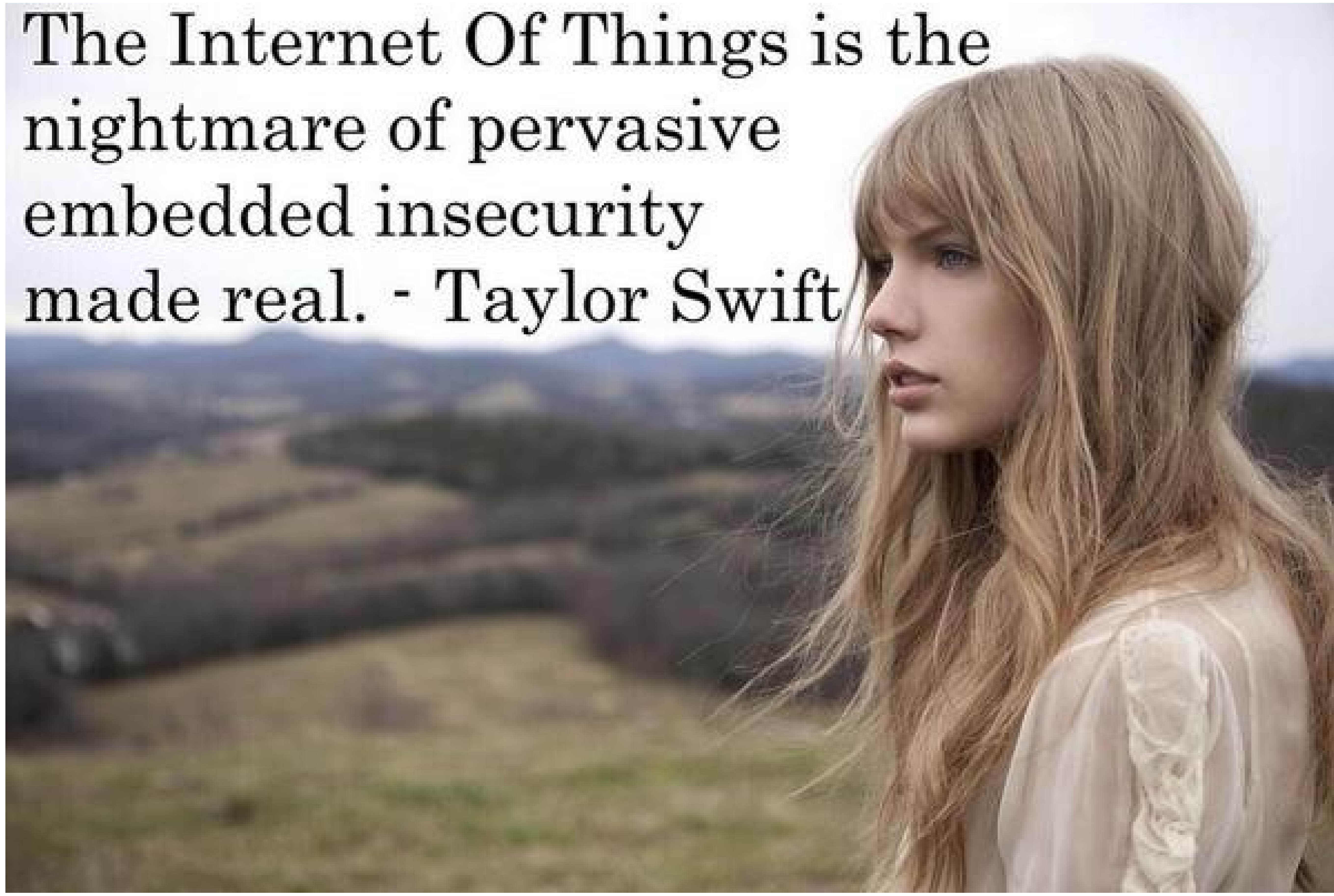
Network Security 2


The Internet Of Things is the nightmare of pervasive embedded insecurity made real. - Taylor Swift

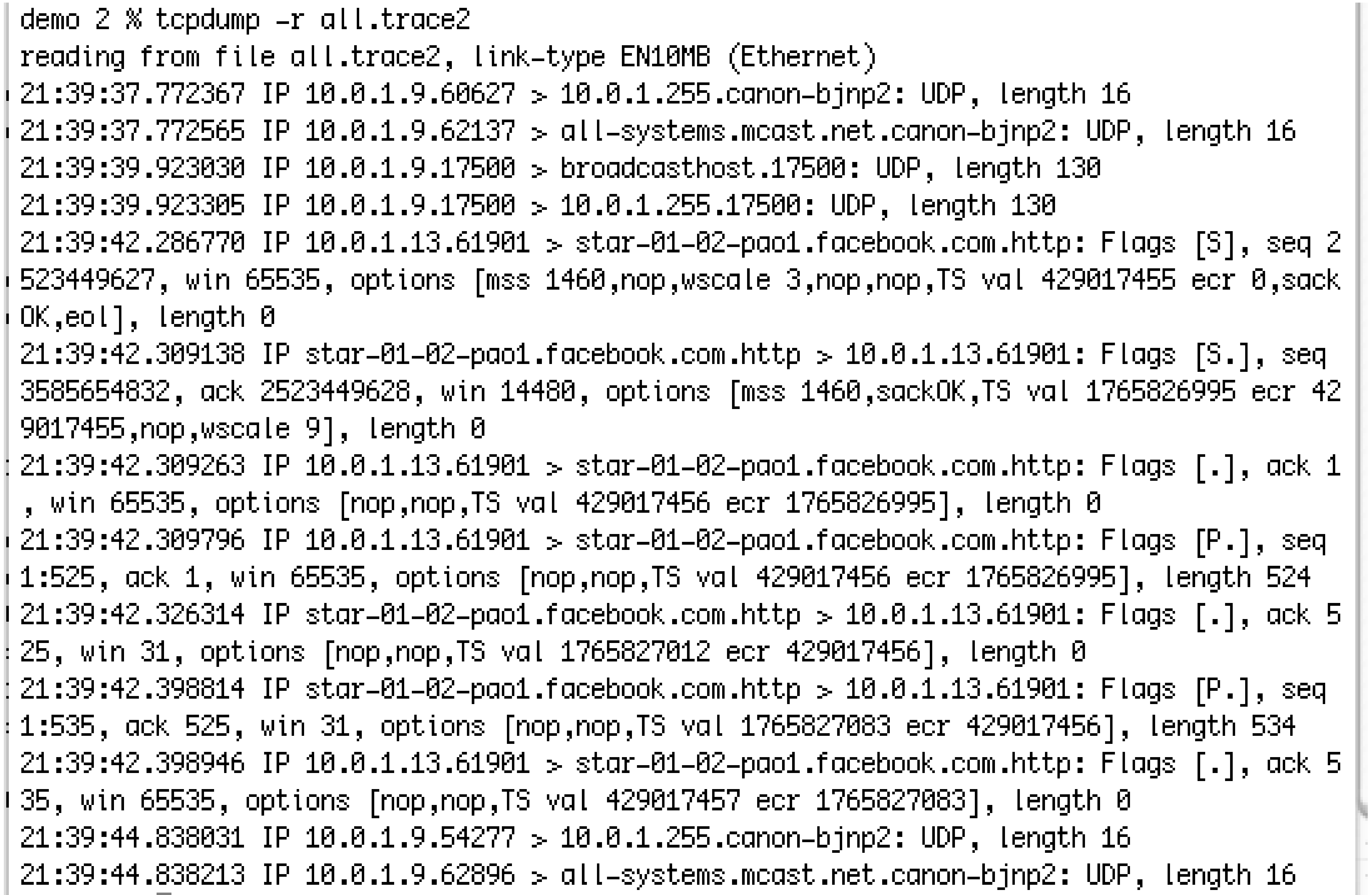
Announcements

- MT1 grades released
- Project 2
  - Desgin Doc draft due Monday
  - Project due July 29
    - proj2.go
    - proj2_test.go
    - Design Doc

Physical/Link-Layer Threats: Eavesdropping

- Also termed ***sniffing***
- For subnets using broadcast technologies (e.g., WiFi, some types of Ethernet), get it for “free”
  - Each attached system’s NIC (= Network Interface Card) can capture any communication on the subnet
  - Some handy tools for doing so
    - **`tcpdump`** (low-level ASCII printout)

TCPDump

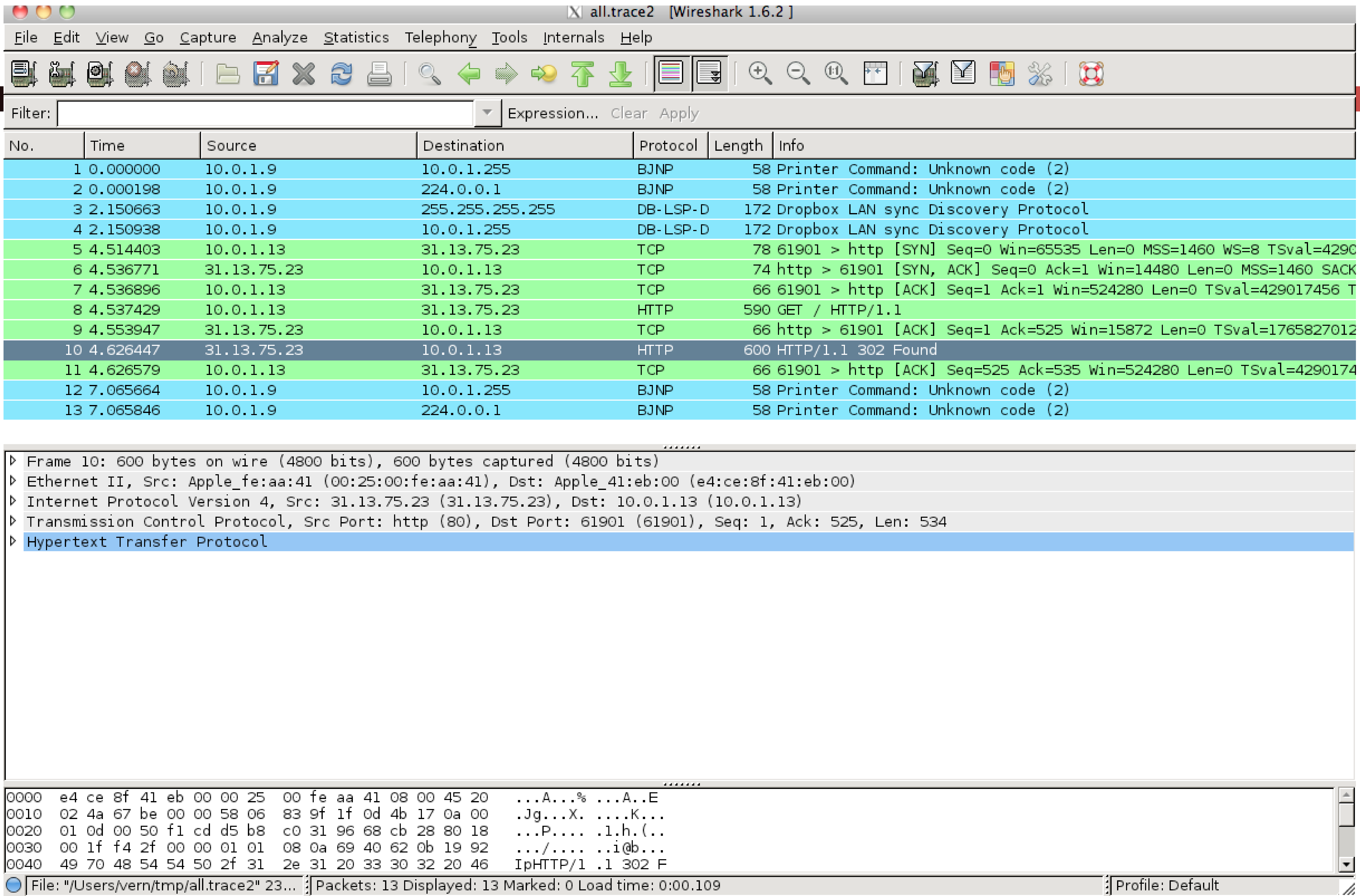


```
demo 2 % tcpdump -r all.trace2
reading from file all.trace2, link-type EN10MB (Ethernet)
21:39:37.772367 IP 10.0.1.9.60627 > 10.0.1.255.canon-bjnp2: UDP, length 16
21:39:37.772565 IP 10.0.1.9.62137 > all-systems.mcast.net.canon-bjnp2: UDP, length 16
21:39:39.923030 IP 10.0.1.9.17500 > broadcasthost.17500: UDP, length 130
21:39:39.923305 IP 10.0.1.9.17500 > 10.0.1.255.17500: UDP, length 130
21:39:42.286770 IP 10.0.1.13.61901 > star-01-02-pao1.facebook.com.http: Flags [S], seq 2
523449627, win 65535, options [mss 1460,nop,wscale 3,nop,nop,TS val 429017455 ecr 0,sack
OK,eol], length 0
21:39:42.309138 IP star-01-02-pao1.facebook.com.http > 10.0.1.13.61901: Flags [S.], seq
3585654832, ack 2523449628, win 14480, options [mss 1460,sackOK,TS val 1765826995 ecr 42
9017455,nop,wscale 9], length 0
21:39:42.309263 IP 10.0.1.13.61901 > star-01-02-pao1.facebook.com.http: Flags [.], ack 1
, win 65535, options [nop,nop,TS val 429017456 ecr 1765826995], length 0
21:39:42.309796 IP 10.0.1.13.61901 > star-01-02-pao1.facebook.com.http: Flags [P.], seq
1:525, ack 1, win 65535, options [nop,nop,TS val 429017456 ecr 1765826995], length 524
21:39:42.326314 IP star-01-02-pao1.facebook.com.http > 10.0.1.13.61901: Flags [.], ack 5
25, win 31, options [nop,nop,TS val 1765827012 ecr 429017456], length 0
21:39:42.398814 IP star-01-02-pao1.facebook.com.http > 10.0.1.13.61901: Flags [P.], seq
1:535, ack 525, win 31, options [nop,nop,TS val 1765827083 ecr 429017456], length 534
21:39:42.398946 IP 10.0.1.13.61901 > star-01-02-pao1.facebook.com.http: Flags [.], ack 5
35, win 65535, options [nop,nop,TS val 429017457 ecr 1765827083], length 0
21:39:44.838031 IP 10.0.1.9.54277 > 10.0.1.255.canon-bjnp2: UDP, length 16
21:39:44.838213 IP 10.0.1.9.62896 > all-systems.mcast.net.canon-bjnp2: UDP, length 16
```

Physical/Link-Layer Threats: Eavesdropping

- Also termed ***sniffing***
- For subnets using broadcast technologies (e.g., WiFi, some types of Ethernet), get it for "free"
  - Each attached system's NIC (= Network Interface Card) can capture any communication on the subnet
  - Some handy tools for doing so
    - **`tcpdump`** (low-level ASCII printout)
    - Wireshark (higher-level printing)

Wireshark: GUI for Packet Capture/Exam.



