1 GSW… Flow Control

In the perfect world, all new information arriving at a receiver can be dealt with in a speedy and efficient manner, as soon as the information arrives. Back in the real world, there are often situations in which the receiver can't cope with the rate at which information is arriving: a server can't write to a disk as fast as the data arrives from a computers memory on a new gigabit network, or a receive buffer fills up while its application is busy doing something else.

To prevent being overwhelmed, some sort of protocol that can say "hang on a minute" is required. This is known as "flow control". It's usually done (if anywhere) at the transport layer and/or the data link layer.

There are three main types of flow control in common use: software flow control, stop-and-wait flow control and sliding window flow control.

1.1 Software Flow Control

Software flow control, sometimes called XON/XOFF[1] control after the most common implementation in RS-232 serial connections, allows a signal to be sent to the transmitter telling it either to continue sending information (XON) or to stop and wait (XOFF) until an XON is sent. It's a very simple and common technique, and there are ASCII codes that are by convention assigned to the XON (0x11) and XOFF (0x13) functions[2].

This is a very efficient form of flow control: if the receiver is capable of accepting data at the maximum rate the transmitter can send it, there is no need to send any additional messages over the communication link at all. But it's not very rugged, in the sense of being immune to errors. If the receiver sends an XOFF command that is corrupted by the physical layer and never arrives at the transmitter, then it can be overwhelmed by more data than it can cope with, and some data will be lost.

There's another problem with the software approach to flow control: it doesn't really work when the transmitter and the receiver are a long way apart. The problem is the delays across the network. If the receiver's buffers start to fill up, and it sends an XOFF to the transmitter, then the flow of arriving packets will not stop immediately. First, the XOFF packet has to get to the transmitter, and during this time the transmitter is continually sending packets. Also, after the transmitter does stop sending packets, there may, on a long link, be several packets *in flight*, in other words sent but not yet received.

The total time that the receiver may continue to receive packets after sending the XOFF signal is therefore the *round trip time* (RTT) of the link: the time it takes the XOFF to get back to the transmitter, and then the last transmitted packet to get back across to the receiver, as shown in the figure below.

[1] The 'X' stands for transmitter. Quite why an 'X' is used to stand for something which doesn't contain a single 'X' anywhere in the word is not something I've been able to find out.

[2] Strictly speaking these ASCII codes are defined as "Device Control 1" and "Device Control 3", but they are most often used to send the XON and XOFF flow control signals.

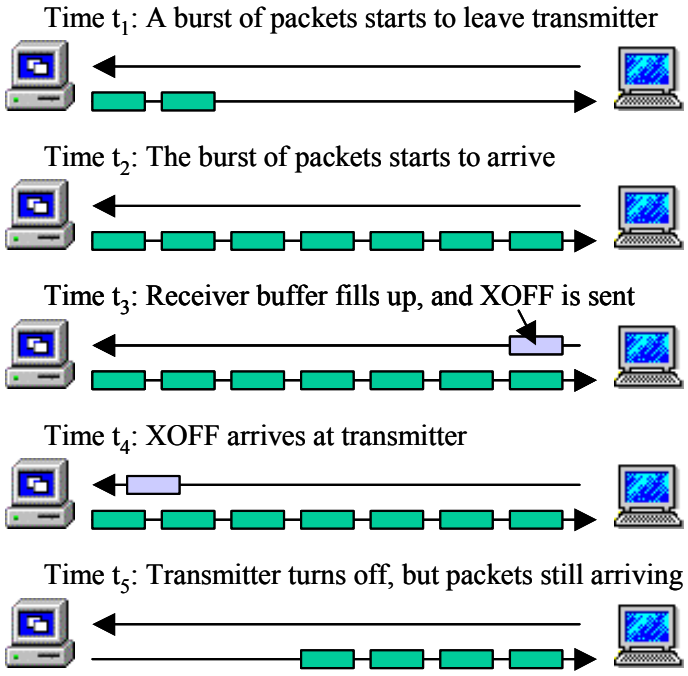

Figure 1 - Illustration of Software Flow Control

On long networks, if the receiver is to make sure that its buffers don't overflow, it has to send the XOFF when there is still enough space in the receive buffer to accommodate the amount of data the transmitter can send in one round-trip-time. On anything other than a very short point-to-point link this can be very difficult, as there is no easy way for the receiver to work out what the round-trip-time is, or what rate the transmitter can send information. For example, it's possible that the transmitter does not always want to send information at the same rate, and sends packets only infrequently for long periods of time, but then occasionally sends long, sustained bursts of packets at much faster rates. In these circumstances, an intelligent receiver trying to work out what is likely to happen to its buffer size can get the answer very badly wrong.

