Module M4

CPSC 317

October 12, 2022

Learning Goals

A. DNS

- Define the purpose of DNS.
- Describe the major design problems that needed to be addressed in the design of DNS
- For each design problem, which needed to be addressed, describe the strategy/technology/design approach used to address the problem.
- Compare and contrast the existing DNS implementation to a large scale distributed database.
- Define the purpose of the various servers in the hierarchy of name servers.
- Trace how DNS resolves a name to an IP address
- Interpret the information returned by dig or the information when viewed in wireshark.
- Apply the information returned by dig to determine the next server to contact in DNS lookup or to determine the IP address returned for the lookup
- Be able to describe the different types of resource records returned by a DNS server.

B. Transport

- Explain the main purpose of the transport layer.
- Define multiplexing and de-multiplexing at the transport level (i.e. sockets and ports).
- Understand the type of services that transport supports: reliable delivery, flow control, congestion control, ordered delivery, segmentation, data bundling, non-duplication, multicast, signalling, partial delivery(not UDP/TCP)
- Know descriptions of TCP/UDP as connection-oriented versus connection-less, bidirectional versus unidirectional, stream oriented versus message oriented.
- Compare and contrast the important services provided by UDP and TCP
- Identify applications that (can) make use of TCP (UDP) and explain why
- For a given application explain whether it is better suited for TCP or UDP
- Explain the purpose of the following C function calls: socket(), connect(), bind(), listen(), accept(), sendto(), recvfrom(), read(), write().
- Describe a typical client-server UDP program and TCP program.
- Given a collection of packets and the IP and TCP or UDP headers determine which packets belong to the same connection/stream of data (matching on port and IP) in order to be correctly de-multiplexed.

C. Network Address Translation

- Given an IP header explain its role in NAT.
- Given transport, link, and network headers for an IP network trace how they are used and changed when in an environment that has one one or more NATing routers.
- Trace what happens when a remote machine wants to connect to the server behind the NATing router.
- Identify the short comings of NATing for certain classes of application protocols.
- Explain why the application layer of a home router gets involved in NATing (i.e., ftp: passive versus active mode)

Module M4

CPSC 317

October 12, 2022

Learning Goals

A. DNS

- Define the purpose of DNS.
- Describe the major design problems that needed to be addressed in the design of DNS
- For each design problem, which needed to be addressed, describe the strategy/technology/design approach used to address the problem.
- Compare and contrast the existing DNS implementation to a large scale distributed database.
- Define the purpose of the various servers in the hierarchy of name servers.
- Trace how DNS resolves a name to an IP address
- Interpret the information returned by dig or the information when viewed in wireshark.
- Apply the information returned by dig to determine the next server to contact in DNS lookup or to determine the IP address returned for the lookup
- Be able to describe the different types of resource records returned by a DNS server.

B. Transport

- Explain the main purpose of the transport layer.
- Define multiplexing and de-multiplexing at the transport level (i.e. sockets and ports).
- Understand the type of services that transport supports: reliable delivery, flow control, congestion control, ordered delivery, segmentation, data bundling, non-duplication, multicast, signalling, partial delivery(not UDP/TCP)
- Know descriptions of TCP/UDP as connection-oriented versus connection-less, bidirectional versus unidirectional, stream oriented versus message oriented.
- Compare and contrast the important services provided by UDP and TCP
- Identify applications that (can) make use of TCP (UDP) and explain why
- For a given application explain whether it is better suited for TCP or UDP
- Explain the purpose of the following C function calls: socket(), connect(), bind(), listen(), accept(), sendto(), recvfrom(), read(), write().
- Describe a typical client-server UDP program and TCP program.
- Given a collection of packets and the IP and TCP or UDP headers determine which packets belong to the same connection/stream of data (matching on port and IP) in order to be correctly de-multiplexed.

C. Network Address Translation

- Given an IP header explain its role in NAT.
- Given transport, link, and network headers for an IP network trace how they are used and changed when in an environment that has one one or more NATing routers.
- Trace what happens when a remote machine wants to connect to the server behind the NATing router.
- Identify the short comings of NATing for certain classes of application protocols.
- Explain why the application layer of a home router gets involved in NATing (i.e., ftp: passive versus active mode)

DYNAMIC HOST CONFIGURATION PROTOCOL

DHCP: Dynamic Host Configuration Protocol

goal: allow host to *dynamically* obtain its IP address from network server when it joins network

- can renew its lease on address in use
- allows reuse of addresses (only hold address while connected/"on")
- support for mobile users who want to join network (more shortly)

