Network Security

And We Call It "Machine Learning"

Network Security

- Why study network security?
  - Networking greatly extends our overall attack surface
    - Networking = the Internet
  - Opportunity to see *how large-scale design affects security issues*
  - Protocols a great example of *mindless agents* in action
- This lecture: sufficient background in networking to then explore security issues in next ~5 lectures
- Complex topic with many facets
  - We will omit concepts/details that aren't very security-relevant
    - But to no small extent we are speed running about 1/2 a dozen worth of "networking" lectures!
  - By all means, ask questions when things are unclear

Protocols

- A protocol is an agreement on how to communicate
- Includes syntax and semantics
  - How a communication is specified & structured
    - Format, order messages are sent and received
  - What a communication means
    - Actions taken when transmitting, receiving, or timer expires
- E.g.: making a comment in lecture?

1. Raise your hand.
2. Wait to be called on.
3. Or: wait for speaker to **pause** and vocalize
4. If unrecognized (after timeout): vocalize w/ “excuse me”

So Let's Do A Google Search...

- Walk into a coffee shop
- Open a laptop
- Search google...

Coffee Shop

Coffee Shop

1. Join the wireless network

Your laptop shouts:
HEY, DOES WIRELESS NETWORK X EXIST?

Coffee Shop

1. Join the wireless network

Wireless access point(s) continually shout:
HEY, I'M WIRELESS NETWORK Y, JOIN ME!

Coffee Shop

1. Join the wireless network

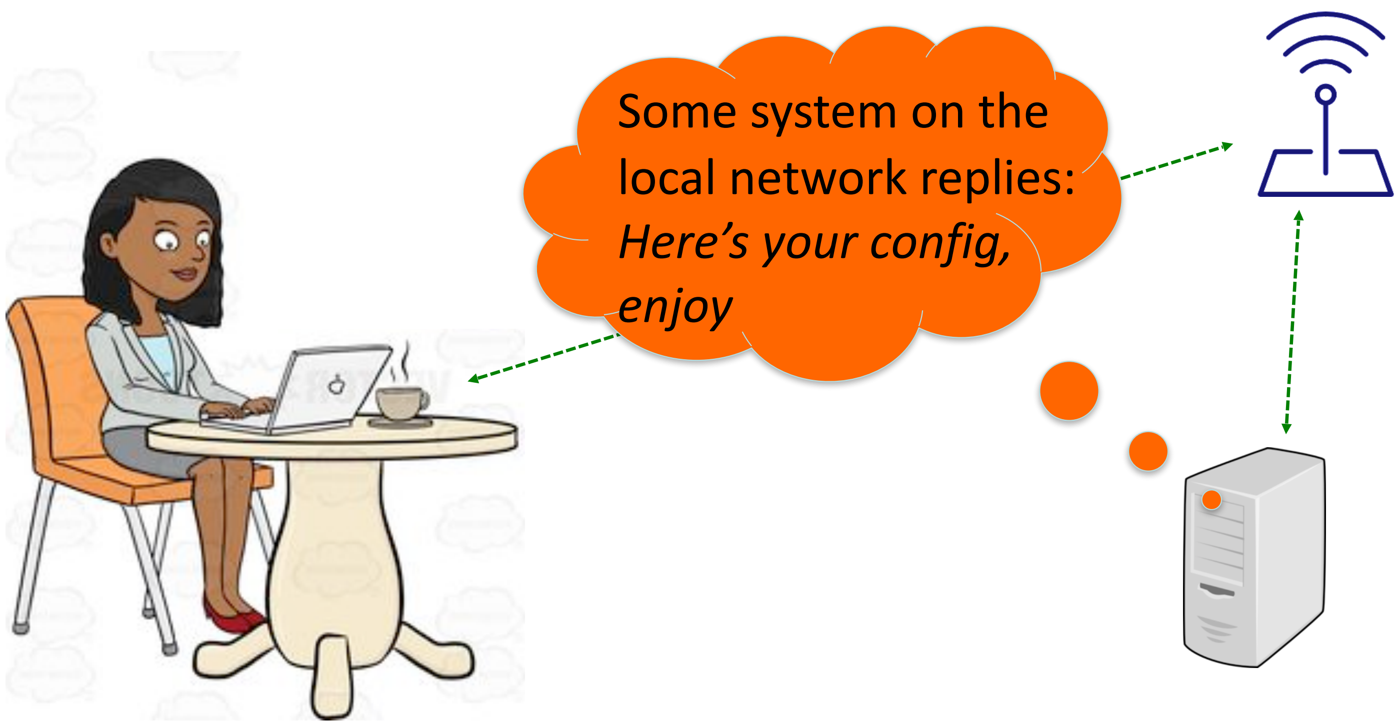
If either match up, your laptop joins the network. Optionally performs a cryptographic exchange.

Coffee Shop

2. Configure your connection

Coffee Shop

2. Configure your connection



Coffee Shop

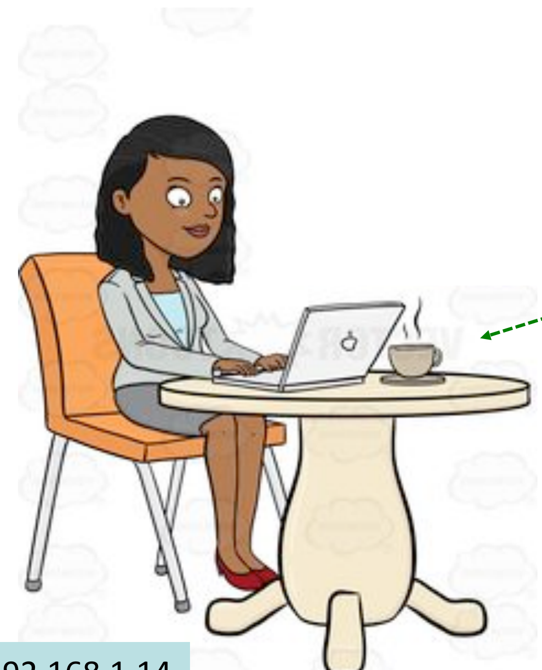
2. Configure your connection

The configuration includes:

(1) An Internet address (**IP address**) your laptop should use; typ. 32 bits (IPv4). May also include 64b of the 128b IPv6 address

(2) The address of a "**gateway**" system to use to access *hosts* beyond the local network

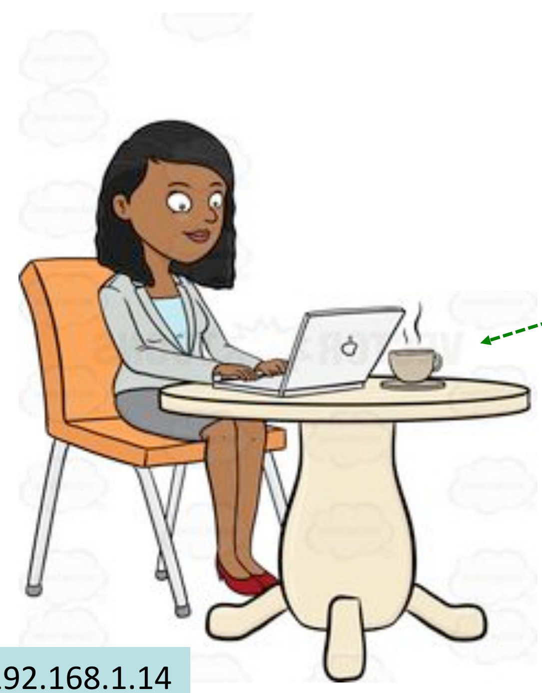
(3) The address of a **DNS server** ("*resolver*") to map names like `google.com` to IP addresses



192.168.1.14

Coffee Shop

3. Find the address of google.com

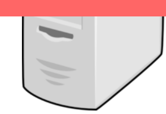


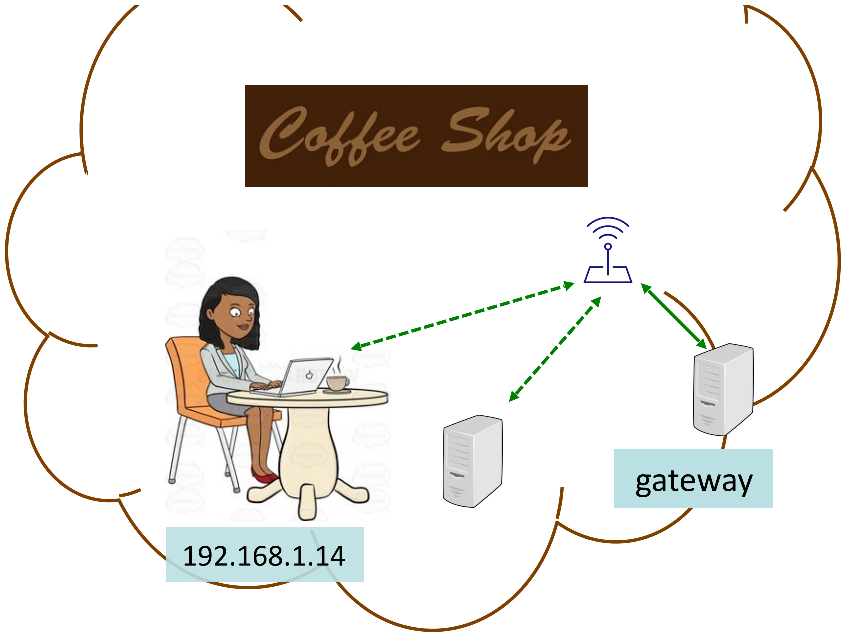
192.168.1.14

Your laptop sends a **DNS** request asking: “*address for google.com*?”

It’s transmitted using the **UDP** protocol (lightweight, unreliable).

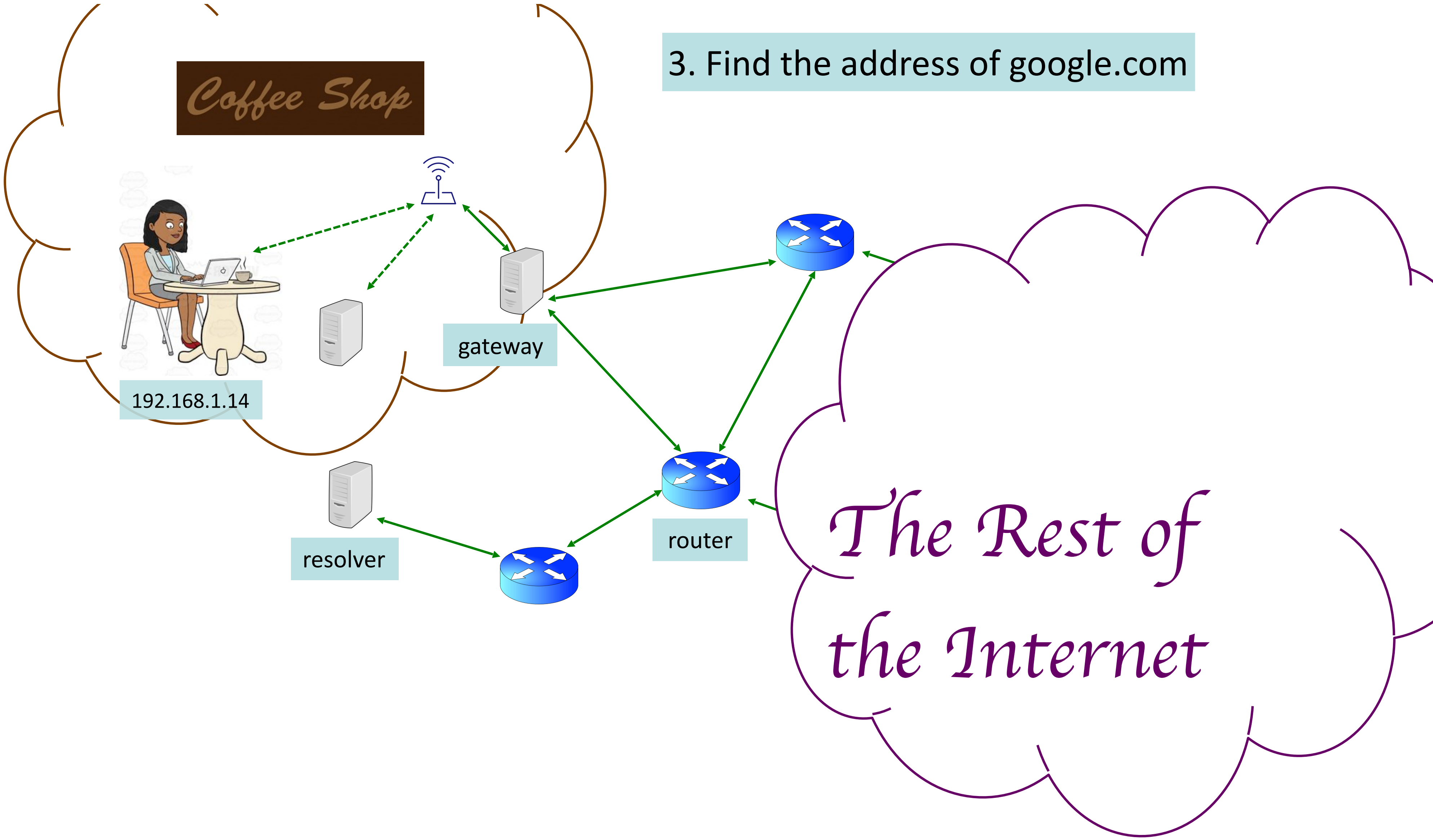
The DNS **resolver** might not be on the local network.



