To appear in INFOCOM'98

Zero Queueing Flow Control and Applications

H. T. Kung and S. Y. Wang
Division of Engineering and Applied Sciences
Harvard University
Cambridge, MA 02138, USA

Abstract

Zero Queueing Flow Control (ZQFC) is a new credit-based flow control method for ATM networks. The receiving node of such a flow-controlled link will have zero queue-occupancy in the steady state. ZQFC uses both link and VC flow control simultaneously over the link to implement the required rate adaptation for achieving zero queueing.

Because of its zero queueing property, ZQFC is able to solve a head-of-the-line blocking problem that may arise when multiple VCs share the same receiver buffer in implementing their flow control.

ZQFC works well with TCP traffic. When there is any queue buildup associated with a TCP connection in a shared buffer, ZQFC will identify the connection and take steps to reduce its arrival rate at the buffer. This will allow many TCP connections to share the buffer efficiently and fairly. It makes sense to deploy ZQFC for just a single link where performance improvement is critical. Simulations using real-life TCP code have demonstrated these advantages of ZQFC.

1. Introduction

A challenge in network flow control has been the design of adaptive schemes capable of adjusting rates or buffer allocations dynamically, in order to improve overall network performance. Zero Queueing Flow Control (ZQFC) described in this paper represents a general idea of adjusting the arrival rate of a connection at a shared buffer based on its buffer occupancy. The goal of ZQFC is that in the steady state, a connection will occupy no buffer space.

In this paper, ZQFC is presented as an ATM-layer credit-based flow control method, enhanced with a new adaptation scheme for VC buffer allocation. The adaptation assures that each VC will occupy zero or a very small amount of buffer space.

Due to its zero-queueing property, ZQFC solves a head-of-the-line blocking problem (i.e., the "hot-spot" problem) associated with those credit-based flow control methods which allow VCs to "over-subscribe" buffer at the receiver. We explain in Section 4 how ZQFC solves the hot-spot problem.

A major applications objective of ZQFC is its support for TCP traffic. We show that for a class of TCP load configurations, ZQFC can improve the network performance significantly. We use a metric in Section 7, the CAB number, to characterize this class. We point out, in Section 5.3, that it can be worthwhile to deploy ZQFC for just a single link where performance improvement is critical.

2. Notations and Assumptions

This paper uses the following notations and assumptions:

• Buff_Size	Buffer size in ATM cells. Buffers are denoted by black rectangles in diagrams; see, e.g., Figures 2 and 1.
• Link_BW	Link bandwidth. All links are assumed to have Link_BW = 1, unless otherwise explicitly stated.
• Cell_Time	Transmission time for one ATM cell on a link of Link_BW = 1.
• Link_RTT	Link round-trip time in Cell_Time.
• Packet_Size	TCP packet size, which is assumed to be 512 bytes.
• Win_Size	TCP window size, varying from 16 kB to 64 kB.

All performance results reported in this paper have been obtained from a simulator based on real-life TCP code. See Appendix A for a description of this simulator.

Each reported simulation result represents an average over a 2-minute real-time execution. We use this long, two-minute execution for simulations to ensure that measured performance will become stable before simulation ends. Typically, each of these results took 5 to 10 hours of 200MHz Pentium Pro CPU time to generate.

3. Background Information

When presented as an ATM-layer flow control method, Zero Queueing Flow Control (ZQFC) can be viewed as a credit-based ATM flow control method. It can be naturally implemented in conjunction with input-buffered ATM switches. Like Quantum Flow Control (QFC) [1], ZQFC uses both VC and link flow control simultaneously, to control flows of VCs over a link. Unlike QFC, ZQFC employs a novel buffer allocation scheme with the objective of achieving zero buffer occupancy, in the steady state, for every VC sharing the buffer. This section briefly reviews input- and output-buffered switches, and VC and link flow control.

3.1 Input-Buffered vs. Output-Buffered Switches

Switches can be classified as input-buffered and output-buffered switches. Figure 1 (a) or (b), respectively, shows two connections over a network made of input-buffered or output-buffered switches. One can view the output-buffered configuration as derived from the input-buffered configuration by moving all its buffers upstream by one position. For instance, buffer X in Figure 1 (a) moves upstream by one hop to arrive in its new position in Figure 1 (b).

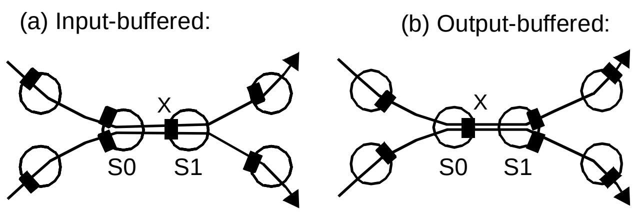


Figure 1: Configurations with (a) input-buffered and (b) output-buffered switches. Black rectangles denote buffers.

Output-buffered switches are easy to understand. Consider, for example, switch S0 of Figure 1 (b). Data arriving at the two input ports will flow immediately to the output buffer X. If the output buffer is full, packets are dropped at the entrance of the buffer. In principle, an output buffer needs to take in data at N-speed, if the switch has N input ports. Thus the implementation of output-buffered switches is not very scalable to large values of N.

