CS5222 Computer Networks and Internets

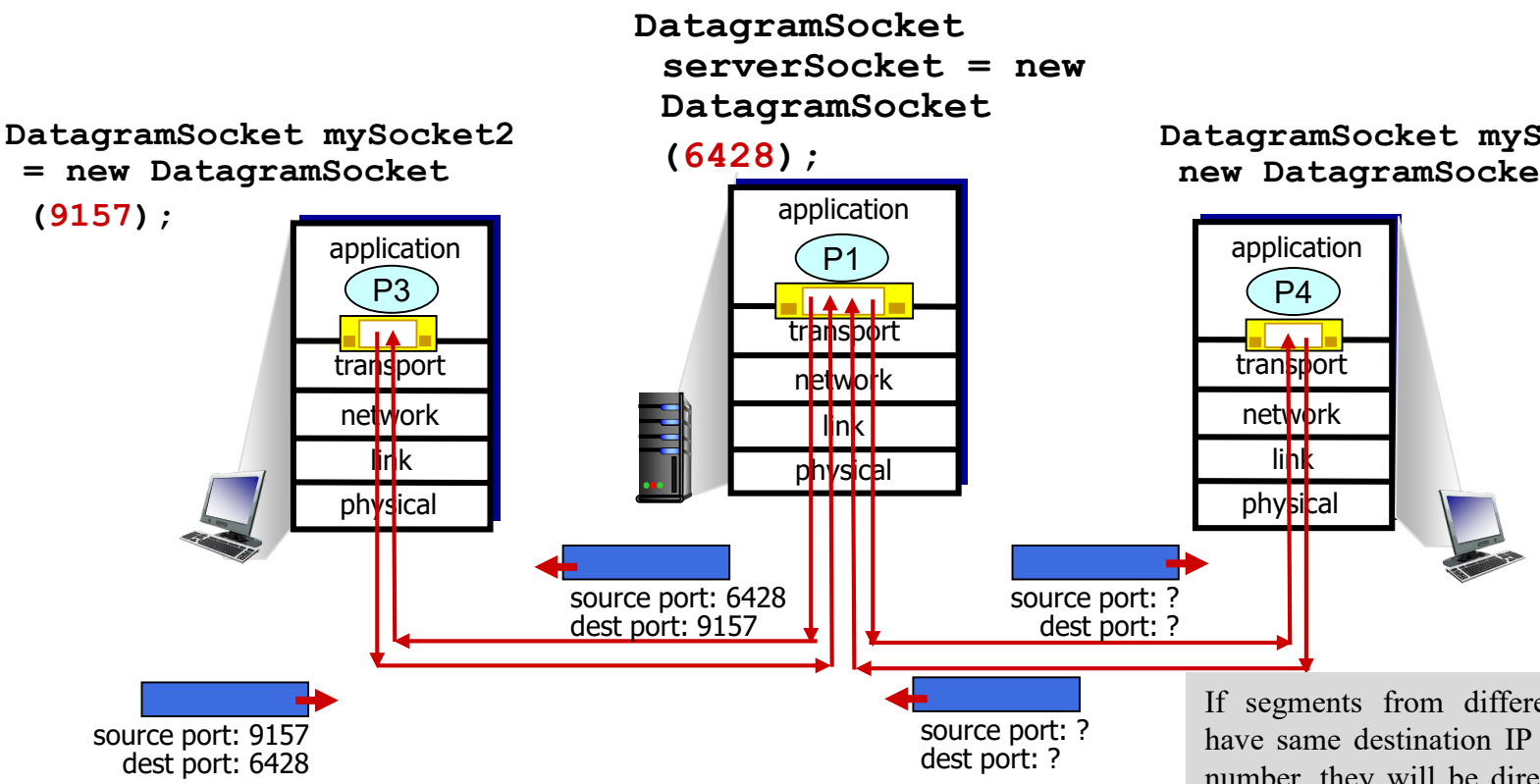
Tutorial 4 (Week 4)

