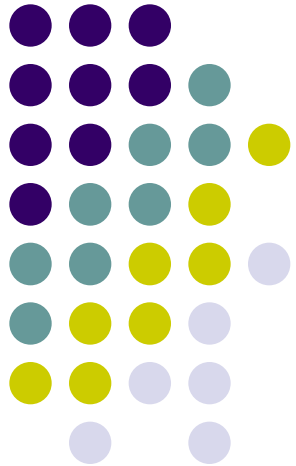
ELEN 4017

Network Fundamentals

Lecture 13 & 14

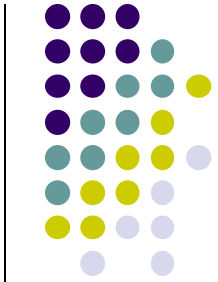


Chapter 3: Transport Layer

Our goals:

- understand principles behind transport layer services:
  - multiplexing/demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- learn about transport layer protocols in the Internet:
  - UDP: connectionless transport
  - TCP: connection-oriented transport
  - TCP congestion control

Purpose of lecture

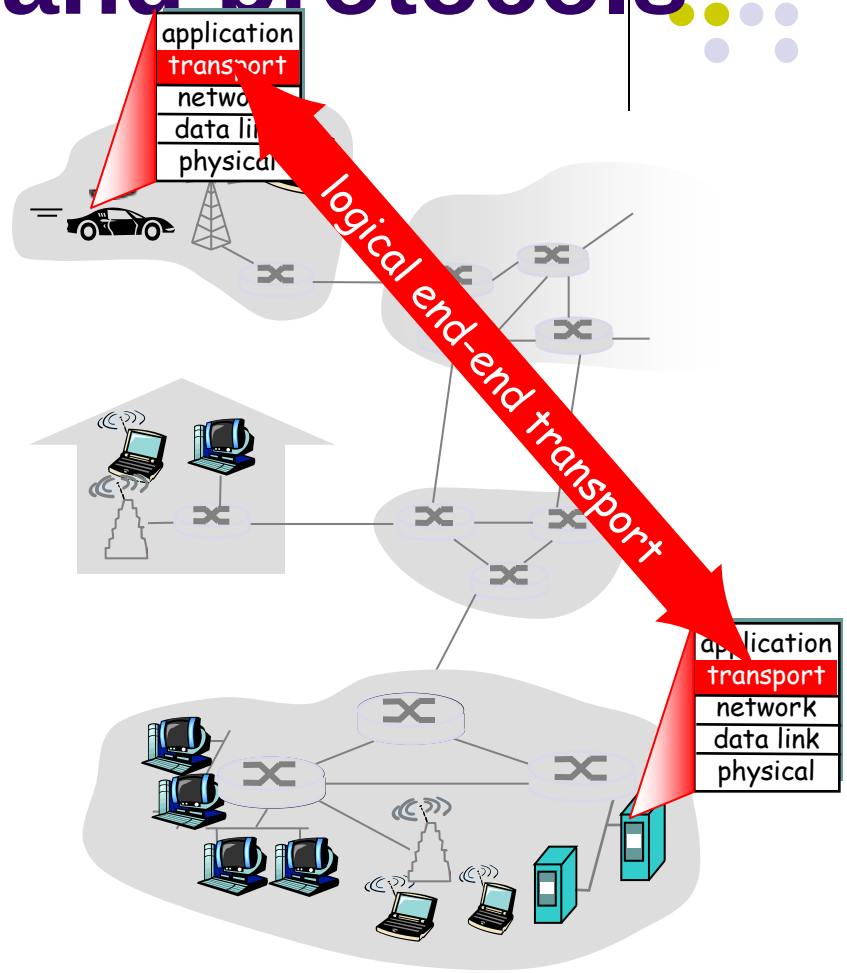


Chapter 3: Transport Layer

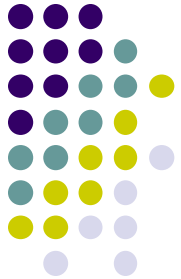
- Transport-layer services
- Multiplexing and demultiplexing
- TCP vs UDP