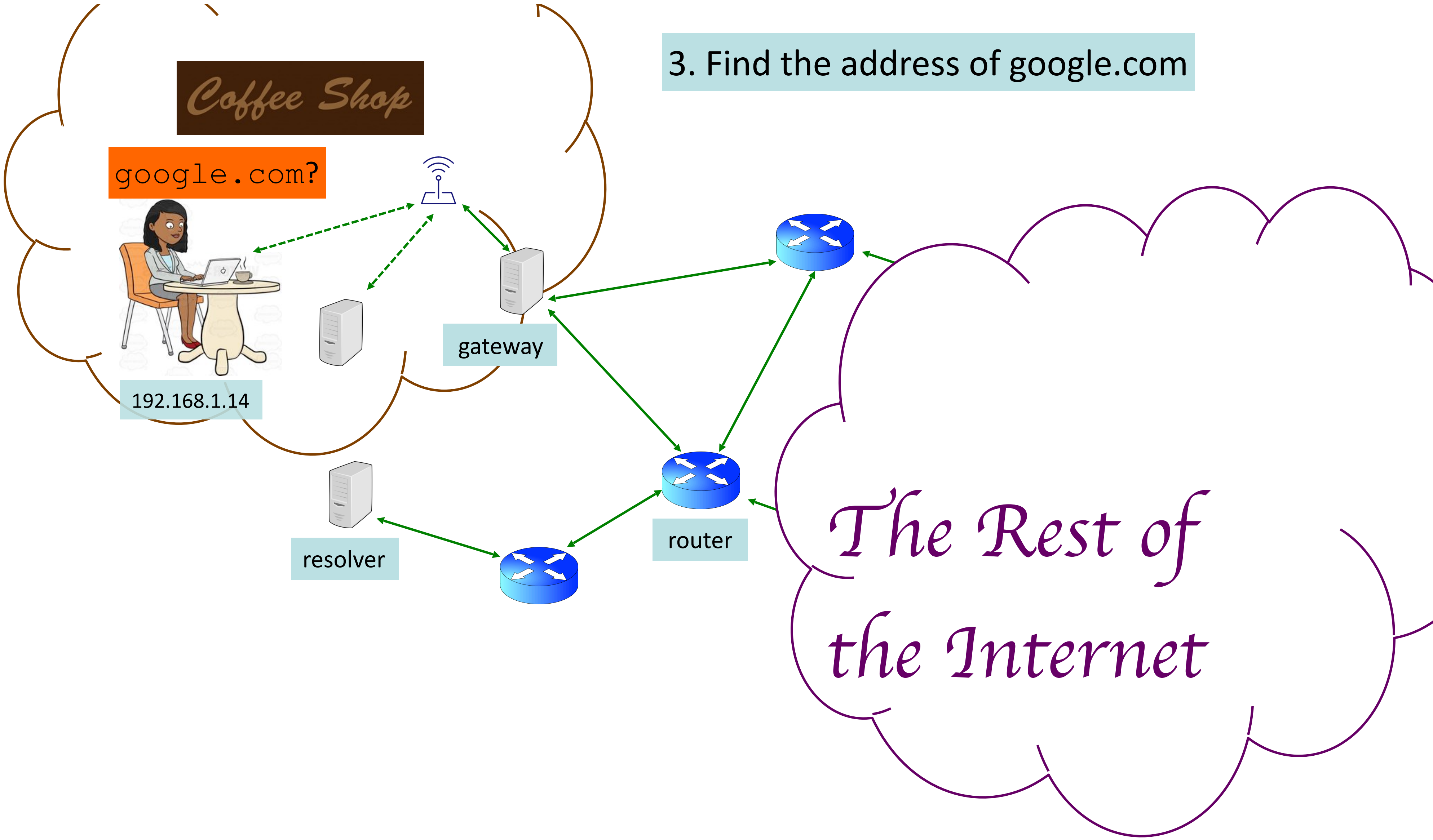


3. Find the address of google.com

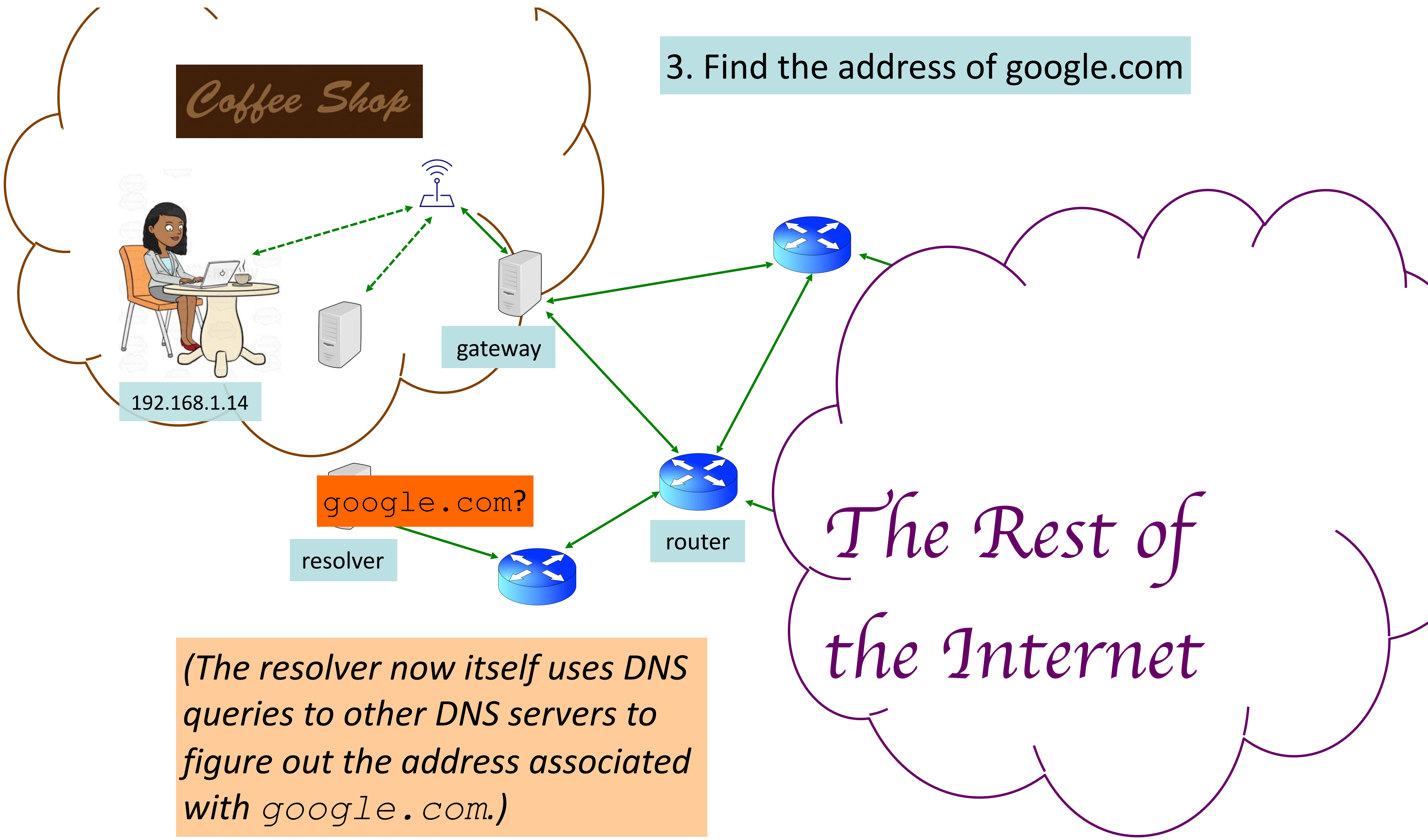
3. Find the address of google.com



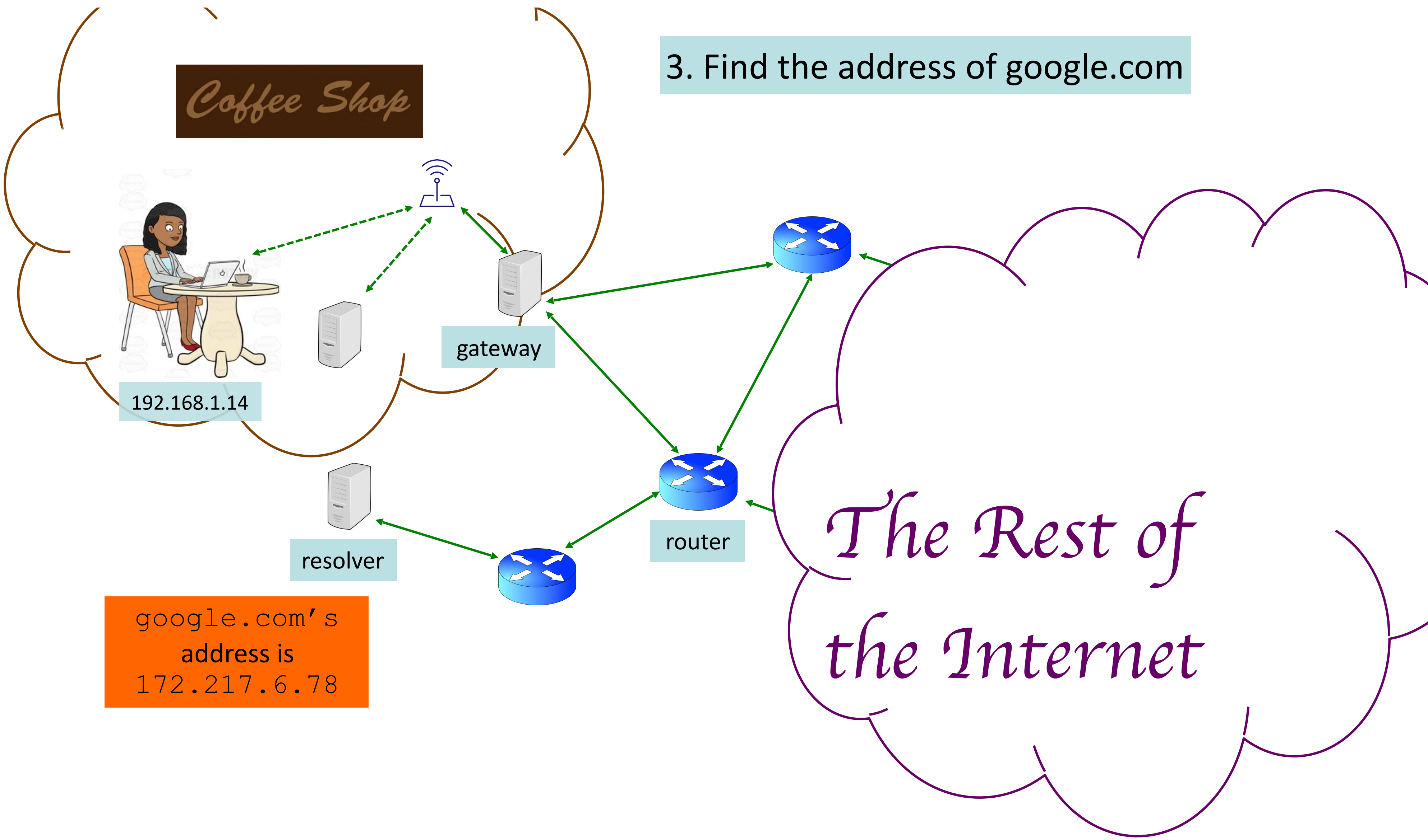
3. Find the address of google.com



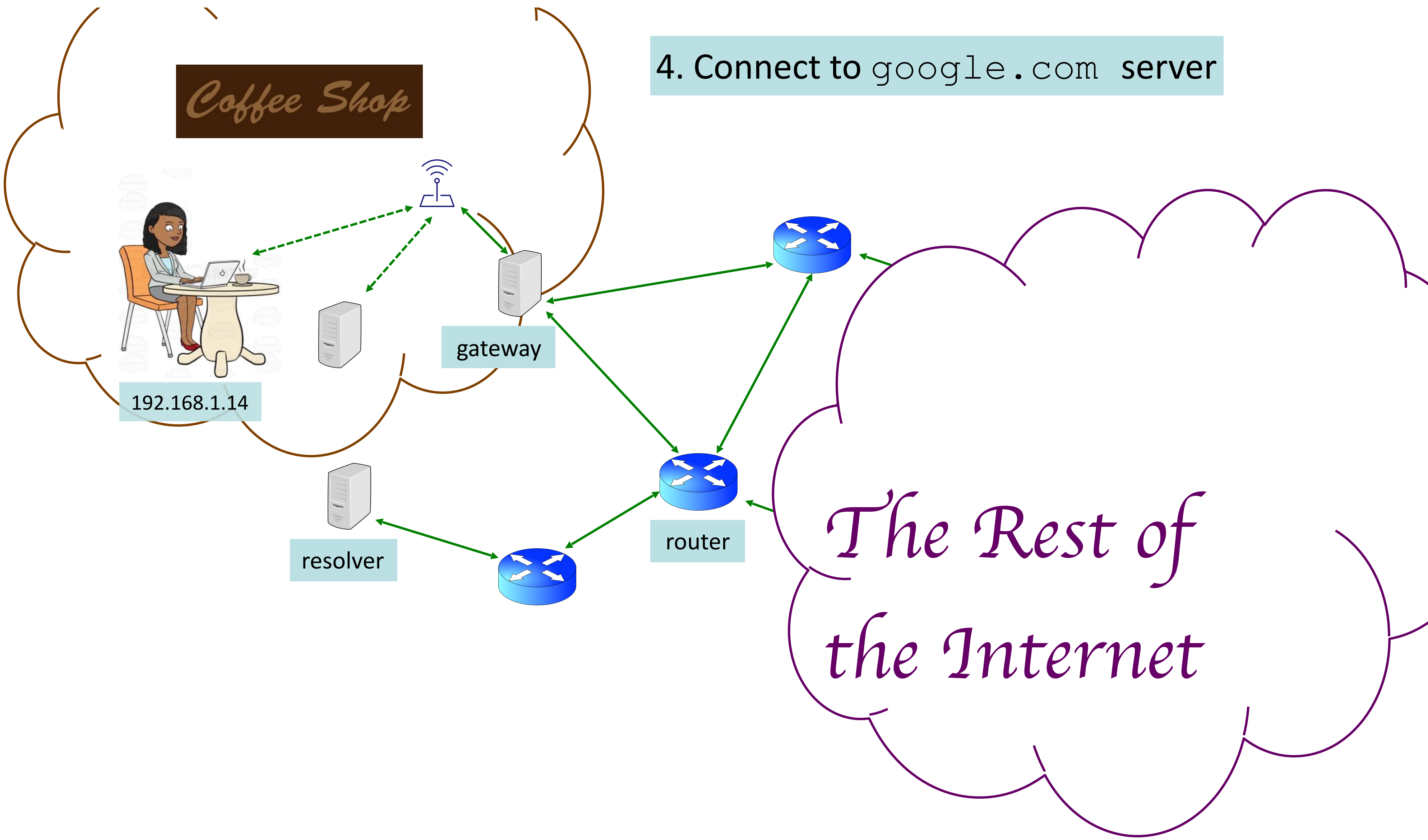
3. Find the address of google.com



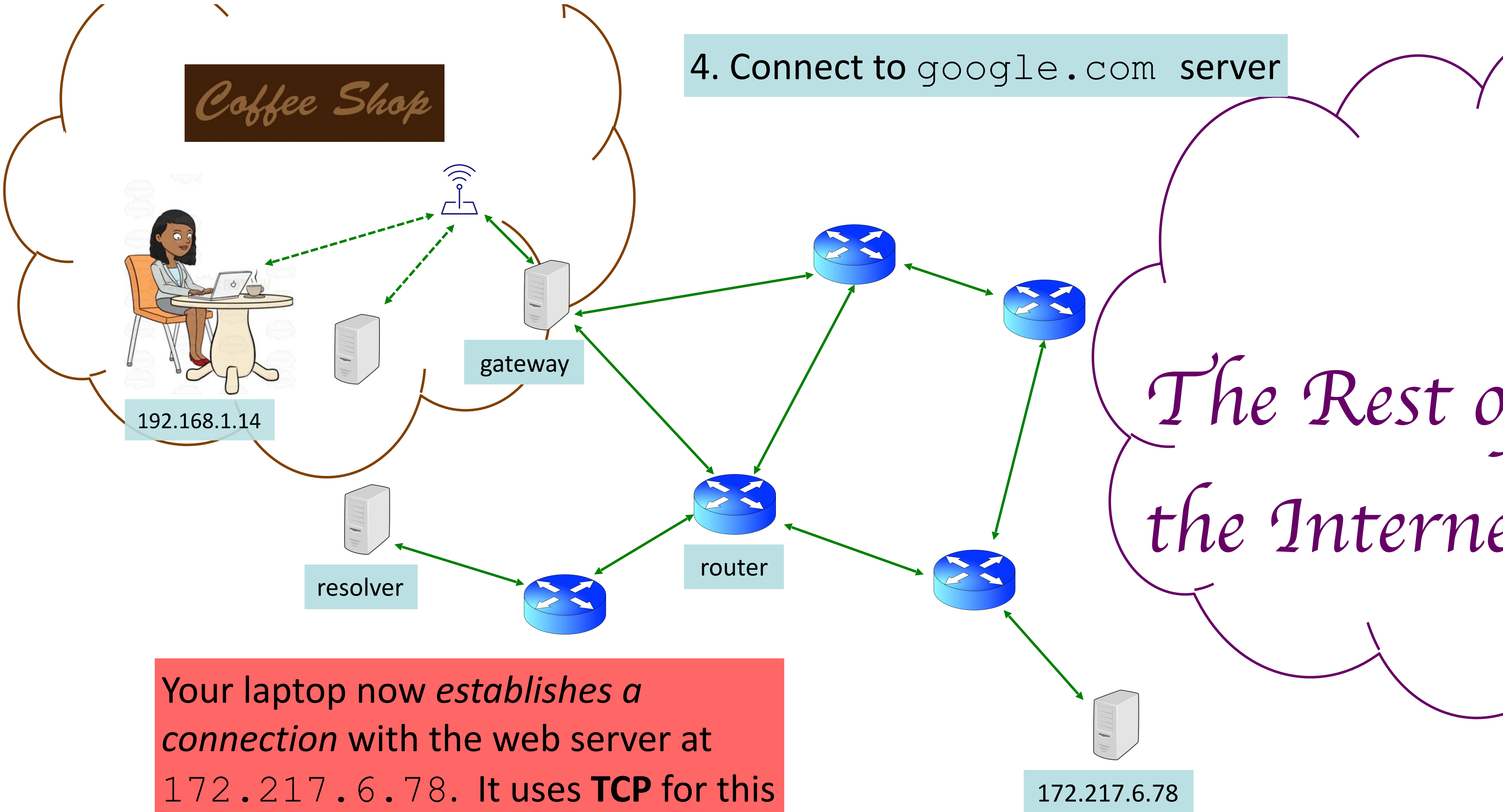
(The resolver now itself uses DNS queries to other DNS servers to figure out the address associated with `google.com`*.)*


