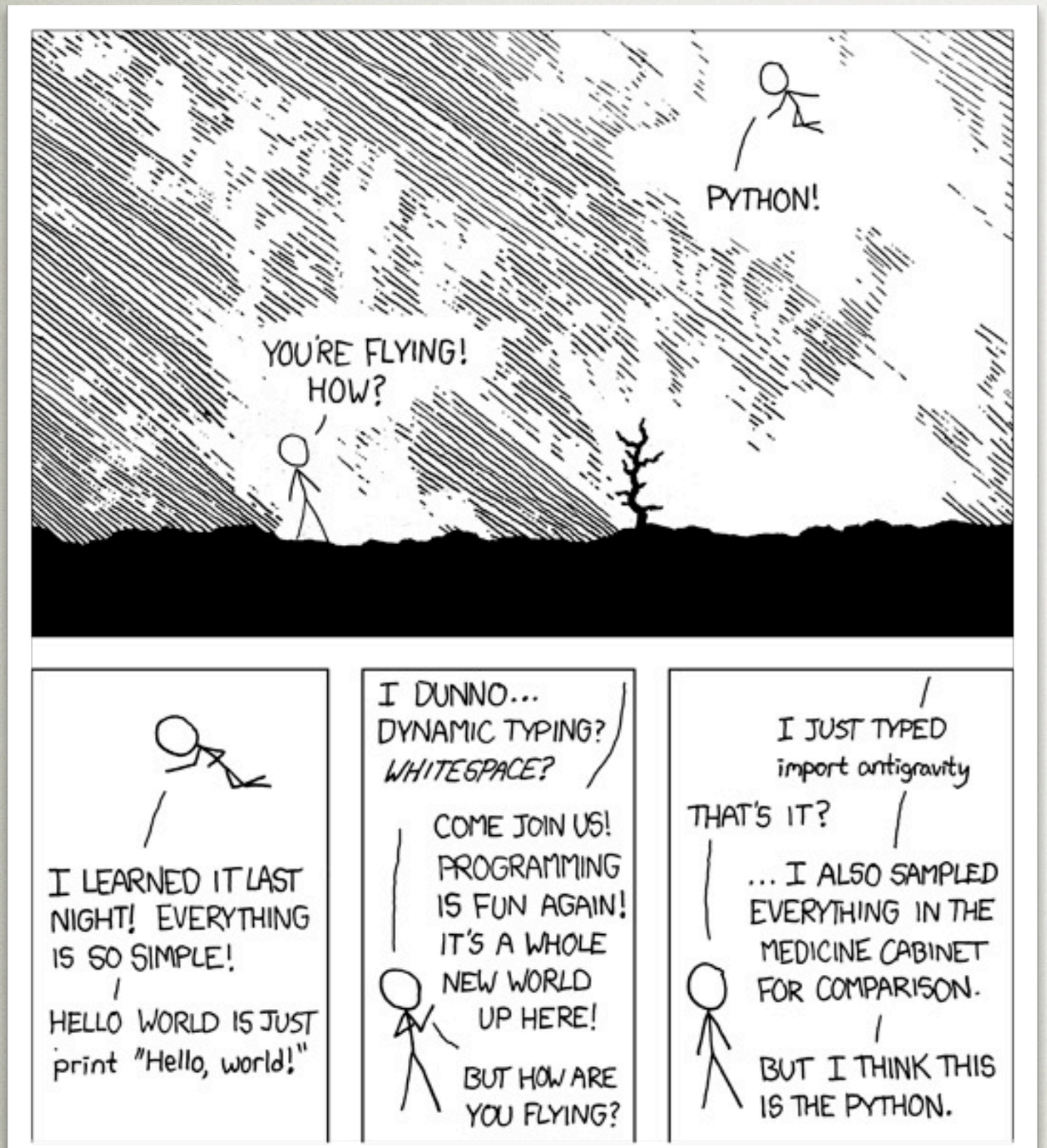


http://xkcd.com/353/

Build a dynamic HTTP server from scratch (in Python)

Brian Dorsey
brian@dorseys.org
http://briandorsey.info

thanks!

Overview & Introductions

introductions

name

python experience

web experience

why this class?

some thoughts...

(not directly related to HTTP...)

Magic

Magic

(or at least, indistinguishable from)

