EECS 388

Introduction to Computer Security

Lecture 12:

Networking 102

Oct 5, 2023

Prof. Ensafi

Web and Network Security

Last two weeks:

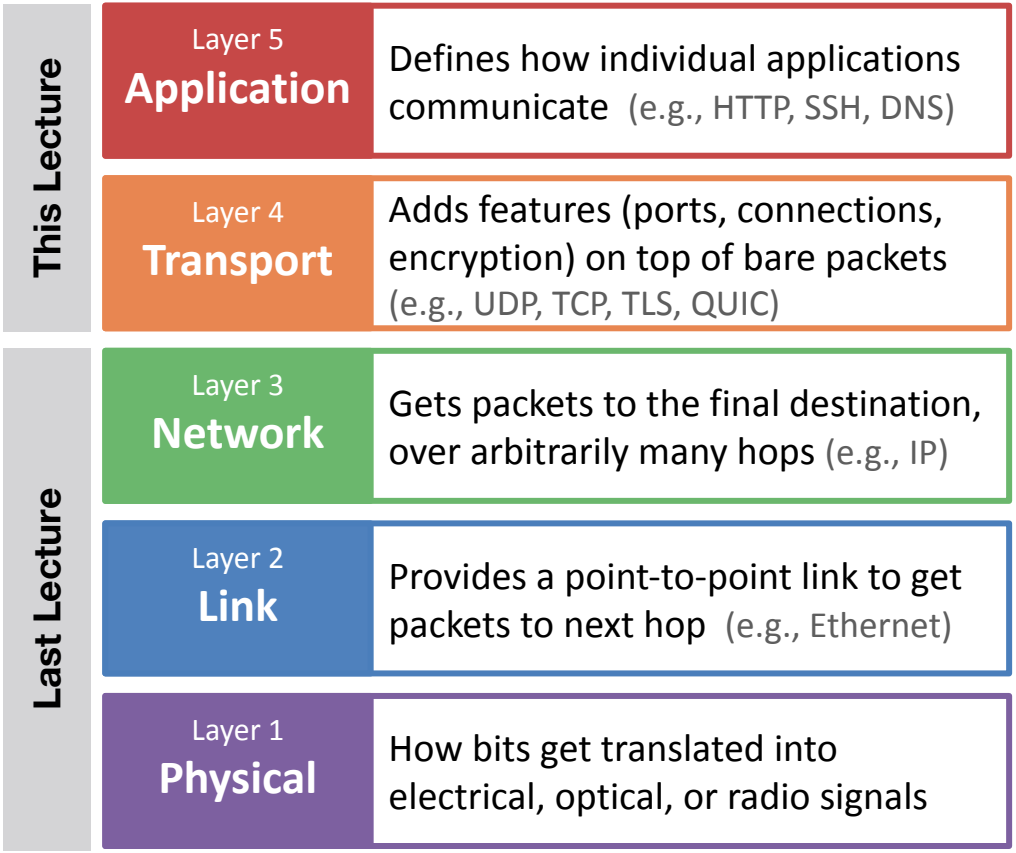
- The Web Platform
- Web Attacks and Defenses
- HTTPS and the Web PKI
- HTTPS Attacks and Defenses

This week:

- Networking 101
- **Networking 102**

Later:

- Network Defense
- Privacy and Anonymity
- Censorship and Circumvention

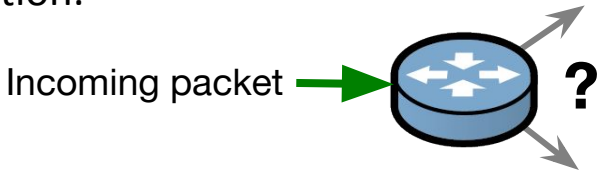


Routing and BGP

Network | Gets packets to final destination over arbitrarily many hops

A **router** is a device with multiple link layers, each part of a different network.

Examines incoming packet's destination IP and **forwards it** out a link that will get it closer to the destination.



Issue: How does each router know where to send the packet next?

ISPs use **Border Gateway Protocol (BGP)** to **announce** their network prefixes and connectivity to neighbors. Routers compute **route tables** from these announcements.

BGP has no authentication: Compromised ISP can *announce someone else's network*!

BGP hijacking is common, usually due to operator error, but sometime malicious. Packets to hijacked network are routed to attacker, who can eavesdrop or MiTM.

BORDER GATEWAY PROTOCOL ATTACK —

Suspicious event hijacks Amazon traffic for 2 hours, steals cryptocurrency

Almost 1,300 addresses for Amazon Route 53 rerouted for two hours.

DAN GOODIN - 4/24/2018, 3:00 PM

Defense: RPKI

Cryptographic method of signing records that associate a BGP route announcement with the correct originating ISP. *Slow adoption…*

Private Addresses/NAT

Network Gets packets to final destination over arbitrarily many hops

IPv4 Address Exhaustion

All ~2^{32} IPv4 addresses have been assigned, current cost is ~$40/IP on secondary markets. (IPv6 has ~2^{128} addresses–should be plenty!)