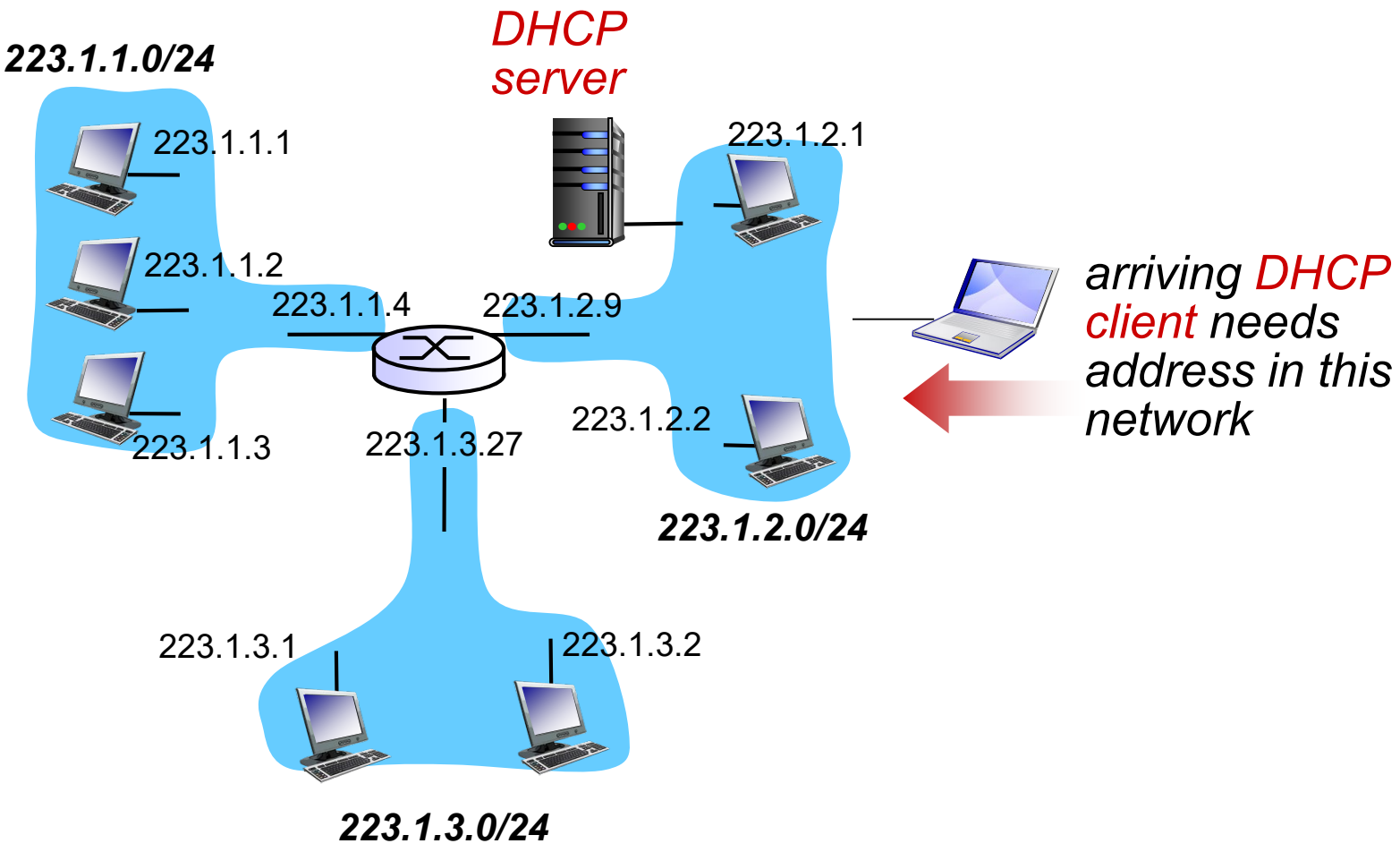
DHCP overview:

- DHCP is built on a client-server model
- DHCP supports several modes of IP-Address allocation

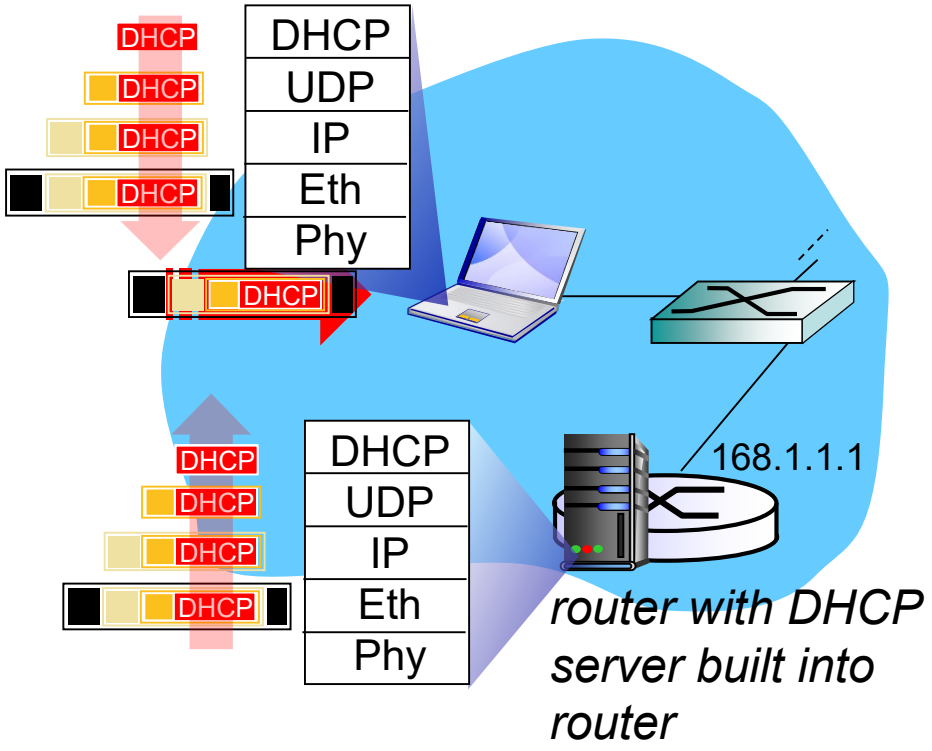
DHCP basic messages:

- host broadcasts "DHCP discover" msg
- DHCP server responds with "DHCP offer" msg
- host requests IP address: "DHCP request" msg
- DHCP server sends address: "DHCP ack" msg

