Sender-driven bandwidth differentiation for transmitting multimedia flows over TCP

K. H. Lau and Jack Y. B. Lee

Department of Information Engineering
The Chinese University of Hong Kong
Hong Kong

ABSTRACT

Over the years the Internet has shown extraordinary scalability and robustness in spite of the explosive growth in geographical reach, user population size, as well as network traffic volume. This scalability and robustness is, in no small part, supported by the Internet's transport protocols, the Transmission Control Protocol (TCP) in particular. Nevertheless, with the rapid growth of multimedia-rich contents in the Internet, such as audio and video, the many strengths of TCP in data delivery are slowly imposing bottlenecks in multimedia data delivery where different media data flows often have different needs for bandwidth. As TCP's congestion control algorithm enforces fair bandwidth sharing among traffic flows sharing the same network bottleneck, different media data flows will receive the same bandwidth irrespective of the actual needs of the multimedia data being delivered. This work addresses this limitation by proposing a new algorithm to achieve non-uniform bandwidth allocation among TCP flows originating from the same sender passing through the same network bottleneck to multiple receivers. The proposed algorithm, called Virtual Packet Substitution (VPS), has four desirable features: (a) it allows the allocation of bottleneck bandwidth between a group of TCP flows; (b) the resultant traffic flows as a whole, maintain the same fair bandwidth sharing property with other competing TCP flows; (c) it can be implemented entirely in the sender's TCP protocol stack; and (d) it is compatible with and does not require modification to existing TCP protocol stack at the clients. Simulation results show that the proposed VPS algorithm can achieve accurate bandwidth allocation while still maintaining fair bandwidth sharing with competing TCP flows.

Keywords: TCP-friendliness, proportional fairness, video streaming, quality of service, packet substitution

1. INTRODUCTION

Over the years the Internet has shown extraordinary scalability and robustness in spite of the explosive growth in geographical reach, user population size, as well as network traffic volume. This scalability and robustness is, in no small part, supported by the Internet's transport protocols, the Transmission Control Protocol (TCP)[4] in particular. The flow and congestion control algorithms in TCP ensure that network bandwidth is shared among competing traffic flows in a fair manner[5], and network congestions are automatically alleviated by throttling the sending rate at the source.

Nevertheless, with the rapid growth of multimedia-rich contents in the Internet, such as audio and video, the many strengths of TCP in data delivery are slowly imposing bottlenecks in multimedia data delivery[1-3]. Specifically, TCP's congestion control algorithm enforces fair bandwidth sharing among traffic flows sharing the same network bottleneck. Thus two multimedia flows going through the same network bottleneck will receive the same bandwidth irrespective of the actual *needs* of the multimedia data being delivered.

For example, suppose a multimedia server is sending two streams of video data S_1 and S_2, of encoded video bit-rates 0.3 Mbps and 0.7Mbps respectively, through the same network bottleneck[17, 18] to two different clients. Now if the network bottleneck has 1Mbps available bandwidth, then in principle there is sufficient bandwidth to transport both video streams. However, if the server simply send both video streams using TCP and rely on TCP's flow and congestion control algorithms to control the sending rates, then the fair bandwidth sharing property of TCP will ensure that each of the video streams will get half the available bandwidth, i.e., 0.5Mbps. Obviously, this is too much for S_1 at 0.3Mbps and too little for S_2 at 0.7Mbps.

Apparently, if the server knows the video streams' bit-rates and can control the transmission rates at the application layer, then it seems the problem can be solved by sending the video at their playback rates instead of the rates allowed by TCP. This approach, however, suffers from two limitations. First, while it is possible for the server application to send data at a rate lower than the rate allowed by TCP, the opposite is simply impossible as the application will soon be blocked from sending more data by the network programming API (e.g., sockets[6]) once the sender's transport buffer is full. Second, even if the server frees up bandwidth from a TCP flow by sending at a lower rate, the saved bandwidth may not be fully transferred to another *specific* TCP flow.