Transport services and protocols

- provide *logical communication* between app processes running on different hosts
- transport protocols run in end systems
  - send side: breaks app messages into segments, passes to network layer
  - rcv side: reassembles segments into messages, passes to app layer
- more than one transport protocol available to apps
  - Internet: TCP and UDP



Transport vs. network layer



- *network layer:* logical communication between hosts
- *transport layer:* logical communication between processes
  - relies on, enhances, network layer services

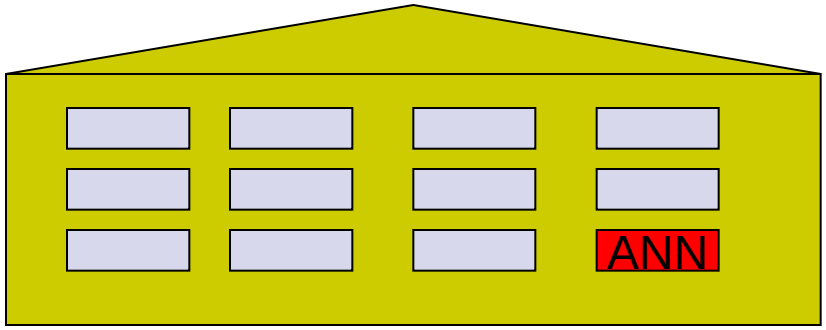
Household analogy

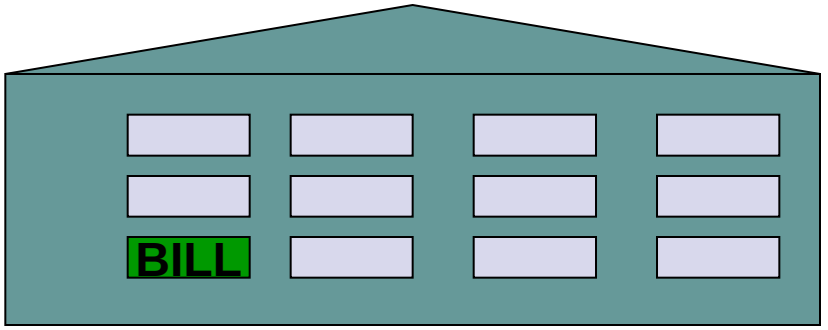
- Consider 2 houses – Jhb and Cape Town
- Each house is home to 12 kids.
- Kids in Jhb are cousins of kids in Cape Town.
- Every week each kid writes a letter to each cousin in the other house. Letters are sent via normal postal service.
- In each house there is 1 kid who is responsible for collecting the letters for postage and distributing the letters on arrival (Ann in Jhb and Bill in Cape Town)

Household analogy:

12 kids sending letters to 12 kids

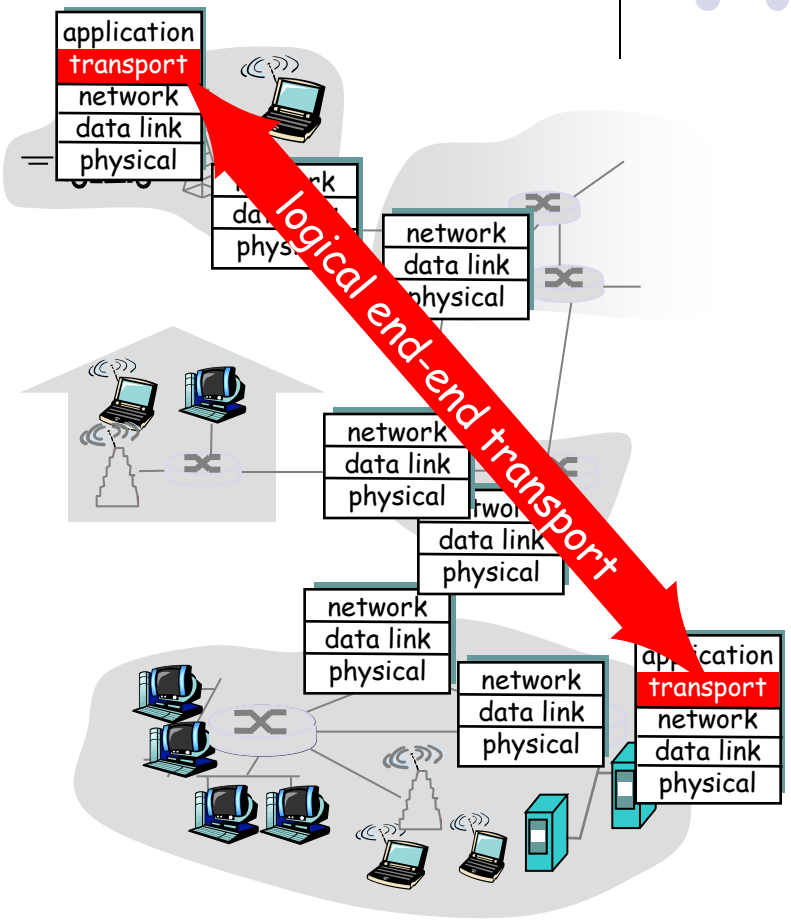
- processes = kids
- app messages = letters in envelopes
- hosts = houses
- transport protocol = Ann and Bill
- network-layer protocol = postal service