IPv4 Private Address Ranges

Three IPv4 networks are reserved for "private" use. ISPs won't route them, but can assign to your own devices for communicating on LAN:

10.0.0.0/8	(2^{24} addresses)
172.16.0.0/12	(2^{20} addresses)
192.168.0.0/16	(2^{16} addresses)

IPv4 Loopback Address

127.0.0.1 (`localhost`) always refers to the local machine. Not reachable from anywhere else.

Network Address Translation (NAT)

Most residential ISPs assign customers only one IPv4 address.

Issue: How to allow many devices at home?

Home routers assign each device an address from private a address range (via DHCP).

Router rewrites Internet-bound packets to replace private source address with its public IP address. Recognizes response packets and rewrites in reverse.

(This complicates hosting servers at home.)

Security benefit: Devices at private IPs not directly addressable from the output world.

Packet Capture and Analysis

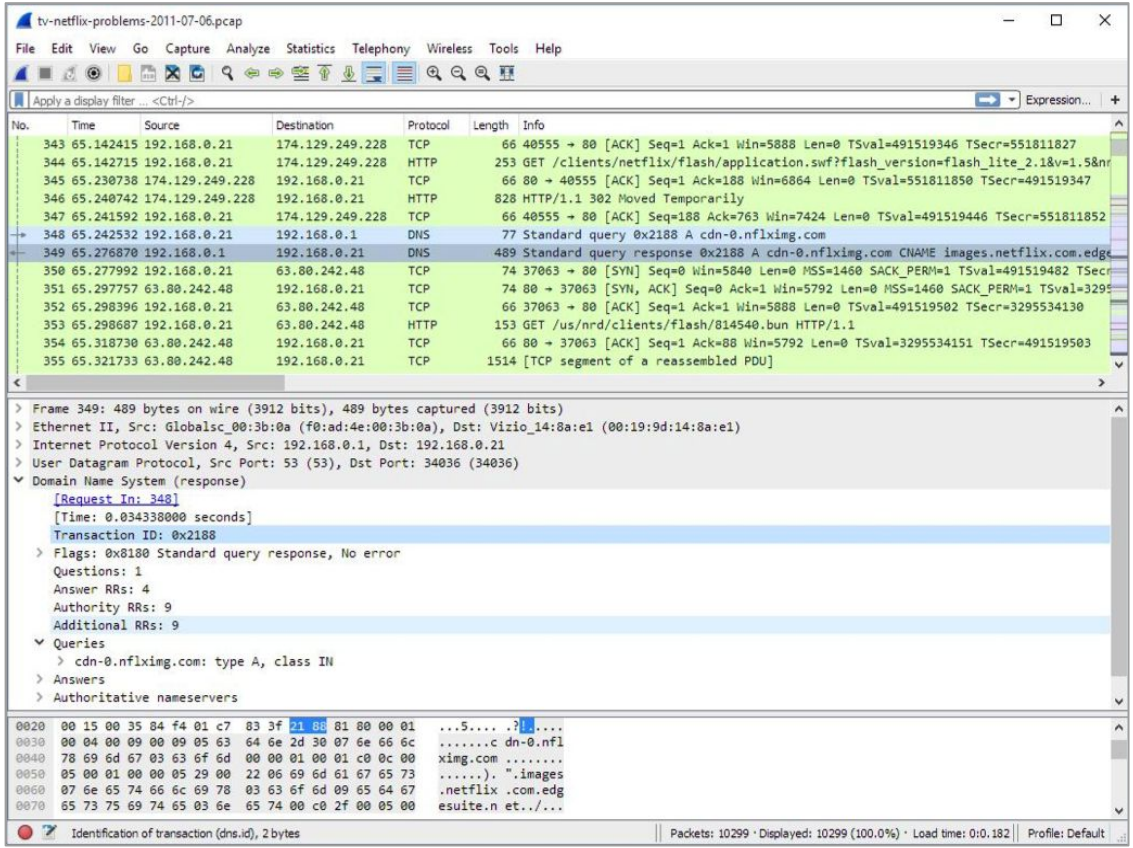
Hosts can record all packets arriving on their network interface ("**packet sniffing**").

Standard data format: **pcap**

Wireshark software:

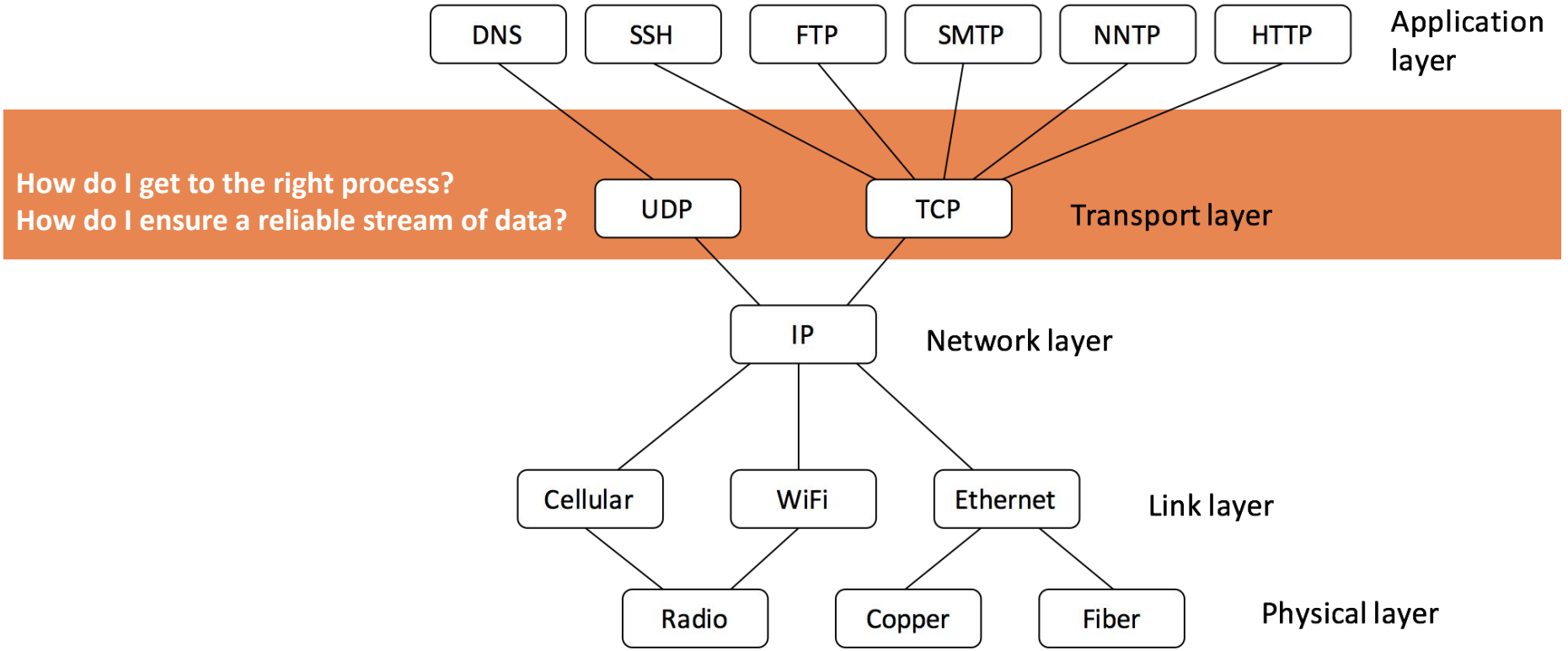
- Examine packets and decode each protocol layer.
- Filter packets by properties.
- Reassemble TCP packets to show complete data stream.
- Decrypt TLS, if provided the session key (normally requires control of client or server).

You'll experiment with Wireshark in lab this week and in Project 3.



Transport Layer

Transport	Adds features (ports, connections, encryption) on top of bare packets



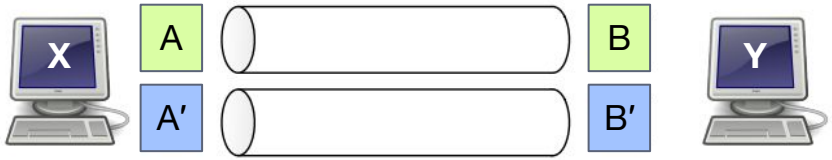
Ports

Transport Adds features (ports, connections, encryption) on top of bare packets

Issue: How do we send data to a *particular application process* running on a host?

Each application is identified by a 16-bit **port number** (1–65535).

TCP and UDP send data from port A on host X to port B on host Y.



With TCP, concurrent connections must have unique 4-tuples (X,A,Y,B).

Using different src. or dest. port allows multiple connections between same hosts.

Many destination port numbers are used for specific applications **by convention** (but servers can choose any port). Examples:

Port	Application
80	HTTP (web)
443	HTTPS (web via TLS)
25	SMTP (mail)
67	DHCP (host configuration)
22	SSH (secure shell)
23	Telnet (plaintext shell)

Ports <1024 are “privileged” – requires root

For source ports, OS usually picks for you, uses high-numbered **ephemeral ports**, frequently in the range 32768-65535.

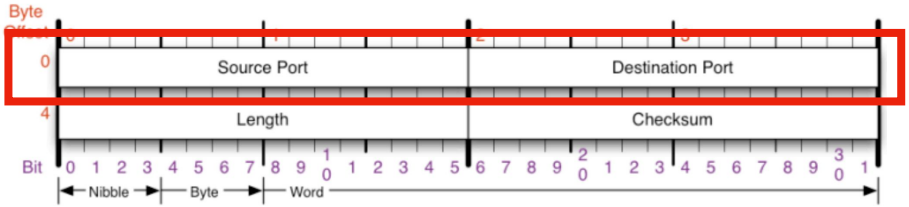
UDP

Transport Adds features (ports, connections, encryption) on top of bare packets

User Datagram Protocol (UDP)

is a transport-layer protocol that is essentially just a wrapper around IP that adds ports to demultiplex traffic by application.