To see why, suppose there are two other competing TCP flows S_3 and S_4 sharing the same bottleneck as the two video streams S_1 and S_2, then any bandwidth, say C bps, freed up by the server (through sending S_1 at a lower rate) will be up for grab by the remaining three competing flows (S_2, S_3, and S_4). Given TCP's fair bandwidth sharing property this means that each of the competing flows (S_2, S_3, and S_4) will receive one-third of the freed bandwidth, i.e., $C/3$ bps. It is easy to see that this *dilution effect* increases with more competing TCP flows sharing the same network bottleneck and in the Internet it is not uncommon to have tens or hundreds of flows going through a network link.

We tackle this problem in this work by proposing a new algorithm to achieve bandwidth allocation among TCP flows originating from the same sender passing through the same network bottleneck to multiple receivers. The proposed algorithm, called Virtual Packet Substitution (VPS), has four desirable features: (a) it allows the allocation of bottleneck bandwidth between a group of TCP flows; (b) the resultant traffic flows as a whole, maintain the same fair bandwidth sharing property with other competing TCP flows; (c) it can be implemented entirely in the sender's TCP protocol stack; and (d) it is compatible with and does not require modification to existing TCP protocol stack at the clients.

2. RELATED WORK

The problem of bandwidth differentiation has been investigated by Mehra *et al.*[7] and Crowcroft *et al.*[8]. Mehra *et al.*[7] proposed a receiver-driven bandwidth allocation algorithm that can allocate bottleneck bandwidth among multiple TCP flows. The principle is to adjust the receiver's TCP advertised windows and the delays in sending acknowledgements such that prioritized and weighted bandwidth sharing can be achieved. However, their algorithm can only allocate bandwidth among flows destined to the *same receiver*, whereas in our study the focus is for flows originating from the *same sender* to multiple receivers.

In another study, Crowcroft and Oechslin[8] proposed the MulTCP congestion control algorithm to achieve differentiated end-to-end service. The principle is to change the rate at which a TCP flow increases and decreases its congestion window to make it behaves like N concurrent TCP flows. Bandwidth allocation can then be achieved by setting the multiplier N for each flow. The protocol is simple and only limited coordination among multiple TCP flows is required. However, as the objective of MulTCP is not to minimize the impact to other competing ordinary TCP traffic, it will cause the competing ordinary TCP flows to lose some of their original share of bandwidth. We compare our proposed protocol with MulTCP in more details in Section 4.

3. PROTOCOL ARCHITECTURE

We need to overcome two problems to support non-uniform bandwidth allocation in TCP. First, we need to modify the congestion control algorithm such that credits received (via acknowledgement packets) for data transmission can be reallocated from one flow to another flow. The goal is to allocate the bottleneck bandwidth to the TCP flows according to application-specified ratios $\{w_1, w_2, \ldots, w_n\}$, i.e., flow i will be allocated a proportion of $w_i / \sum_{\forall j} w_j$ of the bottleneck bandwidth. Note that here the term *bottleneck bandwidth* refers to the total amount of bandwidth that would have been received by n ordinary TCP flows passing through the network bottleneck, and so it will be smaller than the bottleneck link capacity in the presence of other competing TCP flows. Second, we need to regulate the *aggregate* traffic flows such that *as a whole* the group behaves in the same way as ordinary TCP flows so that other competing TCP flows (either from other hosts or from the same hosts but managed by ordinary TCP) will neither receive more nor less bandwidth than normal. The following sections present details of the proposed protocol architecture that achieves these two goals.