Prof Weifa Liang

Weifa.liang@cityu.edu.hk

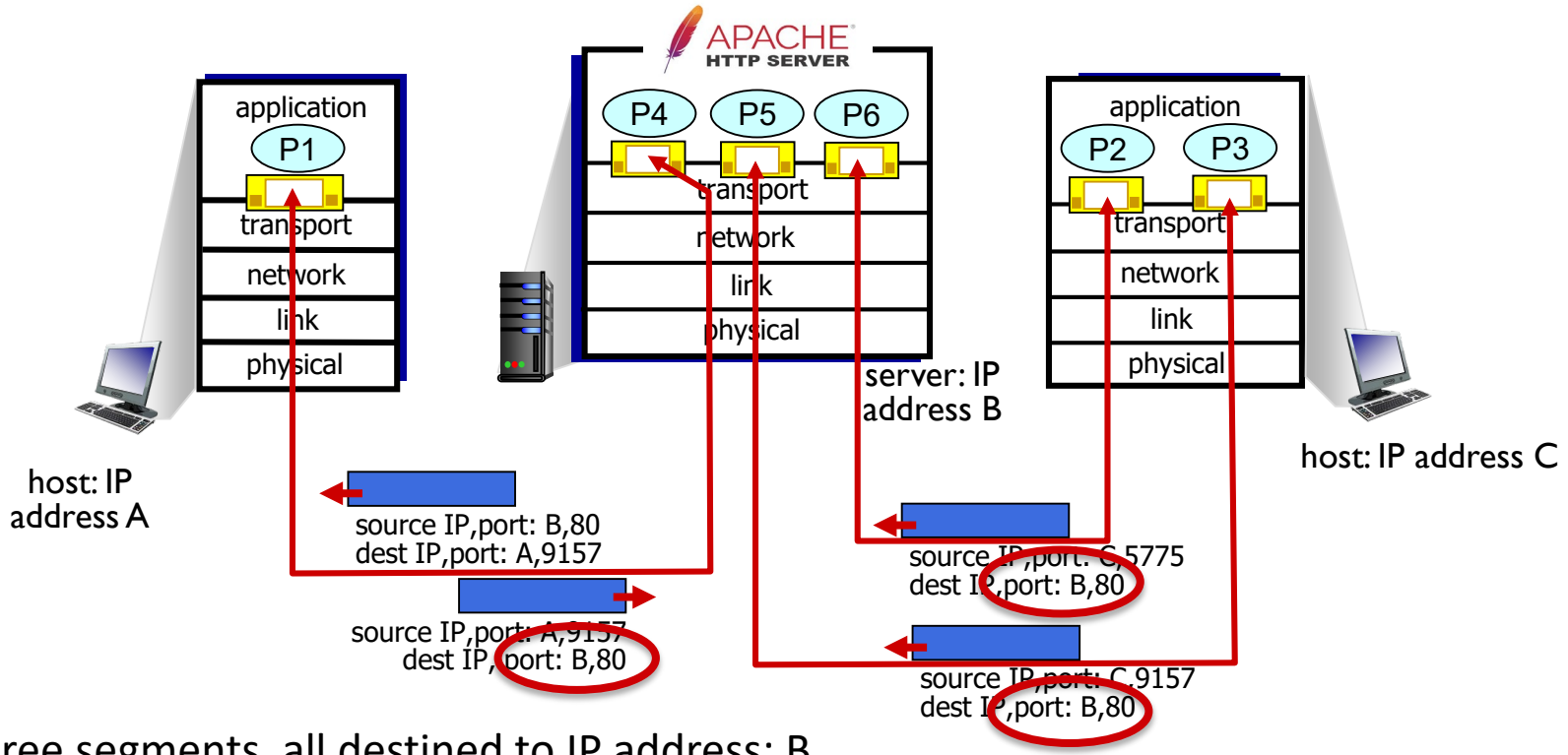
Slides based on book *Computer Networking: A Top-Down Approach.*

