EECS 122 - COMMUNICATION NETWORKS - Fall 1997

Notes on Congestion Control Jean Walrand

Congestion control is essential to the acceptable operations of data networks such as TCP/IP and ATM. Current TCP congestion control strategies are not suitable for many applications. We discuss basic objectives and mechanisms and we then explore a few representative congestion control strategies.

CONTENTS:

Objectives
Mechanisms
Delay/Window Mechanism (Vegas)
Loss/Window Mechanisms (Tahoe, Reno)
Additive Increase - Multiplicative Decrease
Incompatibility of Reno and Vegas
Rate-Based Control: ABR in ATM
A Few Concluding Thoughts
Index

Objectives

The objective of congestion control is to share the network resources efficiently and fairly. Efficiently means that the links are used close to their capacity and that the backlogs in the buffers are small. Fairly is more difficult to define. One definition is that a fair allocation maximizes the minimum average rate per connection.

Mechanisms

Mechanisms for congestion control include reservation and pricing, priority, rate control, and window control. The mechanisms depend on the information they use: The mechanisms can be based on delays (measured round-trip times), on losses of packets (duplicated acknowledgments or time-outs), and on information from routers about congestion. [Back to Top]

Delay/Window Mechanism (Vegas)

We first explain a simple congestion control mechanism that adjusts the window of a retransmission protocol based on round-trip times. Figure 1 illustrates the setup where a single connection uses a bottleneck router with transmission rate C and with a round-trip delay equal to D when there is no queue in the buffer. The source uses a window with size W and we denote the queue size by x. The measured round-trip time is T.

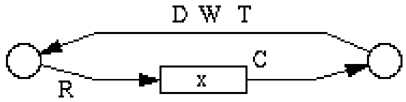


Figure 1: One connection through a bottleneck router

Note that T = D + x/C, W = RD if R < C and W = CD + x if R = C. The source can then determine the router backlog x by calculating x = max{W - RD, 0}. To evaluate D, the source calculates min{T}, the minimum of the round-trip time it has observed. From this, we conclude that the source can use a simple algorithm to adjust x to a small target value, say x = 3:

> **Algorithm 1** (Vegas). Source adjusts W according to following rules:
> If W - RD < 1, increase W;
> If W - RD > 3, decrease W.

Let's try to see how this algorithm would behave if many connections were to implement it. This situation is shown in Figure 2.

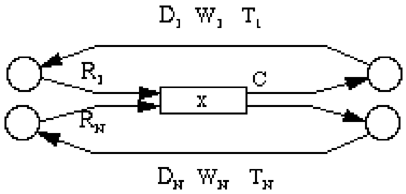


Figure 2: N connections share a bottleneck router

We assume that the N sources use Vegas. In fact,

> **Algorithm 1'** (Vegas). Source k adjusts W_k according to following rules:
> If W_k - R_k D_k < 1, increase W_k;
> If W_k - R_k D_k > 3, decrease W_k.

The justification for this algorithm is that W_k - R_k D_k is an estimate of x_k, the backlog of connection k inside the router. Accordingly, if the algorithm converges and results in x_k being close to 2, then all the connections will have an equal share of the router and have R_k close to C/N. Moreover, the total backlog will be rather small and **the algorithm achieves a fair** and efficient use of the resources. [Back to Top]

Loss/Window Mechanisms (Tahoe, Reno)

We go back to Figure 1 and we consider mechanisms that adjust the window based on observed losses of packets. The source notices that a packet has been lost when the timer for the ACK of that packets runs out (time-out) or when duplicated ACKs arrive at the source (assuming that the ACK number is that of the next byte or next packet expected, as in TCP).

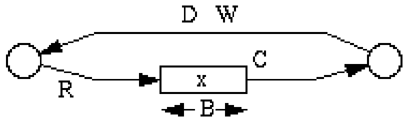


Figure 3: 1 connection through a bottleneck router

When the source detects a loss, it thinks that the loss is due to congestion and decreases the window W of its retransmission protocol. In the absence of losses, the source increases its window size to "fill up the pipe." Let's try a naive protocol first.

> **Algorithm 2** (Naive). Source adjusts W according to following rules:
> While no loss, increase W;
> When loss, decrease W.

