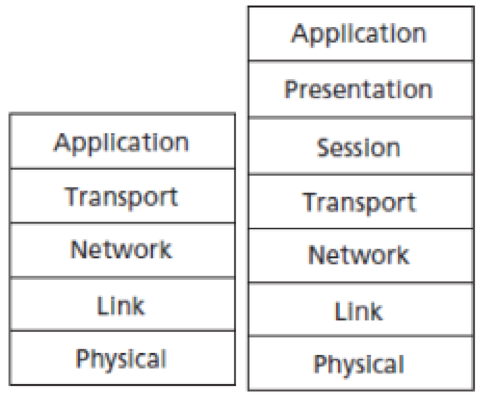
CIS 457 - Data Communications

Nathan Bowman

Images taken from Kurose and Ross book

Transport Layer



a. Five-layer Internet protocol stack

b. Seven-layer ISO OSI reference model

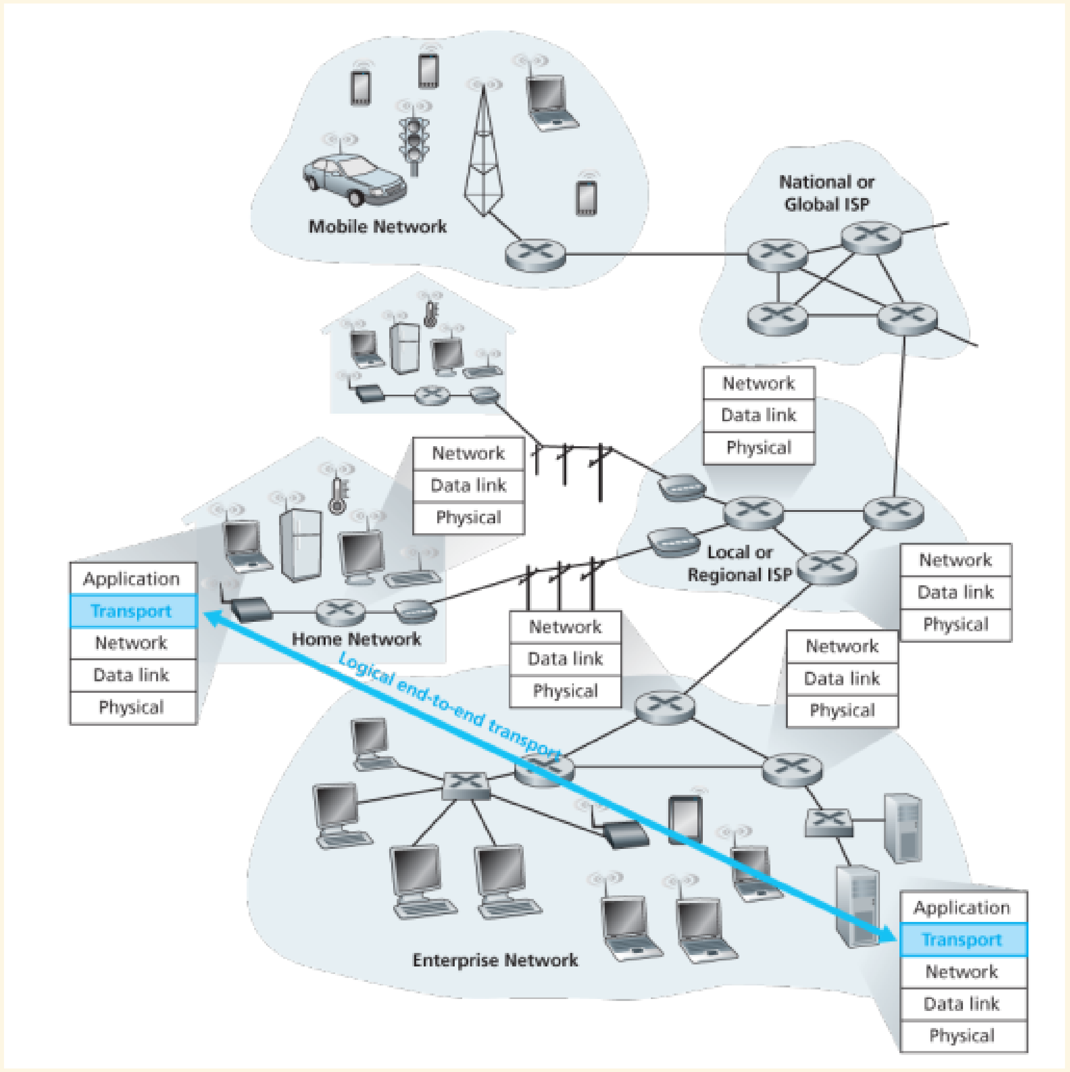
Goal of transport layer is to move messages between processes