Input-buffered switches are more scalable than output-buffered switches. Consider, for example, switch S0 of Figure 1 (a). Data arriving at the two input ports will stay in the input buffers, waiting to be sent to the output port. There is an internal flow control mechanism inside the switch which will inform the input buffers to forward data to an output port when there are transmission slots available on the output link. When an input buffer is full, arriving data from the corresponding input port may be dropped, but there will be no data drops at output buffers. All data paths in an input-buffered switch operate at 1-speed, independently of the value of N, although some control paths may need to operate at N-speed. Since bandwidth requirements for control paths are typically much smaller than data paths, input-buffered switches are much more scalable than output-buffered switches. Because of their high scalability, this paper will focus on ZQFC implementation for input-buffered switches.

3.2 Review of VC and Link Flow Control

VC flow control ensures that, for each VC, the transmitter of the link will not overrun the VC's allocated buffer at the receiver. It is well-known that the buffer allocation of each VC needs to be sufficiently large to allow the VC to transmit at a high rate. To avoid excessive use of buffer, it is therefore important that multiple VCs can dynamically share a fixed-size receiver buffer much smaller than the total buffer allocated to all these VCs.

This buffer "over-subscription" could result in buffer overflow when enough of these VCs want to use their allocated buffer space at the same time. "Link flow control" described below provides a solution to this problem.

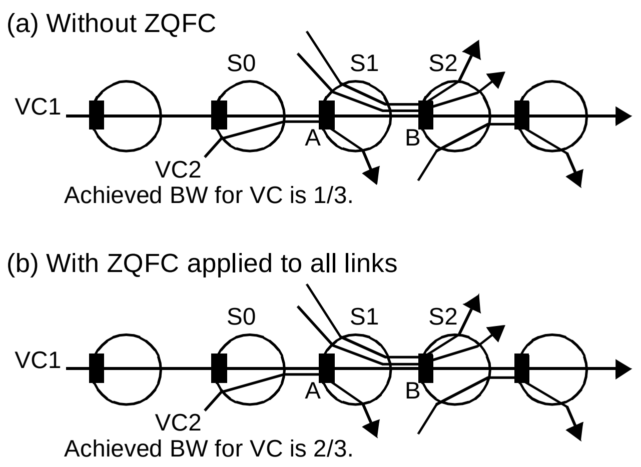
Link flow control prevents possible buffer overflow due to buffer "over-subscription", by ensuring that the total buffer space actually occupied by all the VCs, at any given time, does not exceed the available buffer space at the receiver. When the available space in the receiver's buffer becomes smaller than Link_RTT, link flow control will kick in to slow down or stop all the incoming VCs to prevent buffer overflow.

4. "Hot-Spot" Problem of Link Flow Control

Link flow control, when inappropriately used, can cause the so-called "hot-spot" problem. That is, when congestion develops at one part of the network, link flow control may unnecessarily slow down or stop VCs concerning other parts of the network.

Figure 2 (a) illustrates this "hot-spot" problem, which will arise when ZQFC is not used for this configuration. We assume that a round-robin scheduler is used at each output port in scheduling VCs which have both credits and data to send. Since VC1 shares with two other VCs on the link from S1 to S2, VC1 will get a bandwidth of 1/3. If VC1 enters into buffer A at a rate higher than 1/3, then the occupancy level of buffer A will build up, the available buffer space in A will shrink, and link flow control will kick in to reduce the overall flow rate from S0 to S1. The total flow rate over the link will keep decreasing until the VC1's arriving rate at buffer A has been reduced to 1/3.

Since a round-robin scheduler is used among VCs from S0 to S1, VC2 will output at the same rate as VC1. Because VC1's rate is 1/3, VC2 will have the same rate of 1/3, instead of the remaining link bandwidth of 2/3 left by VC1.


(a) Without ZQFC
S0
S1
S2
VC1
A
B
VC2
Achieved BW for VC is 1/3.
(b) With ZQFC applied to all links
S0
S1
S2
VC1
A
B
VC2
Achieved BW for VC is 2/3.

	(a) Without ZQFC	(b) With ZQFC on All Links
Achieved BW for VC2	1/3	2/3
Utilization of Link (S0, S1)	2/3	1

Figure 2: "Hot-spot" problem of link flow control.
(a) Without ZQFC: Hot spot at the link (S1, S2) can cause under-utilization of link (S0, S1).
(b) With ZQFC: Hot spot problem disappears.
A round-robin scheduler among outputting VCs is assumed at each output port. Black rectangles denote input buffers of size Link_RTT.

Thus, the total utilization of the link from S0 to S1 by VC1 and VC2 together is only 2/3. Our simulation has validated this under-utilization.

The example of Figure 2 (a) illustrates that when link flow control is used, a "hot-spot" originating at one part of a network can cause under-utilization at other parts of the network. That is, for Figure 2, congestion at the output link from S1 to S2 has caused under-utilization of the link from S0 to S1.

One can prove that, under round-robin scheduling, link flow control will force all the VCs over the same link to run at the same rate when the total available space in the shared buffer falls below Link_RTT. This rate shared by all the VCs will be equal to the smallest VC rate allowed by downstream congestion. For example, in Figure 2 (a), VC1 and VC2 will output from S0 at the same rate of 1/3. This is because downstream congestion conditions will allow VC1 to transmit at the rate of 1/3 and VC2 at the rate of 1, and the rate 1/3 is the smaller of the rates 1/3 and 1.