Embarrassment of riches

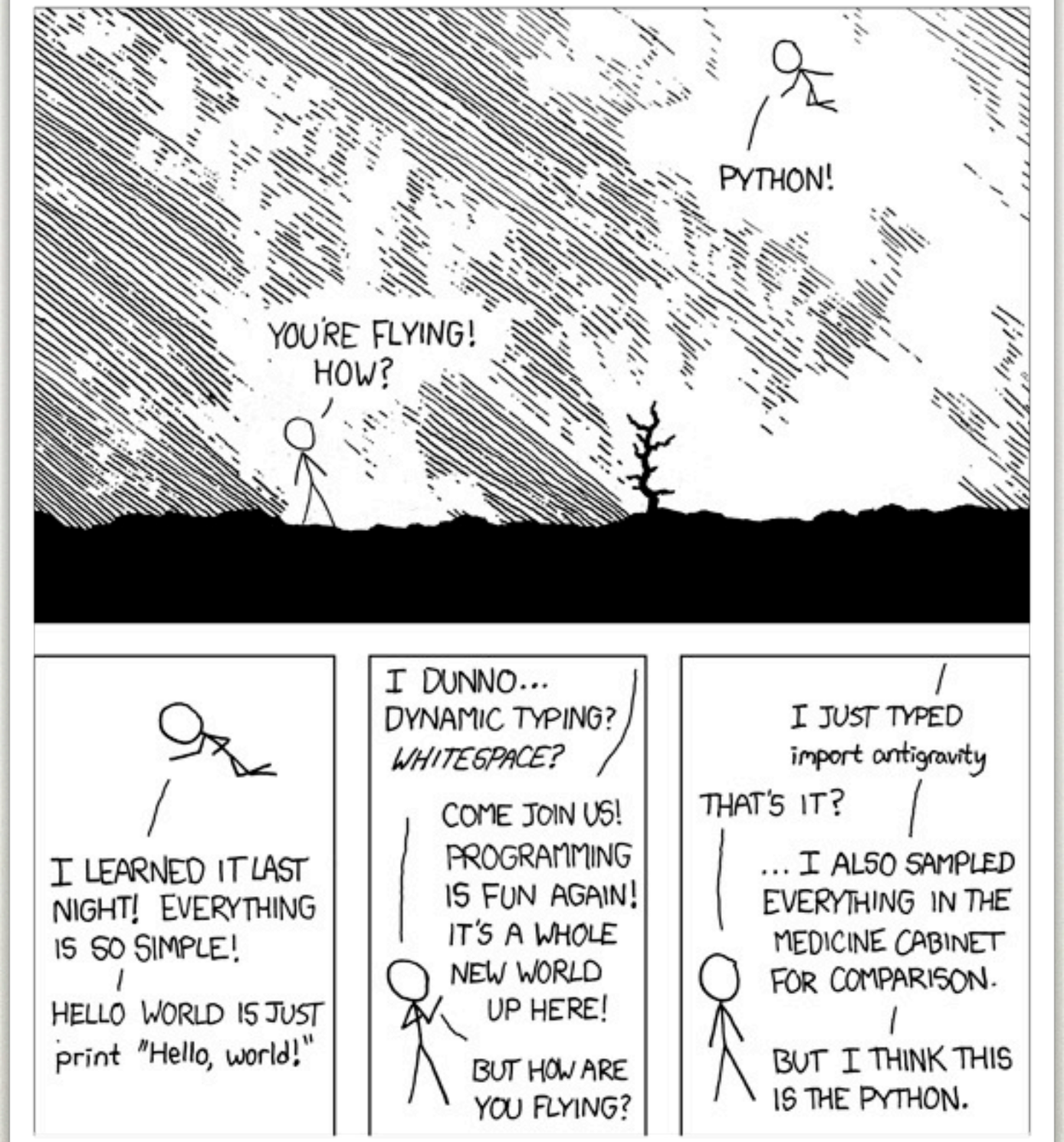
Beware the trappings of authority.

Collaboration

"One of the things I find that's remarkable about Python is that it has a very even learning curve. Maybe it's not even a curve, It's kind of a straight line."
- Bruce Eckel

http://www.artima.com/intv/prodperfP.html

(imagine graph here)



http://xkcd.com/353/

going to try to do the opposite to HTTP

skipping a lot of details covering just enough to build our HTTP server

don't use this server
for real projects

it's optimized for learning,
not performance or
reliability

Unit A
(25)

