AĞ PROGRAMLAMA

DERS 5: Multi-Thread Sunucu

Giriş

- Şu ana kadar gördüğümüz sunucu kodlamamızda temel bir sıkıntı bulunmaktadır.
  - Sadece bir kullanıcının isteğine cevap verebilir.
  - Eş zamanlı olarak gelen istekleri değerlendiremez.
- Gerçek hayat uygulamaları için kabul edilemez. Çözümler:
  - Non-Blocking Server kullanımı (J2SE 1.4 sonrası)
  - Multhithreaded Server kullanımı

Multi-thread veya Non-Blocking Server kullanımının getirdiği avantajlar

- Her bağlantı için yapılacak işlemleri birbirinden ayrı task'lar olarak yürütür. (clean implementation)
- Robust (stabil) bir yapı sunar. Bir bağlantıda bir problem olsa dahi sistem çalışmaya devam edebilir. İki Kademeli olarak bir yapı kuracağız:
  - Main thread gelen istemci bağlantıları için bağımsız threadler oluşturacak (main methodu)
  - Oluşturulan her bağımsız thread ilgili client ile olan haberleşmeyi yürütecek. (ilgili threadin run methodu)
  - Ör: MultiEchoServer – MultiEchoClient : Chapter3
  - Ör: ResourceServer,Producer,ClientThread,Resource (Thread Synchronisation)

Non-Blocking Servers

- J2SE 1.4'den önce Non-Blocking yapıyı simule etmek için InputStream'lerde available methodu kullanılıyordu.
- int available() throws IOException
- Bu method ağ bağlantısına bağlanmış bir InputStream'de gelmiş olan fakat henüz okunmamış byte miktarını döndürür.
- Bir porttaki bağlantıları non-blocking çalıştırmak için her bir istemciyi ayrı bağlantı kurup döngüsel olarak avaliable methodu ile veri olup olmadığı kontrol edilir. (çok uygulanan bir yöntem değildir.)

Non-Blocking Servers

- J2SE 1.4 ile New Input/Output API, NIO.
- package: java.nio
- java.nio.channels
  (klasik streamler yerine tercih edeceğiz streamler byte-oriented çalışırken channellar block-oriented çalışır.)
- Birden fazla kullanıcıyı karşılayabilmek için NIO multiplexing(birden fazla bağlantıyı tek yapı ile kurma) kullanır.
- Bu işlem selector (tek bir yapı) ile yeni bağlantıları gözleme ve var olan bağlantıları devam ettirme şeklinde yapılır.
- Channeller blocking veya non-blocking modda kullanılabilirler.
- Tek bir thread ile bir veya birden fazla channel'ı dinleyebiliriz. Bu sayede işletim sistemi ve sunucu limitlerine gelme problemini ortadan kaldırırız.
- Multi-thread yapılara göre daha karmaşık bir kodlama yapısı olmasına karşın avantajlıdır.

Non-Blocking Servers

- Socket ve ServerSocket için channellar SocketChannel , ServerSocketChannel
- Varsayılan ayar blocking modedur.
  - configureBlocking methodu false ile çağrılarak non-blocking kipine çevrilebilir.

```
serverSocketChannel = ServerSocketChannel.open();
serverSocketChannel.configureBlocking(false);
serverSocket = serverSocketChannel.socket();
.....................................................
//The lines below will occur rather later in the
//program, of course.
socketChannel = serverSocketChannel.accept();
socketChannel.configureBlocking(false);
socket = socketChannel.socket();
```

Non-Blocking Servers

- Sonrasında ServerSocket ayarlanabilir.

```
InetSocketAddress netAddress =
                              new InetSocketAddress(PORT);
serverSocket.bind(netAddress);    //Bind socket to port.
```

- Her channel'ın selector nesnesi ile register edilmesi gerekir. Bunun için SelectionKey'deki 4 adet static constant'dan biri register methoduna gönderilir.
  - SelectionKey.OP_ACCEPT
  - SelectionKey.OP_CONNECT
  - SelectionKey.OP_READ
  - SelectionKey.OP_WRITE