Sits between application layer and network layer

Relies on network layer to transport between hosts

Provides **logical communication** -- communication abstracted from details of underlying network

Two processes can act as though they are directly connected, despite existence of complicated internetwork of links and switches


Mobile Network
National or Global ISP
Network
Data link
Physical
Network
Data link
Physical
Application
Transport
Network
Data link
Physical
Local or Regional ISP
Network
Data link
Physical
Home Network
Network
Data link
Physical
Network
Data link
Physical
Logical end-to-end transport
Enterprise Network
Application
Transport
Network
Data link
Physical

Transport layer implemented in end systems only

Network infrastructure (packet switches) knows nothing about transport layer -- simply see it as payload to be delivered

Application message embedded in transport-layer packets called **segments**

Message may be broken down into smaller parts first, then transport-layer header information added to create segment

Transport-layer segment passed to network layer, where further header information is added to create network-layer packet called **datagram**

Routers on path only ever examine network-layer headers

Entire process reversed at receiving host -- header information from network and transport layers removed and message reassembled

Important to know a little about network layer that transport layer depends on

Internet protocol (IP) provides **best-effort delivery**

This means packets may arrive out of order, corrupted, or not arrive at all to destination host

Each host on internet has unique address -- **IP address** (and may have more than one)

Because transport layer relies on network layer, services provided by transport layer limited by underlying network

If network offers no maximum-delay or minimum-bandwidth guarantees for communication between hosts, impossible for transport layer to provide those guarantees between processes

However, some additional services *can* be added by transport layer (otherwise, why would it exist?)

Nice-to-have features transport layer can provide on unreliable network include

- reliable delivery
- security (encryption)

Primary reason for transport layer, though, is **multiplexing** and **demultiplexing** -- allowing many processes to communicate over single network interface

If each host could run exactly one networked application, transport layer would not be necessary

Sockets, as we have seen, are abstract connection between application and transport layers

Transport layer responsible for delivering to particular socket, which is then read by application

Possible for process to have more than one socket

Need some unique way of identifying individual socket

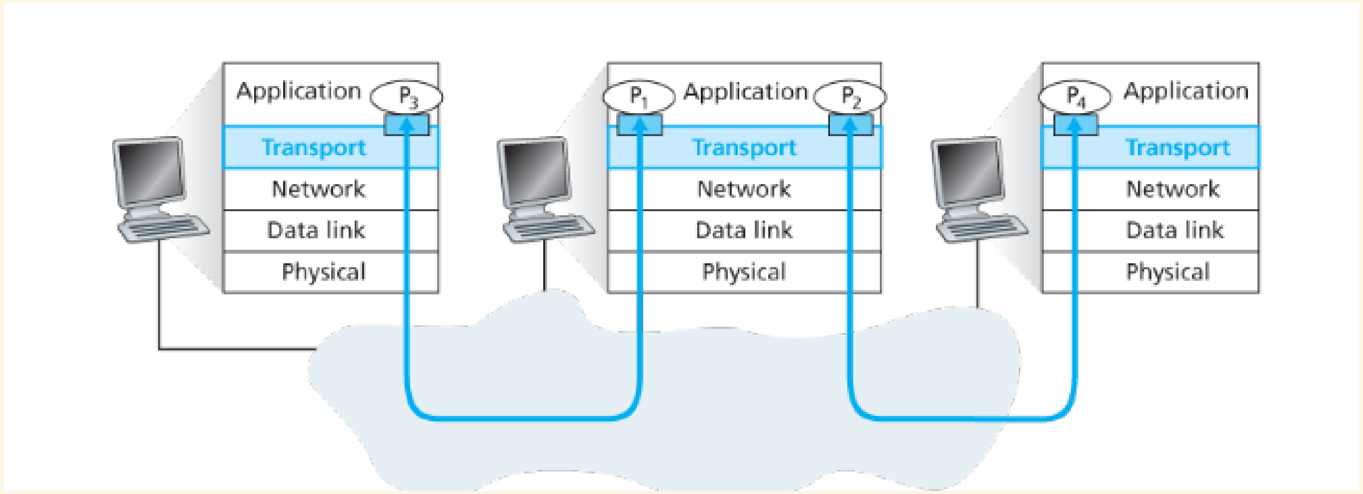
Information to specify socket must be carried by segment -- no other layer needs this information