3. Find the address of google.com
Coffee Shop
192.168.1.14
gateway
resolver
router
google.com's
address is
172.217.6.78
The Rest of
the Internet

4. Connect to `google.com` server

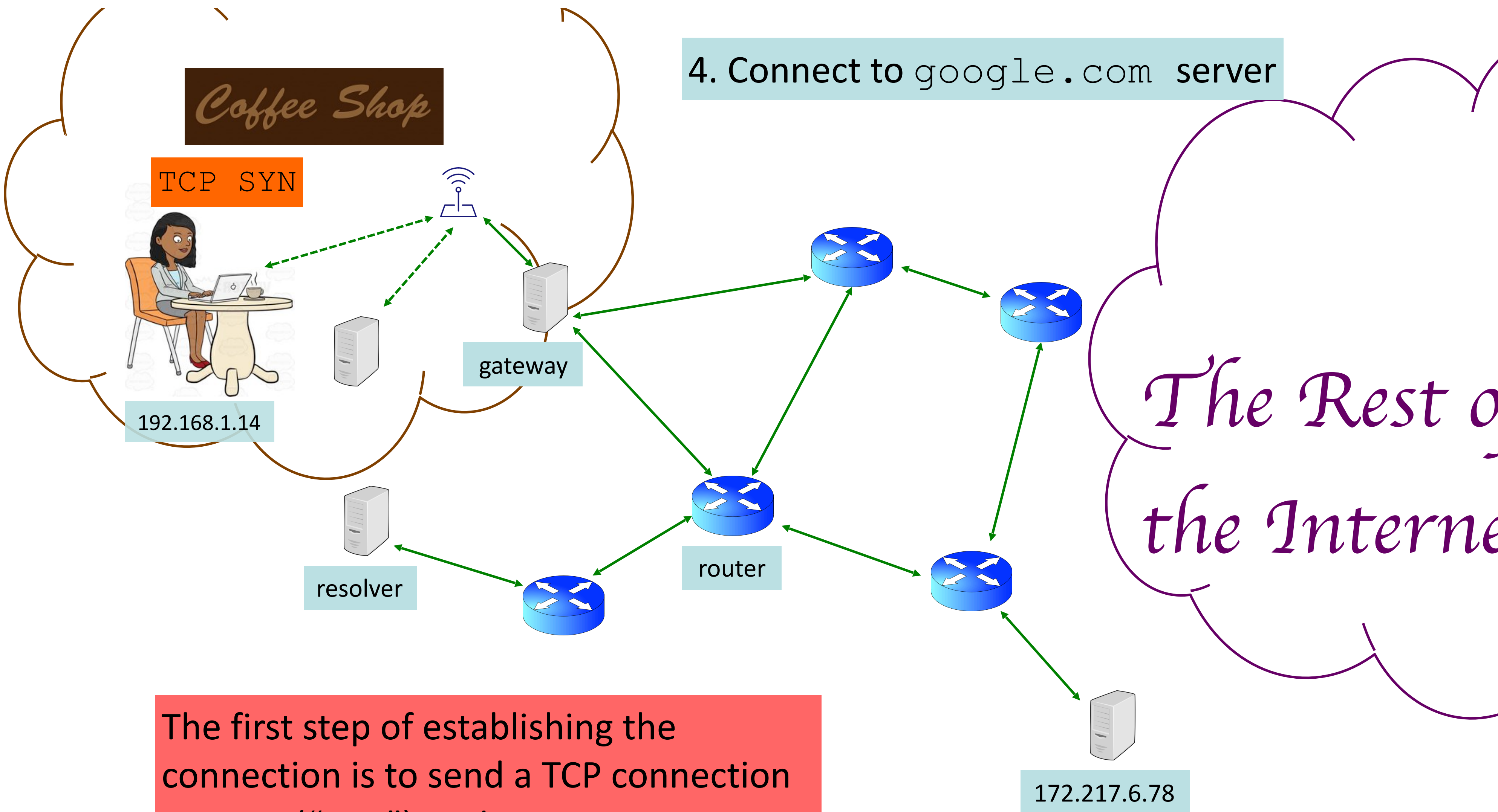


4. Connect to `google.com` server



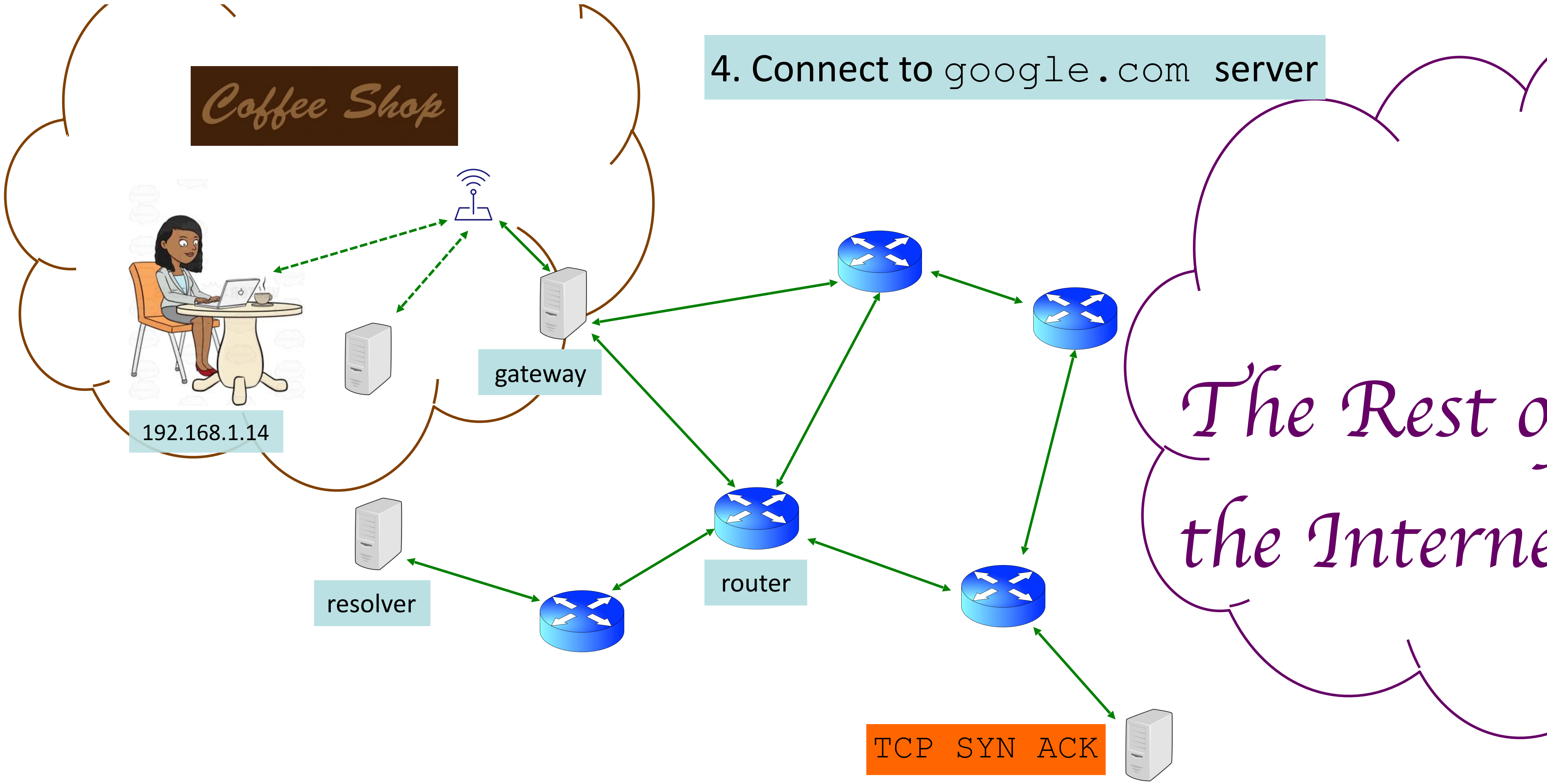
Your laptop now *establishes a connection* with the web server at `172.217.6.78`. It uses **TCP** for this rather than UDP, to obtain reliability.

4. Connect to `google.com` server

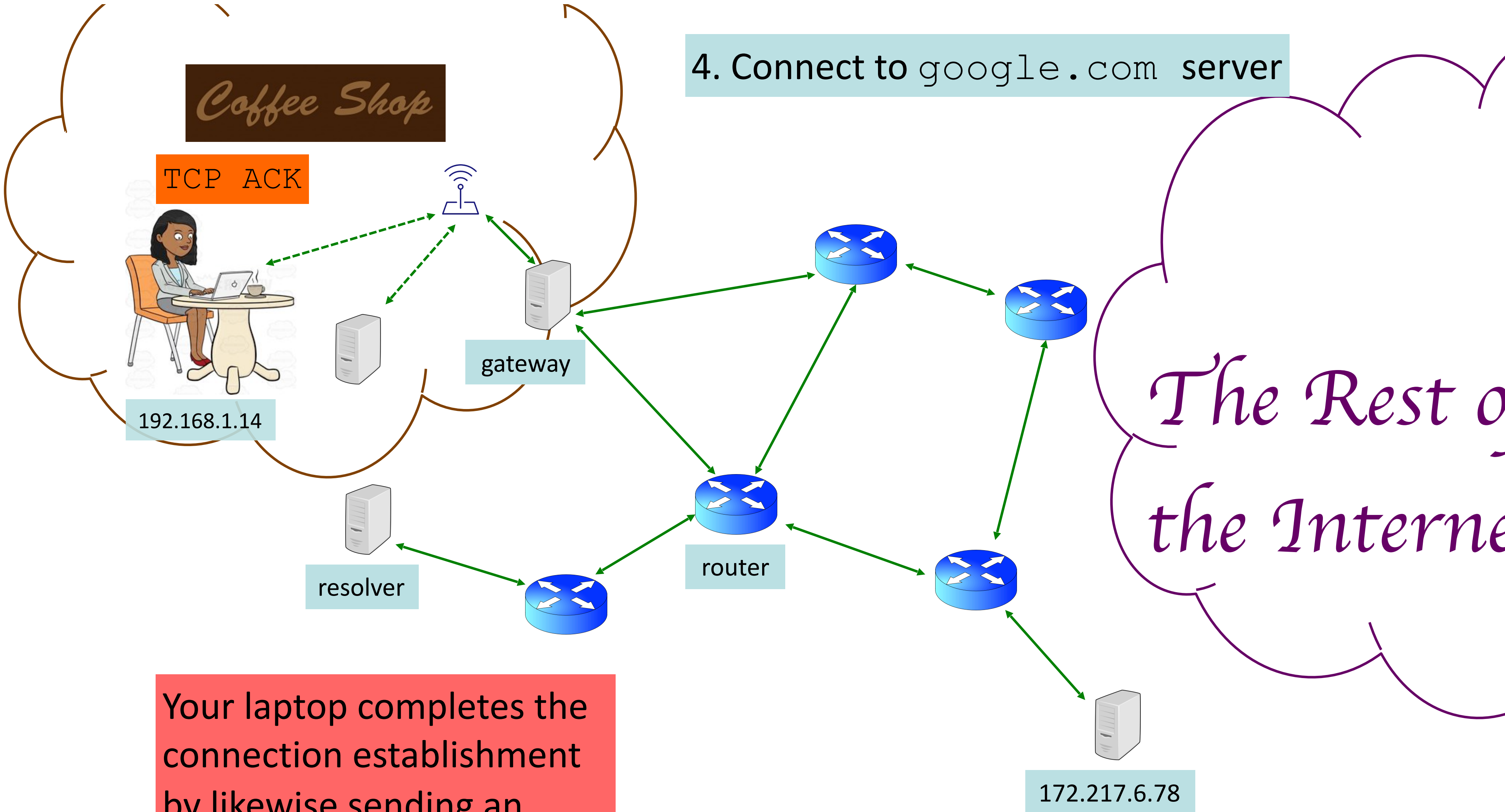


The first step of establishing the connection is to send a TCP connection request ("SYN") to the server.

