Preventing network performance interference with ACK-separation queuing mechanism in a home network gateway using an asymmetric link

Jiyong Park and Seongsoo Hong
Real-Time Operating Systems Laboratory, Seoul National University, Korea
{parkjy, sshong}@redwood.snu.ac.kr

Abstract

In network-enabled consumer electronics development, much of the time and effort is spent analyzing and solving network performance problems. In this paper, we define an instance of such problems discovered while developing a commercial home network gateway. We then analyze its cause and propose a solution mechanism. Our home network gateway uses an asymmetric link (ADSL) and suffers from an undesirable phenomenon where downlink traffic interferes with upload speed. We call this phenomenon the network performance interference problem. While this problem can easily be confused with receive livelock caused by packet contention at the input queue, we find that this is not the case. By performing extensive experiments and analysis, we reveal that our problem is caused by packet contention at the output queue and certain intrinsic characteristics of TCP. We devise an ACK-separation queuing mechanism for this problem and implement it in the home network gateway. Our experiments show that it effectively solves the problem.

1. Introduction

With the rapid convergence of disparate technologies in recent days, consumer electronics are now characterized with a wide variety of versatile features and diverse network connectivity via a broad range of communication protocols. As a consequence, software installed in such devices consists of number of interrelated modules. This makes it challenging to develop embedded software for consumer electronic products containing a network subsystem. In deriving optimal network performance, it is very difficult to predict the effects of selected optimizations since there exist complex interrelationships between heterogeneous network devices and moreover, network performance is not always proportional to the amount of available hardware resources. 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 discuss a network performance optimization problem in a home network gateway that is typical of complex consumer electronic devices. For simplicity, we hereon refer to a home network gateway as a home gateway. Basically, a home gateway is a device that provides connections to the Internet for other consumer devices that constitute a home network. In addition to this, it provides network services such as a firewall, Network Address Translation (NAT), Dynamic Host Configuration Protocol (DHCP), and Simple Network Management Protocol (SNMP). It also manages home devices by storing information about the devices and providing device drivers for them. It often includes a web server so that users can remotely control the devices via a web browser. As such, a home gateway has become a highly complex device that provides composite features.

Usually, a home gateway is connected to the Internet via an Asymmetric Digital Subscriber Line (ADSL) which is an asymmetric link that has different transmission speeds in each direction [9]. This link achieves a higher data transmission speed than the ordinary telephone modem, yet is still very cheap since it utilizes already installed telephone lines. By observing that most Internet traffic patterns have an emphasis on download traffic, it allocates more bandwidth for downstream than for upstream, thus achieving a fast download speed within the limited total bandwidth.

However, as new types of devices and application domains are emerging, upload traffic in home networks is also rapidly increasing. For example, consumer devices such as internet phones, interactive TVs, and web cameras are frequently uploading video and audio data via the Internet. Home PCs also incur significant upload traffic as new communication applications such as Peer-to-Peer (P2P) are becoming popular. As a result, upload speed is an increasingly important factor in determining network performance.

We have developed a commercial home gateway that has the above characteristics. In doing so, we used common off-the-shelf (COTS) software such as the Linux operating system, its protocol stacks, and open source network server software to reduce time-to-market and to save on the cost of the product. However, when we developed a prototype of the home gateway, we found that its bandwidth usage was much less than the available bandwidth of ADSL. We call this undesirable phenomenon the *network performance interference problem on an asymmetric link* where the traffic speed in one direction is interfered with by traffic in the other

direction. For simplicity, we henceforth refer to this problem as the *performance interference problem*. Specifically, in our home gateway using an ADSL, upload speed is reduced to about one third of its maximum speed of 800 Kbps when it is interfered with by packets on the downlink which has a maximum speed of 2.1 Mbps. This leads to unacceptable performance degradation in the entire home network. Such network performance problems are inherently difficult to analyze. In this paper, we determine its cause, propose a solution, and validate the solution by implementing it and performing extensive experiments.

The performance interference problem can be thought of as a special case of a well known problem called receive livelock [2] since their effects are very similar. Receive livelock is a problem where the throughput of the network subsystem is reduced as the rate of arriving packets increases [4]. This happens very often in operating systems that have a single FIFO input queue, buffer all received packets in this queue, and process the packets through a low priority software interrupt handler. When packets are arriving at the input queue faster than the handler processes the packets, packet contention occurs. Some packets are dropped since there is no room in the queue, resulting in a speed reduction. Most operating systems, including Linux that is used in our home gateway, have a packet processing mechanism similar to this. This problem is typically solved by providing separate queues for the uplink and downlink and by using a queue scheduling mechanism that selects a queue fairly to prevent packets on the uplink from being excessively dropped.