Using this information to send packet to correct socket is **demultiplexing**

Transport layer at sending host must

- gather messags from various processes
- add header information to be used later to demultiplex
- pass segments to network layer

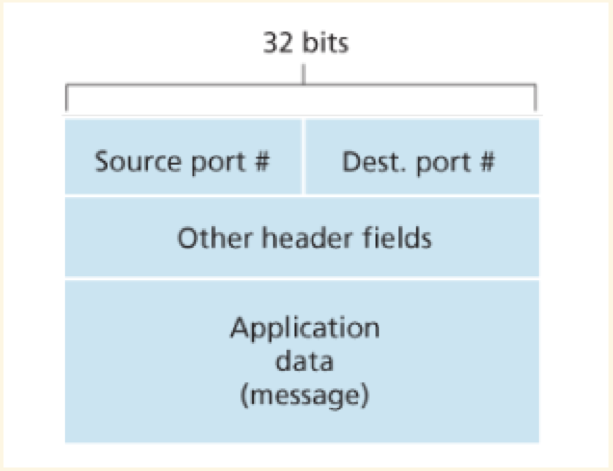
This is **multiplexing**


Application
P_3
Transport
Network
Data link
Physical
P_1
Application
P_2
Transport
Network
Data link
Physical
P_4
Application
Transport
Network
Data link
Physical

As mentioned, sockets need unique IDs, and segments must include information to determine these IDs in segment header

Each host already has unique IP address

Additional **port** numbers used to help identify sockets on hosts


32 bits
Source port #
Dest. port #
Other header fields
Application
data
(message)

Port is 16-bit number, ranges from 0 - 65535

Ports from 0 - 1023 are **well-known ports**

These are registered with IANA and meant to be used for specific protocols

For instance, HTTP servers listen on port 80

Method of multiplexing depends on whether transport-layer service is connection-oriented (TCP) or not connection-oriented (UDP)