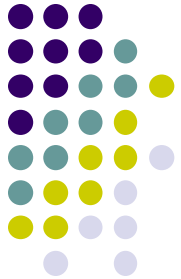


Internet transport-layer protocols

- reliable, in-order delivery (TCP)
  - congestion control
  - flow control
  - connection setup
- unreliable, unordered delivery: UDP
  - no-frills extension of “best-effort” IP
- services not available:
  - delay guarantees
  - bandwidth guarantees

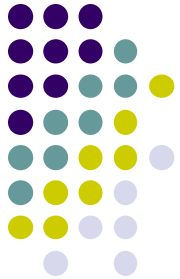


Characteristics of Internet protocol



- Logical communication between hosts
- Best-effort delivery service → no guarantees.
- Does not guarantee:
  - Segment delivery
  - Orderly delivery
  - Data integrity
- Thus IP referred to as an unreliable service.

Services provided by transport layer (TCP and UDP)

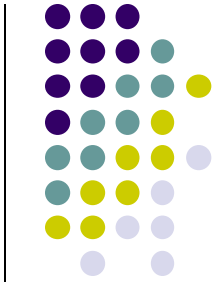


- Extending host-to-host delivery to process-to-process delivery → called transport-layer multiplexing.
- Integrity checking (error detection)
- These are the only services that UDP provides, thus UDP also considered to be unreliable.
- TCP provides:
  - Reliable data transfer
  - Flow control
  - Sequence numbers
  - Acknowledgements
  - Timers
  - Congestion control