Under-utilization caused by the "hot-spot" problem can severely impact the performance of a network. If any VC is downstream congested somewhere, all other VCs sharing the same link can be forced to run at the same slow rate as this slow VC. All these VCs will in turn slow down other VCs in other parts of the network. The chain reaction can quickly bring the whole network to a halt.

Simulations have demonstrated that the "plain QFC" [1] (PQFC), the primitive QFC scheme without any buffer allocation adaptation, indeed exhibits this "hot-spot" problem. In addition to the configuration of Figure 2 (a), our simulation results have shown, for example, that when PQFC is used for the configuration of Figure 9, the utilization of the link from S0 to S1 is only 50%. This simulation result is completely predictable using the reasoning given above. That is, for the three VCs over the link from S0 to S1, the smallest VC rate allowed by downstream congestion is 1/6. Thus, each of the three TCPs in Figure 9 will end up with the same rate of 1/6, and the total rate achievable by them is 3*(1/6) or 50%.

5. Description of ZQFC Algorithm

ZQFC is designed to avoid the "hot-spot" problem described above. Unlike previous credit-based ATM flow control methods, ZQFC makes sure that, in the steady state, the shared buffer in the receiver is not occupied, and therefore link flow control will not kick in. Since link flow control does not generally take place, the "hot-spot" problem described above is avoided.

More precisely, when ZQFC is applied to a link, the receiver follows the following two rules in allocating buffer space for the VCs over the link:

- ***Rule 1:*** *If a VC starts to build up a queue, i.e., the VC is above some high threshold, and if the remaining buffer space shared by all the VCs over the link is less than Link_RTT, then reduce the VC's buffer allocation.*

  By the property of credit-based flow control [4, 7], reducing a VC's buffer allocation will reduce its arriving rate, and will eventually clear its queue occupancy. Note that, when the remaining shared buffer space is Link_ RTT or more, link flow control will not affect the overall flow rate over the link. Thus, in this case, there is no need to reduce the VC's arriving rate.

- ***Rule 2:*** *If the VC queue is empty or below some low threshold, then increase the VC's buffer allocation until Link_RTT is reached.*

  This will guarantee that each VC will eventually get its fair-share use of the link. An interesting consequence of Rule 2 is that idle VCs will eventually get full buffer allocation of Link_RTT. Therefore, when any of these VCs has data to send, it can send them out instantly at the full link rate. (Note that the buffer allocated to each VC is only a virtual buffer rather than a real buffer. Even if every VC eventually receives its full buffer allocation of Link_RTT, the total real buffer can be much smaller than n*Link_RTT where n is the number of VCs. Link control will prevent the overflow of the real buffer.)

This ZQFC adaptation on VC buffer allocation is performed only at the receiver, transparent to the credit protocols between the transmitter and receiver. After having decided to change the buffer allocation of a VC, the receiver will simply reflect the change in the credit amount sent to the transmitter for that VC [7]. Thus, the adaptation will only affect the contents of protocol messages, not the protocols themselves. This implies, for example, that the QFC protocols messages [1] can be used without any modification to implement the adaptation.

The ZQFC scheme used to derive all the simulation results reported in this paper increases VC buffer allocation by a factor of 1.01, and decreases it by a factor of between .5 and 1, inversely proportional to the current queue occupancy of the VC.

Every 2*Link_RTT time, the receiver of the link will check the amount of available space in the input buffer. If the space is less than Link_RTT, the receiver will scan all the arriving VCs over the link to find those whose queue occupancy is above some high threshold (e.g., 20 cells). The receiver will reduce the buffer allocations of these VCs, as described in Rule 1 above. The receiver will also scan all the arriving VCs over the link to find those VCs whose queue occupancy is below some low threshold (e.g., 5 cells). The receiver will increase the buffer allocations of these VCs as described in Rule 2 above.

Our simulation results have shown that an input buffer of 2*Link_RTT cells is large enough for ZQFC to achieve high link utilization. The buffer needs to be at least Link_RTT to absorb the Link_RTT feedback delay inherent to the ZQFC adaptation. By increasing the buffer to 2*Link_RTT, the chance that the available buffer space becomes less than Link_RTT will be significantly reduced. Consequently, by Rule 1 above, the chance that link flow control will kick in to reduce the overall flow over the link is also significantly reduced.

For simple configurations such as that of Figure 2 (b) which greedy traffic sources, an input buffer of size LinK_RTT is sufficient to achieve high link utilization. Our simulation has indeed shown the result of Figure 2 (b), that is, the achieved bandwidth of VC2 is 2/3 and the total utilization of the link from S0 to S1 is 100%.

5.1 Comparisons with Other Adaptation Schemes

ZQFC is fundamentally different from other adaptation schemes in the credit-based flow control area. Previous buffer allocation adaptation methods in [4, 5] are based on recent bandwidth usages of VCs rather than their queue occupancy. These methods tend to occupy VC queues much more than ZQFC and thus reduce the available shared buffer that can be allocated to uncongested VCs. Since a VC's maximum data rate is bounded by the amount of buffer allocated to it, these methods in [4, 5] in general have inferior performance, compared to ZQFC. In particular, in these methods idle VCs will only get little allocation since they use no bandwidth, and as a result, they can not ramp up quickly when they do have data to send. Moreover, measuring the bandwidth usage of a VC is generally more costly than measuring its queue occupancy.