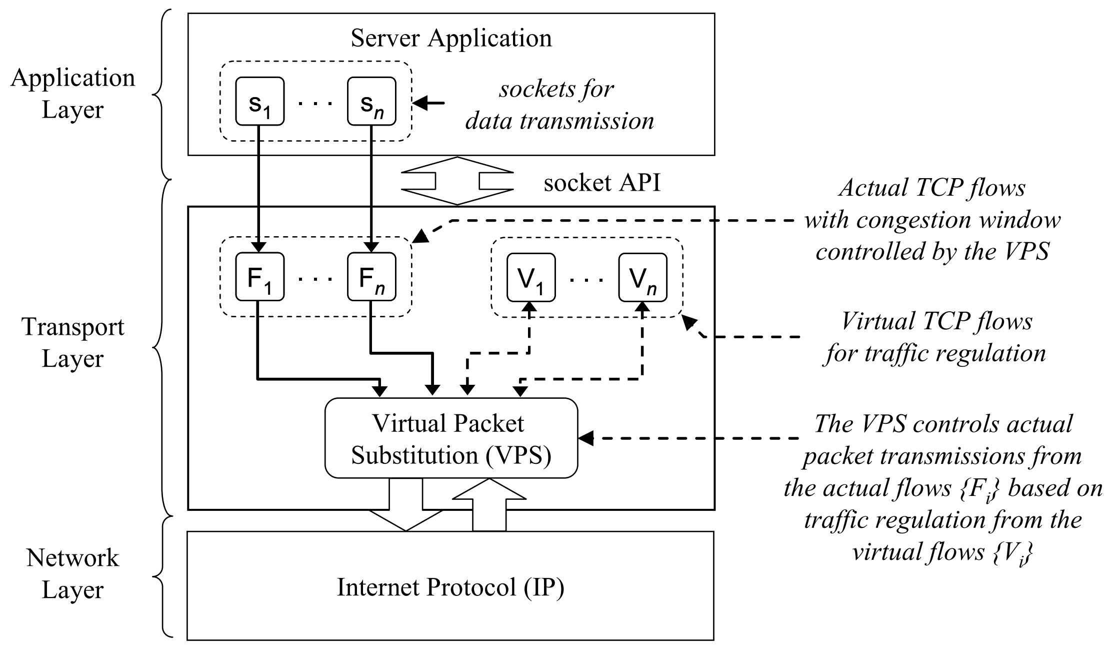

Application Layer
Transport Layer
Network Layer
Server Application
s1 . . . sn
sockets for data transmission
socket API
Actual TCP flows with congestion window controlled by the VPS
F1 . . . Fn
V1 . . . Vn
Virtual TCP flows for traffic regulation
Virtual Packet Substitution (VPS)
The VPS controls actual packet transmissions from the actual flows {Fi} based on traffic regulation from the virtual flows {Vi}
Internet Protocol (IP)

Fig. 1. Protocol architecture and interfaces to application and network layers.

3.1. Operating principle

In ordinary TCP each flow is independent from all other TCP flows. The application will submit data to a TCP flow for transmission using an application protocol interface such as sockets[6]. Upon receiving the application data, TCP will then construct one or more TCP segments and submit them to the IP layer for transmission, subject to the control of the congestion window and the receiver window of the TCP flow. The congestion window is computed by the sender using the well-known Additive Increase Multiplicative Decrease (AIMD) algorithm[9] and this window limits how much data the sender can send while waiting for the receiver's acknowledgements. Upon receiving data correctly, the receiver will send acknowledgement packets back to the sender. The size of the congestion window and the highest sequence number acknowledged by the receiver together determine whether the sender can send more data. The receiver window is also sent as part of the acknowledgement packet, informing the sender of the receiver's buffer availability for flow control purpose. As our focus is on network bottleneck rather than the receiver buffer being the bottleneck so we will assume the receiver window is not the limiting factor and only consider the congestion window in the following discussions. Future work will look into the reallocation of unused bandwidth due to limited advertised window to other flows in the group.

Fig. 1 depicts the proposed protocol architecture. When the server application creates a new TCP flow through the network programming API (e.g., by accepting an incoming connection through a stream socket), the transport will create two flows internally – an *actual flow* denoted by F_i and a *virtual flow* denoted by V_i. The actual flow implements the standard transport buffering functions and receives and buffers data from the application awaiting transmission. The actual flow, however, does not implement congestion control or maintain its own congestion window. Instead, the Virtual Packet Substitution (VPS) module is responsible for passing data from the actual flow to the IP layer for transmission.

3.2. Traffic regulation and bandwidth differentiation

On the other hand, the virtual flow implements the standard TCP congestion control algorithm such as the AIMD and maintains its own congestion window in exactly the same way as ordinary TCP does. However, data transfer in a virtual flow is simulated rather than physically implemented. When a virtual flow sends a new *virtual* TCP segment, it submits the virtual TCP segment to the VPS for processing. The VPS does not actually send the virtual TCP segment, but instead updates its internal counters to reflect the fact that one more TCP segment can be transmitted from the *actual flows*. In other words, the virtual flows are used solely for the purpose of running the standard TCP congestion control algorithms to compute how much data can be transmitted. This ensures that the outbound data traffic conforms to TCP's fair-sharing property.