Connectionless demultiplexing: an example



If segments from different IP addresses have same destination IP address and port number, they will be directed to the same process

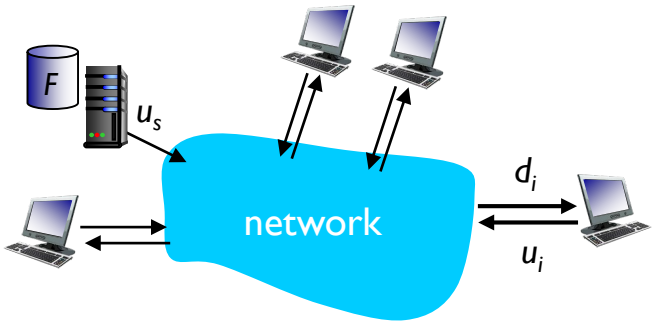
Connection-oriented demultiplexing: example



Three segments, all destined to IP address: B,
dest port: 80 are demultiplexed to *different* sockets

File distribution time: client-server

- *server transmission:* must sequentially send (upload) ***N*** file copies:
- time to send one copy: F/u_s
- time to send ***N*** copies: NF/u_s



- *client:* each client must download file copy
  - d_{min} = minimum client download rate
  - min client download time: F/d_{min}

time to distribute file to N clients using client-server approach

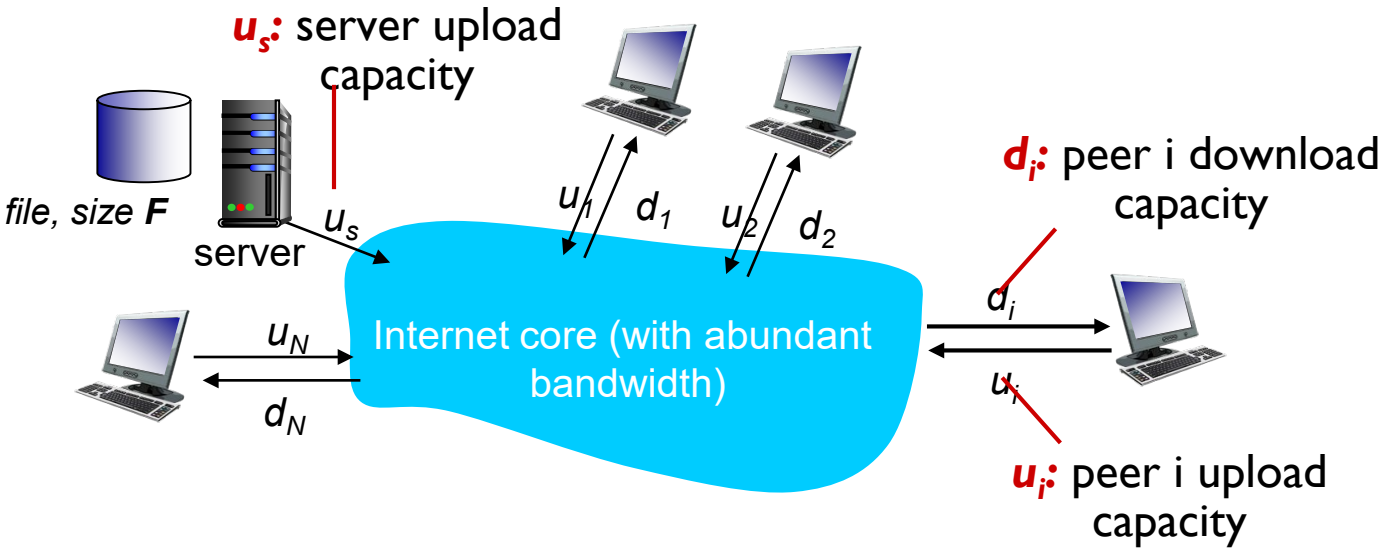
$$D_{c\text{-}s} \geq max\{NF/u_s, F/d_{min}\}$$