In UDP, (destination IP, destination port) pair is enough to specify socket

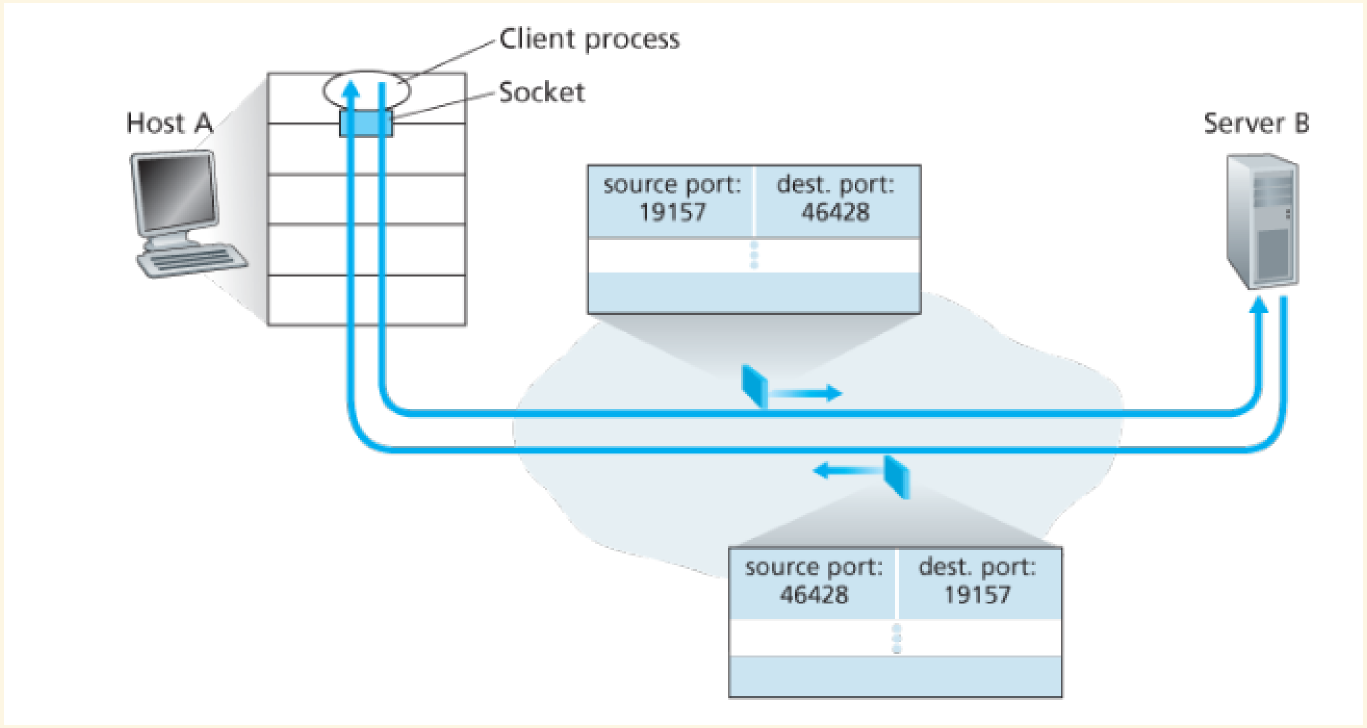
TCP is a bit more complicated

Server application using UDP binds itself to specific port, creating socket to listen

Any message sent over UDP to that IP and port is sent to that socket

Source IP and source port unimportant except to know where to send reply

Several sources from various places on internet can send to same (IP, port) pair over UDP and will reach same socket


Client process
Socket
Host A
Server B
source port: 19157
dest. port: 46428
source port: 46428
dest. port: 19157

For connection-oriented (de)multiplexing (TCP, in the case of the internet), `(dest IP, dest port)` not enough to specify socket

Saw in socket programming example that each incoming connection gets new socket

Many clients can be connected to server on same port, so need some way to distinguish between sockets belonging to various connections

Sockets on TCP identified by 4-tuple

```
(source IP, source port, dest IP, dest port)
```

That is enough to distinguish between connections, so it is enough to specify socket

In TCP, if clients throughout internet send message to same (IP, port) pair, each will be contacting *different* socket

Note that protocol also matters in specifying socket

TCP socket listenting on port 80 is different from UDP socket listening on port 80