However, contrary to expectation, this mechanism does not solve our performance interference problem. This suggests that our problem is clearly different from receive livelock. We show that packet contention at the output queue rather than the input queue is the cause of our problem and the innate characteristics of TCP/IP are also closely related with it. Based on this, we devise an ACK-separation queuing mechanism. It is an output queue management scheme that processes TCP ACK packets separately at the highest priority among received packets. We then apply this mechanism to the home gateway using the queuing discipline of the Linux operating system and evaluate its performance.

There have been many research results on the performance degradation of TCP in asymmetric links. Lakshman and Madhow deal with download speed in an asymmetric link with a low-speed uplink [6]. Kalampoukas and Varma extend this by considering the case when there is traffic in both directions [3]. They propose per-connection queues as a solution. Balakrishnan et al. solve the same problem by using ACK-filtering and ACK-regeneration [5]. These solutions have limited applicability since they require modification at the end systems of the network such as PCs and servers. Thus, they cannot be easily adapted to our situation since we can modify only our home gateway and have no control over the end systems. Furthermore, the proposed solutions enhance either download speed or upload speed. Thus they are not a complete solution for the performance interference problem. Finally, all of these results are derived from simulations and actual validation has not been performed. This paper differs from the previous research results since we implement and evaluate our solution in a real system.

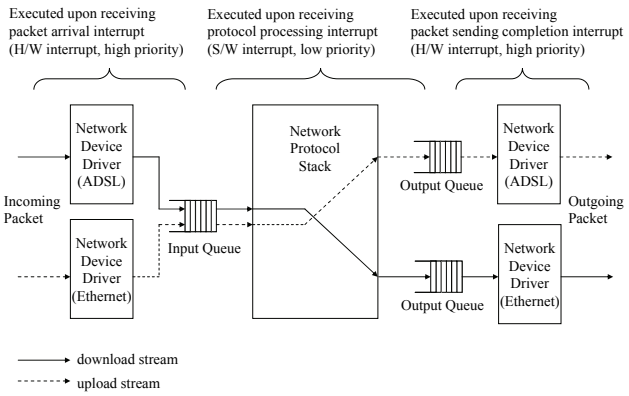

Figure 1. Structure and packet processing mechanisms of the conventional network subsystem in a home gateway.

The rest of the paper is organized as follows. In the next section, we explain conventional packet processing mechanisms and describe the network performance interference problem in detail. Section 3 explains the cause of the problem by examining and analyzing the output queue. Section 4 presents an ACK-separation queuing mechanism as a solution for the problem and gives its implementation results. Finally, Section 5 serves as our conclusion.

2. Problem description

The network performance interference problem on an asymmetric link occurs when legacy protocol processing mechanisms are used in a home gateway. In order to give a detailed account of the problem, we first explain conventional packet processing in the COTS software. We also present the network and software configuration of the experimental setup that we used to reproduce the behavior of the problem so as to analyze it. We then clearly describe the problem.

2.1. Conventional packet processing mechanisms

Our home gateway has one ADSL and one Ethernet interface. It is connected to the Internet through the ADSL link and to the PC as well as various consumer devices through an Ethernet connection. The operating system of the gateway is uClinux [8]. Inside it there are three different protocol stacks, RFC 2684, RFC 2364, and RFC 1577. One of these is selected and used according to the gateway configuration. Basically, they do the same task of transferring a packet received from one network interface to the opposite one. The differences among them lie in the details of how they analyze and handle packets.

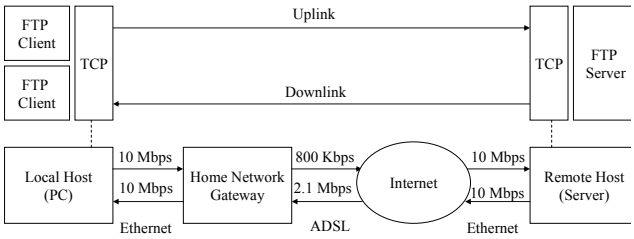

Figure 2. Network configuration for demonstrating performance interference problem. The lower part shows hardware configuration and the upper part shows software configuration.