Another adaptive scheme for buffer allocation has been reported in [11]. The method does not give differential treatments among VCs in the sense that it always allocates the same buffer space to all the VCs sharing the same link in either congested or non-congested condition. This adaptation algorithm tends to bounce between congested and non-congested conditions, and waste a lot of link bandwidth due to link flow control. We have implemented the method on our simulator and measured its performance. For example, for the configuration oft Figure 4 (a), the method in [11] can only achieve a low utilization of 29% for the link (S0, S1). In contrast, ZQFC can achieve a utilization of 95% for the same configuration.

5.2 ZQFC for Input-Buffered and Output-Buffered Switches

ZQFC can be naturally implemented on input-buffer switches. VC flow control over a link is implemented with per-VC flow control messages from the downstream node to the upstream node. Figure 3 (a) depicts how link flow control can be implemented.

For output-buffered switches, each input port of a switch will have a dedicated link FIFO of size Link_RTT, as shown in Figure 3 (b). This buffer can be viewed as a part of the link hardware. Cells in this buffer will be forwarded to an output buffer only when there is space available in the output buffer. Thus, ZQFC implementation for an output-buffered switch assumes a dedicate FIFO for each input link, and an internal flow control mechanism inside the switch.

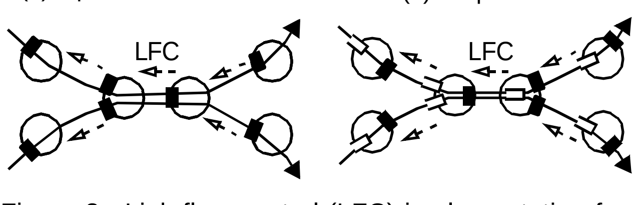


Figure 3: Link flow control (LFC) implementation for (a) input-buffered and (b) output-buffered switches. Arrows denote directions of credit flows.

5.3 Applying ZQFC to a Single Link

ZQFC can be applied strategically to individual links for which performance improvements are most desirable. It turns out that implementing ZQFC on just a single link can already improve network performance for TCP traffic substantially, as demonstrated in Sections 8 and 9 and explained in Section 10. For the rest of the paper, whenever ZQFC is used, we will state explicitly whether it is applied to one link or to all links.

6. Input- and Output-Buffered Switches for TCP Traffic: Performance Examples

Improving network performance for TCP traffic is a main objective for ZQFC. This section describes some related background issues on the performance of input-buffered and output-buffered switches for TCP traffic.

Consider the configuration of Figure 4 where bandwidth of downstream links are limited. For example, for the last link out from S1 in Figure 4 (a), Link_BW = 1/256. (In this paper, the notation [α] means that the maximum-possible bandwidths of all the TCP connections in question is α due to downstream congestion or link bandwidth limitation.)

The output-buffered configuration of Figure 4 (a) achieves a utilization of 94% while the input-buffered of Figure 4 (b) achieves only 18%. The poor performance of the input-buffered case is caused by the fact that the shared input buffer of switch S1 is filled by packets waiting to be output on relatively slow downstream links. The filled buffer will cause packet dropping for all the TCP connections. (As Table 3 shows, ZQFC will fix this problem.)

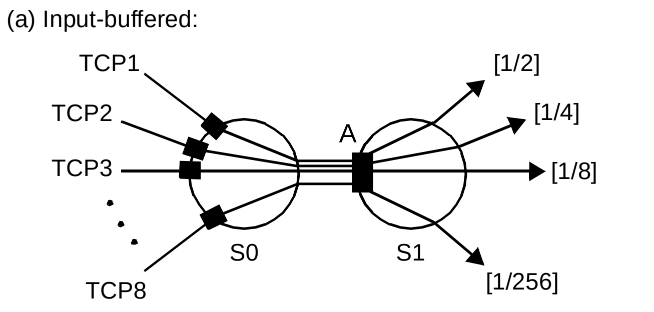
For the rest of this paper we will report performance improvement of ZQFC for TCP traffic on both input- and output-buffered configurations. (Refer to Section 5.2 for ZQFC implementations on these two configurations.)

7. Some Challenging Load Configurations and the CAB Metric

We show in this section a class of challenging TCP load configurations for which, as to be shown in later sections, ZQFC can improve the network performance significantly. Section 7 describes a metric, the CAB number, which will be used to characterize this class of loads. Section 7.3 demonstrates that the network performance decreases as the CAB number decreases.

7.1 CAB Numbers

A collection of TCP connections over a link tend to achieve a high link utilization when their downstream congestion is small. To characterize various degrees of downstream congestion, we use a notion called "the CAB number", which stands for "Configuration Allowed Bandwidth" relative to the bandwidth of the current link. The CAB number of a link is the maximal total bandwidth that all the TCP connections over the link can possibly achieve, subject to the given downstream congestion or link bandwidth limitation, divided by the bandwidth of the current link.



