Intro to Networking

TCP, HTTP, and APIs - oh my!

Andrew Kerr | October 20, 2015

me@andrewjkerr.com

whoami

whoami

- Fifth year Software Engineering @ UF

whoami

- Fifth year Software Engineering @ UF
- Secretary of UFSIT for > 2yrs

whoami

- Fifth year Software Engineering @ UF
- Secretary of UFSIT for > 2yrs
- Full stack web developer

whoami

- Fifth year Software Engineering @ UF
- Secretary of UFSIT for > 2yrs
- Full stack web developer
- Former security intern at Tumblr

whoami

- Fifth year Software Engineering @ UF
- Secretary of UFSIT for > 2yrs
- Full stack web developer
- Former security intern at Tumblr
- Former intern at BlockScore

whoami

Andrew Kerr

@andrew

Professional nerd.

- Fifth year Software Engineering @ UF
- Secretary of UFSIT for > 2yrs
- Full stack web developer
- Former security intern at Tumblr
- Former intern at BlockScore
- Reach me @andrew on UF CISE Slack

Topics

Topics

- Networking layers

Topics

- Networking layers
- Wireshark

Topics

- Networking layers
- Wireshark
- TCP

Topics

- Networking layers
- Wireshark
- TCP
- HTTP

Topics

- Networking layers
- Wireshark
- TCP
- HTTP
- APIs

Tools

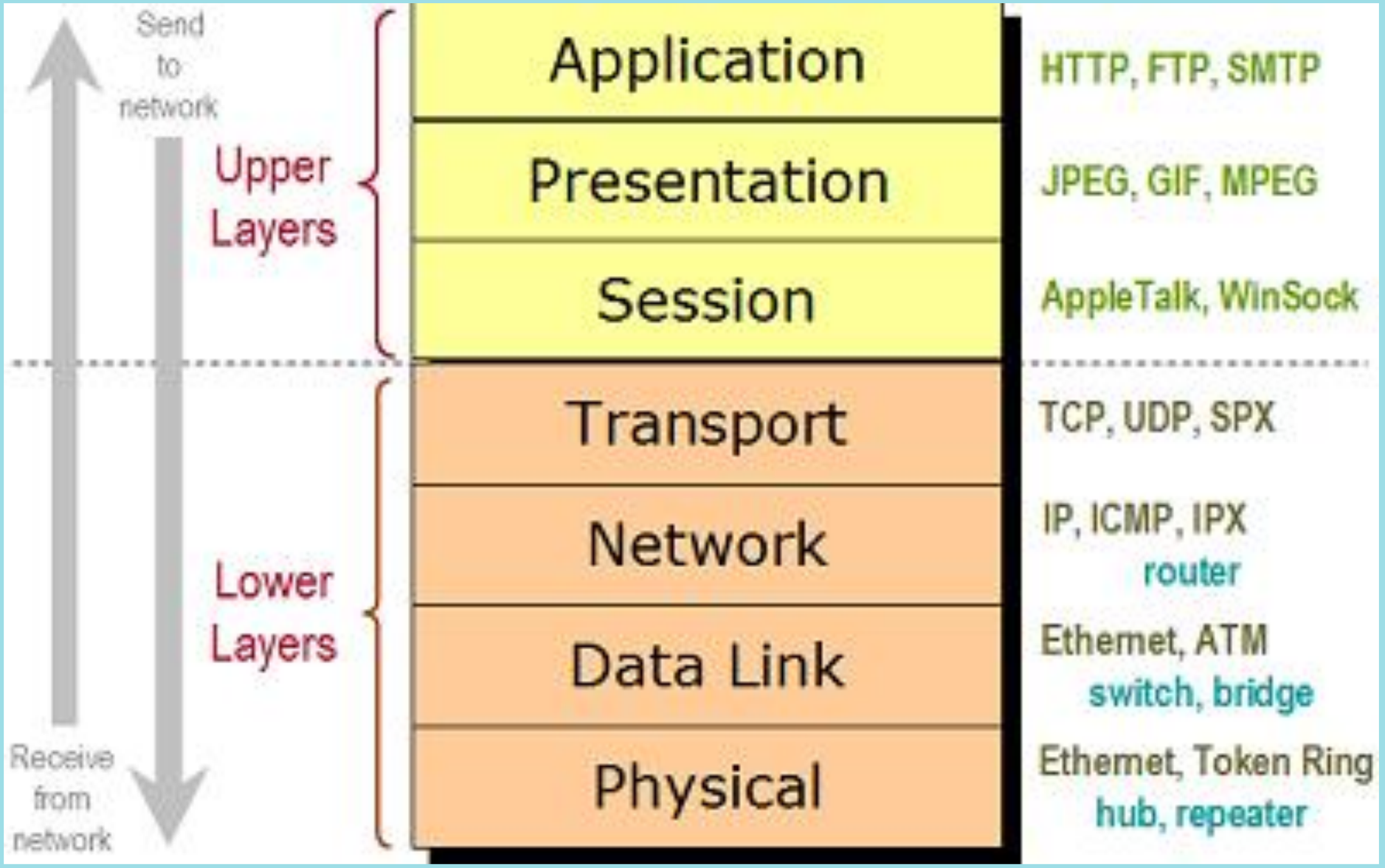
Tools

- Wireshark
- Google Chrome
- cURL
- Postman

Networking Layers

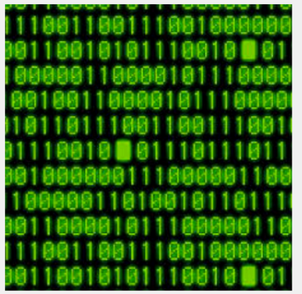
OSI Model

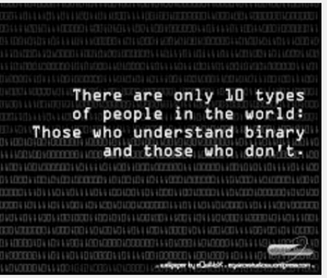
The OSI model is a conceptual **model!**

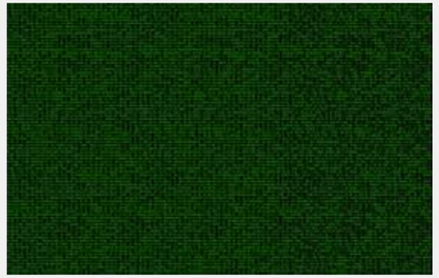

Send to network
Receive from network
Upper Layers
Lower Layers
Application
Presentation
Session
Transport
Network
Data Link
Physical
HTTP, FTP, SMTP
JPEG, GIF, MPEG
AppleTalk, WinSock
TCP, UDP, SPX
IP, ICMP, IPX
router
Ethernet, ATM
switch, bridge
Ethernet, Token Ring
hub, repeater

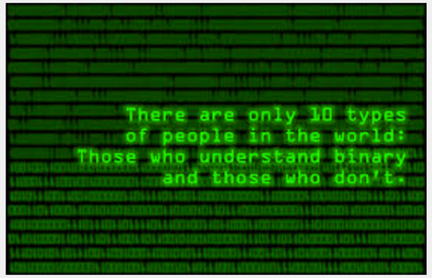
The layers work together to network your computer to the world

How do we send data back and forth?

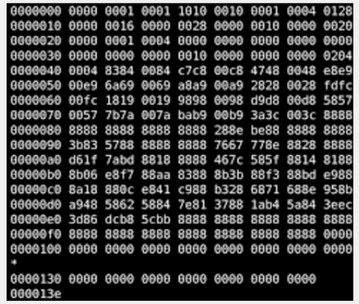



There are only 10 types
of people in the world:
Those who understand binary
and those who don't.




There are only 10 types
of people in the world:
Those who understand binary
and those who don't.