DHCP client-server scenario

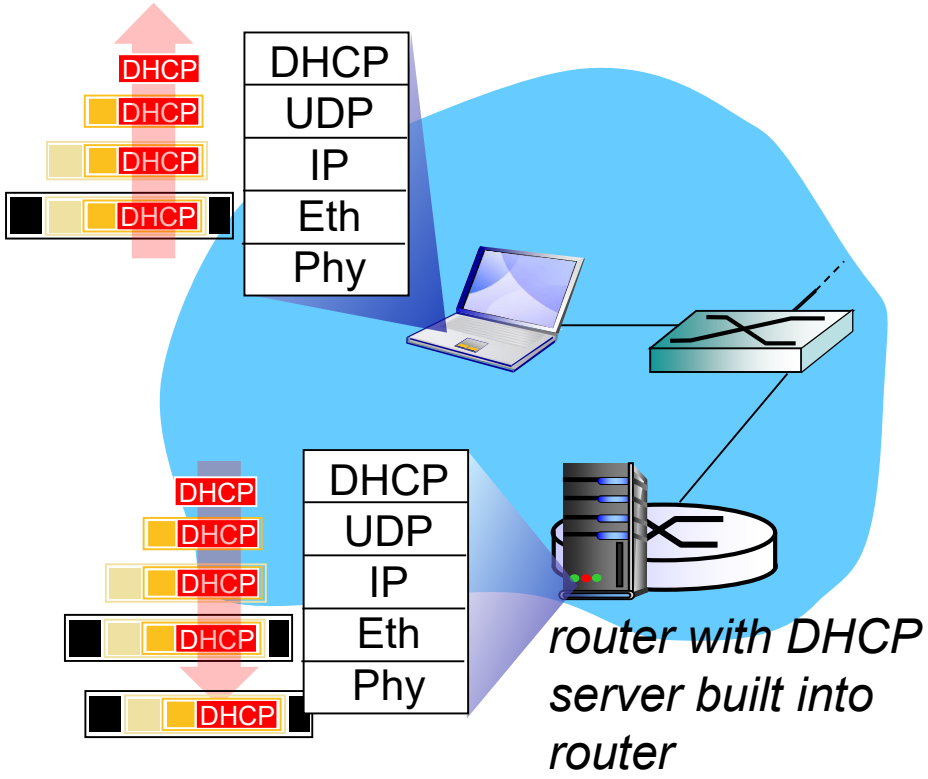


DHCP packet to server



- connecting laptop needs its IP address, address of first-hop router, network prefix, address of DNS server
- DHCP request encapsulated in UDP, encapsulated in IP, encapsulated in 802.3 Ethernet
- Ethernet frame broadcast (dest: FFFFFFFFFFFF) on LAN, received at router running DHCP server
- Ethernet de-muxed to IP de-muxed, UDP de-muxed to DHCP

DHCP message from server

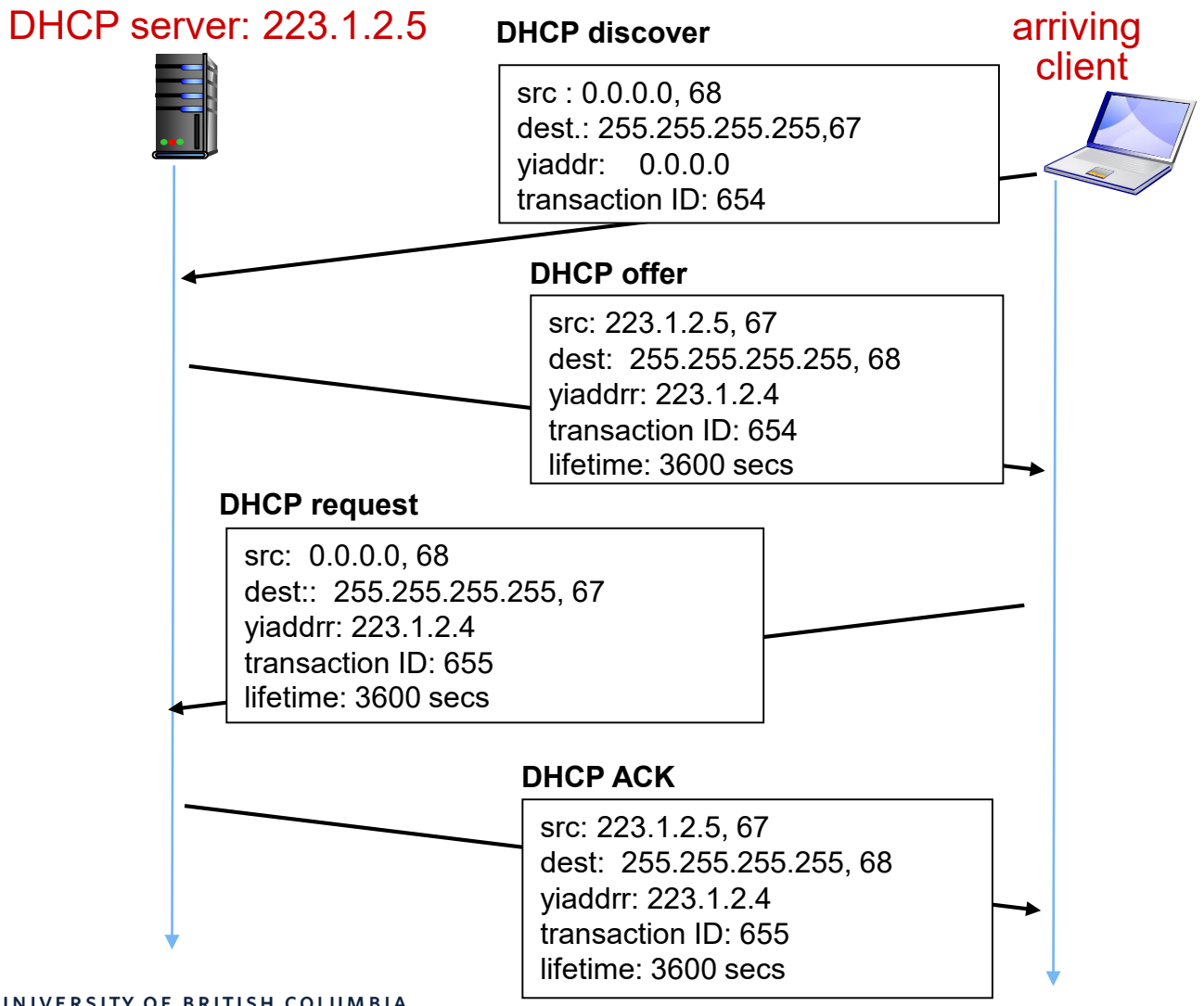


- DCP server formulates DHCP ACK containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server