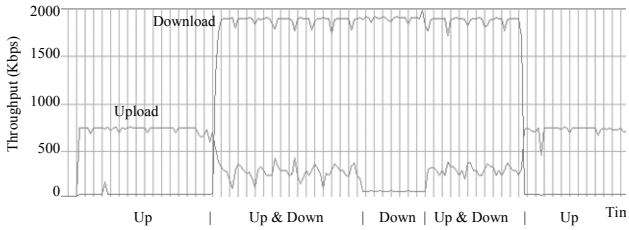

Figure 3. File transfer throughputs for the upload and download directions according to the combinations of upload and download file transfer activities. "Up" and "Down" each represents upload and download file transfer is activated, respectively.

The detailed structure and packet processing mechanism of the conventional network subsystem is depicted in Figure 1. It consists of three elements: 1) the device driver, 2) the network protocol stack, and 3) the input/output queue. There are two network device drivers, each for controlling the ADSL and the Ethernet hardware, respectively. These device drivers basically perform two tasks: copying arrived packets into memory and copying packets stored in the memory into the network interface. The network protocol stack does the actual packet processing. Specifically, it analyzes the packet header, looks up the routing table, determines the network interface where the packet is sent, and sometimes modifies the packet header when NAT is enabled. After being processed, the packet is stored in a specific memory location for transmission. The input/output queues are in-memory data structures used for storing packets between the device drivers and the network protocol stack. The packets are queued and dequeued by 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 home gateway are processed one by one.

A packet is processed by three different handlers such as the packet arrival hardware interrupt handler, the protocol processing software interrupt handler, and the packet sending completion hardware interrupt handler. The first and the third run at maximum priority since they are hardware interrupt handlers. On the other hand, the second runs at a lower priority than the other two because it is a software interrupt handler. It is executed only when the other two handlers are not running and can be immediately preempted by them. Since the execution periods and durations of these three handlers may vary depending on the packet arrival rate and the length of the packet, input and output queues are used as buffers to absorb the temporary difference in speed among them. The sizes of all these queues are set to 100 as in Linux. Since they have a limited capacity, packets may be dropped when they are full.

2.2. Network configuration of experimental setup

Figure 2 shows the network and software configuration of the experimental setup that we used to demonstrate the performance interference problem and to evaluate our solution mechanism. It consists of a local host, the home gateway, the Internet, and a remote host. The local host and the home gateway are connected via an Ethernet link and the home gateway is again connected to the Internet through an ADSL link.

While the Ethernet is a symmetric link that offers 10 Mbps speeds in both directions, the ADSL is an asymmetric link that has different upload and download speeds. The upload speed from the home gateway to the Internet is about 800 Kbps, and the download speed in the opposite direction is about 2.1 Mbps, which is about 2.6 times faster than the upload speed. This corresponds to the ADSL lite service that many Internet Service Providers (ISPs) offer.

To reproduce the performance interference problem, we use FTP programs to generate continuous upload and download traffic, simultaneously. To do so, we execute a multi-threaded FTP server program in the remote host and two FTP client programs in the local host. They exchange data using the TCP protocol. One FTP client sends a file to the server, and the other gets a file from the server. The file used in each transfer is an arbitrary file that is bigger than 100 Mbytes. Each client constantly re-transfers the file. This configuration is made to reflect the characteristics of network applications such as P2P and internet telephony.

2.3. Network performance interference problem

In a network configured as above, the performance interference problem is defined as a phenomenon where upload speed is reduced abruptly when there is both upload and download traffic on an asymmetric link. We measured the file transfer throughput for the 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 Figure 3. We can observe 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. We should definitely avoid the performance interference problem

to smoothly support new emerging application domains in home networks.

3. Analyzing packet contentions at output queue

In this section, we analyze and determine the source of the performance interference problem and validate our analysis through experiments. To do so, we first analyze the operational mechanism of TCP to pin point its relationships with our problem. We go on to analyze packet contention at the output queue. Based on this analysis, we present our solution mechanism in the next section.

This indicates that our problem is at least partially related to the TCP protocol processing. To confirm this, we further analyze the operational mechanism of TCP in the next subsections.

3.1. TCP operations

In TCP [1], a sender basically waits for an acknowledgement (ACK) packet after transmitting a data packet. Since this wait-and-send operation may well lead to low throughput, TCP allows a sender to transmit at most *wnd* data packets without waiting for corresponding ACK packets. This window size *wnd* can vary dynamically and is determined as the minimum value between two variables *maxwnd* and *cwnd*. Since *maxwnd* is the maximum value that *wnd* can be, it becomes the size of the buffer that holds data packets waiting to be transmitted. In most operating systems, its typical range is 20 to 100 and it is configured to be 88 in our home gateway. The *cwnd* is a variable that is incremented each time a new ACK packet arrives. This allows TCP to increase the packet transmission rate as it successfully gets ACK packets. If a data packet is lost, then *cwnd* is reset to one, thus reducing the transmission rate dramatically.