Purpose of lecture

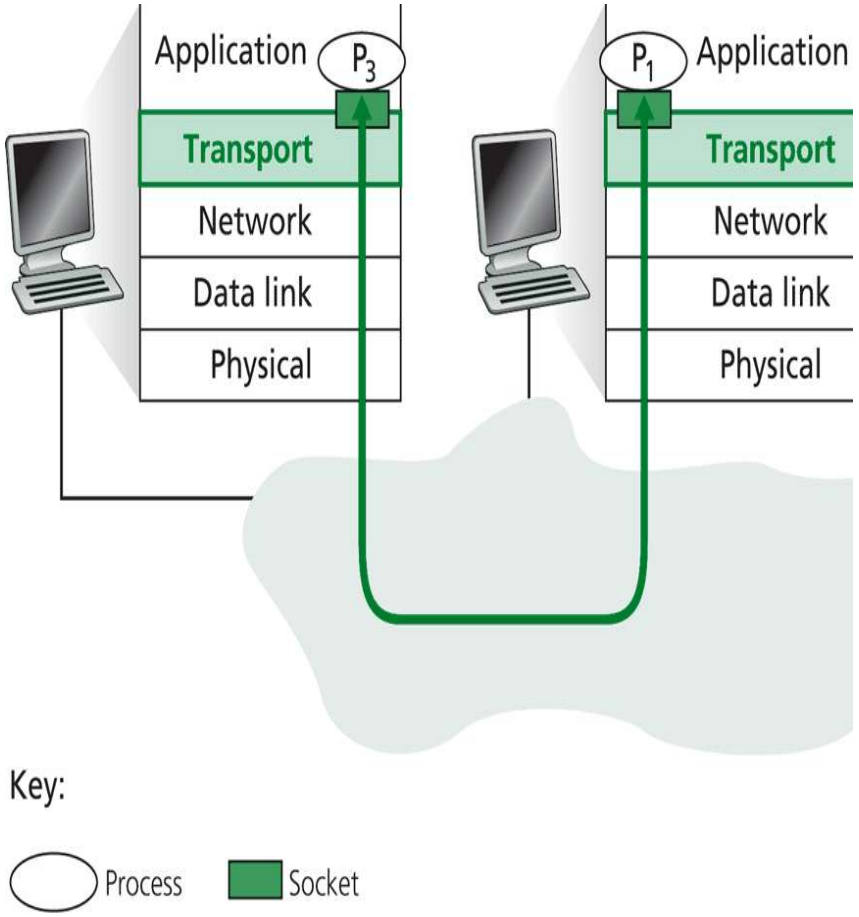
Chapter 3: Transport Layer

- Transport-layer services
- Multiplexing and demultiplexing



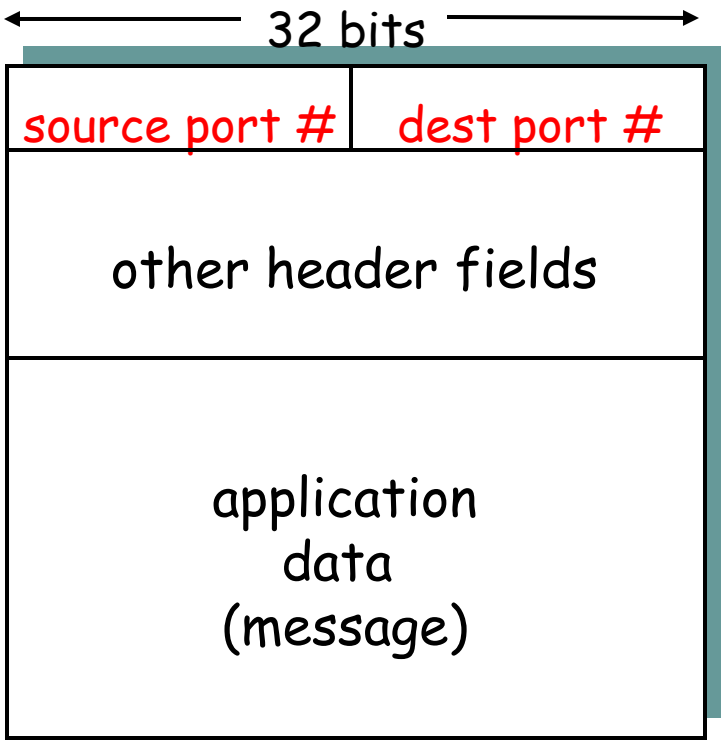
Multiplexing and de-multiplexing

- Single user has the following active sessions:
  - 1 FTP
  - 2 Telnet
  - 1 HTTP (Web browser)
- Thus 4 network application processes running.
- Encapsulation of application data from different processes into segments and passing segments to network layer is called multiplexing.
- Transport layer must deliver data to appropriate process (de-multiplexing).



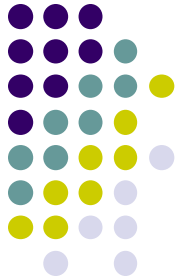
How demultiplexing works

- host receives IP datagrams
  - each datagram has source IP address, destination IP address
  - each datagram carries 1 transport-layer segment
  - each segment has source, destination port number
- host uses IP addresses & port numbers to direct segment to appropriate socket



TCP/UDP segment format

Requirements for multiplexing /de-multiplexing

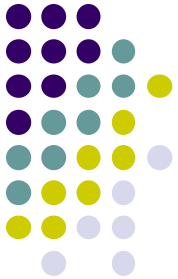


- Sockets must be uniquely identified
- Each segment contains the source port number and destination port number.
- Port number is 16 bit, first 1024 values are well-known (reserved by well-known application protocols)
- Typically the client port number is assigned dynamically and transparently to the application, whilst the server port number is specific.

