Chapter 2

Existing Congestion Control Schemes

There are many existing congestion control schemes for connectionless packet-switched networks. In this chapter, I will review the topic of congestion control in more detail, and examine some traditional control schemes. In the next chapter, I will focus on congestion control schemes which are rate-based, and which are more closely related to the work described in this thesis.

2.1 Jain's Congestion Survey

Jain's review paper [Jain 90] surveys the problems and possible solutions of network congestion control. Jain cites several myths about congestion:

- Congestion is caused by a shortage of buffer space, and can be solved by increasing the amount of buffers.
- Congestion is caused by slow links.
- Congestion is caused by slow processors.
- If not one, then solutions to all of the above will cause the congestion problem to go away.

As shown in [Nagle 87], congestion cannot be solved with a large buffer: this only postpones the inevitable packet loss when network load is larger than network capacity. Slow links by themselves do not cause congestion, but the mismatch of network link speeds helps cause congestion. Similarly, slow processors do not cause congestion: the introduction of a high-speed processor in an existing network may actually increase the mismatch of processing speeds and the chances of congestion. In fact, congestion occurs even if all links and processors are of the same speed.

Jain suggests that there are two main classes of congestion control schemes: *resource creation* schemes, which dynamically increase the capacity of the resource, and *demand reduction* schemes, which reduce the demand on the resource. For the networks under consideration, the bandwidth on each link is generally static; therefore, we must discount resource creation schemes. Demand reduction schemes can be subdivided into three subclasses:

- *Service denial* schemes do not allow further resource allocation. Connection-oriented networks, for example, can prevent new connections from being formed.
- *Service degradation* schemes ask all users (existing as well as new) to reduce their loads.
- *Scheduling* schemes ask users to schedule their demands so that the total demand is less than the capacity. This subclass is a special instance of the service degradation subclass.

In connectionless networks, new network transmissions cannot be prevented, so a service denial approach cannot be used. For these networks, we must consider scheduling and service degradation schemes.

Several feedback techniques have been proposed to implement these approaches [Jain 90], such as:

- *Observing congestion evidence* such as packet loss or increasing packet delays, and reducing the load on the network where possible [Jain 86] [Jacobson 88]. This is done primarily at the Transport Layer and not within the network itself. One drawback with this approach is that packet loss and increasing delays do not always indicate congestion.
- *Feedback messages* which are sent from the congested resource to the source point. Examples are "choke packets" [Majithia & *et al* 79], "quench messages" [Postel 81a] and "permits" [Davis 72]. A drawback with these approaches is that the extra traffic created can exacerbate the congestion.
- *Feedback in routing messages* allows an intermediate resource to set its load level to that of all neighbouring nodes.
- *Rejecting further traffic* causes incoming packets to be lost or not acknowledged. The resulting *backpressure* causes queues to form at other nodes, and the backpressure slowly travels towards the source. This technique is only useful for short duration congestion, and can affect other resources not involved in the congestion.
- *Probe packets* are sent through the network, and source nodes adjust their loads, according to the delays experienced by the probe packets.
- *Feedback fields in packets* can pass congestion information to end hosts [Schwartz 82], [Robinson *et al* 90]. This avoids the overhead of feedback messages, and can be used in both forward and reverse directions.

Also, several alternatives has been proposed for the location of congestion control, such as in the Transport Layer, Network Layer, Data Link Layer, and in specific network entrances, such as routers.

There are several requirements for a good congestion control scheme. The scheme must have a low overhead. The scheme must be fair: defining fairness is not trivial, and no one definition has been widely accepted. The scheme must be responsive to available capacity and demand. The scheme must work in the reduced environment of congestion, including effects such as transmission errors, out-of-sequence packets and lost packets. Finally, the scheme must be socially optimal: it must allow the total network performance to be maximised. Designing a congestion scheme that meets all of these requirements is not trivial.

In conclusion, Jain restates that congestion is not a static resource shortage problem, but a dynamic allocation problem. Congestion control schemes can be classified as resource creation schemes or demand reduction schemes. Congestion control is not trivial because of the number of requirements such as low overhead, fairness etc. A number of network policies affect the choice of congestion control schemes, and one congestion control scheme may not be suitable for all networks. Finally, as networks become larger and heterogeneous, with higher speeds and integrated traffic, congestion will become more and more difficult to handle, but also more and more important to control.