No need to be a binary code expert to see danger…
BE CAREFULL! THINK! DANGER IS EVERYWHERE!…


0000000 0000 0001 0001 1010 0010 0001 0004 0128
0000010 0000 0016 0000 0028 0000 0010 0000 0020
0000020 0000 0001 0004 0000 0000 0000 0000 0000
0000030 0000 0000 0000 0010 0000 0000 0000 0204
0000040 0004 8384 0084 c7c8 00c8 4748 0048 e8e9
0000050 00e9 6a69 0069 a8a9 00a9 2828 0028 fdfc
0000060 00fc 1819 0019 9898 0098 d9d8 00d8 5857
0000070 0057 7b7a 007a bab9 00b9 3a3c 003c 8888
0000080 8888 8888 8888 8888 288e be88 8888 8888
0000090 3b83 5788 8888 8888 7667 778e 8828 8888
00000a0 d61f 7abd 8818 8888 467c 585f 8814 8188
00000b0 8b06 e8f7 88aa 8388 8b3b 88f3 88bd e988
00000c0 8a18 880c e841 c988 b328 6871 688e 958b
00000d0 a948 5862 5884 7e81 3788 1ab4 5a84 3eec
00000e0 3d86 dcb8 5cbb 8888 8888 8888 8888 8888
00000f0 8888 8888 8888 8888 8888 8888 8888 0000
0000100 0000 0000 0000 0000 0000 0000 0000 0000
*
0000130 0000 0000 0000 0000 0000 0000 0000
000013e


01010100 01101000 01101001 01110011
00100000 01101001 01110011 00100000
01110100 01101000 01100101 00100000
01110100 01110101 01110100 01101111
01110010 01101001 01100001 01101100
00100000 01110100 01101111 00100000
01101100 01100101 01100001 01110010
01101110 00100000 01100010 01101001
01101110 01100001 01110010 01111001
00101110 00100000 01001001 00100000
01101000 01101111 01110000 01100101
00100000 01111001 01101111 01110101
00100000 01100101 01101110 01101010
01101111 01111001 00100000 01101001
01110100 00100001

The Past:

- Dial-up modem would translate 1s and 0s into tones

The Past:

- Dial-up modem would translate 1s and 0s into tones
- Receiving modem would decode 1s and 0s and change it back into data

Why doesn't this work now?

Too much data!

Too much data!

(Aka, you kids and your Netflix and chill)

What do we do now?

Fiber!!

Ok, but how do we send data?

Packets and segments!