4. Connect to `google.com` server

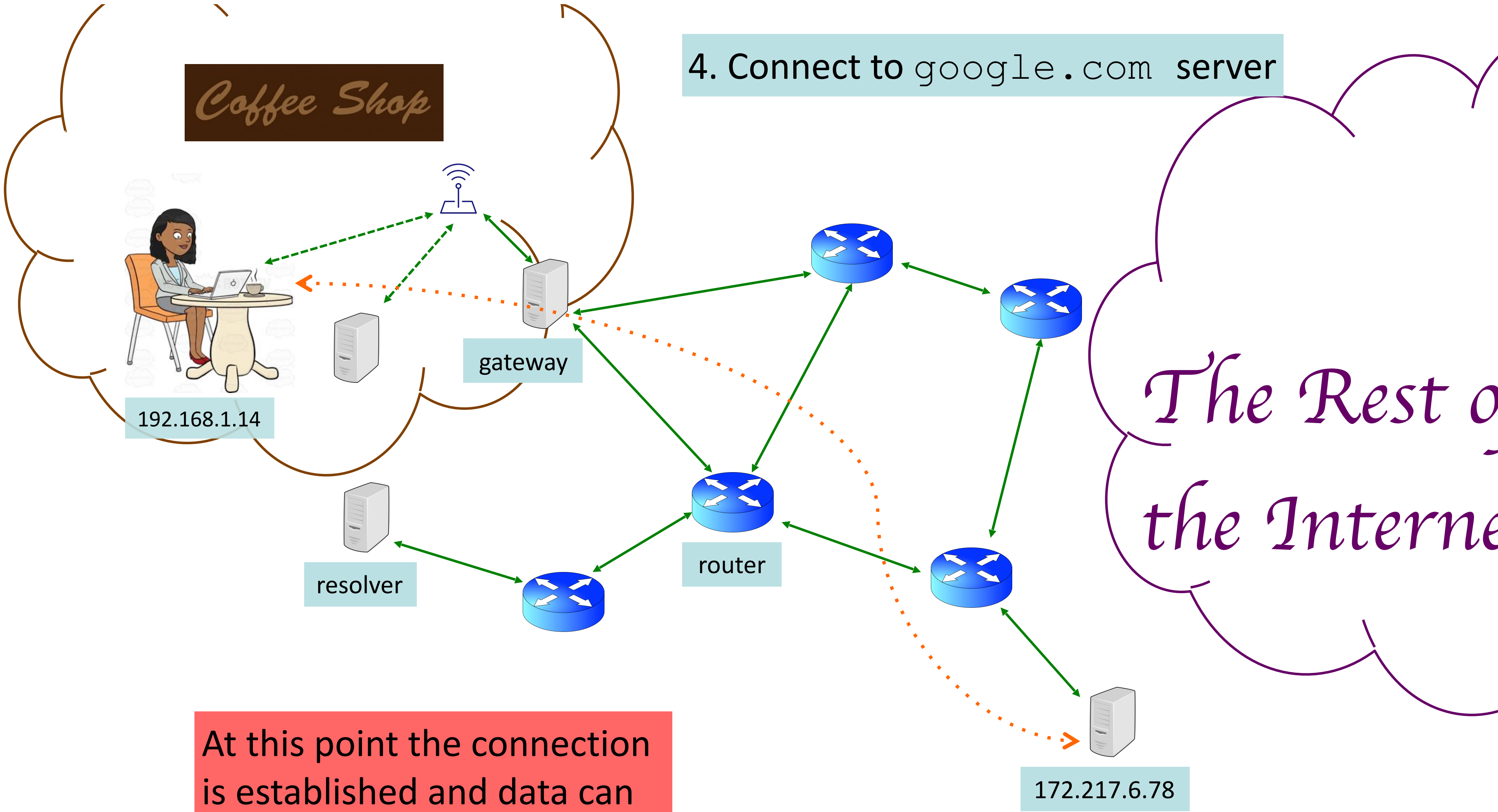


If the server accepts the connection, it replies with a “SYN ACK”.

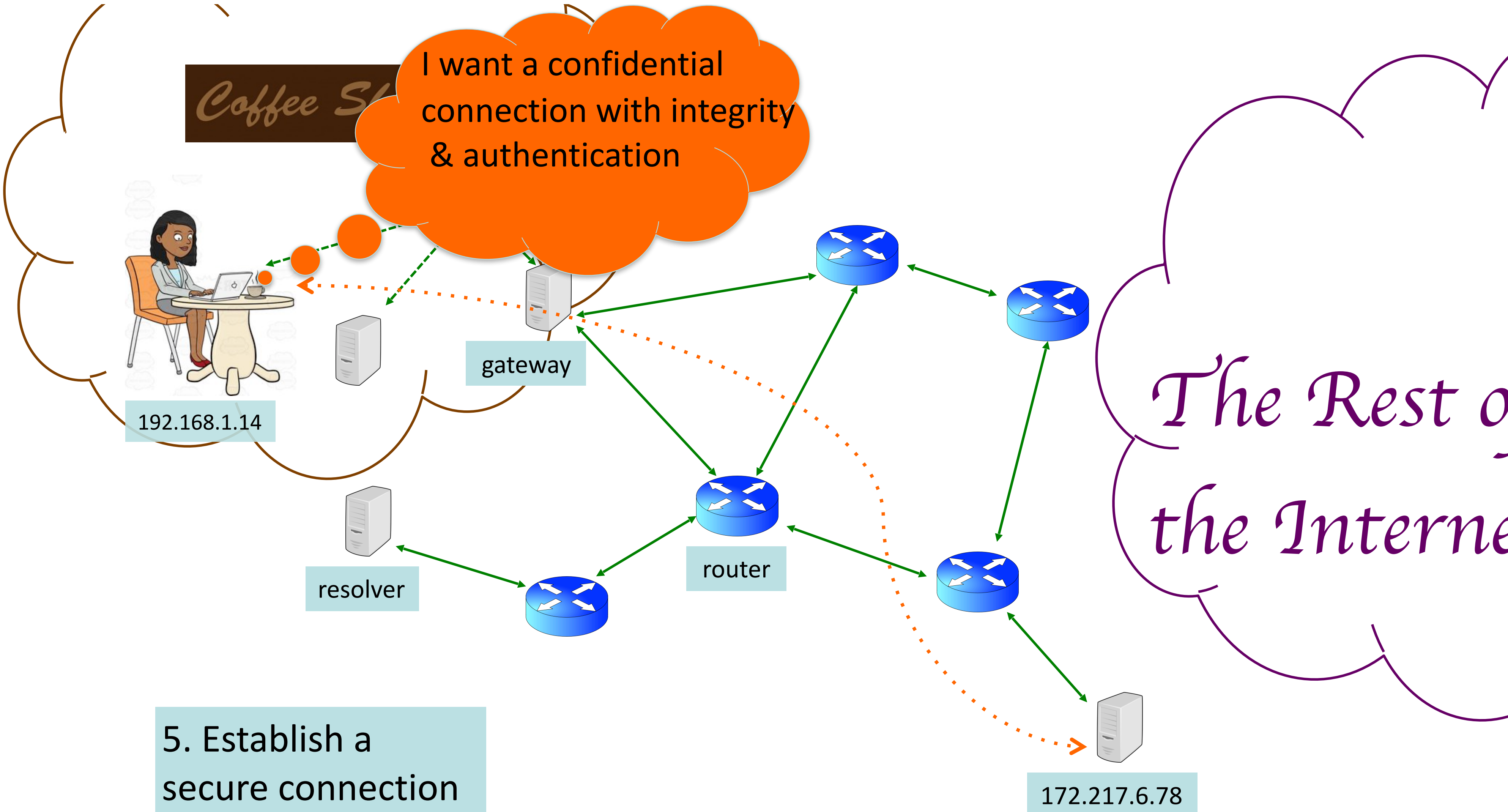


Your laptop completes the connection establishment by likewise sending an acknowledgement.

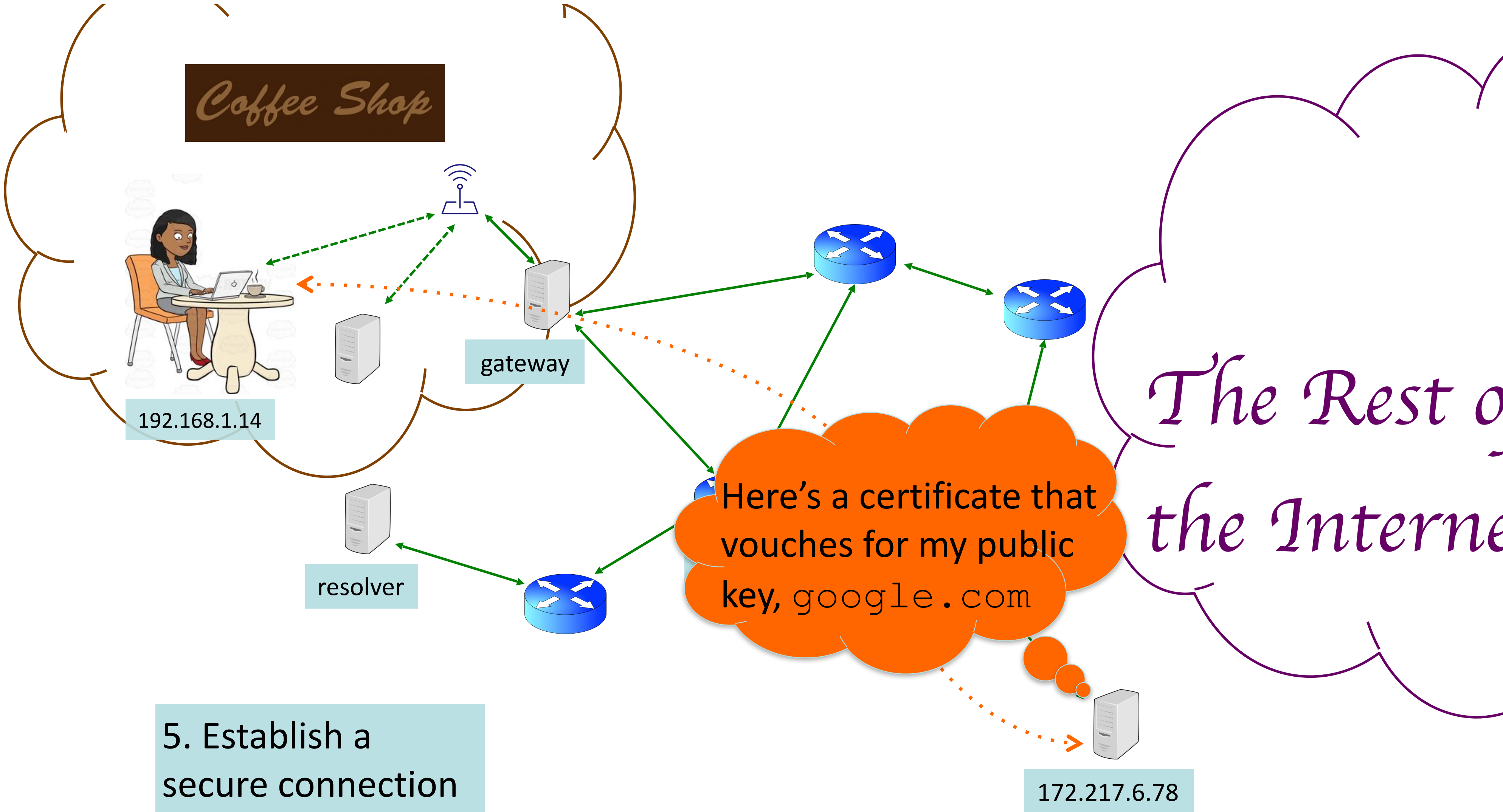
4. Connect to `google.com` server



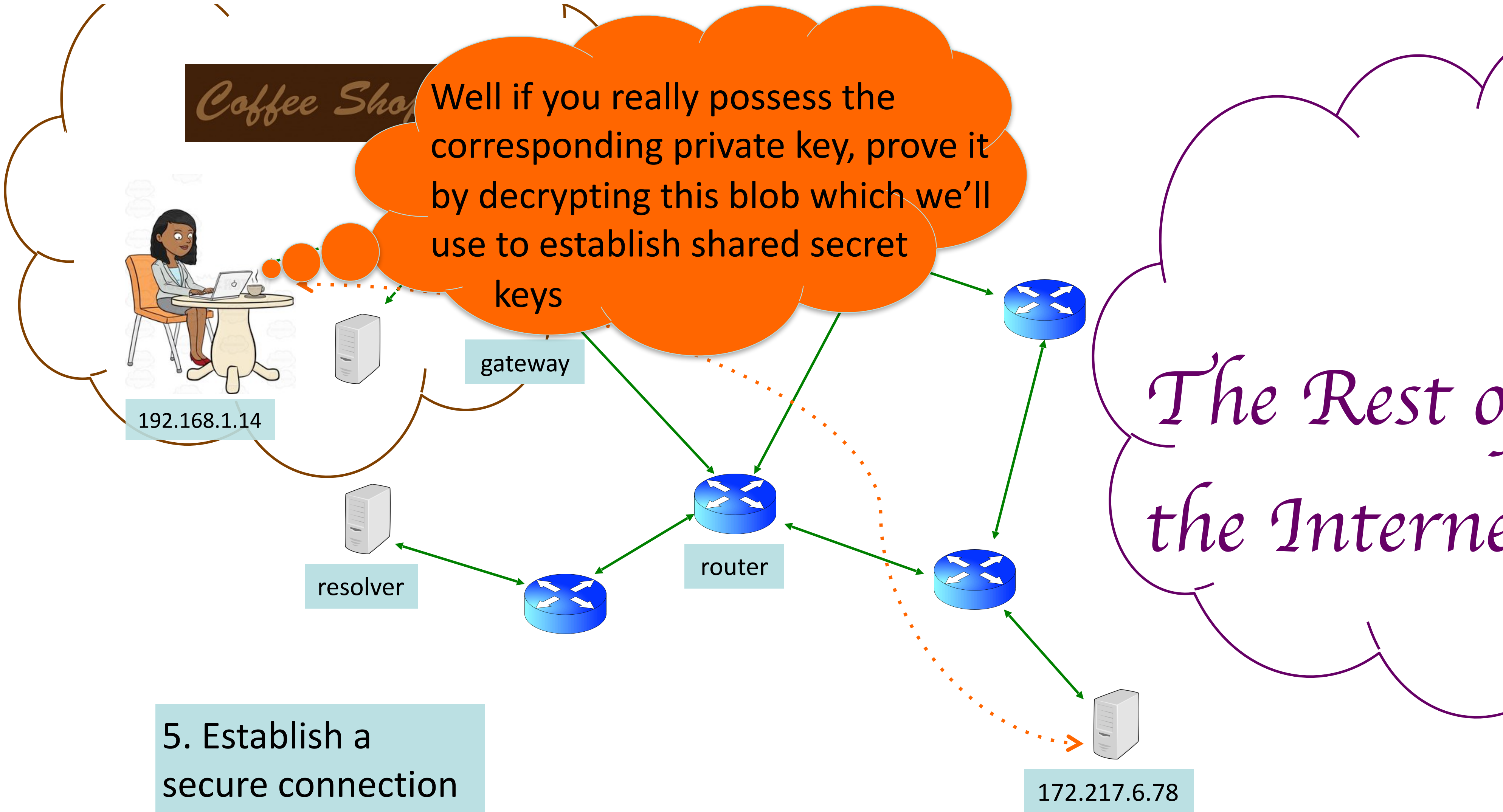
At this point the connection is established and data can be (reliably) exchanged.