Eventually, the VPS module replaces the virtual TCP segment by a physical TCP segment from one of the actual TCP flows. This packet substitution process however, is not fixed in terms of the mapping between virtual TCP flows and actual TCP flows. Instead, the substitution is performed dynamically to allocate the transmission quota to the actual TCP flows according to their application-specified bandwidth ratios $\{w_1, w_2, \ldots, w_n\}$.

For example, consider the case of two flows with bandwidth ratio of w_1:w_2=1:2, i.e., flow 1 and flow 2 are to receive 1/3 and 2/3 of the bottleneck bandwidth respectively. Then for every three virtual TCP segments generated by the two virtual TCP flows, the VPS will replace one of them with physical TCP segment from flow 1, and two of them with physical TCP segments from flow 2. In this way the two actual TCP flows will receive bandwidth according to their respective ratios.

3.3. TCP ACK translation

There is a subtle problem with the previous packet substitution algorithm. Specifically, when TCP acknowledgements are received from the clients, these acknowledgement (ACK) packets will be acknowledging TCP segments of the actual TCP flows because the transmitted TCP segments are really from the actual TCP flows rather than the virtual TCP flows. However, without proper acknowledgement information, the virtual TCP flows will not be able to run their congestion control algorithm to update the congestion window, which in turns control the transmission quota used by the VPS module.

To tackle this problem, we need to keep track of the mappings between the virtual TCP segments generated by the virtual TCP flows and the physical TCP segments transmitted from the actual TCP flows. Every time the VPS performs a packet substitution, it creates a *packet substitution record* (PSR) with fields $\{V_A, V_{Seq}, F_A, F_{Seq}\}$ where V_A, V_{Seq} are the virtual TCP flow's transport address (i.e., source and destination IP addresses and port numbers) and TCP segment sequence number; and F_A, F_{Seq} are the actual TCP flow's transport address and TCP segment sequence number. The new PSR entry is then appended to the corresponding actual TCP flow's PSR list.

Now when an ACK packet arrives from a client, the VPS will lookup the PSR entry in the corresponding actual TCP flow's PSR list, using the fields F_A and F_{Seq} as matching criteria. Next the VPS will generate a *virtual* ACK packet by substituting the actual address F_A by the virtual address V_A, and the actual ACK sequence number F_{Seq} by the virtual sequence number V_{Seq}. In other words the VPS performs a reverse substitution to translate the received actual ACK packet to a virtual ACK packet and sends it to the virtual TCP flow for running the congestion control algorithm. In practice, the PSR table can be implemented as a circular array of PSR entries. Given that the maximum receiver window is relatively limited (e.g., 128), the amount of memory consumed is insignificant. Moreover, as the sequence numbers of adjacent TCP segments are often offset by the same amount (i.e., one MTU), the VPS can lookup a PSR entry simply by using the ACK's sequence number to index directly into the PSR circular array, thereby reducing the processing complexity significantly.

There is, however, one more complication - most of today's Internet hosts run a variant of TCP called Reno with the SACK option[11] enabled by default[*]. With the SACK option the ACK packet may also include additional acknowledgements on discontinuous ranges of sequence numbers. The VPS in this case will generate, as needed, separate ACK packets for the virtual TCP flows. Finally, the processed PSR entries are then removed from the PSR list of the actual TCP flows.

[*] According to the statistics[13], about 90% of the host in the Internet use Windows 98, 2000, XP and Linux as the operating system and all these operating systems enable the SACK option by default[14-16].

4. PERFORMANCE EVALUATION

In this section we use simulation to evaluate performance of the proposed protocol and compare it with the standard TCP Reno[5, 12] and the MulTCP[8], of which the latter also supports bandwidth differentiation. To ease description we will use the name *VPS flows* to refer to traffic flows using the proposed bandwidth differentiation protocol and *TCP flows* to refer to traffic flows using the standard TCP protocol.

4.1. Performance metric

To facilitate comparison, we define a metric to quantify the protocols' accuracy in allocating bandwidth according to the specified ratios. Let there be N flows with application-specified ratios $\{w_1,w_2,\ldots,w_N\}$. Let $\{r_1,r_2,\ldots,r_N\}$ be the actual throughput measured in the simulation. Then for each flow we compute its allocation accuracy, denoted by A_i, from

$$A_i = \frac{r_i / \sum_{k=1}^{N} r_k}{w_i / \sum_{k=1}^{N} w_k} \tag{1}$$

where the numerator is the actual proportion of bandwidth received and the denominator is the proportion as specified by the bandwidth ratios. Therefore if the protocol performs perfectly the two will be the same and the allocation accuracy will be equal to 1. A value smaller/larger than 1 implies that the actual bandwidth received is less/more than that specified by the bandwidth ratios.

