Advanced Network Technologies

Week 2:

Network performance

Network application

Dr. Wei Bao | Lecturer

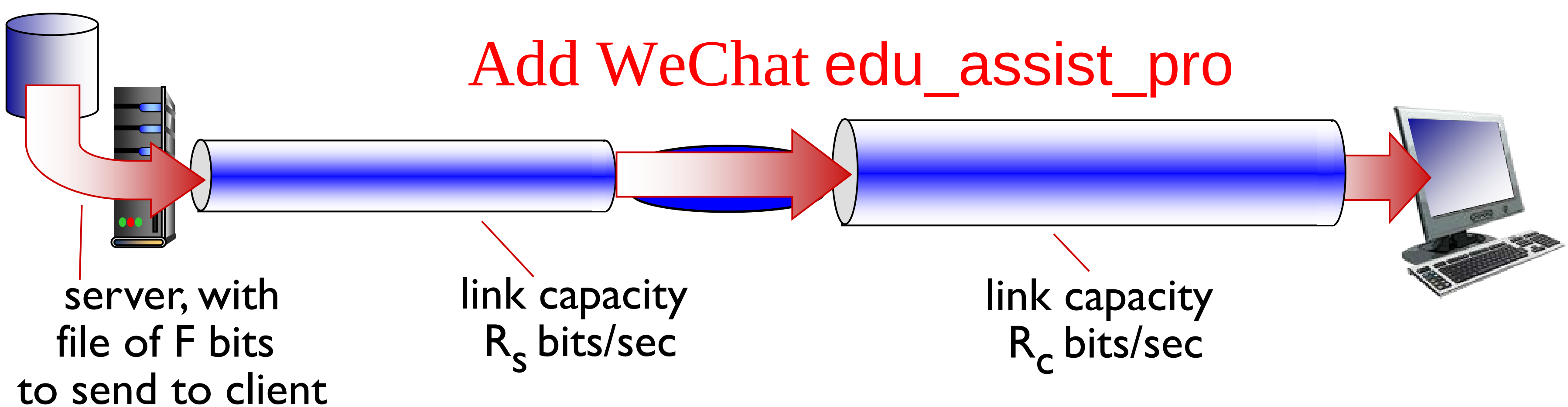
School of Computer Science

THE UNIVERSITY OF SYDNEY

Network Performance:

Throughput

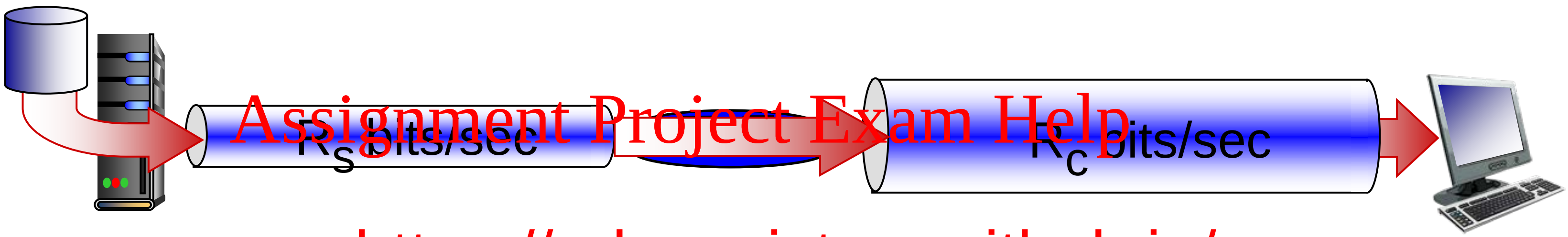
- *throughput:* rate (bits/time unit) at which bits transferred between sender/receiver
  - *instantaneous:* rate at given point in time
  - *average:* rate over long



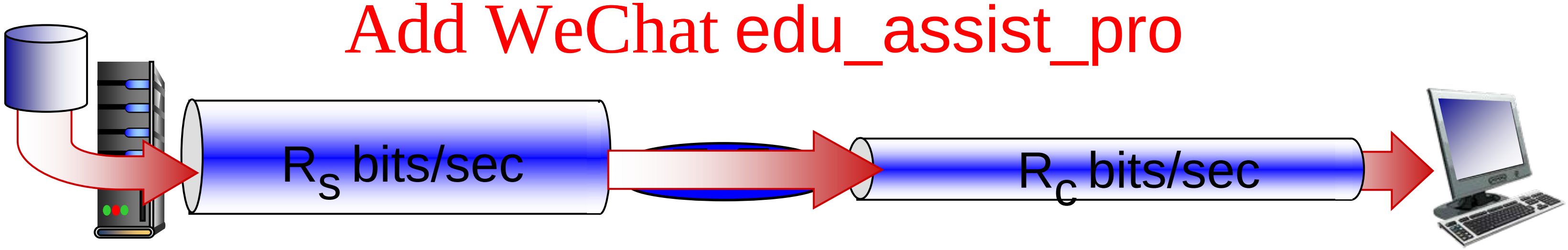
THE UNIVERSITY OF SYDNEY

Throughput (cont'd)

› $R_s < R_c$ What is average end-end throughput?



› $R_s > R_c$ What is



bottleneck link

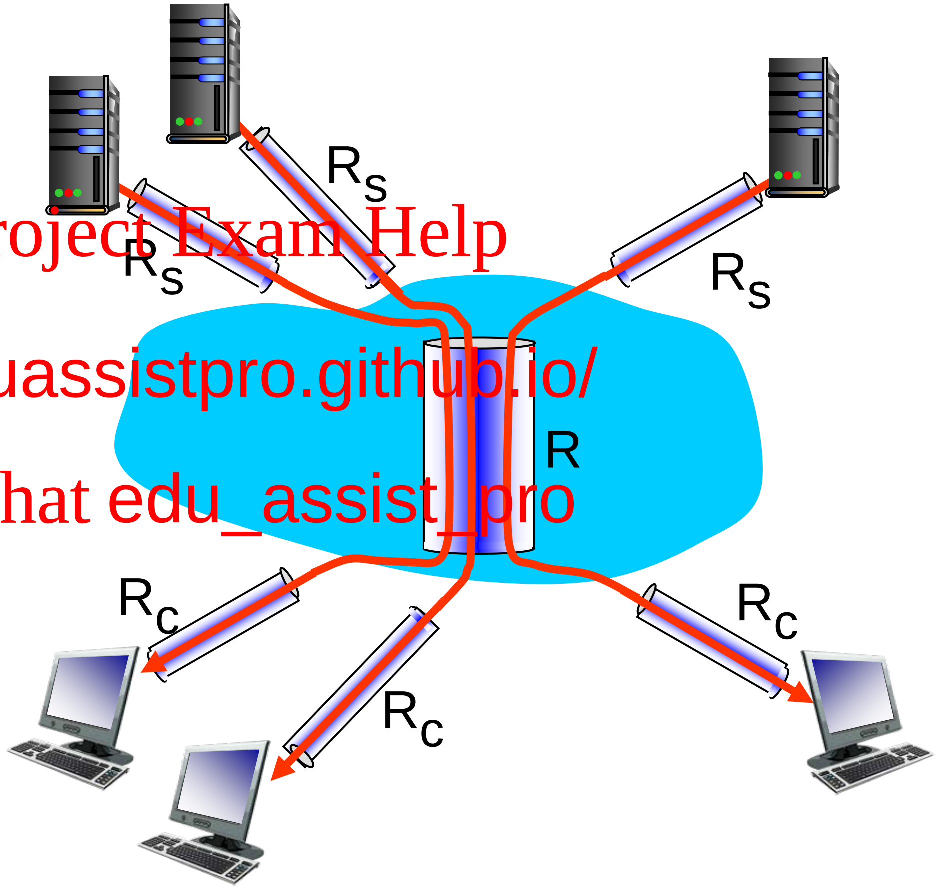
link on end-end path that constrains end-end throughput

THE UNIVERSITY OF SYDNEY

Throughput (cont'd)

Internet Scenario

- per-connection end-end throughput: $\min(R_c, R_s, R/10)$



10 connections (fairly) share backbone bottleneck link R bits/sec

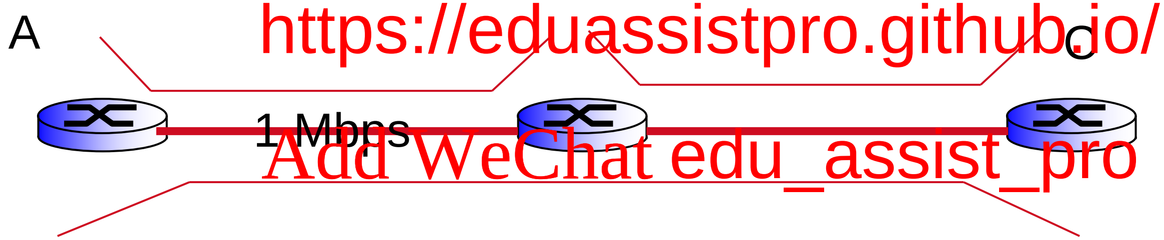
Network Performance:

In reality: two considerations

- Efficiency
- Fairness
- However, they

Network Fairness, Bandwidth allocation

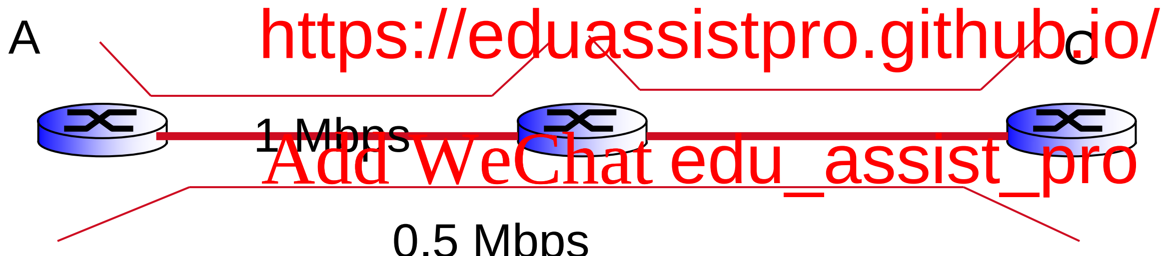
Three flows: A-B, B-C, A-C



Q: How can we allocate the link bandwidths to the three flows?

