Transport / Network layer error detection

- Typically implemented in software, so need simple & fast schemes

- Internet checksum used by IP, TCP, UDP
  - Bytes of data treated as 16-bit integers & summed w/ 16-bit 1s complement arithmetic
  - 1s complement of this sum is the Internet checksum

\[ \begin{align*}
  &\quad 0110 \ 0110 \ 0110 \ 0000 \quad (1) \\
+ &\quad 0101 \ 0101 \ 0101 \ 0101 \quad (2)
\end{align*} \]
\[ 1000 \ 1111 \ 0000 \ 1100 \quad (3) \]
- Add values 2 at a time, adding any overflow back into the LSB

\[ \begin{align*}
  &\quad 0110 \ 0110 \ 0110 \ 0000 \quad (1) \\
+ &\quad 0101 \ 0101 \ 0101 \ 0101 \\
\hline
  &\quad 1011 \ '1011 \ '1011 \ '0101 \no \ overflow \quad (2)
+ &\quad 1000 \ 1111 \ 0000 \ 1100 \\
\hline
  &\quad 0100 \ 1010 \ 1100 \ 0001 \quad (3)
\end{align*} \]

\[ \begin{align*}
  &\quad 0100 \ 1010 \ 1100 \ 0010 \\
+ &\quad 1011 \ 0101 \ 0011 \ 1101 \ \text{add overflow}
\end{align*} \]

(Other implementations may be more efficient depending on the machine)
- Checksum is transmitted in packet header
- Receiver repeats calculation on received data & compares with checksum
  (or equivalently adds data + checksum with 16 bit 1's complement arithmetic & checks that the result is 1111111111111111)
- In TCP & UDP, the Internet checksum is computed over all fields (header + data)
- In IP, the checksum is computed over the IP header
- Relatively weak protection against errors when compared with CRC
- Although many link layer protocols provide error detection, no guarantee that all links do; also bit errors may be introduced when a packet is stored in a router's memory \( \rightarrow \) transport layer error detection still needed but can be more lightweight since most errors are picked up by stronger error detection at link level
although a pure end-to-end argument might suggest that link layer error detection is superfluous, it allows for functions to be incompletely provided at lower levels as a performance optimization (eg it is a waste to transmit a corrupted packet all the way to the end host before detecting the error)
Reliable Transmission

- Consider problem of reliable transmission across an unreliable link
- Usually accomplished using a combination of 2 fundamental mechanisms — acknowledgments & timeouts
  - Receipt of an ACK indicates to the sender of the original frame that it was successfully received
  - Otherwise, after a period of time without receiving an ACK (timeout), the sender retransmits
- General strategy is called automatic repeat request (ARQ)
- Error detection code used

- Alternative — forward error correction (FEC) using error-correcting code (more redundancy)
  - Useful when errors occur frequently, e.g., in wireless, or retransmission latency is unacceptable
ARQ strategies

*Stop & Wait*
- after transmitting a frame, sender waits for an ack before transmitting the next frame.
- if ack is not received within the timeout period, the sender retransmits.

- main disadvantage: can only have 1 outstanding frame at a time, which may be far below link capacity.
  - e.g. a 1.5 Mbps link with a 45 ms RTT
    - delay x bandwidth product = 67.5 Kb
    - if only 1 frame of size 1 KB is sent per RTT, rate = \( \frac{\text{bits per frame}}{\text{time per frame}} \)

\[
\leq \frac{1024 \times 8}{0.045}
\]

\( = 182 \text{ Kbps} \)

\( \approx \frac{1}{8} \times \text{link capacity} \)

- to keep the pipe full, sender should be able to transmit 8 frames before having to wait for an ack.
Concurrent logical channels e.g. ARPANET data link protocol
- multiplex several logical channels onto a single point-to-point link
- run stop-and-wait algorithm on each logical channel
- when sender has a frame to send, it uses the lowest idle channel
- does not preserve frame order
Sliding Window of TCP, X.25 link layer

- limit the amount of unacknowledged data transmitted — limit is called the transmission window \( W \)

- when \( W \) units of data are outstanding, the sender stops & waits for an ACK

  oldest unacknowledged data unit

\[ \text{ACKed} \text{ window cannot be sent yet} \]

- left edge of window corresponds to oldest unacknowledged data unit; if ACKed, window shifts one unit to the right

- for a fixed window size \( W \), packet transmission rate \( = \frac{W}{\text{RTT}} \geq \text{goodput (actual data rate)} \)

  since there are retransmissions usually
- Consider transmission of a length-$L$ packet & a length-$a$ ACK over a link of capacity $c$ & propagation delay $S$

\[
\begin{align*}
  t &= 0 \\
  t &= \frac{L}{c} \\
  t &= S \\
  t &= S + \frac{L + a}{c} \\
  t &= 2S + \frac{L + a}{c}
\end{align*}
\]

- Base RTT (time taken for a packet to be sent & its ACK received) $= 2S + \frac{L + a}{c}$

- BRTT in terms of packet transmission time $\frac{L}{c}$

\[
\frac{2SC}{L} + 1 + \frac{a}{L} \approx \frac{2SC}{L} + 1 \quad \text{if } a \ll L
\]

bandwidth $\times$ (RTT propagation) delay product in packets $= \Delta$
- for a single transmitter to use full link capacity, window size should ideally be $\Delta + 1$ packets

```
sender

receiver
```

- full round-trip pipe if no losses
  - forward link contains $\frac{\Delta}{2}$ back-to-back packets not yet received
  - receiver has received $\frac{\Delta}{2} + 1$ packets
  - reverse link contains $\frac{\Delta}{2}$ Acks spaced by packet times

- multi-hop: sender does not know $\Delta$, nor how many other transfers are sharing the pipe
  → must learn the correct window size adaptively

- even if optimal window size is known, if the sender is not directly connected to the bottleneck link, it should not start by sending a full window of packets all at once → packets would pile up at the bottleneck & might be dropped if buffer space is insufficient
transmitter should increase window size gradually (slow-start)
TCP service model

- **Connection-oriented service**
  - client & server exchange transport-layer control info with each other before starting to exchange application-layer messages (handshaking)
  - after handshaking, a TCP connection exists between the sockets of the 2 processes
  - full-duplex connection (both processes can send messages to each other simultaneously)
  - when application finishes sending, the connection is torn down

- **Reliable & sequential packet transport service**
  - communicating processes can rely on TCP to correctly deliver all data in order to the receiving socket

- TCP also includes flow control & congestion control

- Secure Sockets Layer: application-layer enhancement to TCP providing security services
• TCP runs the sliding window protocol end-to-end over the Internet
  - need an explicit **connection establishment** phase to exchange parameters & establish some shared state, & a **connection teardown** phase to let each host know it is OK to free this state

- RTTs vary widely across different connections & even during a connection ➔ need **adaptive timeout**
- packets may be reordered or substantially delayed ➔ TCP assumes a **maximum segment lifetime MSL** (estimate of how long a packet might live in the Internet), 120s

- Internet hosts vary widely
  ➔ each host needs to learn the other side's receiving buffer space (flow control)

- links traversed en route to destination may vary widely in capacity/congestion ➔ unlike the case of a single link, congestion is not directly visible to sender in the form of queue buildup at the sender
  ➔ need **congestion control mechanisms**