To evaluate the overall accuracy for all flows we compute the overall allocation accuracy, denoted by A, from

$$A = \frac{\left(\sum_{i=1}^{N} A_i\right)^2}{N\sum_{i=1}^{N} A_i^2} \tag{2}$$

with a range from 0 to 1, with 1 indicating perfect allocation and lower values indicating larger deviations from the specified bandwidth ratios.

To evaluate the protocols' bandwidth sharing property, we define a fairness ratio, denoted by F, from

$$F = \sum_{\forall i} r_i \Big/ \sum_{\forall j} s_j \tag{3}$$

where the numerator is the aggregate bandwidth of all VPS flows and the numerator is the aggregate bandwidth of all standard TCP flows. In our simulation the number of VPS flows and standard TCP flows are the same so a value of F=1 implies perfect fair sharing of bandwidth with competing TCP flows, and higher/lower values of F indicates that the VPS flows receive more/less bandwidth than the competing TCP flows.

4.2. Simulation setup

The simulator is developed using the NS2 version 2.28 simulator[10] using the network topology shown in Fig. 2. There are three types of network traffic, including N VPS flows, N Reno TCP flows and a UDP flow generating exponential background traffic at a rate of $40\times2\times N$ kbps. All flows pass through the same bottleneck with $4N$ Mbps bandwidth, 50ms delay, and using the droptail queueing discipline. All other links have 100Mbps bandwidth and 10ms delay. Unless specified otherwise, we use N=3 VPS flows with application-specified bandwidth ratios of {1, 4, 8} and three competing TCP Reno flows. For both TCP Reno and VPS flows we assume the sender always has data to send, and all senders and receivers have the SACK option enabled. Each simulation run lasts for 1 hour of simulated time.

4.3. Performance over different time scales

We first investigate the protocols' allocation accuracy in Fig. 3 with results computed over different time scales, i.e., with bandwidth averaged over different time windows. The results show that VPS flows have *perfect* allocation accuracy of 1 as the VPS module assigns transmission quota to the VPS flows strictly according to the specified bandwidth ratios. In comparison, MulTCP also achieves good allocation accuracy of about 0.85 but the accuracy decreases (with larger coefficient-of-variation) over shorter time scales. Fig. 4 shows the throughput against time when the time scale is 1s. We observe that while there are short-term variations due to TCP's congestion control algorithm, all three VPS flows maintain the specified bandwidth ratio at all times, including periods of slow start and congestion avoidance.

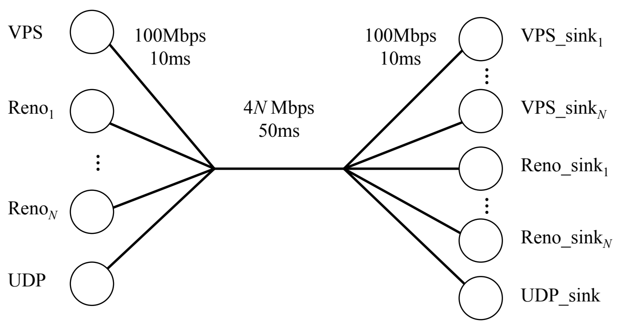


Fig. 2. The simulation topology.