- encapsulation of DHCP server, frame forwarded to client, demuxing up to DHCP at client

- client now knows its IP address, name and IP address of DSN server, IP address of its first-hop router

DHCP client-server scenario



DHCP: more than IP addresses

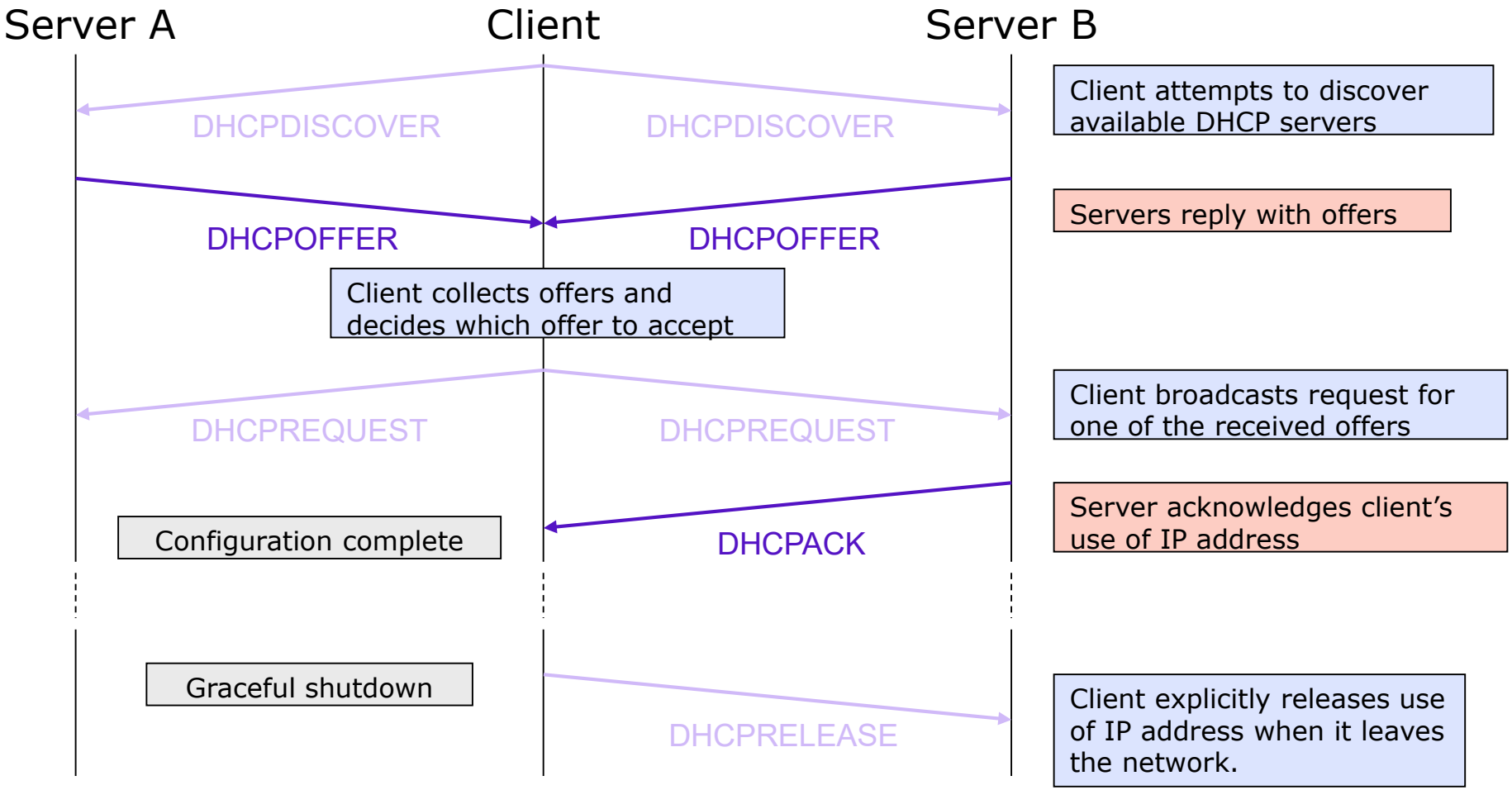
DHCP can return more than just allocated IP address on network:

- address of first-hop router for client
- name and IP address of DNS server
- network mask (indicating network versus host portion of address, network prefix)