1.2 Stop and Wait Flow Control

To solve this problem with delays across the link, we have to put the receiver much more firmly in control of the rate of information exchange, so that it tells the transmitter not only when it wants to receive data, but also how much. There is a price to pay in terms of efficiency, but a well-designed flow control protocol can eliminate the probability of packets being lost due to overflow at the receiver's buffers.

The simplest possible way to achieve this aim is a technique called *stop and wait flow control*. This is about as easy as flow control can get. With stop and wait control, after every packet is sent, the transmitter has to wait to get an "OK, I'm ready for more" signal back from the receiver. Without this ready signal, it cannot transmit any further information. On short links and/or with very long packets, this works fine. However, when the packet is shorter (in time) than the propagation time between the nodes, a lot of the potential link capacity can be wasted, as the transmitter is spending most of its time just waiting for the "ready" signal to come back from the receiver.

A *utilisation* can be defined for networks that are flow-controlled: the utilisation is the ratio of the data actually sent over the network to the data that could have been sent if the transmitter

was transmitting all the time: in other words, the ratio of the *throughput* to the *network capacity*[3].

1.2.1 Utilisation of Stop and Wait Flow Control

To estimate the utilisation of stop and wait flow control, we need to make a few assumptions. Firstly, note that the total time it takes to send a packet to the other end of the network and receive an acknowledgement is composed of six terms:

t_{packet} = the time taken to transmit the packet
t_{prop} = the time taken for the packet to get to the receiver across the network
t_{rxproc} = the time it takes the receiver to process the packet and queue an acknowledgement
t_{ack} = the time taken to transmit an acknowledgement
t_{prop} = the time taken for the acknowledgement to get back to the transmitter[4]
t_{txproc} = the time taken for the transmitter to process the acknowledgement and queue the next packet

In most cases (with communication processors at either end much faster than the link between them), we can neglect t_{rxproc} and t_{txproc}, and write:

$$Utilisation = \frac{tx\_bit\_rate\ (t_{packet})}{tx\_bit\_rate\ (t_{packet} + 2t_{prop} + t_{ack})} = \frac{t_{packet}}{t_{packet} + 2t_{prop} + t_{ack}} \qquad (0.1)$$

In other words, the utilisation is just the ratio of the time spent actively transmitting a packet to the total time between starting to send one packet and starting to send the next packet. Sometimes, especially when the data packets are long, we can make the further assumption that the acknowledgement takes a very short time to transmit with respect to the time taken to transmit the packet, so that:

$$t_{packet} >> t_{ack} \qquad (0.2)$$

and hence we can write:

$$Utilisation = \frac{t_{packet}}{t_{packet} + 2t_{prop}} = \frac{1}{1 + 2a} \qquad (0.3)$$

where a is the ratio of the propagation time across the link to the time required to transmit a packet. This is a useful characteristic for a link (and a network), and we'll continue to use it.

Note for low values of a, the utilisation is pretty good (this implies long packets and/or short propagation times). For high values of a (short packets and/or long propagation times), the utilisation can be pretty awful, and we need to find a better way of doing things.

---

[3] Both throughput and network capacity can be measured in either bits per second or packets per second: when calculating utilisation make sure that they are both in the same units.

[4] Note that we are assuming here that the time taken to get the packet to the receiver (t_{prop}) is the same as the time taken to get the acknowledgement back to the transmitter. This may not be true in all cases, especially if the route taken by packets going in one direction is very different from the route taken by the packets going in the other direction.

The usual way of illustrating *exchanges* or *conversations* between the transmitter and receiver is by drawing arrows showing the passage of the last bit in any packet. For stop-and-wait, such a drawing would look like this:

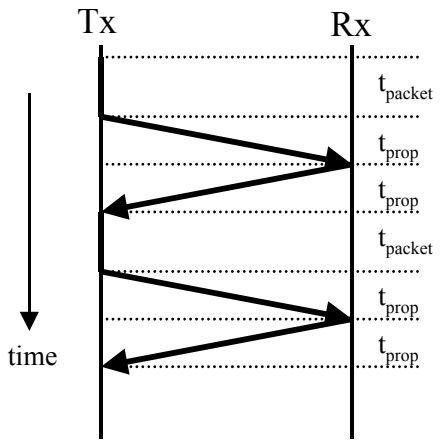

Figure 2 - Stop and Wait Flow Control

1.3 Sliding Window Flow Control

A simple idea: if we increase the buffer size at the receiver, then by returning the "ready" signal the receiver can indicate that it can receive the next N packets (where N is the free space in the buffer), instead of just the next one. Put this another way: when the transmitter receives an acknowledgement for packet X, it knows that it's allowed to transmit all packets up to, and including, $X+N$. N is known as the *window size*. Sometimes this window size N is fixed, other times the receiver varies the size of the window depending on how much space is left in its buffer.

If the propagation time across the network and back is less than the time the transmitter requires to send $N - 1$ packets, then the transmitter can continuously transmit, giving a utilization of 100%. With longer propagation times (or shorter packets) the transmitter will still have to wait, but it can now transmit N packets in the time $t_{packet} + 2t_{prop}$, rather than just one.

For this to work, the receiver and transmitter must be able to identify packets, and this is done by giving each packet a sequence number. The "ready" signal coming back from the receiver contains the number of the next packet it is expecting[5], so by returning a "ready-4" packet, the receiver is saying it is willing to accept another N packets, starting with the packet with sequence number 4. Some of these packets may already have been transmitted.

Another way of thinking of the window size is that it is the maximum number of packets that the transmitter is allowed to send without receiving an acknowledgement for any of them; we call these transmitted but unacknowledged packets "*outstanding packets*".

[5] This is conventional. It could just transmit back the number of the packet that had just arrived, but as we'll see in the chapter on error control, sending the next packet the receiver wants to receive is more useful.

1.3.1 Utilisation of Sliding Window Flow Control

Just as with stop-and-wait control, we can calculate the utilisation for sliding window, here I'll assume that the window size *N* is constant for simplicity.

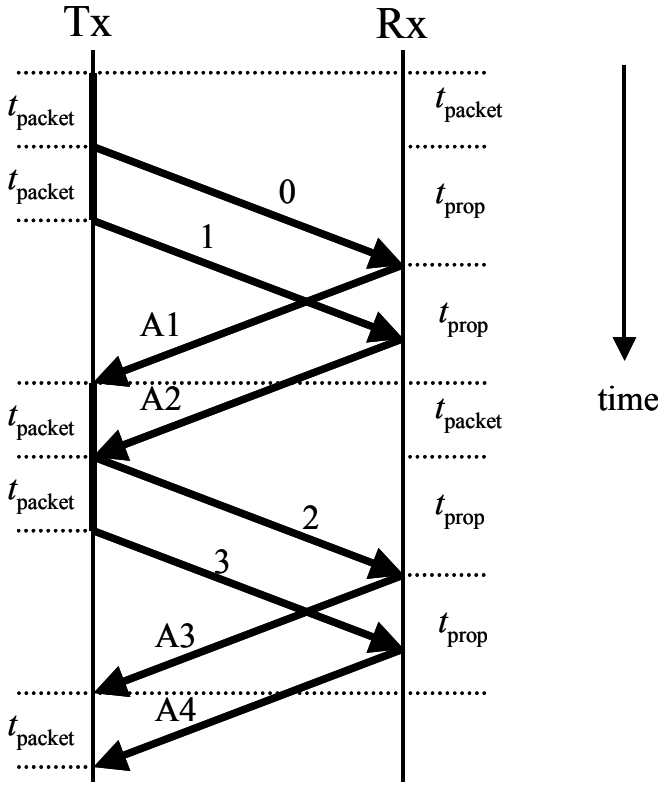

Figure 3 - Sliding Window Flow Control

The diagram above illustrates a simple case with a window size of two. "A1" is the acknowledgement sent after packet 0 has arrived, asking for packet number one, and so on. Clearly, the utilisation in this case is:

$$Utilisation = \frac{2\,t_{packet}}{t_{packet} + 2\,t_{prop}} = \frac{2}{1+2a} \tag{0.4}$$

so in general, for a window size of N, the utilisation might be thought to be:

$$Utilisation = \frac{N\,t_{packet}}{t_{packet} + 2\,t_{prop}} = \frac{N}{1+2a} \tag{0.5}$$

However, this formula cannot be generally true, since with low values of a, this can easily predict a utilisation greater than 100%, which is clearly impossible. This formula is only valid in the cases where $t_{packet} + 2t_{prop} > N\,t_{packet}$, in other words when $1+2a > N$.