To summarize, the transfer speed of TCP decreases when 1) a data packet is lost or 2) an ACK packet experiences long delay even if it does not cause timer expiration. These cases occur when there is packet contention at the input queue and/or the output queue. In what follows, we determine the location of such contention in our home gateway.

3.2. Packet contention at output queue

We analyze packet contention at the output queue. There are two output queues in the home gateway: one is an uplink output queue and the other a downlink output queue. They are directed towards the ADSL link and the Ethernet link, respectively. We first show that there is contention at the uplink output queue and that this causes the loss of both ACK and data packets. We analyze the effect of such packet loss on the upload and download traffic speeds. Then we show that contention does not occur at the downlink output queue, thus it has nothing to do with the performance interference problem.

We begin with analyzing the uplink output queue. This

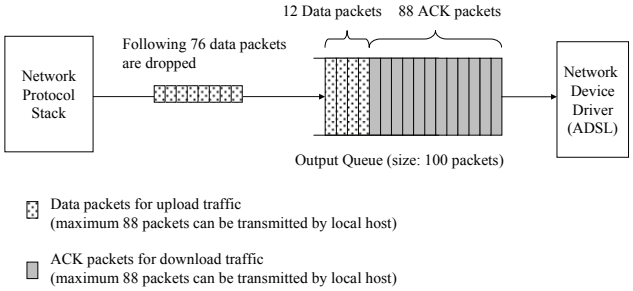

Figure 4. Data packets for the upload traffic are dropped due to the packet contention at output queue.

queue holds data packets for upload traffic and ACK packets for download traffic. It is very likely that packet contention occurs at this queue since there is a huge discrepancy between the 10 Mbps arrival rate and 800 Kbps departure rate. This contention leads to packet dropping at the queue. We analyze the effect of such packet dropping on the download and upload traffic speeds.

Dropping ACK packets does not affect the download speed. This is because ACK packets are cumulative. The latest ACK packet is enough to know 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 remote host does not reduce transmission speed.

However, dropping a single data packet reduces the upload speed. If data packet loss is detected, the local host sets *cwnd* to one and starts to transmit data packets at a slow rate according to the TCP congestion control mechanism. This seriously slows down the data packet arrival rate. On the other hand, the ACK packet arrival rate is maintained since the faster download speed is not affected as stated above. Note that up to 88 ACK packets can be in transit since *maxwnd* is set to 88. Consequently, it is very likely that all of these 88 ACK packets take up the front slots of the uplink output queue. Since the size of the queue is configured as 100, only 12 data packets can be stored in the free slots of the queue. Figure 4 pictorially depicts this situation. If the local host increases its transfer speed by increasing *cwnd*, data packets will soon get dropped since there is not enough room for them at the uplink output queue. As a consequence, the upload data traffic suffers from serious performance degradation. This is the reason for the performance interference problem.

We examine the downlink output queue. Since this queue accepts packets from the relatively slow 2.1 Mbps ADSL link and sends them to the fast 10 Mbps Ethernet link, the possibility of pack contention at this queue is very small.

4. ACK-separation queuing mechanism and implementation results

We have shown that dropping data packets at the uplink

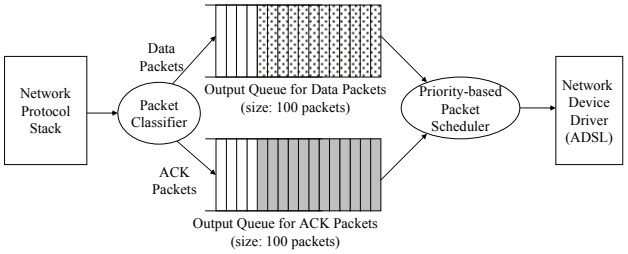

Figure 5. ACK-separation queuing mechanism.

output queue when coupled with the TCP congestion control mechanism becomes the cause of the performance interference problem. Based on this finding, we devise an ACK-separation queuing mechanism for the uplink output queue. Its objective is to avoid dropping data packets as much as possible while providing low latency for ACK packets. In this section, we explain in detail its mechanisms, implementation, and experimental results.