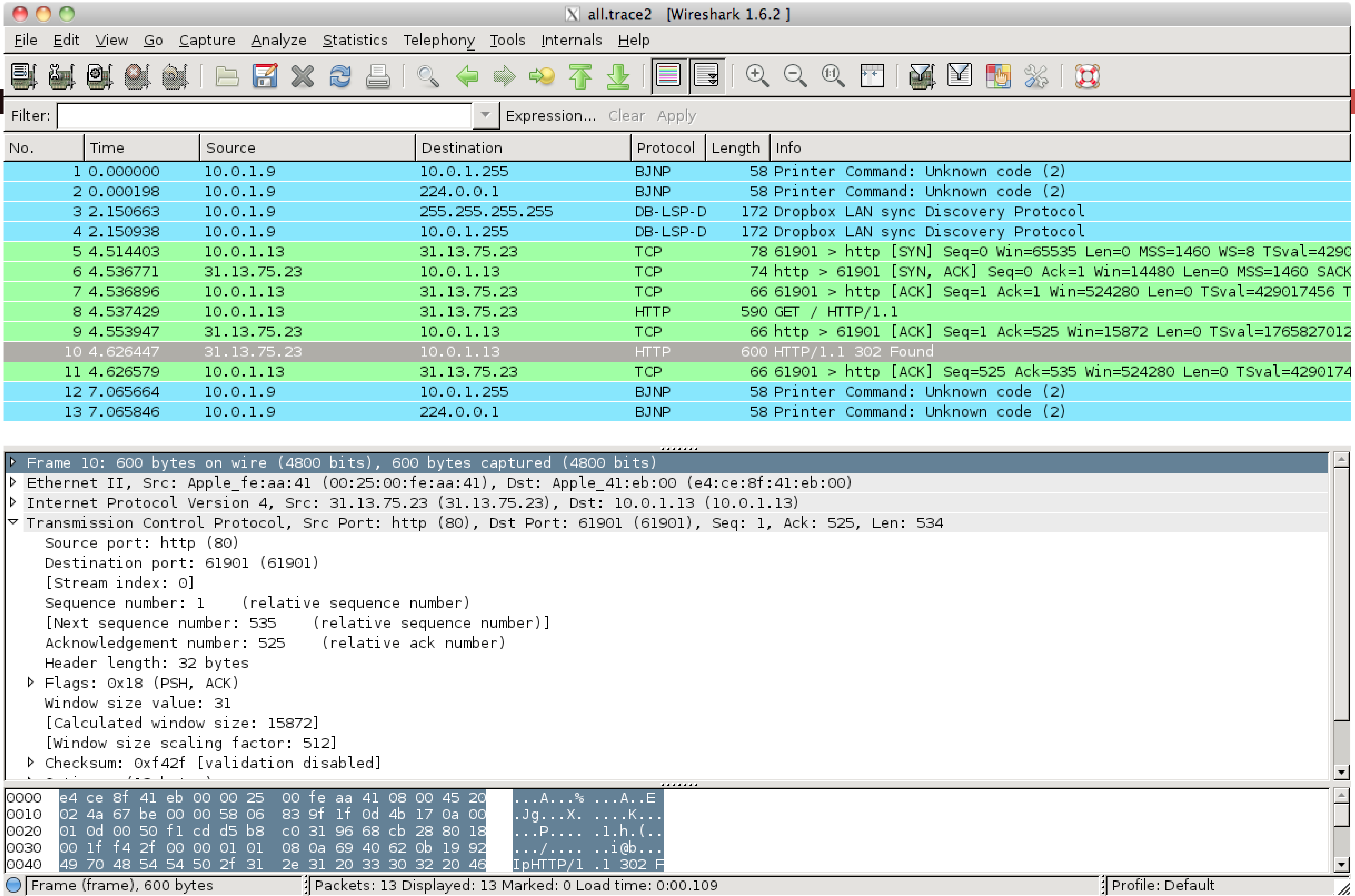
```
No.  Time      Source        Destination      Protocol  Length Info
   1 0.000000  10.0.1.9      10.0.1.255       BJNP         58 Printer Command: Unknown code (2)
   2 0.000198  10.0.1.9      224.0.0.1        BJNP         58 Printer Command: Unknown code (2)
   3 2.150663  10.0.1.9      255.255.255.255  DB-LSP-D    172 Dropbox LAN sync Discovery Protocol
   4 2.150938  10.0.1.9      10.0.1.255       DB-LSP-D    172 Dropbox LAN sync Discovery Protocol
   5 4.514403  10.0.1.13     31.13.75.23      TCP          78 61901 > http [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=8 TSval=4290
   6 4.536771  31.13.75.23   10.0.1.13        TCP          74 http > 61901 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK
   7 4.536896  10.0.1.13     31.13.75.23      TCP          66 61901 > http [ACK] Seq=1 Ack=1 Win=524280 Len=0 TSval=429017456 T
   8 4.537429  10.0.1.13     31.13.75.23      HTTP        590 GET / HTTP/1.1
   9 4.553947  31.13.75.23   10.0.1.13        TCP          66 http > 61901 [ACK] Seq=1 Ack=525 Win=15872 Len=0 TSval=1765827012
  10 4.626447  31.13.75.23   10.0.1.13        HTTP        600 HTTP/1.1 302 Found
  11 4.626579  10.0.1.13     31.13.75.23      TCP          66 61901 > http [ACK] Seq=525 Ack=535 Win=524280 Len=0 TSval=4290174
  12 7.065664  10.0.1.9      10.0.1.255       BJNP         58 Printer Command: Unknown code (2)
  13 7.065846  10.0.1.9      224.0.0.1        BJNP         58 Printer Command: Unknown code (2)
```

```
▷ Frame 10: 600 bytes on wire (4800 bits), 600 bytes captured (4800 bits)
▷ Ethernet II, Src: Apple_fe:aa:41 (00:25:00:fe:aa:41), Dst: Apple_41:eb:00 (e4:ce:8f:41:eb:00)
▷ Internet Protocol Version 4, Src: 31.13.75.23 (31.13.75.23), Dst: 10.0.1.13 (10.0.1.13)
▷ Transmission Control Protocol, Src Port: http (80), Dst Port: 61901 (61901), Seq: 1, Ack: 525, Len: 534
▷ Hypertext Transfer Protocol
```

```
0000  e4 ce 8f 41 eb 00 00 25  00 fe aa 41 08 00 45 20   ...A...% ...A..E
0010  02 4a 67 be 00 00 58 06  83 9f 1f 0d 4b 17 0a 00   .Jg...X. ....K...
0020  01 0d 00 50 f1 cd d5 b8  c0 31 96 68 cb 28 80 18   ...P.... .1.h.(..
0030  00 1f f4 2f 00 00 01 01  08 0a 69 40 62 0b 19 92   .../.... ..i@b...
0040  49 70 48 54 54 50 2f 31  2e 31 20 33 30 32 20 46   IpHTTP/1 .1 302 F
```

```
File: "/Users/vern/tmp/all.trace2" 23...   Packets: 13 Displayed: 13 Marked: 0 Load time: 0:00.109   Profile: Default
```

Wireshark: GUI for Packet Capture/Exam.


all.trace2 [Wireshark 1.6.2]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply

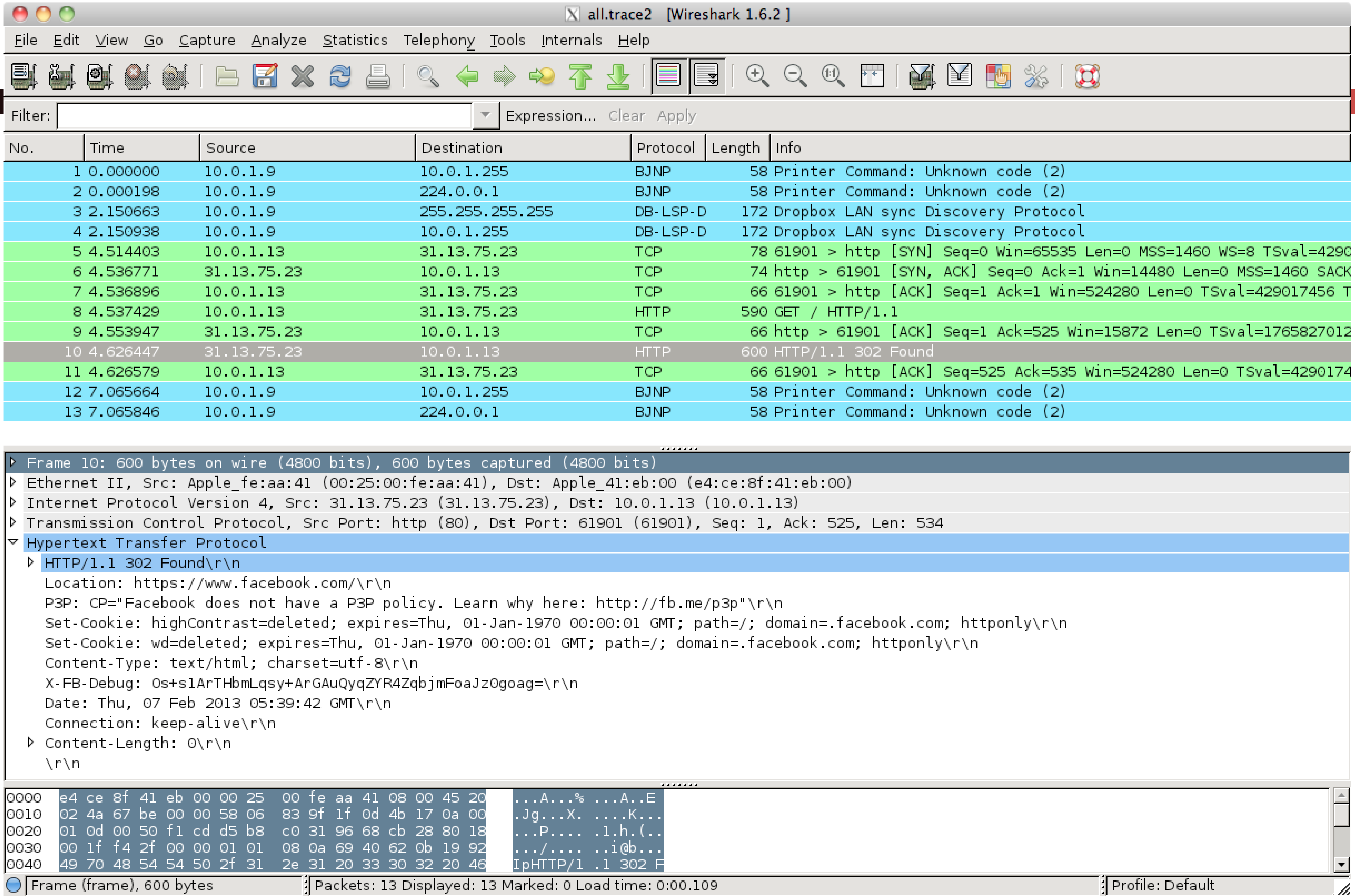
```
No.  Time      Source        Destination      Protocol  Length Info
  1 0.000000   10.0.1.9      10.0.1.255       BJNP          58 Printer Command: Unknown code (2)
  2 0.000198   10.0.1.9      224.0.0.1        BJNP          58 Printer Command: Unknown code (2)
  3 2.150663   10.0.1.9      255.255.255.255  DB-LSP-D     172 Dropbox LAN sync Discovery Protocol
  4 2.150938   10.0.1.9      10.0.1.255       DB-LSP-D     172 Dropbox LAN sync Discovery Protocol
  5 4.514403   10.0.1.13     31.13.75.23      TCP           78 61901 > http [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=8 TSval=4290
  6 4.536771   31.13.75.23   10.0.1.13        TCP           74 http > 61901 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK
  7 4.536896   10.0.1.13     31.13.75.23      TCP           66 61901 > http [ACK] Seq=1 Ack=1 Win=524280 Len=0 TSval=429017456 T
  8 4.537429   10.0.1.13     31.13.75.23      HTTP         590 GET / HTTP/1.1
  9 4.553947   31.13.75.23   10.0.1.13        TCP           66 http > 61901 [ACK] Seq=1 Ack=525 Win=15872 Len=0 TSval=1765827012
 10 4.626447   31.13.75.23   10.0.1.13        HTTP         600 HTTP/1.1 302 Found
 11 4.626579   10.0.1.13     31.13.75.23      TCP           66 61901 > http [ACK] Seq=525 Ack=535 Win=524280 Len=0 TSval=4290174
 12 7.065664   10.0.1.9      10.0.1.255       BJNP          58 Printer Command: Unknown code (2)
 13 7.065846   10.0.1.9      224.0.0.1        BJNP          58 Printer Command: Unknown code (2)
```

```
▷ Frame 10: 600 bytes on wire (4800 bits), 600 bytes captured (4800 bits)
▷ Ethernet II, Src: Apple_fe:aa:41 (00:25:00:fe:aa:41), Dst: Apple_41:eb:00 (e4:ce:8f:41:eb:00)
▷ Internet Protocol Version 4, Src: 31.13.75.23 (31.13.75.23), Dst: 10.0.1.13 (10.0.1.13)
▽ Transmission Control Protocol, Src Port: http (80), Dst Port: 61901 (61901), Seq: 1, Ack: 525, Len: 534
    Source port: http (80)
    Destination port: 61901 (61901)
    [Stream index: 0]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 535    (relative sequence number)]
    Acknowledgement number: 525    (relative ack number)
    Header length: 32 bytes
  ▷ Flags: 0x18 (PSH, ACK)
    Window size value: 31
    [Calculated window size: 15872]
    [Window size scaling factor: 512]
  ▷ Checksum: 0xf42f [validation disabled]
```

```
0000  e4 ce 8f 41 eb 00 00 25  00 fe aa 41 08 00 45 20   ...A...% ...A..E
0010  02 4a 67 be 00 00 58 06  83 9f 1f 0d 4b 17 0a 00   .Jg...X. ....K...
0020  01 0d 00 50 f1 cd d5 b8  c0 31 96 68 cb 28 80 18   ...P.... .1.h.(..
0030  00 1f f4 2f 00 00 01 01  08 0a 69 40 62 0b 19 92   .../.... ..i@b...
0040  49 70 48 54 54 50 2f 31  2e 31 20 33 30 32 20 46   IpHTTP/1 .1 302 F
```

Frame (frame), 600 bytes | Packets: 13 Displayed: 13 Marked: 0 Load time: 0:00.109 | Profile: Default

Wireshark: GUI for Packet Capture/Exam.


all.trace2 [Wireshark 1.6.2]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply

```
No.   Time       Source         Destination      Protocol  Length Info
    1 0.000000   10.0.1.9       10.0.1.255       BJNP         58 Printer Command: Unknown code (2)
    2 0.000198   10.0.1.9       224.0.0.1        BJNP         58 Printer Command: Unknown code (2)
    3 2.150663   10.0.1.9       255.255.255.255  DB-LSP-D    172 Dropbox LAN sync Discovery Protocol
    4 2.150938   10.0.1.9       10.0.1.255       DB-LSP-D    172 Dropbox LAN sync Discovery Protocol
    5 4.514403   10.0.1.13      31.13.75.23      TCP          78 61901 > http [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=8 TSval=4290
    6 4.536771   31.13.75.23    10.0.1.13        TCP          74 http > 61901 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK
    7 4.536896   10.0.1.13      31.13.75.23      TCP          66 61901 > http [ACK] Seq=1 Ack=1 Win=524280 Len=0 TSval=429017456 T
    8 4.537429   10.0.1.13      31.13.75.23      HTTP        590 GET / HTTP/1.1
    9 4.553947   31.13.75.23    10.0.1.13        TCP          66 http > 61901 [ACK] Seq=1 Ack=525 Win=15872 Len=0 TSval=1765827012
   10 4.626447   31.13.75.23    10.0.1.13        HTTP        600 HTTP/1.1 302 Found
   11 4.626579   10.0.1.13      31.13.75.23      TCP          66 61901 > http [ACK] Seq=525 Ack=535 Win=524280 Len=0 TSval=4290174
   12 7.065664   10.0.1.9       10.0.1.255       BJNP         58 Printer Command: Unknown code (2)
   13 7.065846   10.0.1.9       224.0.0.1        BJNP         58 Printer Command: Unknown code (2)
```

```
▷ Frame 10: 600 bytes on wire (4800 bits), 600 bytes captured (4800 bits)
▷ Ethernet II, Src: Apple_fe:aa:41 (00:25:00:fe:aa:41), Dst: Apple_41:eb:00 (e4:ce:8f:41:eb:00)
▷ Internet Protocol Version 4, Src: 31.13.75.23 (31.13.75.23), Dst: 10.0.1.13 (10.0.1.13)
▷ Transmission Control Protocol, Src Port: http (80), Dst Port: 61901 (61901), Seq: 1, Ack: 525, Len: 534
▽ Hypertext Transfer Protocol
  ▷ HTTP/1.1 302 Found\r\n
    Location: https://www.facebook.com/\r\n
    P3P: CP="Facebook does not have a P3P policy. Learn why here: http://fb.me/p3p"\r\n
    Set-Cookie: highContrast=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT; path=/; domain=.facebook.com; httponly\r\n
    Set-Cookie: wd=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT; path=/; domain=.facebook.com; httponly\r\n
    Content-Type: text/html; charset=utf-8\r\n
    X-FB-Debug: Os+s1ArTHbmLqsy+ArGAuQyqZYR4ZqbjmFoaJz0goag=\r\n
    Date: Thu, 07 Feb 2013 05:39:42 GMT\r\n
    Connection: keep-alive\r\n
  ▷ Content-Length: 0\r\n
    \r\n
```

```
0000  e4 ce 8f 41 eb 00 00 25  00 fe aa 41 08 00 45 20   ...A...% ...A..E
0010  02 4a 67 be 00 00 58 06  83 9f 1f 0d 4b 17 0a 00   .Jg...X. ....K...
0020  01 0d 00 50 f1 cd d5 b8  c0 31 96 68 cb 28 80 18   ...P.... .1.h.(..
0030  00 1f f4 2f 00 00 01 01  08 0a 69 40 62 0b 19 92   .../.... ..i@b...
0040  49 70 48 54 54 50 2f 31  2e 31 20 33 30 32 20 46   IpHTTP/1 .1 302 F
```

```
Frame (frame), 600 bytes | Packets: 13 Displayed: 13 Marked: 0 Load time: 0:00.109 | Profile: Default
```