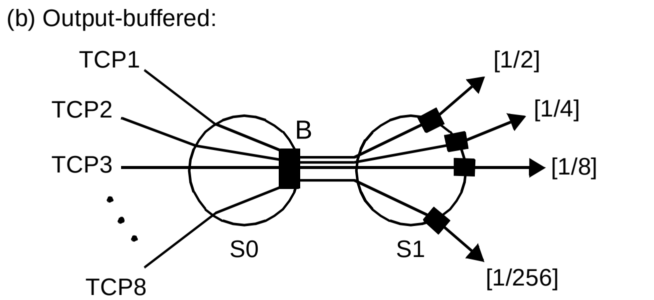


Figure 4: (a) Input-buffered and (b) output-buffered configurations where switch S1's output links have various Link_BW values smaller than 1. (Buff_Size = 960 for buffers A and B, Buff_Size = 50 for all the other buffers; Win_Size = 16 kB; Link_RTT = 620 for the link from S0 to S1; and Link_RTT = 10 for all other links.)

We illustrate the CAB number for the shared link (S0, S1) in Figures 5, 6, and 7. For Figure 5, each of the three TCP connections is not congested downstream and thus can achieve a maximal bandwidth of 1. Consequently, the maximum-possible total bandwidths achievable by the three TCPs is 3. This number divided by the shared link's bandwidth (which is 1) is still 3. Therefore the CAB number for the link is 3. We denote the link as a CAB = 3 link.

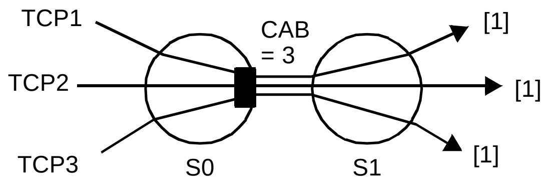


Figure 5: Single-bottleneck CAB = 3 configuration. The three competing TCPs sharing the link (S0, S1) have no downstream congestions. The link is thus a CAB = 3 link. (Buff_Size = 300; Win_Size = 16 kB; and Link_RTT = 80 for all links.)

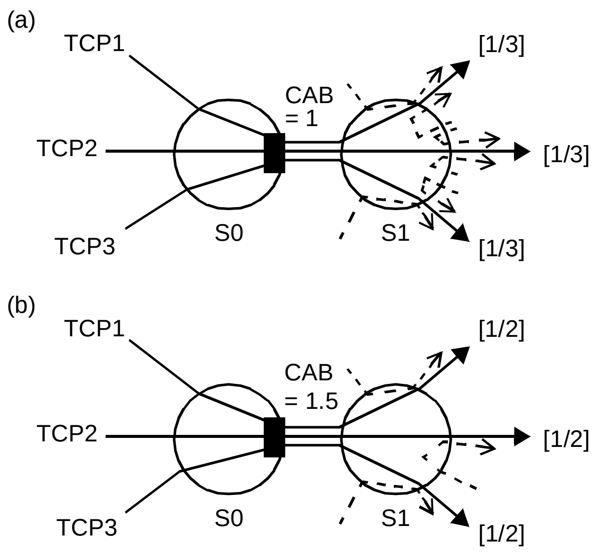


Figure 6: Multi-bottleneck configurations where the three TCPs are subject to congestion at both switches: (a) CAB = 1 configuration, and (b) CAB = 1.5 configuration. (Buff_Size = 300; Win_Size = 16 kB; and Link_RTT = 80 for all links.)

For Figure 6 (a), each TCP connection experiences two bottlenecks, one at S0 and another one at S1. For example, TCP1 competes with TCP2 and TCP3 at its output port of switch S0, and also competes with other two TCPs at its output of switch S1. After the (S0, S1) link, TCP1 competes at S1 with the other two TCPs, and thus TCP1 can achieve a bandwidth of at most [1/3]. The same is true for TCP2 and TCP3. Thus the maximum possible aggregate bandwidth that the three TCP connections can achieve is 1. Therefore, the CAB number for the shared link (S0, S1) is 1.

For Figure 6 (b), each of the three TCP connections can achieve a maximum downstream bandwidth of [1/2]. Thus, the CAB number for the shared link (S0, S1) is 1.5.

Figure 7 gives another example of links with CAB = 1, involving a total of 12 TCP connections. This configuration illustrates the point that a CAB = 1 configuration can arise "naturally" with evenly distributed TCP connections.

Note that the maximum bandwidth achievable on a link is bound above by its CAB number. We define the utilization of a link as follows. Let Y be the measured usage of the link, and suppose that CAB number is λ. If the λ is equal or greater than 1, then the link utilization is defined to be Y. Otherwise, the link utilization is defined to be Y / λ.