Networking:
from hardware to sockets

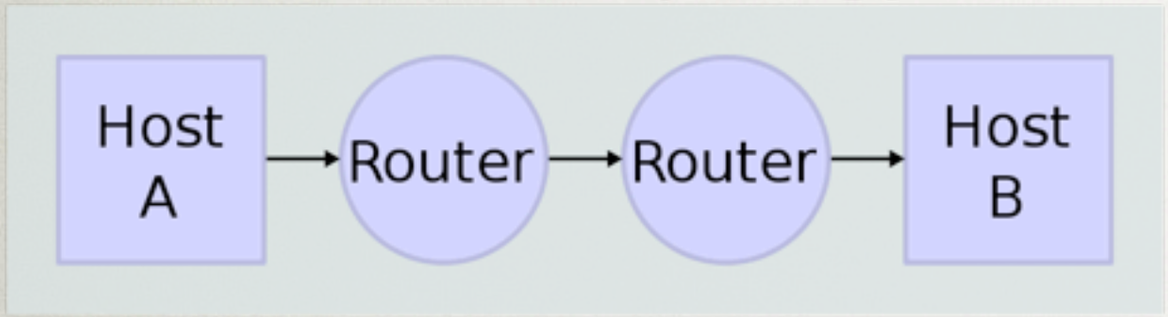
level setting

tcp/ip, IP address, ports, sockets, telnet, HTTP, HTTP Request/Response, verbs

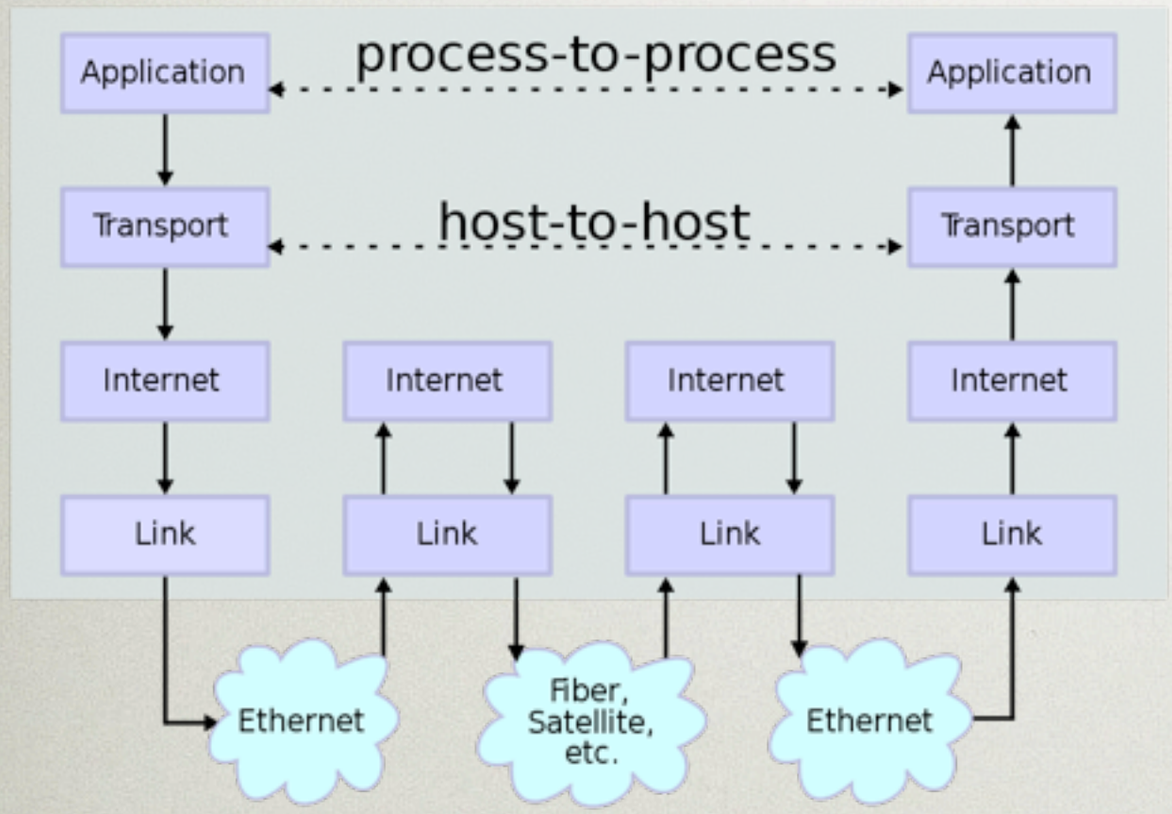
Questions from the readings?

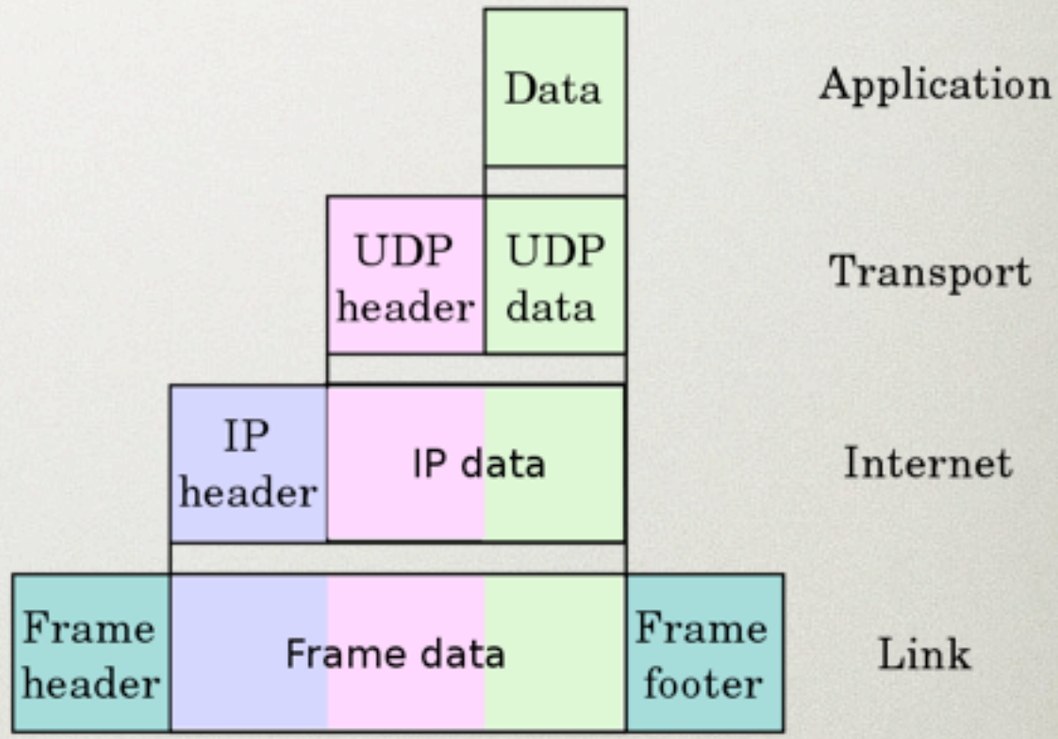
- Very quick overview of web fundamentals
- Internet Protocol Suite
- Internet Socket
- HTTP Made Really Easy