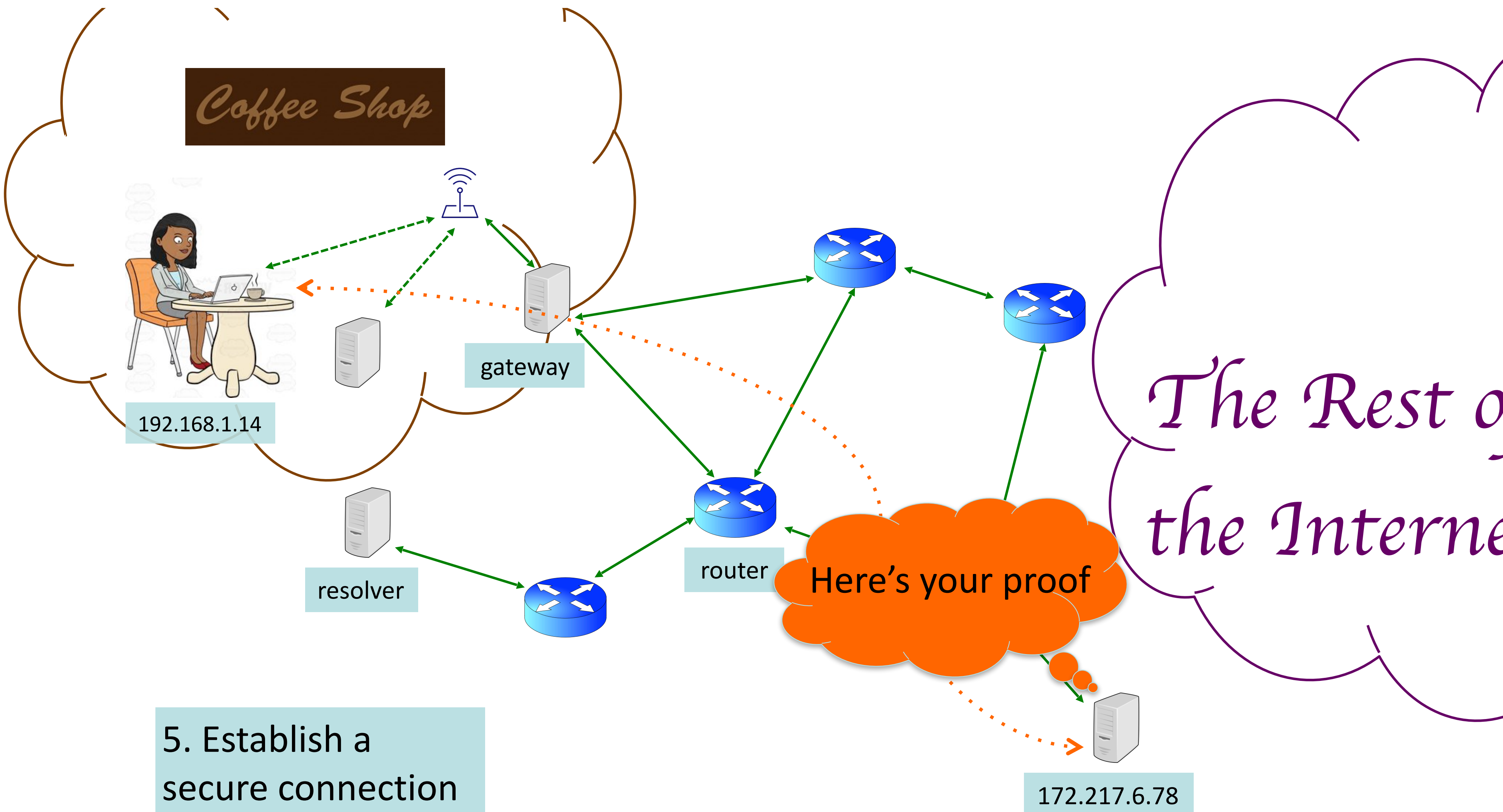
5. Establish a secure connection using **TLS** (https)



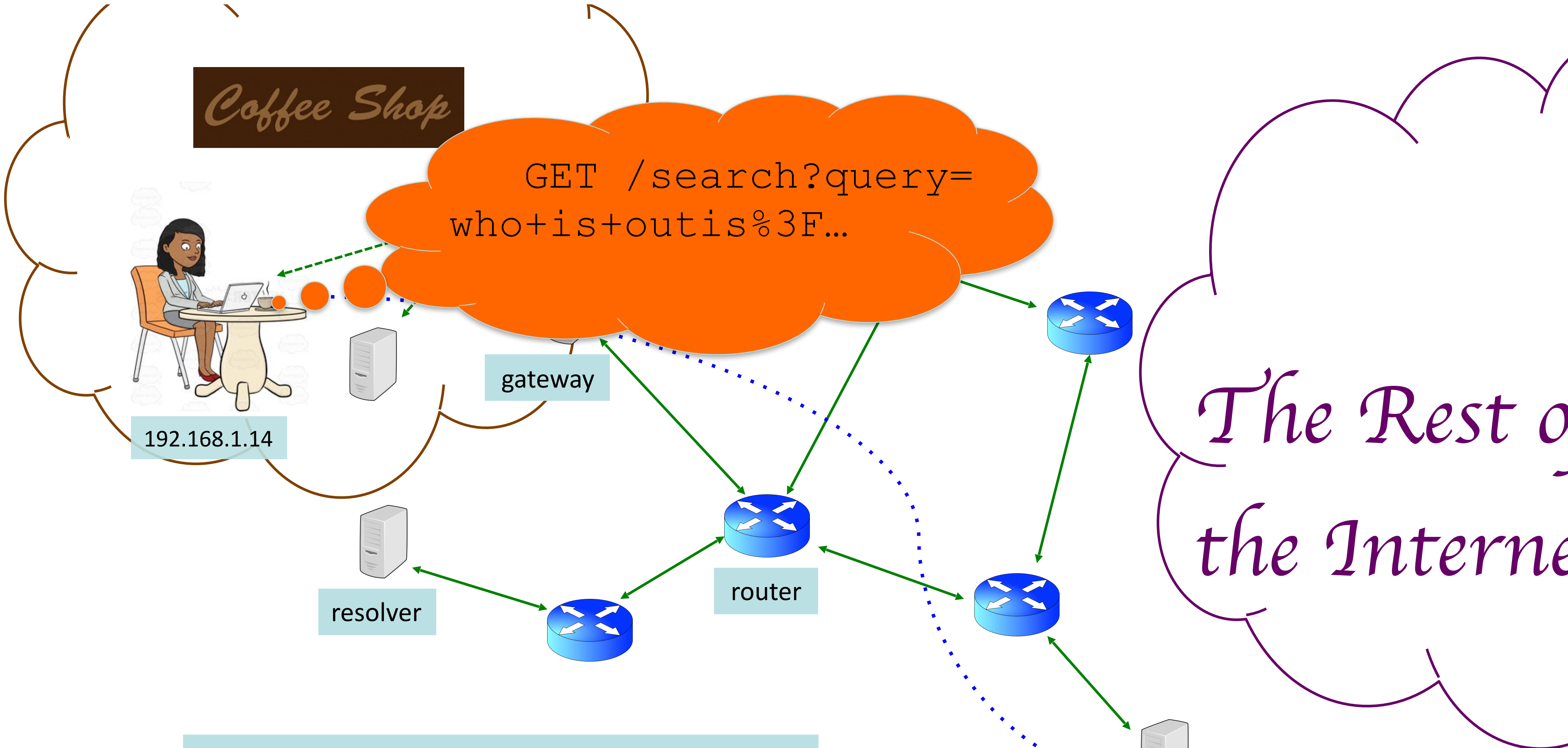
5. Establish a secure connection using **TLS** (https)



5. Establish a secure connection using **TLS** (https)



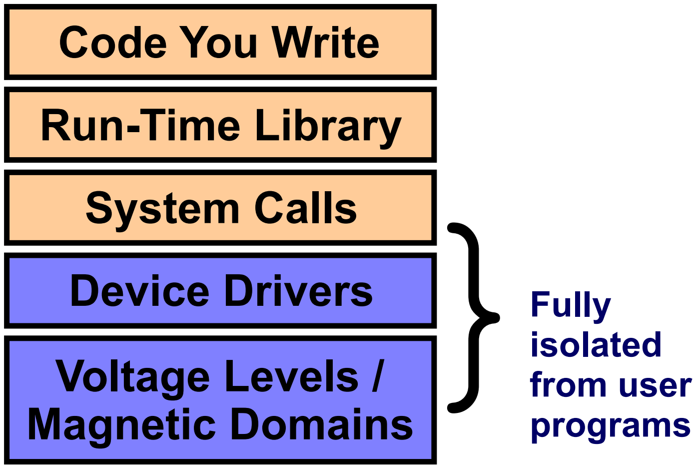
5. Establish a secure connection using **TLS** (https)



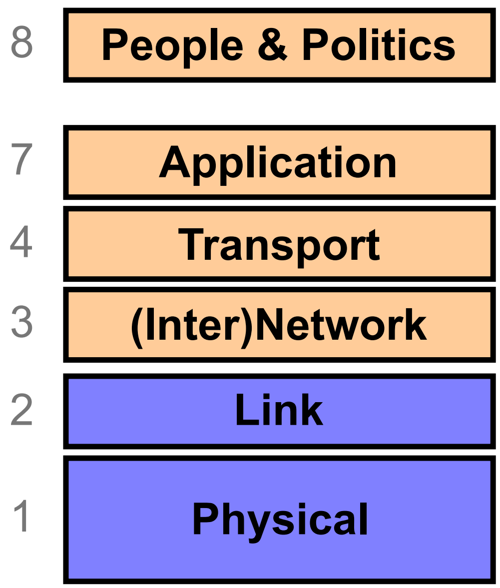
6. Finally, your laptop can send along your query!
(Using HTTP inside the *TLS channel*)

Layering

- Internet design is strongly partitioned into layers
  - Each layer relies on services provided by next layer below …
  - … and provides services to layer above it
- Analogy:
  - Consider structure of an application you've written and the "services" each layer relies on / provides



Internet Layering ("Protocol Stack")



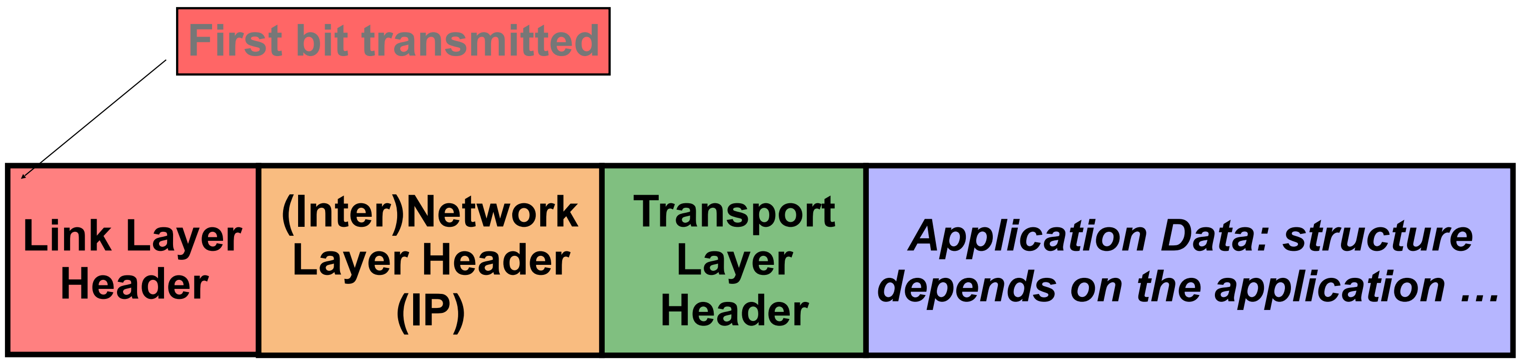
Note on a point of potential confusion: these diagrams are always drawn with lower layers **below** higher layers …

But diagrams showing the layouts of packets are often the *opposite*, with the lower layers at the **top** since their headers precede those for higher layers

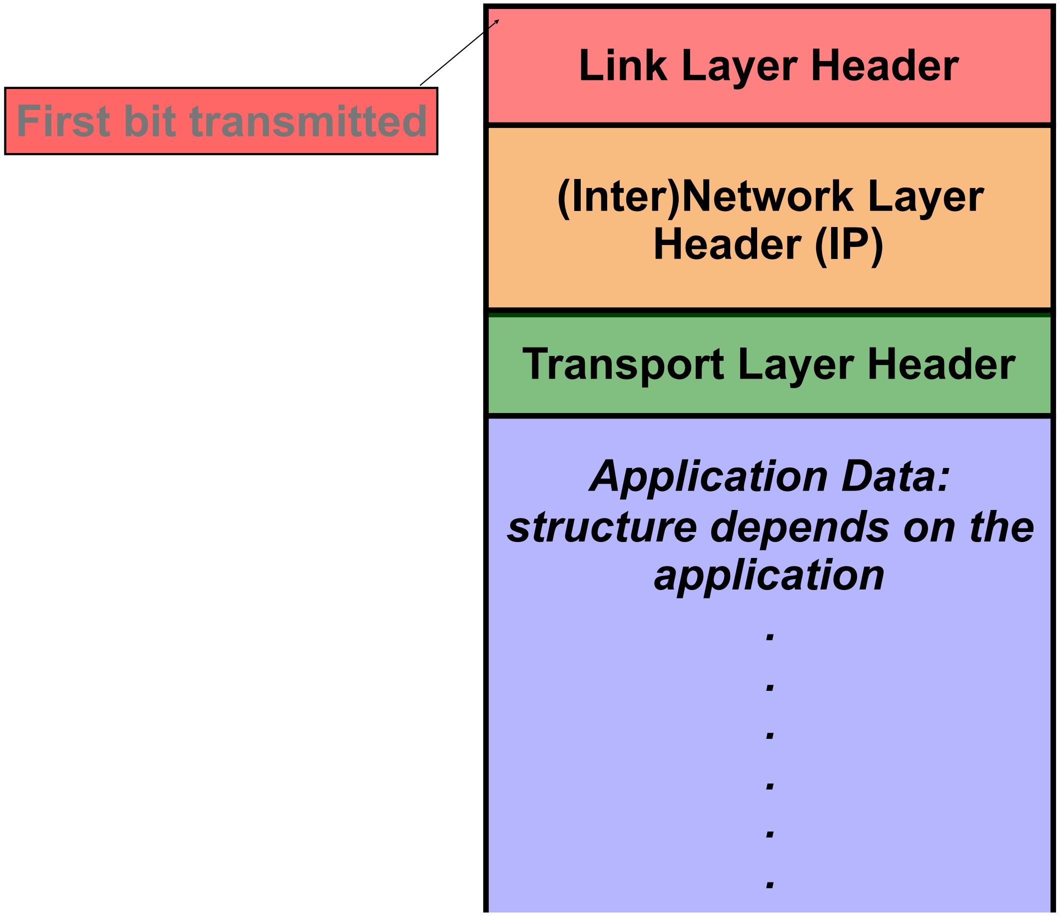
Packets and The Network

- Modern networks break communications up into packets
  - For our purposes, packets contain a variable amount of data up to a maximum specified by the particular network
- The sending computer breaks up the message and the receiving computer puts it back together
  - So the software doesn't actually see the packets per-se
  - Network itself is ***packet switched***: sending each packet on towards its next destination
- Other properties:
  - Packets are received ***correctly*** or not at all in the face of ***random*** errors
    - The network does not enforce correctness in the face of adversarial inputs: They are checksums not cryptographic MACs.
  - Packets may be ***unreliable*** and "dropped"
    - Its up to higher-level protocols to make the connection Reliable

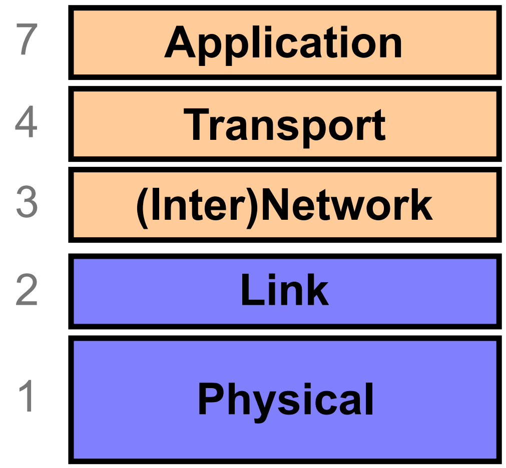
Horizontal View of a Single Packet



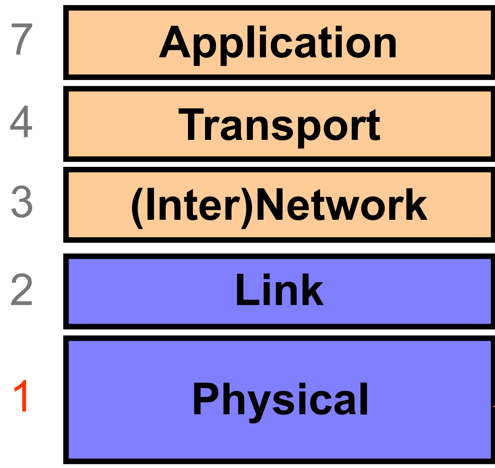
Vertical View of a Single Packet



Internet Layering ("Protocol Stack")