Packets

- Formatted unit of data

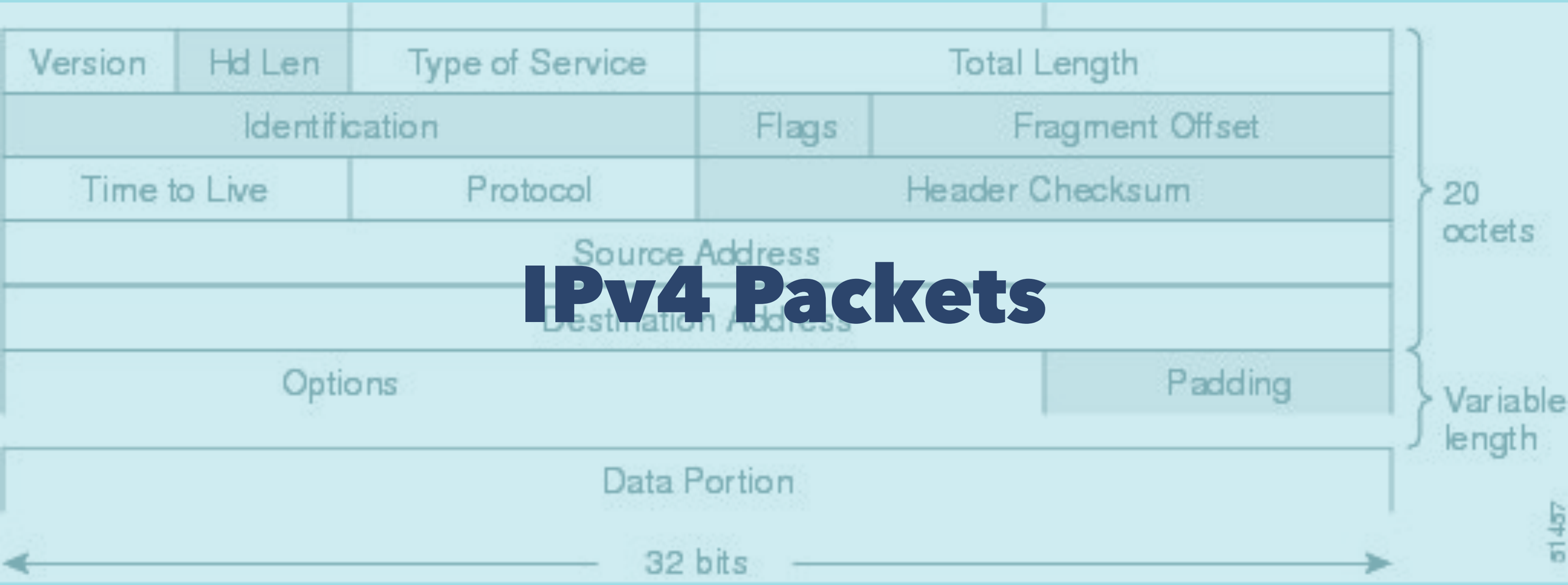
Packets

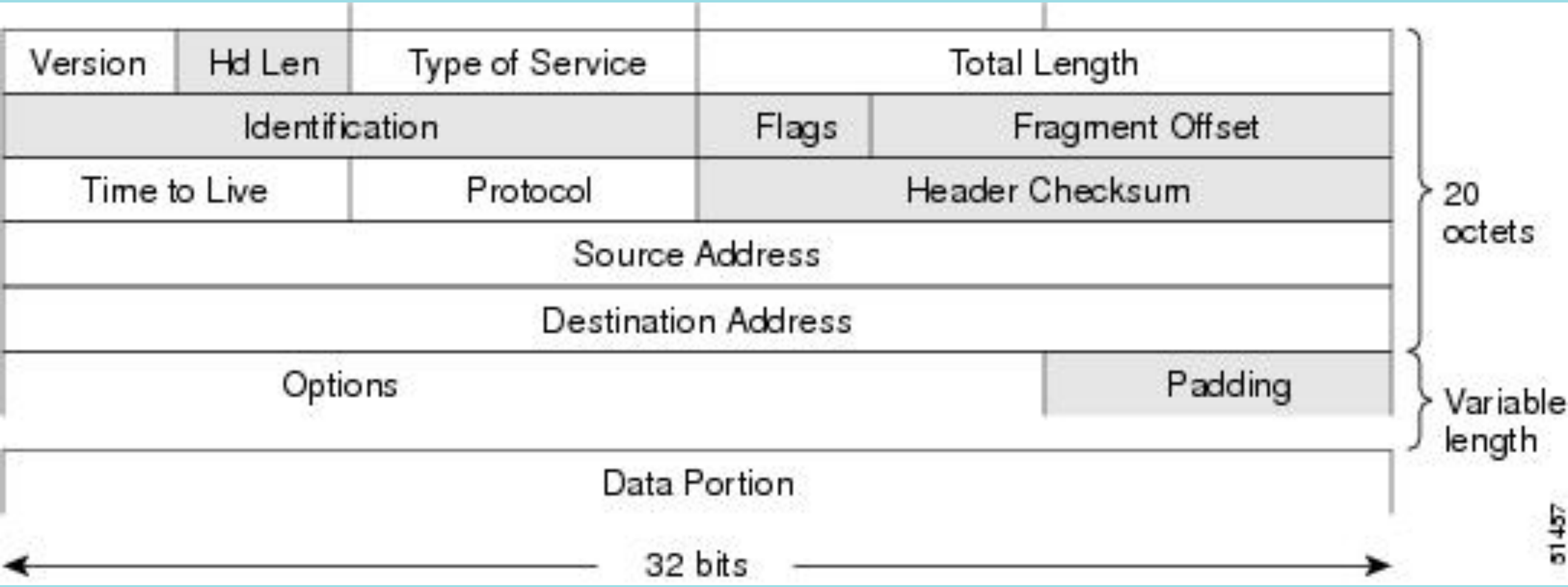
- Formatted unit of data
- Carried by a network

Packets

- Formatted unit of data
- Carried by a network
- Control information and payload

