Simplex Stop and Wait Protocol

- **Flow control** deals with the problem that the sender transmits frames faster than the receiver can accept, and the solution is to limit the sender into sending no faster than the receiver can handle.

- Consider the **simplex case**: data is transmitted in one direction (Note although data frames are transmitted in one direction, frames are going in both directions, i.e. link is **duplex**).

- **Stop and wait**: sender sends one **data frame**, waits for **acknowledgement** (ACK) from receiver before proceeding to transmit the next frame.
  - This simple flow control will break down if ACK gets lost or errors occur → sender may wait for ACK that never arrives.

```c
/* Protocol 2 (stop-and-wait) also provides for a one-directional flow of data from sender to receiver. The communication channel is once again assumed to be error free, as in protocol 1. However, this time, the receiver has only a finite buffer capacity and a finite processing speed, so the protocol must explicitly prevent the sender from flooding the receiver with data faster than it can be handled. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender2(void) {
    frame s; /* buffer for an outbound frame */
    packet buffer; /* buffer for an outbound packet */
    event_type event; /* frame_arrival is the only possibility */
    
    while (true) {
        from_network_layer(&buffer); /* get something to send */
        s.info = buffer; /* copy it into s for transmission */
        to_physical_layer(&s); /* bye-bye little frame */
        wait_for_event(&event); /* do not proceed until given the go ahead */
    }
}

void receiver2(void) {
    frame r, s; /* buffers for frames */
    event_type event; /* frame_arrival is the only possibility */
    
    while (true) {
        wait_for_event(&event); /* only possibility is frame_arrival */
        from_physical_layer(&r); /* get the inbound frame */
        to_network_layer(&r.info); /* pass the data to the network layer */
        to_physical_layer(&s); /* send a dummy frame to awaken sender */
    }
}
```
Simplex Stop and Wait with ARQ

- For noisy link, pure stop and wait protocol will break down, and solution is to incorporate some error control mechanism

- **Stop and wait with ARQ**: Automatic Repeat reQuest (ARQ), an error control method, is incorporated with stop and wait flow control protocol
  - If error is detected by receiver, it discards the frame and sends a negative ACK (NAK), causing sender to re-send the frame
  - In case a frame never got to receiver, sender has a **timer**: each time a frame is sent, timer is set
    - If no ACK or NAK is received during timeout period, it re-sends the frame
  - Timer introduces a problem: Suppose timeout and sender retransmits a frame but receiver actually received the previous transmission → receiver has duplicated copies
  - To avoid receiving and accepting two copies of same frame, frames and ACKs are alternatively labeled 0 or 1: ACK0 for frame 1, ACK1 for frame 0

- An important **link parameter** is defined by

\[
a = \frac{\text{propagation time}}{\text{frame time}} = \frac{R d}{V L}
\]

where $R$ is data rate (bps), $d$ is link distance (m), $V$ is propagation velocity (m/s) and $L$ frame length (bits)
Stop and Wait with ARQ (continue)

- In error-free case, **efficiency** or maximum **link utilisation** of stop and Wait with ARQ is:

\[
U = \frac{1}{1 + 2a}
\]

- Illustration of how stop and wait with ARQ works:
  ACK0: frame 1 is received, waiting for next (frame 0)
  ACK1: frame 0 is received, waiting for next (frame 1)

  This is to have **1-bit sequence number**, and implies receiver have buffer for one frame

- For an LAN with \( R = 10 \text{ Mbps} \) and \( d = 1 \text{ km} \), using \( V = 2 \times 10^8 \text{ m/s} \) and \( L = 500 \text{ bits} \), \( a = 0.1 \) and stop-and-wait procedure has \( U = 0.83 \), which has adequate line utilisation

  But for a satellite link, link utilisation for stop-and-wait procedure may only be \( U = 0.001 \) or lower, which is clearly wasteful
Sliding Window Protocol

- For large link parameter $a$, stop and wait protocol is inefficient.

- A universally accepted flow control procedure is the **sliding window protocol**
  - Frames and acknowledgements are numbered using **sequence numbers**
  - Sender maintains a list of sequence numbers (frames) it is allowed to transmit, called **sending window**
  - Receiver maintains a list of sequence numbers it is prepared to receive, called **receiving window**
  - A sending window of size $N$ means that sender can send up to $N$ frames without the need for an ACK.
  - A window size of $N$ implies buffer space for $N$ frames.
  - For $n$-bit sequence number, we have $2^n$ numbers: $0, 1, \ldots, 2^n - 1$, but the maximum window size $N = 2^n - 1$ (not $2^n$).
  - ACK3 means that receiver has received frame 0 to frame 2 correctly, ready to receive frame 3 (and rest of $N$ frames within window).

- In error-free case, efficiency or maximum **link utilisation** of sliding window protocol is:

  $$U = \begin{cases} 
  1, & N > 2a + 1 \\
  \frac{N}{1+2a}, & N < 2a + 1
  \end{cases}$$

  Thus it is able to maintain efficiency for large **link parameter** $a$: just use large widow size $N$. 
Sliding Window (continue)

- Note that $U = 1$ means that link has no idle time: there are always something in it, either data frames or ACKs.

- Consider the case of 3-bit sequence number with maximum window size $N = 7$.

- This illustration shows that sending and receiving windows can shrink or grow during operation.
  The receiver do not need to acknowledge every frames.

- If both sending and receiving window sizes are $N = 1$, the sliding window protocol reduces to the stop-and-wait.

- In practice, error control must be incorporated with flow control, and we next discuss two common error control mechanisms.
Go-back-n ARQ

- The basic idea of **go-back-n error control** is: If frame $i$ is damaged, receiver requests retransmission of all frames starting from frame $i$.