Layer 1: Physical Layer

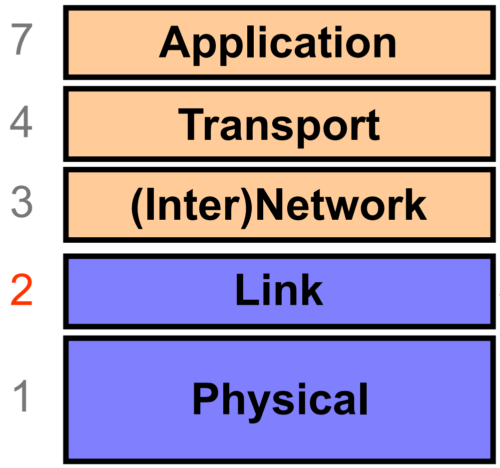


Encoding bits to send them over a <u>single</u> **physical link**

e.g. patterns of

voltage levels /
photon intensities /
RF modulation

Layer 2: Link Layer

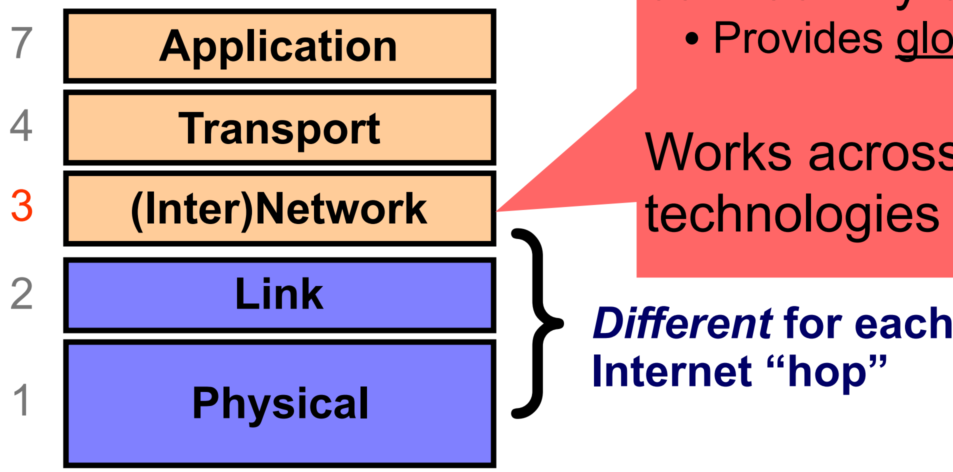


Framing and transmission of a collection of bits into individual messages sent across a single “subnetwork” (one physical technology)

Might involve multiple *physical links* (e.g., modern Ethernet)

Often technology supports broadcast transmission (**every** “node” connected to subnet receives)

Layer 3: (Inter)Network Layer *(IP)*

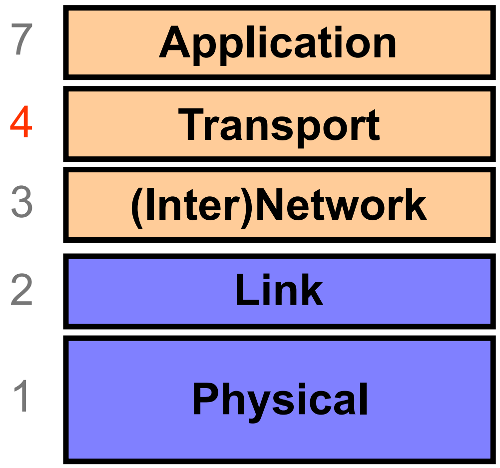


Bridges multiple "subnets" to provide *end-to-end* internet connectivity between nodes

- Provides global addressing

Works across different link technologies

Layer 4: Transport Layer

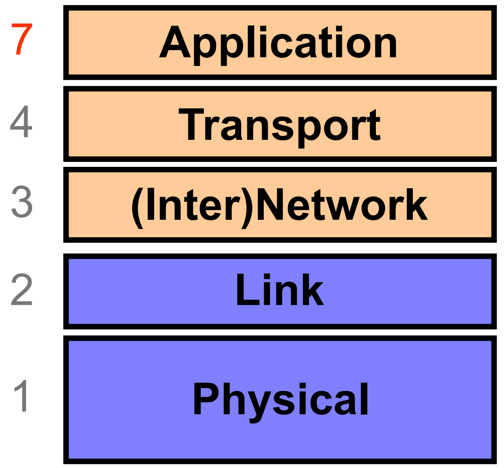


End-to-end communication between processes

Different services provided:
TCP = reliable *byte stream*
UDP = unreliable *datagrams*

(Datagram = single packet message)

Layer 7: Application Layer



Communication of whatever you wish

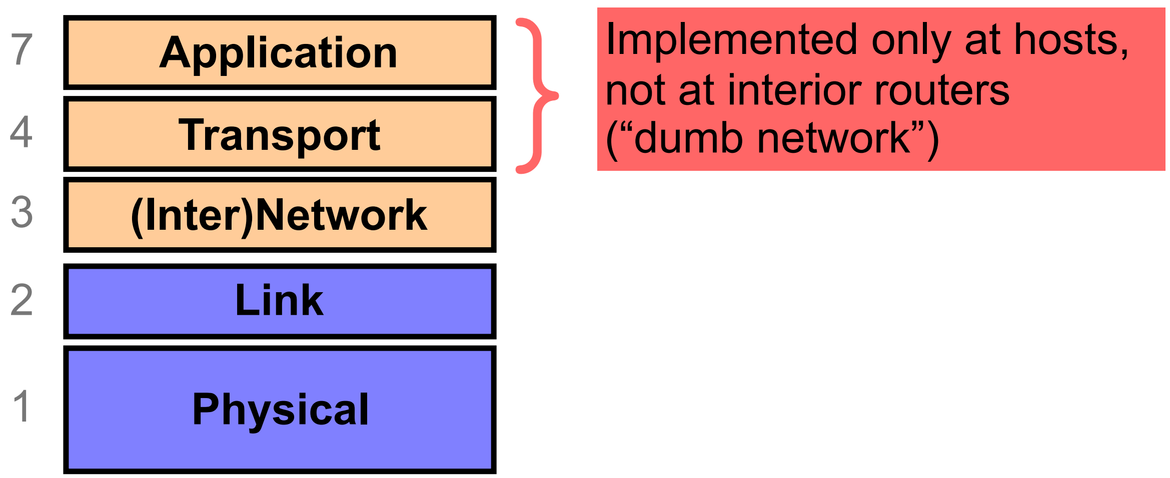
Can use whatever transport(s) is convenient

Freely structured

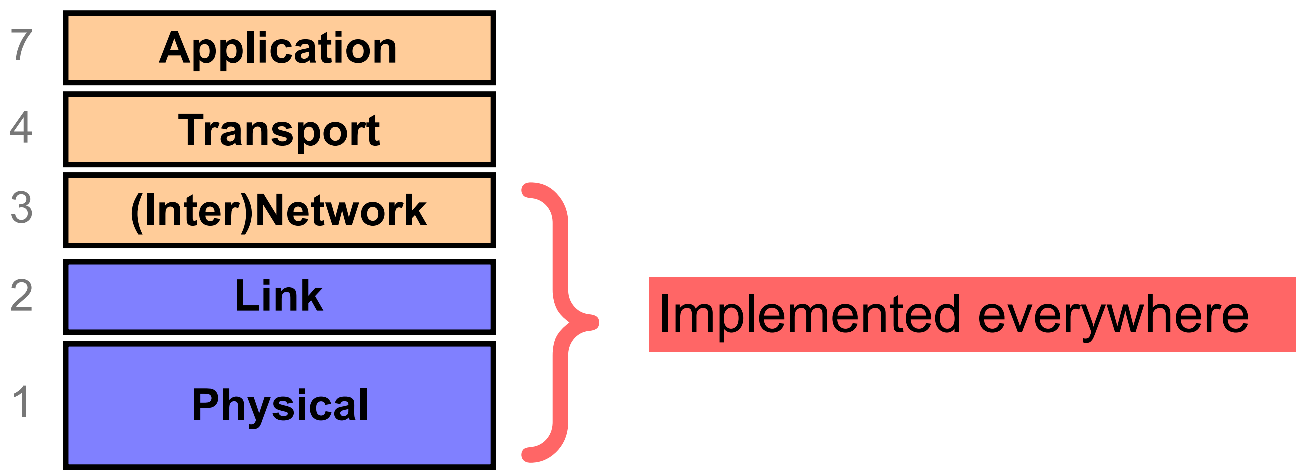
E.g.:

Skype, SMTP (email), HTTP (Web), Halo, BitTorrent

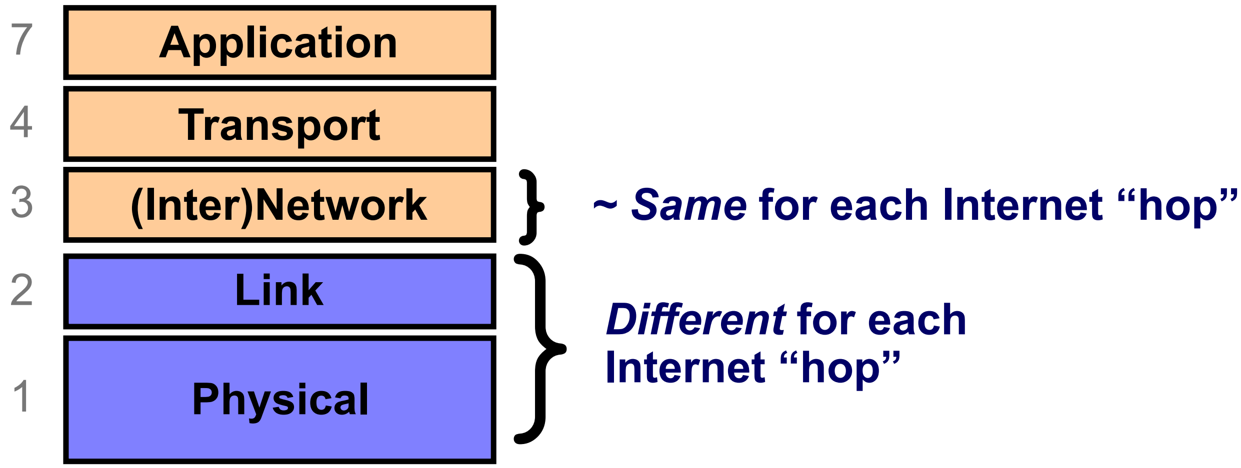
Internet Layering (“Protocol Stack”)



Internet Layering ("Protocol Stack")

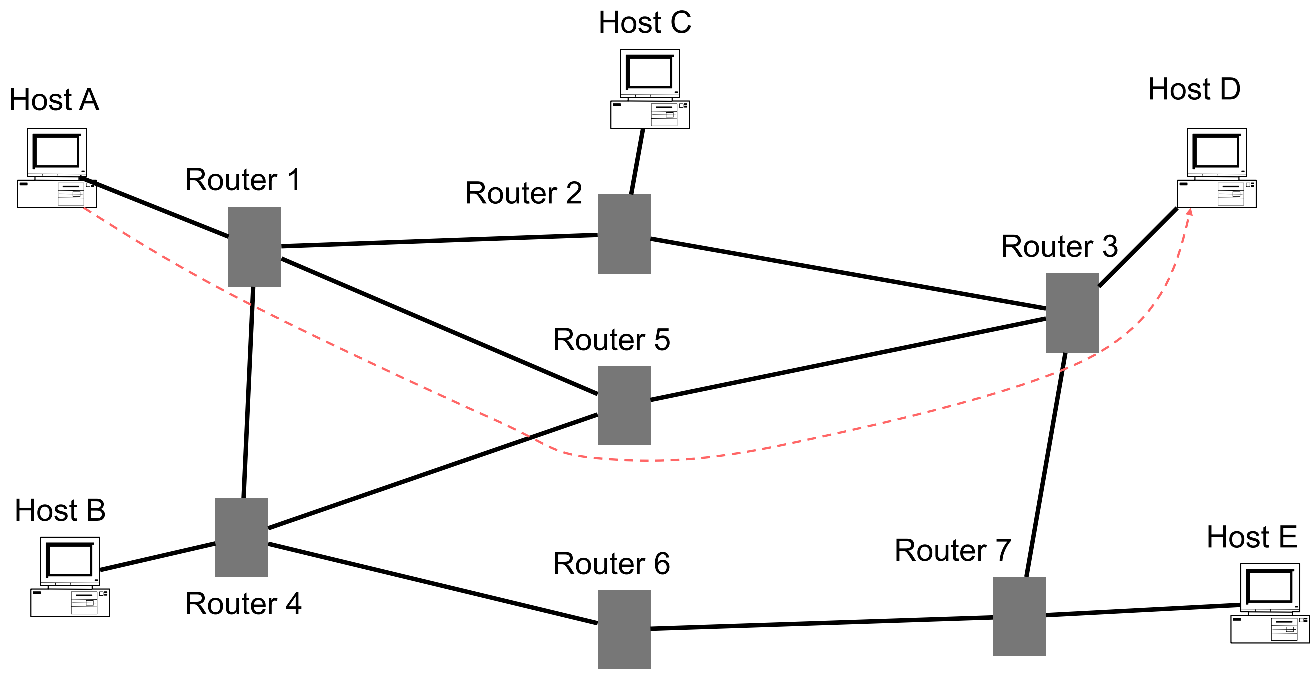


Internet Layering ("Protocol Stack")