IPv4 Packets




Version
Hd Len
Type of Service
Total Length
Identification
Flags
Fragment Offset
Time to Live
Protocol
Header Checksum
Source Address
Destination Address
Options
Padding
Data Portion
20 octets
Variable length
32 bits
51457

Layer 4 Segments

TCP

(Segments)

TCP Segments

- "Transmission Control Protocol"

TCP Segments

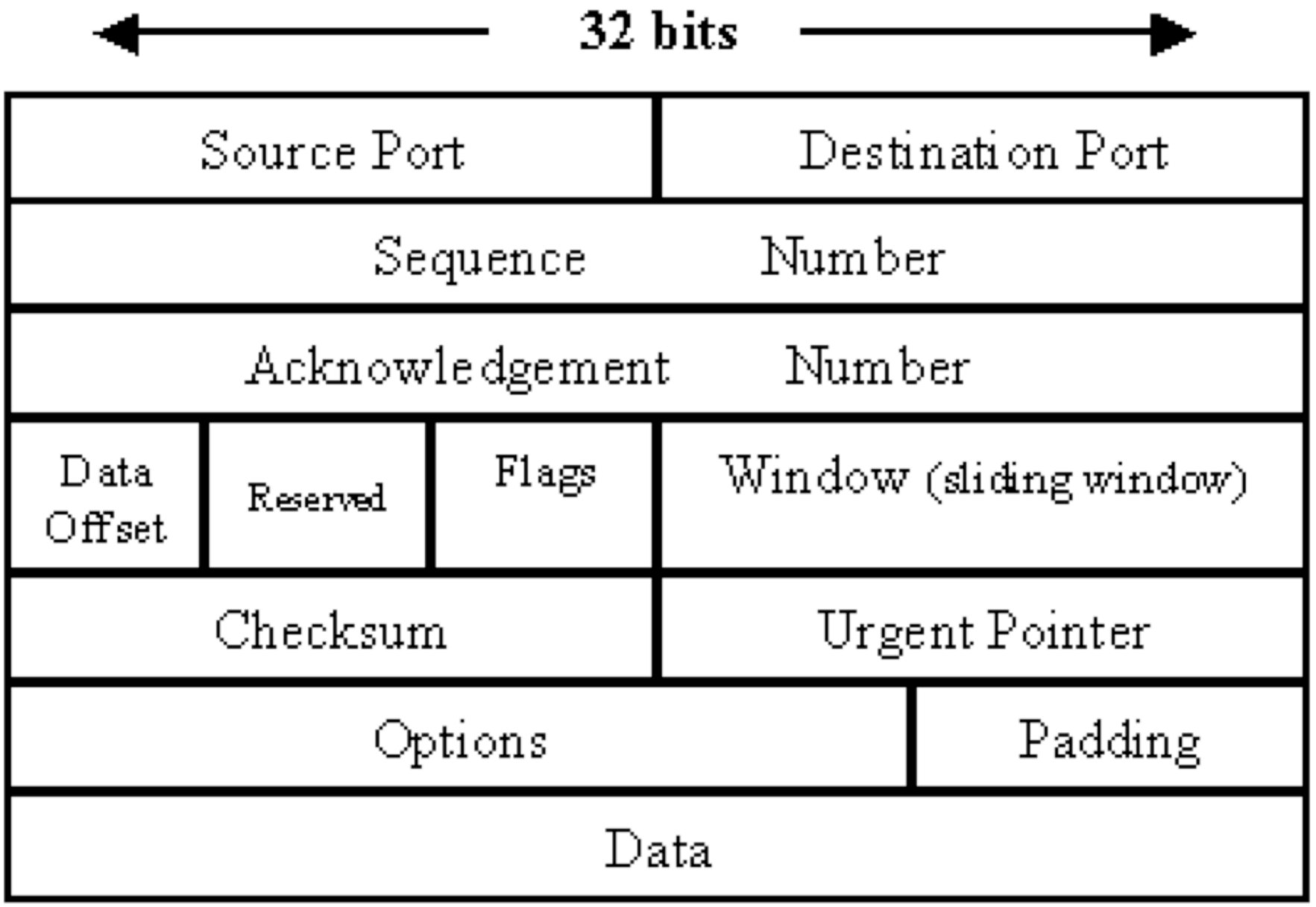
- "Transmission Control Protocol"
- Ordered

TCP Segments

- "Transmission Control Protocol"
- Ordered
- Error checked

TCP Segments

- "Transmission Control Protocol"
- Ordered
- Error checked
- Accepts a data stream and breaks it into chunks


32 bits
Source Port
Destination Port
Sequence Number
Acknowledgement Number
Data Offset
Reserved
Flags
Window (sliding window)
Checksum
Urgent Pointer
Options
Padding
Data

Putting it all together...