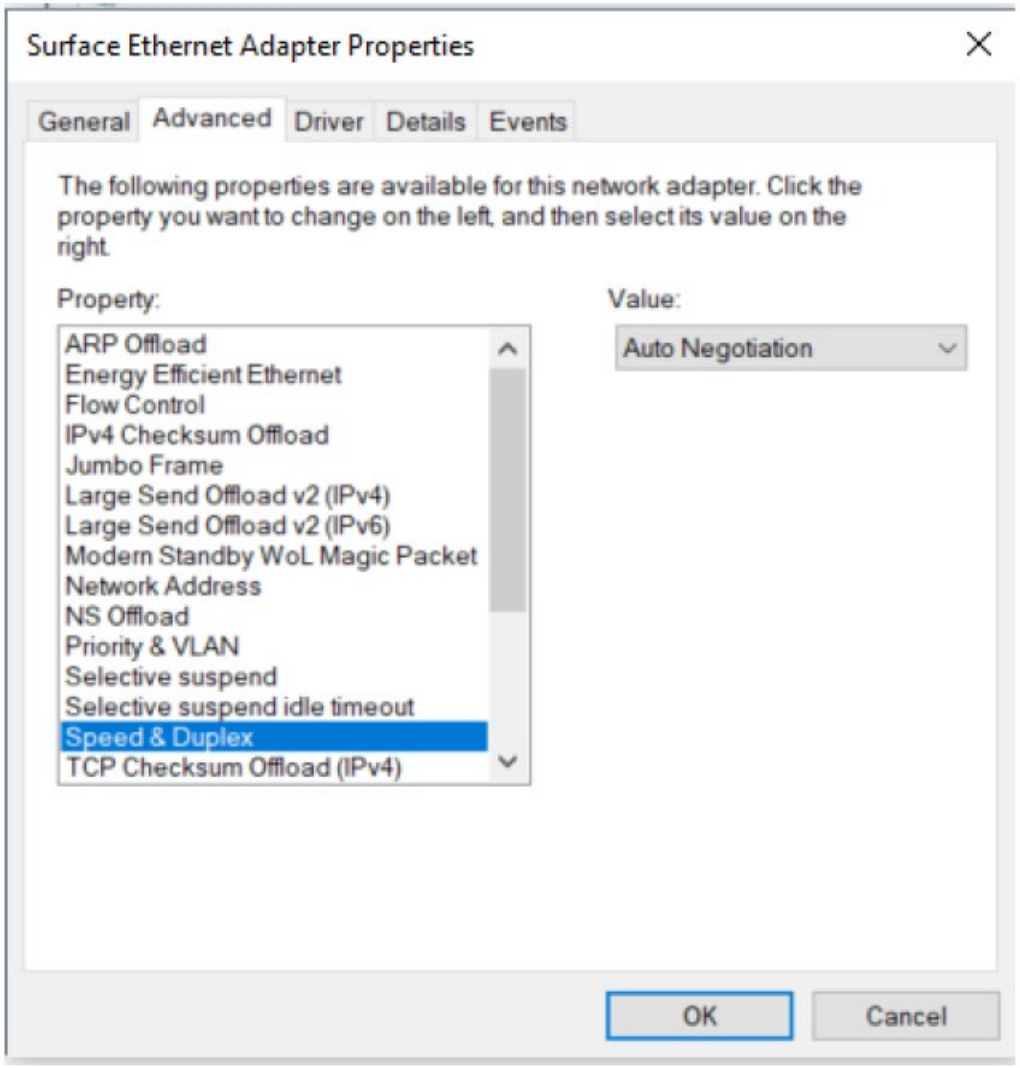
Why

- Why use leases?
- What if two machines request an IP at the same time?
- What if there are more than one servers?

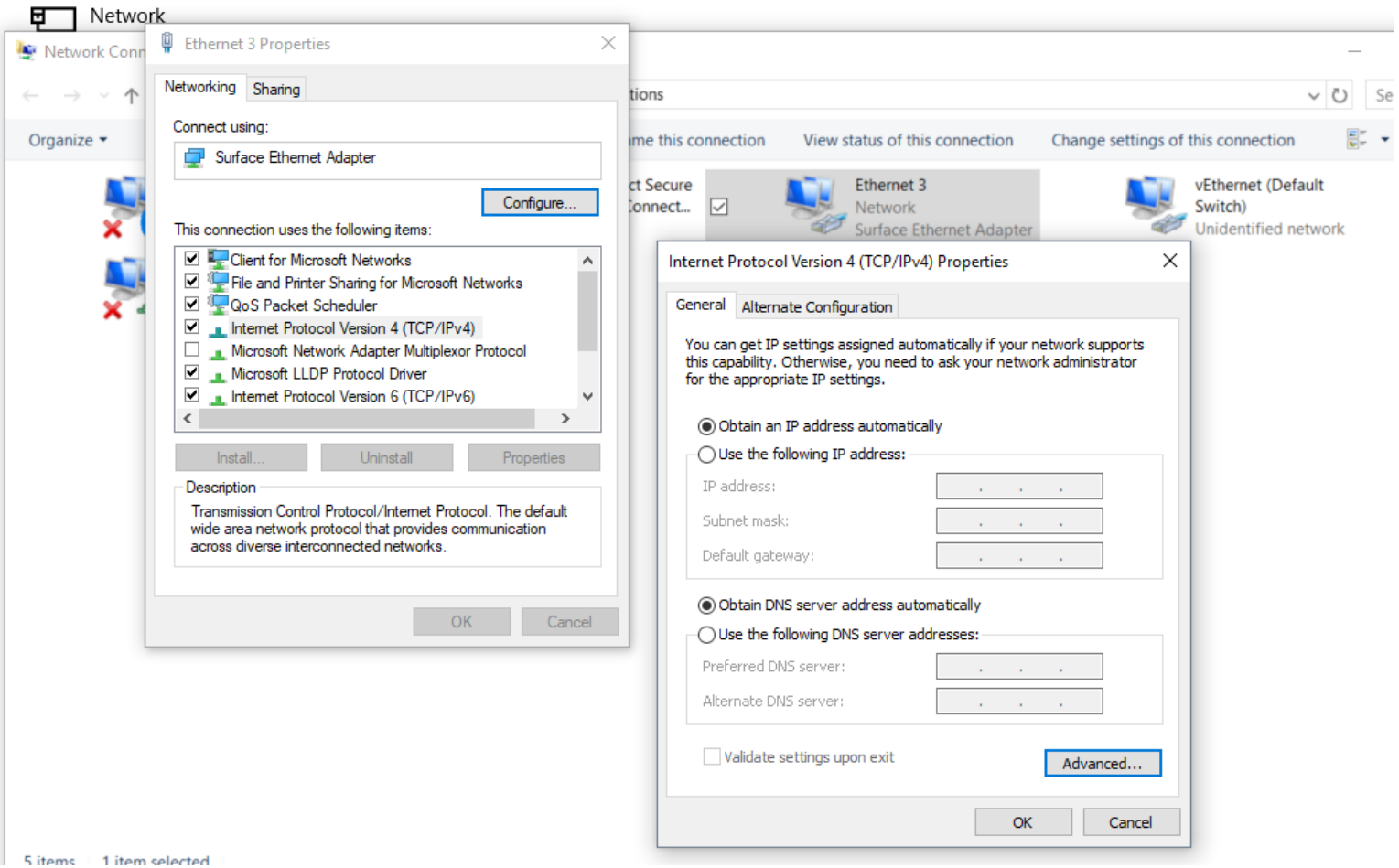
DHCP Conversation (two servers)