So far we are not precise about how we adjust and decrease W. Let's say that W increases at rate u and decreases to 1. (We explain below why this is not a good idea.) For W small, R = W/D and x = 0. When W reaches CD, x starts growing and R saturates to C. Thus, as before, x = max{W - RD} and R = C when x > 0. A packet is lost when x reaches B, i.e., when W reaches CD + B. With our adjustment mechanism, we expect W to grow from 1 to CD + B + uD when we detect the loss (with a delay of D) and then the window drops to 1. Figure 3 shows the evolution of the window size with Algorithm 2. This is not very good: the average window size is about half of the desired value (assuming u not too large), we lose many packets that have to be retransmitted.

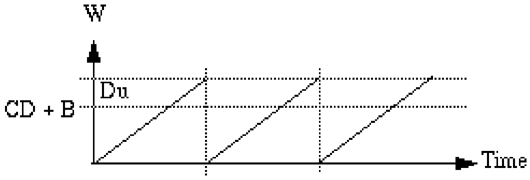


Figure 3: Window size for Algorithm 2

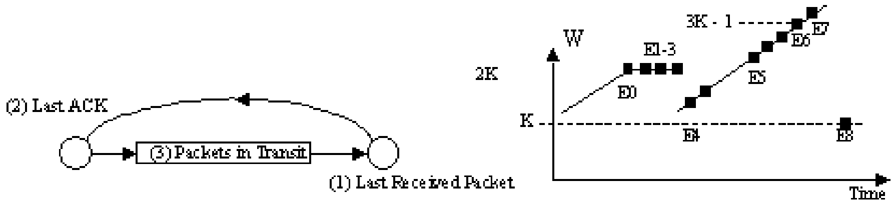
How do we **increase the window size linearly**? One way to do this is as follows. Assume that the window size is W. The source increases the window size by 1/W every time it gets an ACK. (The window size is measured in bytes and one unit here is some number of bytes, say 64k. Accordingly, 1/W makes sense: For instance, if W = 16, 1/W is 4 kbytes.) During one round-trip time D, the source gets back about W ACKs and increases the window W times by about 1/W each time, so that it increases the window size by one unit every D seconds. Thus, the window size increases linearly with rate u = 1/D. Note that the rate depends on D; this fact will come back to haunt us later.

Three simple ideas can be used to improve this algorithm. We look at these ideas one by one.

First improvement (Slow Start): Increase window size fast initially and increase more slowly when we approach the value that was being used when the loss occurred. Thus, if the loss occurs when the window size is W, we set V = W/2, reset W = 1, increase W fast until W reaches V, then increase W linearly as before (this last precaution is called **congestion avoidance**). How do we **increase the window size fast**? Here is a simple method: The source increases the window size W by 1 every time it gets and ACK. As a result, during one round-trip time D the source gets W ACKs and increases the window size by W. Accordingly, W doubles every D seconds: W grows exponentially: W(t) = 2^{t/D}, roughly. Algorithm 2 with this first improvement is called Slow Start with Congestion Avoidance.

Second Improvement (Fast Retransmit): Do not decrease the window size from W to 1 when you detect a duplicated ACK; instead replace W by W/2. We discuss below the motivation for this multiplicative decrease of W (by 50%). The motivation for not going down to 1 is clear by looking at Figure 3.

Third Improvement (Fast Recovery): The objective of this modification is to control better the backlog inside the pipe than by simply resetting the window size. The steps in Figure 4 explain the mechanism.