Non-Blocking Servers

```
selector = Selector.open();
serverSocketChannel.register(selector,
                             SelectionKey.OP_ACCEPT);
.....................................................
//The line below will occur rather later in the
//program, of course.
socketChannel.register(selector,
                             SelectionKey.OP_ READ);
```

- Son aşama olarak da verileri saklayacak bir buffer nesnesine ihtiyaç duyulur.

```
buffer = ByteBuffer.allocate(2048);
```

Non-Blocking Servers

- int numKeys = selector.select();
- Çağrılarak ilgili selectorde o ana kadar toplanmış eventlerin sayısı elde edilebilir.
- Eğer hiçbir event yok ise kod ilgili satırda en az bir event oluşana kadar bekler.
- Oluşan her bir event için SelectionKey (package java.nio.channels) adında bir nesne oluşturulur. Ve bunların bir

```
Set eventKeys = selector.selectedKeys();
Iterator keyCycler = eventKeys.iterator();
```

```
while (keyCycler.hasNext())
{
    SelectionKey key =
                (SelectionKey)keyCycler.next();
```

Non-Blocking Servers

```
int keyOps = key.readyOps();

if ((keyOps & SelectionKey.OP_ACCEPT) ==
                                SelectionKey.OP_ACCEPT)
{
    acceptConnection(key);          //Pass key to
                                    //processing method.
    continue; //Back to start of key-processing loop.
}
if ((keyOps & SelectionKey.OP_READ) ==
                               SelectionKey.OP_READ)
{
    acceptData(key); //Pass key to processing method.
}
```

- Ör: MultiEchoServerNIO

Uygulama

Çok kullanıcılı sunucu örneği. Karşılıklı mesajlaşma N-N. Bir kullanıcıdan gelen yazı sunucuya bağlı tüm kullanıcılara iletilecek. NIO kullanarak.

Web Uygulamaları

- Protocol Nedir?
  - Bir işi yapmak için karşılıklı mutabık kalınan metin.
- Ağ Programlamada Protokol tanımlama gerekli mi?
  - Her zaman İstemci ve Sunucu yazılımlarını aynı kişi(grup) yazmayabilir. Haberleşmenin nasıl olacağının bir yerden öğrenilmesi gerekir.
  - Kodlamayı yapan kişinin iletişimin durumlarına göre (state) programın nasıl davranacağını çözebileceği soyut bir anlatım gereklidir. Kodlama üzerinden takip edilemez.
- Protokoller bilgisayar ağlarında her katmanın tasarımı için kullanılırlar.
- Herkese açık (FTP , SMTP) veya Firmaya özgü (Skype) olabilirler.

Uygulama Katmanı Protokol Tanımlamaları

- types of messages exchanged,
  - e.g., request, response
- message syntax:
  - what fields in messages & how fields are delineated
- message semantics
  - meaning of information in fields
- rules for when and how processes send & respond to messages

open protocols:

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

proprietary protocols:

- e.g., Skype

Web Uygulamaları

- Herkese açık Protokoller:
- RFC : «Request for Comments» Döükümanları.
- Ör:
- RFC 1: İlk RFC dokimanı : http://www.rfc-editor.org/rfc/rfc1.txt (artık geçerliliğ olmayan bir RFC'dir.)
-

RFC Örnekleri

RFC 791[68]—The Internet Protocol (IP)

RFC 793[69]—The Transmission Control Protocol (TCP)

RFC 854[70]—The Telnet Protocol

RFC 959[71]—File Transfer Protocol (FTP)

RFC 1350[72]—The Trivial File Transfer Protocol (TFTP)

RFC 1459[73]—Internet Relay Chat Protocol (IRC)

RFC 1918[74]—Address Allocation for Private Internets

RFC 2131[75]—Dynamic Host Configuration Protocol (DHCP)

RFC 2616[76]—Hypertext Transfer Protocol (HTTP)

RFC 2821[77]—Simple Mail Transfer Protocol (SMTP)