2.2 Traditional Control Schemes

In most traditional connectionless packet-switched networks, the functionality of the Network Layer and Transport Layer are clearly separated [Clark 88]. The Network Layer is responsible for the forwarding of packets to their destination host, across one or more links which may have widely different transmission characteristics: total bit rate, overall bit error rate, medium access method, multiplexing scheme etc. In the networks under consideration, this delivery of packets is “best effort”, with no guarantee of packet delivery: intermediate routers may discard packets for any reason. The Network Layer perceives *flows* of traffic from source hosts to destination hosts.

The Transport Layer is responsible for such things as the setup of virtual circuits (also known as *connections*) between a source application and a destination application, the reliable and in-sequence delivery of data to the destination application, and flow & congestion control. Not all of these value-added services are required by every application, and some applications may desire other services such as constant-bit-rate connections.

In these connectionless packet-switched networks, congestion control schemes are generally implemented in the Transport Layer of the source and destination hosts. The Transport Layer must counter packet loss, which is often caused by network congestion. As well, an existing Transport Layer flow control scheme can often be adapted to perform some congestion control.

The Transport Layer, as the basic generator of a traffic flow, is the essential “source” of the traffic flow. Unfortunately, while it is responsible for the flow of traffic that may cause network congestion, the Transport Layer is divorced from the Network Layer. Therefore, the Transport Layer is unable to see the effect of its traffic flow on the intermediate nodes between it and the destination of its traffic.

The Transport Layer is only able to determine that there is network congestion in two ways:

- By being informed of existing network congestion from the Network Layer (as is done in the Source Quench mechanism described in Section 2.4.1).
- By observing the effects of congestion on the traffic flow: delayed arrival of acknowledgments, and acknowledgments that indicate the loss of data (as is described in [Jacobson 88]).

Delayed acknowledgments and loss of data do **not** specifically imply network congestion. They may be caused by packet corruption and temporary traffic instability due to new flows of data being created. Therefore, transport protocols that rely on the second method to deduce the existence of congestion may not detect congestion when it exists, and may believe that congestion exists when it doesn’t.

Transport protocols which perform end-to-end reliability must retransmit data when it is lost in transit. In general, retransmissions are prompted either by the arrival of a negative acknowledgment, or the expiry of a timer set to detect the non-arrival of any acknowledgment. Neither of these events specifically imply the loss of data. Transport protocols which use these events to prompt data retransmission may retransmit data unnecessarily, burdening the network with redundant transmissions and increasing the chance of congestion.

Many existing transport protocols suffer from the problems described above. The transport protocol which typifies the Transport Layer in these traditional networks, and which has seen the most research effort in its congestion control, is TCP. Let us examine this work.

2.3 Transmission Control Protocol

The Transmission Control Protocol, or TCP, provides reliable end-to-end data transmission across a possibly unreliable network such as the Internet. TCP was influenced by the NCP protocol [Reynolds & Postel 87] and by [Cerf & Kahn 74]. Originally standardised in [Postel 81b], it has since been improved upon by several researchers.

2.3.1 Sliding Window Flow Control

TCP uses a dynamic Go-Back-N sliding window for end-to-end flow control. The size of the window is negotiated during the connection setup, and can vary during the course of data transmission. The receiver returns an 'Advertised' window size in every acknowledgment which indicates the amount of buffer space the receiver has left for this traffic flow. The sender calculates an 'Effective Window' size, which is how much more traffic it can transmit to the receiver. The effective window size is equal to the receiver's advertised window size minus the amount of data currently outstanding. If the effective window size is less than one byte, then the sender cannot transmit any further data.

The sliding window in TCP also affects its retransmission strategy. If an acknowledgment from the destination shows that some data beginning at point N is lost, the source may need to retransmit **all** outstanding data from N onwards, even if the destination has already received much of this data. This causes unnecessary network traffic and increases the network load.

Since the original specification of TCP was published, its flow control has been found to be fraught with several problems and deficiencies. There have been several proposals that remedy these problems, such as the "Silly Window Syndrome" [Clark 82] and "Small Packet Avoidance" [Nagle 84]. Other remedies address the use of the sliding window for congestion control, and are described below.

Nearly all implementations of TCP have these sliding window extensions. TCP's sliding window operation is now well-behaved but very inelegant.

2.3.2 Sliding Window Congestion Control

The initial TCP specification allowed hosts to transmit as much data as the Effective Window size, to fill the destination's buffers. This, however, could cause significant congestion in intermediate routers, and indeed was one of the main causes for the high Internet congestion in the mid 1980s [Nagle 84] [Jacobson 88].

Several modifications to the sliding window scheme were proposed in [Jacobson 88] which dramatically reduced the problem of network congestion. These were "Multiplicative Decrease", "Slow Start" and "Fast Retransmit".