Network Fairness, Bandwidth allocation

Three flows: A-B, B-C, A-C

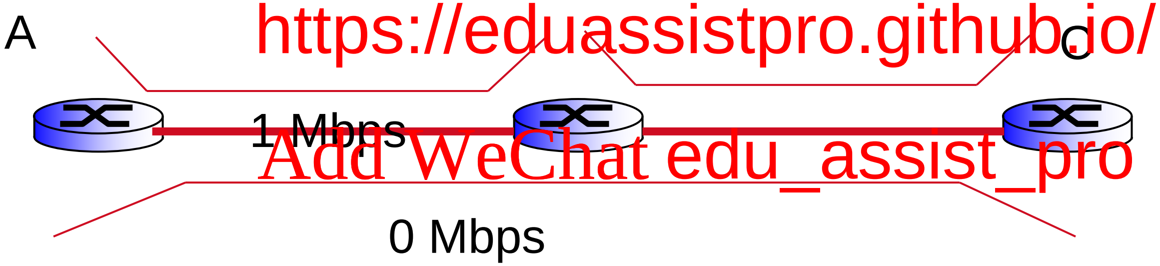


Very fair!

However: Network throughput, only 1.5Mbps

Network Fairness, Bandwidth allocation

Three flows: A-B, B-C, A-C

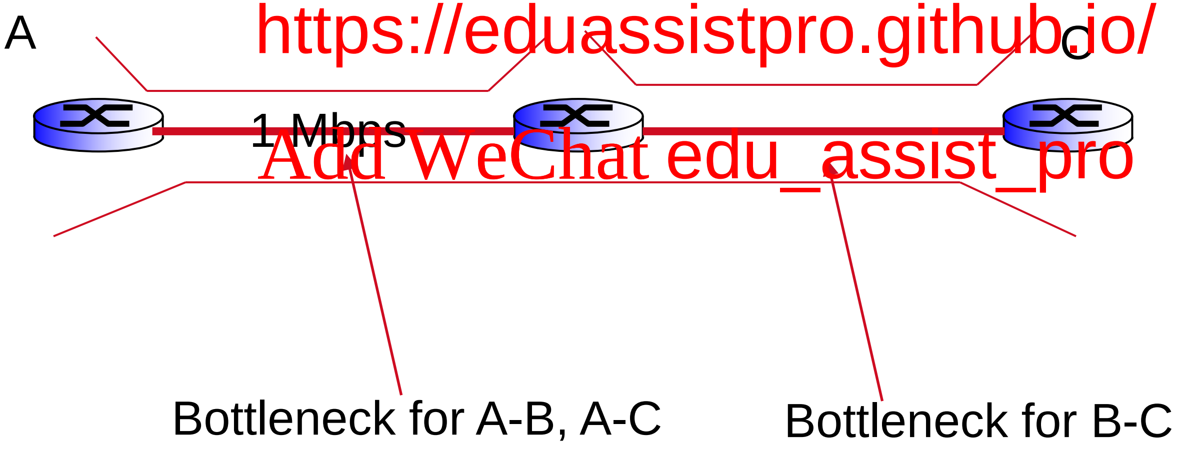


Very unfair!

However: Network throughput, 2Mbps

Fairness

Bottleneck for a flow: The link that limits the data rate of the flow



Max-min Fairness

- Maximize the minimum
- Try to increase the “poorest” as much as possible
  - A richer flow can be sacrificed.

- Try to increase t s much as possible

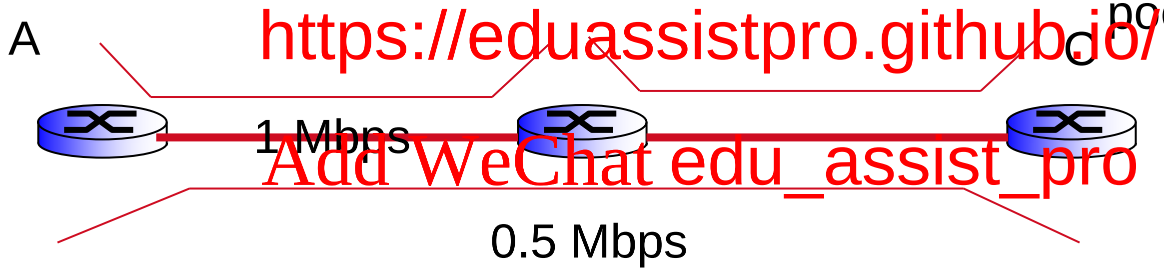
  - A richer flow can
  - A poorer flow cannot be sacrific

- Try to increase the third “poorest” as much as possible
- …
- Max-min Fairness criteria: if we want to improve one flow, we can only achieve this by sacrificing a poorer or equal flow.

Max-min Fairness

Bottleneck for a flow: The link limits its data rate

Even this is large, but it does hurt poorer flows



Bottleneck approach

Bottleneck approach

› 1 Start with all zero flows, potential flow set = {all flows}

› 2 Slowly increase flows in the potential flow set until there is a (new) link saturated

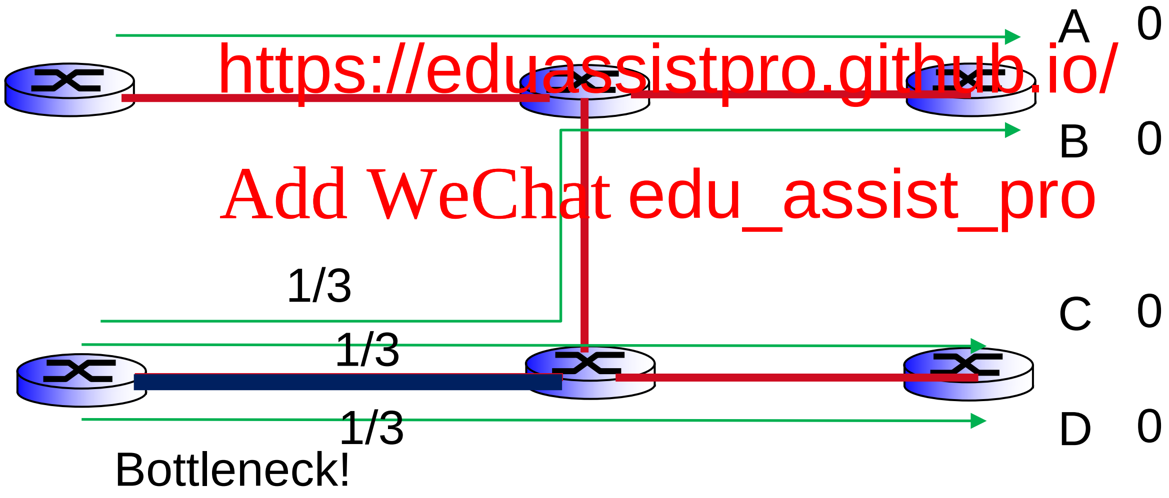
- “Pouring water in the n

› 3 Hold fix the flows th them from the potential flow set

› 4 If potential flow set is not empty, go to as potential to increase)