RFC 3330[78]—Special-Use IPv4 Addresses

RFC 3493[79]—Basic Socket Interface Extensions for IPv6

RFC 3542[80]—Advanced Sockets Application Program Interface (API) for IPv6

RFC 3849[81]—IPv6 Address Prefix Reserved for Documentation

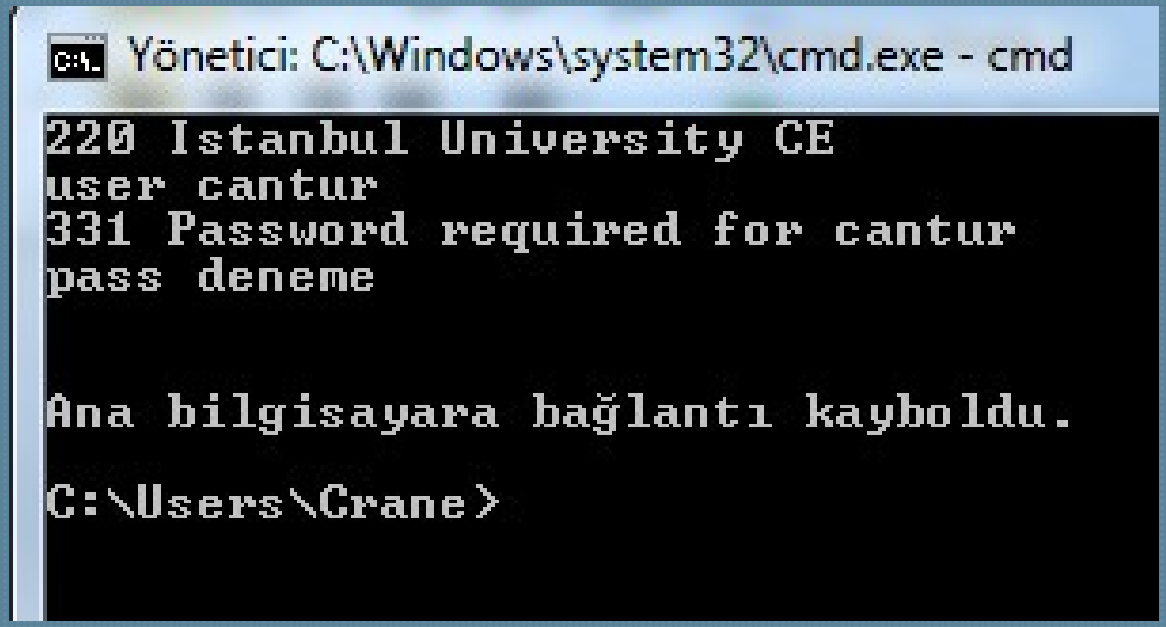
RFC 3920[82]—Extensible Messaging and Presence Protocol (XMPP)

FTP: RFC İncelemesi

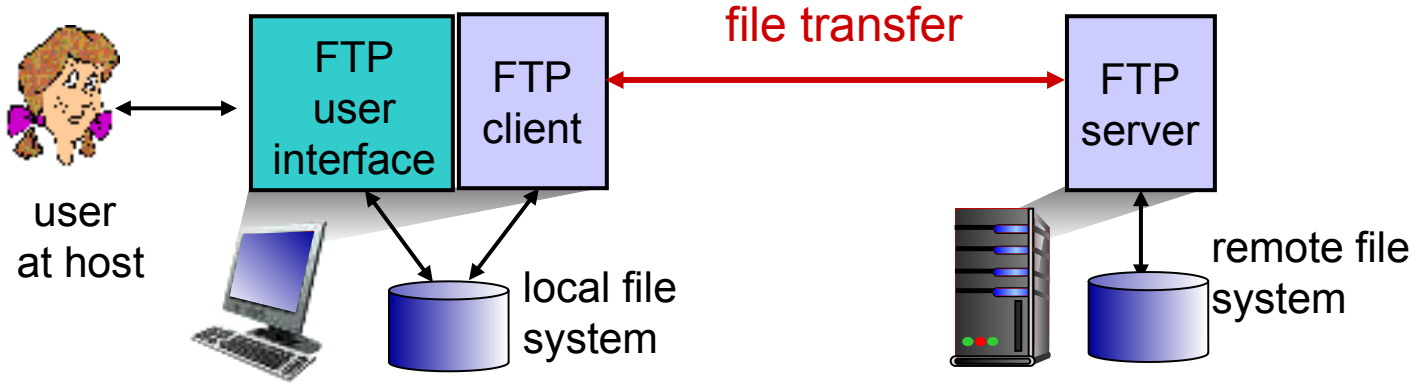
- http://www.rfc-editor.org/rfc/rfc959.txt
- Bir FTP Client yazmak istersek ilgili socket programlamada veri iletimini nasıl gerçekleştireceğimizi ilgili Protokol tanımlamasından öğreniriz.
- FTP protokol dokümanını inceleyiniz.

