The Transport Layer

- The transport layer provides reliable, cost-effective data transport from the source machine to the destination machine, independent of the physical network in use.
- The relationship of the network, transport, and application layers: figure 6-1.
- Why transport layer?

  - The users have no control over the subnet. They cannot solve the problem of poor network service by using better routers or improving lower layer protocols.

  - The existence of the transport layer makes it possible for the transport service to be more reliable than the underlying network service.

Elements of Transport Protocols

- In some ways, transport protocols resemble the data link protocols: both deal with error control, sequencing, flow control, etc.
- There are also differences due to dissimilarities between the operating environments: physical channel vs. subnet.

1. No addressing needed vs. explicit addressing of destination.

2. Simple vs. complex process in establishing a connection.

3. Potential existence of storage capacity in the subnet. ("frame arrive or lost" vs. "packet/TPDU arrive, lost, queued & delayed").

4. Buffering and flow control for a large varying number of connections in the transport layer require a different approach.

Addressing

- When an application process wishes to set up a connection to a remote application process, it must specify which one to connect to.
- Transport address (or TSAP) in the Internet is (IP address, port).
  - TSAP: Transport Service Access Point
  - NSAP: Network SAP (e.g., IP address).
- Figure 6-8 illustrates the relationship between NSAP, TSAP, network connection, and transport connection for a connection-oriented subnet.

Addressing

- How does the user process on host 1 know that the time-of-day server is attached to TSAP 122?
- Possible methods:

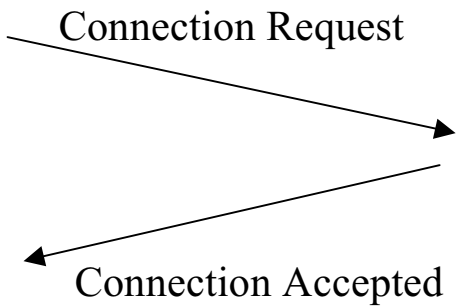
1. Well-known port number.
   -need many TSAP addresses, and many processes active and listening to a stable TSAP.

2. Use a special **process server** that acts as a proxy for less-heavily used servers.
   - Process server listens to a set of ports at the same time.
   - after it gets incoming request, the process server spawns off the requested server, allowing it to inherit the existing connection with the user.

3. Use name server: (for hardware type server which can not created on the fly)
   - a user first query the name server to get the TSAP address of the target server process.

Establishing A Connection

- Sounds easy but actually tricky.
- How about



- What if packets are lost, delayed, or duplicated?
- Worst nightmare: network congestion => time-out => duplicate Connection_Request => multiple connection setup.

Establishing A Connection

- How to handle delayed duplicates?

1. Give each connection a connection identifier

Each transport entity records in a table the released connection's (peer transport entity, connection identifier) pair. Whenever a connection request comes in, it could be checked against the table, to see if it belongs to a previously released connection.

It requires each transport entity to maintain a certain amount of history information indefinitely.

Problem arises when the machine crashes and loses its memory.

Establishing A Connection

- How to handle delayed duplicates?

2. Assume restricted packet lifetime (using hop count, time stamp, or subnet design)

Define T: multiple of packet life time

The packet and its Ack will be dead by T seconds later.

- Each host has a time-of-day clock (never crashes), a binary counter that increments itself at uniform intervals. Clocks at different hosts need not be synchronized.

