Lecture 2-bis.

Internet Transport Protocols

As seen by the application developer point of view
The primary (in principle unique) role of Internet transport protocols

- Extend IP's delivery service (between two end systems) to a deliver service between two APPLICATION PROCESSES running on the end systems

- MAPPING to OSI language:
  - Port number = TSAP (Transport Service Access Point)
  - IP address = NSAP (Network Service Access Point)
Transport Layer Protocols

Entire network seen as a pipe

Internet

G.Bianchi, G.Neglia, V.Mancuso
The Internet level view

Information units travelling in the network: IP packets

<table>
<thead>
<tr>
<th>Header IP</th>
<th>IP Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>src &amp; dest IP addr</td>
<td></td>
</tr>
</tbody>
</table>

| Header transport | Transport data |
| src & dest ports |             |

<table>
<thead>
<tr>
<th>App prot header</th>
<th>Data</th>
</tr>
</thead>
</table>
Where are port numbers?

TCP or UDP

IP Header

IP Data

Transport data

Transport Header

App prot header

Data

Ipaddr src | Ipaddr dest | prtc | ...

G.Bianchi, G.Neglia, V.Mancuso
Transport Control Protocol (TCP)

- connection oriented
  - TCP connections
- reliable transfer service.
- TCP functions
  - application addressing (ports)
  - error recovery (acks and retransmission)
  - reordering (sequence numbers)
  - flow control
  - congestion control
Services Provided by TCP

⇒ Connection-oriented service: preliminary handshaking procedure creates a full duplex TCP connection

⇒ Reliable transport service: communicating processes can rely on TCP to deliver all the messages sent without error and in the proper order.

⇒ TCP does not provide:

⇒ a minimum transmission rate guaranteed (sending rate is regulated by TCP congestion control)

⇒ any sort of delay guarantees (the World Wide Wait …)
User Datagram Protocol (UDP)

- Connectionless
  - UDP packets
- Offers unreliable transfer service (*send and pray*).
- UDP functions
  - Application addressing (ports)
  - Error checking
Services Provided by UDP

- connectionless (no handshaking)
- arbitrary sending rate service
  - no congestion control mechanism present

- UDP minimalist lightweight service model does not provide:
  - any guarantee of reception, any guarantee of order
  - any guarantee on delay
**UDP**

- **Connectionless**
  - UDP packets
- **unreliable transfer service**
  - send and pray
- **UDP functions**
  - application addressing (ports)
  - error checking

**TCP**

- **connection oriented**
  - TCP connections
- **reliable transfer service**
  - all bytes sent are recv
- **TCP functions**
  - application addressing (ports)
  - error recovery (acks and retransmission)
  - reordering (sequence numbers)
  - flow control
  - congestion control
# Service Requirements

<table>
<thead>
<tr>
<th>Application</th>
<th>Data Loss</th>
<th>Bandwidth</th>
<th>Time sensitive?</th>
</tr>
</thead>
<tbody>
<tr>
<td>file transfer</td>
<td>no loss</td>
<td>elastic</td>
<td>no</td>
</tr>
<tr>
<td>electronic mail</td>
<td>no loss</td>
<td>elastic</td>
<td>no</td>
</tr>
<tr>
<td>Web documents</td>
<td>no loss</td>
<td>elastic</td>
<td>no</td>
</tr>
</tbody>
</table>
| real-time audio/video | loss-tolerant | audio: few Kbps to 1Mbps  
                        |                | video: 10's Kbps to 5 Mbps | yes: 100's of msec |
| stored audio/video| loss-tolerant  | same as interactive audio/video | yes: few seconds |
| interactive games | loss-tolerant  | few Kbps to 10's Kbps | yes: 100's msecs |
| financial applications | no loss | elastic | yes and no |
# Common Applications and related transport