If the assumption that packet loss generally indicates network congestion is true, then this also indicates that the source has too much in-transit data. The Multiplicative Decrease proposal suggests:

1. Add a variable *cwnd* (the congestion window) to the sender's state.
2. When packet loss is detected, set *cwnd* to half the current window size (the multiplicative decrease).
3. On each acknowledgment for successful data reception, increase *cwnd* by $1/cwnd$.
4. At any time, transmit packets so that the amount of in-transit data is the minimum of the Effective Window and *cwnd*.

With this scheme, TCP slowly increases its congestion window size until data is lost, then the congestion window is halved. This helps to drain excess packets from the network. This pattern of increase and decrease continues until an equilibrium is reached. Changes within the network itself may disturb the equilibrium, but the Multiplicative Decrease scheme helps to return data transmission to a near-equilibrium point.

The Multiplicative Decrease scheme does not address how equilibrium is initially reached, only what to do when this has occurred. Reaching equilibrium is covered by Slow Start:

1. When starting data transmission, or when the Effective Window size in the source is zero, set *cwnd* to one packet.
2. On each acknowledgment for successful data reception, increase *cwnd* by one packet.
3. At any time, transmit packets so that the amount of in-transit data is the minimum of the Effective Window and *cwnd*.
4. Revert to Multiplicative Decrease on packet loss.

Slow Start causes an exponential increase in the congestion window until data is lost. Initially, *cwnd* is one packet. When the ack for this packet arrives, *cwnd* becomes two and two new packets are transmitted. When their acks arrive, *cwnd* is incremented twice to become four, and so on. Eventually, the congestion window becomes so large that packet loss occurs, and Multiplicative Decrease takes over.

Slow Start is also used when the source cannot transmit because the effective window size is zero. If this window of outstanding data is acknowledged all at once by the destination, then the source could retransmit a whole advertised window of data, which may cause short-term network congestion. Instead, Slow Start is used to ramp the source's window of outstanding data up to this advertised size.

These two schemes have been very effective in reducing the network congestion caused by TCP's sliding window. However, they depend upon the accurate detection of packet loss in order to operate correctly. TCP's use of timers to detect packet loss is covered in the next section. Another scheme to detect packet loss, "Fast Retransmit" was also proposed in [Jacobson 88].

Even though TCP sets a timer to detect if a packet is lost (as no acknowledgment arrives), acknowledgments for other data may indicate the loss of a packet. This is because the Go-Back-N acknowledgment scheme used by TCP only acknowledges the highest point of *contiguous* data received to date.

With Fast Retransmit, data is retransmitted once three or more acknowledgments with the same sequence number are received by the source, even if the packet loss timer has not yet expired. Two acknowledgments with the same sequence number may have been a consequence of data reordering within the network. Fast Retransmit helps TCP to react to packet loss faster than with just the packet loss timer.

2.3.3 Retransmission Timer Deficiencies

TCP normally uses the expiry of a timer to detect the non-arrival of an acknowledgment: this in turn prompts data retransmission. The dilemma here is that if the timer is set too small, it expires before an acknowledgment can arrive, and data is retransmitted unnecessarily. If it is set too large, the protocol reacts too slowly to the loss of data, reducing the rate of data transmission and throughput.

The acknowledgment for transmitted data should return at the round-trip time. However, the round-trip time is anything but constant, and it varies due to the load on the network and short- and long-term congestion within the network. Thus, estimating the round-trip time is very difficult. The original specification for TCP [Postel 81b] suggested a simple algorithm for estimating the round-trip time.

Both Zhang [Zhang 86] and Jacobson [Jacobson 88] demonstrate that the algorithm does not cope with the large round trip time variance often experienced when the Internet is heavily loaded. Jacobson describes a method for estimating the variance in the round-trip time. This method causes fewer unneeded TCP timer timeouts, reducing unneeded packet retransmissions and reducing the load and the congestion in the network. All major implementations of TCP now have Jacobson's round-trip estimation algorithm in their TCP code.

Karn [Karn & Partridge 87] added to Jacobson's work by showing an ambiguity in round-trip time estimation due to acknowledgments for packets that have been retransmitted. Karn presents an algorithm for removing the ambiguity from round-trip time estimates. Again, all major implementations of TCP now have Karn's algorithm in their TCP code.

These solutions show that round-trip time estimation is difficult to do in conditions when it has large variance. Incorrect round-trip time estimation leads to network congestion and/or poor throughput and can further add to round-trip time variance, compounding the problem.

