FLOW CONTROL

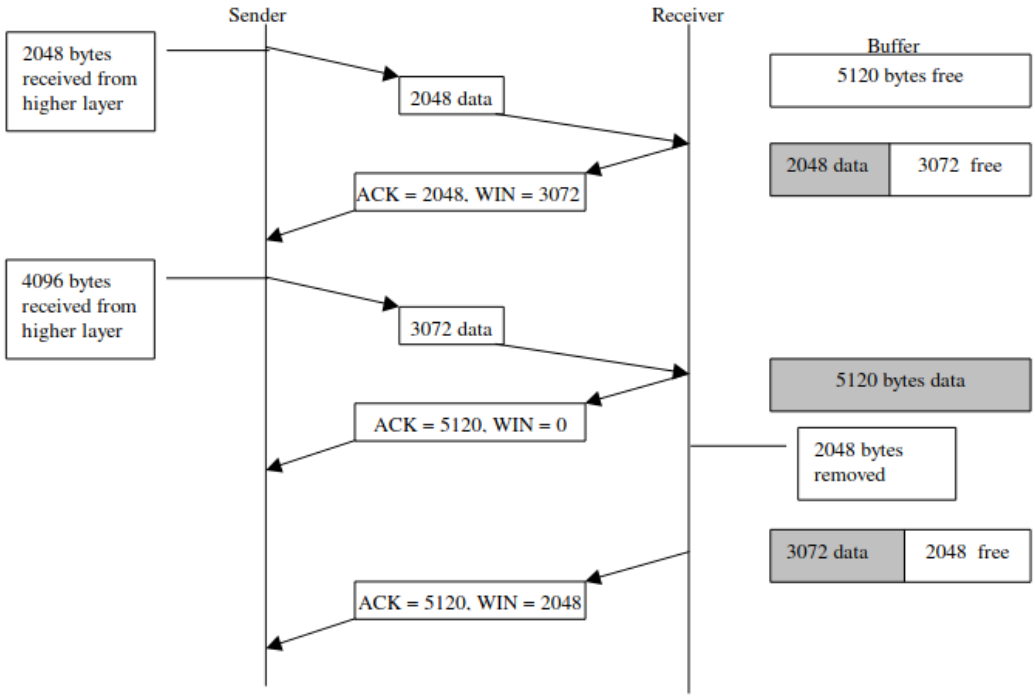

Figure 1: Flow control in TCP

The transmission process in TCP is regulated by a sliding window flow control policy. However, this policy is slightly different from the SRP and GBN algorithms you have seen in the previous lab. In particular, the receiver not only acknowledges any segments received correctly, but it also advertises the available buffer space. This flow control method thus prevents the receiver from being flooded with information.

All acknowledgements or windows are treated as byte streams. Thus, TCP will not acknowledge a certain PDU, but the number of bytes received correctly. Similarly, it will advertise the number of bytes available in the receiver's buffer, rather than the number of PDUs which it would be able to receive.

For example, in figure 1 the TCP layer on the senders side initially receives 2048 bytes from a higher layer for transmission. Assuming the connection to the receiver's side has already been established, a 2048 byte segment is transmitted. Upon successful receipt, the receiver returns an acknowledgement, acknowledging the receipt of the 2048 bytes and advertising that its receive buffer still has 3072 bytes free for more data. The sender then obtains more data from its higher layer protocol for transmission, say 4096 bytes. However, since the advertised window was only 3072, a segment of 3072 bytes is transmitted. Upon receipt of this segment, the receiver's buffer is then full, and the receiver advertises this by sending an acknowledgement with window size 0. Only after the receiver has removed some of the bytes from its buffer, it will advertise again that it now has more space available to receive data. This indicates to the sender that it may now continue to transmit, in case it has received more data from its higher layer protocols.

NAGLE'S A:GORITHM:

One of the problems faced by TCP/IP is the overhead associated with small user messages. For example, a TELNET application of an interactive editor sends individual characters for transmission. Each character typed on the senders keyboard is segmented into a TCP PDU, adding 20 bytes of overhead. This TCP PDU is then passed down to the IP layer, which adds another 20 bytes of overhead. Thus, a 1-byte message generates a 41-bytes packet for the data-link layer. Furthermore, the receivers TCP immediately acknowledges the receipt of this PDU, sending a minimum of 40 bytes to the sender (assuming no flows in the reverse direction). Then, the editor at the receivers side reads the byte from the buffer, thus generating a window update packet, similar to the one shown in figure 1. This generates another 40-byte packet. And finally, the editor echoes the character back to the sender so that its screen can be updated. Thus, another 41-byte message is sent from the sender to the receiver. This sequence of events is repeated for every 1-byte message generated by the keyboard. To summarize, a 1-byte message generates a flow of 41 bytes in the direction sender-receiver and 121 bytes in the direction receiver-sender.

No doubt that this procedure is very inefficient and wastes a lot of time and bandwidth. To overcome this problem, Nagle's algorithm is used. This algorithms resolves the problem by only sending the first character as outlined in the last paragraph. In the meantime, the remaining characters typed at the sender's side are buffered, until the outstanding character is acknowledged. When this acknowledgement arrives at the sender, hopefully a number of characters have accumulated in the buffer, and they can all be transmitted in a single TCP PDU. Furthermore, on the receiver's side, the acknowledgements are artificially delayed for a pre-specified amount of time. This means that both the window update and the acknowledgement for the character can be piggy-backed onto the echoed-PDU. Using these two schemes, only 2 PDU's flow in each direction, generating much less overhead traffic and thus a more efficient transmission channel.

NAGLE ALGORITHM:

If there is data to send but the window is open less than MSS, then we may want to wait some amount of time before sending the available data.But how long?

If we wait too long, then we hurt interactive applications like Telnet

If we don’t wait long enough, then we risk sending a bunch of tiny packets and falling into the silly window syndrome

The solution is to introduce a timer and to transmit when the timer expires

We could use a clock-based timer, for example one that fires every 100 ms

Nagle introduced an elegant self-clocking solution

Key Idea

As long as TCP has any data in flight, the sender will eventually receive an ACK

This ACK can be treated like a timer firing, triggering the transmission of more data

When the application produces data to send if both the available data and the window ≥ MSS

send a full segment

else

if there is unACKed data in flight buffer the new data until an ACK arrives

else

send all the new data now

Silly Window Syndrome

2.1.5 SILLY-WINDOW SYNDROME

The silly-window syndrome is somewhat related to the above algorithm. Rather than considering the effect of a sender generating 1-byte messages, it considers the effect of the receiver removing data from the buffer in 1-byte chunks. In this case, every time 1-byte is removed from the receiver's buffer, a window update would be transmitted to the sender, indicating that another 1 byte is now free at the receiver's buffer. The sender would transmit a 1-byte message, generating a 41-byte segment. The situation described above would re-occur.

This problem has been addressed by Clark in 1982. His solution is to delay the window update until an appropriate amount of buffer space is available at the receiver's side. Typically, this appropriate amount is defined to be at least 1 maximum PDU size, as determined upon connection establishment. Alternatively, the window update would be transmitted if half of the receiver's buffer is available.

Note that conditions leading to Nagle's algorithm and the silly-window syndrome are related. Similarly, the solutions to both problems are related. Basically, artificial delays are introduced to buffer data and PDUs are only transmitted if it is considered efficient to do so.