Distributed Queue:
Issues:
- Packets can be dropped, reordered
- Delays may vary
  latency — time between send & receive
  throughput — how much data over time

Idea:
- Put sequence numbers on packets
  (receiver can reorder the packets)
- Use sequence # if there's a missing one,
  the receiver could complain
  sender could ask the receiver if it received
  everything
. Sender puts seq #s on packets
. Receiver sends ACKs for a particular seq, #
. If sender doesn't get ACK, resend

Diagram:

1. \[ \circ \rightarrow \circ \]
2. \[ \circ \rightarrow \text{ACK} \]
3. \[ \circ \rightarrow \text{ACK} \]
4. \[ \circ \rightarrow \text{ACK} \]
5. \[ \circ \rightarrow \text{ACK} \]
Sender algorithm
\[i = 0\]
\[\text{while } (i < \text{numPackets}) \{\]
\[\quad \text{Send packet } i\]
\[\}
\[\text{in parallel:}\]
\[\quad \text{when ACK}_i \text{ is received}\]
\[\quad i = i + 1\]

Problem: SLOW due to latency — round trip delay between each packet

Idea: Send several packets before waiting for the ACK

Receiver algorithm
\[i = 0\]
\[\text{on receiving packet } j\]
\[\quad \text{if } j = = i\]
\[\quad \text{send } \text{ACK}_i \text{ repeatedly}\]
\[\quad i = i + 1\]
\[\text{in parallel:}\]
\[\quad \text{deliver packets to application in order}\]
One way:
Send a burst
wait for ACK that entire burst was received

Problem: periods of no activity

Another way:
messages
before sending this message,
wait for ACK from

Receiver reorders packets for the application
Algorithm for adjusting the window size:

1. Start small
2. Gradually increase until don't get an ACK in time to advance the window
3. Drop off quickly

TCP sliding window protocol
Transmission Control Protocol
TCP/IP