2.3.4 TCP Vegas

Despite the improvements described above, TCP only implements congestion *recovery*, as it continually increases its congestion window until packets are lost. This probing of the network increases buffer occupancy in routers, shifting network operation past the knee-point of peak power (Figure 2, pg. 4).

Most of the improvements in [Jacobson 88] were implemented in the 4.3BSD-Tahoe operating system, and this version of TCP is known as TCP Tahoe. The 4.3BSD-Reno operating system added Fast Recovery, TCP header prediction and delayed acks to TCP, and this version of TCP is therefore known as TCP Reno.

Since the implementation of TCP Reno, several researchers have highlighted shortcomings in its implementation and in its congestion recovery scheme [Wang & Crowcroft 81] [Wang & Crowcroft 91] [Brakmo & Peterson 95]. Recently, further proposed modifications to TCP, known as TCP Vegas [Brakmo *et al* 94], suggest several enhancements to TCP Reno. Their research also describes a congestion *avoidance* scheme which allows TCP to find an equilibrium in transmission, without inducing network congestion to do so.

In TCP Reno, the Fast Retransmit scheme depends on at least three duplicate acks, which is unlikely in high congestion situations where there is large packet loss. TCP Vegas records the system time for each packet transmitted; therefore, it can determine the round-trip time accurately for each acknowledgment. If a duplicate acknowledgment has a round-trip time greater than the time set for the data's packet loss timer, then the packet is retransmitted. Three or more duplicate acknowledgments are not required. TCP Vegas also retransmits data before the expiry of the packet loss timer for other reasons.

In this modified Fast Retransmit scheme, the congestion window is never reduced when data is retransmitted because it was lost *before* the congestion window was previously reduced. This was possible in TCP Reno and would cause lower throughput.

The congestion avoidance scheme in TCP Vegas, which replaces Multiplicative Decrease, depends on the idea that the congestion window is directly proportional to the throughput expected by TCP; the congestion window should only be increased if the throughput expected by TCP is increasing. TCP Vegas approximates a measurement of the expected throughout for the connection, and uses this to update the congestion window, as described in [Brakmo *et al* 94]

Results given in [Brakmo *et al* 94] and [Ahn *et al* 95], from simulations and Internet experiments, indicate that TCP Vegas gives greater throughput for less packet loss than TCP Reno. TCP Vegas was also found to keep buffer occupancy in routers at a lower level than TCP Reno. These results are confirmed by the experimental results described in this thesis.

Although TCP Vegas appears to provide very good congestion control, several researchers[1] believe that TCP Vegas is too aggressive in its use of bandwidth, penalising TCP Reno sources and not allowing new TCP connections to obtain a fair share of the available bandwidth. Other papers suggest deficiencies in TCP Vegas [Ait-Hellal & Altman 96]. The verdict on TCP Vegas is still out.

2.3.5 Summary

TCP as first specified was a complicated transport protocol, and the improvements to it since then has made it even more complicated. The most widely-used implementation, TCP Reno, only has congestion recovery. Even worse, TCP must cause network congestion to determine its congestion window size, and its reliance on round-trip estimation ensures that both error recovery and congestion recovery are not optimal. Suggested congestion avoidance schemes such as TCP Vegas have not yet been fully accepted by the networking community.

Chapters 9 and 10 highlight the congestion deficiencies in TCP Reno, and also show the benefits in congestion control that TCP Vegas affords.

[1]Sally Floyd, Van Jacobson and others, End-to-end mailing list *end2end-tf@isi.edu*, 1994.

2.4 Network-based Congestion Control

Congestion control schemes where network components such as routers participate should be more effective than purely end-to-end control schemes. As most network congestion occurs in routers, they are in an ideal position to monitor network load and congestion, and use this information in a congestion control scheme.

Network-based congestion control schemes can feedback congestion data to transmitting sources, thus altering their behaviour directly. Alternatively, network-based schemes can change the packet processing within routers: this can help alleviate congestion, and may also indirectly affect sources' behaviour.

We will examine several schemes that feedback congestion data to sources, and several schemes that alter packet processing behaviour within routers.

2.4.1 ICMP Source Quench

The ICMP Source Quench is the only congestion control mechanism in the Network Layer of the TCP/IP protocol suite. Both routers and hosts play a part in the mechanism to control congestion. When a router believes itself to be congested, it sends 'Source Quench' packets back to the source of the packets causing the congestion. On receipt of these packets, the host should throttle its data rate so as to prevent router congestion. It can then slowly increase its data rate, until source quench packets begin to arrive again.

The mechanism is simple, and suffers from many deficiencies in both the effectiveness of the mechanism, and the specification of its implementation. Let us tackle the latter first.