UDP header:



Applications that use UDP:

- DNS
- QUIC
- Many multiplayer games and real-time communications apps

UDP properties:

- Connectionless.
- Unreliable data transfer.
- No flow control.
- No congestion control.
- **No security.**

UDP is highly vulnerable to **impersonation**: If **off-path** attacker knows what ports a client and server are using, it can inject packets (with spoofed source IP addresses) that appear to be part of the communication.

Using UDP allows applications to provide missing properties at a higher level as needed, which sometimes allows better performance.

TCP

Transport Adds features (ports, connections, encryption) on top of bare packets

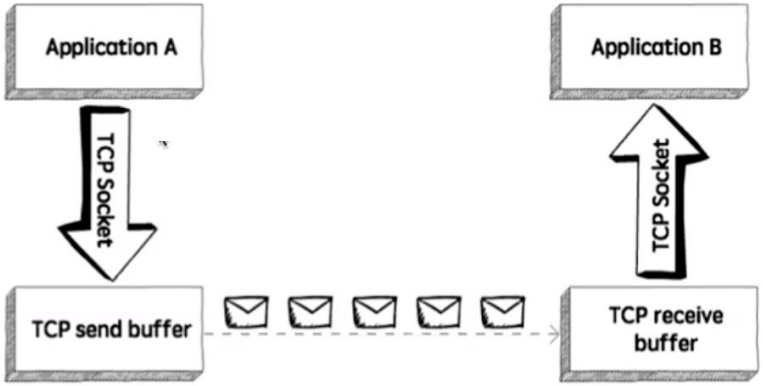
Transmission Control Protocol (TCP)

is a transport-layer protocol providing:

- **Connection-oriented semantics**
- **A reliable, bi-directional *byte stream***
- **Congestion control**

TCP does not provide: strong security.

Data carried in **segments**, each one IP packet.

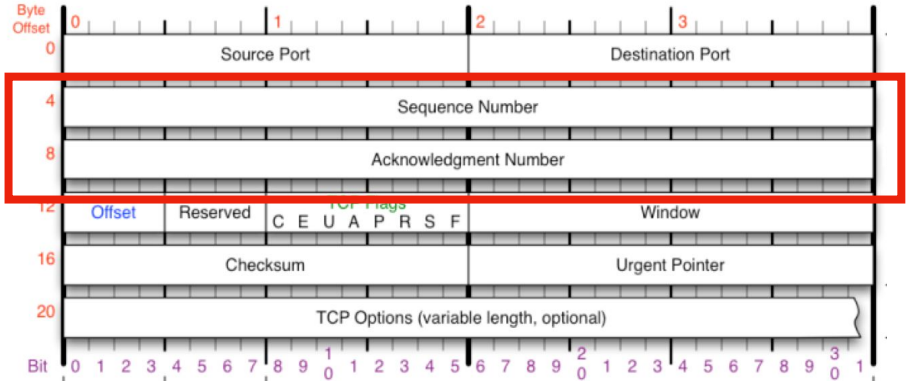


TCP provides these despite IP packets being dropped, re-ordered, and duplicated.

Each segment has 32-bit **sequence number** (where this data fits in the stream) and **acknowledgement number** (what data sender expects to receive next).

Receiver **reassembles** segments as output stream. Sender **retransmits** unacknowledged segments.

TCP header:



TCP Handshake

Transport Adds features (ports, connections, encryption) on top of bare packets

TCP introduces the notion of a **session**.

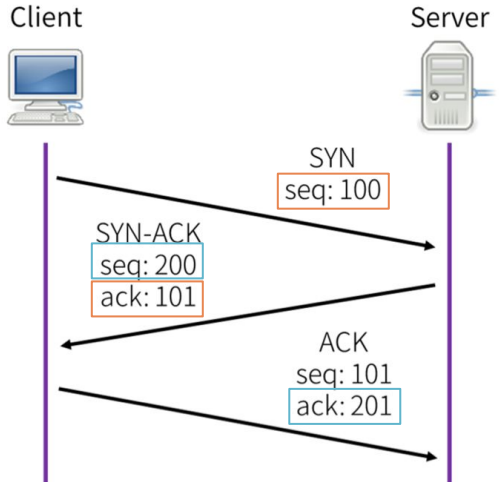
Server **listens** **on a port** to allow connections.
Clients **connect** **to the port** at the server's IP.

Every TCP session begins with a **three-way handshake**.

Special packets with **flag bits** set:

1. SYN
2. SYN+ACK
3. ACK

Client and server each pick random 32-bit **initial sequence numbe** acknowledge the other'



***On-path* attacks against TCP are trivial.**

Can **off-path** attackers ***impersonate*** another host when initiating a TCP connection?

Off-path attacker can spoof src. IP address and send initial SYN packet to server… but **can't complete three-way handshake** without guessing server's sequence number. (In theory, only 1 in 2^{32} chance.)

Can **off-path** attackers ***disrupt*** a TCP session?

