999.
[10] L. Kalampoukas, A. Varma, K.K. Ramakrishnan, Two-way TCP traffic over rate controlled channels: effects and analysis, IEEE/ACM Transactions on Networking 6 (1998) 729–743.
[11] T.V. Lakshman, U. Madhow, The performance of TCP/IP for networks with high bandwidth-delay products and random loss, IEEE/ACM Transactions on Networking 5 (1997) 336–350.
[12] K. Phanse, L.A. DaSilva, K. Kidambi, Effects of Competing Traffic on the Performance of TCP/IP over Asymmetric Links 25th Annual IEEE Conference on Local Computer Networks, Tampa, Florida, USA, 2000, pp. 542–543.
[13] Y. Tian, K. Xu, N. Ansari, TCP in wireless environments: problems and solutions, IEEE Communications Magazine 43 (2005) S27–S32.
[14] P. Papadimitriou, V. Tsaoussidis, On TCP performance over asymmetric satellite links with real-time constraints, Computer Communications 30 (2007) 1451–1465.
[15] I.T. Ming-Chit, D. Jinsong, W. Wang, Improving TCP performance over asymmetric networks, ACM Computer Communications Review 30 (2000) 45–54.
[16] S. Kalyanaraman, D. Shekhar, K. Kidambi, TCP/IP Performance Optimization over ADSL, 1999.
[17] L. Yu, Y. Minhua, Z. Huimin, The Improvement of TCP Performance in Bandwidth Asymmetric Network 14th IEEE International Symposium on Personal, Indoor and Mobile Radio Communication, IEEE, Beijing, China, 2003, pp. 482–486.
[18] J. Park, S. Hong, Preventing network performance interference with ACK-separation queuing mechanism in a home network gateway using an asymmetric link 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, Daegu, Korea, 2007, pp. 550–555.
[19] T. Taleb, N. Kato, Y. Nemoto, REFWA: an efficient and fair congestion control scheme for LEO satellite networks, IEEE/ACM Transactions on Networking 14 (2006) 1031–1044.
[20] K. Xu, Y. Tian, N. Ansari, TCP-Jersey for wireless IP communications, IEEE Journal on Selected Areas in Communications 22 (2004) 747–756.
[21] K. Xu, Y. Tian, N. Ansari, Improving TCP performance in integrated wireless communications networks, Computer Networks 47 (2005) 219–237.
[22] F. Louati, C. Barakat, W. Dabbous, Handling Two-Way TCP Traffic in Bandwidth Asymmetric Networks, INRIA, 2003.
[23] F. Louati, C. Barakat, W. Dabbous, Handling Two-Way TCP Traffic in Asymmetric Networks 7th IEEE International Conference on High Speed Networks and Multimedia Communications, Toulous, France, 2004.
[24] W. Al-Khatib, K. Gunavathi, A new approach to improve TCP performance over asymmetric networks, Electronics and Electrical Engineering 7 (2006).
[25] Q. Xia, X. Jin, M. Hamdi, Dual queue management for improving TCP performance in multi-rate infrastructure WLANs IEEE International Conference on Communications, 2008, pp. 2531–2535.
[26] H. Balakrishnan, V.N. Padmanabhan, G. Fairhurst, M. Sooriyabandara, RFC 3449-TCP Performance Implications of Network Path Asymmetry, 2002.
[27] Arcturus. uClinux: Embeded Linux/Microcontroller Project.
[28] ITU. ITU-T Rec. G.992.2 (07/99) Splitterless Asymmetrical Digital Subscriber Line (ADSL) Transceivers, 1999.
[29] K.R. Sollins, RFC 783-TFTP Protocol (revision 2), 1981.
[30] B. Hubert, Linux Advanced Routing & Traffic Control HOWTO.
[31] L.L. Peterson, B.S. Davie, Computer Networks: A Systems Approach, Morgan Kaufmann, 2000.
[32] V. Jacobson, Modified TCP Congestion Avoidance Algorithm end2end-interest mailing list, 1990.
[33] W. Stevens, RFC 2001-TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms, 1997.
[34] L.S. Brakmo, S.W. O'Malley, L.L. Peterson, TCP vegas: new techniques for congestion detection and avoidance, ACM Computer Communications Review 24 (1994).
[35] K. Xu, N. Ansari, Stability and fairness of rate estimation based AIAD congestion control in TCP, IEEE Communications Letters 9 (2005) 378–380.
[36] M. Mathis, J. Semke, J. Mahdavi, T. Ott, The macroscopic behavior of the TCP congestion avoidance algorithm, ACM SIGCOMM Computer Communication Review 27 (1997) 67–82.
[37] S.H. Low, A duality model of TCP and queue management algorithms, IEEE/ACM Transactions on Networking 11 (2003) 525–536.
[38] J. Padhye, V. Firoiu, D.F. Towsley, J.F. Kurose, Modeling TCP reno performance: a simple model and its empirical validation, IEEE/ACM Transactions on Networking 8 (2000) 133–145.
[39] G. Hasegawa, M. Murata. Analysis of dynamic behaviors of many TCP connections sharing tail-drop/RED routers IEEE Global Telecommunications Conference, 2001.
[40] S. McCanne, S. Floyd. ns–Network Simulator.
[41] M. Dischinger, A. Haeberlen, K.P. Gummadi, S. Saroiu. Characterizing Residential Broadband Networks Internet Measurement Conference 2007, San Diego, CA, USA, 2007.

Please cite this article in press as: J. Park et al., Preventing TCP performance interference on asymmetric links using ACKs-first variable-size queuing, Comput. Commun. (2010), doi:10.1016/j.comcom.2010.09.010