Network Topology



Data Flow





http://en.wikipedia.org/wiki/Internet_Protocol_Suite

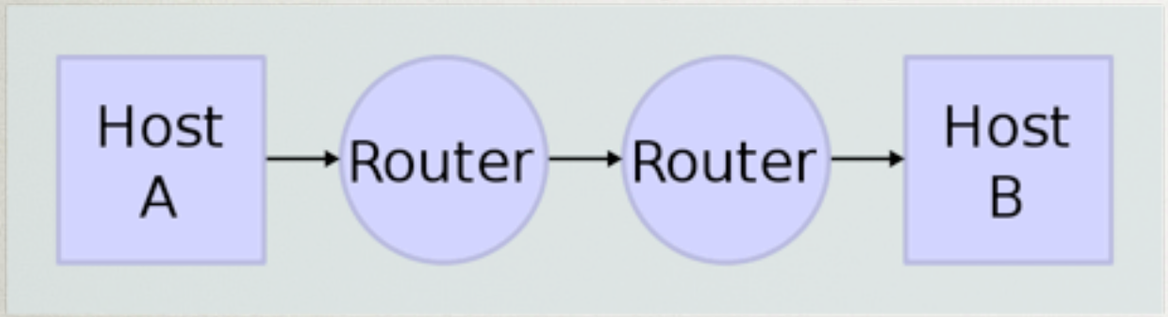
IP addresses

ports

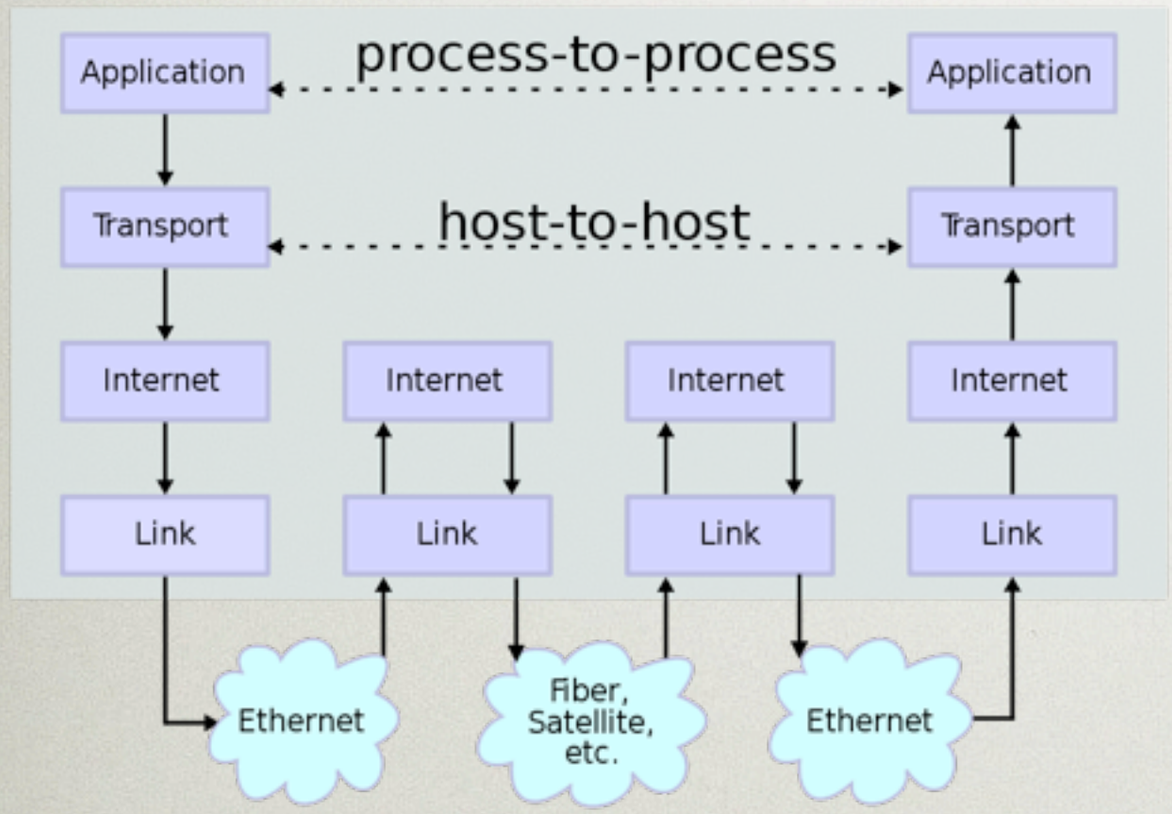
transports

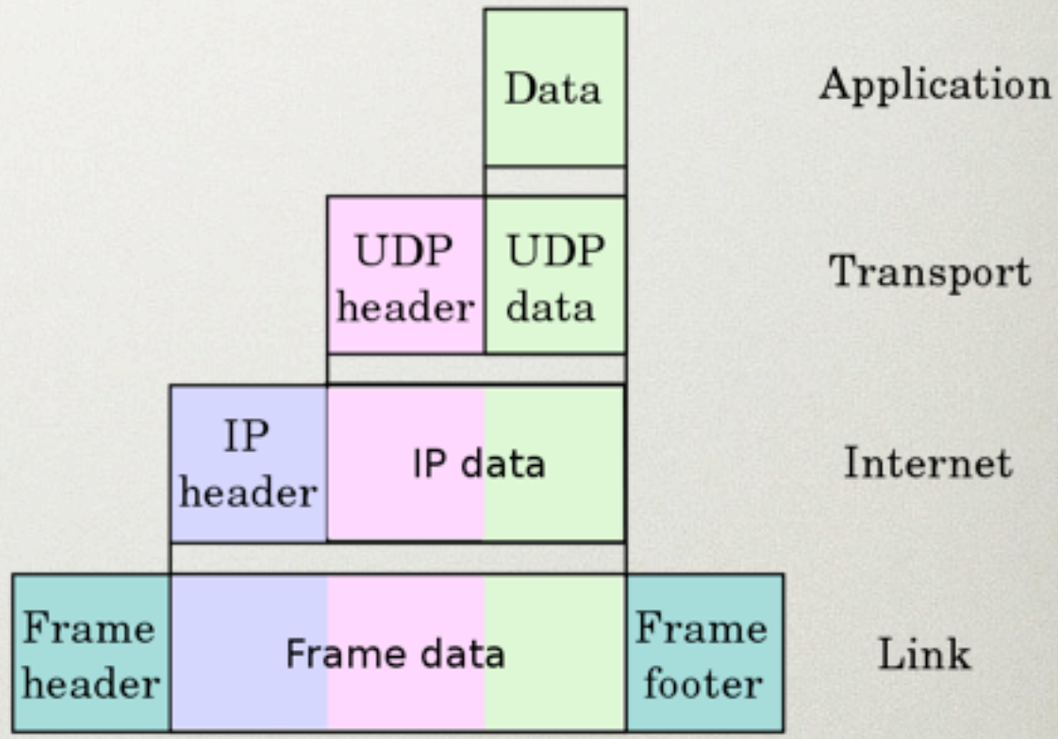
TCP / UDP

Network Topology



Data Flow





http://en.wikipedia.org/wiki/Internet_Protocol_Suite

Socket abstraction (IP address, port, transport)