In a **TCP reset attack**, attacker spoofs a packet with RST flag set. Needs right port numbers (~16 bits). Receive will close the connection if sequence number is within window size W of correct. W = 32-64K on modern OSes. (In theory, only 1 in $2^{16+32}/W$ chance.)

In practice, less-than-random values and clever **side-channels** often make off-path attacks easier.

Socket Programming

Transport Adds features (ports, connections, encryption) on top of bare packets

TCP/IP is typically implemented by the OS.

The **socket API** is a widely supported interface that let apps utilize TCP, UDP, other transports.

You'll experiment with sockets in Project 3.

Examples below use the socket API from Python to implement "echo" client and server over TCP.

```python
# tcp_echo_client.py
import socket

# Create TCP/IP socket and connect to destination server/port
server_address = ('127.0.0.1', 10000)
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print('connecting to %s port %s' % server_address)
sock.connect(server_address)

# Send data
message = b"Don't forget! TCP is not encrypted."
print('sending "%s"' % message)
sock.sendall(message)

# Read and output response
remaining = len(message)
while remaining > 0:
    data = sock.recv(4096)
    remaining -= len(data)
    print('received "%s"' % data)

print('closing socket')
sock.close()
```

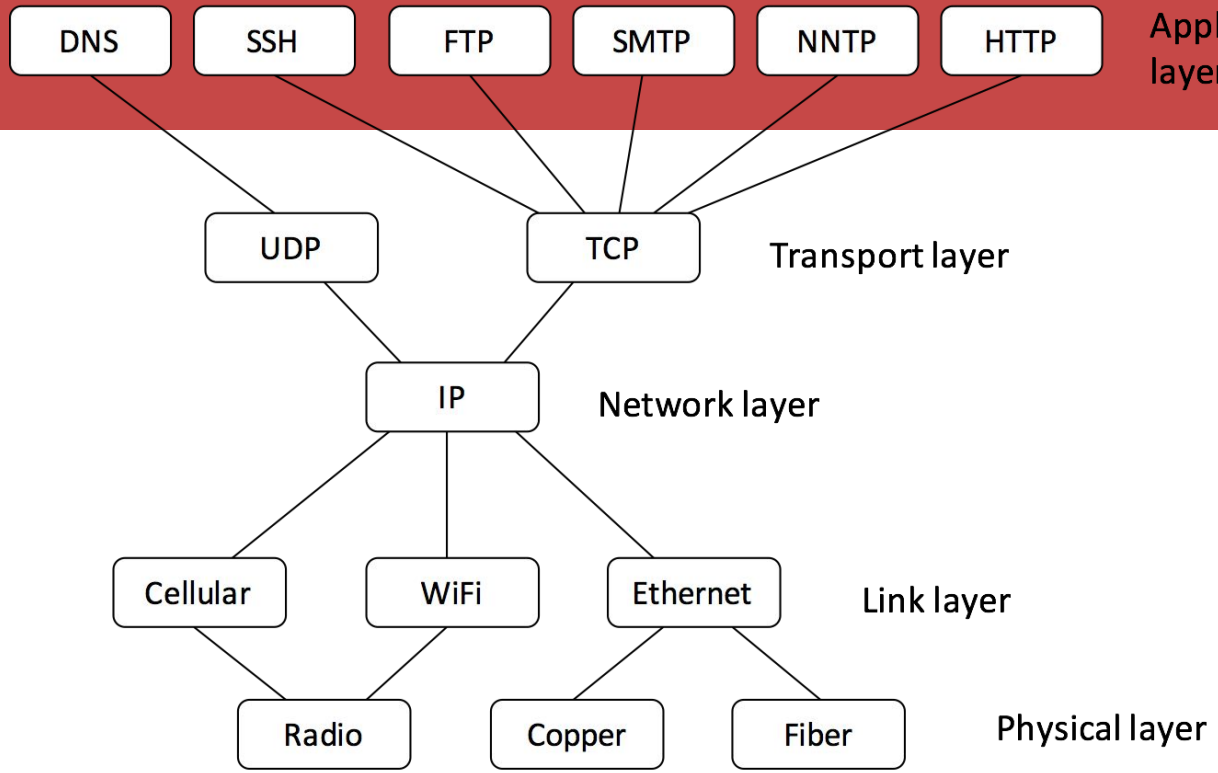
```python
# tcp_echo_server.py
import socket

# Create a TCP/IP socket listening on port 10000
server_address = ('0.0.0.0', 10000)
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind(server_address)
sock.listen(1)
print('listening on IP %s port %s' % server_address)
while True:
    connection, client_address = sock.accept()
    print('connection from', client_address)
    try:
        # Receive the data in small chunks and retransmit it
        while True:
            data = connection.recv(4096)
            if data:
                print( 'received "%s", echoing' % data)
                connection.sendall(data)
            else: break
    finally:
        print('closing connection from', client_address)
        connection.close()
```

Application Layer

Application	Defines how individual applications communicate

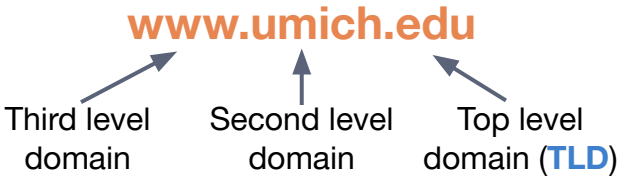
How does the application structure data?