Processes/sockets and ports

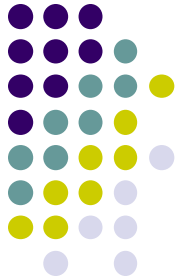
- Process is program running.
- A process can only communicate via a doorway – a socket.
- The socket is identified by:
  - IP address
  - Port Number
- Connection oriented transport requires a socket per connection → source and destination info is used.

UDP vs TCP



- TCP offers the following over UDP:
  - Connection oriented
  - Flow control
  - Orderly delivery (➔ sequence numbers for packets)
- Thus for TCP to achieve this, each **TCP connection** is assigned its own socket.
- To find the correct socket to enter, look at source IP & port, as well as destination IP and port.

TCP sockets – client side

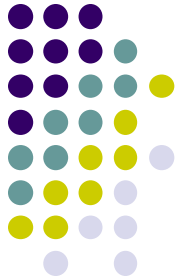


- If you open many TCP connections, then each connection **on client**, will use a different port number.

Socket 1:
SourceIP: ClientIP
Source Port: 100

Socket 2:
SourceIP: ClientIP
Source Port: 101

TCP sockets – server side

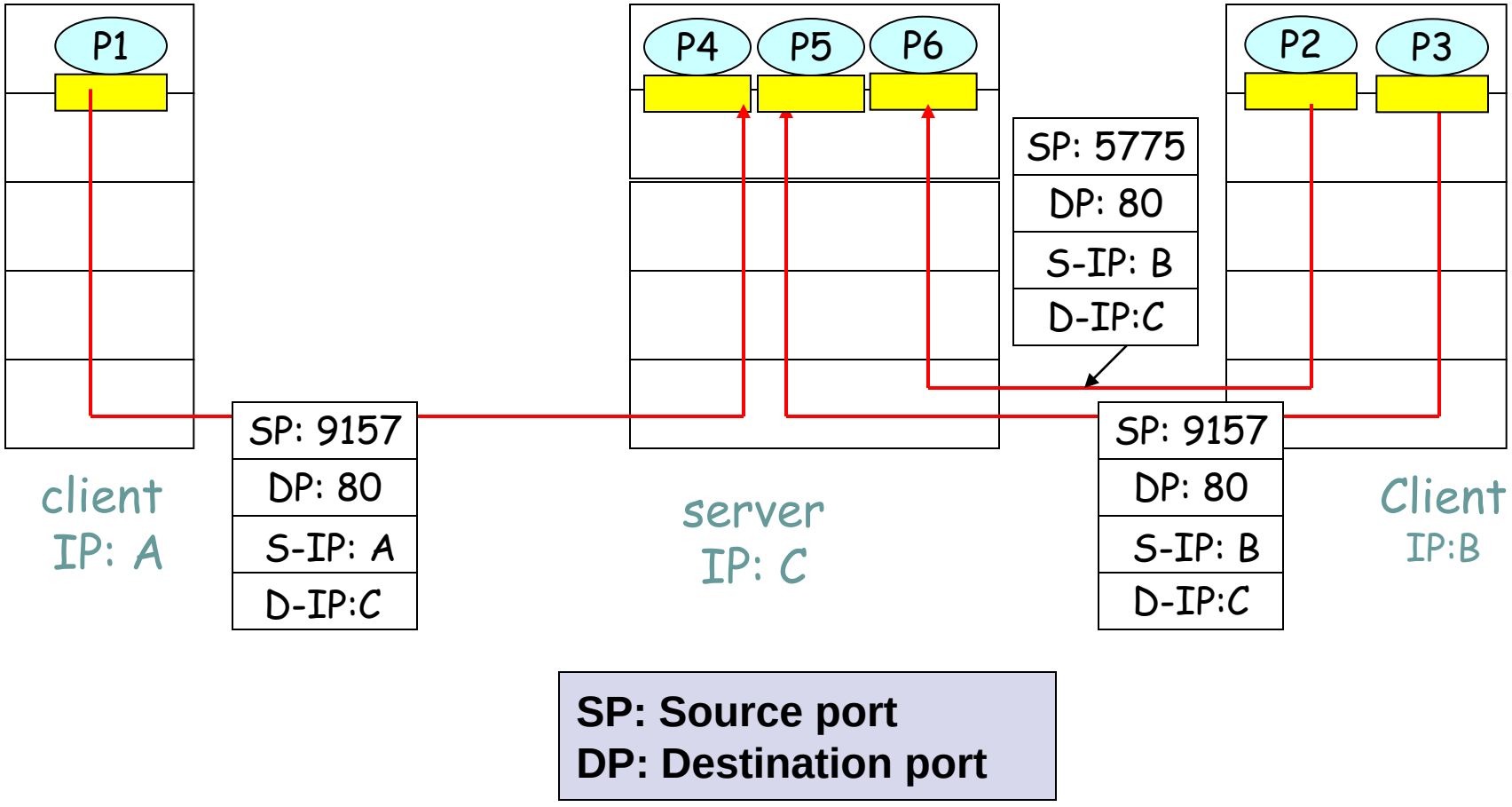