TCP Transmission Control Protocol	UDP User Datagram Protocol
stream of data	packets of data
guaranteed delivery	no guarantees
connection setup (3 packets)	no connection
error checking	no error checking
ordered delivery	no order guarantees
lost packets are resent	lost packets are lost
HTTP, FTP, SMTP, SSH, telnet, most control data	DNS, DHCP, SNMP, VOIP, most game data & streaming media

text vs. binary

strings vs. bytes

unicode vs. encodings

server vs. client

(TCP echo server)

```
import socket

host = ''
port = 50000
backlog = 5
size = 1024
s = socket.socket(socket.AF_INET,
                  socket.SOCK_STREAM)
s.bind((host,port))
s.listen(backlog)
while True:
    client, address = s.accept()
    data = client.recv(size)
    if data:
        client.send(data)
    client.close()
```

http://ilab.cs.byu.edu/python/socket/echoserver.html

(TCP echo client)

```
import socket

host = 'localhost'
port = 50000
size = 1024
s = socket.socket(socket.AF_INET,
                  socket.SOCK_STREAM)
s.connect((host,port))
s.send('Hello, world')
data = s.recv(size)
s.close()
print 'Received:', data
```

http://ilab.cs.byu.edu/python/socket/echoclient.html

```
import socket

host = ''
port = 50000
backlog = 5
size = 1024
s = socket.socket(socket.AF_INET,
                  socket.SOCK_STREAM)
s.bind((host,port))
s.listen(backlog)
while True:
```

```
client, address = s.accept()

data = client.recv(size)
if data:
    client.send(data)

client.close()
```

```
import socket

host = 'localhost'
port = 50000
size = 1024
s = socket.socket(socket.AF_INET,
                  socket.SOCK_STREAM)
```

```
s.connect((host,port))

s.send('Hello, world')
```

```
data = s.recv(size)

s.close()

print 'Received:', data
```

Lab A

(25)

Lab A

https://github.com/briandorsey/http_tutorial

git clone git://github.com/briandorsey/http_tutorial.git

svn checkout http://github.com/briandorsey/http_tutorial.git/trunk http_tutorial

- open a console and run `echo_server.py`
- open a second console and run `echo_client.py` a few times
- use telnet client to connect to the echo server:
  telnet localhost 50000
  python -m telnetlib localhost 50000
- open two python interpreters and manually input both programs, starting with the server side until it blocks, then just enough of the client to unblock the server.
  (skip the "`while True:`" line on the server side)
- modify `echo_server.py` to ignore what the client sends, and always reply with the time
- Bonus: raw socket version of whatismyip.com - modify the server to tell the client what its' IP address is.

how'd that go?

Unit B
(25)

HTTP

reading: “HTTP Made Really Easy”

did it work?

```
GET /path/to/index.html HTTP/1.0
```

HTTP Requests

HTTP Responses

stateless

required first line
optional headers
“blank” line (CRLF)
body (data)

```
GET /path/file.html HTTP/1.0
User-Agent: HTTPTool/1.0
[blank line here]
```

request

---

response

```
HTTP/1.0 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Content-Type: text/html
Content-Length: 1354

<html>
<body>
<h1>Happy New Millennium!</h1>
(more file contents)
  . . .
</body>
</html>
```

```
GET /path/to/index.html HTTP/1.0
```