DNS

Application | Defines how individual applications communicate

The **Domain Name System (DNS)** is a delegatable, hierarchical database.



DNS is used, e.g., to resolve **hostnames** to IP addresses:

```
$ host www.umich.edu
www.umich.edu has address 141.211.243.251
www.umich.edu has IPv6 address 2607:f018:1:1::1
```

Decentralized design achieves global scale and avoids any single point of failure.

DNS operates at the application layer. OSes provide DNS client ("**name resolution**") service.

DNS associates names with **resource records**. Many **record types**, e.g.:

A	IPv4 address
AAAA	IPv6 address
MX	Inbound mail server
CAA	Certificate authority authorization
CNAME	Alias to another name
TXT	Arbitrary text
NS	Delegated name server

Two kinds of **DNS servers**:

Recursive servers: Answer queries from clients by asking other servers. ISPs/orgs. run them; also public servers: CloudFlare (**1.1.1.1**), Google (**8.8.8.8**)

Authoritative servers: Domain owner run them to gives definitive answers for its part of namespace.

DNS Zones

Application | Defines how individual applications communicate

Control over each region of namespace ("**zone**") delegated to **authoritative** servers.

E.g., the zone "**umich.edu**" (which contains "www.umich.edu") has three authoritative servers:

```
$ dig www.umich.edu
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 5635
;; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 4

;; QUESTION SECTION:
;www.umich.edu.                 IN    A

;; ANSWER SECTION:
www.umich.edu.          1800    IN    A    141.211.243.251

;; AUTHORITY SECTION:
www.umich.edu.          86092   IN    NS   dns1.itd.umich.edu.
www.umich.edu.          86092   IN    NS   dns2.itd.umich.edu.
www.umich.edu.          86092   IN    NS   dns3.umich.edu.
```

The **root zone** (".") contains all TLDs. Controlled by 13 authoritative servers.

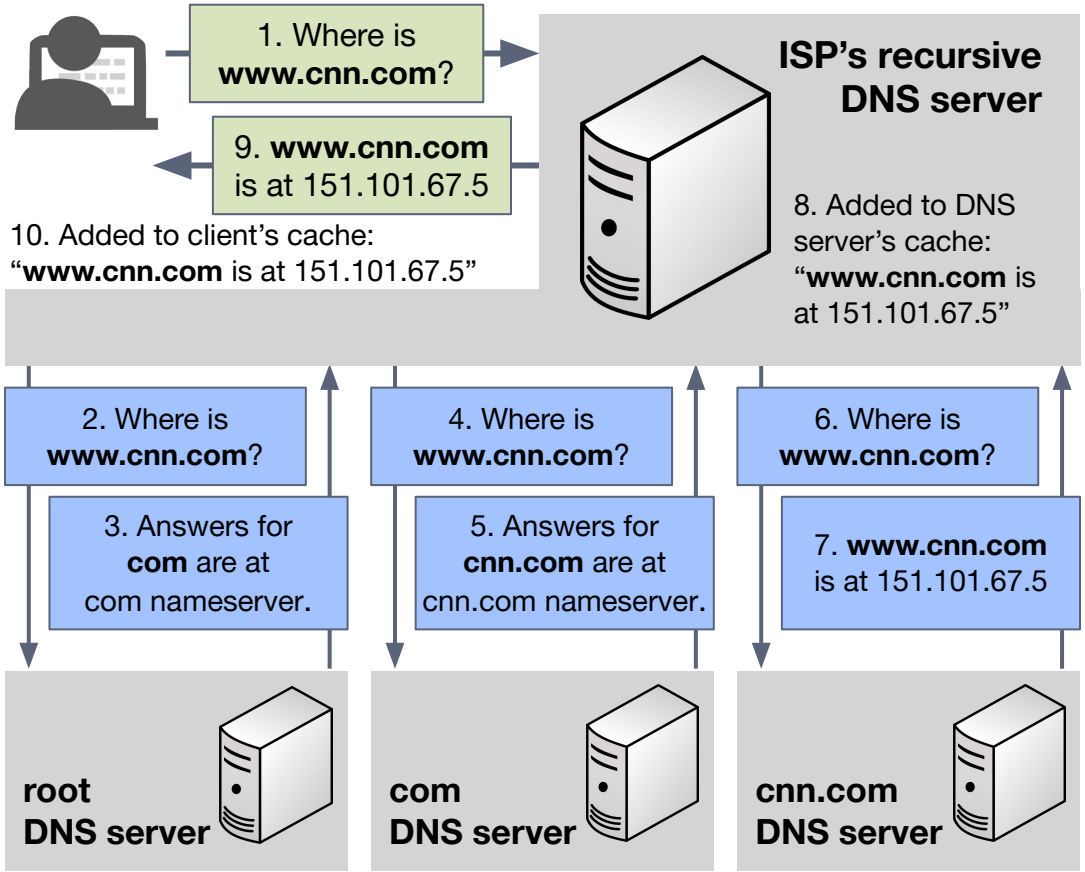
DNS servers are provisioned with these IPs:

