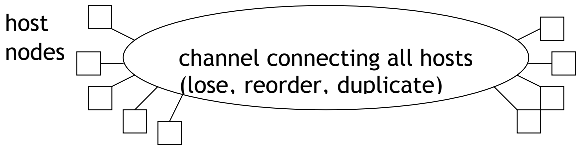
Computer and Network Security
CMSC 414 Spring 2011

Udaya Shankar
shankar@cs.umd.edu

INTERNET CONTEXT

Internet Context (without security modifications)

- The original Internet consists of host nodes attached to a channel.

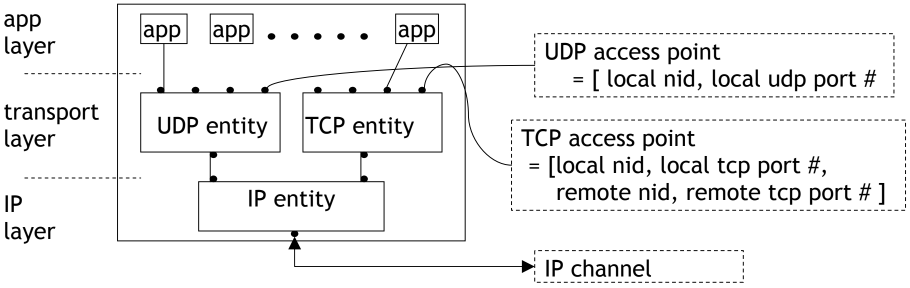


host nodes

channel connecting all hosts (lose, reorder, duplicate)

- Every node has an IP address (network id or *nid* for short), which identifies the node to the rest of the Internet.
- The channel allows any node x to send IP packets to any other node y. All that node x needs to know is node y's IP address, which goes in the packet header.
- Messages sent from node x to node y are received subject to loss, reordering, and duplication.
- Below, *internet id* refers to an id understood by Internet protocols (IP, UDP, TCP, etc), in contrast to application-specific ids.

Structure of a host node



UDP access point
= [local nid, local udp port #

TCP access point
= [local nid, local tcp port #,
remote nid, remote tcp port #]

IP channel

- IP entity: internet id = nid (= 32-bit IP address in IPv4)
- UDP entity: internet id = [local nid, protocol id (= 'UDP')]
- TCP entity: internet id = [local nid , protocol id (= 'TCP')]
- UDP and TCP use (16-bit) port numbers
- App entity: internet id = access point to which it is attached.
  - UPD access point = [protocol id, local nid, local port#]
  - TCP access point = [protocol id, local nid, local port#
    remote nid, remote port#]
  - Can also have application-specific ids: 'Alice', 'Bob', URL, etc. Recognizable only by appropriate "peer" apps.

Packet structure

- IP packet: [header, payload]
  - header = [sndr nid, dst nid, protocol id]
  - payload = transport protocol message (UDP, TCP, IP, …)
- Transport protocol message:
  - UDP: [sndr port#, rcvr port#, application message]
  - TCP: [sndr port#, rcvr port#, application message]
  - IP: [sndr port#, rcvr port#, application message]
- The term "UDP packet header" usually refers the IP and UDP headers:
  - So UDP packet: [header, app message]
    header = [protocol id, sndr nid, sndr port#, rcvr nid, rcvr port#]
- Similarly for "TCP packet"
  - TCP packet: [header, app message]
  - header = [protocol id, sndr nid, sndr port#, rcvr nid, rcvr port#]

UDP sockets interface

- App can *attach* as server, supplying local port#
  - UDP entity attaches app to local port#
    (fails if local port# is already attached).
  - app's local nid, local port# are fixed at this point.
- App can *attach* as client, supplying remote nid and remote port#
  - UDP entity attaches app to any unattached local port
    (fails if no local port is unattached).
  - app's local nid, local port#, remote nid, remote port# fixed at this point.
- An attached client app can do a *send*, supplying data.
  - UDP entity constructs and sends UDP packet with data as app message and local nid/port#, remote nid/port# in header.
- An attached server app can do a *send*, supplying data and remote nid/port#.
  - UDP entity constructs and sends packet with data in as app message and local nid/port#, remote nid/port# in header.
- An attached app can do a *receive*.
  Blocks until UDP entity gets an incoming packet for local nid/port#,
  at which point the call returns and the packet is passed to the app.
- UDP packets are subject to loss, reordering, duplication.
- An attached app can *detach*, freeing up its local port#.