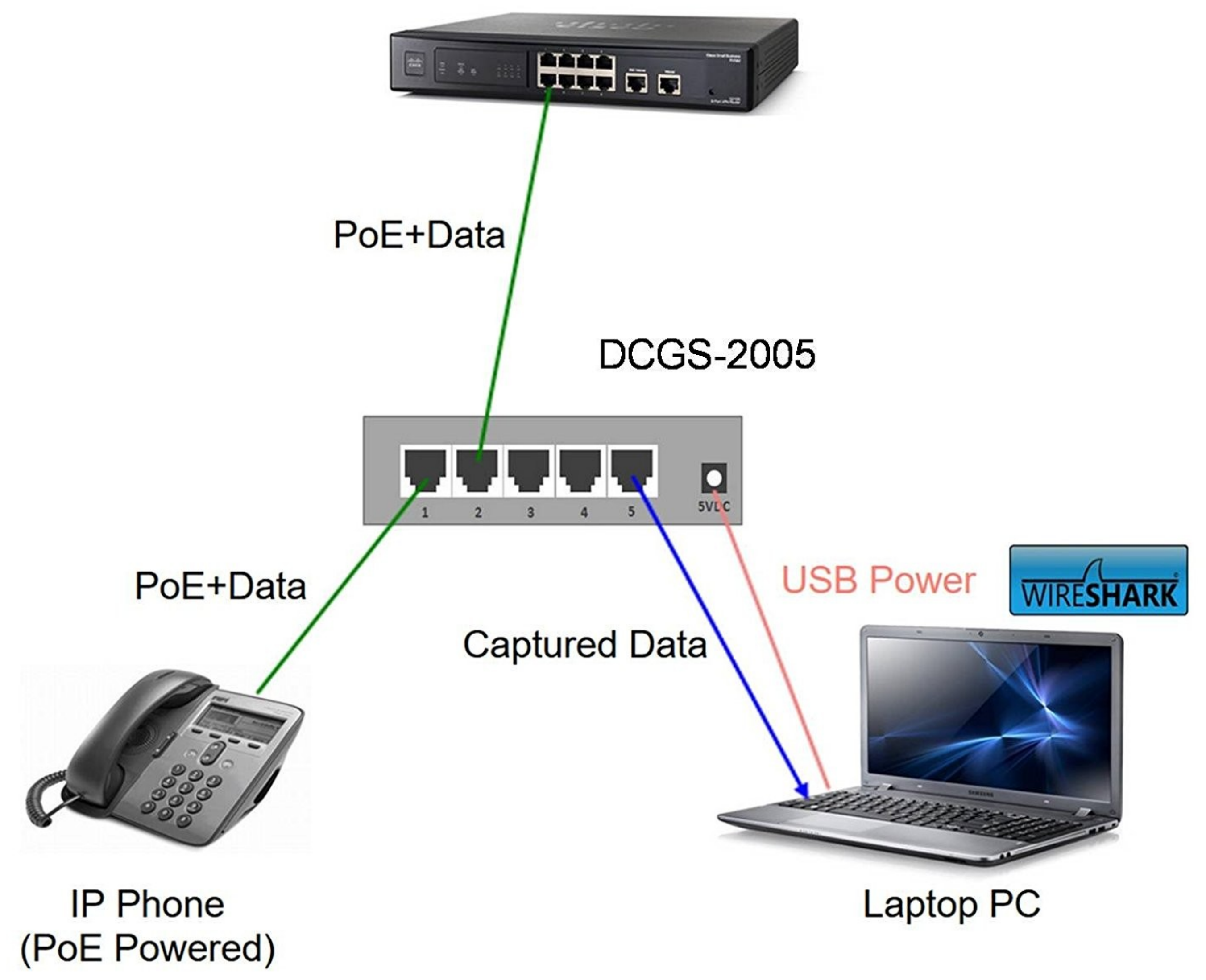
Physical/Link-Layer Threats: Eavesdropping

- Also termed ***sniffing***
- For subnets using broadcast technologies (e.g., WiFi, some types of Ethernet), get it for “free”
  - Each attached system’s NIC (= Network Interface Card) can capture any communication on the subnet
  - Some handy tools for doing so
    - **`tcpdump`** (low-level ASCII printout)
    - Wireshark (higher-level printing)
    - Zeek (previously called Bro): scriptable real-time network analysis; see zeek.org
- You can also "tap" (mirror) a link or configure a "mirror port"

One Cool Toy: DualComm DCGS-2005

- A $200, 5-port Ethernet switch...
  - With some bonus features
- Built in port "mirror"
  - All traffic to and from port 1 is copied to port 5
- Powered through a USB connection
  - So no need for an extra power supply
- Power-Over-Ethernet passthrough
  - Port 2 can send power to port 1 so you can tap IP phones...

Setup Diagram of DCGS-2005



Operation Ivy Bells

By Matthew Carle
Military.com

At the beginning of the 1970's, divers from the specially-equipped submarine, USS Halibut (SSN 587), left their decompression chamber to start a bold and dangerous mission, code named "Ivy Bells".

The Regulus guided missile submarine, USS Halibut (SSN 587) which carried out Operation Ivy Bells.

ALASKA
Arctic Ocean
RUSSIA
Bering Sea
Sea of Okhotsk
Kamchatka Peninsula
CHINA
Sakhalin
Kuril Islands
NORTH KOREA
SOUTH KOREA
Sea of Japan
Pacific Ocean
NORTH
0 200 400 600 800 1,000
Kilometres

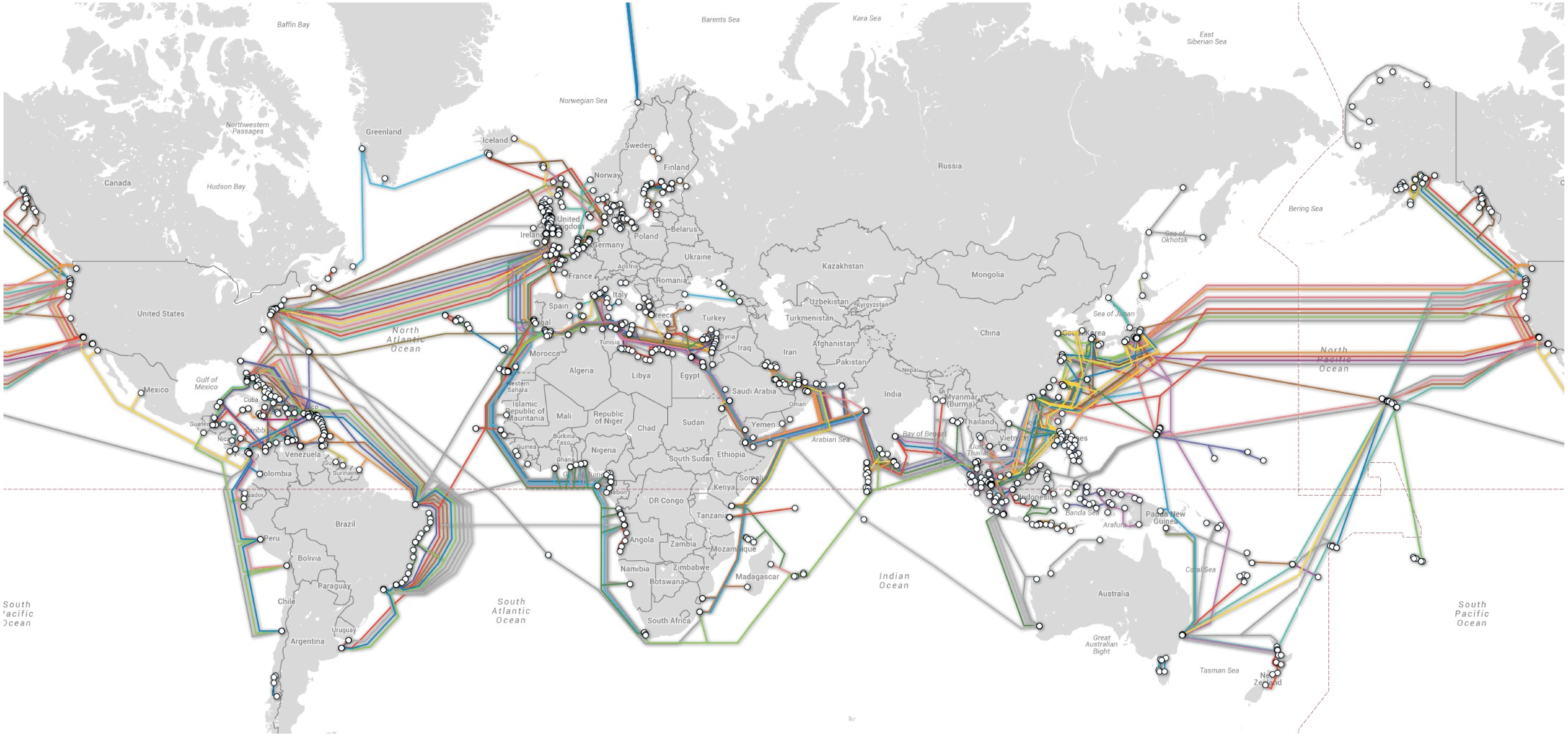
In an effort to alter the balance of Cold War, these men scoured the ocean floor for a five-inch diameter cable carry secret Soviet communications between military bases.

The divers found the cable and installed a 20-foot long listening device on the cable. designed to attach to the cable without piercing the casing, the device recorded all communications that occurred. If the cable malfunctioned and the Soviets raised it for repair, the bug, by design, would fall to the bottom of the ocean. Each month Navy divers retrieved the recordings and installed a new set of tapes.

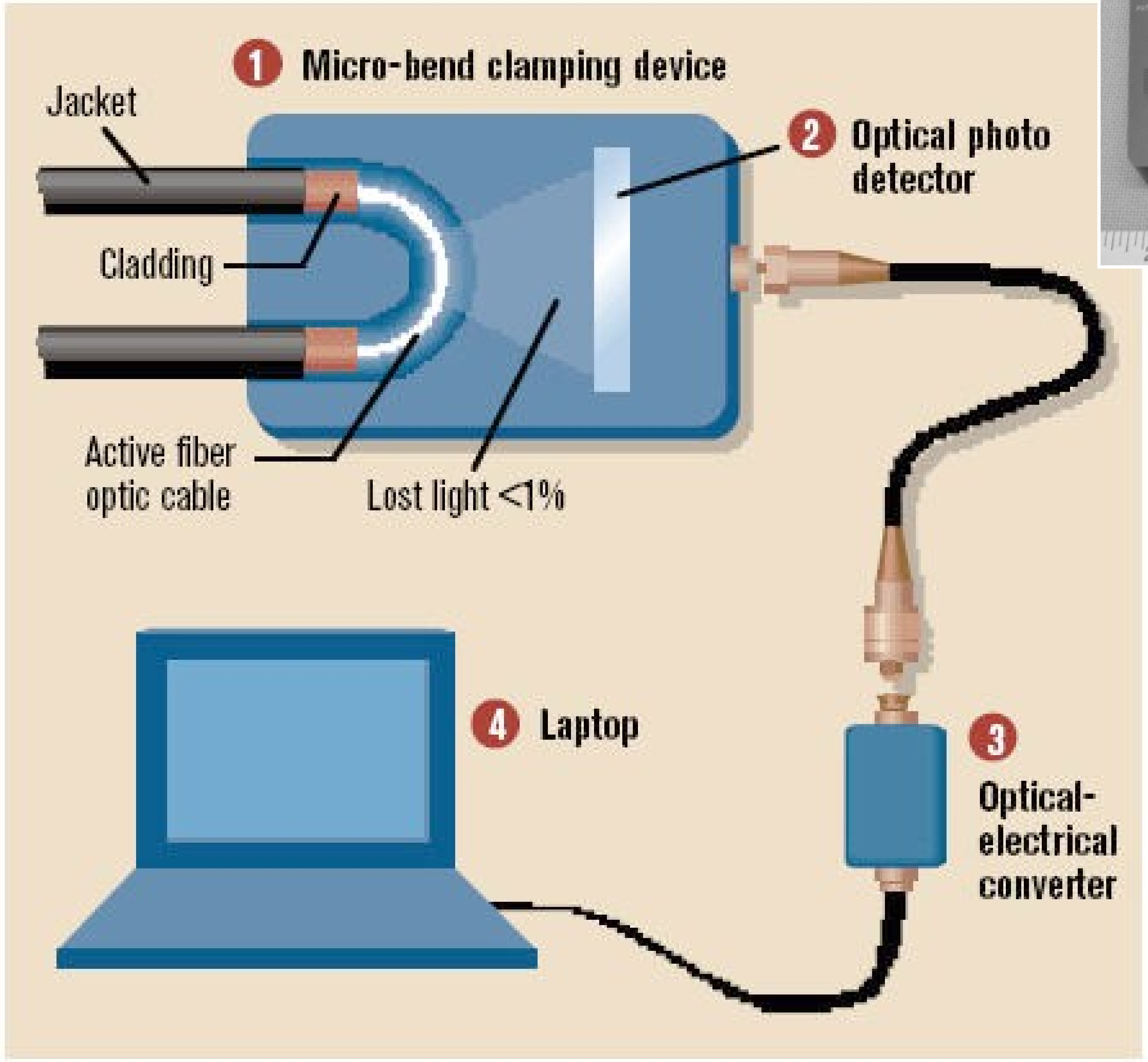
Upon their return to the United States, intelligence agents from the NSA analyzed the recordings and tried to decipher any encrypted information. The Soviets apparently were confident in the security of their communications lines, as a surprising amount of sensitive information traveled through the lines without encryption.

