Prop. & Trans. Delay

Hosts A⟶B connected by single link | if (Host⟶ pack at t=0) | when(t=d_{trans}){ endOfPack = (just leaving Host)}
$R ⟶ rate(bps) of link from A⟶B | if(d_{prop}>d_{trans}) | when(t=d_{trans}){ firstBitOfPack = (in link & hasn't reached Host(B)
$L ⟶ size of pack. Sent from A⟶B | if(d_{prop}<d_{trans}) | when(t=d_{trans}){ firstBitOfPack = (just reached Host(B)
$m ⟶ meters | if($d_{prop}$=$d_{trans}$) | $m = ($L/$R) * ($s)
$s ⟶ prop. Speed along link ($m/sec) | d_{trans} = L/R seconds
d_{prop}=m/s seconds | $d_{end\text{-}to\text{-}end}$ = (m/s + L/R) seconds

---

$d_{end\text{-}to\text{-}end}$ for packet with multiple links and switches with no queueing delays

Let d_i, s_i, R_i – length, prop speed, & transmission rate of link-i for i=1,2,...,total links | Switch delays = d_{proc}

d_{proc}= $L/R_1 + L/R_2 + \ldots + L/R_{total\text{-}links} + d_1/s_1 + d_2/s_2 + \ldots + d_{(total\text{-}links)}/s_{(total\text{-}links)}$ + (number of switches)*(d_{proc})

Queue-Delay | all packets have length L, Transmission Rate=R, x bits of the currently-being-transmitted packet have been transmitted, and n packets are already in queue

Queueing delay = (nL + (L - x))/R

Throughput

M = (number of client server pairs). R_s, R_c, and R are rates of the server links, client links and network link.
If all other link have abundant capacity and there is no other traffic,
Throughput = min{R_s, R_c, R/M}
If – $M Paths between the server and the client. No two paths share any link. Path k(k=1,...,M) consists of N links with transmission rates R1k, R2k,...,Rnk. (1)Server can only use one path to send data to client. (2)Server can use all M paths
(1)Max throughput = max{min{R_1^1, R_2^1,...,R_N^1},min{R_1^2,R_2^2,..., RN2},...,min{R_1^M,R_2^M, ... , R_N^M}}
(2)Max throughput= summation(k=1⟶M) min{ R_1^k, R_2^k,...,R_N^k}

Bandwidth delay & Bit Width

Calculate bandwidth-delay product, R*d_{prop} | R=(X)Mbps=X*(1e+6) delay = Xbits
If a file is sent continuously as one large message the maximum number of bits that will be in link=R*d_{prop}
Bandwidth-delay product of a link is the the maximum number of bits that can be in the link.
Width of a bit(meters) in the link = length of the link/bandwidth-delay product
width of bit when (s=prop speed, R = trans. Rate) = s/R
-x=photoSize, min value of x for microwave link to be cont. Transm. ,(R=Mbps) | (R*10^6)(interval between photo) = X bits

----

Packet Travel Through switches to destination

(hop occurs when a packet is passed from one network segment to the next.)
($1)Time from source to first packet = ($s = (#.#) * 10^k)*/(link speed Mbps = (#.#) * 10^k)= Z msec
($2)Time from source host to destination host with store-&-forward switching = (Z secs) * (X hops) = Y msec
-if each pack is 10,000 bits long how long does it take to move the first packet from source host to first switch if link is 2Mbps. (1*10^4)/(2*10^6)=5msec. Second packet would be 2 * 5msec = 10msec
Time for file source ⟶ destination with message sementation = ($2)+(($N^{th}$ packet received)*($1))
(Pro)Message segmentation⟶errors aren't tolerated⟶huge packet is retransmitted. (Con) Smaller packets wait in queue

---

HTTP

Persistent⟶ forces requests and responses over one TCP connection | Non-Persistent⟶ sent over separate TCP connects.
Date header⟶ object modification details. | First (X) bytes of document being returned⟶ chars following blank <cr><lf>
Time taken for DNS look up + establish TCP connection + send&receive object ⟶ $2RTT_0 + (RTT_1 + RTT_2 + \ldots + RTT_n)$

Caching & Access link Utilization

total caching delay = Internet delay + access delay + LAN delay
access link utilization = delay with cache - (.x(percentage of requests satisfied @ orig) * (Mbps)/(Access link Mbps)

DNS

DNS name resolution: requesting host(1)⟶ ⟵(8)local DNS serv(2)⟶⟵(7)root DNS(3)⟶⟵(6)TLD DNS(4)⟶⟵(5)Authoritative DNS(dns.cs.umass.edu – gaia.cs.umass.edu)

Client Server | P2P

u_i: peer I upload capacity | d_i: peer I download capacity | u_s: server upload capacity | Time to send one copy: F/u_s
Time to send N Copies NF/u_s | d_{min} : min client download rate | max client download time: F/d_{min}
Time to distribute F(file size) to N clients using cli-serv approach: $D_{c\text{-}s}$>=max{NF/u_s,F/d_{min}} |
Time to distribute F(file size) to N clients using P2P approach: D_{P2P}>=max{F/U_s,F/d_{min},NF/(u_s + summation(u_i))}
c---

