Date 11/02/2024

CSE421 – Computer Networks – SFQ

Lecture 06 – Transport layer : UDP and TCP

* Process-to-Process communication
  (PC1's app^n to PC2's app^n – comm^n)
* PDU (Protocol Data Unit) – Segment (Header + Data)
  ↳ Address : Port address / Port number.
* Transport layer – logical comm^n bet^n processes.
* Network layer – " " " hosts.
* Functions of Transport layer –
  (I) Segmenting the data
  (II) Reassembling " segments into streams of app^n data.
  (III) Adds port address to identify different app^n
  (IV) Multiplexing
  (V) Flow control
  (VI) Initiating + terminating a session
  (VII) Error Recovery

UDP + TCP

TCP

* Web App^n – require data to be complete wf no errors or gaps & they can accept a slight delay to ensure this → TCP – Reliable

Email

* Online games – can deal wf occasional errors but cannot accept any delay → UDP – Fast

Streaming

– Hence, different app^n have different requirements

(*) User Datagram Protocol (UDP) –
  (I) No connect establishment which could add delay.
  (II) simple – no connection state at sender/receiver state
  (III) header size ↓
  (IV) no congestion or error control

Function-01 : Segmentation and Reassembly

* session layer to transport layer – Encapsulation
* transport " to session " – Decapsulation
* Transport layer divides the data into segments and adds a header for delivery over the network.

* UDP – connectionless + unreliable; out of order data not rearranged; lost data not re-sent; no flow and error control.

Date: ...../...../.....

UDP Headers → 8 bytes (64 bits)

UDP – provides S and D ports

Header	Data

↳ 16 bits Source (S) Port Number
16 bits Destn (D) " "
16 bits Total length of Header+Data (8) (?) — how much data will this segment carry?
16 bits Checksum (sender end) ↳ to check if data is properly received.
64 bits

Function 02 – Identifying Appn w/ Port Numbers:
16 bits – represented as one decimal number
Range: 0 ~ 65535
(i) 0 ~ 1023 – well known ports (web-80; POP-110; SMTP-25)
(ii) 1024 ~ 49151 – registered ports (for an appn you made)
(iii) 49152 ~ 65535 – private/dynamic ports (end devices – PC or mobile) ↙ assign themselves randomly (based on availability)

Ex. www.google.com
↳ HTTP Protocol
↳ Web Server – 80

case-01

Request: client sets D port number → 80
S port number → dynamic port (49152 ~ 65535)

Response: server sets D port number → dynamic port
S port number → 80

case-02

Multiple sessions of the same web server: — (from the same PC)
Client uses two different dynamic ports as its S but destination (web server) port number remains same.
– different S ports keep the sessions unique on the server.

PC1: S → 49650 Web Server: S → 80
(i) D → 80 (i) D → 49650

(ii) S → 49653 (ii) S → 80
D → 80 D → 49653

different devices
↑ – chance of overlapping port addresses

Date: ..../..../....

case-03 Same webserver from two different PCs w/ same port number :—
PC keeps track of its own port numbers but has no record of other PC's port numbers.

PC1 : S → 49650 Web server : S → 80
D → 80 D → 49650 ?
IP → 172.16.230.5 IP → 207.22.146.33

PC2 : S → 49650
D → 80
IP → 172.16.230.6

* So in order to differentiate bet^n the two ports/devices, the transport layer uses the socket address.
Socket Address → IP Address : Port
↳ always unique (Network layer)

PC1 172.16.230.5 : 49650 ⟷ 207.22.146.33 : 80 } web server
PC2 172.16.230.6 : 49650 ⟷ 207.22.146.33 : 80

Use of UDP — (I) Streaming (IV) HTTP/3
(II) DNS
(III) SNMP

Reliability ↑ — add reliability at app^n layer + app^n specific error recovery

If UDP identifies an error using checksum then it drops the entire packet. Does not have the ability to tell the sender that there was an error and to send it again. → No error control. (cannot fix the error)
* checksum is not a 100% safe method – cannot detect error in case of bit flip caused by corruption

↗ connectionless (UDP)

Function-03 : Multiplexing and Demultiplexing.
Sending End – Input : multiple ; Output : one (App^n to Transport)
Receiving End – Input : one ; Output : Multiple (Transport to App^n)

Date: ...../...../.....

Connectionless Demultiplexing (UDP)	Connection-Orientated Demultiplexing (TCP)
* uses destination port number to identify UDP socket to which the segment will be directed. * n clients → 1 socket	* 4-tuple to identify TCP socket (I) source IP address (II) source port " (III) dest^n IP " (IV) dest^n port " * n clients → n sockets