TCP/IP

- Work together to transfer data!

TCP/IP

- Work together to transfer data!
- IP packet has source/destination IP, TCP segment has source/destination port

TCP/IP

- Work together to transfer data!
- IP packet has source/destination IP, TCP segment has source/destination port

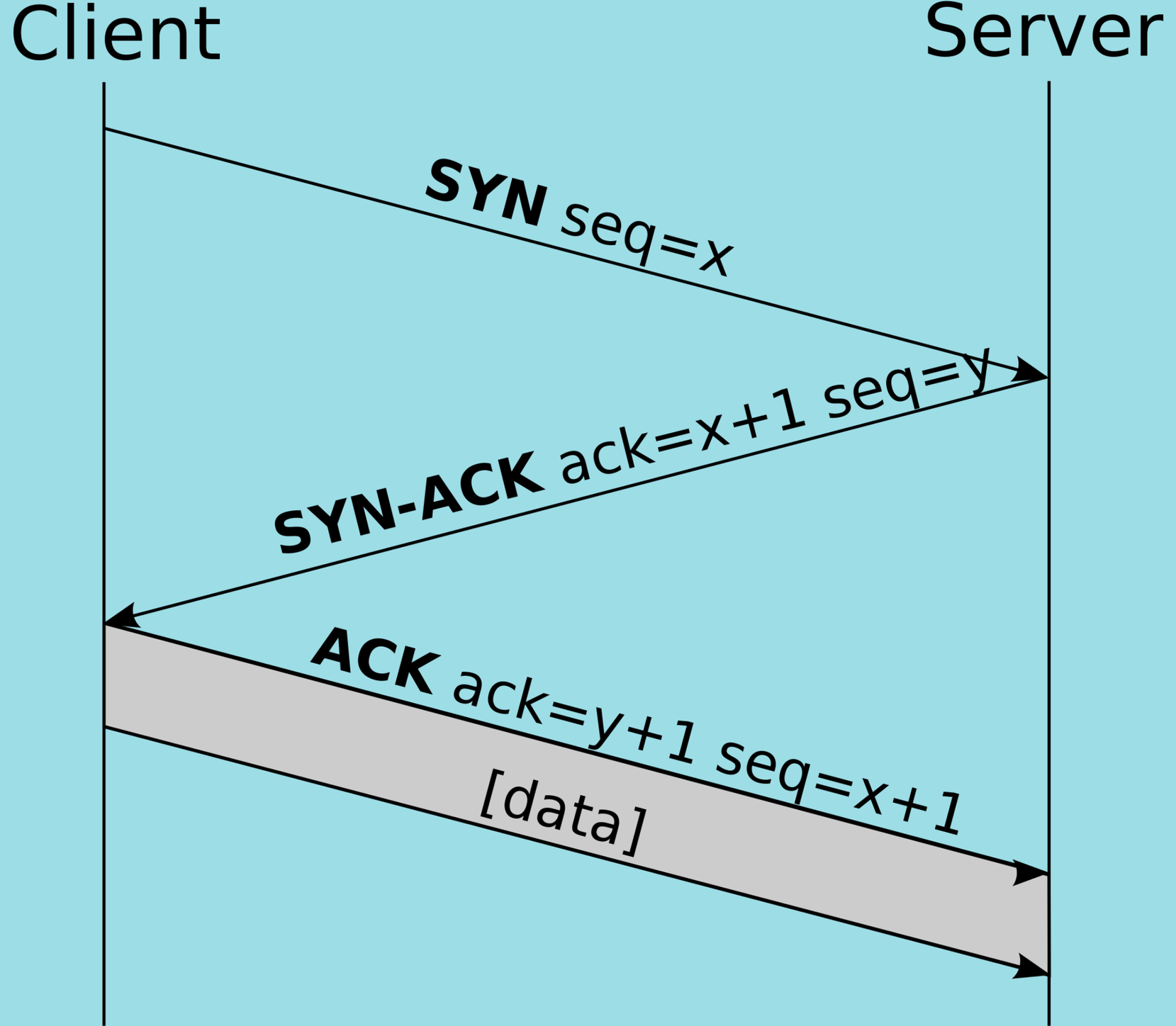
TCP Handshake

TCP Handshake

- A "three way handshake" before a client connects to a server

TCP Handshake

- A "three way handshake" before a client connects to a server
- Must be completed before data is exchanged


Client
Server
SYN seq=x
SYN-ACK ack=x+1 seq=y
ACK ack=y+1 seq=x+1
[data]

Cool dude, but how do I load my Tumblr dashboard?!

Cool dude, but how do I load my Twitter timeline?!

Cool dude, but how do I load my Facebook newsfeed?!

Cool dude, but how do I load my pictures of funny cats?!

Cool dude, but how do I load my pictures of cute cats?!

Or, basically...

How do we interact with things on the web?

Hypertext Transfer Protocol

Hypertext Transfer Protocol (HTTP)

Hypertext Transfer Protocol (HTTP)

- Provides methods (verbs) that we can perform

Hypertext Transfer Protocol (HTTP)

- Provides methods (verbs) that we can perform
- Defined GET, POST, and HEAD in HTTP/1.0

Hypertext Transfer Protocol (HTTP)

- Provides methods (verbs) that we can perform
- Defined GET, POST, and HEAD in HTTP/1.0
- Added OPTIONS, PUT, DELETE, TRACE, and CONNECT in HTTP/1.1

Hypertext Transfer Protocol (HTTP)

- Provides methods (verbs) that we can perform
- Defined **GET**, **POST**, and HEAD in HTTP/1.0
- Added OPTIONS, **PUT**, **DELETE**, TRACE, and CONNECT in HTTP/1.1

How does HTTP work?

How does HTTP work?