UDP/TCP checksums | Given 3 8-bit bytes: XXXXXXXX, YYYYYYYY, ZZZZZZ
Find complement of the resultant sum – find sum of first 2 bytes, add that sum to the last byte, add carry if necessary, calculate 1's complement of the resultant sum.
Reciever detects errors | if sum contains all 1's then there are no errors, if sum contains at least one 0, there are errors

----

Channel Utilization | CU = N * ((L/R)/((L/R)+RTT) // solve for N
if solving for window size for channel utilization to be greater than X%, plug in . X as CU in formula, solve for N
(utilization)fraction of time sender busy sending – (L/R)/(RTT + L/R)

TCP MSS trans|max value of L so TCP seq nums !exhausted. TCP seq field=4bytes| 4x8bits=32bits – 2^{32}/1024=4.19G bytes
solve time **L**-transmit file = (2^{32} + ((2^{32}/MSS)*(#bytes of header added to each segment)) * 8bits/((Link speed)x10^{Y} bps)
===

Host communcation on TCP connection |
seqence number of first segment = x, source port first segment=y, dest port =z in 2nd segment A->Bsequence number=sequence number of the $first segment + number of bytes of data in $first segment.
1st segment before 2nd : Ack#=**sequence number(above formula)|** 2nd segment before 1st : Ack#=**x**

RTT | estimatedRTT= (1-(alpha))*EstimatedRTT+(alpha)*SampleRTT_Value |
DevRTT=(1-(B)).DevRTT+B*|SampleRTT_Value-EstimatedRTT **|** TimeoutInterval = EstimatedRTT+4*DevRTT

Wireshark TCP Handshakes **and identifying UDP parts of packet**

TCP 3 way handshake(wireshark info section)

57966 ⟶ 80[SYN] Seq=0 Win=65535 Len=0…
80 ⟶ 57996[SYN, ACK] Seq=0 Ack=1 Win=…
57996 ⟶ 80[ACK]Seq=1 Ack=1 Win=131904

TCP 4 way handshake(wireshark info section)

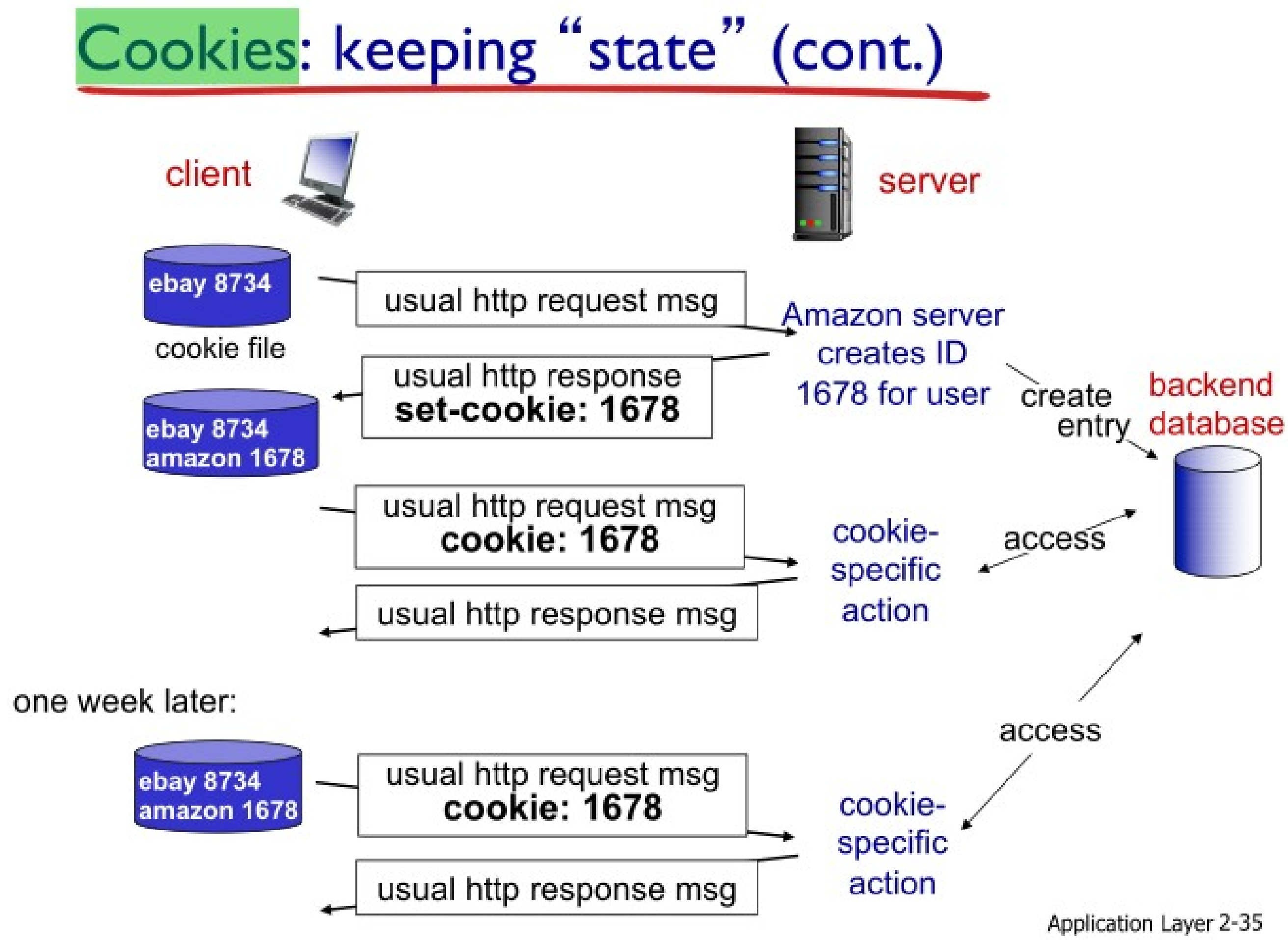
80⟶ 57966[FIN,ACK] seq=348 Ack=309 Win=28032
57966⟶80[ACK] seq=309 Ack=349 Win=131552
57966⟶80[FIN,ACK] Seq=309 Ack=349 Win=131552
80⟶57966[ACK] Seq=349 Ack=310 Win=28032