The Source Quench mechanism is defined in [Postel 81a]. In effect, the specification of the Source Quench mechanism makes **no** proscriptions with regards its implementation in any Internet protocol stack. The entire Source Quench mechanism is thus left to each implementation to decide on its particular behaviour. This may lead to behaviour that is different to the spirit of the specification, and possibly to badly-behaved congestion control between two implementations of the same mechanism.

Source Quench's effectiveness as a congestion control mechanism is also flawed. This is chiefly due to two problems. Firstly, [Postel 81a] does not clearly state how a router or destination decides when it is congested, who to send a source quench message to, how often to send source quench messages, and when to stop. Secondly, [Postel 81a] does not clearly state how a host should react to a source quench message, by how much it should reduce its output rate, if it should inform upper layers in the network stack, and how to properly increase its output rate in the absence of source quench messages.

Even worse, a router or destination may be congested and not send any source quench messages. Sources in receipt of a source quench message cannot determine if a router or destination dropped the datagram that caused the source quench message. The source also does not know to what rate it should reduce its transmission to prevent further source quench messages.

Since the original specification of the Source Quench mechanism, several further papers have described alternatives to improve the performance of the mechanism, and to tighten the mechanism's specification ([Nagle 84] and [Braden & Postel 87]). One serious problem, raised by Nagle in [Nagle 84], is that sources who receive source quench messages may not inform the corresponding Transport Layer of the source quench. Most implementations only inform TCP, leaving other protocols such as UDP [Postel 80] unaware that they are causing congestion. This is a serious problem, as the indication of a congestion problem is not communicated to the layer that is generating the packets that are causing the problem.

Some of the deficiencies in Source Quench were recognised in [Braden & Postel 87], and while it mandated certain parts of the Source Quench mechanism, it still left others non-mandatory, with suggested behaviour given. The document notes that "Although the Source Quench mechanism is known to be an imperfect means for Internet congestion control, and research towards more effective means is in progress, Source Quench is considered to be too valuable to omit from production routers."

Source Quench was reviewed in [Mankin & Ramakrishnan 91]. The review is small, and is given here in summary, with editorial emendations.

> The method of router congestion control currently used in the Internet is the Source Quench message. When a router responds to congestion by dropping datagrams, it may send an ICMP Source Quench message to the source of the dropped datagram. This is a congestion recovery policy.
>
> [...]
>
> [A] significant drawback of the Source Quench policy is that its details are discretionary, or, alternatively, that the policy is really a family of varied policies. Major Internet router manufacturers have implemented a variety of source quench frequencies. It is impossible for the end-system user on receiving a Source Quench to be certain of the circumstances in which it was issued. [...]
>
> To the extent that routers drop the last arrived datagram on overload, Source Quench messages may be distributed unfairly. This is because the position at the end of the queue may be unfairly often occupied by the packets of low demand, intermittent users, since these do not send regular bursts of packets that can preempt most of the queue space.

In summary, it can be seen that Source Quench is a rudimentary congestion control method. It is loosely specified, which can lead to badly-behaved operation across different implementations. It may not limit a source's transmission rate, depending on the transport protocol, and it may drop packets unfairly.

Despite this, Source Quench is in wide operation throughout the Internet, and provides the sole network-level congestion control for it.

2.4.2 The Dec Bit Mechanism

The Dec Bit mechanism, also known as the Congestion Indication mechanism, is a *binary feedback* congestion avoidance mechanism developed for the Digital Network Architecture at DEC [Jain *et al* 87], and has since been specified as the congestion avoidance mechanism for the ISO TP4 and CLNP transport and network protocols.

In Dec Bit, all network packets have a single bit, the 'Congestion Experienced Bit', in their headers. Sources set this bit to zero. If the packet passes through a router that believes itself to be congested, it sets the bit to a one. Acknowledgment packets from the destination return the received Congestion Experienced Bit to the source. If the bit is set, the source knows there was some congestion along the path to the destination, and takes remedial action. In DECNET, the source adjusts its window size. In TP4, the destination alters the advertised window size, rather than returning the bit to the source.

Dec Bit can be used as either a congestion avoidance or a congestion recovery mechanism, but is used as an avoidance mechanism in [Jain *et al* 87]. To be used as an avoidance mechanism, routers must determine if their load is at, below or above their peak power point (see Figure 2, pg. 4). This is achieved by averaging the size of the router's output queues. If they have more than one buffered packet on average, that output interface's load is above the peak power point; if less than one packet, the interface's load is below the peak power point.