7.2 High Utilization under No Downstream Congestion

As mentioned earlier, high utilization of a link can generally be expected under no or light downstream congestion. An example is the "single-bottleneck" configuration of Figure 5. For this configuration, each of the three competing

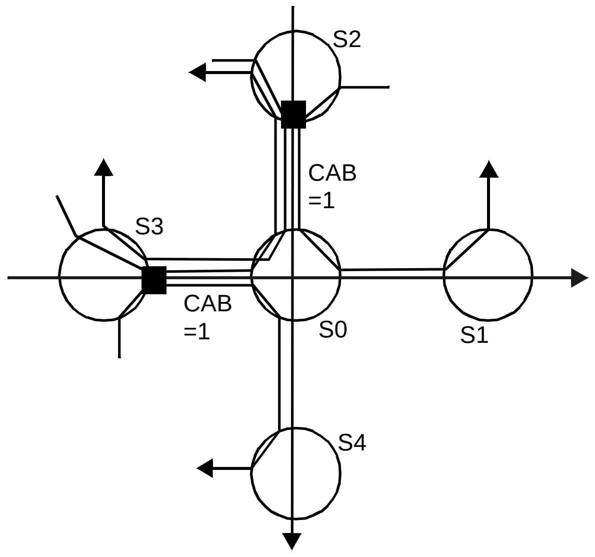


Figure 7: Multi-bottleneck CAB = 1 configuration, with 12 TCP connections symmetrically distributed. Switches S1, S2, S3 and S4 each have three TCPs going to the other three switches. To avoid complicating the diagram, only those six TCPs from S2 and S3 are drawn. Since over the link between any two connecting switches there are three competing TCPs in either direction, each TCP can get at most 1/3 of the link bandwidth. Thus, each of these links has CAB = 1. (Buff_Size = 300; Win_Size = 16 kB; and Link_ RTT = 80 for all links.)

TCP connections over the bottleneck link experiences no further congestion in other parts of the network. Thus, the CAB number for the link has a high value of 3. Under this situation, the TCP connections in aggregate will use the link efficiently. For example, in this case they will achieve a high utilization of 99.9%.

Although achieving high utilization in aggregate, the three TCP connections exhibit an oscillatory behavior, as depicted by Figure 8. Each TCP will periodically time-out for a lengthy period and resume later at a rate much higher than 1/3. The three TCPs rarely transmit all at the same time. TCP simulation results of this paper all exhibit this oscillatory behavior, when ZQFC enhancements are not applied to the link.

The high link utilization for the single-bottleneck configuration of Figure 5 results from the fact that, at any given time, at least one TCP connection can make good use of the link bandwidth. Thus multiple TCP connections can, in aggregate, achieve high utilization, although each of them may operate in an oscillatory manner.

We note that the oscillatory behavior of Figure 8 will disappear after ZQFC is applied to the bottleneck link. Our simulations have confirmed this, although because of space limitation this result is not shown in this paper.

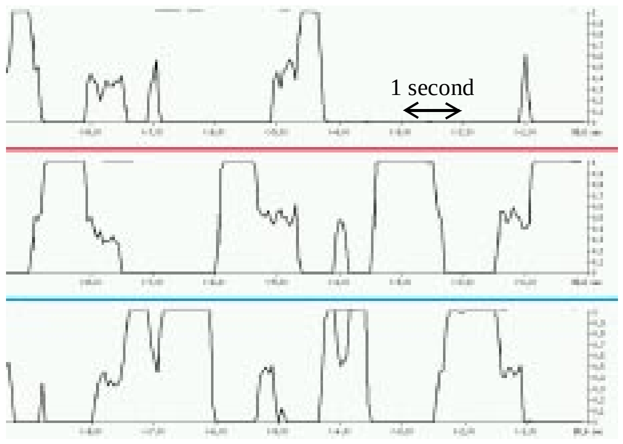


Figure 8: Oscillatory behavior of the TCP connections of Figure 5.

7.3 Network Performance as a Function of the CAB Number

Table 1 demonstrates that for Figures 5, 6, 7, and 9, when ZQFC is not applied to the shared link (S0, S1), the link utilization will decrease as its CAB number decreases.

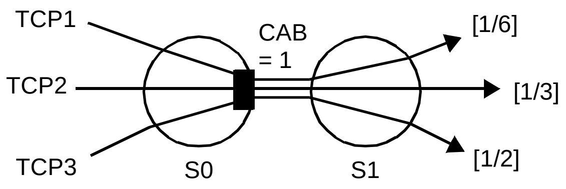


Figure 9: Multi-bottleneck CAB = 1 configuration, which is similar to that of Figure 6 (a) or (b), but has uneven downstream congestion. (Buff_Size = 300; Win_Size = 16 kB; and Link_RTT = 80 for all links.)

Case #	# TCPs	Network Config.	CAB #	ZQFC on All Links	Without ZQFC
1	3	Fig. 6 (a)	1	99	33
2	3	Fig. 9	1	99	61
3	3	Fig. 6 (b)	1.5	99	75
4	3	Fig. 5	3	99	99
5	12	Fig. 7	1	99	60

Table 1: Link utilization, in percentage (%), increases with the CAB number of the link, for the configuration of Figure 9.

Table 2 shows a similar result for the configuration of Figure 9, assuming that the changes in the CAB number result from increase in the bandwidth of the link (S0, S1).

Case #	CAB #	Network Config.	ZQFC on All Links	Without ZQFC
6	.5	Fig. 9	99	46
7	1	Fig. 9	99	61
8	2	Fig. 9	99	75
9	4	Fig. 9	99	84
10	8	Fig. 9	99	90

Table 2: Link utilization, in percentage (%), increases with the CAB number of the link, for the configuration of Figure 9.