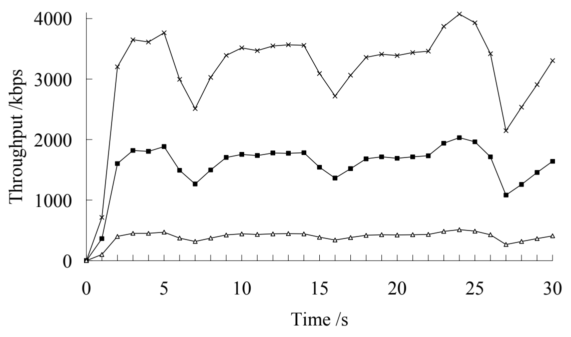
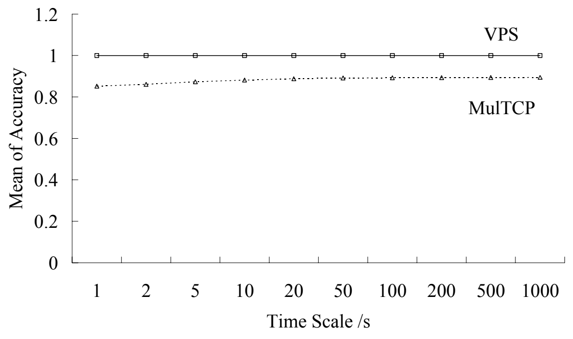


Fig. 4. Throughput of VPS flows against time.



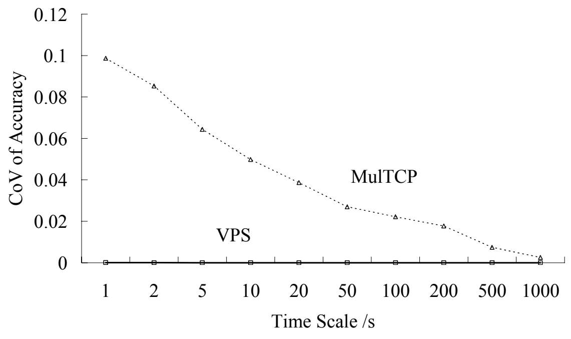
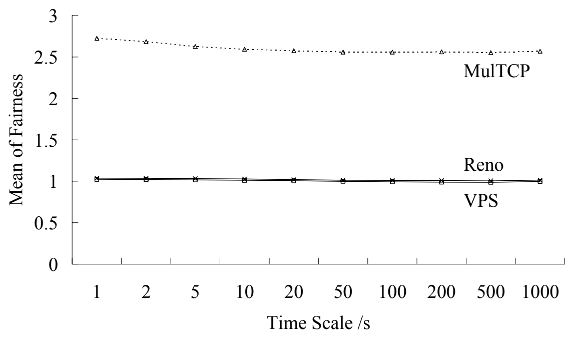


Fig. 3. Mean (left) and CoV (right) of allocation accuracy over different time scales.



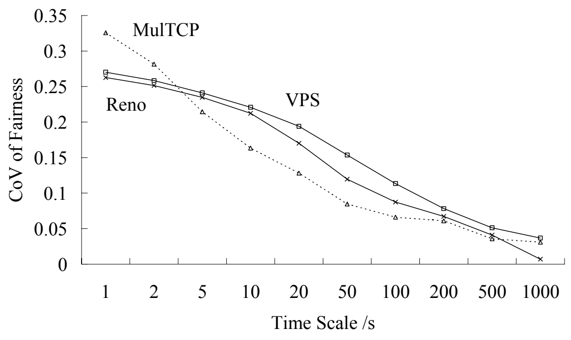


Fig. 5.   Mean (left) and CoV (right) of fairness over different time scales.

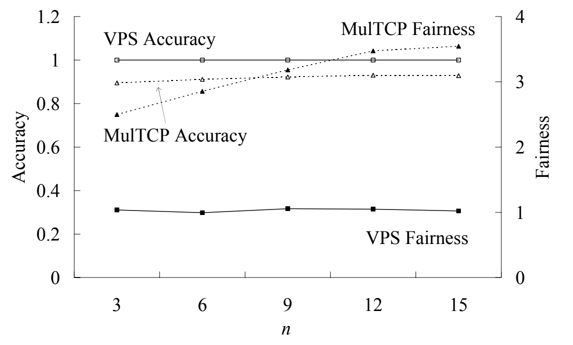


Fig. 6. Scalability to more traffic flows.

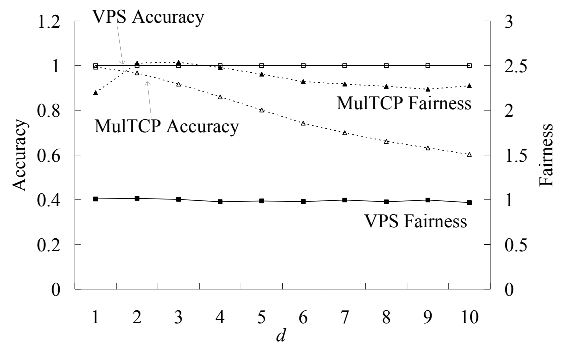


Fig. 7. Scalability to wider bandwidth ratios.