As noted by Jain et al. [Jain *et al* 87], determining a good averaging function is not easy. Using instantaneous queue sizes is an inaccurate indication of the load on an interface. On the other hand, a too-large averaging interval gives a distorted indication of the real load. They settled for an average taken from the second-last 'regeneration point' to the current time, where a 'regeneration point' is a point when the queue size is zero, as shown in the following figure from [Jain *et al* 87]:

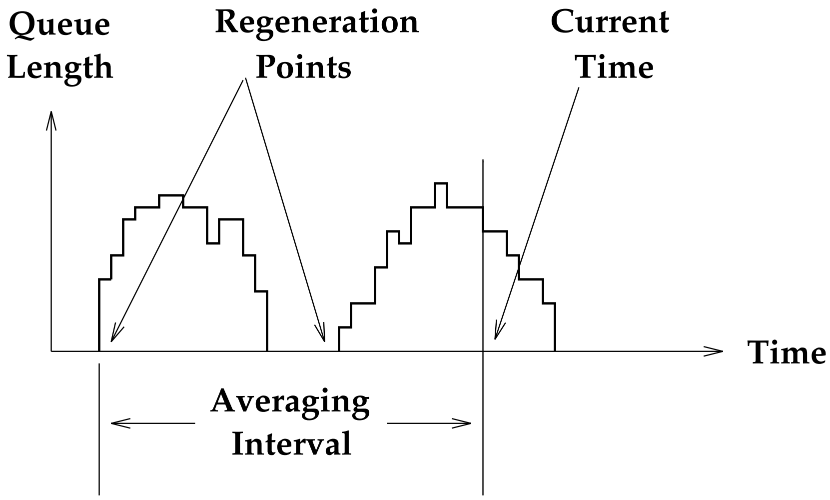

Figure 3: Interval for Queue Length Averaging [Jain *et al* 87]

As well as detecting congestion, a router must determine which set of sources should be asked to adjust their load, by setting the Congestion Experienced Bit in their packets. Sources taking more than their fair share of the resource have their bit set; other sources do not have their bit set.

A source reacts to the Congestion Experienced Bit by either increasing its window size by a constant (if the bit is not set), or decreasing the window size by a multiplicative factor. Once the window size changes, new Congestion Experienced Bits are ignored for one round-trip's time, to allow the traffic's effect on the network to be monitored and returned to the source via another Congestion Experienced Bit.

Jain et. al show that Dec Bit is robust and brings the network to a stable operating point with minimal oscillations in sources' window size. Since this paper, DecBit has been found to be a good congestion avoidance scheme. However, the convergence time can be several round-trip times, which does not help to alleviate short-term network congestion.

2.4.3 The Q-Bit Mechanism

An extension to DecBit, Q-Bit [Rose 92], suggests feeding a further bit of congestion information back to the transmitting source. This Q-bit is set when a packet received by a router could not be retransmitted immediately, but had to be queued for transmission. Rose states that this gives additional information to the source:

- when an overload-condition is approaching, a queue is building up and thus the Q-bit is set. This happens some time before C-bits [Congestion Experienced bits] will be set. Thus in this situation the sender should refrain from increasing its rate, keeping it constant instead.
- after the queue has vanished, C-bits may still last for a certain while. The Q-bit on the other hand will be turned off as soon as the queue disappears, and the sender should refrain from decreasing the load in that situation, keeping it constant again.

Sources react to either set C-bits or set Q-bits by lowering their rate. If the C-bit is set but the Q-bit is off, the source keeps its rate constant. The scheme also introduces a 'Freeze Time': a period after a source's window is reduced where no further windows reductions will occur. This helps to make the Q-bit scheme more stable when there are several concurrently transmitting sources.

In comparison with DecBit, Q-bit gives equal if not better results, and can guarantee fair division of bandwidth for traffic flows with different path lengths. The scheme is also useful for rate-based transport protocols as well as window-based protocols. As with DecBit, convergence to equilibrium still takes several round-trips to achieve.

2.4.4 Summary

Network-based schemes which feedback congestion information to sources give them direct information about the congestion state of the network. This should allow the sources to alter their transmission behaviour so as to reduce the congestion. The schemes described above have two main deficiencies. Firstly, only the existence (or the lack) of congestion is reported to the source: it cannot determine the extent or the severity of the congestion. Secondly, the sources use Sliding Window flow control, and it may take several round-trips for a source to reduce its window to a point which minimises congestion.