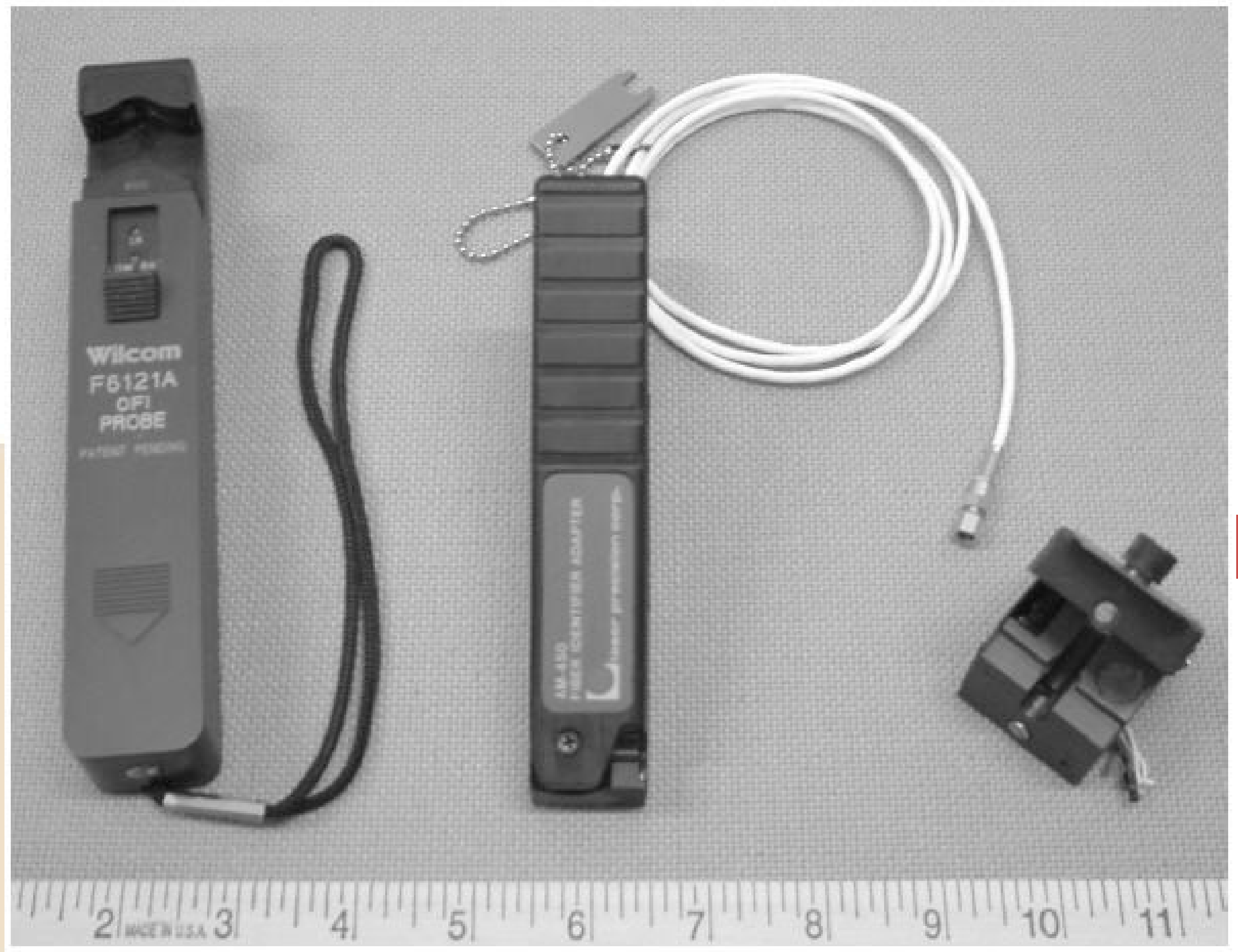
prison. The original tap that was discovered by the Soviets is now on exhibit at the KGB museum in Moscow.

Tapping the Whole Planet



Stealing Photons





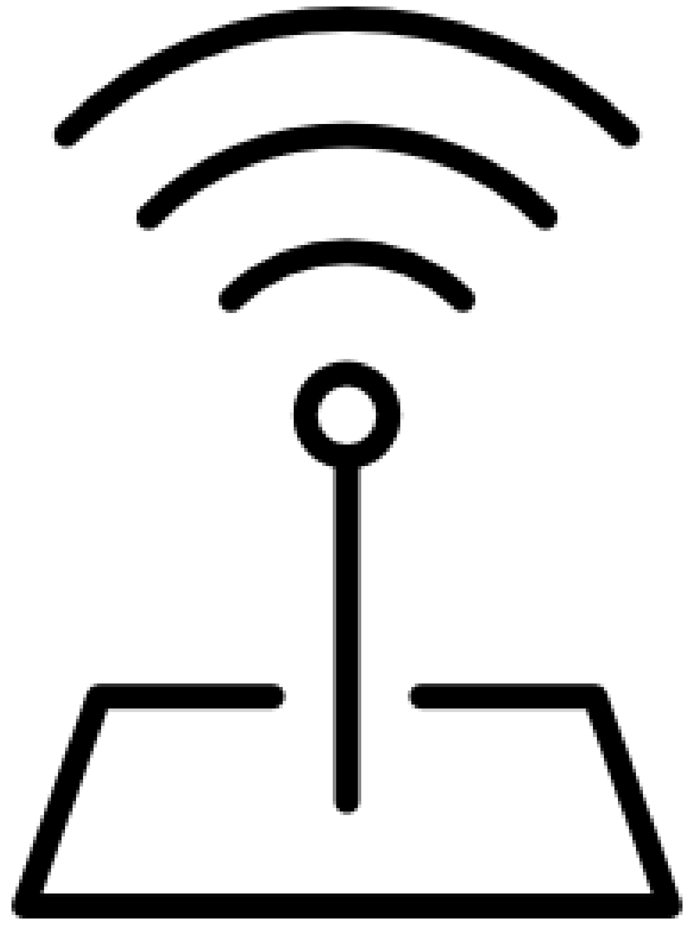
The Rogue AP...

- Your phone/computer keeps broadcasting "Is network X available"?
  - If there is no password, why not just say "Yeah, I'm here!!!"
- Your phone happily connects...
  - To the ***attacker's internet connection***
- The attacker as a man-in-the-middle...
  - Can now extract pretty much all non-encrypted communication data...
  - "Hey web-browser, spit up effectively all cookies that are sent on non-TLS connections..."

Wireless Ethernet Security Option: WPA2 Pre Shared Key

- This is what is used these days when the WiFi is “password protected”
  - The access point and the client have the same pre-shared key (called the PSK key)
  - Goal is to create a shared key called the PTK (Pairwise Transient Key)
- This key is derived from a combination of both the password and the SSID (network name)
  - PSK = PBKDF2(SHA1, passphrase, ssid, 4096, 256)
- Use of PBKDF
  - The SSID as salt ensures that the same password on different network names is different
  - The iteration count assures that it is ***slow***
    - Any attempt to brute force the passphrase should take a lot of time per guess

The WPA 4-way Handshake



SNonce + MIC

Ack

ANonce

GTK + MIC

Computed PTK = F(PSK, ANonce SNonce, AP MAC, Client MAC)

Computed PTK = F(PSK, ANonce SNonce, AP MAC, Client MAC)

Remarks

- This is ***only*** secure if an eavesdropper doesn’t know the pre shared key
  - Otherwise an eavesdropper who sees the handshake can perform the same computations to get the transport key
    - However, by default, network cards don't do this:
      This is a "do not disturb sign" security. It will keep the maid from entering your hotel room but won't stop a burglar
- Oh, and given ANonce, SNonce, MIC(SNonce), can attempt an offline ***brute-force attack***
  - And since people don't chose good passwords, it will almost certainly succeed:
    People have built single systems that can try ~8M passwords/second!
  - And can execute a "***deauthentication attack***" to cause the client to disconnect and then reconnect:
    Running another handshake
- The MIC is really a MAC, but as MAC also refers to the MAC address, they use MIC in the description
- The GTK is for broadcast
  - So the AP doesn’t have to rebroadcast things, but usually does anyway

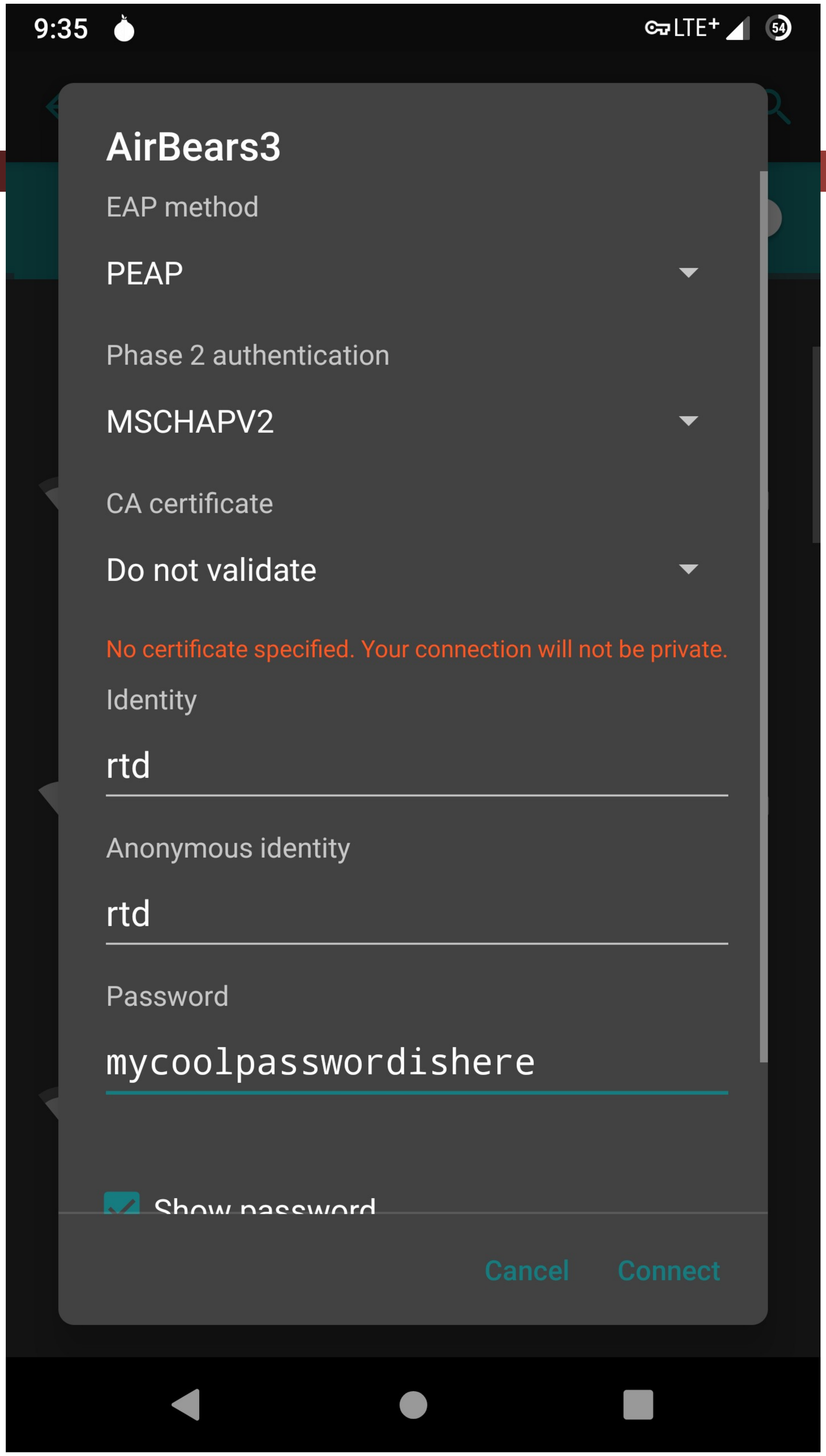
Rogue APs and WPA2-PSK...

- You can ***still do a rogue AP***!
  - Just answer with a random ANonce...
  - That gets you back the SNonce and MIC(SNonce)
    - Which uses as a key for the MIC = F(PSK, ANonce, SNonce, AP MAC, Client MAC)
- So just do a brute-force dictionary attack on PSK
  - Since PSK = PBKDF2(SHA1, pw, ssid, 4096, 256)
    - So 8192 SHA-1 invocations... Yawn.
  - Verify the MIC to validate whether the guess was correct
- Because lets face it, people don't chose very good passwords...
  - Anyone want to build a full hardware stack version to do this for next DEFCON?
    - Using a Xilinx PYNQ board? Dual core ARM Linux w a 13k logic cell FPGA

Actually Making it Secure: WPA Enterprise

- When you set up Airbears 2, it asks you to accept a public key certificate
  - This is the public key of the ***authentication*** server, not the access point
- Now before the 4-way handshake:
  - Your computer first handshakes with the authentication server
    - This is secure using public key cryptography
  - Your computer then authenticates to this server
    - With your username and password
- The server now generates a unique key that it both tells your computer and tells the base station
  - So the 4 way handshake is now secure since its a unique PSK

Actually Making it Secure?? WPA Enterprise


9:35

AirBears3

EAP method
PEAP

Phase 2 authentication
MSCHAPV2

CA certificate
Do not validate

No certificate specified. Your connection will not be private.

Identity
rtd

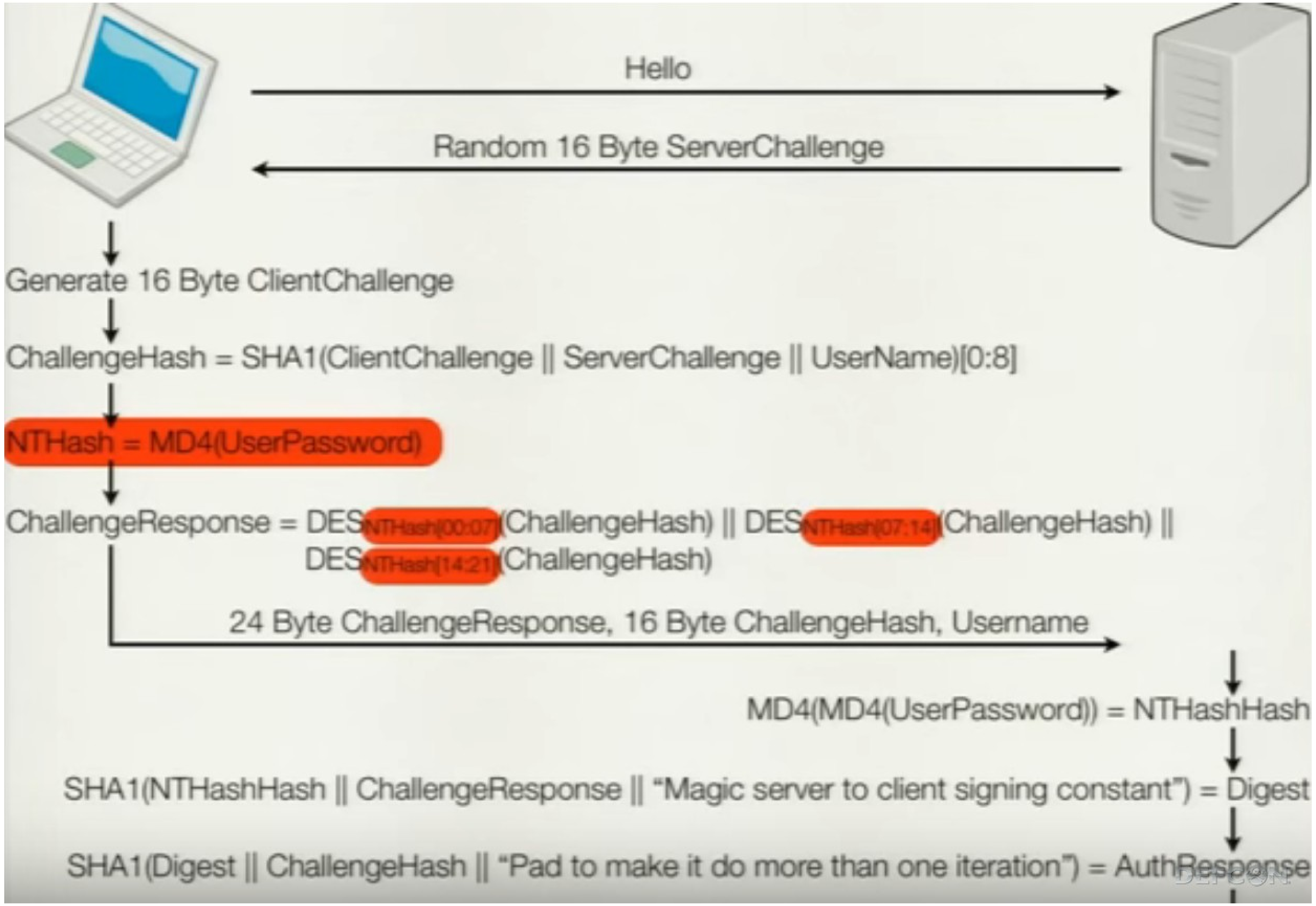
Anonymous identity
rtd

Password
mycoolpasswordishere

Show password

MS-CHAPv2

https://www.youtube.com/watch?v=gkPvZDcrLFk


Hello
Random 16 Byte ServerChallenge
Generate 16 Byte ClientChallenge
ChallengeHash = SHA1(ClientChallenge || ServerChallenge || UserName)[0:8]
NTHash = MD4(UserPassword)
ChallengeResponse = DES NTHash[00:07](ChallengeHash) || DES NTHash[07:14](ChallengeHash) || DES NTHash[14:21](ChallengeHash)
24 Byte ChallengeResponse, 16 Byte ChallengeHash, Username
MD4(MD4(UserPassword)) = NTHashHash
SHA1(NTHashHash || ChallengeResponse || "Magic server to client signing constant") = Digest
SHA1(Digest || ChallengeHash || "Pad to make it do more than one iteration") = AuthResponse
DEFCON