increases linearly in N

File distribution: client-server vs P2P

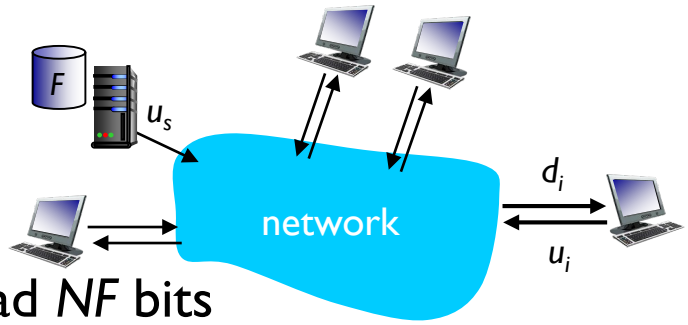
Q: How much time to distribute file (size ***F***) from 1 server to ***N*** *peers*?

- upload/download capacity is limited resource



File distribution time: P2P

- *server transmission:* must upload at least *one* copy:
  - time to send one cop
- *client:* each client must download file copy
  - min client download time: F/d_{min}
  - *clients:* as aggregate download NF bits
  - max upload rate (limiting max download rate) is: $u_s + \Sigma u_i$



time to distribute file to N clients using P2P approach

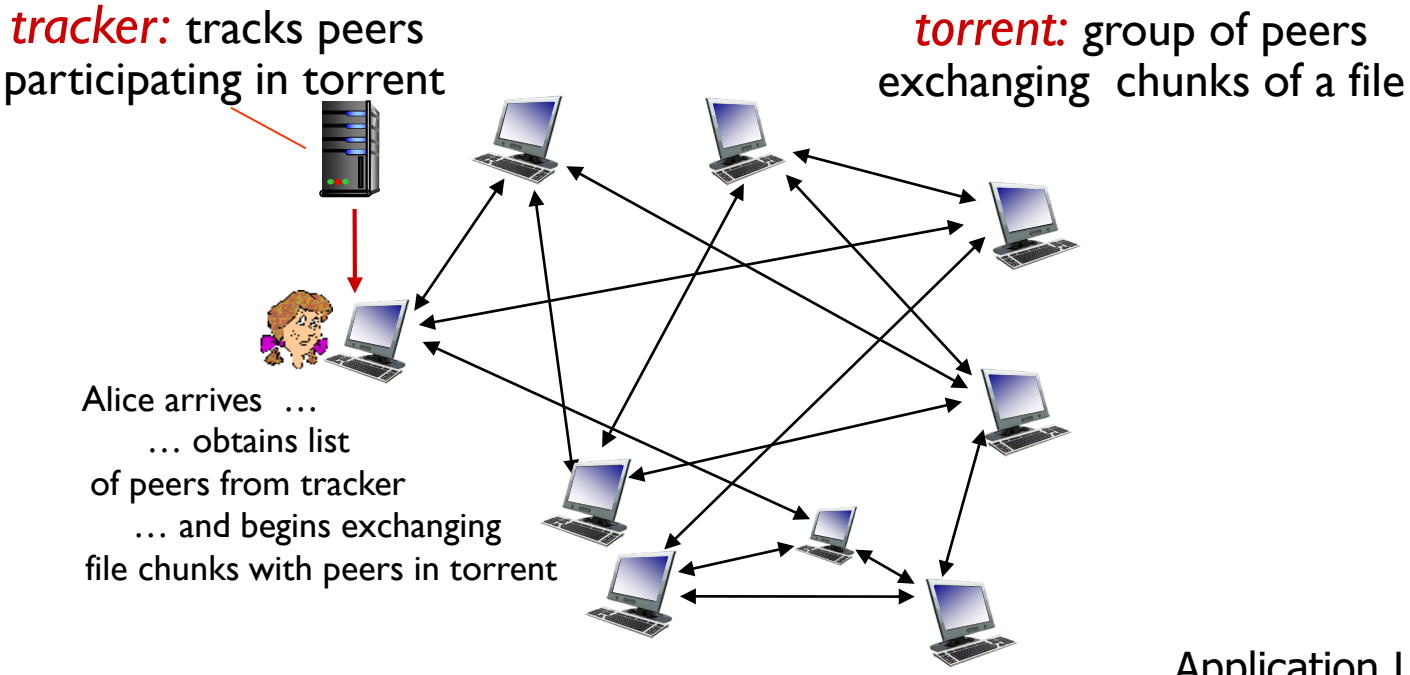
$$D_{P2P} \geq max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$$