- On the server, all the requests will be for a specific destination port e.g. port 80 for HTTP traffic.
- Thus the server, will maintain a **unique socket for each connection**. Why?
- This socket will be identified by:
  - Source IP, Source port
  - Destination IP, Destination port

Socket 1:
SourceIP: ClientIP
Source Port: 100
DestIP:ServerIP
DestPort:80

Socket 2:
SourceIP: ClientIP
Source Port: 101
DestIP:ServerIP
DestPort:80

TCP – 1 socket per connection



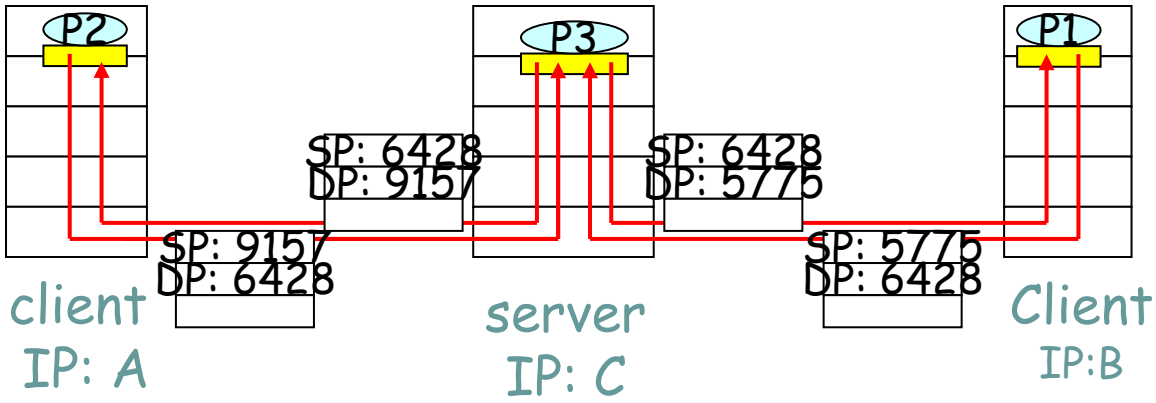
SP: Source port
DP: Destination port

UDP

- UDP is connectionless. Thus there is no need for the server to maintain separate sockets for each incoming user.

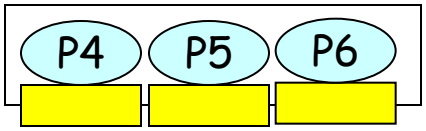
UDP Socket

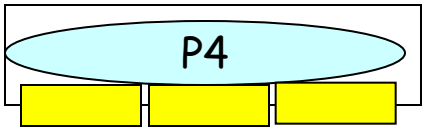
Destination IP
Destination Port



Linking sockets to processes

- Here there is no rule.
- **Generally, 1 socket per process**.
- However for better performance e.g. web servers, you could have multiple sockets linking to 1 process.