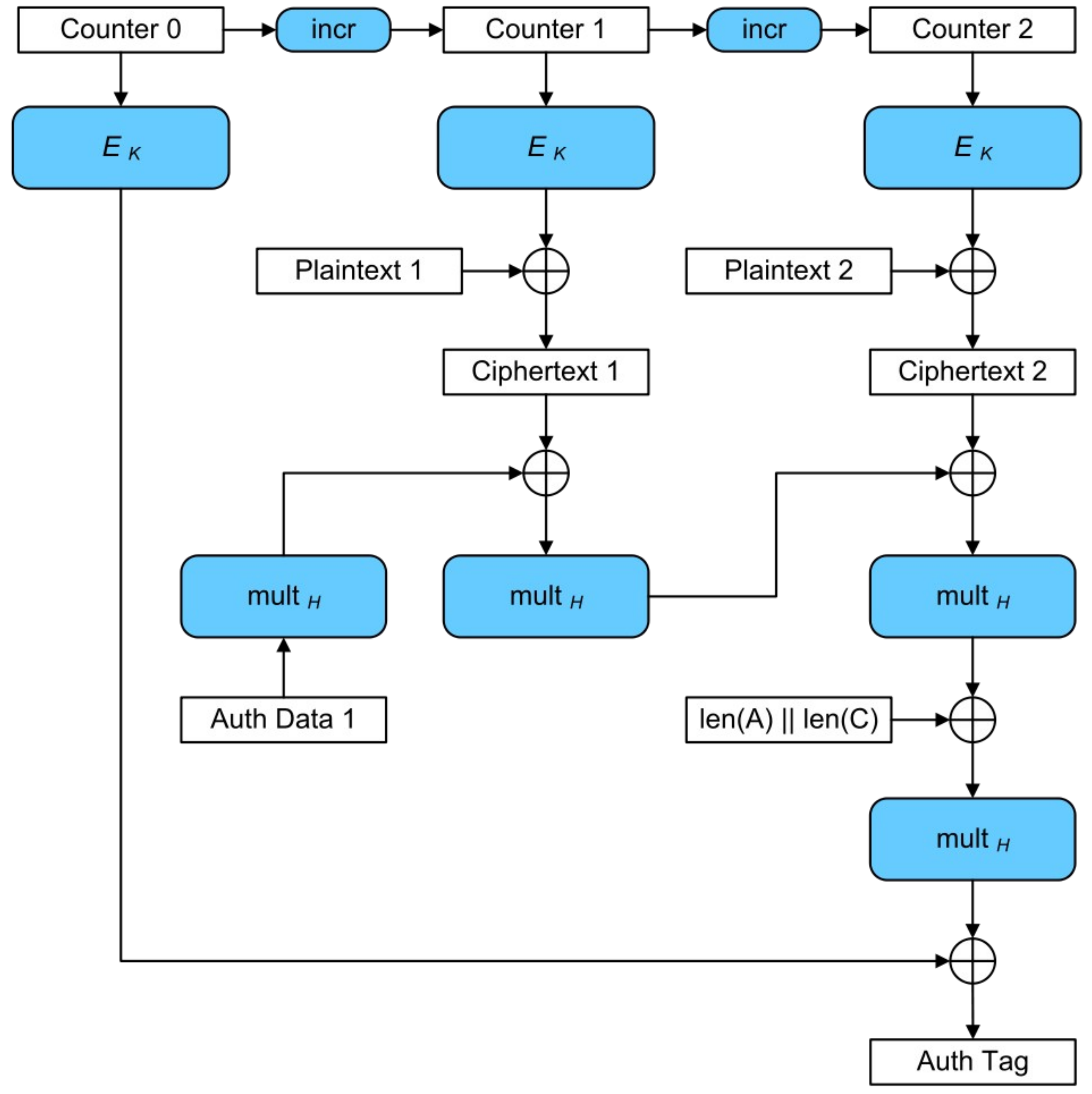
crack.sh

The Latest Hotness: KRACK attack...

- To actually encrypt the individual packets: IV of a packet is {Agreed IV || packet counter}
  - Thus for each packet you only need to send the packet counter (48 bits) rather than the full IV (128b)
- Multiple different modes
  - One common one is CCM (Counter with CBC-MAC)
    - MAC the data with CBC-MAC
      Then encrypt with CTR mode
  - The highest performance is GCM (Galois/Counter Mode)
- But if you thought CTR mode was bad on IV reuse...
  - GCM is worse: A couple of reused IVs can reveal enough information to forge the authentication!
- Discovered a year ago, fairly quickly patch, but...

GCM...

- GCM is like CTR mode with a twist...
  - The confidentiality is pure CTR mode
  - The "Galois" part is a hash of the cipher text
    - The only secret part being the "Auth Data"
- Reuse the IV, what happens?
  - Not ***only*** do you have CTR mode loss of confidentiality...
  - But if you do it enough, you lose confidentiality on the Auth Data...
  - So you lose the integrity that GCM supposedly provided!



And Packets Get "Lost"

- Even a wired network will "drop packets"
  - A message is sent but simply never delivered
- Its far worse on wireless
  - A gazillion things can go wrong, including other transmitters
    - And noise like a microwave oven!
- So you have to design for packets to be rebroadcast...
- In the WPA handshake, what do you do when you receive the 3rd packet?
  - Initialize the key you use for encrypting the packets
  - Set the packet counter to 0

And A Replay Attack...

- What if the attacker listens for the third step in the handshake...
  - And then repeats it?
- Why, the client is supposed to reinitialize the key and agreed IV...
  - Which on many implementations, ***also resets the packet counter***...
  - Oh, and Linux (and Android 6) is worse... It reinitializes the key ***to zero!***
- So what does that mean?

Attack Scenario...

- Attacker is close to target
- Attacker captures the 3rd step in the handshake
- Attacker repeatedly replays this to the client
- Client now repeats IVs for encryption...
- Other modes. Annoyance: the damage is minor
- CCM-mode: Attacker can now decrypt in practice thanks to IV reuse
- GCM-mode...
    - Attacker can now decrypt ***and forge packets***:
      Reusing the IV also reveals the MAC-secret!

Mitigations...

- Like all attacks on WiFi, it requires a "close" attacker...
  - 100m to a km or two...
- If you use WPA2-PSK, aka a "WiFi Password", who cares?
  - Unless your WiFi password sounds like a cat hawking up a hairball, you don't have enough entropy to resist a brute-force attacks
- If you use WPA2-Enterprise, this ***may*** matter...
  - But lets face it, there are so many more critical things to patch first...
  - And why are you treating the WiFi as trusted anyway?

But Broadcast Protocols Make It Worse...

- By default, both DHCP and ARP broadcast requests
  - Sent to ***all*** systems on the local area network
- DHCP: Dynamic Host Control Protocol
  - Used to configure all the important network information
    - Including the DNS server:
      If the attacker controls the DNS server they have complete ability to intercept all traffic!
    - Including the Gateway which is where on the LAN a computer sends to:
      If the attacker controls the gateway
- ARP: Address Resolution Protocol
  - "Hey world, what is the Ethernet MAC address of IP X"
  - Used to find both the Gateway's MAC address and other systems on the LAN

Broadcast Protocols And The LAN

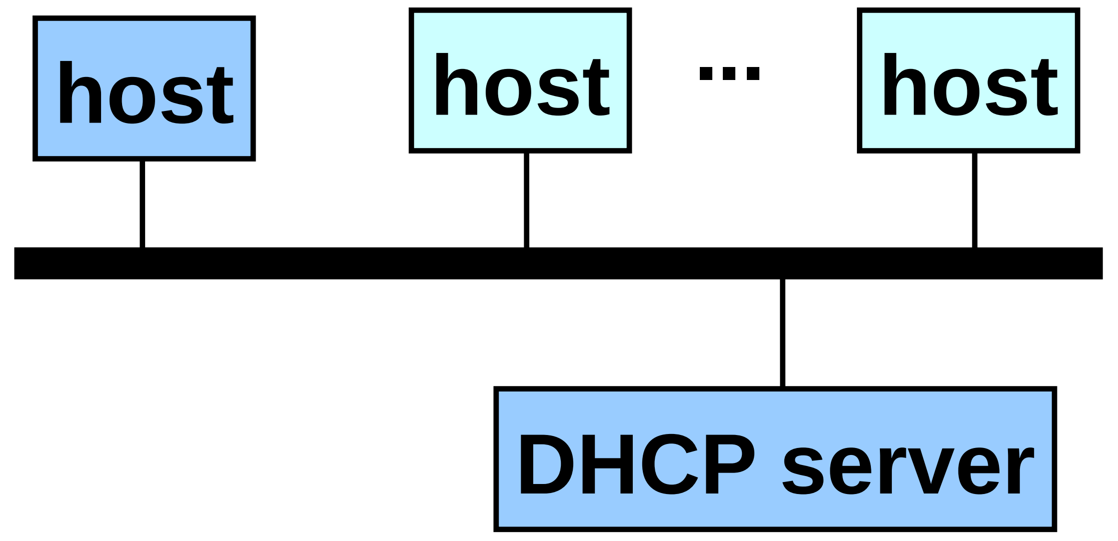
- A rogue device on the LAN can respond to these
  - As long as it arrives first, the attacker wins
- DHCP: Give "bad" gateway...
  - Can directly intercept all traffic to the Internet
- DHCP: Give "proper" gateway but a bad DNS server...
  - Now can intercept all desired traffic by just giving bad DNS response
- ARP: Give "bad" answer for ARP requests to gateway...
  - Can directly intercept all traffic to the Internet

2. Configure your connection

Your laptop shouts:
HEY, ANYBODY, WHAT BASIC CONFIG DO I NEED TO USE?

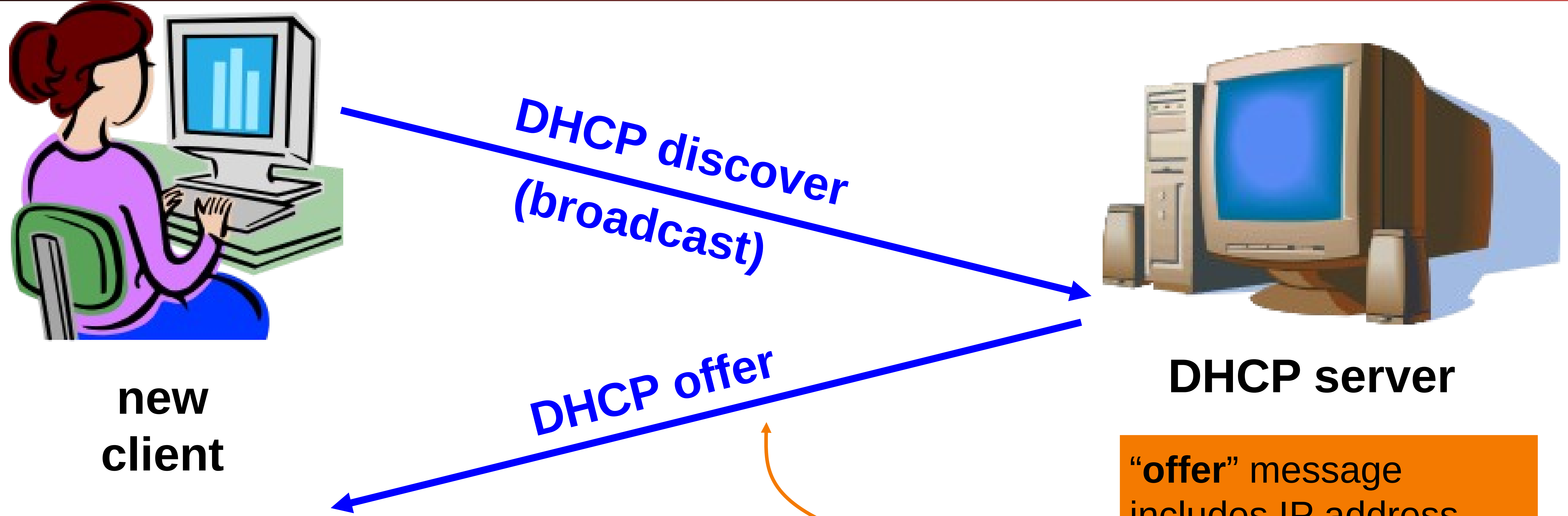
Internet Bootstrapping: DHCP

- New host doesn't have an IP address yet
  - So, host doesn't know what source address to use
- Host doesn't know *who to ask* for an IP address
  - So, host doesn't know what destination address to use
- (Note, host does have a separate WiFi address)
- Solution: *shout* to "**discover**" server that can help
  - Broadcast a server-discovery message (layer 2)
  - Server(s) sends a reply offering an address



DHCP = Dynamic Host Configuration Protocol

Dynamic Host Configuration Protocol

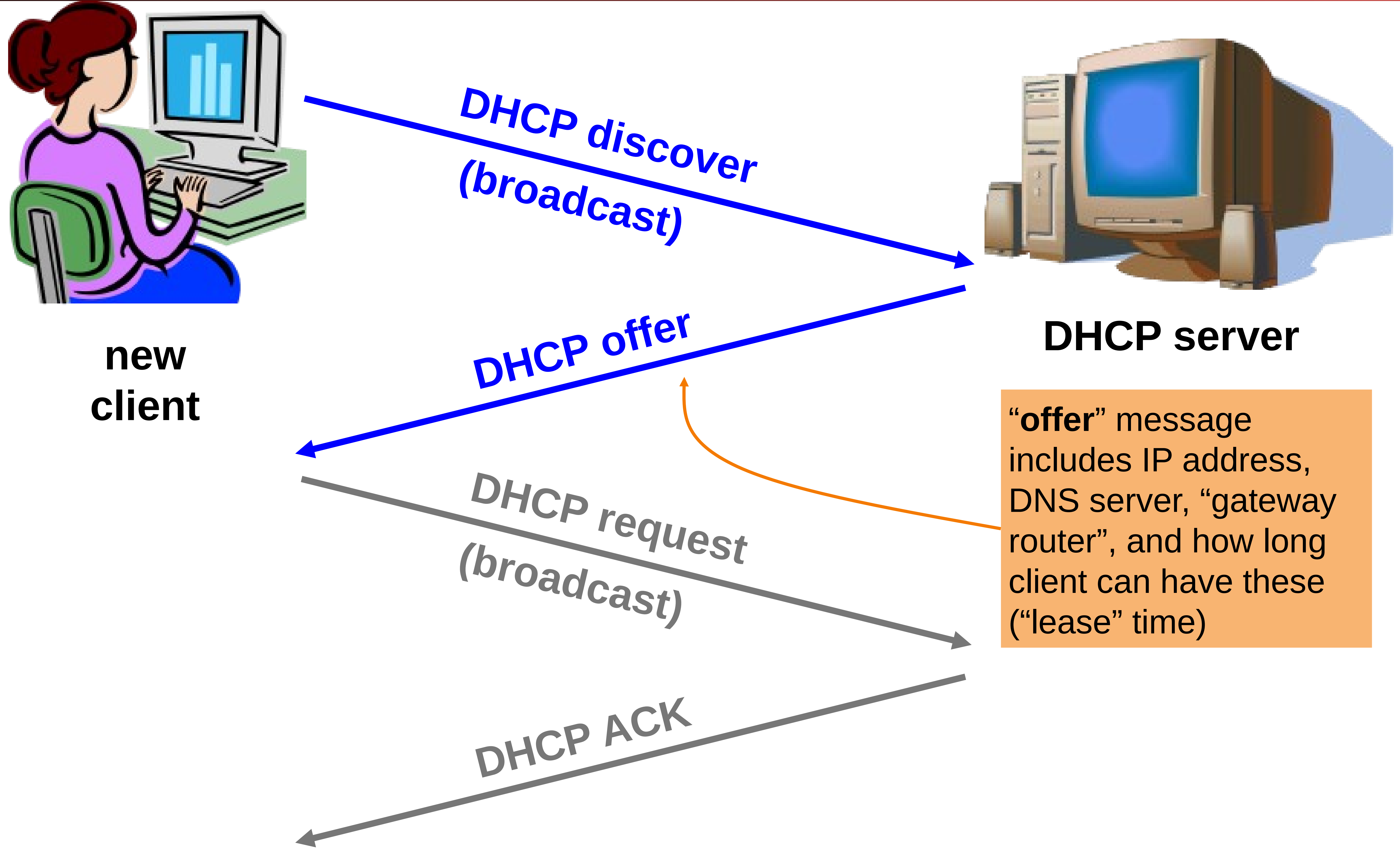


“**offer**” message includes IP address, DNS server, “gateway router”, and how long client can have these (“lease” time)