increases linearly in N …

… but so does this, as each peer brings service capacity

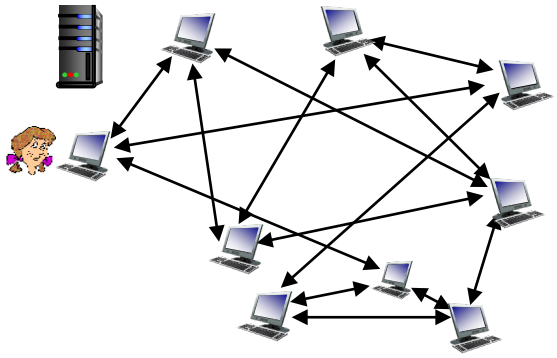
P2P file distribution: BitTorrent

- file divided into 256Kb chunks
- peers in torrent send/receive file chunks



P2P file distribution: BitTorrent

- peer joining **torrent**:
  - has no chunks, but will accumulate them over time from other peers
  - registers with **tracker** to get list of peers, connects to subset of peers ("neighbors")



- while downloading, peer uploads chunks to other peers
- peer may change group of peers with whom it exchanges chunks
- *churn:* peers may come and go
- once peer has an entire file, it may (selfishly) leave or (altruistically) remain in torrent

BitTorrent: requesting, sending file chunks

Requesting chunks:

- at any given time, different peers have different subsets of file chunks
- periodically, Alice asks each peer for list of chunks that they have
- Alice requests missing chunks from peers
  → **"rarest first" rule**

Sending chunks: tit-for-tat

- Alice sends chunks to those four peers currently sending her chunks *at highest rate*
  - other peers are "choked" by Alice (do not receive chunks from her)
  - re-evaluate top 4 every 10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - "optimistically unchoke" this peer
  - newly chosen peer may join top 4

UDP checksum

Goal: detect errors (*i.e.,* flipped bits) in transmitted segment

sender:

- treat contents of UDP segment (including UDP header fields and IP addresses) as sequence of 16-bit integers
- checksum: addition (**1s complement** sum) of segment content
- checksum value put into UDP checksum field

receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - Not equal - error detected
  - Equal - no error detected. *But maybe errors nonetheless?*

- Work on your questions

1. Suppose we used 8-bit sums instead of 16-bit sums to compute UDP checksum. What's the checksum of three bytes: 01010011, 01010111, 01110100?