UDP: User Datagram Protocol [RFC 768]

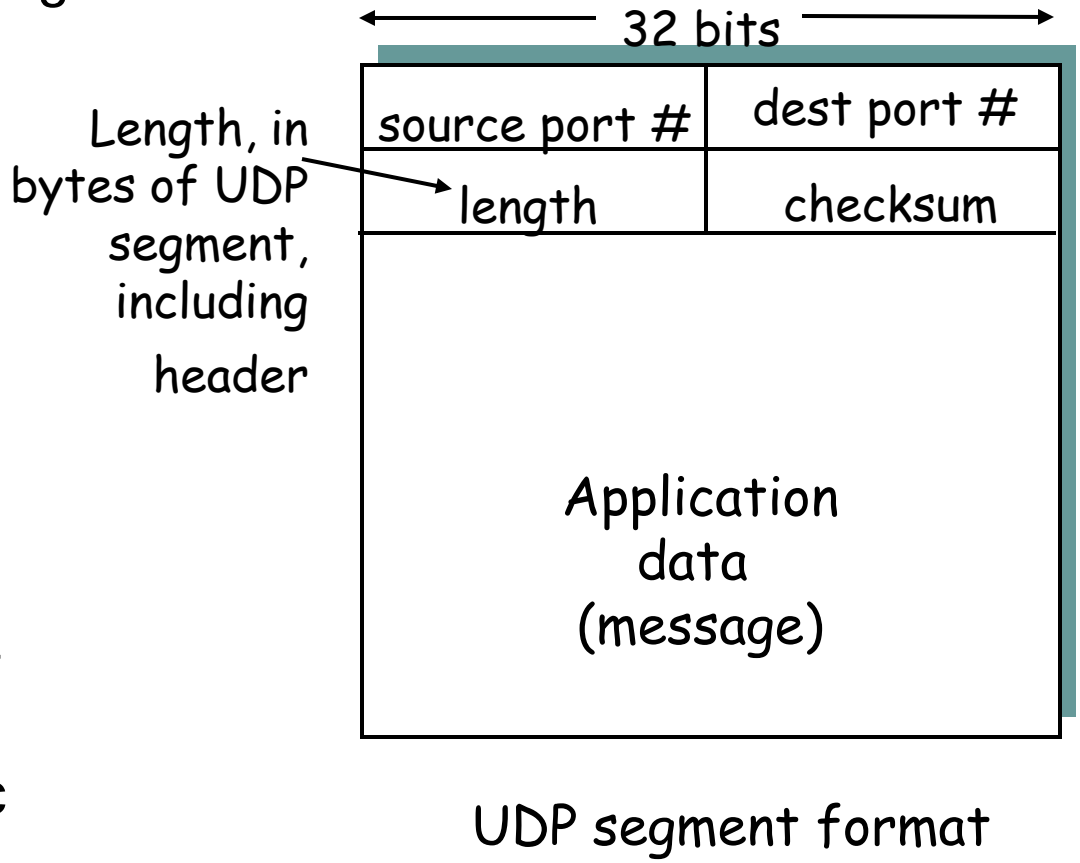
- "no frills," "bare bones" Internet transport protocol
- "best effort" service, UDP segments may be:
  - lost
  - delivered out of order to app
- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

Why is there a UDP?

- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small segment header
- no congestion control: UDP can blast away as fast as desired

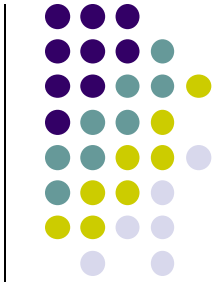
UDP: more

- often used for streaming multimedia apps
  - loss tolerant
  - rate sensitive
- other UDP uses
  - DNS
  - SNMP
- reliable transfer over UDP: add reliability at application layer
  - application-specific error recovery!