When $1 + 2a < N$, the first acknowledgement arrives back at the transmitter before it has had time to finish sending N packets. In this case the transmitter doesn't need to wait before it sends the next packet, and can just continue sending packets continuously, so the utilisation is 100 %. We can write:

$$Utilisation = \begin{cases} \dfrac{N}{1+2a} & N \le 1+2a \\ 1 & N > 1+2a \end{cases} \qquad (0.6)$$

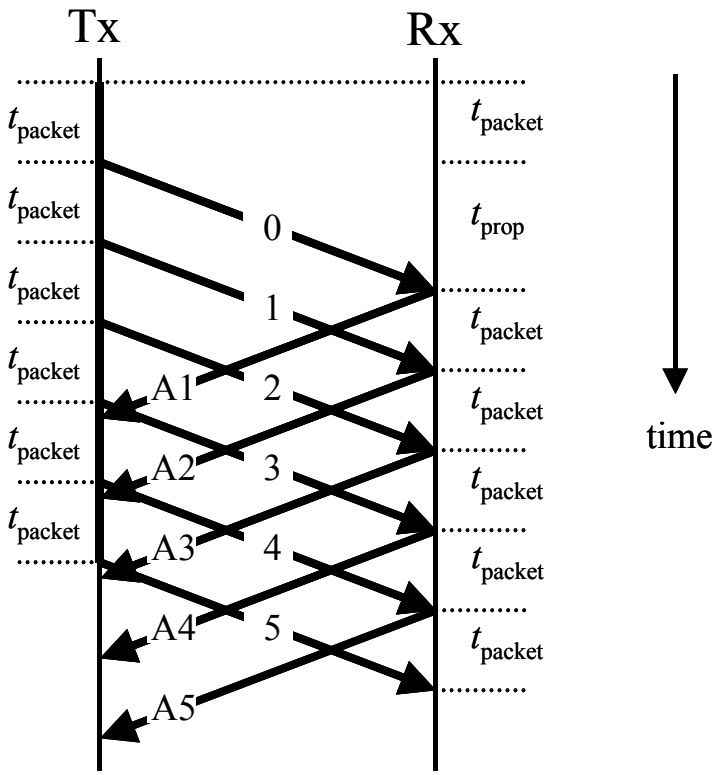

Figure 4 - Sliding Window with 100% Utilisation

At least in theory, provided the window size N is chosen large enough so that the transmitter cannot send N packets before the first "ready" signal is received, this scheme can provide 100% utilisation.

1.4 Key Points

- Flow control is necessary to prevent receive buffers overflowing. It's required whether there are any errors in the transmission or not.
- Flow control requires a bi-directional link: the receiver must be able to send signals back to the transmitter.
- Flow control reduces *utilisation*. Utilisation is defined as *throughput* / *network capacity*. *Throughput* is the amount of useful data that arrives at the other end of the link, the *network capacity* is the maximum possible amount of data that could be sent across the link.
- Software flow control is simple, but does not always guarantee that overflows will not occur, especially on long/fast links.
- Stop-and-wait flow control is very simple, but inefficient on long/fast links.

- Sliding window flow control can provide much greater utilisation on long/fast links at the expense of larger buffers at the receiver and greater complexity.

1.5 Tutorial Questions

1) A network link is so long that there is just time to transmit ten packets before the first packet has been fully received. Derive an expression for the utilisation of a sliding window flow-control scheme as a function of the window size, and plot the results. Assume no transmission errors and state any other assumptions you make.

2) A link from York to Leeds (30 miles, at a propagation speed of 2/3 of the speed of light), uses 64 byte packets, a transmission speed of 10 Mbit/s, and stop-and-wait flow control. What is the maximum utilisation of this link?

3) Consider the link in the previous question, if a sliding window protocol was used. How big must the window size be if the maximum utilisation is to be at least 50%?

*4) Some protocols use a window size defined not in terms of a number of packets, but in terms of the total number of bytes of information within the packets. What are the advantages and disadvantages of doing this?

*5) One well-known protocol has a window size of 64 kbytes of information. Lots of people use this protocol over geo-stationary satellites.

a) What is the maximum possible throughput of this link?
b) What is the maximum possible utilisation?
c) What could be done to improve the throughput of this system?

(Note: geo-stationary satellites are approximately 36,000 km away, and the speed of light is $3x10^8$ m/s.)