

Transport Layer (I)

Overview

- Transport layer is an intermediate layer in computer networks, which bridges the application layer and network layer.
 - *Application Layer*: needs reliable, bidirectional stream-oriented service (continuous flow of data)
 - *Network Layer*: provides unreliable datagram service
 - Transport layer follows the **End-to-End Principle**
 - it tries to keep the network as simple as possible
 - so, all “smarts” are implemented at the end nodes
-

Functions of Transport Layer

- Addressing
 - we know that transport layer uses port numbers, why?
 - transport layer: identify logical communication between processes
 - network layer: maintain logical communication between hosts
 - Error control
 - packet loss detection and retransmissions
 - Flow and congestion control
 - controlling the transmission rate
 - Session management
 - establish, manage, and terminate connections
-

Error-control Method

- We will first talk about the error control method in the transport layer, which provide the reliability for this layer.
- The reliability is achieved if packets are delivered despite unreliability of the network layer or it detects that delivery is impossible.
- The error-control method used in the transport layer is **Automatic Repeat reQuest** (ARQ). This method has six components:
 1. **acknowledgement**
 - receiver explicitly tells sender that a packet is received OK
 - negative acknowledgments (NAK): receiver explicitly tells sender that pkt had errors (this is unnecessary, since duplicate ACK at sender addresses this)
 - so, sender will know whether the packet is delivered or whether the packet needs to be retransmitted.

2. **timeout**

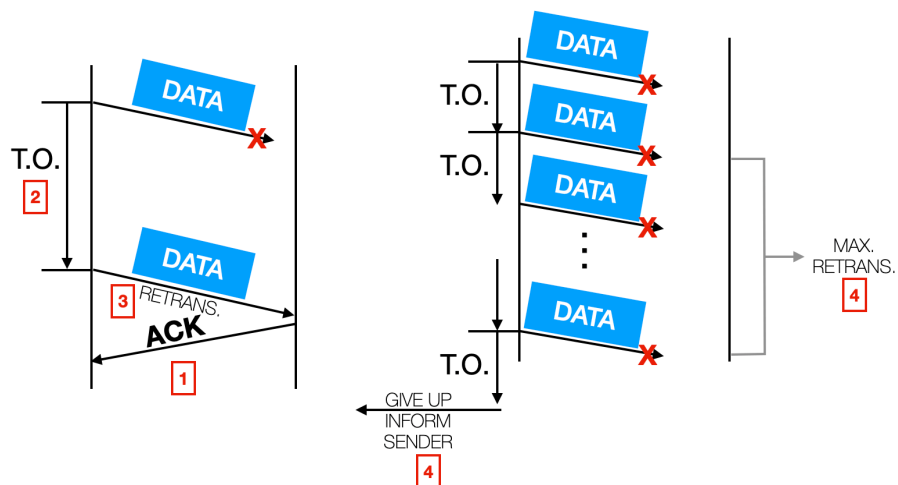
- how long should the sender wait for ACK? what if ACK/data is lost? how does sender know whether the packet is received or not?
- timeout is used to solve the problem. Sender triggers a timer immediately after transmit a packet. If sender does not receive ACK within a certain time (we call it timeout), retransmit the packet.

3. **retransmission**

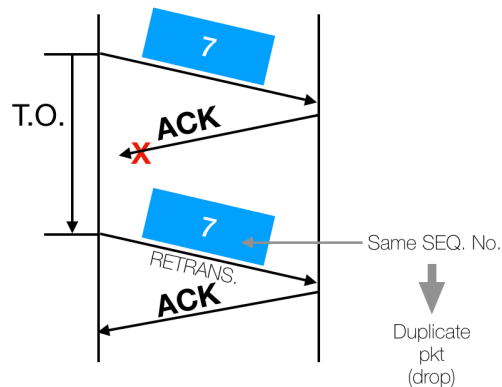
- sender retransmit the current packet if timeout

4. **give up after k retransmissions**

- what if the link is very bad? very lossy? [sender gives up after k retransmissions]
- prevent infinite retransmissions



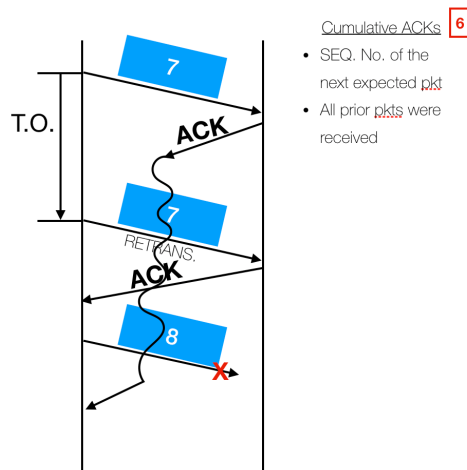
- The above components work well if DATA is lost, shown in the left figure below. What if DATA is received but ACK is lost? The sender will retransmit due to timeout. Then there will be two copies at the receiver side, which is confusing and may lead to bad user experience. The duplicate packet transmission also wastes computing and network resource. How do we solve it?



5 SEQ. No.

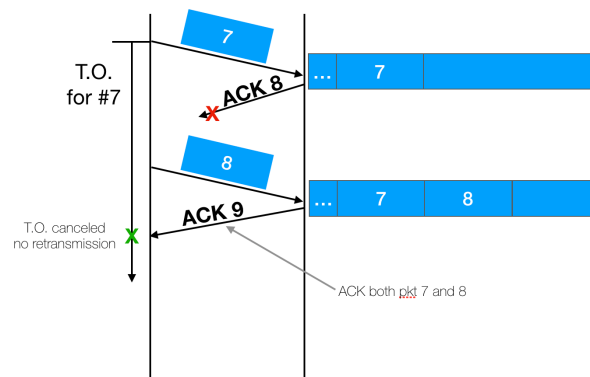
5. sequence numbers on data packets

- handle duplicates happened in retransmission
 - sender adds a unique sequence number to each packet
 - receiver can detect duplicate packets based on their sequence number and discard duplicate packets (shown as the right figure above)
- So far so good. However, it's possible that ACK will be received by the sender out of order. If so, it's hard to tell which packets have been received. As shown in the figure below, it's hard to tell whether the last received ACK is for packet 7 or packet 8, and packet 8 is lost.



6. cumulative acknowledgment numbers

- add sequence numbers on ACKs
- the sequence numbers on ACKs are not the received packet numbers but the next expected packet. This improves the efficiency, as an ACK indicates all prior packets were received even if ACK for a prior received packet is lost, show as the figure below.



Disclaimer: Notes adapted from the textbook and online resources.