4.1. ACK-separation queuing mechanism

The ACK-separation queuing mechanism has two improvements over the conventional uplink output queue. They are 1) separate queues for ACK and data packets and 2) a priority-based queue scheduler. Figure 5 shows its overall structure.

The separate ACK queue prevents ACK packets from taking up slots in the original uplink output queue and causing data packets to be dropped. In order to isolate ACK packets from an incoming packet stream provided by the protocol stack, a packet classifier inspects packet headers and stores the ACK packets in the newly added queue. Remaining data packets are stored in the old queue as they were. With this queue configuration, upload performance can reach its maximum speed of 800 Kbps since data packets are not dropped.

The new queue scheduler gives ACK packets priority over data packets. We select this policy to reduce the queuing delay of ACK packets and to eventually maintain the maximum download speed of 2.1 Mbps. If ACK packets experience a long delay, download speed gets slowed down as explained before. However, there is a possibility that such priority-driven queuing causes starvation to data packets. Fortunately, in our network configuration, this does not occur because bandwidth demand for ACK packets is much smaller than that for data packets. This can be shown easily as follows. Since one ACK packet is generated for each data packet, the number of ACK packets transferred to uplink per second is the same as that of data packets transferred through the downlink. It is 179.2 packets/s (= 2.1 Mbps / 1500 bytes/packet). Since each ACK packet is only 66 bytes long, they consume 92 Kbps (66 bytes × 179.2 packets/s) in the uplink. This equals to 11.5 % of the entire uplink bandwidth of 800 Kbps. Therefore, there is still enough bandwidth left in the uplink for transferring data packets.

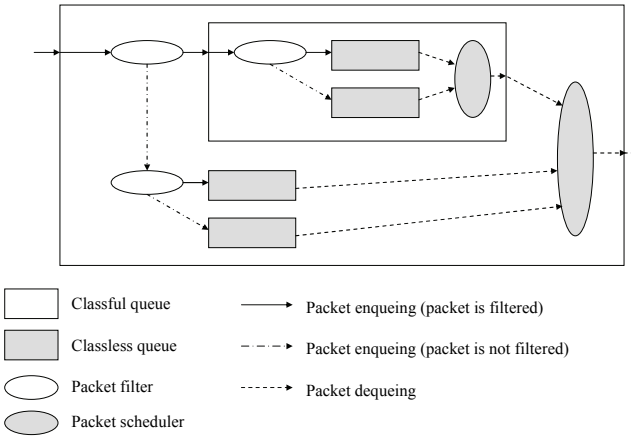

Figure 6. An example of a queue configured using queuing discipline.

4.2. Implementation using queuing discipline

The proposed queuing mechanism is implemented and applied to the network subsystem of our home gateway using the queuing discipline provided by the Linux kernel [7]. It allows users to modify the configurations of the network input and output queues via system calls at run-time. They can replace the default FIFO queue with other queues with different policies. In addition, several queues can be hierarchically structured to form a more complex one.

In the queuing discipline, there are two types of queues: classful and classless queues. A classless queue cannot contain any other queue. A FIFO queue, priority queue, and random early drop (RED) queue are examples. A classful queue can contain other queues. Examples include a class-based queue (CBQ) and a hierarchical token bucket (HTB) queue. A classful queue has packet filter objects that determine where to put a packet among the several internal queues. Users can program the packet filter objects to control packet selection. Also, in a classful queue, there is a packet scheduler object that determines one internal queue from which a packet is taken. It is configured by assigning priorities or selection ratios to the internal queues. Figure 6 is an example of a queue configured using the queuing discipline feature. If a packet does not match the condition of a packet filter, the packet tries to match that of the next packet filter. When the packet reaches the last packet filter and has not matched any conditions, it is transferred to the default queue. Some queues limit the rate at which packets can be inserted. The HTB queue is an example of such a queue. It internally uses a token bucket queue to limit the rate of inserted packets to a constant speed.

Our uplink output queue is modified using the queuing discipline as shown in Figure 7. At the top level, there is an HTB queue that has two FIFO queues whose sizes are configured as 100 packets. One of these queues is for ACK packets, and the other for data packets. The latter is the default queue. A packet filter object in the HTB queue is

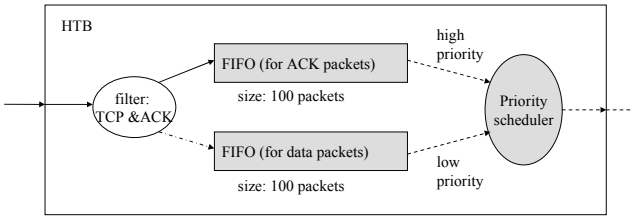

Figure 7. Structure of the enhanced output queue configured using queuing discipline. Legends are the same as Figure 6.

configured to match an ACK packet. We set high priority to the ACK queue and low priority to the default queue.