For sections of UDP: info will allways be something like | 51164⟶krb524(4444) Len = 1470 and source and destination ip addresses will be static

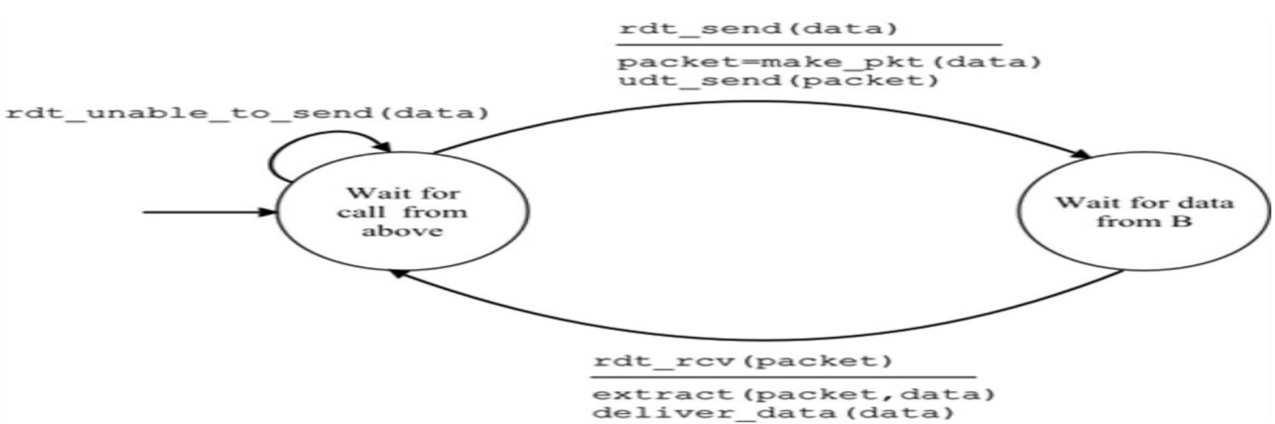
----

TCP closing a connection: ⟶ = to server state; ⟵ = to client state
clientSocket.close()⟶FINbit=1, seq=x⟵ACKbit=1; ACKnum=x+1(waitForServerClose)⟵FINBit=1,seq=y⟶ACKbit=1; ACKnum=y+1 | Client state: ESTAB⟶FIN_WAIT_1⟶FIN_WAIT_2⟶TIMED_WAIT⟶CLOSED |
Server state: ESTAB⟶CLOSE_WAIT⟶LAST_ACK⟶CLOSED

RDT FSM (/*Illustration*/)


Cookies: keeping "state" (cont.)
client
server
ebay 8734
cookie file
usual http request msg
Amazon server creates ID 1678 for user
usual http response
set-cookie: 1678
create entry
backend database
ebay 8734
amazon 1678
usual http request msg
cookie: 1678
cookie-specific action
access
usual http response msg
one week later:
access
ebay 8734
amazon 1678
usual http request msg
cookie: 1678
cookie-specific action
usual http response msg
Application Layer 2-35

RDT FSM


rdt_send(data)
packet=make_pkt(data)
udt_send(packet)
rdt_unable_to_send(data)
Wait for call from above
Wait for data from B
rdt_rcv(packet)
extract(packet,data)
deliver_data(data)