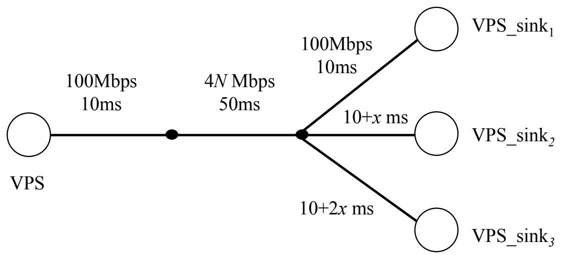


Fig. 8. Heterogeneous network

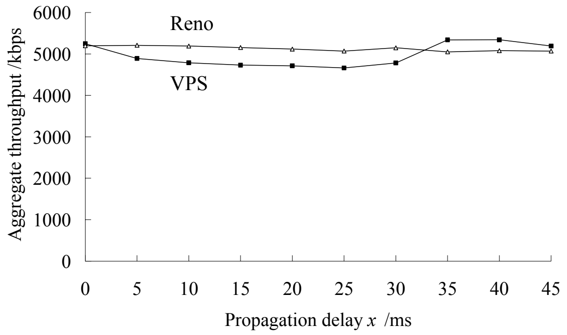


Fig. 9. Aggregate throughput of VPS and Reno flows in heterogeneous network

In the same experiment we also measured the fairness ratio of VPS, MulTCP, and TCP Reno and plotted the results in Fig. 5. We observe that both VPS flows and Reno TCP flows achieve a fairness of close to 1, suggesting that the proposed protocol can maintain fair bandwidth sharing with ordinary TCP flows. In comparison, fair bandwidth sharing is not part of the MulTCP protocol design goal and therefore MulTCP flows are substantially more aggressive than ordinary TCP flows.

All three protocols, including TCP Reno, show increased variation in fairness over shorter time scales. We conjecture that this is due to the short-term dynamics of TCP's congestion control algorithm in exploring the available network bandwidth and in reacting to short-term congestions triggered by packet loss.

4.4. Scalability

To investigate the protocols' scalability to more traffic flows, we re-run the experiments by varying the number of flows *N* from 3 to 15 and plot the results in Fig. 6 showing simultaneously both the allocation accuracy and fairness ratio using two vertical axes. In terms of allocation accuracy both VPS and MulTCP performs consistently when the number of flows is increased from 3 to 15. However, while VPS flows maintain consistent fairness, MulTCP flows become increasingly more aggressive when the number of flows is increased.

In another experiment, we vary the bandwidth ratios to investigate the protocols' performance when the bandwidth ratios are wider apart. We define a ratio difference, denoted by *d*, to set the bandwidth ratio for flow *i* to equal to $w_i=1+d(i-1)$. Thus larger values of *d* will widen the difference of bandwidth ratios between successive flows.

Fig. 7 plots the allocation accuracy and fairness for VPS and MulTCP. Again, the VPS flows perform consistently over the entire parameter range with negligible variations in both allocation accuracy and fairness. By contrast, the allocation accuracy of MulTCP decreases with wider bandwidth ratios as the higher-ratio MulTCP flows generate more bursty traffic, thus leading to more frequent packet loss.

4.5. Heterogeneous network

In the above simulations all the receivers have the same round trip time (RTT) to the sender. To investigate the effect of receivers with different RTTs, we conducted another set of simulations using the network topology shown in Fig. 8. Specifically, we vary the propagation delay difference *x* from 0ms to 45ms to implement heterogeneous RTTs. For example, when *x* is equal to 45ms, the RTT of the three receivers are equal to 140ms, 230ms and 320ms respectively.

Fig. 9 compares the aggregate throughput of the three VPS flows with the aggregate throughput of ordinary TCP flows under the same network setup. We observe that despite the differences in RTT for the three receivers, VPS can still maintain aggregate throughput similar to ordinary TCP. The authors are currently investigating the performance of VPS in even more complex network topologies to further evaluate its strengths and limitations.