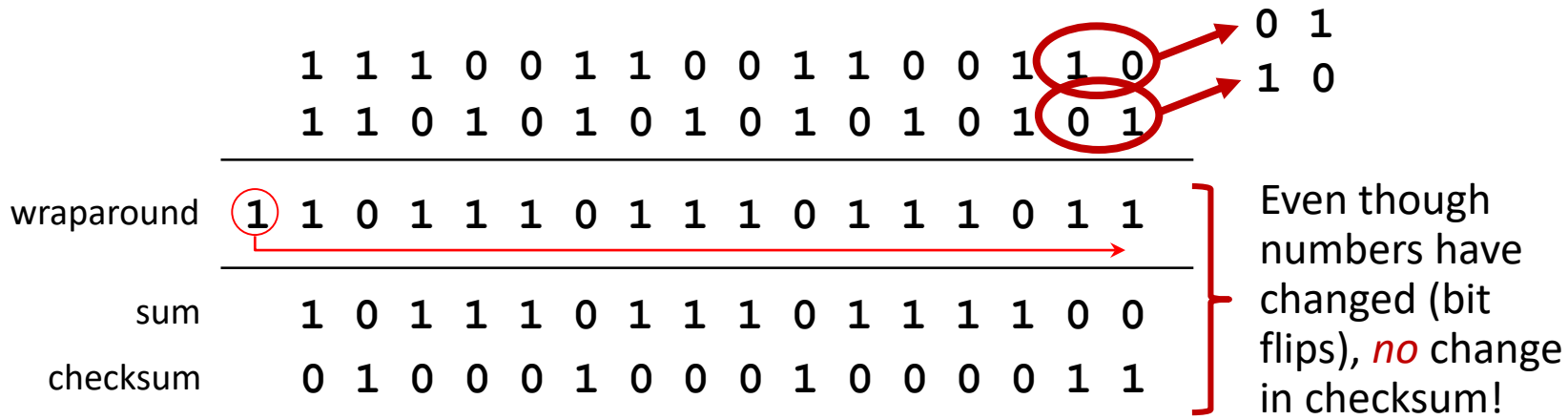
```
  01010011
  01010111
+ 01110100
 100011110
```

- Add the carry out to the least significant bit, we have 00011111,
- the checksum is: 11100000
- Q: what do we get if we compute: sum + checksum ?

2. With the 1's complement of the sum, how does the receiver detect errors? Is it possible that a 1-bit error will go undetected? How about a 2-bit error?

- **Answer**: at the receiver, we can add up the 4 bytes (3 bytes + checksum). If the result is all 1s, there was no error; otherwise, there is an error.
- It is impossible that a 1-bit error will go undetected, as any 1-bit error will make the result have at least one 0.
- It is possible that a 2-bit error will go undetected as it is possible that the sum remains the same when 2 bits are flipped.

UDP checksum: undetected error!



3. Consider distributing a file of F = 15 Gbits to N peers. The server has an upload rate of u_s = 30 Mbps, and each peer has a download rate of d_i = 2 Mbps and an upload rate of u = 700 Kbps. Give the minimum distribution time for N = 10 and N = 1,000 for both (a) client-server distribution; and (b) P2P distribution.

- For calculating the minimum distribution time, we use:
  - For client-server: $D_{CS} >= \max\{ N F / u_s, F / d_{min}\}$, d_{min} is the minimum download rate
  - For p2p: $D_{p2p} >= \max\{ F / u_s, F / d_{min}, N F / (u_s + \sum u_i)\}$
- Before plugging the values of F and u into these formulas, we convert them to Mbps.
- N=10
- $D_{CS} >= \max\{ 10 * 15*10^3 / 30, \mathbf{15*10^3 / 2} \}$
- = 7,500 sec (2nd term dominates because # of users is small)
- $D_{P2P} >= \max\{15*10^3 / 30, \mathbf{15*10^3 / 2}, 10 * 15*10^3 / (30 + 10*0.7) \}$
- = max{ 500, **7,500**, 4,054 } = 7,500 sec
- N=1,000
- $D_{CS} >= \max\{ \mathbf{1{,}000*(15*10^3 / 30)}, (15*10^3 / 2)$
- = max{ **500,000**, 7,500 } = 500,000 sec
- $D_{P2P} >= \max\{15*10^3/30, 15*10^3 / 2, \mathbf{1{,}000 * 15*10^3 / (30 + 1{,}000*0.7)} \}$
- = max{ 500, 7,500, 20,548 } = 20,548 sec

Numerator and denominator both scale with N, (almost) cancel each other out

4. Suppose that Alice writes a Bittorrent client that doesn't allow other clients to download any data from her system. She claims that, by using this client, she can join a torrent and still receive a complete copy of the shared file. Is Alice's claim possible? Why or why not?