(2)	(3) Packets in Transit	(1)	W	Event
L - 1	L + 2K - 1, ..., L + 3, L + 2, ~~L + 1~~, L	L - 1	2K	[E0] L was lost
L - 1	L + 2K - 1, ..., L + 3, L + 2	L + 1	2K	[E1] 1st dup ACK: Don't change W
L - 1	L + 2K - 1, ..., L + 3	L + 2	2K	[E2] 2nd dup ACK: Don't change W
L - 1	L, L + 2K - 1, ..., L + 4	L + 3	K + 3	[E3] 3rd dup ACK: Retransmit L, reset W
L - 1	L, L + 2K - 1, ..., L + 5	L + 4	K + 4	[E4] W + 1 for each ACK (even dup)
.	...	...	...	Continue
L - 1	L + 2K, L, L + 2K - 1, ..., L + K	L + K - 1	2K + 1	[E5] W large enough => send new packet
L - 1	L + 2K + 1, L, L + 2K - 1, .., L + K + 1	L + K	2K + 2	Continue increasing W, sending new packets
.	...	...	...	
L - 1	L + 3K - 2, ..., L	L + 2K - 1	3K - 1	[E6] Continue
L + 2K - 1	L + 3K - 1, ..., L + 2K	L	3K	[E7] End of Fast Recovery
L + 2K	L + 3K, ..., L + 2K + 1	L + 2K	K	[E8] Reset W = K and resume Congestion Avoidance

Figure 4: Fast Recovery. The objective is to go from 2K to K packets in the pipe after detecting 3 duplicate ACKS

In the history (now, folklore) of TCP fine-tuning, the original implementation, **Tahoe**, incorporated slow start and congestion avoidance. The next version, **Reno**, included fast retransmit and fast recovery.
[Back to Top]

Additive Increase - Multiplicative Decrease

What happens when a number of connections adjust their transmission rate? How can we hope that the adjustments converge to a fair allocation of the transmission rate? We discuss a simple argument that shows that one particular mechanism, called additive increase and multiplicative decrease, has a chance to converge to such a fair allocation. Figure 5 summarizes the evolution of the transmission rates of two connections that share one bottleneck.

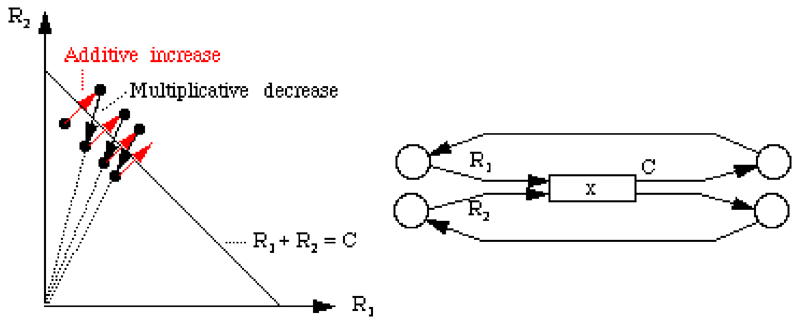


Figure 5: Additive increase and multiplicative decrease converges to a fair allocation

The figure shows that the two sources increase their rate by the same amount when the router is not congested, i.e., when the sum of the rates is less than C. Such an increase corresponds to the red arrows. When the sum of the rates exceeds C, the sources find out about the congestion and reduce their rates by a common factor. Such a decrease corresponds to a black arrow that points toward the origin (0, 0). Iterating this process, we see that the two rates converge to C/2.

To try to make all the sources aware of the congestion, instead of only the source that happened to lose a packet, some routers implement a random early discard of packets when they become congested. Such routers are called RED (cute if inefficient!).

Of course, when implementing such an algorithm, the sources do not update their rates by the same amount. When there is no congestion, as we discussed in "How to increase the window linearly," we learned that the rate of increase depends on the round-trip delay. If we take this fact into account, we find that additive increase and multiplicative decrease does not converge to a fair allocation. [Back to Top]

Incompatibility of Reno and Vegas

Now for a bit of controversy. We have seen that Vegas converges (at least in the simplistic case of a single bottleneck) to a fair allocation. However, most implementations of TCP use Reno. What happens if a source implements Vegas and others implement Reno? Consider Figure 6.

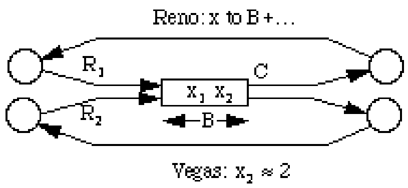


Figure 6: Mixing Reno and Vegas

As we explained, Vegas tries to maintain x_2 close to 2. On the other hand, Reno pushes x all the way to overflow, then backs off more or less cleverly. In any case, Reno ends up with x_1 substantially larger than x_2. Since the rate of a connection is proportional to the fraction of its packets in the router, we see that Vegas loses in this competition.