FTP RFC İncelemesi

- Telnet ile FTP bağlantısı
  - Windows Telnet Kurulumu
    - pkgmgr /iu:"TelnetClient«
  - telnet ce.istanbul.edu.tr 21



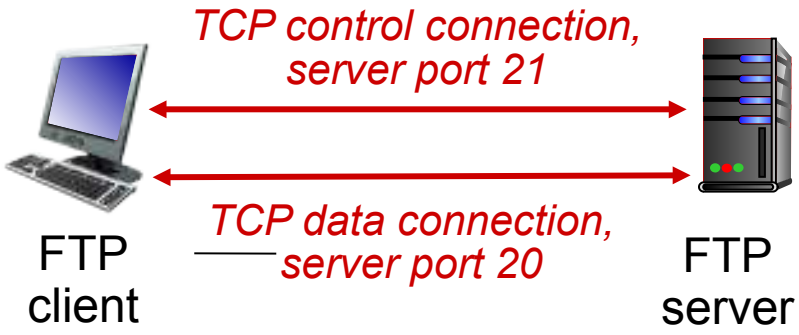
FTP: the file transfer protocol



- transfer file to/from remote host
- client/server model
  - *client:* side that initiates transfer (either to/from remote)
  - *server:* remote host
- ftp: RFC 959
- ftp server: port 21

FTP: separate control, data connections

- FTP client contacts FTP server at port 21, using TCP
- client authorized over control connection
- client browses remote directory, sends commands over control connection
- when server receives file transfer command, *server* opens *2nd* TCP data connection (for file) *to* client
- after transferring one file, server closes data connection



- server opens another TCP data connection to transfer another file
- control connection: *"out of band"*
- FTP server maintains "state": current directory, earlier authentication

FTP commands, responses

sample commands:

- sent as ASCII text over control channel
- **`USER` *`username`***
- **`PASS` *`password`***
- **`LIST`** return list of file in current directory
- **`RETR filename`** retrieves (gets) file
- **`STOR filename`** stores (puts) file onto remote host

sample return codes

- status code and phrase (as in HTTP)
- **`331 Username OK, password required`**
- **`125 data connection already open; transfer starting`**
- **`425 Can't open data connection`**
- **`452 Error writing file`**

Web and HTTP

First, a review…

- *web page* consists of *objects*
- object can be HTML file, JPEG image, Java applet, audio file,…
- web page consists of *base HTML-file* which includes *several referenced objects*
- each object is addressable by a *URL,* e.g.,

`www.someschool.edu/someDept/pic.gif`

host name | path name

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 (continued)

uses TCP:

- client initiates TCP connection (creates socket) to server, port 80
- 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

protocols that maintain "state" are complex!

- past history (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 then closed
- downloading multiple objects required multiple connections