HTTP Requests

```
GET /path/to/index.html HTTP/1.0
```

resources (nouns)

```
GET /path/to/index.html HTTP/1.0
```

resources != files

```
GET /path/to/index.html HTTP/1.0
```

methods (verbs)

```
GET /path/to/index.html HTTP/1.0
```

GET
POST
PUT
DELETE

`GET /path/to/index.html HTTP/1.0`

POST = Create

GET = Read

PUT = Update

DELETE = Delete

GET /path/to/index.html HTTP/1.0

safe

side
effects

GET

POST
PUT
DELETE

```
GET /path/to/index.html HTTP/1.0
```

idempotent

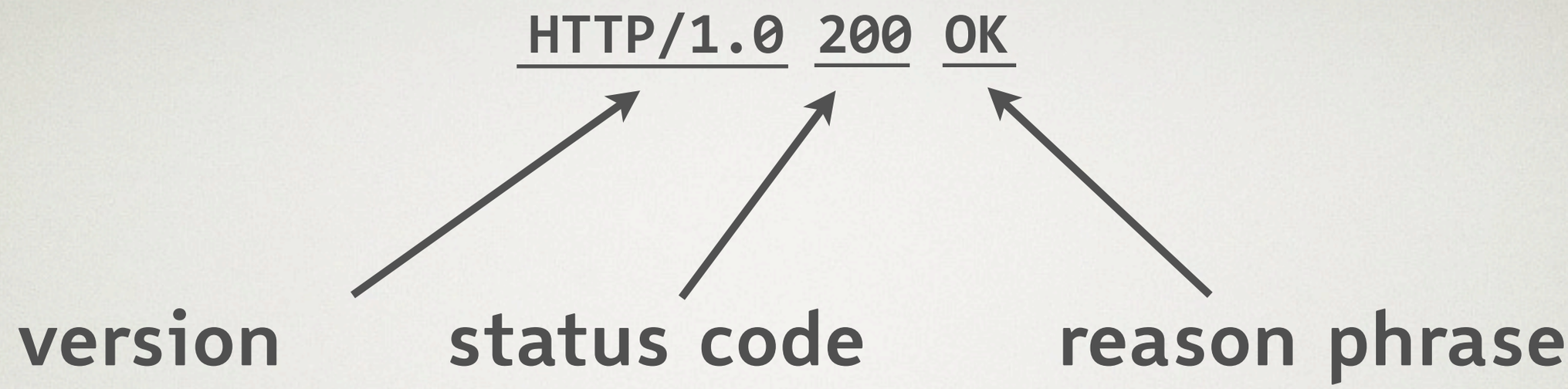
GET
PUT
DELETE

changes after each request

POST

```
HTTP/1.0 200 OK
```

HTTP Responses


HTTP/1.0 200 OK
version
status code
reason phrase

```
HTTP/1.0 200 OK
```

- 1xx indicates an informational message
- 2xx indicates success of some kind
- 3xx redirects the client to another URL
- 4xx indicates an error on the client's part
- 5xx indicates an error on the server's part

`HTTP/1.0 200 OK`

- 200 OK
- 404 Not Found
- 301 Moved Permanently
- 302 Moved Temporarily
- 303 See Other (HTTP 1.1 only)
- 500 Server Error

http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

```
GET /path/file.html HTTP/1.0
User-Agent: HTTPTool/1.0
[blank line here]
```

request

---

response

```
HTTP/1.0 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Content-Type: text/html
Content-Length: 1354

<html>
<body>
<h1>Happy New Millennium!</h1>
(more file contents)
  . . .
</body>
</html>
```

Header-Name: value

Quick reference to HTTP headers

http://www.cs.tut.fi/~jkorpela/http.html

body data

Content-Type: xyz

(mime types)

DEBUGGING TOOLS

- windows: http://www.fiddler2.com/fiddler2/
- windows, mac & linux: http://www.charlesproxy.com/
- firefox: http://getfirebug.com/
- Safari, Chrome, Firefox and IE: all have some tools built-in

questions?

HTTP telnet demo

HTTP curl demo

browser network tools demo

Lab B

(20)

Lab B

- Experiment with making manual HTTP requests using telnet. Connect with:
  telnet www.website.com 80
  type:
  `GET / HTTP/1.0`
  (then enter twice)
- Open your browser's developer tools, enable the network panel and open a few websites you regularly visit. Look at HTTP headers for some of the resources.
- Turn the echo client into an HTTP client.
- Bonus: accept a URL argument.
  (`sys.argv` may be helpful)

Unit C
(25)

30 Minute Webserver

The Thirty Minute Web Server

(presented in about thirty minutes)

demo

story

review

line continuation
triple quotes
string interpolation

```
header = \
"""HTTP/1.0 200 Okay
Server: ws30
Content-type: %s

%s
"""
```

mime types

codewalk

Lab C
(25)

safe place to fail

safe place to break stuff!

Lab C

https://github.com/briandorsey/http_tutorial
git clone git://github.com/briandorsey/http_tutorial.git
svn checkout http://github.com/briandorsey/http_tutorial.git/trunk http_tutorial

- Run `thirty_minute_webserver.py`
- Point your web browser at it, experiment. Break code, add prints, etc.
- Add `/time` url which returns HTML with the actual current time.
- Bonus: Instead of showing the content of `.py` files, run the script and return its' output instead. (`subprocess.check_output()` will be useful) Test with `print_time.py` script.

Wrap Up

```
python -m SimpleHTTPServer 8000
```

httplib

```
import httplib

conn = httplib.HTTPConnection("www.python.org")
conn.request("GET", "/index.html")
r1 = conn.getresponse()
print r1.status, r1.reason
data1 = r1.read()

conn.close()
```

http://docs.python.org/release/2.6.5/library/httplib.html

urllib2

```
import urllib2
f = urllib2.urlopen('http://www.python.org/')
print f.read()
```

http://docs.python.org/release/2.6.5/library/urllib2.html#examples

requests

```
import requests
r = requests.get('http://www.python.org/')
print r.text
```

http://docs.python-requests.org/en/latest/index.html

Did you have an HTTP epiphany?

Is HTTP mysterious?