<table>
<thead>
<tr>
<th>Application</th>
<th>Application-layer protocol</th>
<th>Underlying Transport Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>electronic mail</td>
<td>SMTP (RFC 821)</td>
<td>TCP</td>
</tr>
<tr>
<td>remote terminal access</td>
<td>Telnet (RTC 854)</td>
<td>TCP</td>
</tr>
<tr>
<td>Web</td>
<td>HTTP (RFC 2068)</td>
<td>TCP</td>
</tr>
<tr>
<td>file transfer</td>
<td>FTP (RFC 959)</td>
<td>TCP</td>
</tr>
<tr>
<td>remote file server</td>
<td>NFS</td>
<td>UDP or TCP</td>
</tr>
<tr>
<td>streaming multimedia</td>
<td>Proprietary (e.g., Real Networks)</td>
<td>UDP or TCP</td>
</tr>
<tr>
<td>Internet telephony</td>
<td>proprietary (e.g. Vocaltec)</td>
<td>typically UDP</td>
</tr>
</tbody>
</table>
A closer look at applications: The Socket Interface

(HTTP example)

Client

HTTP Application Process (Browser)

Socket

TCP software

Server

HTTP Application Process (HTTP Daemon)

Socket

TCP software

HTTP request

HTTP response

INTERNET

TRANSPORT PROTOCOL (for HTTP: TCP)

G.Bianchi, G.Neglia, V.Mancuso
Sockets in Unix OSs

- Just file descriptors (everything is a file in Unix!)
- “stream sockets” using TCP
- “datagram sockets” using UDP
- Common I/O file functions: read(), write()
- More powerful I/O functions: send(), receive()
- Other specific function: socket(), bind(), connect(), listen(), accept()
The application developer view

- the only mean for apps to send/receive messages is through sockets
- “doors” that hide transportation infrastructure to processes
- Very limited control on transport protocol (buffer sizing, variables)
Why it is trivial (!) to write networking apps?

**Application software duties:**
- open socket (e.g. C, C++, JAVA function call, OS call, external library primitive)
- Injects message in its own socket
- being confident message is received on the other side

**TCP software: in charge of managing segments!**
- reliable message transport when TCP used
- Segmentation performed by TCP transmitter
- Receive buffer necessary to ensure proper packet’s order & reassembly
An open question

녹 Socket: OS interface between the application and the transport level

녹 Ports: numbers in the transport header to identify the specific application

녹 Which is the relation?
 녹 We focus on the server
A first hypothesis

→ one-to-one mapping

socket ←→ port #
Trivial refinement

The socket is on a specific host,
i.e. port# has a local meaning

socket $\leftrightarrow$ (IP address, port #)
How to reach server socket: pair of IP Address and Port Number

- **Mail srv**
- **Web srv**
- Proprietary app

**Incoming request**

**Transport layer**
- 25
- 80
- 3211
- ...

**Internet layer**
- 131.175.15.1

**(IP, port): a unique identification of an application layer service to which requests need to be sent**

**The first contact needs well known port #**
Demultiplexing at receiver (1)

Information entering app
Software (managed by app Developer)

Transport SW: checks segment;
Sends to application sw based on Port number
**Application demux**

IP SW: checks IP packet;
Sends to transport sw
**Transport demux**
Demultiplexing at receiver (2)

Information entering app
Software (managed by app Developer)

Transport SW: checks segment;
Sends to application sw based on
Port number

Application demux

IP SW: checks IP packet;
Sends to transport sw
selects whether UDP or TCP

Transport demux

G.Bianchi, G.Neglia, V.Mancuso
Does IP software know about transport protocol?