DNS server = system used by client to map hostnames like `gmail.com` to IP addresses like `74.125.224.149`

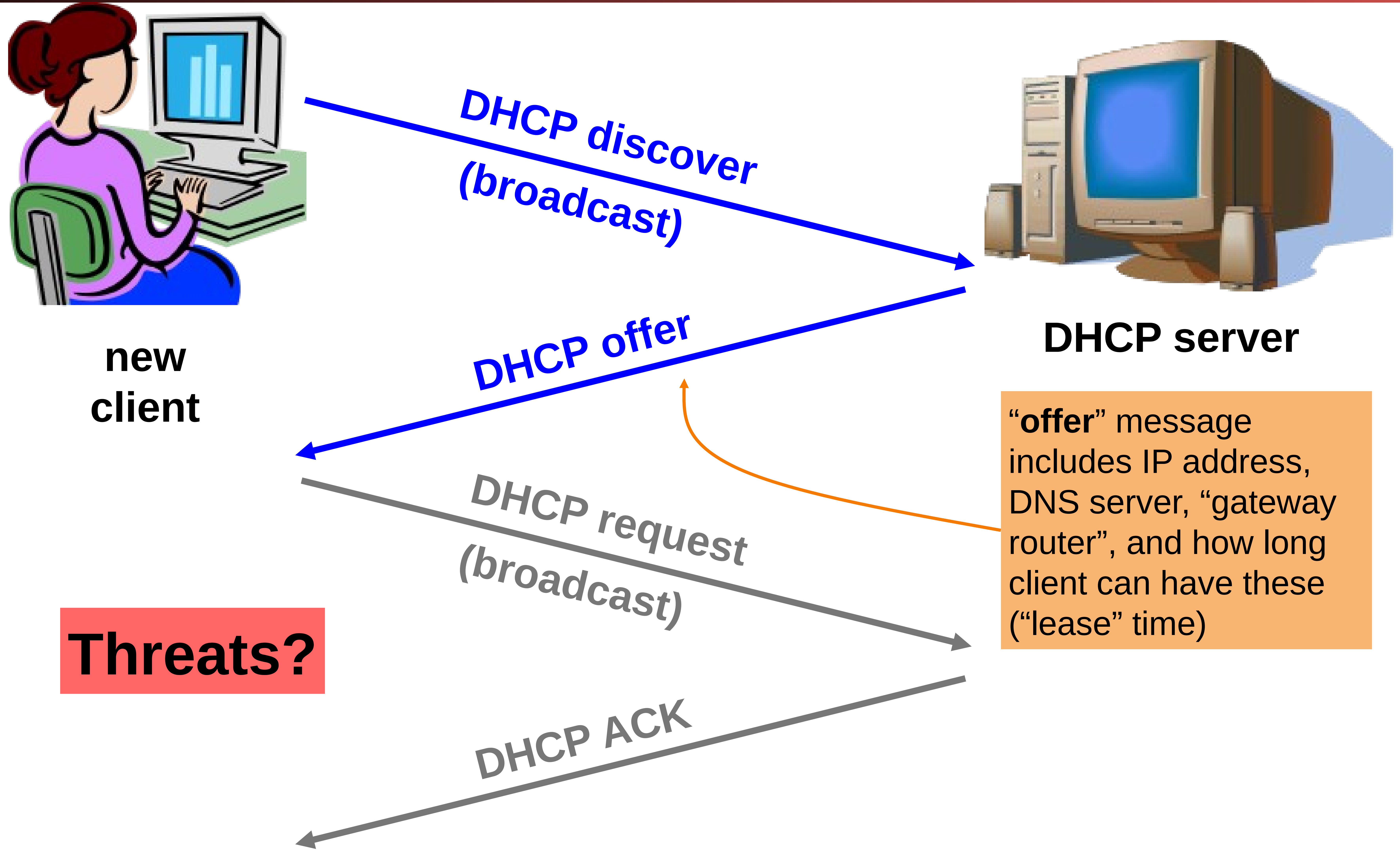
Gateway router = router that client uses as the first hop for all of its Internet traffic to remote hosts

Dynamic Host Configuration Protocol



"**offer**" message includes IP address, DNS server, "gateway router", and how long client can have these ("lease" time)

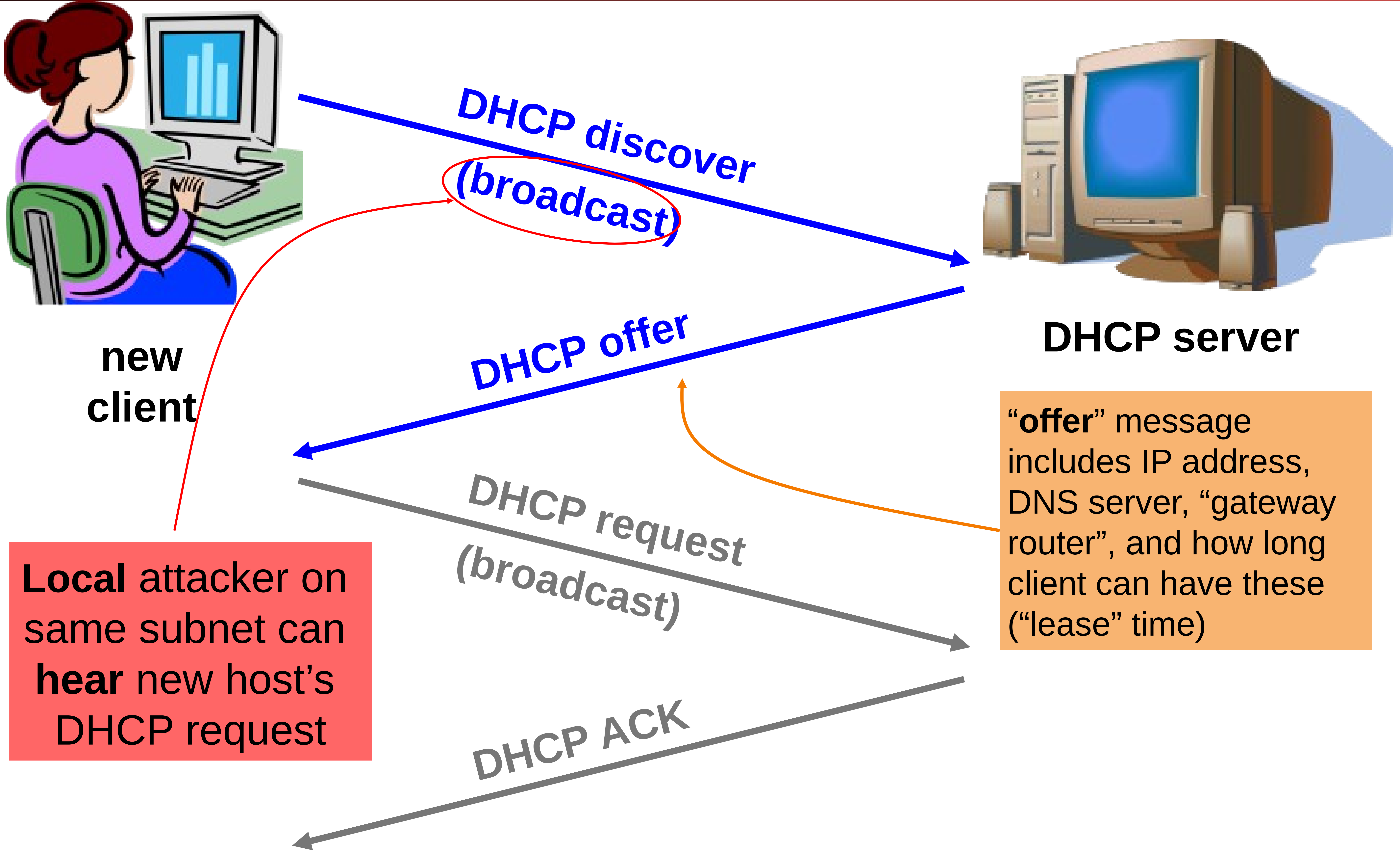
Dynamic Host Configuration Protocol



"**offer**" message includes IP address, DNS server, "gateway router", and how long client can have these ("lease" time)

Threats?

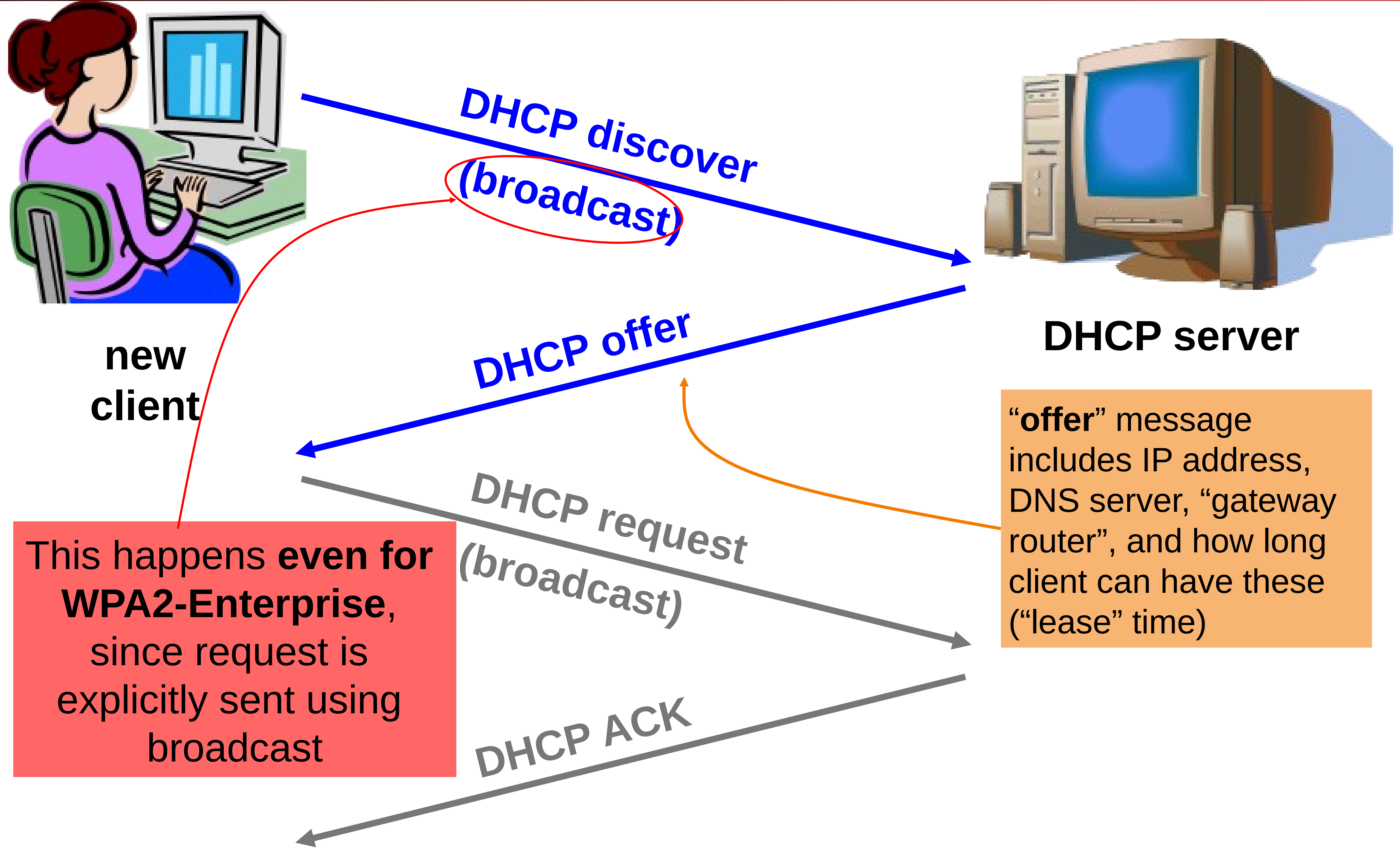
Dynamic Host Configuration Protocol



“**offer**” message includes IP address, DNS server, “gateway router”, and how long client can have these (“lease” time)

Local attacker on same subnet can **hear** new host’s DHCP request

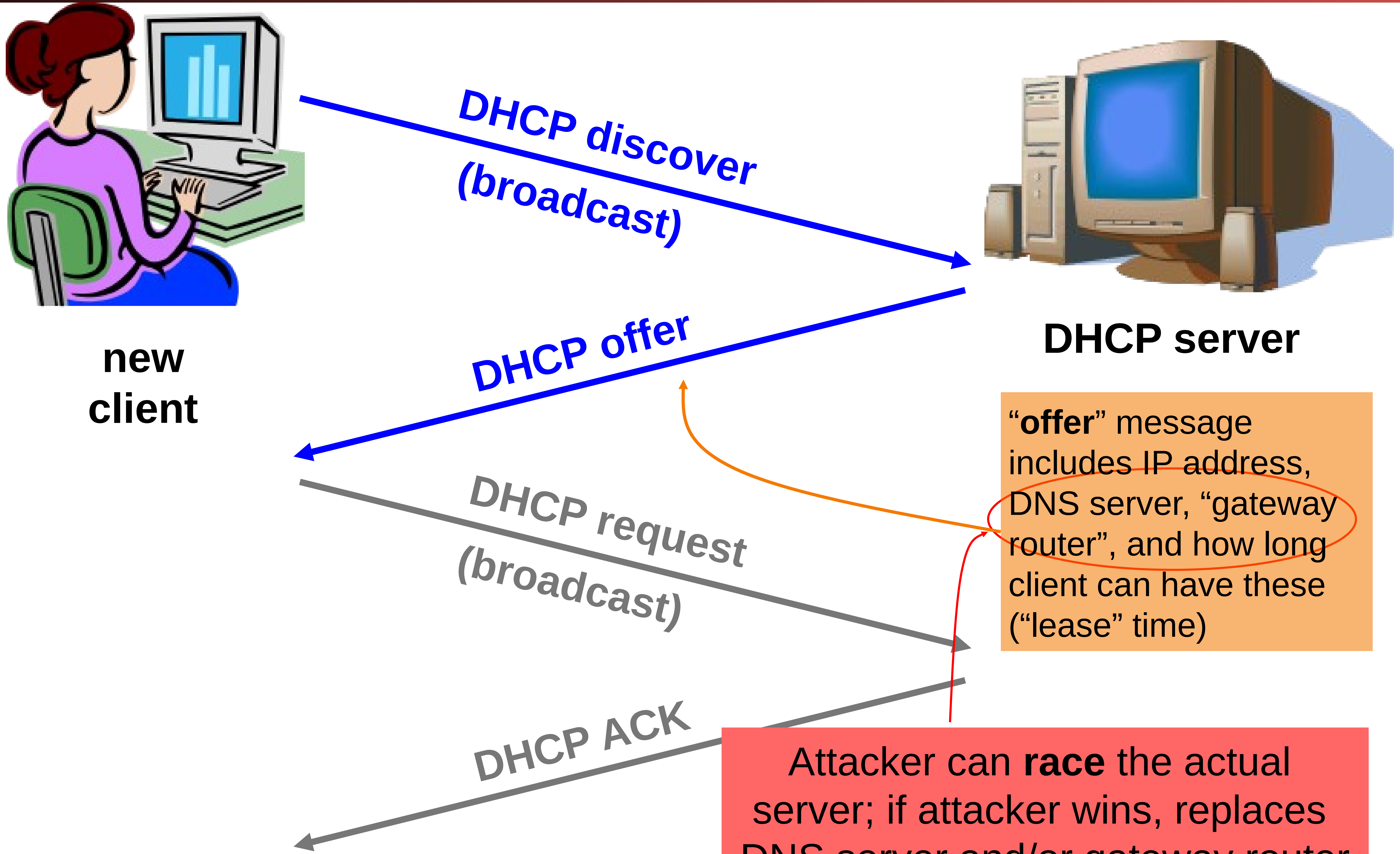
Dynamic Host Configuration Protocol



"**offer**" message includes IP address, DNS server, "gateway router", and how long client can have these ("lease" time)

This happens **even for WPA2-Enterprise**, since request is explicitly sent using broadcast

Dynamic Host Configuration Protocol



"**offer**" message includes IP address, DNS server, "gateway router", and how long client can have these ("lease" time)

Attacker can **race** the actual server; if attacker wins, replaces DNS server and/or gateway router

DHCP Threats

- Substitute a fake DNS server
  - Redirect any of a host’s lookups to a machine of attacker’s choice (e.g., **`gmail.com`** = **`6.6.6.6`**)
- Substitute a fake gateway router
  - Intercept all of a host’s off-subnet traffic
  - Relay contents back and forth between host and remote server
    - Modify however attacker chooses
  - This is one type of invisible Man In The Middle (MITM)
    - Victim host generally has no way of knowing it’s happening! 😟
    - (Can’t necessarily alarm on peculiarity of receiving multiple DHCP replies, since that can happen benignly)
- How can we fix this?