UDP segment format

UDP checksum



Goal: detect “errors” (e.g., flipped bits) in transmitted segment

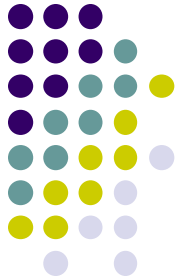
Sender:

- treat segment contents as sequence of 16-bit integers
- checksum: addition (1’s complement sum) of segment contents
- sender puts checksum value into UDP checksum field

Receiver:

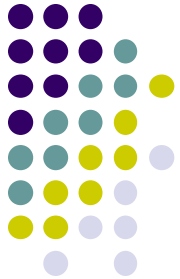
- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. *But maybe errors nonetheless?* ….

Benefits of UDP



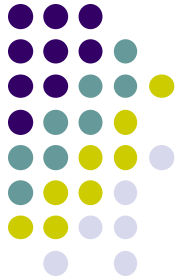
- Finer application level control over what data is sent and when:
  - When application sends data to UDP, it will be packaged and immediately sent to network layer
  - With TCP if links are congested, the sender will be throttled, thus message would be delayed.
  - TCP will also attempt to send a message until it is delivered.
  - E.g. Real time apps (e.g. VoIP) require a minimum sending rate, can tolerate some data loss and don't require re-transmission, thus are better suited to UDP.

Benefits of UDP



- No connection establishment
  - TCP establishes a connection by performing a handshake, thus adding some delay.
  - UDP does not introduce such a delay.
  - E.g. DNS runs over UDP. If it used TCP it would be much slower.
- No connection state
  - For the purpose of reliable delivery and flow control TCP must store connection state.
  - This requires more overhead on the hosts, and thus a UDP server can typically handle many more active clients compared to TCP.
- Packet overhead
  - TCP requires 20 bytes whereas UDP requires 8 bytes.

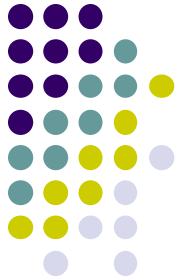
Popular internet apps and their underlying protocols



Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP	TCP
Remote terminal access	Telnet	TCP
Web	HTTP	TCP
File transfer	FTP	TCP
Remote file server	NFS	Typically UDP
Streaming multimedia	typically proprietary	Typically UDP
Internet telephony	typically proprietary	Typically UDP
Network management	SNMP	Typically UDP
Routing protocol	RIP	Typically UDP
Name translation	DNS	Typically UDP

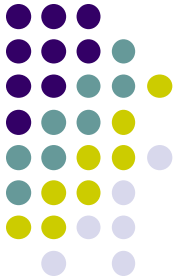
Figure 3.6 ◆ Popular Internet applications and their underlying transport protocols

UDP applications



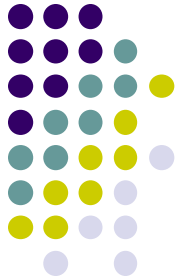
- Routing Information Protocol (RIP) – used to update routing tables. Updates are sent periodically e.g. every 5 min, thus lost updates are not critical.
- Simple Network Management Protocol (SNMP) – used to manage networks e.g. configure nodes, report errors, alarms. Typically used when network is in a stressed state e.g. congestion. Thus TCP would not be effective in such conditions.
- Real-time apps – IP telephony, streaming video react very poorly to TCP congestion control, resulting in many service providers choosing UDP.

UDP criticisms



- The use of UDP for multi-media apps has been criticized.
- If all users start to stream high bit-rate video, there would be so much packet overflow at routers that very few UDP packets would actually reach destination.
- Moreover uncontrolled UDP senders would cause TCP senders to throttle their flows.
- Thus overall throughput would be very poor.
- There have been proposals to extend UDP to introduce congestion control mechanisms.
- Example: Intelligent Packet Switch (telco)

Port scanning



- Port scanners are apps that sequentially scan ports looking for ports that accept TCP/UDP connections.
- Since some ports are reserved by well-known applications, its possible to infer what app is running on a host by doing a port scan.
- As an example MS SQL server (Database) runs on port 1341.
- This fact was exploited by the Slammer Worm to exploit a vulnerability in SQL Server.