Feedback

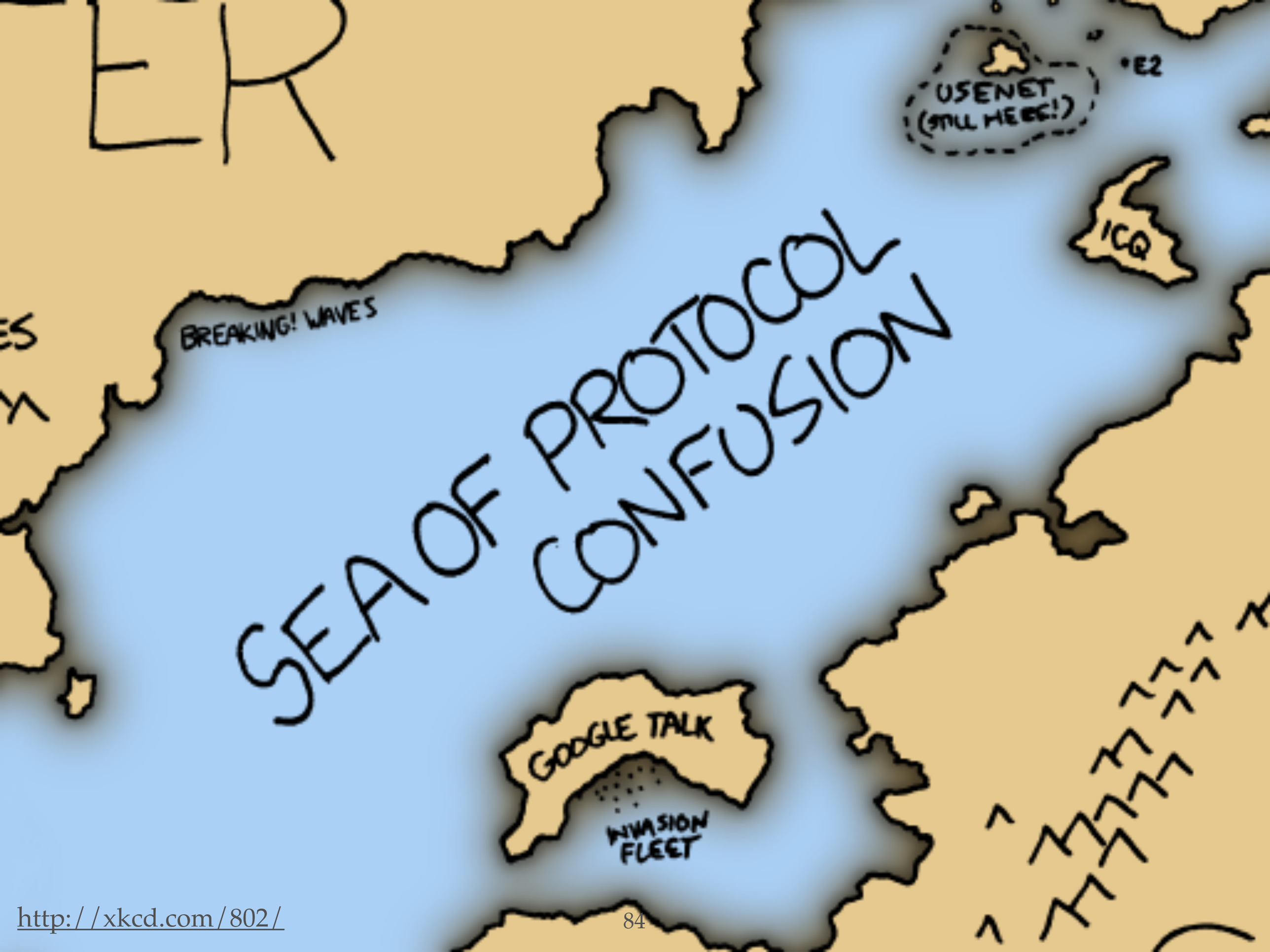
- Write me a note with feedback on this tutorial: brian@dorseys.org
- Did you have an HTTP epiphany? Is HTTP mysterious?
- What worked well, what worked poorly.
- Additional info which should be included?
- Comments

Followup Excercises

- Update thirty_minute_webserver.py
- Instead of showing the content of .py files, run the script and return its' output instead. (the stdlib subprocess.Popen() will be useful)
- test with script which outputs the time (print_time.py)
- update the print_time.py script to output html
- You've now written a dynamic webserver! (and re-invented part of CGI!)

Bonus (25)

MORE PROTOCOLS


ER
USENET
(STILL HERE!)
*E2
ICQ
ES
M
BREAKING! WAVES
SEA OF PROTOCOL
CONFUSION
GOOGLE TALK
INVASION
FLEET

SMTP

```
S: 220 foo.com Simple Mail Transfer Service Ready
C: EHLO bar.com
S: 250-foo.com greets bar.com
S: 250-8BITMIME
S: 250-SIZE
S: 250-DSN
S: 250 HELP
C: MAIL FROM:<Smith@bar.com>
S: 250 OK
C: RCPT TO:<Jones@foo.com>
S: 250 OK
C: RCPT TO:<Green@foo.com>
S: 550 No such user here
C: RCPT TO:<Brown@foo.com>
S: 250 OK
C: DATA
S: 354 Start mail input; end with <CRLF>.<CRLF>
C: Blah blah blah...
C: ...etc. etc. etc.
C: .
S: 250 OK
C: QUIT
S: 221 foo.com Service closing transmission channel
```

http://tools.ietf.org/html/rfc5321#appendix-D

POP3

```
C: <client connects to service port 110>
S: +OK POP3 server ready <1896.6971@mailgate.dobbs.org>
C: USER bob
S: +OK bob
C: PASS redqueen
S: +OK bob's maildrop has 2 messages (320 octets)
C: STAT
S: +OK 2 320
C: LIST
S: +OK 2 messages (320 octets)
S: 1 120
S: 2 200
S: .
C: RETR 1
S: +OK 120 octets
S: <the POP3 server sends the text of message 1>
S: .
C: DELE 1
S: +OK message 1 deleted
C: RETR 2
S: +OK 200 octets
S: <the POP3 server sends the text of message 2>
S: .
C: DELE 2
S: +OK message 2 deleted
C: QUIT
S: +OK dewey POP3 server signing off (maildrop empty)
C: <client hangs up>
```