8. Performance Improvement by ZQFC for Input-Buffered Switches

Consider the input-buffered configuration of Figure 4 (a). Recall from Section 6 that without ZQFC, the utilization of the link (S0, S1) is only 18%. Suppose ZQFC is applied to just the link (S0, S1). Then because of zero-queueing, flows destined to slow links out from S1 will no longer be able to occupy much of buffer X. This allows all the TCPs to use the buffer fairly, and as a result, they no longer time-out due to packet losses at the buffer. This results in the high utilization of the link (S0, S1) at 90%, as depicted in Table 3.

Case #	# TCPs	Network Config.	CAB #	ZQFC on one Link (S0, S1)	Without ZQFC
11	2	Fig. 4 (a)	1	90	18

Table 3: Utilization in percentage (%) for input-buffered switches configuration of Figure 4 (a).

ZQFC can also improve fairness among competing TCPs with different RTTs. Table 4 shows improved fairness between two TCPs for the configuration of Figure 10 when ZQFC is implemented in the link (S0, S1)

The results of Table 4 show that, without ZQFC, there is a strong bias against connections with long end-to-end RTTs. These results are consistent to similar results published previously [2].

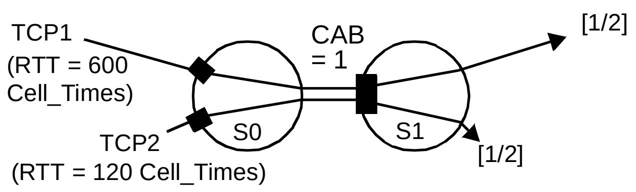


Figure 10: Input-buffered switches configuration similar to Figure 4 (a), with two competing TCPs of different RTT values.

	ZQFC on One Link (S0, S1)	Without ZQFC
Achieved BW for TCP1	.42	.12
Achieved BW for TCP2	.49	.48

Table 4: Fairness improvement by ZQFC for the input-buffered switches configuration of Figure 10.

9. Performance Improvement by ZQFC for Output-Buffered Switches

For output-buffered switches, ZQFC can improve network performance for TCP traffic using the implementation described in Section 5.2. Table 1 summarizes performance, with or without ZQFC, for configurations of Figures 6, 7, 5 and 9. Note that the performance improvements resulting from ZQFC are significant when the CAB number is 1 or near 1.

For the configuration of Figure 9, we have also simulated the case when ZQFC is implemented only on the link (S0, S1) rather than all links. Table 5 summarizes utilization improvement due to ZQFC in this case

Case #	# TCPs	Network Config.	CAB #	ZQFC on One Link (S0, S1)	Without ZQFC
12	3	Fig. 9	1	75	61

Table 5: Utilization in percentage (%) for the configuration of Figure 9.

10. Analysis of Performance Improvement by ZQFC for TCP Traffic

We provide explanations to the performance improvements of ZQFC on TCP traffic, as demonstrated in the two preceding sections. Recall that a TCP connection will keep growing its congestion window as long as there are no packet losses. At a congested gateway, a TCP connection, independently of its actual transmission speed, can put its maximum window worth of packets in the shared gateway buffer. When this buffer is full, any new TCP connections with small windows will easily time out when their packets are dropped at the gateway [9]. This can result in low utilization, unfairness and long delays.

ZQFC will identify these offending TCPs whose packets are occupying the buffer beyond their fair use, and then clear their buffer occupancy by slowing down their arrival rates at the buffer. Even if ZQFC is just applied to a single link, the slowed down rates will force the offending TCP to drop packets at the upstream node. As a result, after a network RTT the TCP sender will reduce the congestion window of the connection and slow down its transmission speed, as desired. This is the reason why single-link or single point ZQFC can work, as demonstrated in Tables 3, 4 and 5.

11. Concluding Remarks

ZQFC, as its name suggests, has the objective of achieving zero queue-occupancy in the steady state. This property enables ZQFC to solve a head-of-the-line blocking problem that may arise when multiple VCs share the same buffer in implementing their flow control. Consequently ZQFC can allow for aggressive buffer over-subscription, while achieving high link utilization and instant bandwidth ramp-up for idle VCs.

The results of Sections 8 and 9 have demonstrated that ZQFC can improve network performance for TCP traffic substantially. The performance gain can be realized even if ZQFC is applied only to a single link.

Zero queueing in the steady state, in addition to the benefits mentioned above, can also shorten the end-to-end queueing delay, and make it predicable. This can be important for real-time and interactive application programs.

To be specific, this paper presented ZQFC and its performance in the context of ATM networks. The basic ideas of zero queueing are applicable to other networks. In a future paper, we will describe an implementation of ZQFC on an IP network.

Acknowledgments

This work was supported in part by research funding from Nortel and Sprint.