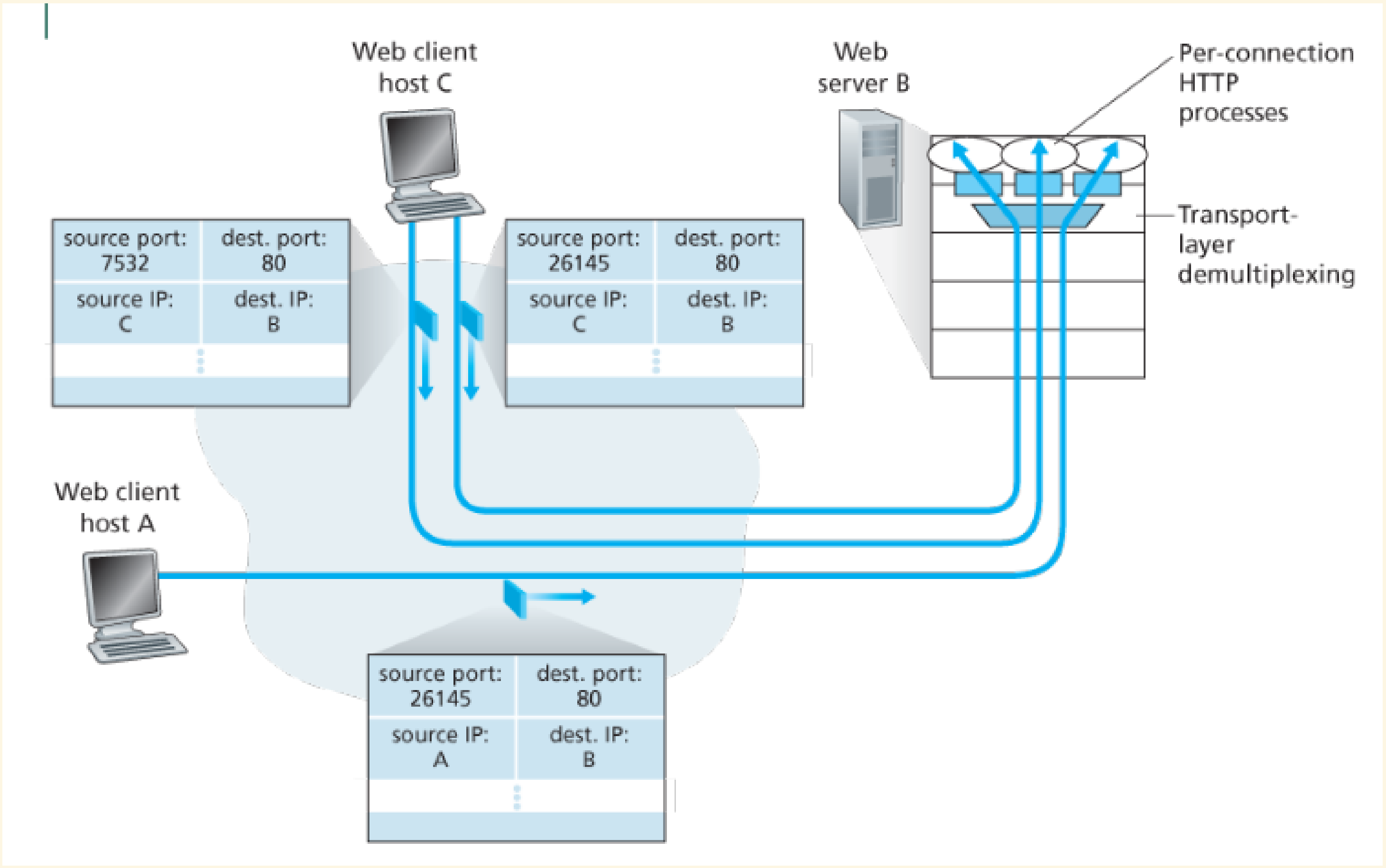
Some sources will say that TCP socket is 5-tuple, not 4-tuple, with additional entry being protocol -- for example:

```
(TCP, 192.168.3.1, 80, 192.168.3.55, 4023)
```

To establish connections, use "welcome" socket, as seen in programming examples

Remote host contacts welcome socket, which creates new socket corresponding to 4-tuple of destination information and source information for particular client

All future packets from that client automatically delivered to this new socket until connection is closed


Web client host C
Web server B
Per-connection HTTP processes
Transport-layer demultiplexing
source port: 7532
dest. port: 80
source IP: C
dest. IP: B
source port: 26145
dest. port: 80
source IP: C
dest. IP: B
Web client host A
source port: 26145
dest. port: 80
source IP: A
dest. IP: B