An example:

- Notice that all possible cases of damaged frame and ACK / NAK must be taken into account.

- For $n$-bit sequence number, maximum window size is $N = 2^n - 1$ not $N = 2^n$ → with $N = 2^n$ confusion may occur.

- Consider $n = 3$, if $N = 8$ what may happen:
  - Suppose that sender transmits frame 0 and gets an ACK1
  - It then transmits frames 1,2,3,4,5,6,7,0 (this is allowed, as they are within the sending window of size 8) and gets another ACK1
  - This could mean that all eight frames were received correctly
  - It could also mean that all eight frames were lost, and receiver is repeating its previous ACK1
  - With $N = 7$, this confusing situation is avoided.
Selective-Reject ARQ

- In **selective-reject** ARQ error control, the only frames retransmitted are those receive a NAK or which time out

  An illustrative example:

- Selective-reject would appear to be more efficient than go-back-n, but it is harder to implement and less used

- The window size is also more restrictive: for $n$-bit sequence number, the maximum window size is $N = \frac{2^n}{2}$ to avoid possible confusion

- Go-back-n and selective-reject can be seen as trade-offs between **link bandwidth** (data rate) and data link layer **buffer space**
  - If link bandwidth is large but buffer space is scarce, go-back-n is preferred
  - If link bandwidth is small but buffer space is pretty, selective-reject is preferred
From Simplex to Duplex

- So far, we consider data transmission in one direction (simplex), although the link is duplex.
- If two sides exchange data (duplex), each needs to maintain two windows: one for transmitting and one for receiving.
- In **duplex** communication, frames transmitted from either side can be data, ACKs and NAKs → the need to distinguish them.
- **Frame type:** Recall in frame header there is a control field, and part of it is typically used as frame type field to tell what type the frame is.
- **Piggybacking:** In duplex situations, piggybacking is often used → If one has data and an ACK to send, it sends both in one frame.
- Discussion so far: data link layer is primarily concerned with making point-to-point link reliable
  - It is responsible for transmitting frames from sender to receiver (service to network layer), and can only uses physical layer to do the job.
  - It has to take into account that transmission error may occur and sender/receiver may operate at different speeds → error control/flow control (ACKs, NAKs, CRC, windows, sequence numbers).
  - Next lecture will see how all these fit into some data link layer protocols.
ELEC3030 (EL336) Computer Networks

S Chen

while (true) {  
    wait_for_event(&event);  // four possibilities: see event_type above */
    switch (event) {
      case network_layer_ready:  // the network layer has a packet to send */
        /* Accept, save, and transmit a new frame. */
        from_network_layer(&buffer[next_frame_to_send]);  // fetch new packet */
        nbuffered = nbuffered + 1;  // expand the sender's window */
        send_data(next_frame_to_send, frame_expected, buffer);  // transmit the frame */
        inc(next_frame_to_send);  // advance sender's upper window edge */
        break;
      case frame_arrival:  // a data or control frame has arrived */
        from_physical_layer(&r);  // get incoming frame from physical layer */
        if (r.seq == frame_expected) {  // Frames are accepted only in order. */
            to_network_layer(&info);  // pass packet to network layer */
            inc(frame_expected);  // advance lower edge of receiver's window */
        }
      /* Ack implies n – 1, n – 2, etc. Check for this. */
      while (between(ack_expected, r.ack, next_frame_to_send)) {  // Handle piggybacked ack. */
          nbuffered = nbuffered – 1;  // one frame fewer buffered */
          stop_timer(ack_expected);  // frame arrived intact; stop timer */
          inc(ack_expected);  // contract sender's window */
      }
      break;
      case checksum_err:  // just ignore bad frames */
      case timeout:  // trouble; retransmit all outstanding frames */
        next_frame_to_send = ack_expected;  // start retransmitting here */
        for (i = 1; i <= nbuffered; i++) {  // send_data(next_frame_to_send, frame_expected, buffer);  // resend 1 frame */
            inc(next_frame_to_send);  // prepare to send the next one */
        }
    }
    if (nbuffered < MAX_SEQ)  // enable_network_layer();
        enable_network_layer();
    else  // disable_network_layer();
        disable_network_layer();
}

Sliding Window with Go-back-n C Codes
Protocol Verification

- How to know a protocol really works → specify and verify protocol using, e.g. finite state machine
  - Each protocol machine (sender or receiver) is at a specific state at every time instant
  - Each state has zero or more possible transitions to other states
  - One particular state is initial state: from initial state, some or possibly all other states may be reachable by a sequence of transitions

- Simplex stop and wait ARQ protocol:
  - State $SRC$: $S = 0, 1$ → which frame sender is sending; $R = 0, 1$ → which frame receiver is expecting; $C = 0, 1, A$ (ACK), $-$ (empty) → channel state, i.e. what is in channel
  - There are 9 transitions
  - Initial state (000): sender has just sent frame 0, receiver is expecting frame 0, and frame 0 is currently in channel
  - Transition 0 consists of channel losing its contents, transition 1 consists of channel correctly delivering frame 0 to receiver, and so on
  - During normal operation, transitions 1,2,3,4 are repeated in order over and over: in each cycle, two frames are delivered, bringing sender back to initial state
Summary

- Flow control and error control techniques for data link layer:
  Stop and wait ARQ, sliding window, go-back-n, selective-reject (repeat)

- Data link layer (part I) discussed so far:
  It is concerned with making a point-to-point link reliable, and is responsible for transmitting frames from sender to receiver, can only use physical layer to do job

- Error control and flow control (ACKs, NAKs, CRC, sliding windows, sequence numbers, go-back-n etc.):
  How these are included in a data link layer protocol will be discussed in next lecture