persistent HTTP

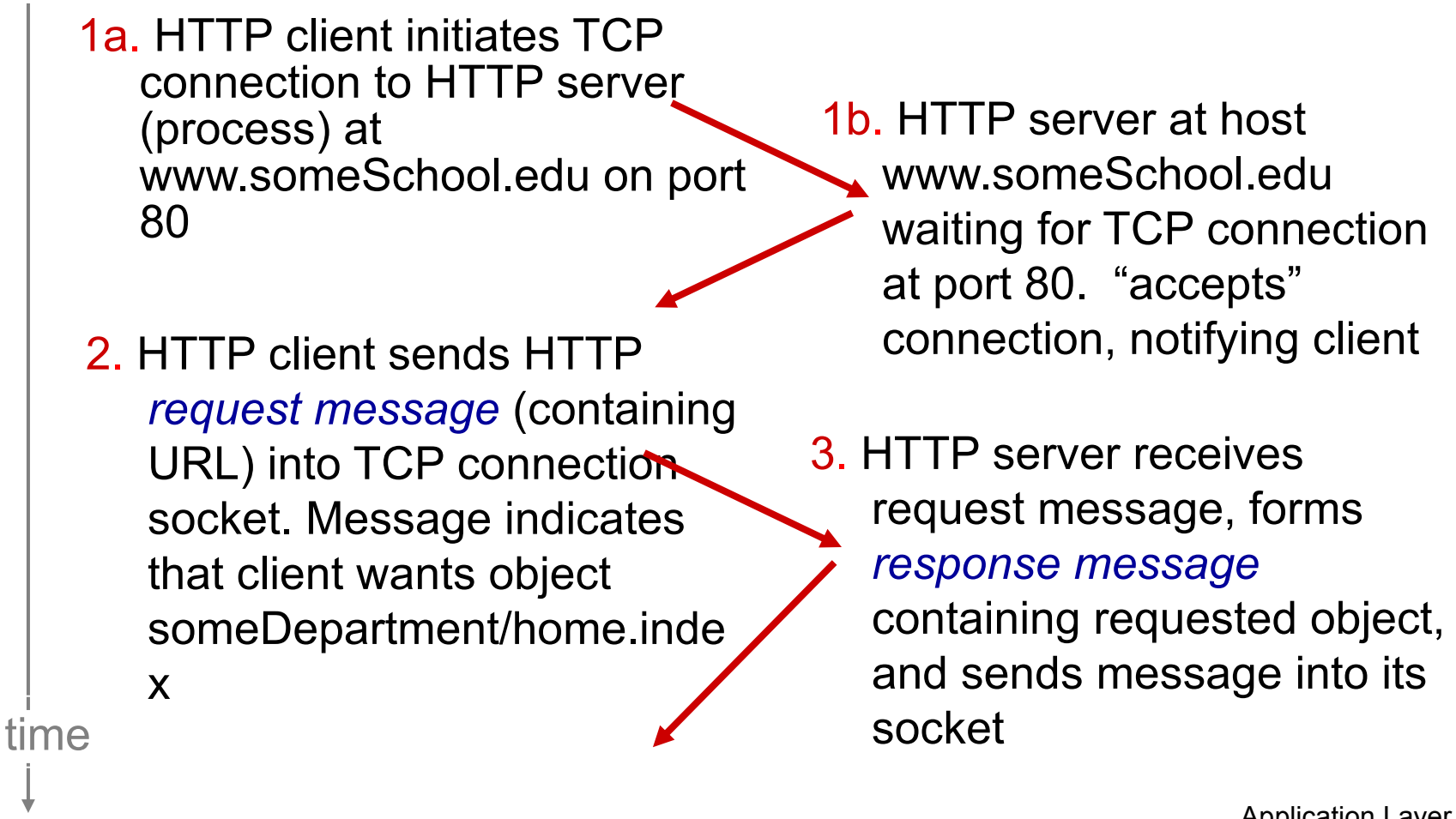
- multiple objects can be sent over single TCP connection between client, server