References

[1] The QFC Alliance, *Quantum Flow Control*, Version 2.0, 1995 (available via http://www.qfc.org)

[2] S. Floyd, and V. Jacobson, "On Traffic Phase Effects in Packet-Switched Gateways." *Internetworking: Research and Experience*, Vol.3 No.3, September 1992, pp.115-156

[3] S. Keshav, "REAL: a Network Simulator", Report 88/472, Computer Science Dept., Univ. of California at Berkeley, 1988

[4] H. T. Kung, T. Blackwell and A. Chapman, "Credit-Based Flow Control for ATM Networks: Credit Update Protocol, Adaptive Credit Allocation, and Statistical Multiplexing,". *SIGCOMM '94*, pp. 101-114

[5] H. T. Kung and K. Chang, "Receiver-Oriented Adaptive Buffer Allocation in Credit-Based Flow Control for ATM Networks," *INFOCOM '95*, pp. 239-252

[6] H. T. Kung and A. Chapman, "The FCVC (Flow-Controlled Virtual Channels) Proposal for ATM Networks," Version 2.0, 1993. A summary appears in *Proc. 1993 International Conf. on Network Protocols*, pp. 116-127

[7] H. T. Kung and R. Morris, "Credit-Based Flow Control for ATM Networks," *IEEE Network Magazine*, March/April 1995, pp. 40-48

[8] H. T. Kung and S. Y. Wang, "Client-Server Performance on Flow-Controlled ATM Networks: A Web Database of Simulation Results," *INFOCOM'97*, pp. 1220-1228

[9] D. Lin and R. Morris, "Dynamics of Random Early Detection," *SIGCOMM '97*

[10] S. McCanne and S. Floyd. Ns (network simulator), 1995 (available via http://www-nrg.ee.lbl.gov/ns)

[11] C. Ozveren, R. Simcoe, and G. Varghese, "Reliable and Efficient Hop-by-Hop Flow Control," *SIGCOMM '94*, pp. 89-100

[12] A. Romanow and S. Floyd, "Dynamics of TCP Traffic over ATM Networks", *IEEE Journal on Selected Areas in Communications*, Vol. 13, No. 4, May 1995, pp. 79-88

Appendix A. Simulator Used in This Paper

To study the performance of ZQFC and other related methods, we have developed a "high-fidelity" simulator which uses real-life TCP/IP implementations on real hosts. It is important that we have a high degree of confidence on the correctness of the simulator.

Our simulator architecture differs from traditional ones in how it integrates the various simulation programs for the following functions:

1. Links with various delays and bandwidths.
2. ATM switches which forward cells and implement various ATM-layer protocols.
3. Hosts that use TCP/IP protocol to send and receive packets.
4. Application programs that generate network traffic.

Unlike traditional approaches, our simulator architecture does not combine parts 1, 2, 3 and 4 together to form a single complex program. Instead, our simulator only combines 1 and 2 to form an ATM network simulator. When 3, 4 and the ATM network simulator run concurrently in a BSD UNIX environment, their executions together simulate an ATM network and generate network traffic.

In fact, our network simulator need only simulate a network of ATM switches (part 1 and 2). The required TCP/IP protocol processing at hosts in a simulated ATM network is simulated by the "real-life" TCP/IP code in the simulation host's kernel by using tunnel network interfaces found in most UNIX environments. A tunnel network interface is no different from an Ethernet interface except that it is a pseudo interface with no physical network attached to it. Application programs used in the simulation system are real-world applications, which can be existing or yet-to-be-developed programs. An application program sends/receives its data to/from the kernel for transmission/reception via the normal socket system calls. The kernel, acting as any host in a simulated ATM network, then uses selected protocols (e.g., TCP, UDP or IP) to send/receive these data to/from the tunnel network interface associated with a VC in an simulated ATM network. This application program, thinking that its data is exchanged over a real physical network, never knows that its data is in fact exchanged over the simulated ATM network.

The architecture of our simulator is depicted in Figure 11. The

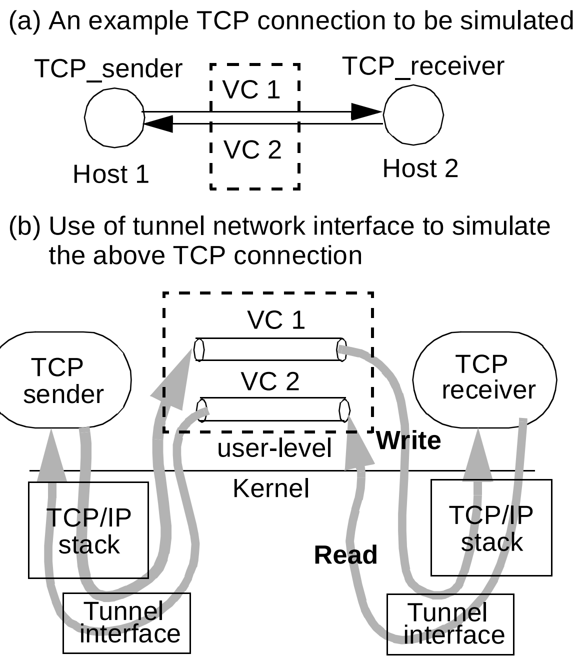


Figure 11: Use of the tunnel network interface to simulate a TCP connection in an ATM network. The top and bottom rectangular boxes, with dash-line boundaries, stand for an ATM network and simulator, respectively.

kernel part of our simulator is NetBSD 1.2_BETA, derived from BSD 4.4, with the "Reno" version of the TCP/IP protocol code.

Our network simulator is an ATM simulator [8], which has been used extensively for more than a year in testing credit-based flow control protocols over ATM networks, especially the QFC scheme [1]. To emulate packet switching, we have implemented EPD [12] on our simulator.

Our application programs are two simple network programs (stcp and rtcp), which are basically the TCP-sender and TCP-receiver. The load generated by the TCP-sender is greedy, in the sense that the TCP-sender always has data to send whenever it is allowed to send under the TCP flow control.