TCP sockets interface - 1

Attaching as a client

- App attaches as client by doing a *connect*, supplying remote nid/port#.
  - Blocking call that returns with "accept" or "reject".
    - accept: connection established with remote app.
    - reject: connection attempt not successful; app is detached.
  - TCP entity does the following in this call:
    - associates an unattached local port# to the app
    - attaches app to access point [local nid/port#, remote nid/port#]
    - (re)sends connect message to remote node
      until response message ("accept" or "reject") is received,
      at which point the call returns.

TCP sockets interface – 2

Attaching as a server

- App attaches as server by doing a *listen*, supplying local port#.
  - Blocking call that returns with an incoming connect request
    (containing requesting client's nid and port#).
  - TCP entity does the following in this call:
    - attaches app to access point [local nid/port#, remote null]
    - waits for incoming connect request such that
      rcvr nid/port# = local nid/port# and
      access point [rcvr nid/local port#, sndr nid/port#] is unattached
    - passes this incoming connect request back in call return.

At this point, the app can do one of three things:

1. Reject the connect request (for whatever reason) and go back to listening.
2. Assign another app (e.g., another thread) to accept the connect request, and go back to listening.
3. Accept the connect request itself
   (If another connect request arrives, the TCP entity rejects it.)

TCP sockets interface – 3

Accepting a connect request

- App does an *accept*, supplying remote nid/port# being accepted.
  - This call behaves very much like the *connect* call
    (because TCP entity ensures that the connect request is current)
  - Blocking call that returns with "ack" or "reject":
    - ack: connection established with remote app.
    - reject: connection attempt not successful; app can go back to listening
  - TCP entity does the following in this call:
    - attaches app to access point [local nid/port#, remote nid/port#]
    - (re)sends accept message to remote node
      until response message ("accept ack" or "reject") is received.

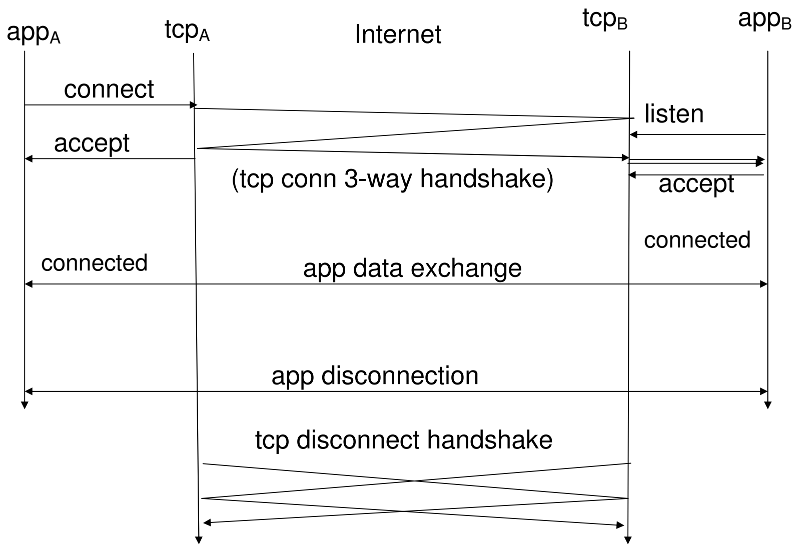
- If *accept* returns with ack, the app is now connected to the client.
- If *accept* returns with reject, the app is not connected to the client.
  - If the app was previously listening, it can go back to listening
  - If the app was created to accept this specific connect request,
    it can now peacefully die.

TCP sockets interface – 4

Connected app

- A connected app can do *sends*, supplying data with each send.
- A connected app can do *receives*, receiving data with call.
  - a receive call is blocking (as with UDP).
- A connected app can do a *disconnect*.
  - after this app cannot send data but continues to receive data.
  - The app is terminated only after remote also does a disconnect
- A connected app can always *abort*.

Example application session over TCP


app$_A$
tcp$_A$
Internet
tcp$_B$
app$_B$
connect
listen
accept
(tcp conn 3-way handshake)
accept
connected
connected
app data exchange
app disconnection
tcp disconnect handshake