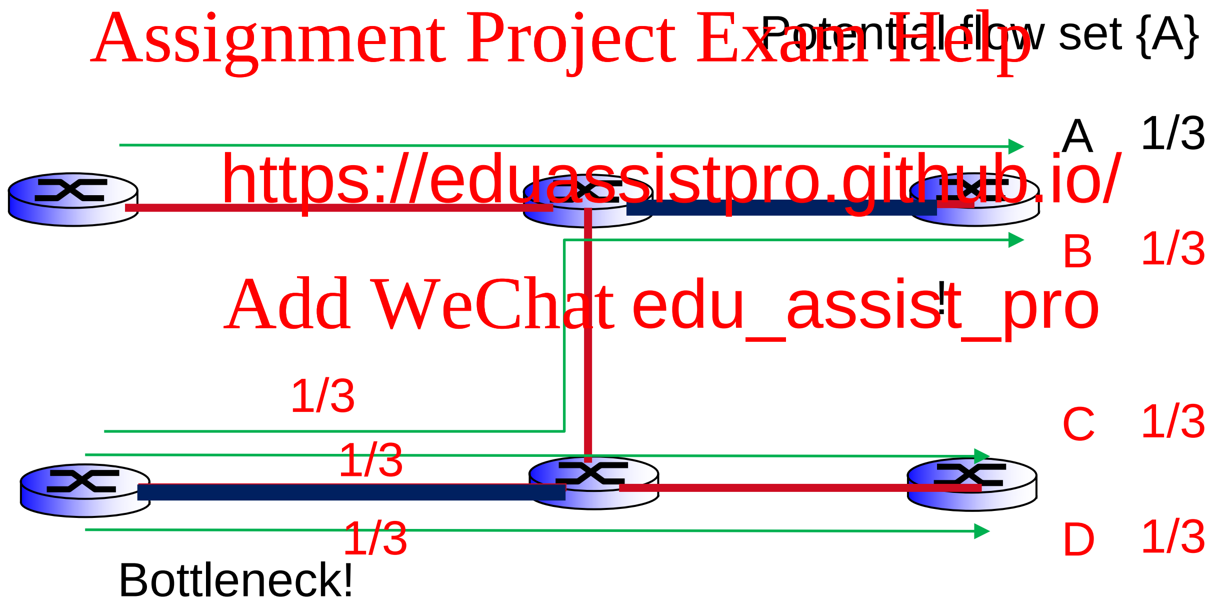
Each link between two routes with capacity 1

Potential flow set {A, B, C, D}



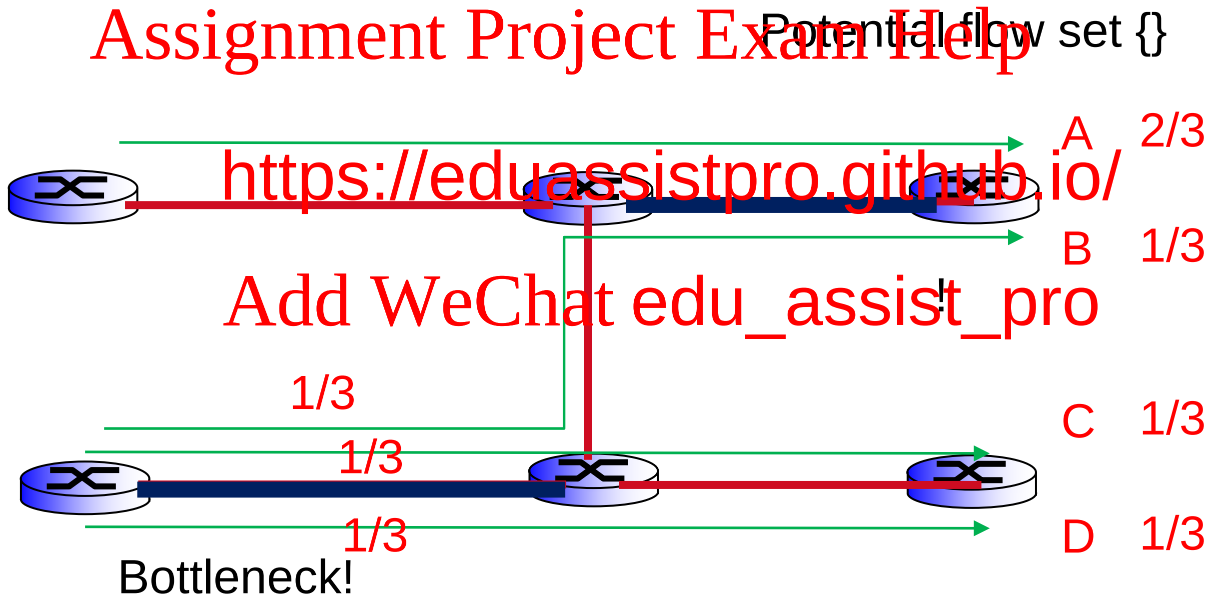
Bottleneck approach

Each link between two routes with capacity 1



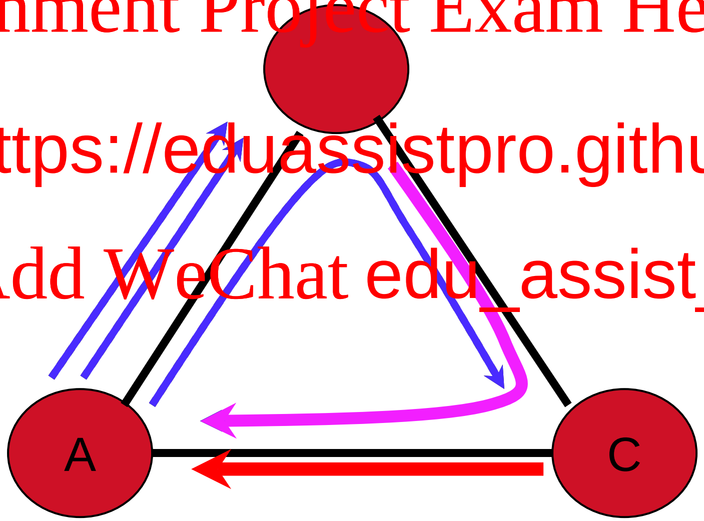
Bottleneck approach

Each link between two routes with capacity 1

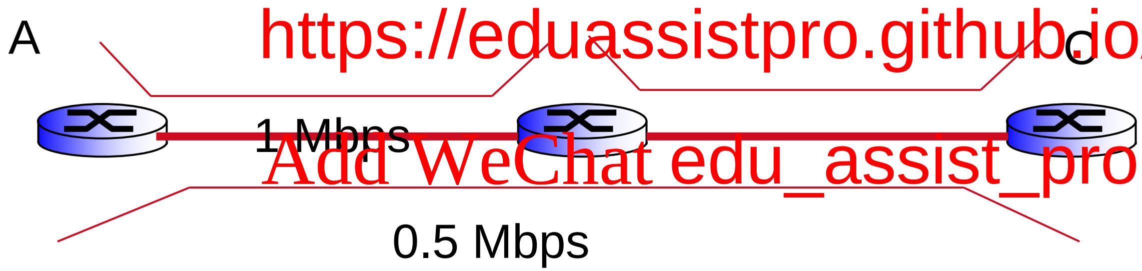


Can you solve the following problem?

link rate: AB=BC=1, CA=2



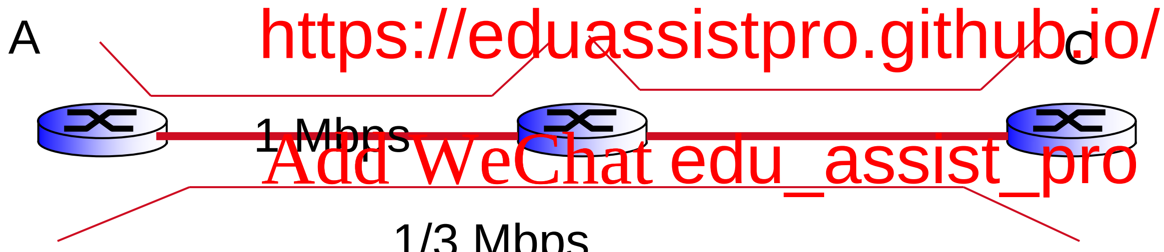
More comment: Max-min fairness is too fair!



You are using two links. How can we get a same share?

More comments

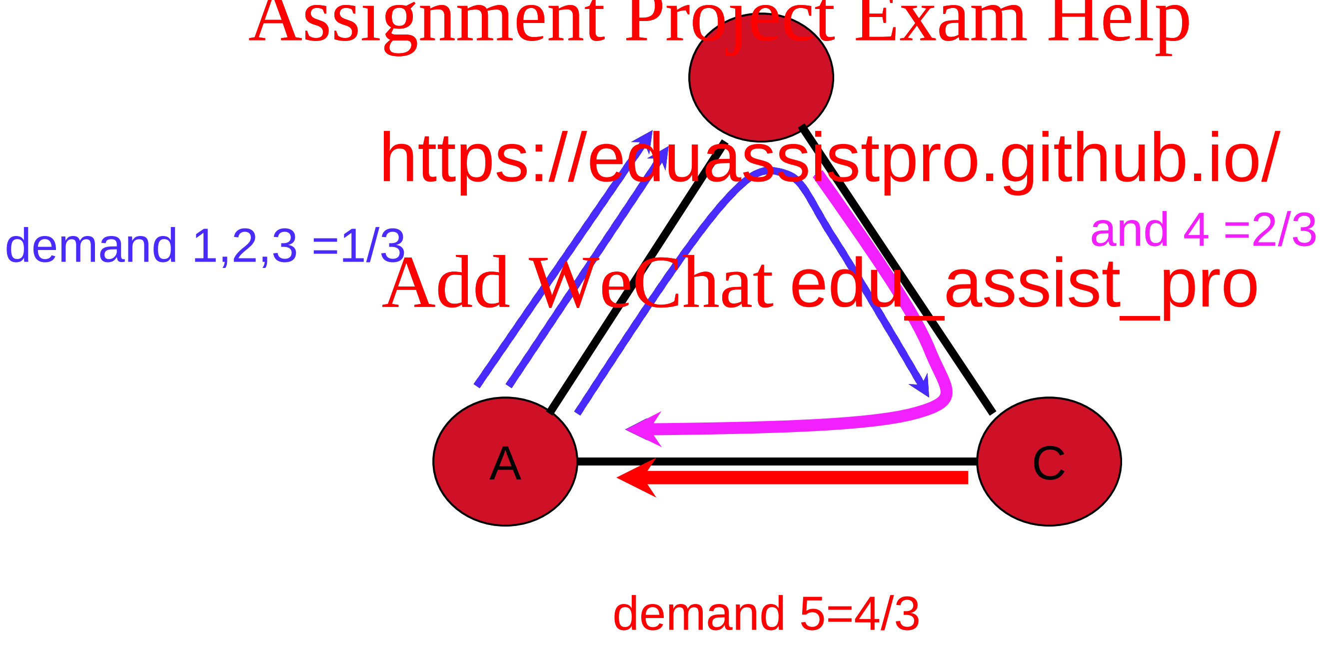
Another form of fairness
proportional fairness