HOSTNAME	IP ADDRESSES	OPERATOR
a.root-servers.net	198.41.0.4, 2001:503:ba3e::2:30	Verisign, Inc.
b.root-servers.net	199.9.14.201, 2001:500:200::b	University of Southern California, Information Sciences Institute
c.root-servers.net	192.33.4.12, 2001:500:2::c	Cogent Communications
d.root-servers.net	199.7.91.13, 2001:500:2d::d	University of Maryland
e.root-servers.net	192.203.230.10, 2001:500:a8::e	NASA (Ames Research Center)
f.root-servers.net	192.5.5.241, 2001:500:2f::f	Internet Systems Consortium, Inc.
g.root-servers.net	192.112.36.4, 2001:500:12::d0d	US Department of Defense (NIC)
h.root-servers.net	198.97.190.53, 2001:500:1::53	US Army (Research Lab)
i.root-servers.net	192.36.148.17, 2001:7fe::53	Netnod
j.root-servers.net	192.58.128.30, 2001:503:c27::2:30	Verisign, Inc.
k.root-servers.net	193.0.14.129, 2001:7fd::1	RIPE NCC
l.root-servers.net	199.7.83.42, 2001:500:9f::42	ICANN
m.root-servers.net	202.12.27.33, 2001:dc3::35	WIDE Project

DNS hierarchy: Root servers delegate each TLD to separate servers, which delegate each second-level domain, etc.

Name Resolution

Application Defines how individual applications communicate



DNS servers and clients cache responses to avoid extra lookups.

Each record has a **time-to-live (TTL)**, set by authoritative server.

Decremented each second. When it reaches zero, record is discarded.

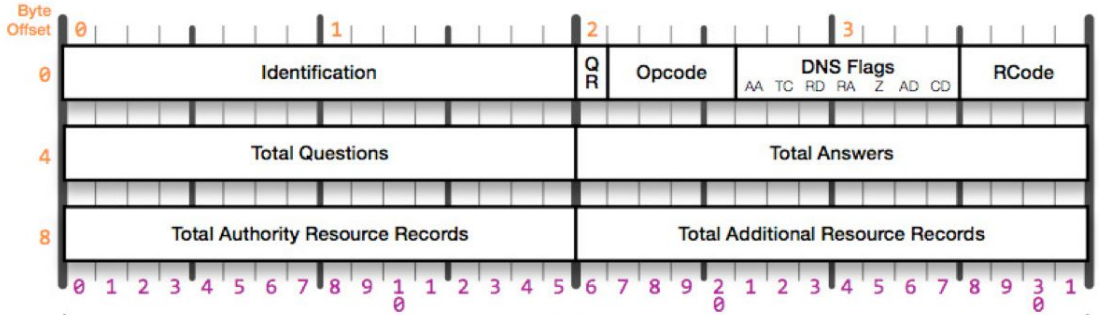
DNS Packets

Application Defines how individual applications communicate

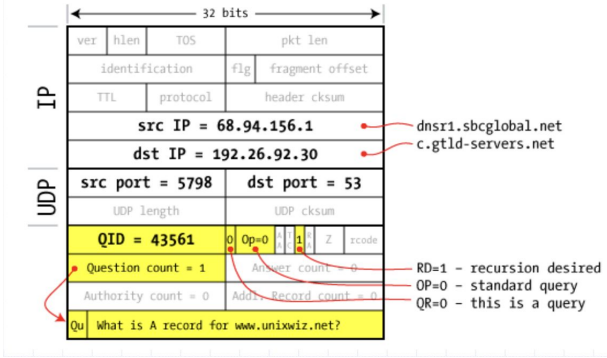
DNS (typically) uses UDP packets.

Four sections: questions, answers, authority, additional records

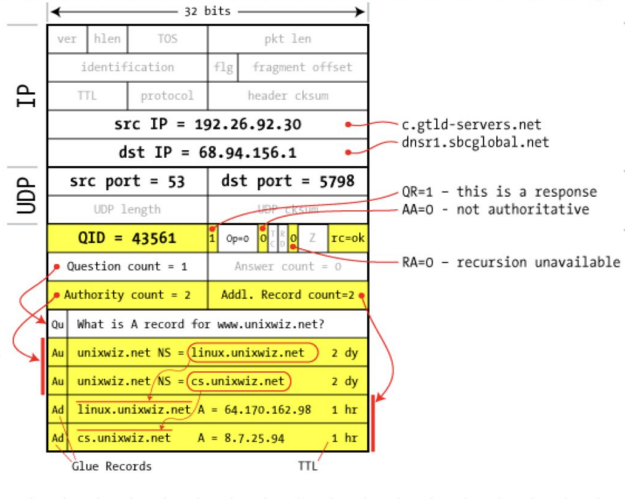
Identification: 16-bit random value (QID) that links response to query