http://www.faqs.org/docs/artu/ch05s03.html

IMAP

```
C: <client connects to service port 143>
S: * OK example.com IMAP4rev1 v12.264 server ready
C: A0001 USER "frobozz" "xyzzy"
S: * OK User frobozz authenticated
C: A0002 SELECT INBOX
S: * 1 EXISTS
S: * 1 RECENT
S: * FLAGS (\Answered \Flagged \Deleted \Draft \Seen)
S: * OK [UNSEEN 1] first unseen message in /var/spool/mail/esr
S: A0002 OK [READ-WRITE] SELECT completed
C: A0003 FETCH 1 RFC822.SIZE                    Get message sizes
S: * 1 FETCH (RFC822.SIZE 2545)
S: A0003 OK FETCH completed
C: A0004 FETCH 1 BODY[HEADER]                   Get first message header
S: * 1 FETCH (RFC822.HEADER {1425}
<server sends 1425 octets of message payload>
S: )
S: A0004 OK FETCH completed
C: A0005 FETCH 1 BODY[TEXT]                     Get first message body
S: * 1 FETCH (BODY[TEXT] {1120}
<server sends 1120 octets of message payload>
S: )
S: * 1 FETCH (FLAGS (\Recent \Seen))
S: A0005 OK FETCH completed
C: A0006 LOGOUT
S: * BYE example.com IMAP4rev1 server terminating connection
S: A0006 OK LOGOUT completed
C: <client hangs up>
```

http://www.faqs.org/docs/artu/ch05s03.html

IMAP (reordering)

```
C: <client connects to service port 143>
S: * OK example.com IMAP4rev1 v12.264 server ready
C: A0001 USER "frobozz" "xyzzy"
S: * OK User frobozz authenticated
C: A0002 SELECT INBOX
S: * 1 EXISTS
S: * 1 RECENT
S: * FLAGS (\Answered \Flagged \Deleted \Draft \Seen)
S: * OK [UNSEEN 1] first unseen message in /var/spool/mail/esr
S: A0002 OK [READ-WRITE] SELECT completed
C: A0003 FETCH 1 RFC822.SIZE                    Get message sizes
C: A0004 FETCH 1 BODY[HEADER]                   Get first message header
C: A0005 FETCH 1 BODY[TEXT]                     Get first message body
S: * 1 FETCH (RFC822.SIZE 2545)
S: A0003 OK FETCH completed
S: * 1 FETCH (RFC822.HEADER {1425}
<server sends 1425 octets of message payload>
S: )
S: A0004 OK FETCH completed
S: * 1 FETCH (BODY[TEXT] {1120}
<server sends 1120 octets of message payload>
S: )
S: * 1 FETCH (FLAGS (\Recent \Seen))
S: A0005 OK FETCH completed
C: A0006 LOGOUT
S: * BYE example.com IMAP4rev1 server terminating connection
S: A0006 OK LOGOUT completed
C: <client hangs up>
```

http://www.faqs.org/docs/artu/ch05s03.html

GNU Go Text Protocol (GTP)

```
1 boardsize 7
=1

2 clear_board
=2

3 play black D5
=3

4 genmove white
=4 C3

5 play black C3
?5 illegal move

6 play black E3
=6

7 showboard
=7
   A B C D E F G
 7 . . . . . . . 7
 6 . . . . . . . 6
 5 . . + X + . . 5
 4 . . . + . . . 4
 3 . . O . X . . 3
 2 . . . . . . . 2     WHITE (O) has captured 0 stones
 1 . . . . . . . 1     BLACK (X) has captured 0 stones
   A B C D E F G

8 quit
=8
```

http://www.gnu.org/software/gnugo/gnugo_19.html

HTTP

```
GET /index.html HTTP/1.1
Host: www.example.com
```

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Etag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Content-Length: 438
Connection: close
Content-Type: text/html; charset=UTF-8

<data>
```

http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

many others only run or optionally run over HTTP

now, in Python

smtplib

```
import smtplib

template = """From: %s
To: %s
Subject: %s

"""

from_addr = 'Darth Vader <darth@deathstar.com>'
to_addrs = 'briandorsey@gmail.com'
subject = "I'm your father."
message = 'message body'
headers = template % (from_addr, to_addrs, subject)

s = smtplib.SMTP('mail.blueboxgrid.com')
s.ehlo()
s.sendmail(from_addr, to_addrs, headers + message)
s.close()
```

poplib

```
import poplib
import string, random

SERVER = "pop.spam.egg"
USER  = "mulder"
PASSWORD = "trustno1"

# connect to server
server = poplib.POP3(SERVER)

# login
server.user(USER)
server.pass_(PASSWORD)

# list items on server
resp, items, octets = server.list()

# download a random message
id, size = string.split(random.choice(items))
resp, text, octets = server.retr(id)
```

http://effbot.org/librarybook/poplib.htm

imaplib

```
import imaplib
import string, random

SERVER = "imap.spam.egg"
USER  = "mulder"
PASSWORD = "trustno1"

# connect to server
server = imaplib.IMAP4(SERVER)

# login
server.login(USER, PASSWORD)
server.select()

# list items on server
resp, items = server.search(None, "ALL")
items = string.split(items[0])

# fetch a random item
id = random.choice(items)
resp, data = server.fetch(id, "(RFC822)")
text = data[0][1]
```

http://effbot.org/librarybook/imaplib.htm

httplib

```
import httplib

conn = httplib.HTTPConnection("www.python.org")
conn.request("GET", "/index.html")
r1 = conn.getresponse()
print r1.status, r1.reason
data1 = r1.read()

conn.close()
```

http://docs.python.org/release/2.6.5/library/httplib.html

urllib2

```
import urllib2
f = urllib2.urlopen('http://www.python.org/')
print f.read()
```

http://docs.python.org/release/2.6.5/library/urllib2.html#examples

what's included in the standard library?

non-stdlib libraries?

a library exists for pretty much every protocol

for SSH / SFTP

Paramikio

http://www.lag.net/paramiko/

```
url = 'http://www.python.org/'
webbrowser.open_new_tab(url)
```