5. CONCLUSIONS AND FUTURE WORK

This work presented a Virtual Packet Substitution (VPS) algorithm for the allocation of bandwidth among TCP flows originating from the same sender passing through the same network bottleneck to multiple receivers. As the VPS algorithm assigns transmission quota strictly according to the specified bandwidth ratios, it can achieve perfect bandwidth allocation accuracy over time scales as short as one second. Moreover, VPS can maintain excellent fairness with competing TCP flows by computing the transmission quota from virtual flows running the standard TCP Reno congestion control algorithm. The capability to allocate non-uniform bandwidth between TCP flows opens many new possibilities for network services. For example, a service operator may use bandwidth differentiation to provide different quality of service to users of different subscription levels (i.e., more bandwidth for premium subscribers). A media server may dynamically adjust the bandwidth ratios to react to quality feedbacks from clients, and so on. This work merely introduces a new tool and more works are needed to explore the applications and optimization of the newfound tool to various network applications and services.

ACKNOWLEDGEMENT

This work was funded in part by a Direct Grant, an Earmarked Grant (CUHK4211/03E) from the HKSAR Research Grant Council, and the UGC Area of Excellence in Information Technology Scheme (AoE/E-01/99). This work is affiliated with the Microsoft-CUHK Joint Laboratory for Human-centric Computing and Interface Technologies.

REFERENCES

1. H. Balakrishnan, H. S. Rahul and S. Seshan, "An Integrated Congestion Management Architecture for Internet Hosts," *ACM SIGCOMM Computer Communication Review*, Vol.29, Issue 4, pp.175-187, Oct. 1999.
2. V. N. Padmanabhan, "Coordinating Congestion Management and Bandwidth Sharing for Heterogeneous Data Streams," *Proc. NOSSDAV* 99.
3. H. Balakrishnan, V. N. Padmanabhan, S. Seshan, M. Stemm and R.H. Katz, "TCP Behavior of a Busy Internet Server: Analysis and Improvements," *Proc. IEEE INFOCOM*, pp.252-262, Mar. 1998.
4. W. R. Stevens, *TCP/IP Illustrated, Vol. 1*, Addison-Wesley, New York, 1994.
5. V. Jacobson, "Congestion Avoidance and Control," *Proc. of ACM SIGCOMM*, 1988.
6. W. R. Stevens, *UNIX Network Programming*, Prentice-Hall, Inc., Upper Saddle River, NJ, 1990.
7. P. Mehra, A. Zakhor and C. Vleeschouwer, "Receiver-driven Bandwidth Sharing for TCP," *Proc. IEEE INFOCOM*, pp. 1145-1155, Mar. 2003.
8. J. Crowcroft and P. Oechslin, "Differentiated End-to-End Internet Services Using a Weighted Proportional Fair Sharing TCP," *Computer Communication Review*, Vol.28(3), pp.53-67, Jul. 1998.
9. D. M. Chiu and R. Jain, "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks," *Computer Networks and ISDN Systems*, Vol.17, pp.1-14, 1989.
10. S. McCanne, S. Floyd, "ns-2 Network Simulator" – http://www.isi.edu/nsnam/ns/.
11. M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, "TCP selective acknowledgement and options," *RFC2018*, IETF, Oct. 1996.
12. V. Jacobson, "Modified TCP Congestion Avoidance Algorithm," Technical report, 30 Apr. 1990. ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt.
13. OS Platform Statistics, http://www.w3schools.com/browsers/browsers_stats.asp.
14. SACK Support for Various Operating Systems, http://www.psc.edu/networking/projects/tcptune/.
15. SACK Support for Windows .NET, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/ wcetcpip/html/cmcontcpselectiveacknowledgment.asp.
16. SACK Support for Linux 2.4 and after, http://www.linuxpakistan.net/man.php?query=tcp&apropos=0 &section=0&type=2.
17. D. Katabi, I. Bazzi and X. Yang, "A Passive Approach for Detecting Shared Bottlenecks," *Proc. IEEE Computer Communications and Networks*, pp.174-181, Oct. 2001.
18. O. Younis and S. Fahmy, "On Efficient On-line Grouping of Flows with Shared Bottlenecks at Loaded Servers," *Proc. IEEE Int. Conf. Network Protocols*, pp.175-184, Nov. 2002