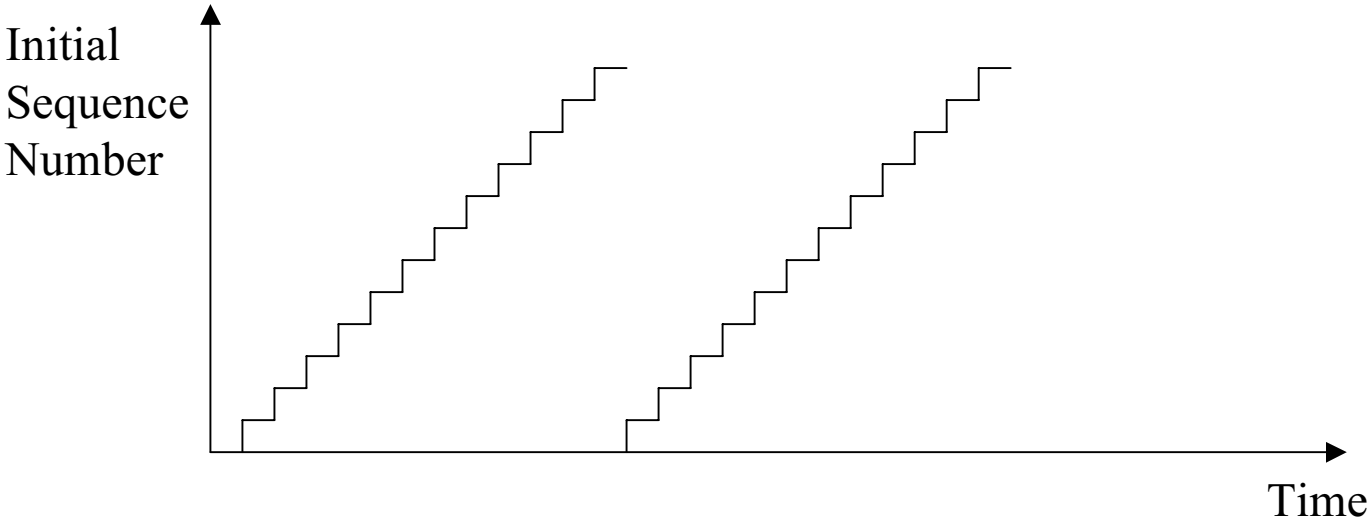
- Basic idea: ensure that two identically numbered TPDUs are never outstanding at the same time.

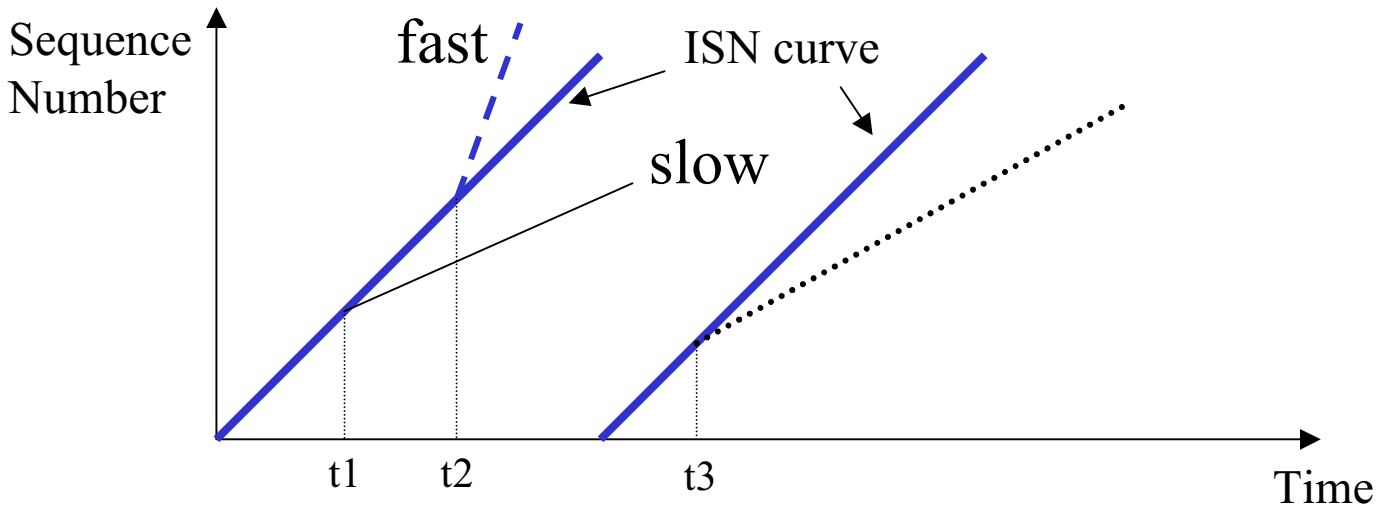
- When a connection is set up, the low-order *k* bits of the clock are used as the initial sequence number.

- Each connection starts numbering its TPDUs with a different sequence number.

- Once both transport entities have agreed on the initial sequence number, any sliding window protocol can be used for data flow control.

Sequence Number





Handle Delayed Duplicates

- When a host crashes and comes up again, its transport entity does not know where it was in the sequence space.

- Solution 1: let transport entity be idle for T seconds after a recovery to let all old TPDUs die off.