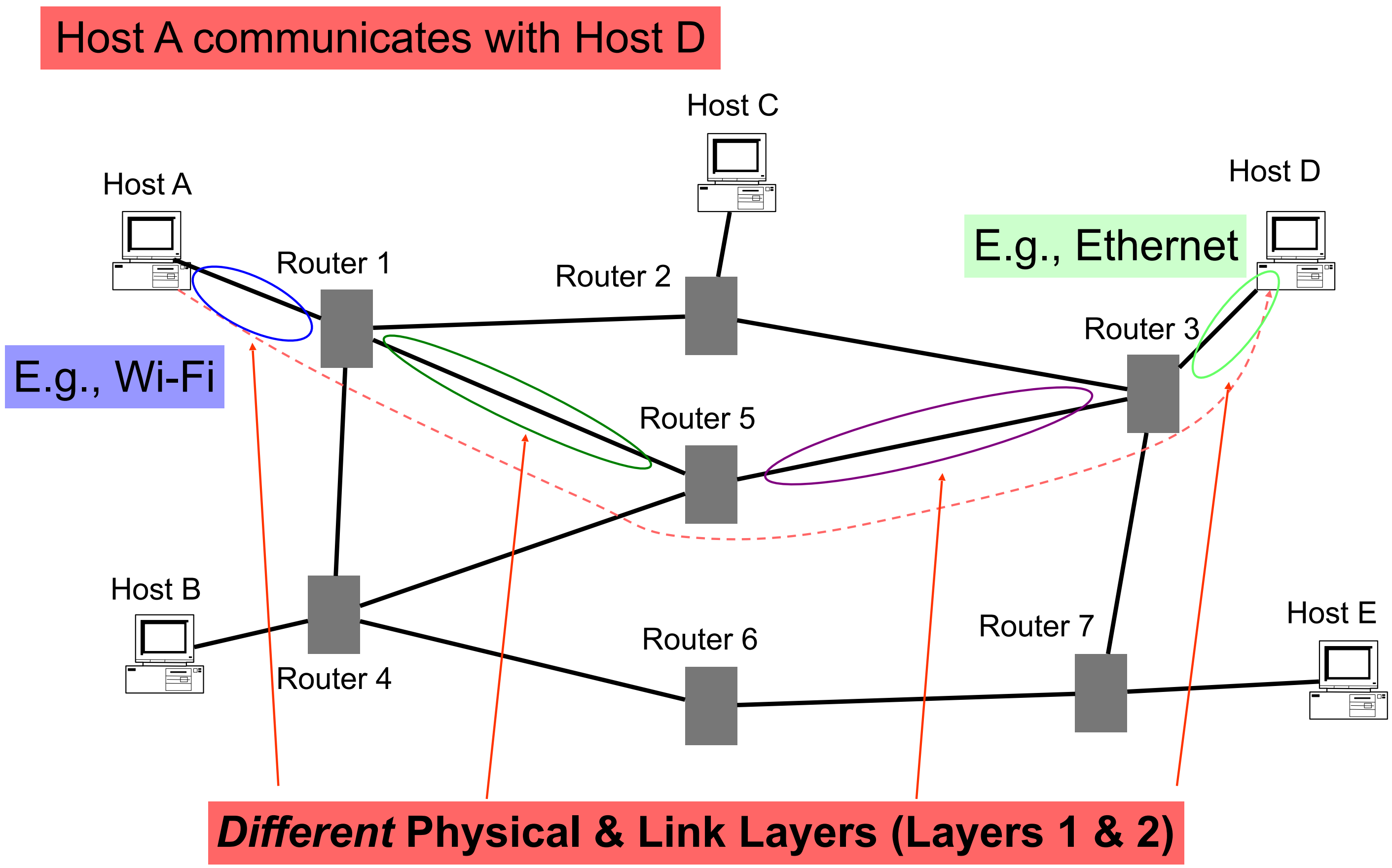


Hop-By-Hop vs. End-to-End Layers

Host A communicates with Host D

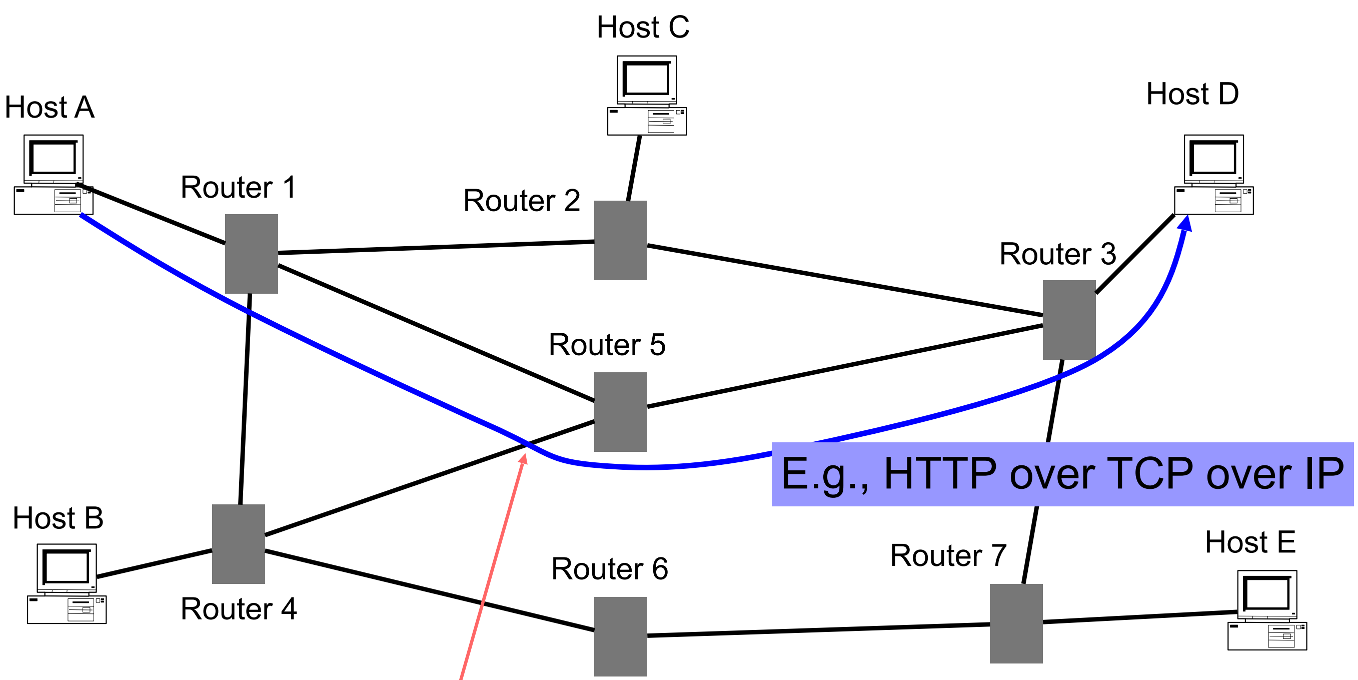


Hop-By-Hop vs. End-to-End Layers



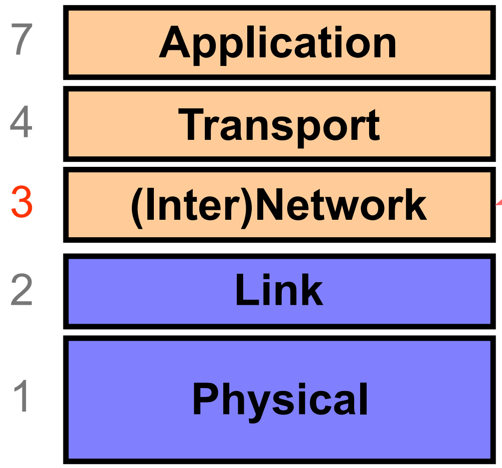
Hop-By-Hop vs. End-to-End Layers

Host A communicates with Host D



Same Network / Transport / Application Layers (3/4/7)
(Routers **ignore** Transport & Application layers)

Layer 3: (Inter)Network Layer *(IP)*

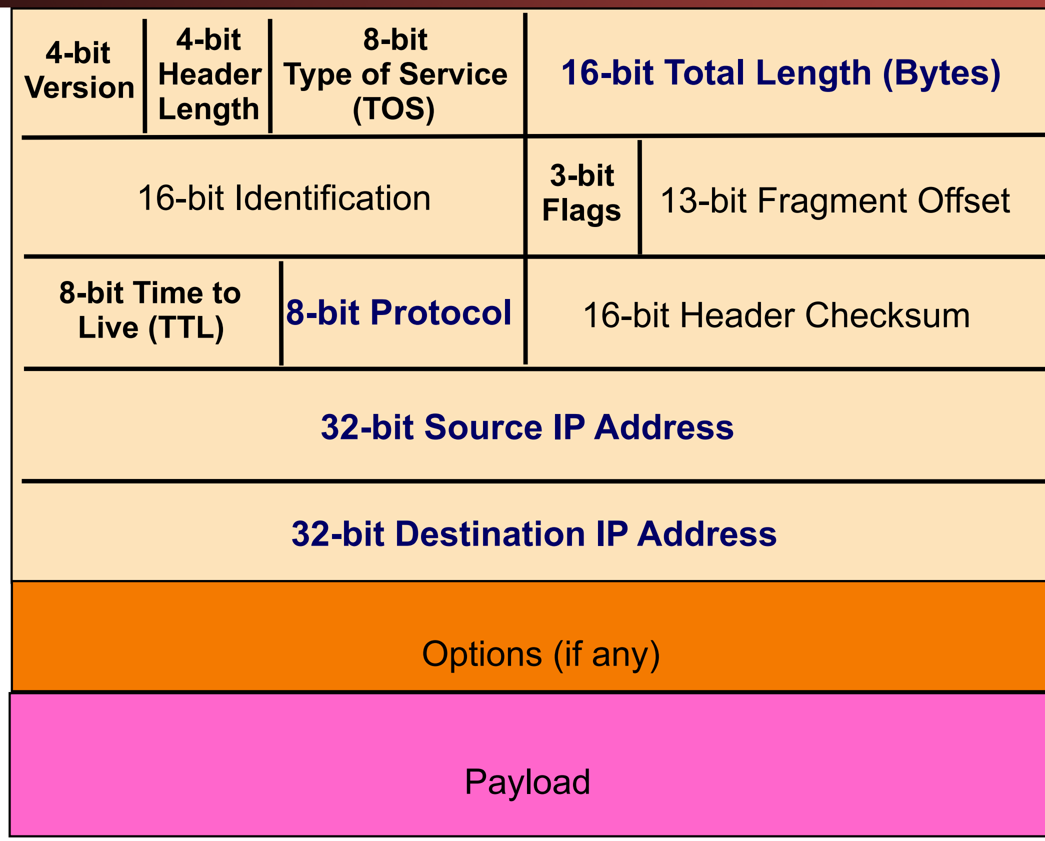


Bridges multiple "subnets" to provide *end-to-end* internet connectivity between nodes