1. Client makes a request

How does HTTP work?

1. Client makes a request

```
Accept:text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding:gzip, deflate, sdch
Accept-Language:en-US,en;q=0.8,es;q=0.6,sl;q=0.4
Cache-Control:no-cache
Connection:keep-alive
DNT:1
Host:ufsit.org
Pragma:no-cache
Upgrade-Insecure-Requests:1
User-Agent:(trimmed)
```

How does HTTP work?

1. Client makes a request
2. Server responds!

How does HTTP work?

1. Client makes a request
2. Server responds!

```
Access-Control-Allow-Origin:*
Cache-Control:max-age=600
Content-Encoding:gzip
Content-Type:text/html; charset=utf-8
Date:Tue, 20 Oct 2015 07:24:25 GMT
Expires:Tue, 20 Oct 2015 07:34:25 GMT
Last-Modified:Mon, 19 Oct 2015 22:56:06 GMT
Server:GitHub.com
Transfer-Encoding:chunked
```

Ok, Wireshark time!

Need just HTTP info?

Chrome Inspector to the rescue!

APIs

APIs

- APIs normally use HTTP to communicate

APIs

- APIs normally use HTTP to communicate
- Different types of APIs, but want to focus on one

REST APIs

- REpresentational State Transfer (REST)

REST APIs

- REpresentational State Transfer (REST)
- Resources are identified with a URI
  - Ex: `/clubs/sit`

REST APIs

- REpresentational State Transfer (REST)
- Resources are identified with a URI
  - Ex: `/clubs/sit`
- Resources can be operated with HTTP verbs
  - Ex: `DELETE /clubs/sit`

HTTP Verbs

HTTP Verbs

- GET
- POST
- PUT
- DELETE

GET

- "Get" data from a resource

GET

- "Get" data from a resource
- Pretty easy, eh?

DELETE

- "Delete" the resource

POST

- "Create" a resource

PUT

- "Modify" a resource

Best way to communicate?

Postman

Postman

(punny, right?)