4.3. Experimental results

We have measured the performance of the enhanced home gateway and compared it with that of the original one which has the conventional uplink output queue. Figure 8 shows the result of our measurements. The enhanced home gateway yields 2.0 Mbps download speed and 750 Kbps upload speed while the original one suffers from performance interference resulting in only 300 Kbps upload speed. This clearly suggests that the proposed queuing mechanism and its implementation effectively solve the performance interference problem.

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

5. Conclusions

In this paper, we have identified the network performance interference problem on an asymmetric link where upload speed is interfered with by traffic on the downlink. While this problem can easily be confused with receive livelock caused by packet contention at the input queue, we have found that our problem is different from it. By performing extensive experiments and analysis, we have revealed that our problem is caused by data packet dropping at the uplink output queue and the TCP congestion control mechanism. Based on this finding, we have devised an ACK-separation queuing mechanism for the uplink output queue that consists of dual queues and a queue scheduler. This mechanism can effectively reduce the amount of data packet dropping at the uplink output queue while providing low latency for ACK packets. We have implemented it using the queuing discipline provided by the Linux kernel. Our experimental results show that it effectively solves the problem.

There are two research directions along which our solution can be extended. First, we are extending the proposed mechanism for other network links such as a satellite link that have a higher degree of asymmetricity and longer transmission delays than ADSL. We believe that such links require different queuing mechanisms than the one proposed here since ACK packets for download traffic can consume a much larger portion of the uplink bandwidth than ADSL. Thus, ACK traffic leaves only a small portion of the uplink bandwidth to data packets. Second, we are looking to provide a general model and solution for the network performance interference problem by parameterizing diverse factors such as bandwidth, delay, and packet size. The results look promising.

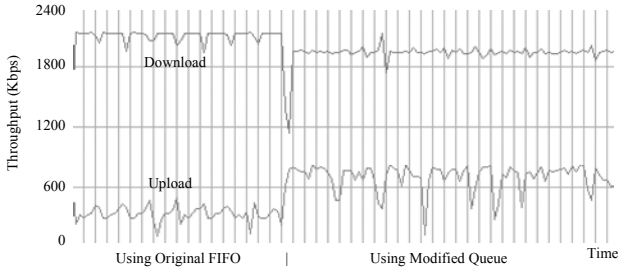

Figure 8. Changes in upload/download speed when the ACK-separation queuing mechanism is used. It achieves 2 Mbps and 750 Kbps for download and upload traffic, respectively. This means that the performance interference problem has disappeared.

6. References

[1] Jacobson, V., "Congestion avoidance and control," *ACM SIGCOMM Computer Communications Review*, vol. 18, no. 4, pp. 314-329, Aug. 1998.

[2] Jeffrey C. Mogul, "Eliminating Receive Livelock in an Interrupt-Driven Kernel," *ACM Transaction on Computer Systems (TOCS)*, vol. 15, no. 3, pp. 217-252, Aug. 1995.

[3] L. Kalampoukas, A. Varma, "Improving TCP Throughput over Two-Way Asymmetric Links: Analysis and Solutions," *Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and Modeling of Computer Systems*, pp. 78-89, Jun. 1998.

[4] Ramakrishnan, K.K., "Scheduling issues for interfacing to high speed networks," *Proceedings of IEEE Global Telecommunications Conference*, pp. 622-626. 1992.

[5] H. Balakrishnan et al., "The Effects of Asymmetry on TCP Performance," *Proceedings of the 3rd annual ACM/IEEE international conference on Mobile computing and networking*, pp. 77-89, Sep. 1997.

[6] T. V. Lakshman, U. Madhow, "Window-based error recovery and flow control with a slow acknowledgement channel: a study of TCP/IP performance," *Proceedings of INFOCOM.*, pp. 1199, Apr. 1997.

[7] Bert Hubert, *Linux Advanced Routing & Traffic Control HOWTO*, http://lartc.org/howto.

[8] uClinux: Embedded Linux/Microcontroller Project, http://www.uclinux.org.

[9] DSL Forum, http://www.dslforum.org.