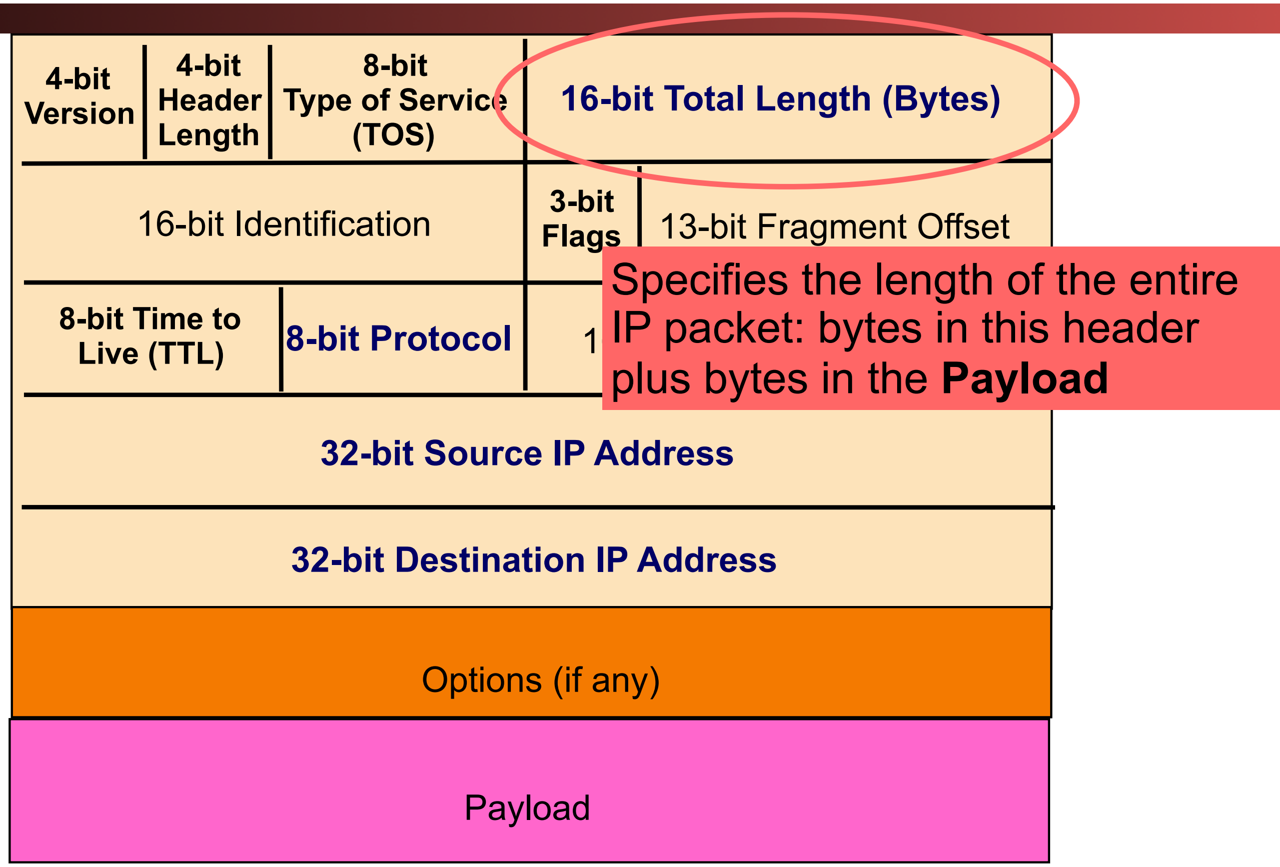
- Provides global addressing

Works across different link technologies

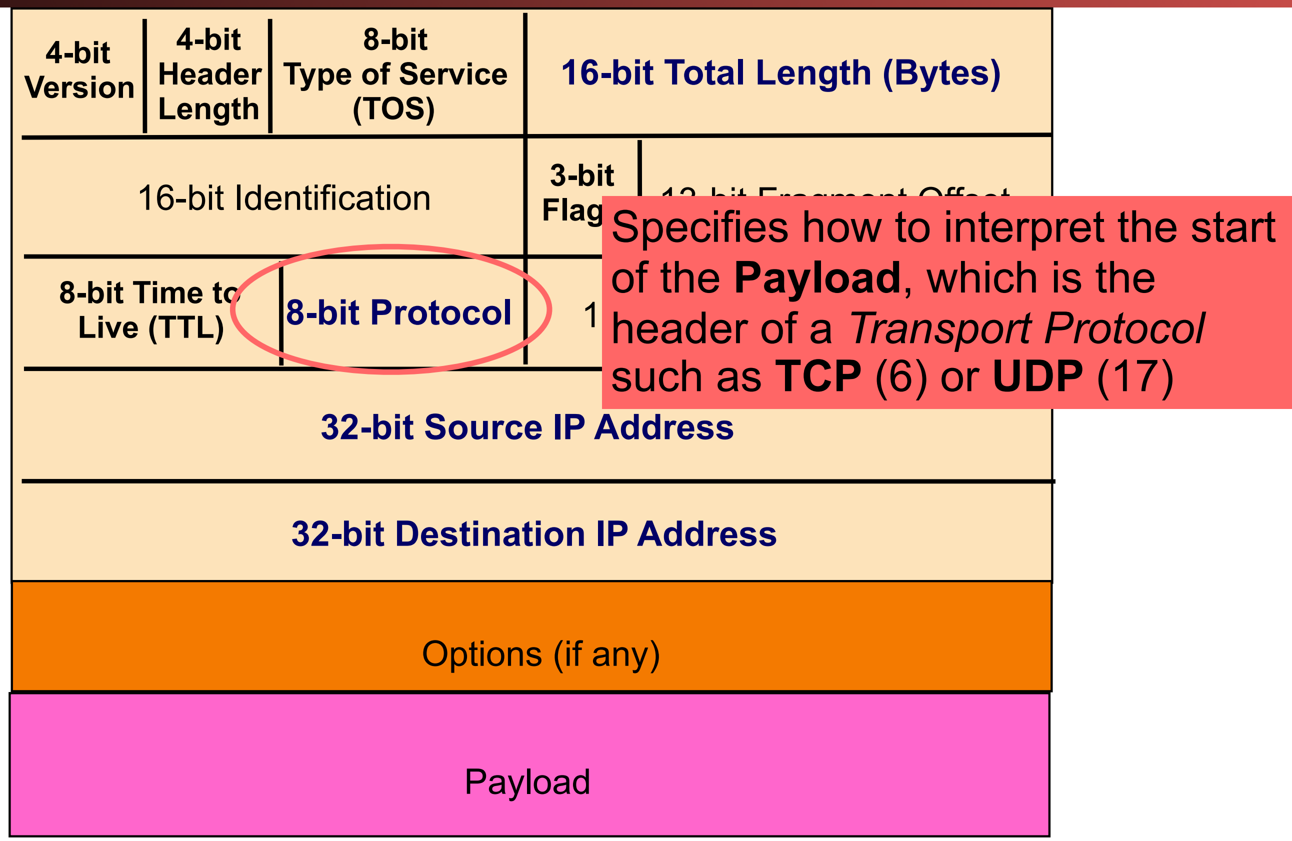
IP Packet Structure



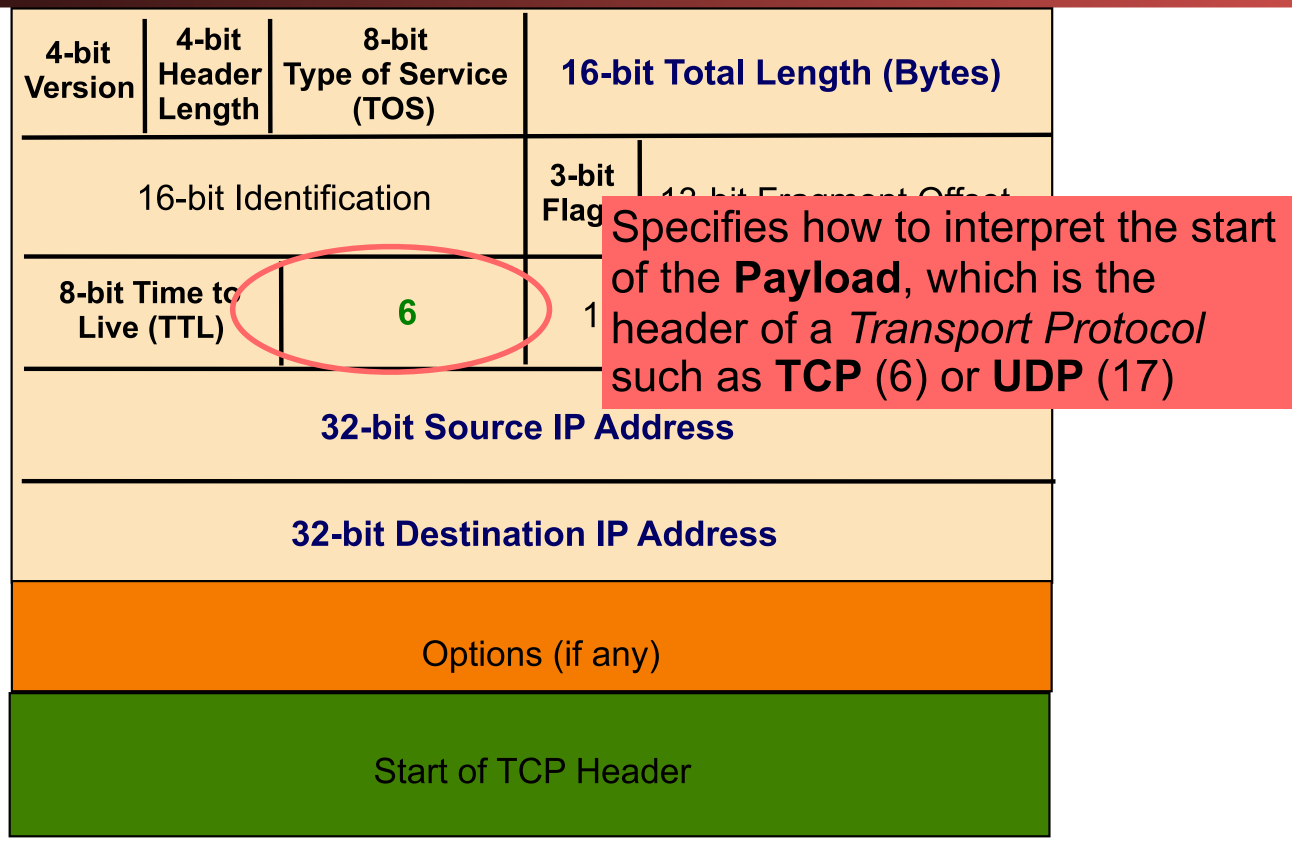
IP Packet Structure



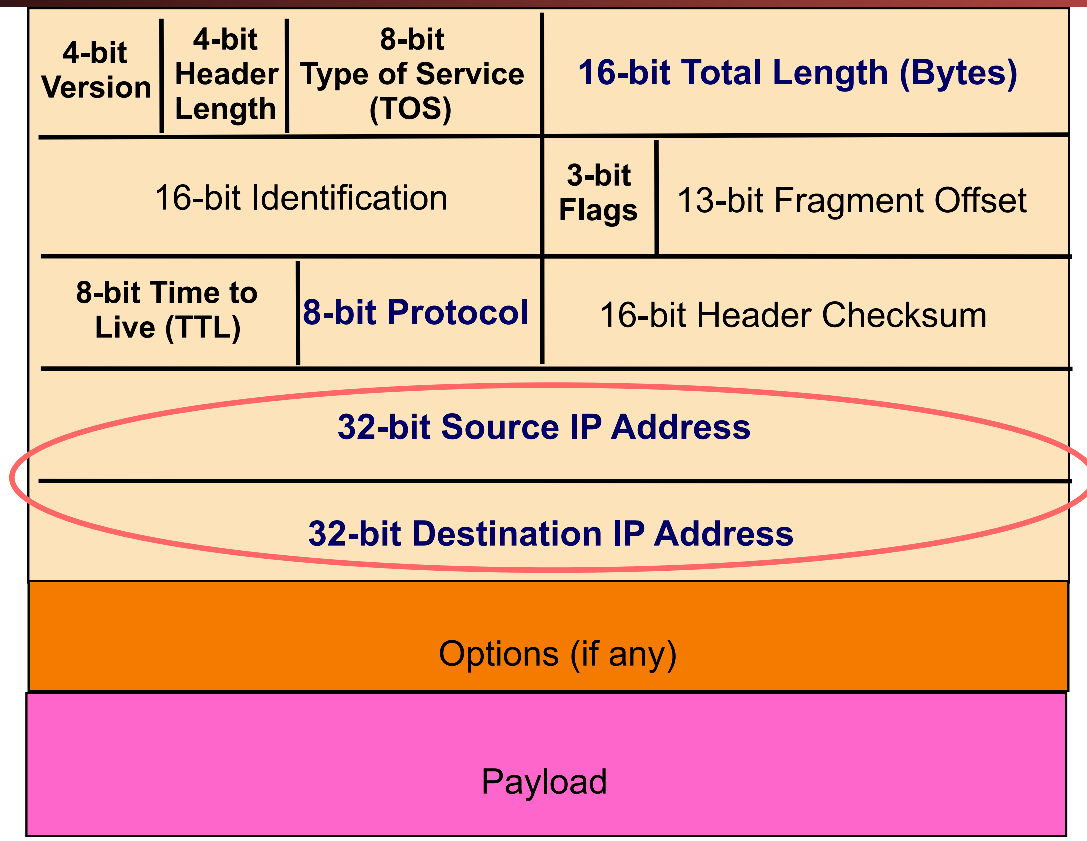
IP Packet Structure



IP Packet Structure



IP Packet Structure



IP Packet Header (Continued)

- Two IP addresses
  - Source IP address (32 bits in main IP version, IPv4)
  - Destination IP address (32 bits, likewise)
- Destination address
  - Unique identifier/locator for the receiving host
  - Allows each node to make forwarding decisions
- Source address
  - Unique identifier/locator for the sending host
  - Recipient can decide whether to accept packet
  - Enables recipient to send reply back to source

The Basic Ethernet Packet: The near-universal Layer 2

- An Ethernet Packet contains:
  - A preamble to synchronize data on the wire
    - We normally ignore this when talking about Ethernet
  - 6 bytes of destination MAC address
    - In this case, MAC means media access control address, not message authentication code!
  - 6 bytes of source MAC address
  - Optional 4-byte VLAN tag
  - 2 bytes length/type field
  - 46-1500B of payload

DST MAC	SRC MAC	VLAN	Type	PAYLOAD

The MAC Address

- The MAC acts as a device identifier
  - The upper 3 bytes are assigned to a manufacturer
    - Can usually identify product with just the MAC address
  - The lower 3 bytes are assigned to a specific device
    - Making the MAC a de-facto serial #
- Usually written as 6 bytes in hex:
  - e.g. `13:37:ca:fe:f0:0d`
- A device ***should ignore*** all packets that aren't to itself or to the broadcast address (`ff:ff:ff:ff:ff:ff`)
  - But almost all devices can go into ***promiscuous mode***
    - This is also known as "sniffing traffic"
- A device generally should only send with its own address
  - But this is enforced with software and can be trivially bypassed when you need to write "raw packets"

The Hub...

- In the old days, Ethernet was simply a shared broadcast medium
  - Every system on the network could hear every sent packet
- Implemented by either a long shared wire or a “hub” which repeated every message to all other systems on the network
  - Thus the only thing preventing every other computer from listening in is simply the network card’s default to ignore anything not directed at it
- The hub or wire is incapable of enforcing sender's MAC addresses
  - Any sender could simply lie about it’s MAC address when constructing a packet

The Hub Yet Lives!

- WiFi is effectively “Ethernet over Wireless”
  - With ***optional*** encryption which we will cover later
- Open wireless networks are just like the old Ethernet hub:
  - Any recipient can hear all the other sender’s traffic
  - Any sender can use any MAC address it desires
- With the added bonus of easy to hijack connections
  - By default, your computer sends out “hey, is anyone here” looking for networks it knows
  - For open networks, anybody can say “Oh, yeah, here I am” and your computer connects to them

Rogue Access Points...

- Since unsecured wireless has no authentication...
  - And since devices by default shout out "hey, is anyone here network X"
- You can create an AP that simply responds with "of course I am"
  - The mana toolkit: https://github.com/sensepost/mana
- Now simply relay the victim's traffic onward
  - And do whatever you want to any unencrypted requests that either happen automatically or when the user actually does something
- I ***suspect*** I've seen this happening around Berkeley
  - Seen an occasional unencrypted version of a password protected network I'd normally use
- Recommendations:
  - Do ***not*** remember unsecured networks
  - Do ***not*** have your computer auto-join open networks

`tcpdump`

- The **`tcpdump`** program allows you to see packets on the network
  - It puts your computer's card into promiscuous mode so it ignores MAC addresses
- You can add additional filters to isolate things
  - EG, only to and from your own IP
  - **`sudo tcpdump -i en0 host {myip}`**
- Note: this is ***wiretapping***
  - DO NOT RUN on a random open wireless network without a filter to limit the traffic you see
  - Only run without filters when connected to your own network
    - But do run it when you get home!

Broadcast is Dangerous: Packet Injection

- If your attacker can see your packets…
  - It isn't just an information leakage
- Instead, an attacker can also ***inject*** their own packets
  - The low level network does not enforce any ***integrity or authenticity***
- So unless the high level protocol uses cryptographic checks…
- The target simply accepts the ***first*** packet it receives as valid!
  - This is a "race condition attack", whichever packet arrives first is accepted

Packet Injection in Action: Airpwn

But Airpwn ain't a joke…

- It is trivial to replace "look for .jpg request and reply with redirect to goatse" with "look for .js request and reply with redirect to exploitive javascript"
  - This JavaScript would start running in the target's web browser, profile the browser, and then use whatever exploits exist
- The requirements for such an attack:
  - The target's traffic must not be encrypted
  - The ability to see the target's traffic
  - The ability to determine that the target's traffic belongs to the target
  - The ability to inject a malicious reply

So Where Does This Occur?

- Open wireless networks
  - E.g. Starbucks, and ***any wireless network without a password***
  - Only safe solution for open wireless is ***only*** use encrypted connections
    - HTTPS/TLS, **`ssh`**, or a Virtual Private Network to a better network
- On backbones controlled by nation-state adversaries!
  - The NSA's super-duper-top-secret attack tool, QUANTUM is ***literally*** airpwn without the goatse!
    - Not an exaggeration: Airpwn only looks at single packets, so does QUANTUM!

It's also *too* easy

- Which is why it isn't an assignment!
- Building it in scapy, a packet library in python:
  - Open a sniffer interface in one thread
    - Pass all packets to a separate work thread so the sniffer doesn't block
  - For the first TCP data packet on any flow destined on port 80
    - Examine the payload with a simple regular expression to see if its a fetch for an image (ends in .jpg or .gif) and not for our own server
      - Afterwards whitelist that flow so you ignore it
  - If so, construct a 302 reply
    - Sending the browser to the target image
  - And create a fake TCP packet in reply
    - Switch the SYN and ACK, ports, and addresses
    - Set the ACK to additionally have the length of the request
    - Inject the reply

Detecting Injected Packets: Race Conditions

- Clients ***can*** detect an injected packet
  - Since they still see the original reply
- Packets can be duplicated, but they should be consistent
  - EG, one version saying “redirect”, the other saying “here is contents” should not occur and represents a ***necessary*** signature of a packet injection attack
- Problem: often detectable too late
  - Since the computer may have acted on the injected packet in a dangerous way before the real reply arrives
- Problem: nobody does this in practice
  - So you don't actually see the detectors work
- Problem: “Paxson’s Law of Internet Measurement”
  - “The Internet is weirder than you think, even when you include the effects of Paxson’s Law of Internet Measurement”
  - Detecting bad on the Internet often ends up inadvertently detecting just odd: Things are always more broken then you think they are

Wireless Ethernet Security Option: WPA2 Pre Shared Key

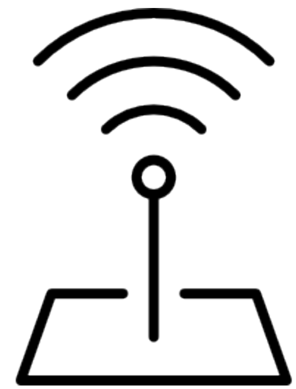
- This is what is used these days when the WiFi is "password protected"
  - The access point and the client have the same pre-shared key (called the PSK key)
  - Goal is to create a shared key called the PTK (Pairwise Transient Key)
- This key is derived from a combination of both the password and the SSID (network name)
  - PSK = PBKDF2(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

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



ANonce

GTK + MIC

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 a brute-force attack
- 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(pw, ssid, 4096, 256)
  - 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
- 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

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

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 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

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