- **Answer**:
- Yes, it is possible.
- Note that even though Alice will never become one of the top-4 hosts of other peers, she might nevertheless become "optimistically unchoked" by other peers and start receiving chunks for some time.
- Thus, if she waits long enough, she is likely to receive all chunks of the file.

- 5. Consider a short, 10-meter link with a **propagation speed** of $300*10^6$ m/sec, over which a sender can **transmit at a rate of 150 bits/sec** in both directions. Suppose that packets containing data are 100,000 bits long, and packets that contain only control messages (TCP or HTTP GET request) are 200 bits long. Assume that there are N parallel connections with each having 1/N of the link bandwidth.
- Consider the HTTP protocol and suppose that each downloaded object is 100 Kbits long, and that the base HTML file contains 10 referenced objects located at the same host.

a) Would parallel downloads via 10 parallel instances of non-persistent HTTP make sense in this case? How long does it take?

- 5. Consider a short, 10-meter link with a **propagation speed** of $300*10^6$ m/sec, over which a sender can **transmit at a rate of 150 bits/sec** in both directions. Suppose that packets containing data are 100,000 bits long, and packets that contain only control messages (TCP or HTTP GET request) are 200 bits long. Assume that there are N parallel connections with each having 1/N of the link bandwidth.
- Consider the HTTP protocol and suppose that each downloaded object is 100 Kbits long, and that the base HTML file contains 10 referenced objects located at the same host.

a) Would parallel downloads via 10 parallel instances of non-persistent HTTP make sense in this case? How long does it take?

Answer:

- Let Tp denote the one-way propagation delay between the client and the server.
  - Tp = 10 / (300 * 10^6) = 0.03333 microsec. (1 second=10^6 micorseconds)
- Each downloaded object can be completely put into one data packet.
- Time to get the base HTML file:
  - **B** = (200/150+Tp + 200/150 +Tp + 200/150+Tp + 100,000/150+ Tp)
- Time to get 10 objects **over parallel connections** of **non-persistent HTTP**:
  - 200/(150/10)+Tp + 200/(150/10) +Tp + 200/(150/10)+Tp + 100,000/(150/10)+ Tp
- In total:
  - 7,377 + 8*Tp (seconds)

Only (1/10)-th of bandwidth

- 5. Consider a short, 10-meter link with a **propagation speed** of $300*10^6$ m/sec, over which a sender can **transmit at a rate of 150 bits/sec** in both directions. Suppose that packets containing data are 100,000 bits long, and packets containing only control (TCP or HTTP GET request) are 200 bits long. Assume that N parallel connections each gets 1/N of the link bandwidth. Now consider the HTTP protocol and suppose that each downloaded object is 100 Kbits long, and that the base HTML file contains 10 referenced objects located at the same host.

b) Consider persistent HTTP (without parallel connections). Do you expect significant gains over the non-persistent case?

- Answer: total time = time to get the base HTML file (**B**) + 10 req/res sent back-to-back:
- After having received the base HTML file, the client sends 10 requests back-to-back
- This takes 10 * 200/150 + Tp = 13 sec
- However, as soon as the server receives the 1st request, it will start transmitting the first object, which takes 100,000 / 150 = 667 sec
- Since 13 < 667, the next 9 client requests arrive at the server before the server has finished transmitting the 1st object.
- Therefore, the server can transmit each of the 10 objects as soon as it is done with the previous one. This takes: 10*100,000/150 + Tp
- In total:
- **B** + (200/150 + Tp) + (10*100,000/150 + Tp) = 6668 + 6 * Tp (seconds)
- Conclusion: Persistent HTTP is faster.