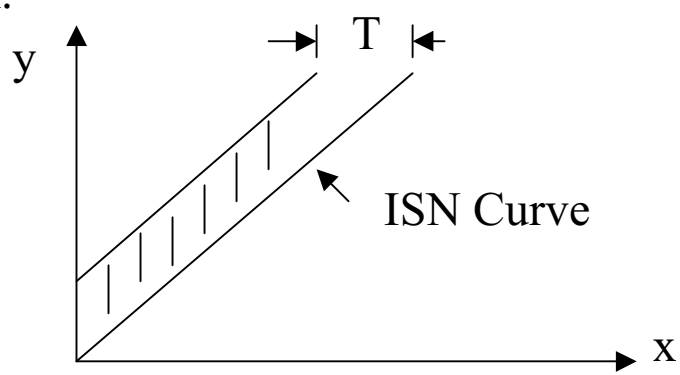
  T may be large for a complex network => unattractive.

- Solution 2: introduce a new restriction on use of sequence number.

  Failure example in the next VG.

  Must prevent sequence number from being used (i.e., reassigned to TPDUs) for a time T before their potential use as initial sequence number.

  The illegal combinations of time and sequence number are shown as the forbidden region.

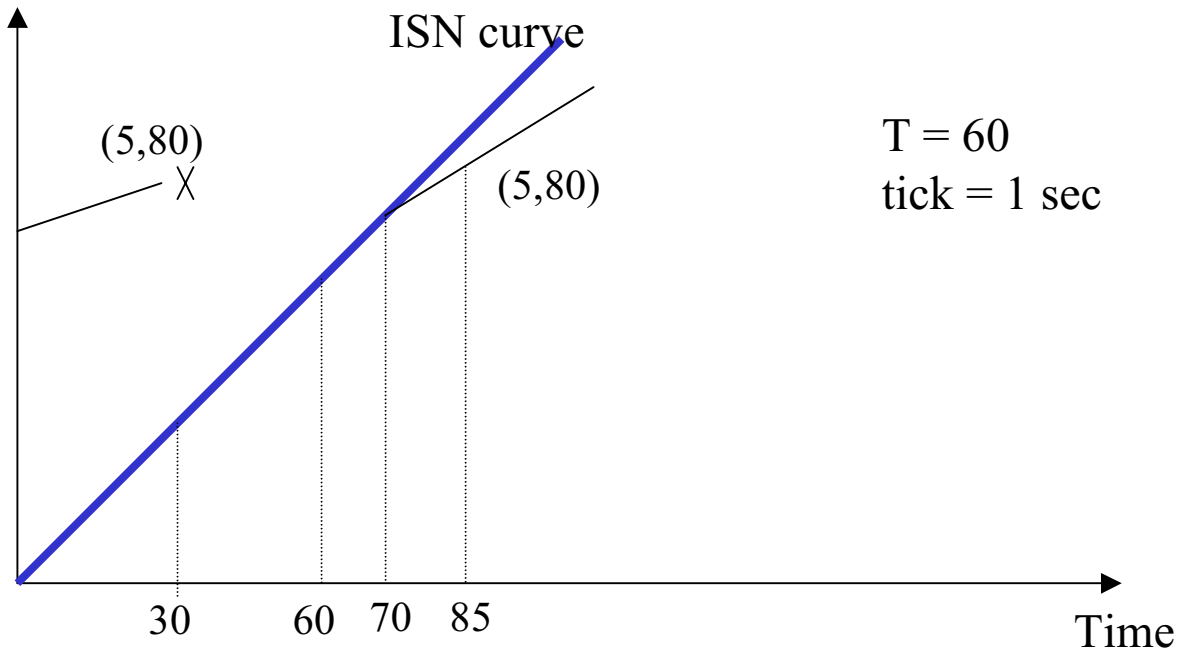


$y > x$

$y < x+T$

Handle Delayed Duplicates

Ex:



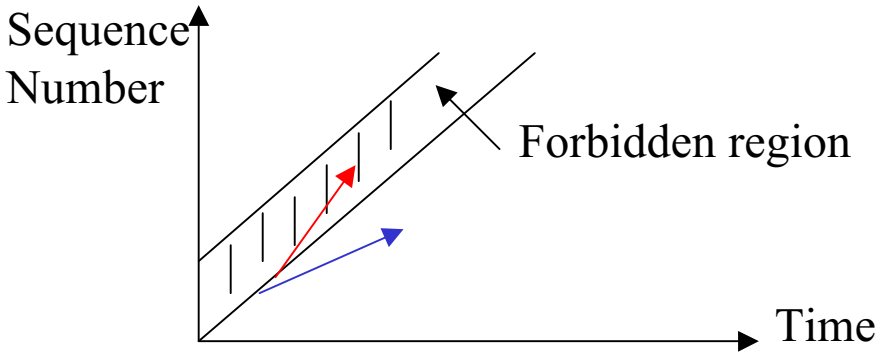
- at t=30, TPDU x is labeled (5,80), then host crashes.
- at t=60, host restarts, open connection 0-4
- at t=70 host reopen connection 5, ISN=70

  at t=85, TPDU (5,80) is sent to the network

- if TPDU x arrives at the receiver before new TPDU (5,80), x will be accepted and correct (5,80) will be rejected as a duplicate.

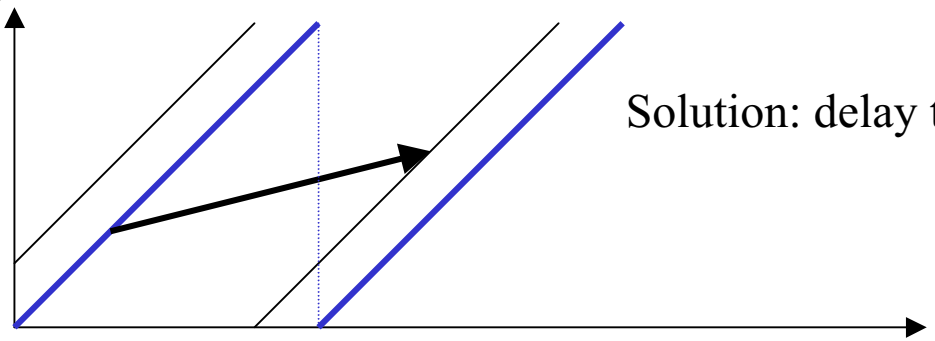
Handle Delayed Duplicates

- Before sending any TPDU on any connection, the transport entity must read the clock and check to see that it is not in the forbidden region.



- Consequence: maximum data rate is one TPDU/tick. To allow higher data rate, use short tick (msec).

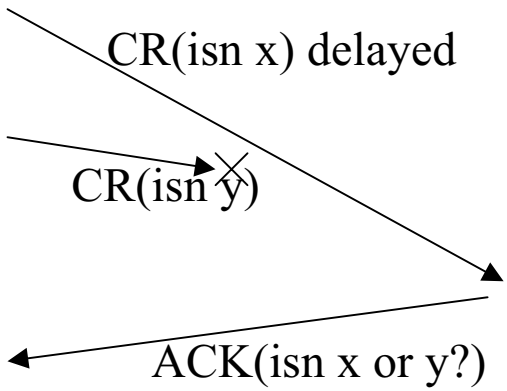
- Sending at data rate lower than the clock rate can also run into the forbidden region from the left.



Solution: delay the TPDU for T seconds

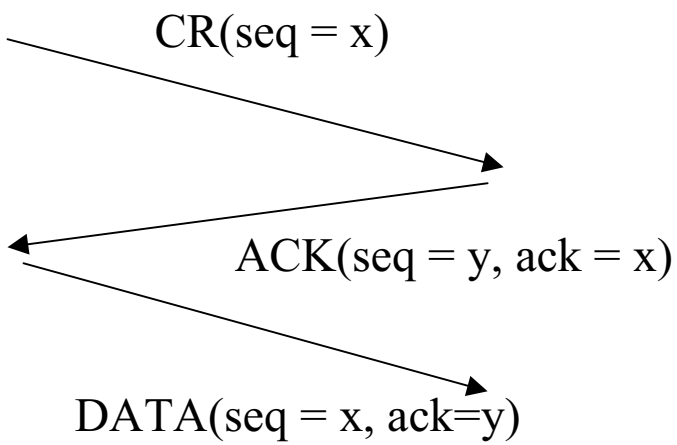
Establishing A Connection

- The clock-based method solves the delayed duplicate problem for DATA TPDUs.
- For this method to be useful, a connection must first be established (via control TPDUs) for both sides to agree on initial sequence number.
- Control TPDUs may also be delayed.



Three-Way Handshake

- Figure 6-11.
- Key point: need “sequence number” in CR and ACK.
- There is no combination of old conn_req, conn_acpt, or other TPDUs that can cause the protocol to fail and have a connection set up by accident. (How about bit arrors?)