Longer routes are penalized

Can you solve the following problem?

link rate: AB=BC=1, CA=2



The Application Layer

Some network applications

- e-mail
- web
- text messaging
- remote login
- P2P file sharing
- multi-user network games
- streaming stored video (YouTube, Netflix)
- voice over IP (e.g., Skype)
- real-time video conferencing
- social networking
-

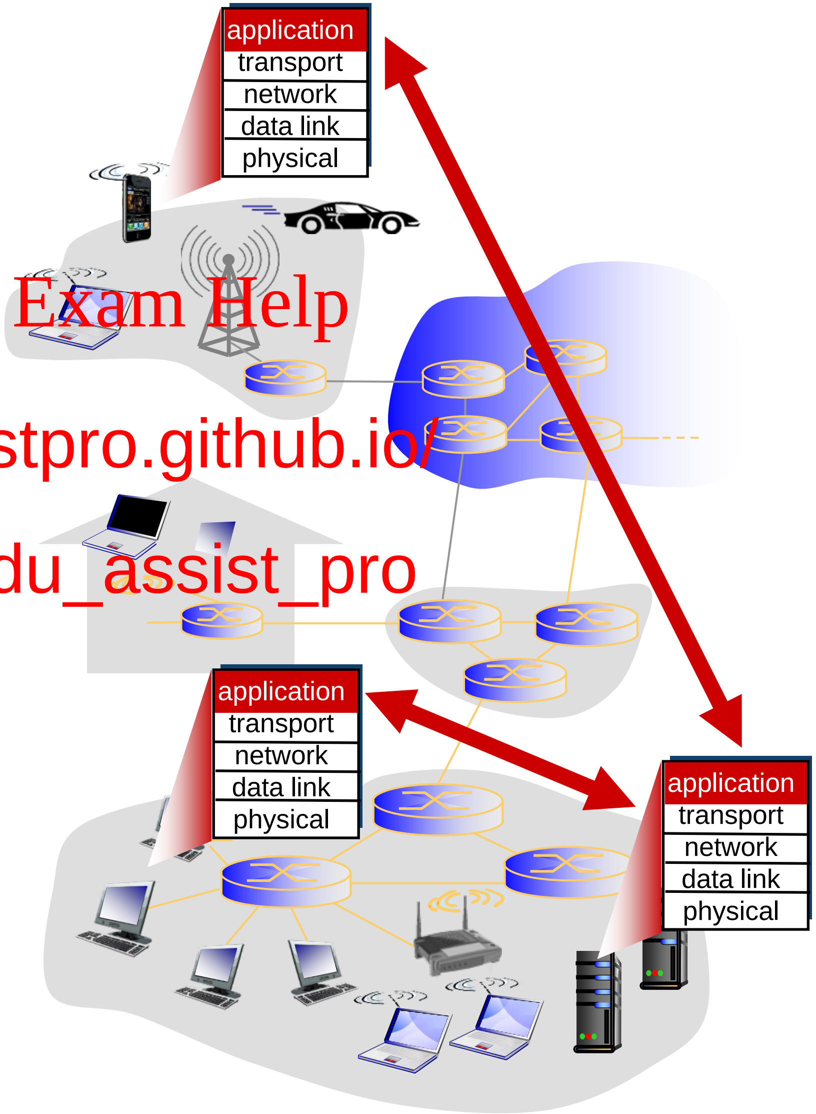
Creating a network app

write programs that:

- run on (different) *end systems*
- communicate over network
- e.g., web server software communicates with bro

no need to write software for network-core devices

- network-core devices do not run user applications
- applications on end systems allows for rapid app development, propagation

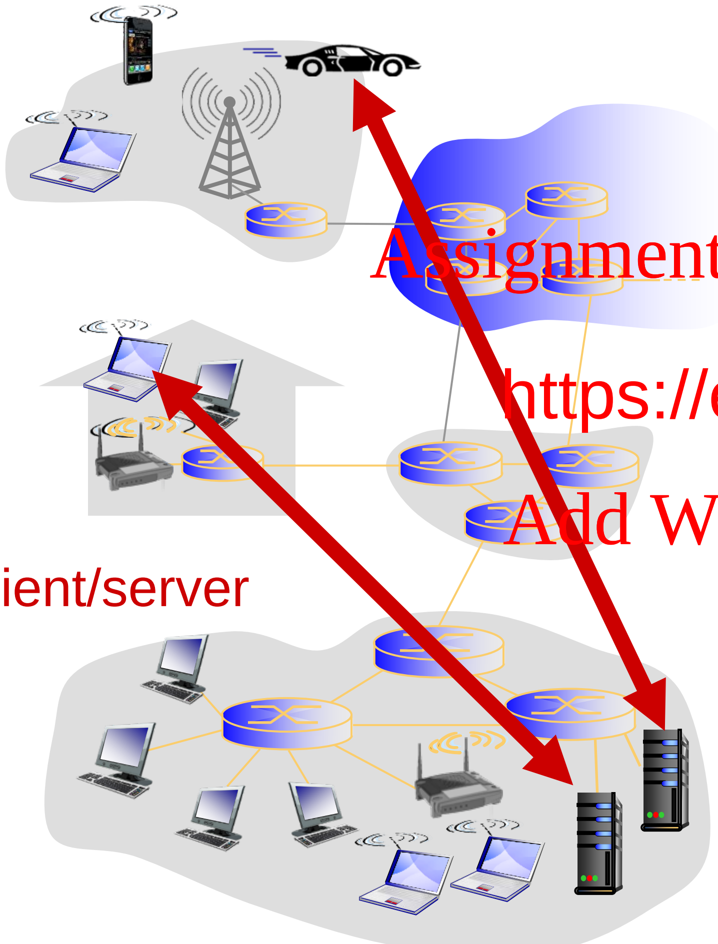


Possible structure of applications

› Client-server

› Peer-to-peer (P2P)

Client-server architecture



client/server

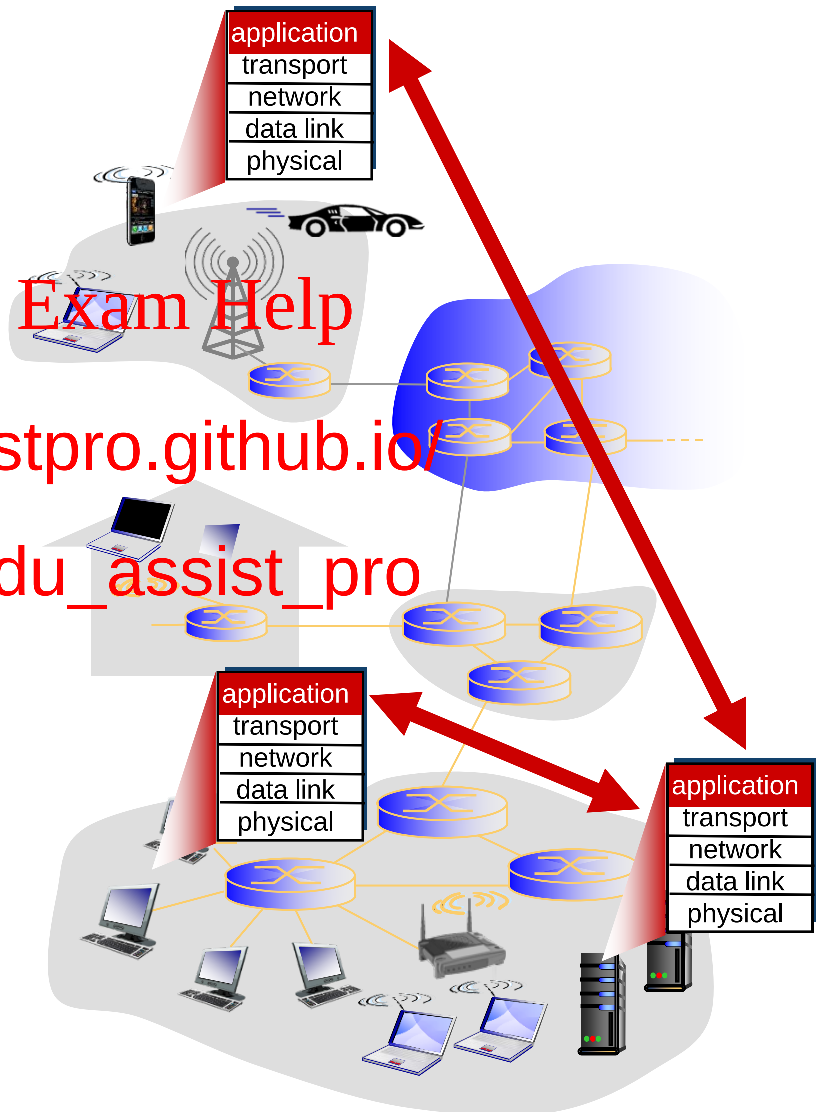
server:

- always-on
- permanent IP address
- rs for scaling

- e with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provid return to other pee
  - *self scalability* – new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses
  - complex management



Process communicating

process: program running within a host

- within same host, two processes comm
  inter-process co
  (defined by OS)
- processes in different hosts communicate by exchanging messages

clients, servers

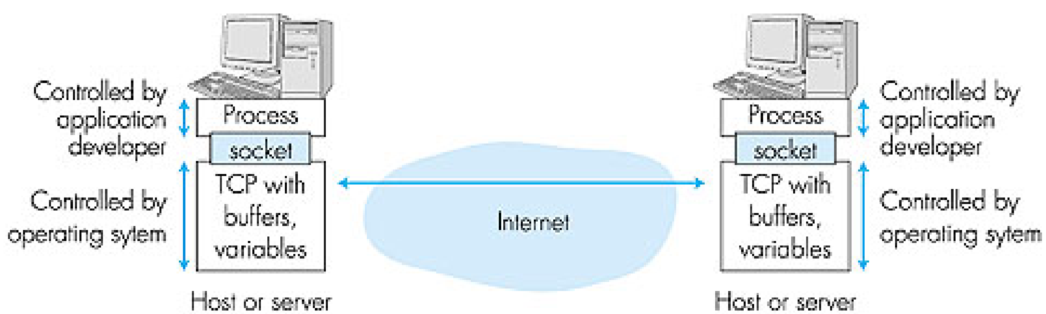
client process: process that initiates communication

ocess: process that
be co

- aside: applications with P2P architectures have client processes & server processes

Sockets

- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - sending process tructure on other side of door to at receiving process



Addressing processes

- to receive messages, process must have *identifier*
- host device has unique 32-bit IP address (or 128 in IPv6)
- *Q:* does IP addres which process runs suffice for identifying the process?
  - *A:* no, *many* processes can be running on same host

- *identifier* includes both IP address and port numbers associated with process on host.

port numbers:

ver: 80

25

- to send HTTP message to gaia.cs.umass.edu web server:
  - IP address: 128.119.245.12
  - port number: 80
- more shortly…

App-layer protocol defines

- types of messages exchanged,
  - e.g., request, response
- message syntax:
  - what fields in messages & how fields are delineated
  - e.g. First line: method.
  - message semantics
  - meaning of information in fields
  - e.g. 404 means "not found"
- rules for when and how processes send & respond to messages

open protocols:

- defined in RFCs
- allows for interoperability
- e.g., HTTP, SMTP

protocols:

What transport service does an app need?

data integrity

- some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
- other apps (e.g., au tolerate some loss

timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"

throughput

- some apps (e.g., multimedia) require minimum amount of throughput to be tive "
- apps ("elastic apps") ver put they get

Internet transport protocols services

TCP service:

- *reliable transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when net overloaded
- *does not provide:* timing, minimum throughput guarantee
- *connection-oriented:* setup required between client and server processes

UDP service:

- *unreliable data transfer* between sending and receiving process
- *...vide:* reliability, ...rol, cong... ...ing, throughput ... or connection setup,

Internet apps: application, transport protocols

application	application layer protocol	underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web		TCP
file transfer		TCP
streaming multimedia	HTTP RTP [RFC 18	CP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	TCP or UDP

Web and HTTP

First, a review…

- web page consists of *base HTML-file* which includes *several referenced objects*
  - *HTML: HyperText Markup Language*
- object can be JPEG i
- each object is addressable by a *URL (Unif ocator)*, e.g.,

$$\underbrace{\texttt{www.someschool.edu}}_{\text{host name}}\underbrace{\texttt{/someDept/pic.gif}}_{\text{path name}}$$

File: usually base-html file
(HyperText Markup Language)

xxxxxxx
www.aaa.edu/
yyyyyyyyyyy
www.aaa.edu/Obj2.jpg
zzzzzzzzz

Browser shows

xxxxxxxxx

yyyyyyyyy

zzzzzzzzz

HTTP overview

HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, (using HTTP protocol) and "displays" Web objects
  - *server:* Web server sends (using HTTP protocol) objects in response to requests

HTTP overview (cont'd)

uses TCP:

- client initiates TCP connection (creates socket) to server, port 80
  - How to know IP ad
  - DNS (Domain Nam
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

HTTP is "stateless"

- server maintains no information about past client requests

aside

Is that
are complex!

- y (state) must be maintained
- if server/client crashes, their views of "state" may be inconsistent, must be reconciled

HTTP connections

non-persistent HTTP

- at most one object sent over TCP connection
  - connection th
- downloading multiple objects required multiple connections

persistent HTTP

- multiple objects can be sent over single TCP n between c er

THE UNIVERSITY OF SYDNEY

Non-persistent HTTP

suppose user enters URL:

www.someSchool.edu/someDepartment/home.index

(contains text, references to 10 jpeg images)

1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80

1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client

2. HTTP client sends HTT

message into TCP connection socket. Message indicates that client wants page someDepartment/home.index

eceives request message, *se message* containing requested page, and sends message

5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects to download

4. HTTP server closes TCP connection.

time

Non-persistent HTTP

suppose user enters URL:

www.someSchool.edu/someDepartment/home.index

(contains text, references to 10 jpeg images)

1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80

1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client

2. HTTP client sends HTT

message into TCP connection socket. Message indicates that client wants object someDepartment/object1.jpg

eceives request message, se *message* containing requested object, and sends message

5. HTTP client receives response message containing object, displays the object.

4. HTTP server closes TCP connection.

time ↓

6. Steps 1-5 repeated for each of 10 jpeg objects

HTTP: response time

RTT (definition): time for a small packet to travel from client to server and back

HTTP response time:

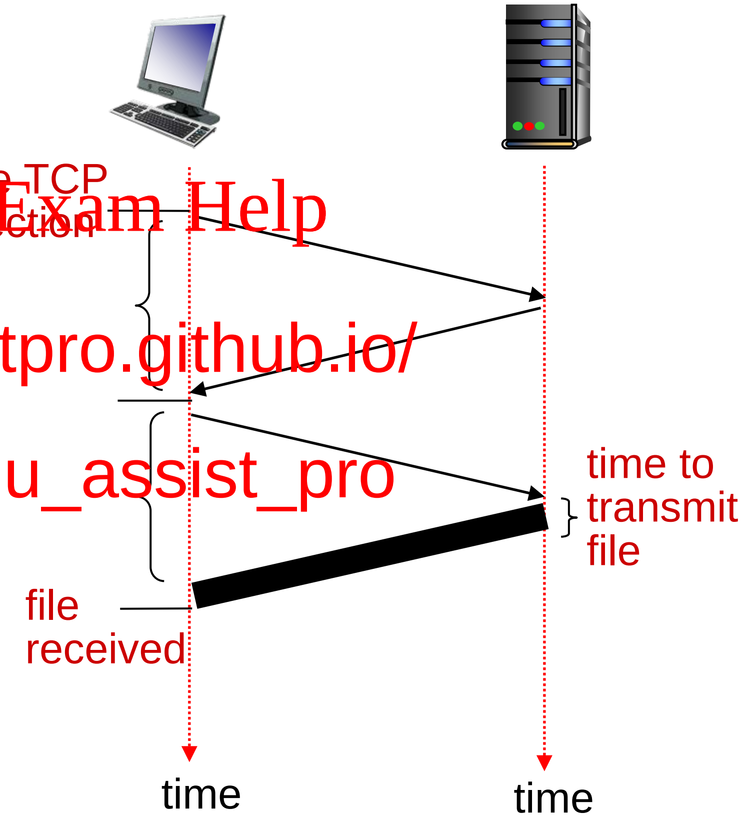
› one RTT to initiate TCP connection

› one RTT for HTTP request and first few bytes of HTTP response to return

› file transmission time

› non-persistent HTTP response time =

  2RTT+ file transmission time



Persistent HTTP

suppose user enters URL:
www.someSchool.edu/someDepartment/home.index

(contains text, references to 10 jpeg images)

1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80

1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. “accepts” connection, notifying client

2. HTTP client sends HTT *message* into TCP connection socket. Message indicates that client wants page someDepartment/home.index

eceives request message, se *message* containing requested page, and sends message

5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects to download

TCP is still on

time

Persistent HTTP

suppose user enters URL:
www.someSchool.edu/someDepartment/home.index

(contains text, references to 10 jpeg images)

2. HTTP client sends HTT*P request message* into TCP connection socket. Message indicates that client wants object someDepartment/object1.jpg

...eceives request message, ...se *message* containing requested object, and sends message