2.5 Packet Discarding Mechanisms

The main aim of congestion control is to reduce the rate of source transmissions to that which can be sustained by the network, thus preventing loss of data due to congestion. However, in the short term, routers must drop packets when buffers overflow. This is a form of congestion recovery.

Because Transport protocols like TCP increase their sliding window until packets are lost, there is usually at least one router along the path of a traffic flow which is congested, and has several packets queued for transmission. In this situation, a router may decide to drop packets *before* buffers overflow: this is a form of congestion avoidance. The choice of which packet to drop can also reduce any unfairness in network usage (such as was asserted against TCP Vegas in Section 2.3.4).

A router typically chooses the most recently received packet to discard when buffers are full, and this is known as Drop Tail, as packets are normally queued on the tail of the router's buffer. This is not the only choice available to a router.

Nagle [Nagle 84] suggests a strategy of discarding the latest packet from the source that has the most packets buffered in the router. This would appear to penalise traffic flows overusing available bandwidth. In fact, a traffic flow may be underutilising the bandwidth between its source and destination, and still have the most packets buffered. Therefore, this scheme may penalise the 'wrong' traffic flows.

Another strategy would be to discard an incoming packet if it duplicates a packet which is already buffered in the router. This would provide some protection against poor Transport Layer implementations, or protocols where round-trip time estimation is poor.

2.5.1 Random Drop

Random Drop [Hashem 90] is a mechanism by which a router is able to randomly drop a certain fraction of its input traffic when a certain condition is true. The premise of Random Drop is that the probability of a randomly chosen packet belonging to a particular connection is proportional to the connection's rate of traffic. The main appeal of Random Drop is that it is stateless. A router using Random Drop does not have to perform any further parsing of packet contents to implement the mechanism.

Random Drop can be used as either a congestion recovery or a congestion avoidance mechanism. In the former, a router randomly drops packets when it becomes overloaded. In the latter, packets are randomly dropped to keep the router at its optimum power position.

Random Drop depends heavily on packet sources interpreting packet loss as an indicator of network congestion. Protocols such as TCP do make this interpretation. However, other protocols such as UDP do not make this interpretation. Therefore, Random Drop may not be useful in a heterogeneous source environment.

Another problem is that Random Drop does not distinguish between "well-behaved" and "ill-behaved" sources. Packets are dropped for sources that are congesting a router, and for sources that are not congesting a router. Similarly, for low-rate connections such as keyboard 'Telnet' connections, the loss of one packet may be a significant loss of the connection's overall traffic.

When Random Drop is used for congestion recovery (RDCR), instead of dropping the packet that causes the input buffer to overflow, a randomly chosen packet from the buffer is dropped. Mankin [Mankin 90] analyses Random Drop Congestion Recovery theoretically and as implemented in a BSD 4.3 kernel. She notes that RDCR is valuable only if the full buffer contains more packets for high-rate traffic flows than for low-rate traffic flows, and that packet arrivals for each source are uniformly distributed. In reality, Mankin notes that correlations such as packet trains and TCP window operations make the distribution non-uniform.

Floyd et. al [Floyd & Jacobson 91] note that Drop Tail can unfairly discriminate against some traffic flows where there are phase differences between competing flows that have periodic characteristics. Their analysis shows that Random Drop can alleviate some of this discrimination, and led to the development of Random Early Detection.

2.5.2 Random Early Detection

Random Early Detection (RED) [Floyd & Jacobson 93] is a form of Random Drop used as congestion avoidance. A RED router randomly drops incoming packets when it believes that is is becoming congested, implicitly notifying the source of network congestion by the packet loss. RED is essentially designed to work with TCP.

Floyd and Jacobson's results indicate that Random Early Drop works well as a congestion avoidance scheme, providing fair bandwidth allocation and coping with bursty traffic better than DecBit. They also note that Random Early Drop is reasonably simple, appropriate for a wide range of environments, and does not require modifications to existing TCP implementations.

It is important to note, however, that Random Early Drop does not work with non-TCP protocols, still causes packet loss, and still takes time to find equilibrium. For transport protocols which can interpret the DecBit C-bit, Floyd and Jacobson note that RED can be used as a DecBit replacement where the C-bit is set instead of dropping the packet.

2.5.3 Summary

Loss of packets (intentional or otherwise) is considered detrimental, as work has been wasted to get the packet where it is within the network. However, if packet loss is unavoidable, packet discarding mechanisms can help to even out any unfairness in network resource usage. Where Transport protocols use packet loss to infer congestion, intentional packet dropping can be used as a form of congestion avoidance.