Non-persistent HTTP

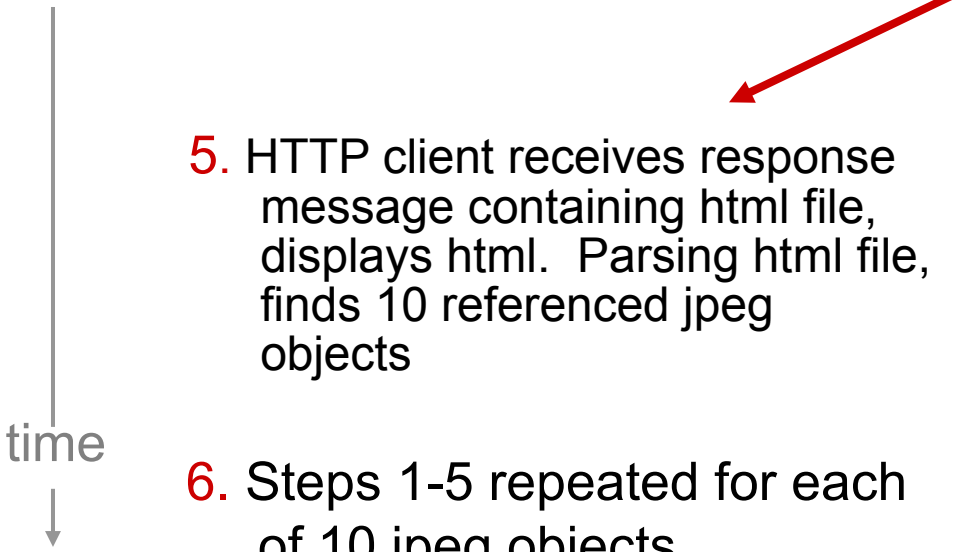
suppose user enters URL:

`www.someSchool.edu/someDepartment/home.index`

(contains text, references to 10 jpeg images)



Non-persistent HTTP (cont.)



4. HTTP server closes TCP connection.

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

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

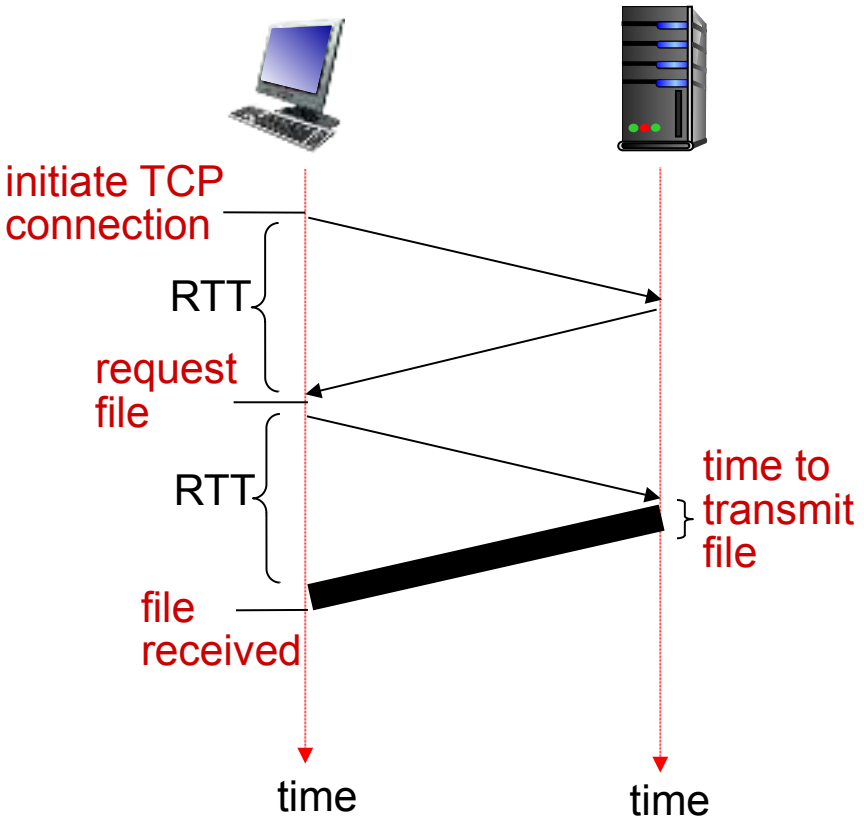
Non-persistent HTTP: response time

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

non-persistent HTTP issues:

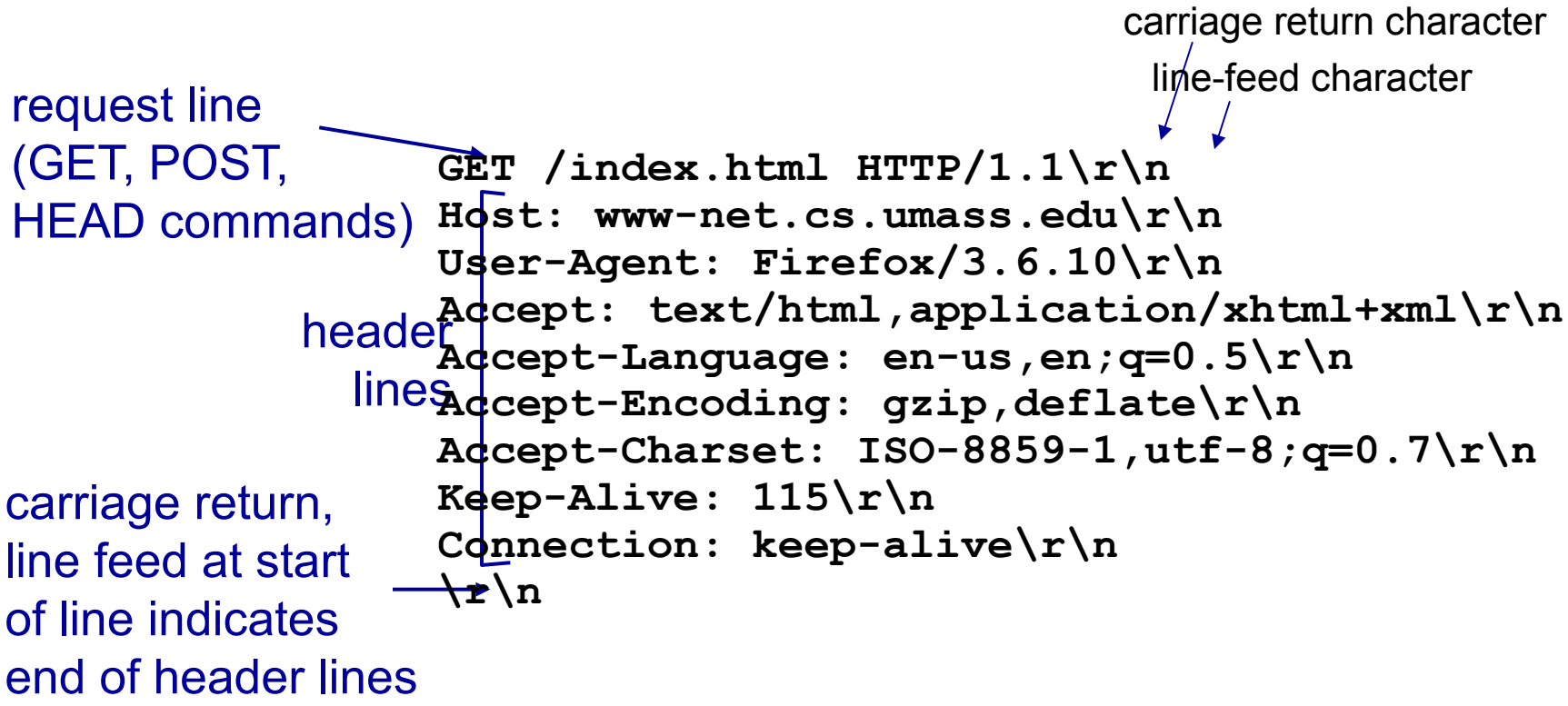
- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open parallel TCP connections to fetch referenced objects

persistent HTTP:

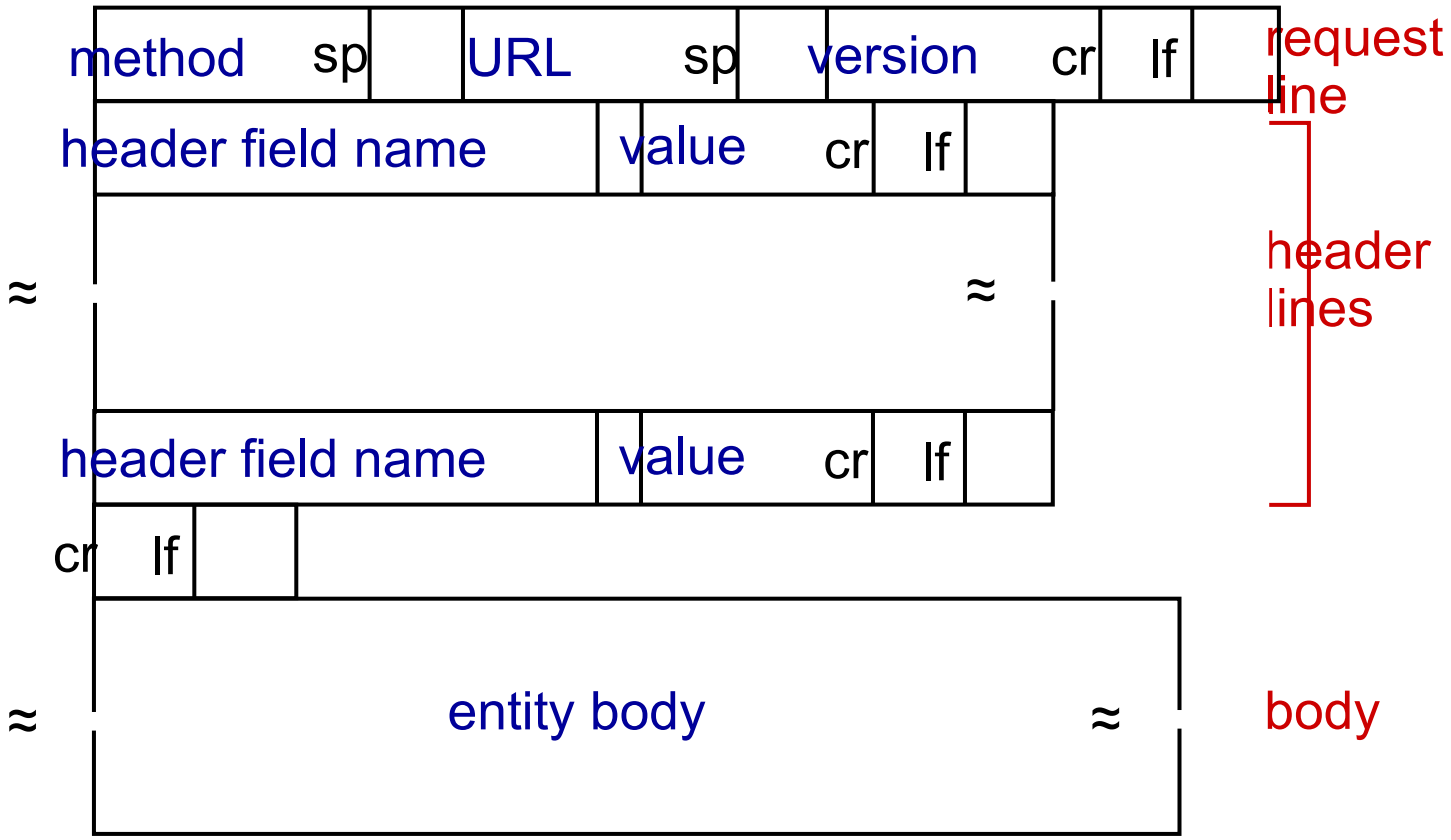
- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

HTTP request message

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



HTTP request message: general format



Uploading form input

POST method:

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

URL method:

- uses GET method
- input is uploaded in URL field of request line:

```
www.somesite.com/animalsearch?monkeys&banana
```

Method types

HTTP/1.0:

- GET
- POST
- HEAD
  - asks server to leave requested object out of response

HTTP/1.1:

- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field

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-Modified: Tue, 30 Oct 2007 17:00:02
   GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\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 succeeded, requested object later in this msg

301 Moved Permanently

- requested object moved, new location specified later 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

Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet cis.poly.edu 80
```

opens TCP connection to port 80 (default HTTP server port) at cis.poly.edu. anything typed in sent to port 80 at cis.poly.edu

2. type in a GET HTTP request:

```
GET /~ross/ HTTP/1.1
Host: cis.poly.edu
```

by typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. look at response message sent by HTTP server!

(or use Wireshark to look at captured HTTP request/response)