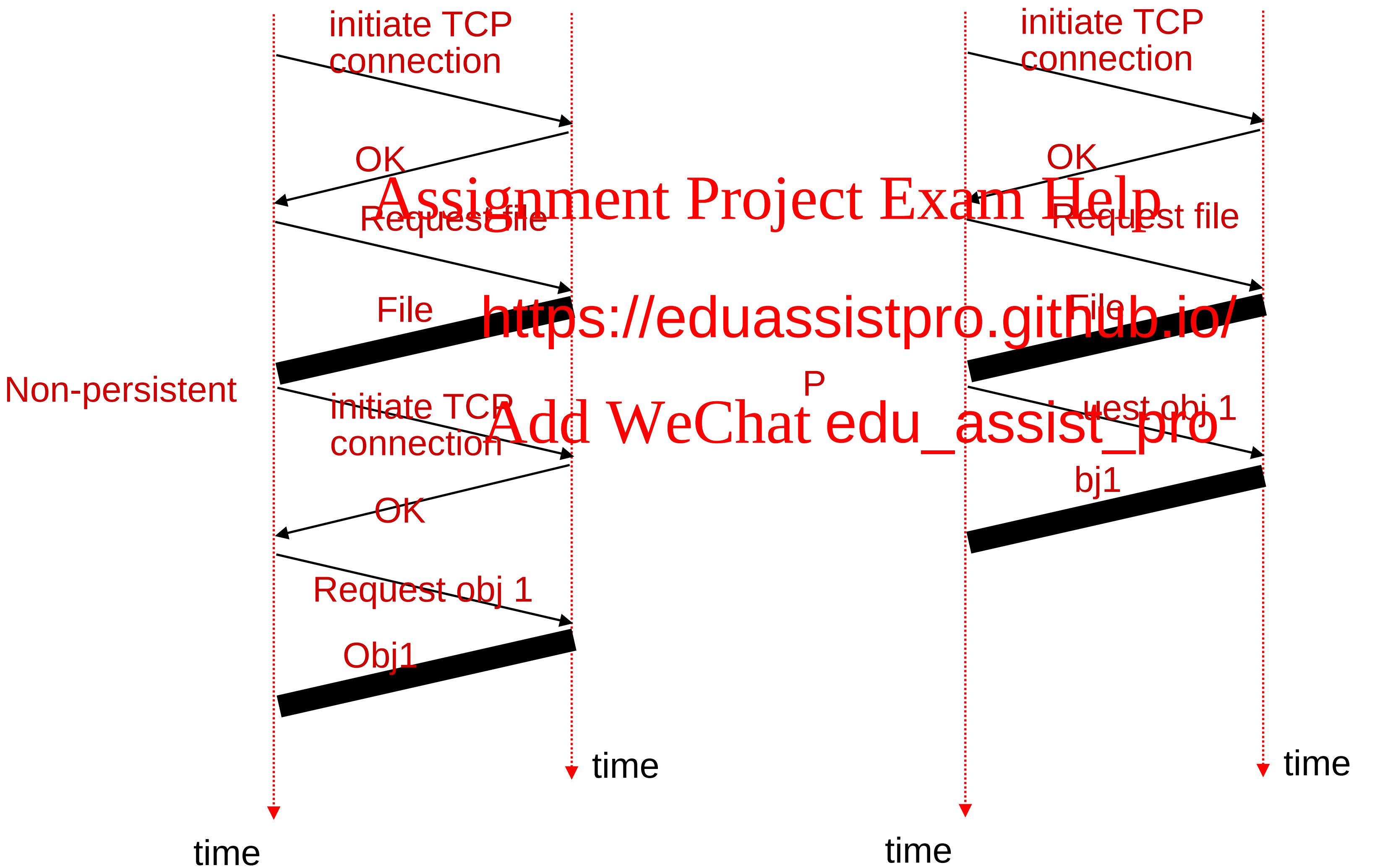
4. HTTP client receives response message containing object, displays the object.

Repeated for each of 10 jpeg objects

10 rounds later HTTP server closes TCP connection.

time

Non-persistent vs. persistent



Persistent HTTP

non-persistent HTTP issues:

- requires 2 RTTs + file transmission time per object

persistent HTTP:

- server leaves connection open after sending response
- nt HTTP messages same client/server open co
- requests as soon as it encounters a referenced object
- as little as one RTT + file transmission time for all the referenced objects

HTTP request message

- two types of HTTP messages: *request*, *response*
- HTTP request message:
  - ASCII (human-readable format)

carriage return character

line-feed character

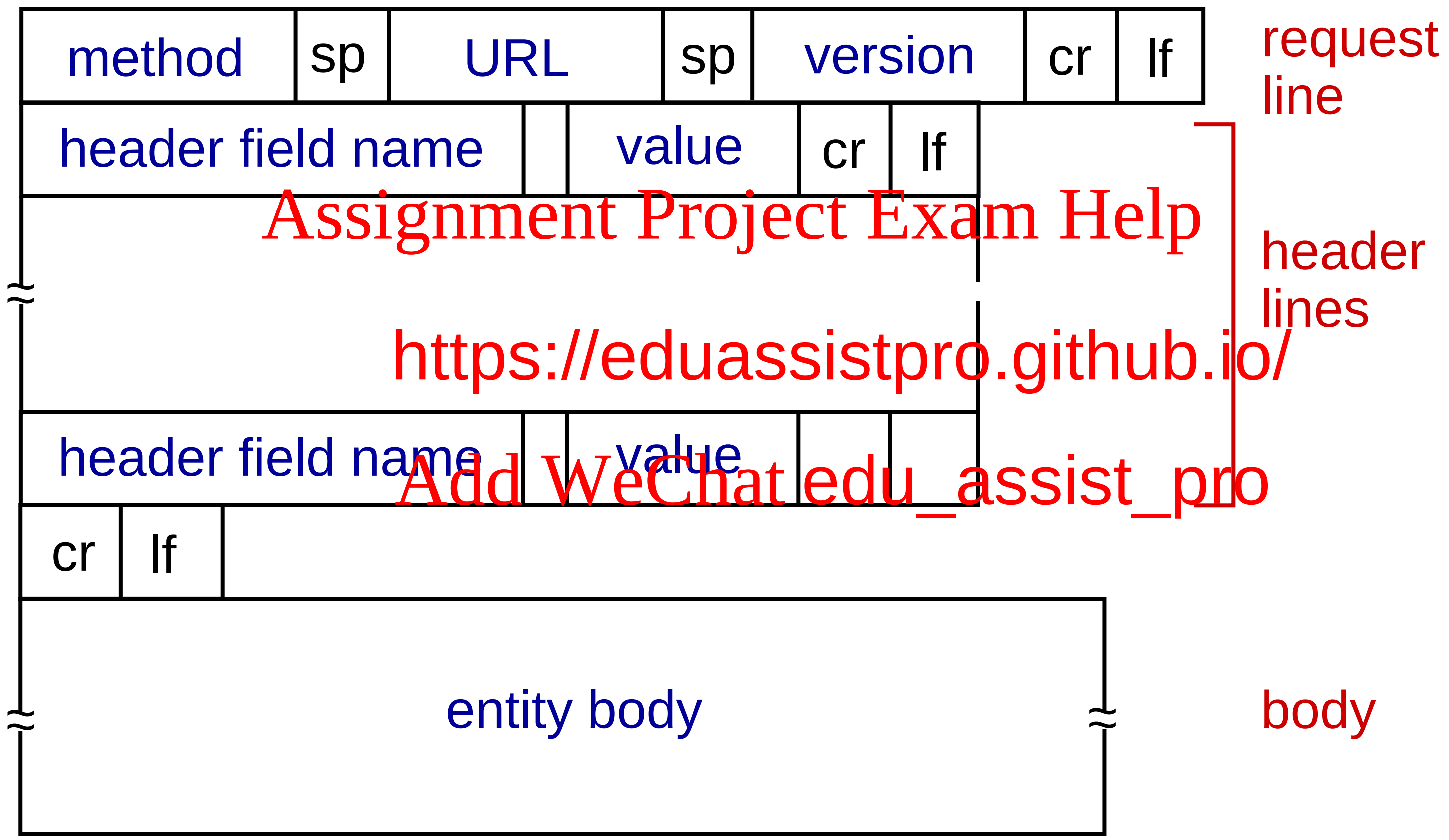
request line
(GET, POST,
HEAD commands)

header
lines

carriage return,
line feed at start
of line indicates
end of header lines

```
                        1\r\n
Host: www-net.          u\r\n
User-Agent: Fi          0\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

HTTP request message: general format



GET method

POST method:

- web page often includes form in
- input is uploaded to server in entity body

HTTP/1.0:

› GET

› POST

› HEAD

- asks server to leave requested object out of response

HTTP/1.1:

› GET, POST, HEAD

› PUT

- le in entity body
  ified in URL

› DELETE

- deletes file specified in the URL field

THE UNIVERSITY OF SYDNEY

HTTP response message

status line (protocol status code status phrase)

header lines

data, e.g., requested HTML file

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last                           2007 17:00:02
   G
ETag               "              \n
Accept-Ranges: byt
Content-Length: 26
Keep-Alive: timeou            00\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
   1\r\n
\r\n
data data data data data ...
```

HTTP response status codes

› status code appears in 1st line in server-to-client response message.