The situation is much worse if a connection deliberately does not back off when the router is congested.

For instance, you can implement a connection over UDP and ignore all delays in ACKs. All the Reno and Vegas connection would back off to make room for you. As we know democracy has its danger. [Back to Top]

Rate-Based Control: ABR in ATM

So far we have looked at window-based congestion control. We now examine one example of rate-based control. Such a form of control is implemented in a service called "Available Bit Rate" in asynchronous transfer mode networks. A number of telephone operators (e.g., Pacific Bell) use ABR/ATM to provide a fast data service to Internet service providers and other corporate clients.

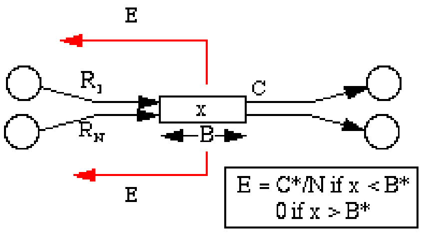


Figure 7: Explicit rate control in ABR

Figure 7 illustrates a very simple example of one for of ABR. The switch selects a target value B* for its buffer occupancy and some rate C* > C. In addition, the switch monitors the number N of ABR connections that are active. (Many connections may be set up but dormant.) The sources have to limit their transmission rate (with a leaky bucket, see below) to the rate E (for explicit) advertised by the switch. If all the sources follow this algorithm and are eager to transmit when they are active, then the total rate into the buffer is C* when x < B* and is 0 when x > B*. Accordingly, the rate of increase of x is C* - C > 0 when x < B* and is - C when x > B*. Consequently, x should converge to B*. If we take into account the propagation delay, we see that the sources might, in the worst case, transmit at rate C* for some time T even after x has crossed the value B*. When this happens, the largest possible value of x is B* + (C* - C)T. Similarly, the lowest value of x if all the sources are always eager is B* - CT. If we have a bound on T we can calculate the values of C* and B* that lead to efficient use of the switch (x > 0 always) and no overflow (x < B always).

When there are multiple switches, they each advertise an explicit rate and the sources have to respect the minimum value they received from the switches. [Back to Top]

A Few Concluding Thoughts

We have not discussed **reservations**. Reservations are necessary if some connections require a minimum rate to operate satisfactorily. Reservation requires signaling and that the routers or network operations and management system keep track of the committed resources.

We have not discussed **priorities** or other scheduling strategies that either favor some connections over others or that attempt to guarantee some level of service to some connections. Countless variations on scheduling policies have been proposed since the early 1960s and their impact on the quality of service of the connections has been studied since. Some of these ideas have been rediscovered recently by the

Internet researchers and simple ones are being implemented. [Back to Top]

Index

ABR: Available Bit Rate - A service in asynchronous transfer mode that attempts to use its spare resources fairly and efficiently.

ATM: Asynchronous transfer mode - A technology that transports information in 53-byte cells along virtual circuits. The objective of this technology is to control the quality of service it provides to connections. ATM is used in Internet backbones and by telephone operators to provide data services.

Additive increase, multiplicative decrease: A strategy for adjusting transmission rates that, under idealistic assumptions, converges to a fair sharing of the bottleneck link.

Congestion Avoidance: Slow down increase of window when get to 50% of value that caused loss.

Delay/Window control: A strategy for adjusting transmission rates used in Vegas.

Fast Retransmit: Drop window by 50% insterad of all the way down.

Fast Recovery: Reduce the pipe backlog by 50%.

Increasing the window linearly: Motivated by additive increase and used in congestion avoidance.

Losses/Window control: A strategy for adjusting transmission rates used in Tahoe and Reno.

Priorities: Scheme for offering different qualities of service to connections.

Rate-based control: Used in ABR.

RED: Random early discard. A strategy for informing sources "equally" that they congest a router.

Reservations: Necessary if resources require some quality of service.

Reno: An implementation of congestion control in TCP. See text.

Slow Start: Increasing the window exponentially.

Tahoe: An early implementation of congestion control in TCP. See text for details.

Vegas: A proposed congestion control algorithm for TCP. See text.

[Back to Top]