TCP or UDP

IP Header

Ipaddr src | Ipaddr dest | prtc | … | IP Data

Transport Header

Port src | Port dest | … | Transport data

App prot header | Data

YES

G.Bianchi, G.Neglia, V.Mancuso
Remarks

➔ When opening socket, needs to specify which transport to use!
➔ UDP port numbers are independent from TCP ones!
   ⇒ This means that TCP looks at TCP ports, while UDP looks at UDP ones
➔ Normally (for pure convenience) port number = same meaning for TCP and UDP
   ⇒ if a well known service is offered by both TCP and UDP, the port number is the same
   ⇒ if a well known (low port number) service is offered for one protocol only, the corresponding port for the other protocol is generally unused
➔ BUT possibly the same port number has different meaning for TCP and UDP....
   ⇒ Details in RFC1700 or http://www.iana.org/assignments/port-numbers
Consequence

- If two applications employ different protocols, they can employ the same port #
- Mapping refinement

socket \(\leftrightarrow\) (protocol, IP addr., port #)

- Is it enough? Not always
(TCP) Connections
identified by sockets at its ends

CLIENT

IP: 151.100.37.9

WEB SERVER

IP: 131.175.21.1

G.Bianchi, G.Neglia, V.Mancuso
Managing multiple connections

A socket for each connection is needed, because each connection has its own status.

It is not true for UDP (connectionless).

CLIENT A
IP: 151.100.37.9

CLIENT B
IP: 193.47.31.18

WEB SERVER
IP: 131.175.21.1

G.Bianchi, G.Neglia, V.Mancuso
Managing multiple connections

If each port identifies a socket...

How can a new client know the listen port?

Are the port numbers enough?
Managing multiple connections

Connections can be distinguished by client and server IP addresses and ports

Connection A
[<151.100.37.9,3124>,<131.175.21.1,80>]

Connection B
[<193.47.37.9,12134>,<131.175.21.1,80>]
Managing multiple connections

Two sockets for the same server port!

Connection A
[<151.100.37.9,3124>,<131.175.21.1,80>]

Connection B
[<193.47.31.18,12134>,<131.175.21.1,80>]
Demultiplexing at receiver (3)

**IP SW**: checks IP packet; Sends to transport sw

**Transport SW**: checks segment; Sends to application sw based on IP addresses and Port numbers

**Application demux**

Information entering app
Software (managed by app Developer)

**Transport demux**

IP SW: checks IP packet; Sends to transport sw selects whether UDP or TCP

Also information about the IP addresses is needed at the transport level
Conclusions

→ A socket always identifies unique protocol and port

socket → (protocol, IP addr, port #)

→ It can identify also address and port of the remote application

socket → (prot, src IP addr, src port, dest IP addr, dest port)
Conclusions

Protocol and port can identify a unique socket

socket ← (protocol, IP addr, port #)

but in general more information is required

socket ← (prot., src IP addr, src port, dest IP addr, dest port)
How to reach client socket
another pair of IP Address and Port Number

⇒ The server needs to know:
  ⇨ The host to which send a response
      ⇒ src IP address
  ⇨ The application software process at client side
capable of correctly interpret the response
      ⇒ src port #

⇒ Generally client DOES NOT use a well known port #
  ⇨ It is not needed (the client starts talking)
  ⇨ OS just assigns one available (Ephemeral ports)

Typical question: WHY every PC needs an IP address?
More complex issue: HOW your home PC gets an IP address?
Port numbers

→ 16 bit address (0-65535)

→ well known port numbers for common servers
  ⇒ FTP 20, TELNET 23, SMTP 25, HTTP 80, POP3 110, … (full list: http://www.iana.org/assignments/port-numbers)

→ number assignment
  ⇒ 0-1023 (system) well known ports: service contact ports assigned by IANA, on most systems they can only be used by system (or root) processes or by programs executed by privileged users.
  ⇒ 1024-49151 (user) registered ports: service contact ports listed by IANA, on most systems they can be used by ordinary user processes or programs executed by ordinary users.
  ⇒ 49152-65535 dynamic/private ports.
Last remark about terminology

- Sometimes socket is considered synonym of the quintet:
  (prot., src IP addr, src port, dest IP addr, dest port)