Releasing A Connection

- Two styles of terminating a connection:
  - Asymmetric release: one party hang up, connection is broken.
  - Symmetric release: requires both ends to release.
- Asymmetric release may result in data loss (fig 6-12).
- Symmetric release does the job when each process has fixed data to send and knows clearly when it has sent it.
- In other situations, determining that all work has been done and the connection should be terminated is not so obvious.
- Ex: “I am done. Are you done too?” “I am done too. Good bye”. => release the connection
- Two-Army problem (fig 6-13).

Two Army Problem

B1,B1 B2,B2

WWW

- The blue armies want to synchronize their attacks.

  - B1 sends messengers to deliver message “we attack at dawn, March 29”; waiting for ACK.

  - B2 receives message and sends ACK “Got it. Will attack at dawn, March 29.”; waiting for ACK.

  - B1 receives ACK; needs to send ACK’s ACK to B2.

  - ….

- No protocol exists that solves this problem.
- This problem is different from the three-way handshake for connection establishment since conn-establish is always initiated by one side. It does not need both side to act simultaneously.

Releasing A Connection

- Releasing using three-way handshake is adequate; since both sides need to terminate exactly at the same time.
- Fig 6-14.
- If intial DR and N retransmissions are all lost => sender gives up and releases => receiver still active => half open.

A. by not allowing sender to give up

- if receiver side can time-out, sender will go on forever,

  no response will return after receiver close.

- if receiver side can not time-out, protocol hangs.

B. If no TPDUs have arrived for a certain number of seconds, the connection is automatically disconnected.

- need timer mechanism

- need keep-alive TPDUs for idle duration.

Flow Control and Buffering

- Similar to data link layer: use sliding window, …
- Difference: a host has many connections, more than a router has => impractical to implement the data link buffering strategy in the transport layer.
- If the network service is unreliable, the sender must buffer all TPDUs sent.
- The optimum trade-off between source buffering and destination buffering depends on the type of traffic carried by the connection.
- For low-BW bursty traffic, the sender retains a copy of the TPDU until it is ACKed.
- For file transfer and other high-BW traffic, better to buffer at the receiver.

Flow Control and Buffering

- As connections are opened and closed, and as traffic pattern changes, the sender and reciver need to dynamically adjust their buffer allocation.
- A reasonably general way to manage dynamic buffer allocation is to decouple the buffering from the ACK.
- The receiver separately piggybacks both ACK and buffer allocation onto the reverse traffic.
- Fig 6-16, a potential problem.
- Solution: periodically, host sends control TPDUs giving the ACK and buffer status on each connection.

Flow Control and Buffering

- Besides buffer space, which imposes a limit on the sender‘s data rate, the carrying capacity of the subnet can also limit the data rate.
- If the network can handle C TPDUs/sec, and the cycle (RT) time (transmission, propagation, queueing, processing time at the receiver, and return of ACK) is r, then the sender‘s window should be Cr. (to avoid too many TPDUs unAcked?)
- The carrying capacity can be determined by counting the number of TPDUs acked during some time period and then dividing it by the time period.
- The RTT can be measured exactly and a running mean maintained.

Multiplexing

- Multiplexing several conversations onto one connection.

  - Save table space in router and buffer sapce
  - reduce connection bill.

- Inverse Multiplexing to get high bandwidth.

Crash Recovery

- Router crash is easier to handle since transport entities are alive at the host.
- How to recover from host crashes?

- Assume stop-and-wait is adopted.

- Server crashes and comes up => table reinitiated => does not know where it was.

- Sender sends a broadcast TPDU to all other hosts, requesting its clients to inform it the status of the open connections.

- Each client can be either in S1: one TPDU outstanding

  or in S0: no TPDU outstanding

- possible solution: client retransmits only if it is in S1.

- scenario: server needs to “send ack”(A), “write to application process”(W)

Crash Recovery

- A Crash (W) => client (in S0) will not retransmit => missing TPDU.
- W Crash (A) => client (in S1) will retransmit => undetected duplicate TPDU.
- No matter how the sender and receiver are programmed, there are always situations where the protocol fails to recover properly.
- Fig 6-18, 8 combinations.
- Recovery from a layer N crash can only be done by layer N+1, and only if high layer retains enough status information.
- A truly end-to-end ACK, whose receipt means that the work has actually been done, and lack thereof means that it has not, is probably impossible to achieve.