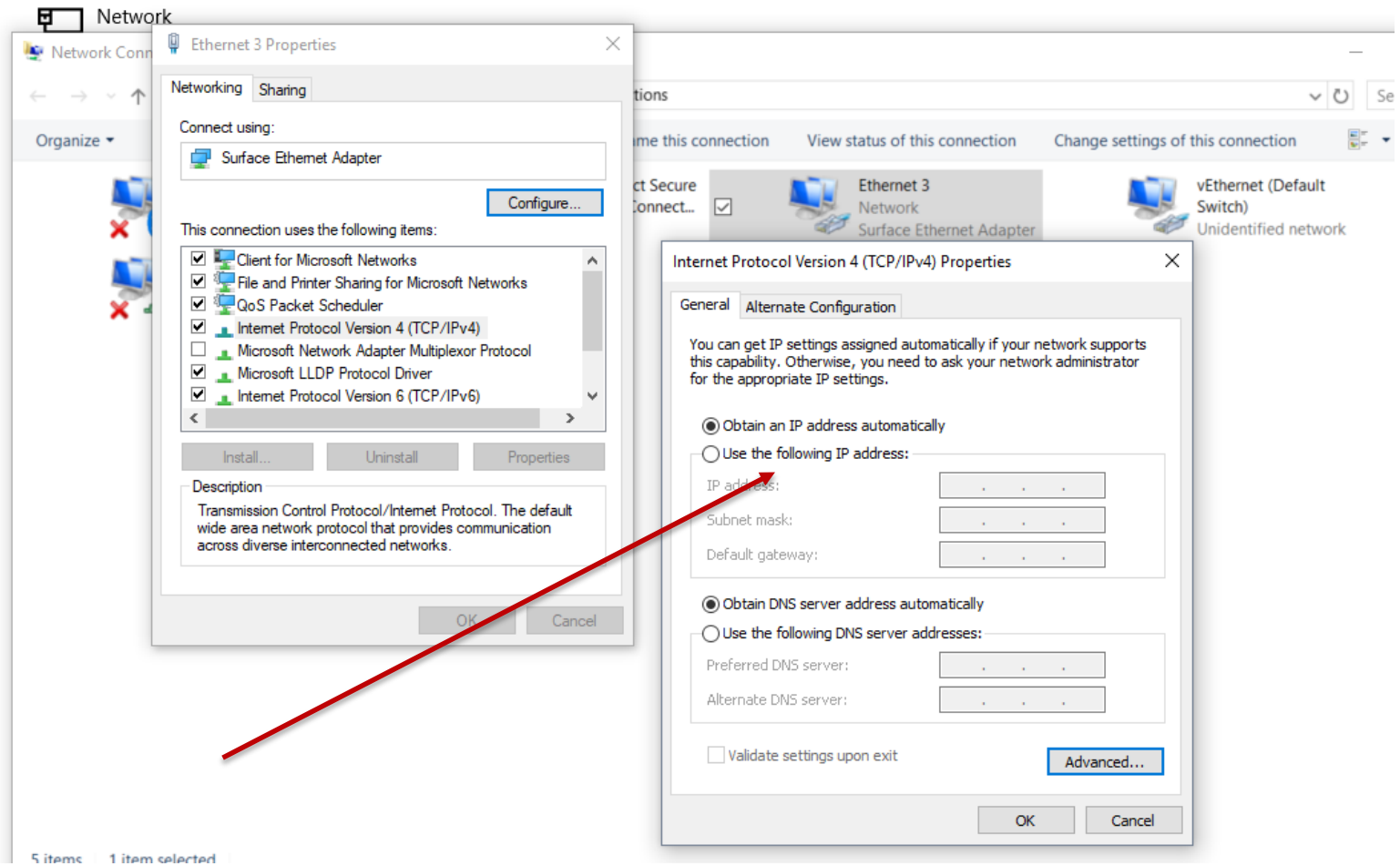
Configuring the Adapter



Configuring the Host



Configuring the Host

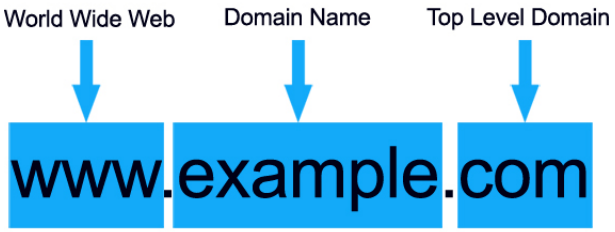


Plug and Play

DOMAIN NAME TRANSLATION

Domain Name

- A **domain name** is an identification string that defines a realm of administrative autonomy, authority or control within the Internet.
- Domain names are formed by the rules and procedures of the Domain Name System (DNS).