Hard, because we lack a ***trust anchor***

DHCP Conclusion

- DHCP threats highlight:
  - Broadcast protocols inherently at risk of local attacker spoofing
    - Attacker knows exactly when to try it …
    - … and can see the victim’s messages
  - When initializing, systems are particularly vulnerable because they can lack a trusted foundation to build upon
  - Tension between ***wiring in trust*** vs. ***flexibility and convenience***
  - MITM attacks insidious because no indicators they’re occurring

So How Do We Secure the LAN?

- Option 1: We don't
  - Just assume we can keep bad people out
  - This is how most people run their networks:
    "Hard on the outside with a goey chewy caramel center"
  - Option 1 variant:
    The LAN is a festering pool just like the rest of the Internet, so treat everything as hostile always all the time!
- Option 2: ***smart*** switching and active monitoring

The Switch

- Hubs are very inefficient:
  - By broadcasting traffic to all recipients this greatly limits the aggregate network bandwidth
- Instead, most Ethernet uses switches
  - The switch keeps track of which MAC address is seen where
- When a packet comes in:
  - If there is no entry in the MAC cache, broadcast it to all ports
  - If there is an entry, send it just to that port
- Result is vastly improved bandwidth
  - All ports can send or receive at the same time

Smarter Switches: Clean Up the Broadcast Domain

- Modern high-end switches can do even more
  - A large amount of potential packet processing on items of interest
- Basic idea: constrain the broadcast domain
  - Either filter requests so they only go to specific ports
    - Limits other systems from listening
  - Or filter replies
    - Limits other systems from replying
- Locking down the LAN is very important practical security
  - This is ***real*** defense in depth:
    Don't want 'root on random box, pwn whole network'
  - This removes "***pivots***" the attacker can try to extend a small foothold into complete network ownership
- This is why an Enterprise switch may cost $1000s yet provide no more real bandwidth than a $100 Linksys.

Smarter Switches: Virtual Local Area Networks (VLANs)

- Our big expensive switch can connect a lot of things together
  - But really, many are in ***different*** trust domains:
    - Guest wireless
    - Employee wireless
    - Production desktops
    - File Servers
    - etc...
- Want to isolate the different networks from each other
  - Without actually buying separate switches

VLANs

- An ethernet port can exist in one of two modes:
  - Either on a single VLAN
  - On a trunk containing multiple specified VLANs
- All network traffic in a given VLAN stays only within that VLAN
  - The switch makes sure that this occurs
- When moving to/from a trunk the VLAN tag is added or removed
  - But still enforces that a given trunk can only read/write to specific VLANs
- VLAN tag is ***automatically*** added internally when appropriate to constrain internal traffic

Putting It Together: If I Was In Charge of UC networking...

- I'd isolate networks into 3+ distinct classes
  - The plague pits (AirBears, Dorms, etc)
  - The mildly infected pits (Research)
  - Administration
- Administration would be locked down
  - Separate VLANs
  - Restricted DHCP/system access
  - Isolated from the rest of campus

Addressing on the Layers On The Internet

- Ethernet:
  - Address is 6B MAC address, Identifies a machine on the local LAN
- IP:
  - Address is a 4B (IPv4) or 16B (IPv6) address, Identifies a system on the Internet
- TCP/UDP:
  - Address is a 2B port number, Identifies a particular listening server/process/activity on the system
    - Both the client and server have to have a port associated with the communication
  - Ports 0-1024 are for privileged services
    - Must be root to accept incoming connections on these ports
    - Any thing can do an outbound request to such a port
  - Port 1025+ are for anybody
    - And high ports are often used ephemerally

UDP:
Datagrams on the Internet

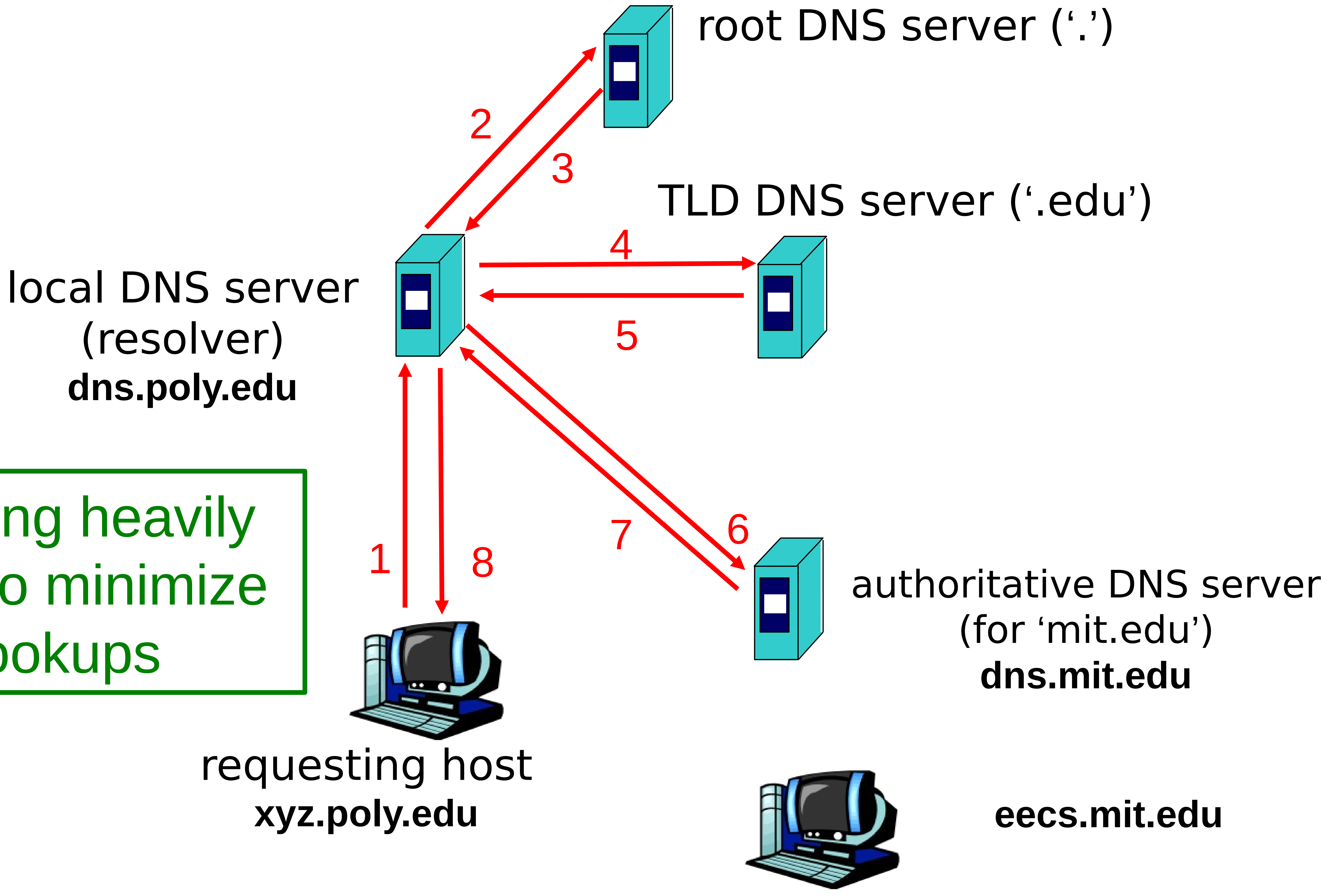
- UDP is a protocol built on the Internet Protocol (IP)
- It is an "unreliable, datagram protocol"
  - Messages may or may not be delivered, in any order
  - Messages can be larger than a single packet
    - IP will fragment these into multiple packets (mostly)
- Programs create a socket to send and receive messages
  - Just create a datagram socket for an ephemeral port
  - Bind the socket to a particular port to receive traffic on a specified port
  - Basic recipe for Python: https://wiki.python.org/moin/UdpCommunication

DNS Overview

- DNS translates www.google.com to 74.125.25.99
  - Turns a human abstraction into an IP address
  - Can also contain other data
- It's a performance-critical distributed database.
- DNS security is critical for the web.
  (Same-origin policy ***assumes*** DNS is secure.)
  - Analogy: If you don't know the answer to a question, ask a friend for help (who may in turn refer you to a friend of theirs, and so on).
- Based on a notion of hierarchical trust:
  - You trust . for everything, com. for any com, google.com. for everything google…

DNS Lookups via a *Resolver*

Host at **`xyz.poly.edu`** wants IP address for **`eecs.mit.edu`**



Caching heavily used to minimize lookups

Security risk #1: malicious DNS server

- Of course, if *any* of the DNS servers queried are malicious, they can lie to us and fool us about the answer to our DNS query
- (In fact, they used to be able to fool us about the answer to other queries, too. We'll come back to that.)

Security risk #2: on-path eavesdropper

- If attacker can eavesdrop on our traffic… we’re hosed.
- Why? We’ll see why.

Security risk #3: off-path attacker

- If attacker can’t eavesdrop on our traffic, can he inject spoofed DNS responses?
- This case is especially interesting, so we’ll look at it in detail.

DNS Threats

- DNS: path-critical for just about everything we do
  - Maps hostnames ⇔ IP addresses
  - Design only **scales** if we can minimize lookup traffic
    - #1 way to do so: caching
    - #2 way to do so: return not only answers to queries, but additional info that will likely be needed shortly
      - The "glue records"
- What if attacker eavesdrops on our DNS queries?
  - Then similar to DHCP, ARP, AirPwn etc, can spoof responses
- Consider attackers who *can't* eavesdrop - but still aim to manipulate us via *how the protocol functions*
- Directly interacting w/ DNS: **`dig`** program on Unix
  - Allows querying of DNS system
  - Dumps each field in DNS responses

```
dig eecs.mit.edu A
```

Use Unix "dig" utility to look up IP address ("A") for hostname eecs.mit.edu via DNS

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.                  IN      A

;; ANSWER SECTION:
eecs.mit.edu.           21600   IN      A       18.62.1.6

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                11088   IN      NS      STRAWB.mit.edu.

;; ADDITIONAL SECTION:
STRAWB.mit.edu.         126738  IN      A       18.71.0.151
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.                  IN      A

;; ANSWER SECTION:
eecs.mit.edu.           21600   IN      A       18.62.1.6

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                11088   IN      NS      STRAWB.mit.edu.

;; ADDITIONAL SECTION:
STRAWB.mit.edu.         126738  IN      A       18.71.0.151
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

The question we asked the server

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.                  IN      A

;; ANSWER SECTION:
eecs.mit.edu.           2160

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                11088   IN      NS      STRAWB.mit.edu.

;; ADDITIONAL SECTION:
STRAWB.mit.edu.         126738  IN      A       18.71.0.151
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

A 16-bit **transaction identifier** that enables the DNS client (`dig`, in this case) to match up the reply with its original request

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode
;; flags: qr rd ra; QU                                            ONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.                  IN      A

;; ANSWER SECTION:
eecs.mit.edu.           21600   IN      A       18.62.1.6

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                11088   IN      NS      STRAWB.mit.edu.

;; ADDITIONAL SECTION:
STRAWB.mit.edu.         126738  IN      A       18.71.0.151
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

“**Answer**” tells us the IP address associated with `eecs.mit.edu` is `18.62.1.6` and we can cache the result for 21,600 seconds

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.                  IN      A

;; ANSWER SECTION:
eecs.mit.edu.           21600   IN      A       18.62.1.6

;; AUTHORITY SECTION:
mit.edu.                                                    edu.
mit.edu.                                                    edu.
mit.edu.                                                    .edu.

;; ADDITIONAL SECTION:
STRAWB.mit.edu.         126738  IN      A       18.71.0.151
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

In general, a single Resource Record (RR) like this includes, left-to-right, a DNS name, a time-to-live, a family (IN for our purposes - ignore), a type (A here), and an associated value

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cm
;; Got answer:
;; ->>HEADER<<- opcode
;; flags: qr rd ra; QU                                              3

;; QUESTION SECTION:
;eecs.mit.edu.

;; ANSWER SECTION:
eecs.mit.edu.           21600   IN      A       18.62.1.6

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                11088   IN      NS      STRAWB.mit.edu.

;; ADDITIONAL SECTION:
STRAWB.mit.edu.         126738  IN      A       18.71.0.151
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

"**Authority**" tells us the name servers responsible for the answer. Each RR gives the hostname of a different name server ("NS") for names in `mit.edu.` We should cache each record for 11,088 seconds.

If the "**Answer**" had been empty, then the resolver's next step would be to send the original query to one of these name servers.

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.

;; ANSWER SECTION
eecs.mit.edu.

;; AUTHORITY SECTION:
mit.edu.                 11088   IN      NS      BITSY.mit.edu.
mit.edu.                 11088   IN      NS      W20NS.mit.edu.
mit.edu.                 11088   IN      NS      STRAWB.mit.edu.