› some sample codes:

200 OK

- request succeed his msg

301 Moved P

- requested object moved, new loca ater in this msg (Location:)

400 Bad Request

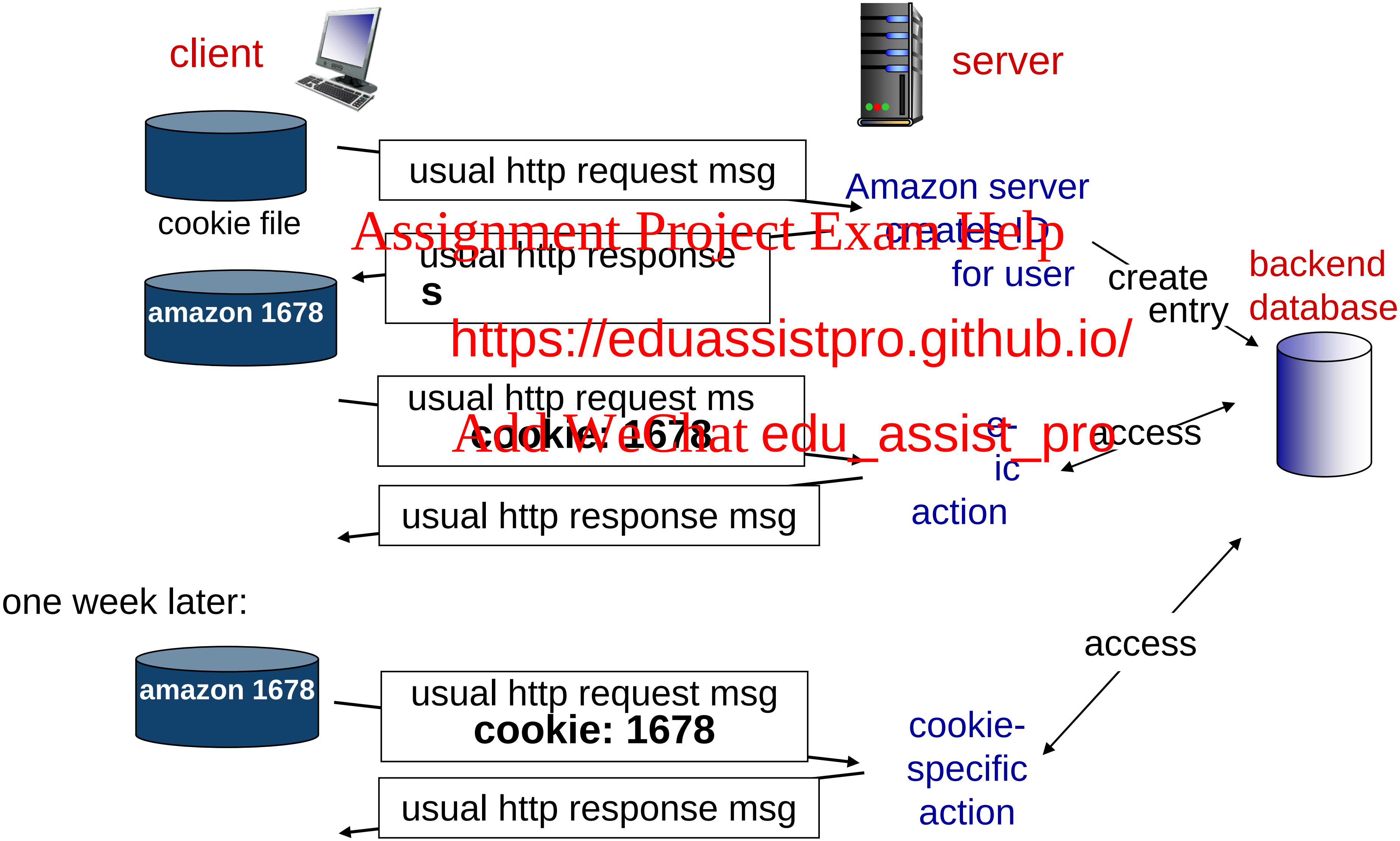
- request msg not understood by server

404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

Cookies: keeping “state” (cont’d)



User-server state: cookies

many Web sites use cookies

four components:

1) cookie header line of HTTP *response* message

2) cookie header message

3) cookie file kep by user's browser

4) back-end database at Web sit

Cookies (cont'd)

what cookies can be used for:

- authorization
- shopping carts
- recommendations
- user session state

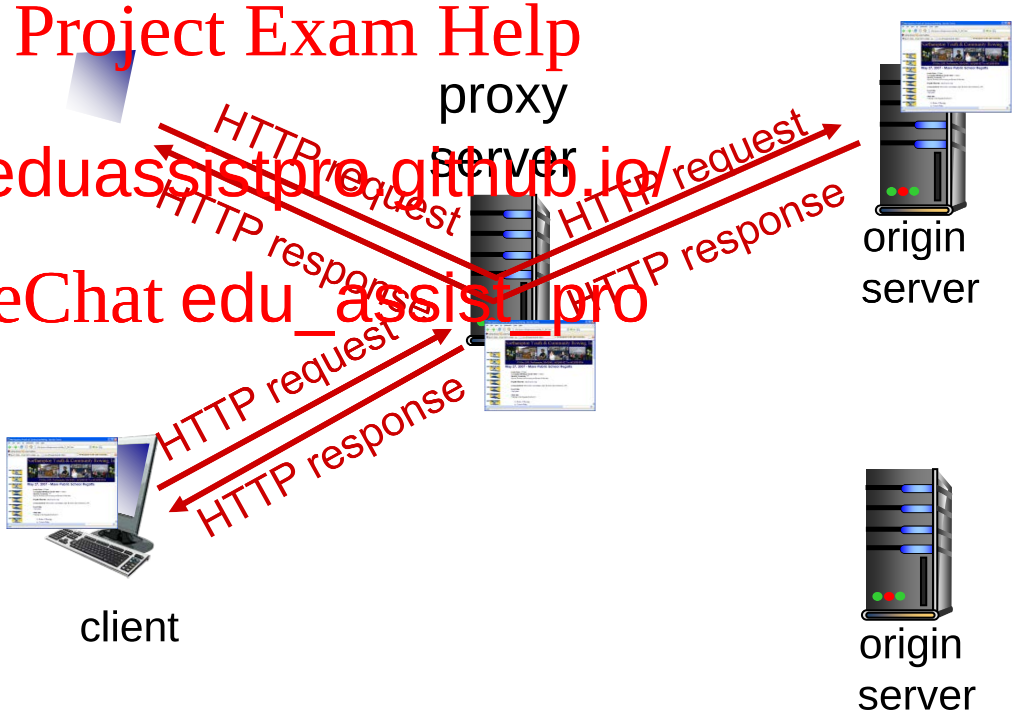
how to keep "state":

- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: http messages carry state

Web caches (proxy server)

goal: satisfy client request without involving origin server

- user sets browser: Web accesses via cache
- browser sends all HTTP requests to cache
- if object in cache:
  - then cache returns object
  - else cache requests object from origin server, then returns object to client



More about Web caching

- Q: Does the cache act as a client or a server?

More about Web caching

- R: cache acts as both client and server
  - server for original requesting client
  - client to origin serv
- typically cache is installed by ISP (university, company, residential ISP)

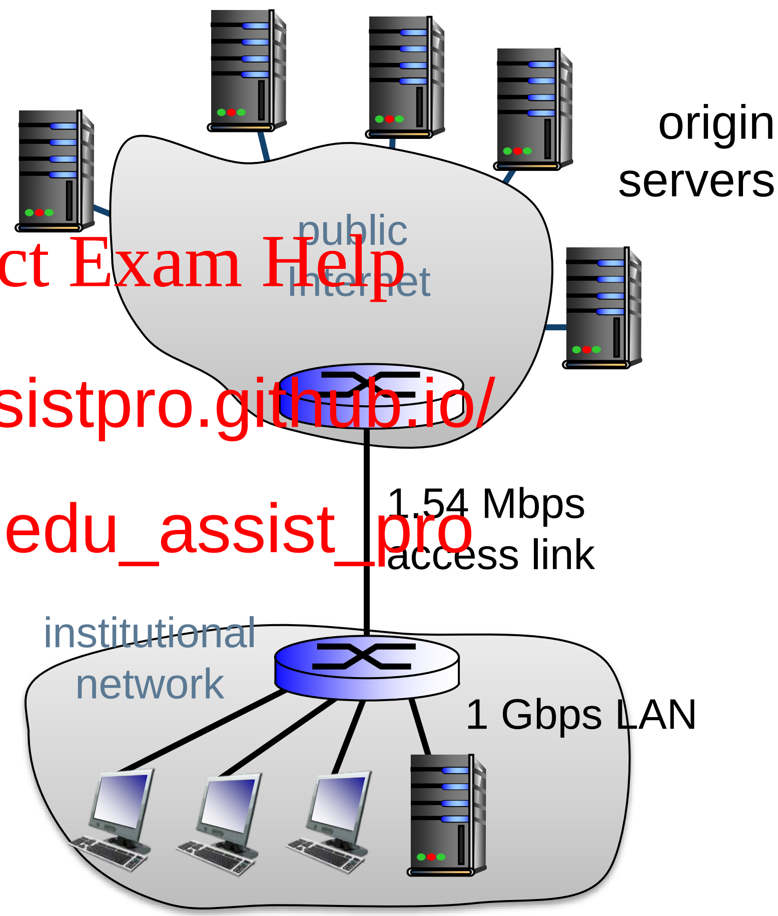
why Web caching?