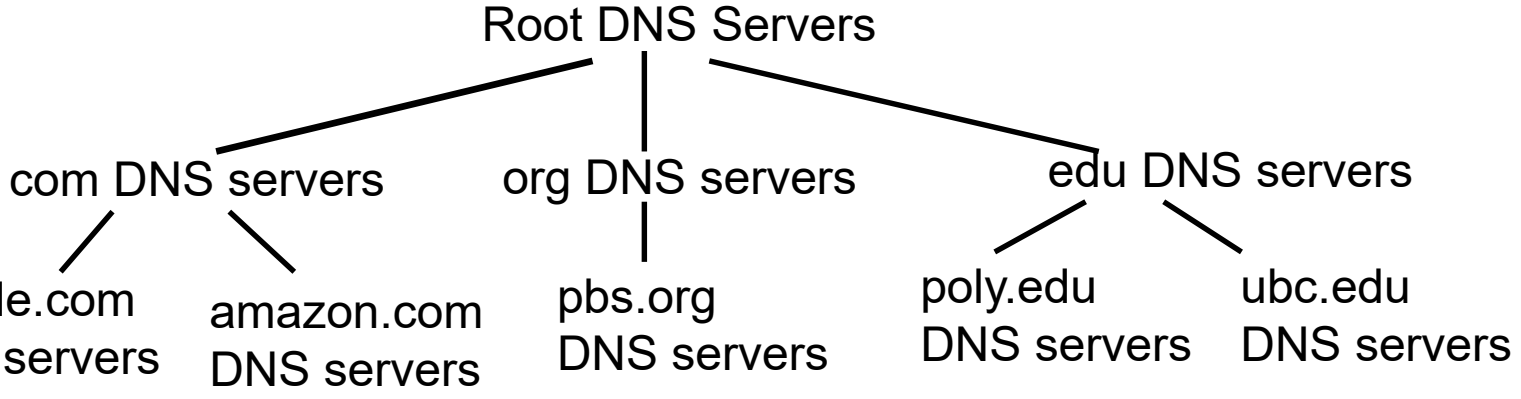


WHAT PROBLEM IS DNS SOLVING?

What is it?

- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol* host, routers, name servers to communicate to *resolve* names (address/name translation)
  - note: core Internet function, implemented as *application-layer* protocol
  - complexity at network's "edge"

DNS: Root name servers



13 root name servers worldwide

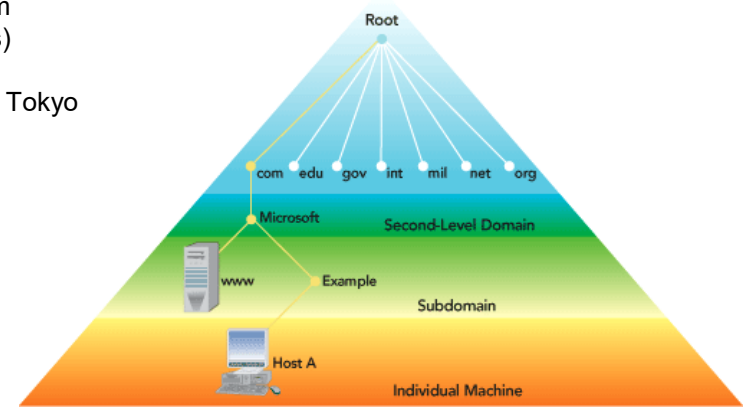
a Verisign, Dulles, VA
c Cogent, Herndon, VA (also Los Angeles)
d U Maryland College Park, MD
g US DoD Vienna, VA
h ARL Aberdeen, MD
j Verisign, (11 locations)

k RIPE London (also Amsterdam, Frankfurt)
l Autonomica, Stockholm (plus 3 other locations)

m WIDE Tokyo

e NASA Mt View, CA
f Internet Software C. Palo Alto, CA (and 17 other locations)

b USC-ISI Marina del Rey, CA
l ICANN Los Angeles, CA



TLD and Authoritative Servers

- Top-level domain (TLD) servers: responsible for com, org, net, edu, etc, and all top-level country domains uk, fr, ca, jp.
  - Network solutions maintains servers for com TLD
  - Educause for edu TLD
- Authoritative DNS servers: organization's DNS servers, providing authoritative hostname to IP mappings for organization's servers (e.g., Web and mail).
  - Can be maintained by organization or service provider