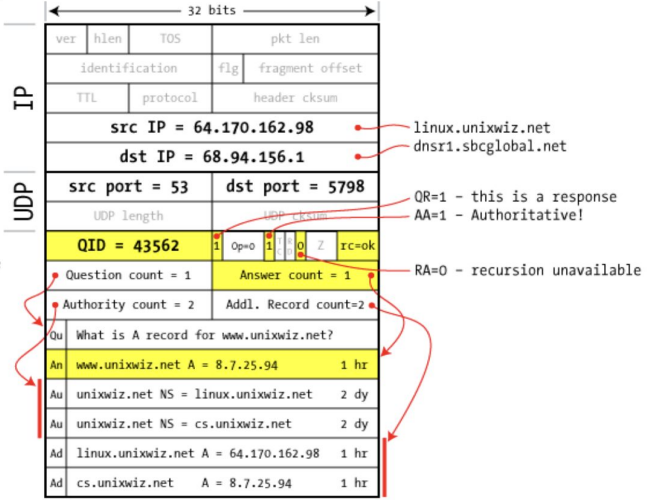
Request Packet



Response Packet



Authoritative Response Packet



You'll experiment with DNS packets in Project 3.

Attacking DNS

Application | Defines how individual applications communicate

DNS compromise allows MITM attacks.

An attacker who subverts DNS can return malicious responses that direct clients to visit attacker's server in place of real server.

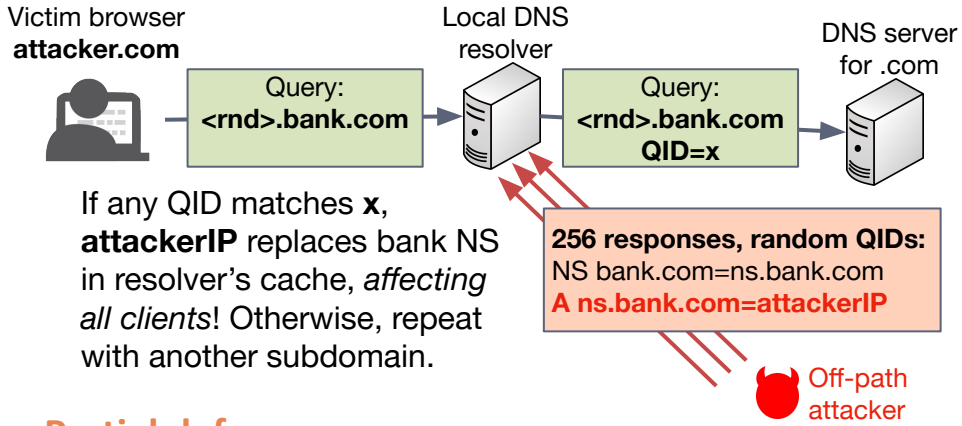
Examples of DNS attacks:

- **DNS hijacking:** Attacker hacks into home router, changes DHCP settings to point clients to attacker-controller DNS server…
- **Hosts file hijacking:** Malware edits OS `hosts` file to redirect target sites to malicious server.
- **DNS monitoring** and **DNS blocking:** ISP monitors DNS requests to track users. Injects fake responses to block censored sites.

Off-path DNS cache poisoning

Attack discovered by Dan Kaminsky in 2008.

DNS authenticates responses with 16-bit QueryID. Valid responses may include names not requested, if same DNS server controls them ("in bailiwick")...



Partial defense:

Randomize UDP source port (16 bits of entropy).

Randomize capitalization in query (1 bit per letter).

Encrypted DNS

Application Defines how individual applications communicate

DNSSEC adds authentication and integrity to DNS responses.

- Authoritative DNS servers sign DNS responses using cryptographic key.
- Clients can verify that a response is legitimate by checking signature through PKI similar to HTTPS.
- ***Does not*** provide confidentiality. Requests and responses are plaintext.

Issue: Slow adoption by domains and resolvers. Most people don't use DNSSEC and probably never will. Need other defenses.

DNS-over-TLS and **DNS-over-HTTPS** operate DNS over an encrypted channel.

- DNS packets, but using TLS or HTTPS as transport layer instead of plaintext UDP.
- Also JSON: https://dns.google/query?name=eecs388.org
- Provides confidentiality and integrity for connection from client to DNS server.
- ***Does not*** guarantee results authenticated. Server's onward queries to authoritative servers may fall back to UDP.

Chrome and Firefox now use encrypted DNS where available. However, by default they fall back to UDP when encrypted DNS is blocked…***allows downgrade attacks.***

Coming Up

Reminders:

Web Project due Today at 6 PM

Midterm Exam, Friday, Oct. 20, 7–8:30 p.m.

Monday

Network Defense

Denial of service attacks; firewalls, IDSes, VPNs, zero trust

Wednesday

Authentication & Passwords

Passwords, online and offline guessing