- reduce response time for client request
- affic on an i access link

Caching example

assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers:15/sec (1.5 Mbps service)
- RTT from institutional router to any origin server: 2 sec
- access link rate: 1.54



consequences:

- LAN utilization: 0.15%
- LANU = avg req rate * size / link bandwidth
- access link utilization = 99% *problem!*
- ALU = avg req rate * size / link bankwidth
- total delay = 2 sec + minutes + usecs

Q: what happens with fatter access link?

Caching example: fatter access link

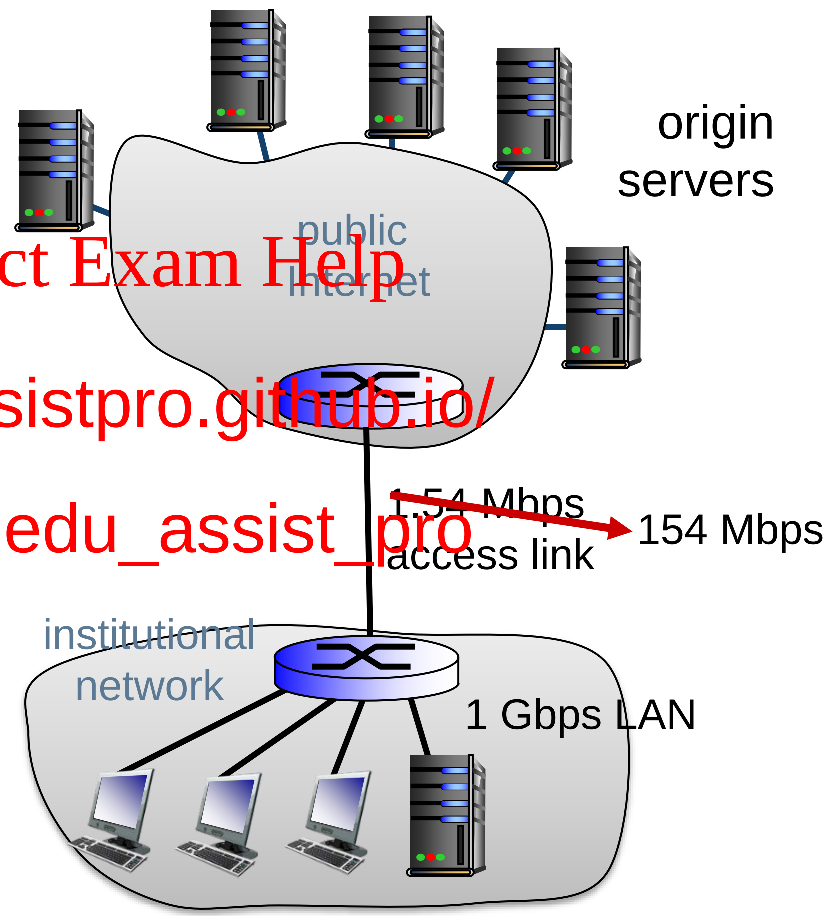
assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers: 15/sec
- RTT from institutional router to any origin server: 2 sec
- access link rate: 1.54

consequences:

- LAN utilization: 0.15%
- access link utilization = 99% 0.99
- total delay = 2 sec + minutes + usecs

msecs



Cost: increased access link speed (not cheap!)

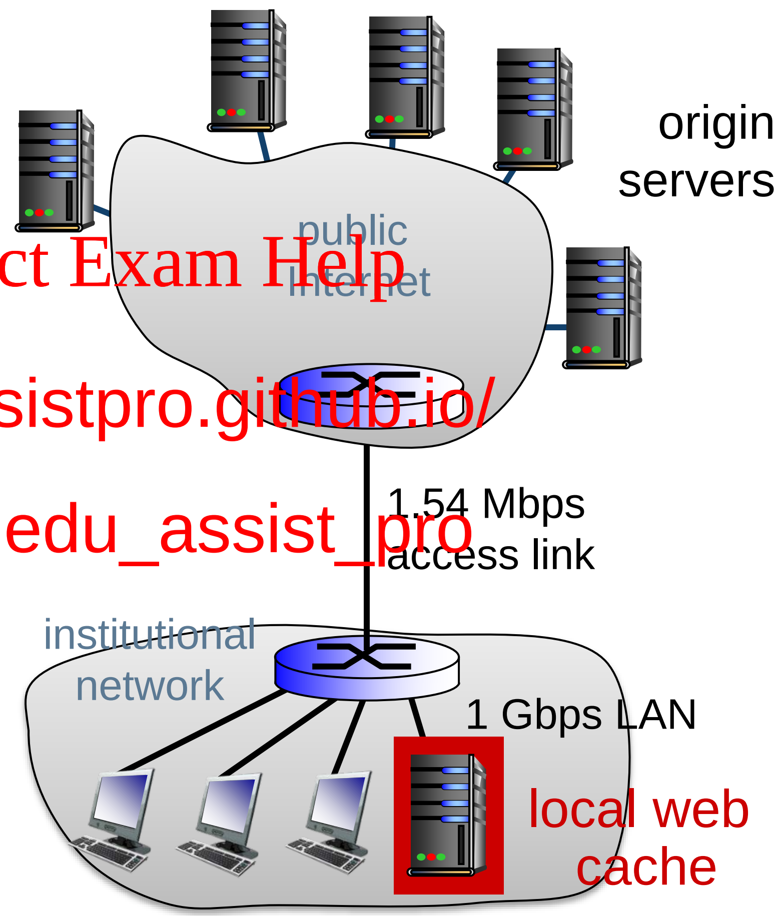
Caching example: install local cache

assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers:15/sec
- RTT from institutional router to any origin server: 2 sec
- access link rate: 1.54

consequences:

- LAN utilization: 0.15%
- access link utilization = 0%
- total delay = usecs



Cost: web cache (cheap!)

Caching example: install local cache

Calculating access link utilization, delay with cache:

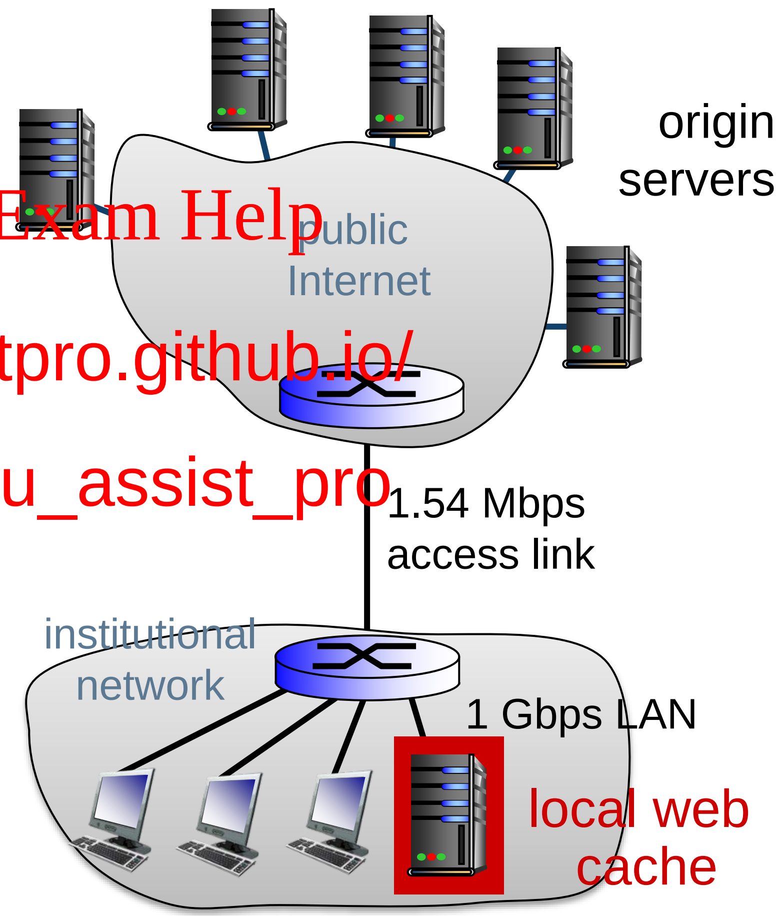
- suppose cache hit rate is 0.4
  - 40% requests satisfied at cache,
  - 60% requests satisfied at origin
- access link utilizatio
  - 60% of requests use acce
- average total delay

= 0.6 * (delay from origin servers) +0.4 * (delay when satisfied at cache)

Link utilization is around 60%, queueing delay is small enough

= 0.6 (~2.x second) + 0.4 (~usecs)

less than with 154 Mbps link (and cheaper too!)



Conditional GET

- *Goal:* don't send object if client has up-to-date cached version
  - no object transmission delay
  - lower link utilization
- *client:* specify date of cached copy in HTTP request

  `If-modified-since: <date>`
- *server:* response contains no object if cached copy is up-to-date:

  `HTTP/1.0 304 Not Modified`