2.6 Packet Queueing Mechanisms

If a packet received by a router cannot be immediately retransmitted, it must be queued for retransmission. If a router is congested and has a number of packets already queued for transmission, the position where the new packet is inserted in the queue can have a positive congestion control effect, and can reduce unfair resource utilisation. In effect, a packet queueing mechanism can also be considered a queue reordering mechanism. Several queueing mechanisms have been studied which have an effect upon network congestion.

The standard packet queueing mechanism is First-In-First-Out, or First-Come-First-Served. Packets are removed from the retransmit queue (and retransmitted) in the same order that they are queued. FIFO is flow-neutral; it does not distinguish between flows with different characteristics.

2.6.1 Nagle's Fair Switching

In [Nagle 87], Nagle suggests an alternative to FIFO queueing. Take a router with two or more network interfaces. Congestion can occur if packets arrive from one interface faster than they can be dispatched to the other interfaces. Nagle replaces the packet queueing mechanism with a round-robin approach, choosing packets from queues based on the source address of each traffic flow. If a queue is empty, it is skipped. Thus, each flow gets equal access to the output interface; if one source transmits packets faster than the router can process them, it does not affect the available bandwidth for other sources: instead, the queue for the source increases, affecting its own performance.

Nagle concludes that the proposed method of packet scheduling only makes access to packet switches fair, but not access to the network as a whole.

2.6.2 Fair Queueing

A problem with Nagle's Fair Switching is that the variation in packet sizes are not taken into account. A traffic flow with 1500-octet packets gets three times as much bandwidth as a flow with 500-octet packets. Fair Queueing [Demers *et al* 89] works like Nagle's scheme in that packets are separated into separate per-flow queues, and are serviced in a round-robin fashion, but packet size variation is taken into account.

Fair Queueing simulates a strict bit-by-bit round robin queueing scheme by determining at which time each packet would finish being transmitted if done in a bit-by-bit fashion. This time is used to schedule the packet transmissions from each of the per-flow queues. Once a packet's transmission commences, it cannot be interrupted. Therefore, Fair Queueing only approximates a strict bit-by-bit round robin queueing scheme.

The effect of Fair Queueing is to limit each of N traffic flows to $1/N$ of the available output bandwidth, while there are packets in each of the per-flow queues. This prevents greedy traffic flows from obtaining an unfair amount of output bandwidth. Of course, when some per-flows are empty, they are skipped, so that the $1/N$ limit can be exceeded.

Other packet queueing mechanisms, such as Hierarchical Round Robin [Kalmanek *et al* 90], Fair Share [Shenker 94], FQNP and FQFQ [Davin & Heybey 90], have been proposed which also address the issue of fairness in resource usage.

2.6.3 Summary

Packet queueing schemes can help to ensure network bandwidth is distributed fairly, but only when buffer occupancy within routers is high, past their optimum point of operation. These schemes affect the choice of which packets to drop, and thus implicitly feed back congestion information to badly-behaved traffic sources. Packet queueing schemes, thus, are not congestion control schemes themselves, but can help alleviate congestion in conjunction with existing control schemes.

2.7 Summary

Several classes of congestion control and fair resource allocation schemes have been reviewed in this chapter. Nearly all of the control schemes have been congestion recovery schemes. Traditional Transport protocols such as TCP probe for excess bandwidth by increasing transmission rates until the network becomes congested. Only when implicit congestion information (such as packet loss) is perceived is the transmission rate reduced.

Network-based schemes which monitor the congestion state of the network and return this information to the source have been proposed and implemented. Because the Transport Layer now receives explicit congestion information, it can react to congestion more quickly. However, several round-trips are often required to reach equilibrium.

Within the network, routers with high buffer occupancies can choose different packet dropping or queueing schemes to reduce congestion and distribute resources more fairly. The drawback here is that packets are typically discarded, even with congestion avoidance schemes. This lowers the throughput of sources and wastes the resources used to forward the packet to the point where it is discarded.

The traditional control schemes reviewed suffer several drawbacks. End-to-end schemes may not utilise network resources fairly: traffic flows compete for resources, and the network does not enforce fairness. This can be partially addressed with packet queueing and packet dropping schemes. Sliding window flow mechanisms can admit bursts of packets into the network which cause short-term congestion. Finally, congestion can be detected only by end-to-end packet loss, increases in delays, or via congestion indicators such as DecBit and Source Quench: two of these indicators can be inaccurate, and the time to react to these indicators can be several round-trip times. [Eldridge 92] notes that "window-based ... congestion controls could be effective under long network delays but [are] ineffective under short network delays". The next chapter will review some rate-based control schemes which attempt to address the deficiencies outlined in this chapter.