;; ADDITIONAL SECTION:
STRAWB.mit.edu.          126738  IN      A       18.71.0.151
BITSY.mit.edu.           166408  IN      A       18.72.0.3
W20NS.mit.edu.           126738  IN      A       18.70.0.160
```

"**Additional**" provides extra information to save us from making separate lookups for it, or helps with bootstrapping.

Here, it tells us the IP addresses for the hostnames of the name servers. We add these to our cache.

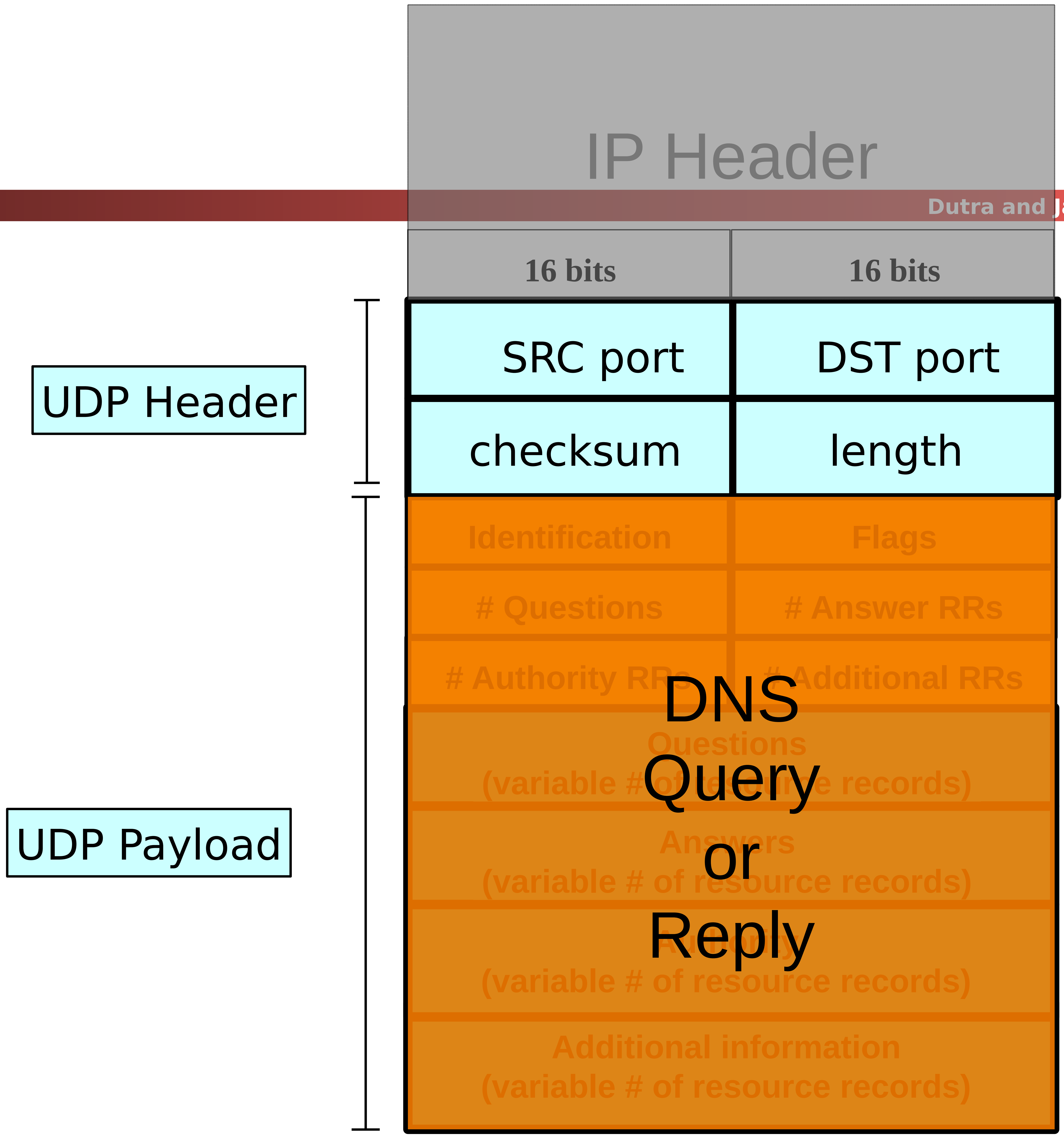
DNS Protocol

Lightweight exchange of *query* and *reply* messages, both with same message format

Primarily uses UDP for its transport protocol, which is what we'll assume

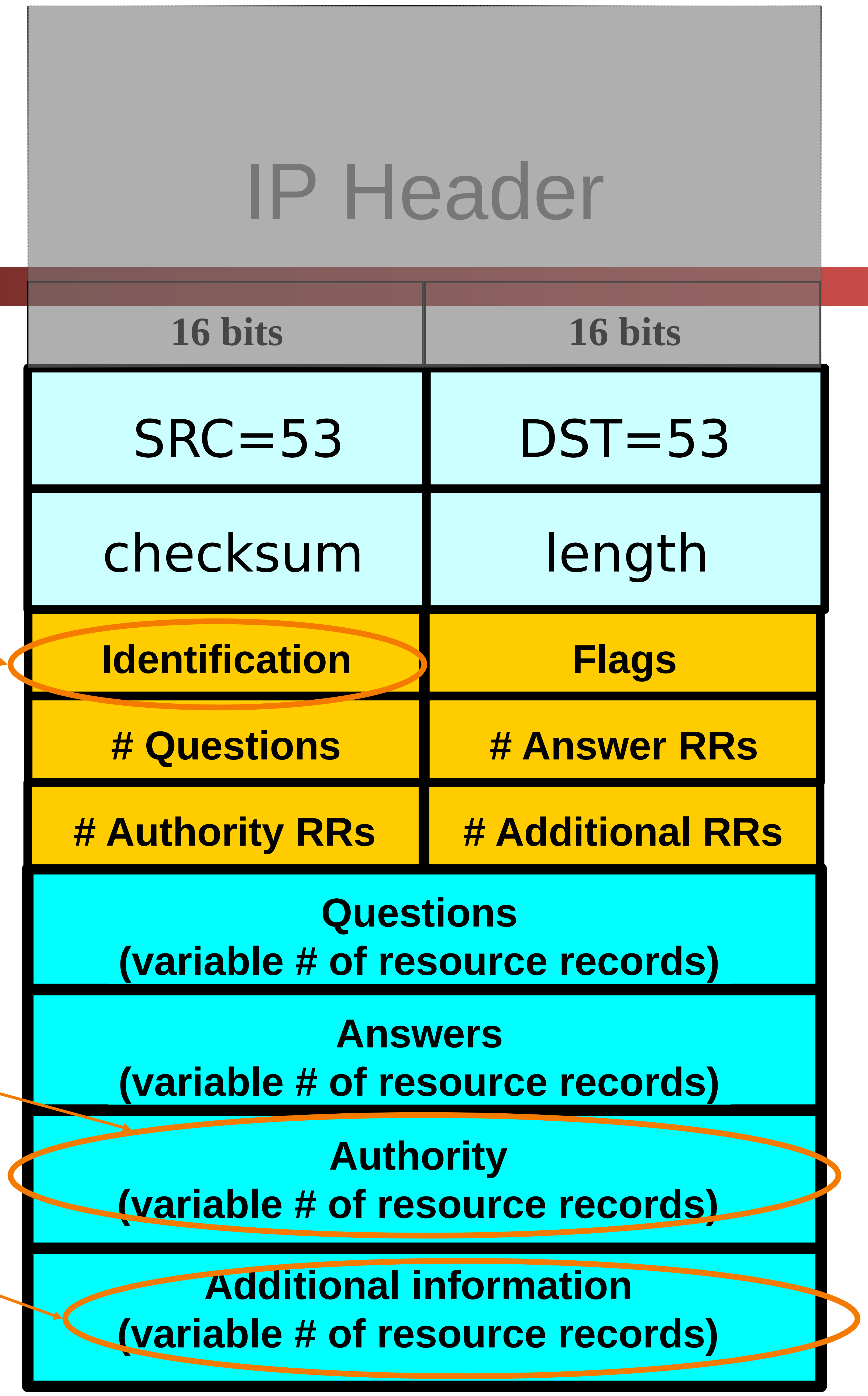
Servers are on port 53 always

Frequently, clients used to use port 53 but can use any port


IP Header
16 bits
16 bits
UDP Header
SRC port
DST port
checksum
length
UDP Payload
Identification
Flags
# Questions
# Answer RRs
# Authority RRs
# Additional RRs
Questions (variable # of resource records)
Answers (variable # of resource records)
Authority (variable # of resource records)
Additional information (variable # of resource records)
DNS Query or Reply

Message header:

- Identification: 16 bit # for query, reply to query uses same #
- Along with repeating the Question and providing Answer(s), replies can include "**Authority**" (name server responsible for answer) and "**Additional**" (info client is likely to look up soon anyway)
- Each Resource Record has a Time To Live (in seconds) for **caching** (not shown)


IP Header
16 bits
16 bits
SRC=53
DST=53
checksum
length
Identification
Flags
# Questions
# Answer RRs
# Authority RRs
# Additional RRs
Questions (variable # of resource records)
Answers (variable # of resource records)
Authority (variable # of resource records)
Additional information (variable # of resource records)

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QU                              901
;; flags: qr rd ra; QUERY:                               ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.

;; ANSWER SECTION:
eecs.mit.edu.           21                              1.6

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                11088   IN      NS      STRAWB.mit.edu.

;; ADDITIONAL SECTION:
STRAWB.mit.edu.         126738  IN      A       18.71.0.151
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

What if the mit.edu server is untrustworthy? Could its operator steal, say, all of our web surfing to berkeley.edu’s main web server?

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.

;; ANSWER SECTION:
eecs.mit.edu.           21600   IN      A       18.62.1.6

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                11088   IN      NS      STRAWB.mit.edu.

;; ADDITIONAL SECTION:
STRAWB.mit.edu.         126738  IN      A       18.71.0.151
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

Let's look at a flaw in the original DNS design (since fixed)

dig eecs.mit.edu A

What could happen if the mit.edu server returns the following to us instead?

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.

;; ANSWER SECTION:
eecs.mit.edu.          21600   IN      A       18.62.1.6

;; AUTHORITY SECTION:
mit.edu.               11088   IN      NS      BITSY.mit.edu.
mit.edu.               11088   IN      NS      W20NS.mit.edu.
mit.edu.               11088   IN      NS      www.berkeley.edu.

;; ADDITIONAL SECTION:
www.berkeley.edu.      100000  IN      A       18.6.6.6
BITSY.mit.edu.         166408  IN      A       18.72.0.3
W20NS.mit.edu.         126738  IN      A       18.70.0.160
```

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.                  IN      A

;; ANSWER SECTION:
eecs.mit.edu.

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                11088   IN      NS      www.berkeley.edu.

;; ADDITIONAL SECTION:
www.berkeley.edu.       100000  IN      A       18.6.6.6
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

We'd dutifully store in our cache a mapping of `www.berkeley.edu` to an IP address under MIT's control. (It could have been any IP address they wanted, not just one of theirs.)

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.                  IN      A

;; ANSWER SECTION:
eecs.mit.edu.                                               6

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                11088   IN      NS      www.berkeley.edu.

;; ADDITIONAL SECTION:
www.berkeley.edu.       100000  IN      A       18.6.6.6
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

In this case they chose to make the mapping last a long time. They could just as easily make it for just a couple of seconds.

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.                   IN      A

;; ANSWER SECTION:
eecs.mit.edu.           21600    IN      A       18.62.1.6

;; AUTHORITY SECTION:
mit.edu.                11088    IN      NS      BITSY.mit.edu.
mit.edu.                11088    IN      NS      W20NS.mit.edu.
mit.edu.                30       IN      NS      www.berkeley.edu.

;; ADDITIONAL SECTION:
www.berkeley.edu.       30       IN      A       18.6.6.6
BITSY.mit.edu.          166408   IN      A       18.72.0.3
W20NS.mit.edu.          126738   IN      A       18.70.0.160
```

How do we fix such **cache poisoning**?

dig eecs.mit.edu A

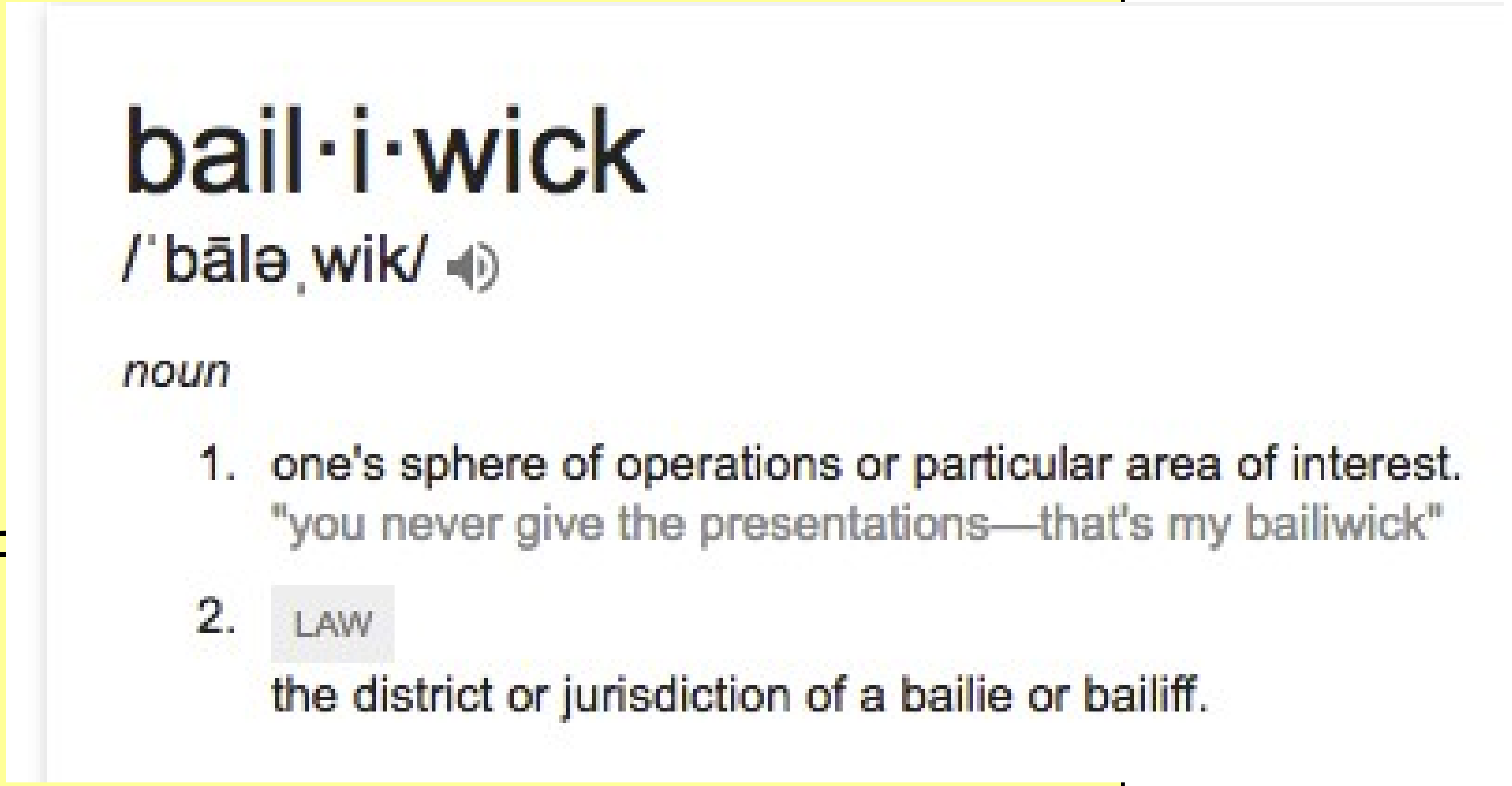
```
; ; <<>> DiG 9.6.0-AP
;; global options: +c
;; Got answer:
;; ->>HEADER<<- opcod
;; flags: qr rd ra; Q

;; QUESTION SECTION:
;eecs.mit.edu.

;; ANSWER SECTION:
eecs.mit.edu.           21600   IN      A       18.62.1.6

;; AUTHORITY SECTION:
mit.edu.                11088   IN
mit.edu.                11088   IN
mit.edu.                11088   IN

;; ADDITIONAL SECTION:
~~www.berkeley.edu      100000  IN~~
BITSY.mit.edu.          166408  IN
W20NS.mit.edu.          126738  IN
```

Don’t accept **Additional** records unless they’re for the domain we’re looking up

E.g., looking up `eecs.mit.edu` ⇒ only accept additional records from `*.mit.edu`

No extra risk in accepting these since server could return them to us directly in an **Answer** anyway.

This is called "***Bailiwick*** checking"



bail·i·wick

/ˈbāləˌwik/

noun

1. one's sphere of operations or particular area of interest. "you never give the presentations—that's my bailiwick"
2. LAW
   the district or jurisdiction of a bailie or bailiff.