SIMPLE USE OF DNS

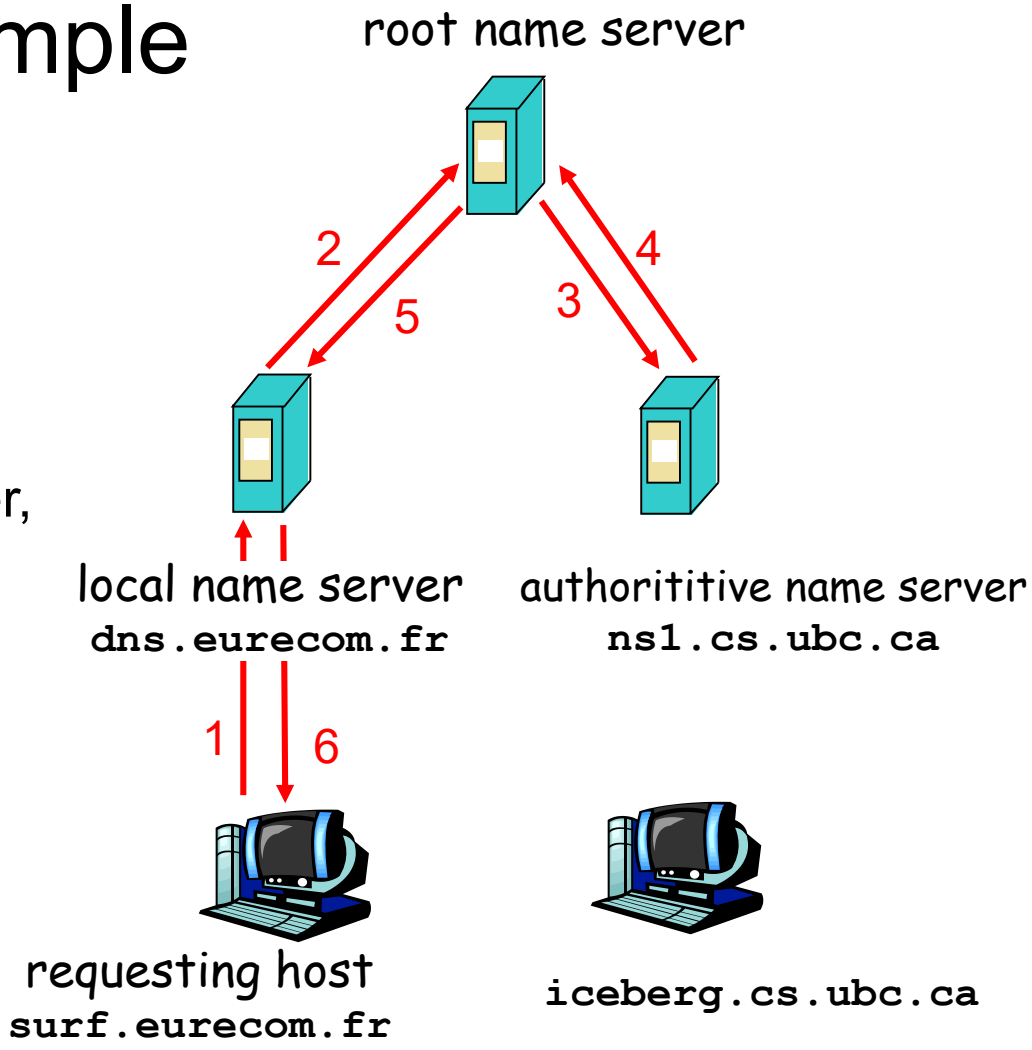
Local Name Server

- Does not strictly belong to hierarchy
- Each ISP (residential ISP, company, university) has one.
  - Also called “default name server”
- When a host makes a DNS query, query is sent to its local DNS server
  - Acts as a proxy, forwards query into hierarchy.
- Queries can be iterative or recursive

Simple DNS example

host **`surf.eurecom.fr`**
wants IP address of
`iceberg.cs.ubc.ca`

1. contacts its local DNS server, **`dns.eurecom.fr`**
2. **`dns.eurecom.fr`** contacts root name server
3. root name server contacts authoritative name server, **`ns1.cs.ubc.ca`**



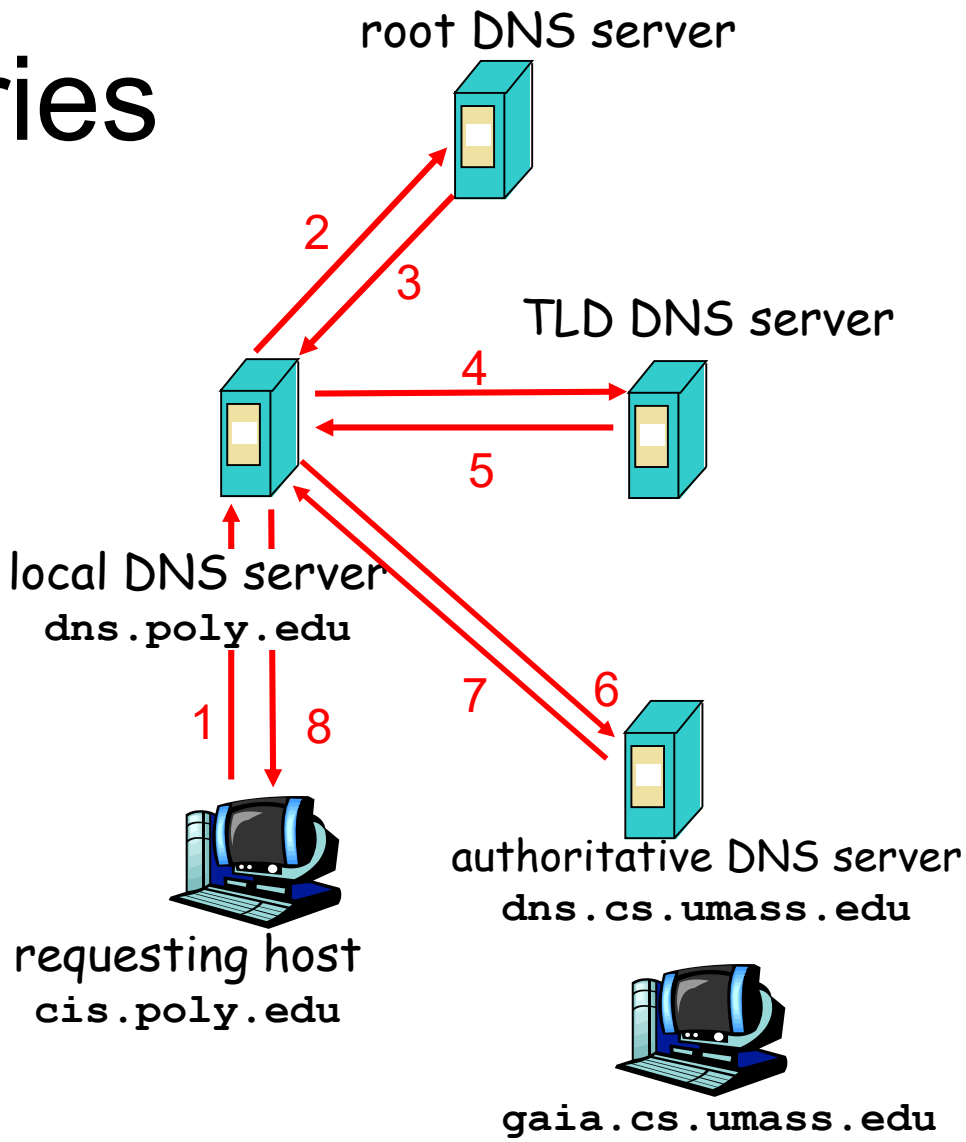
http://www.dnsreport.com/

Iterative Queries

- Host at cis.poly.edu wants IP address for gaia.cs.umass.edu

<u>iterated query:</u>

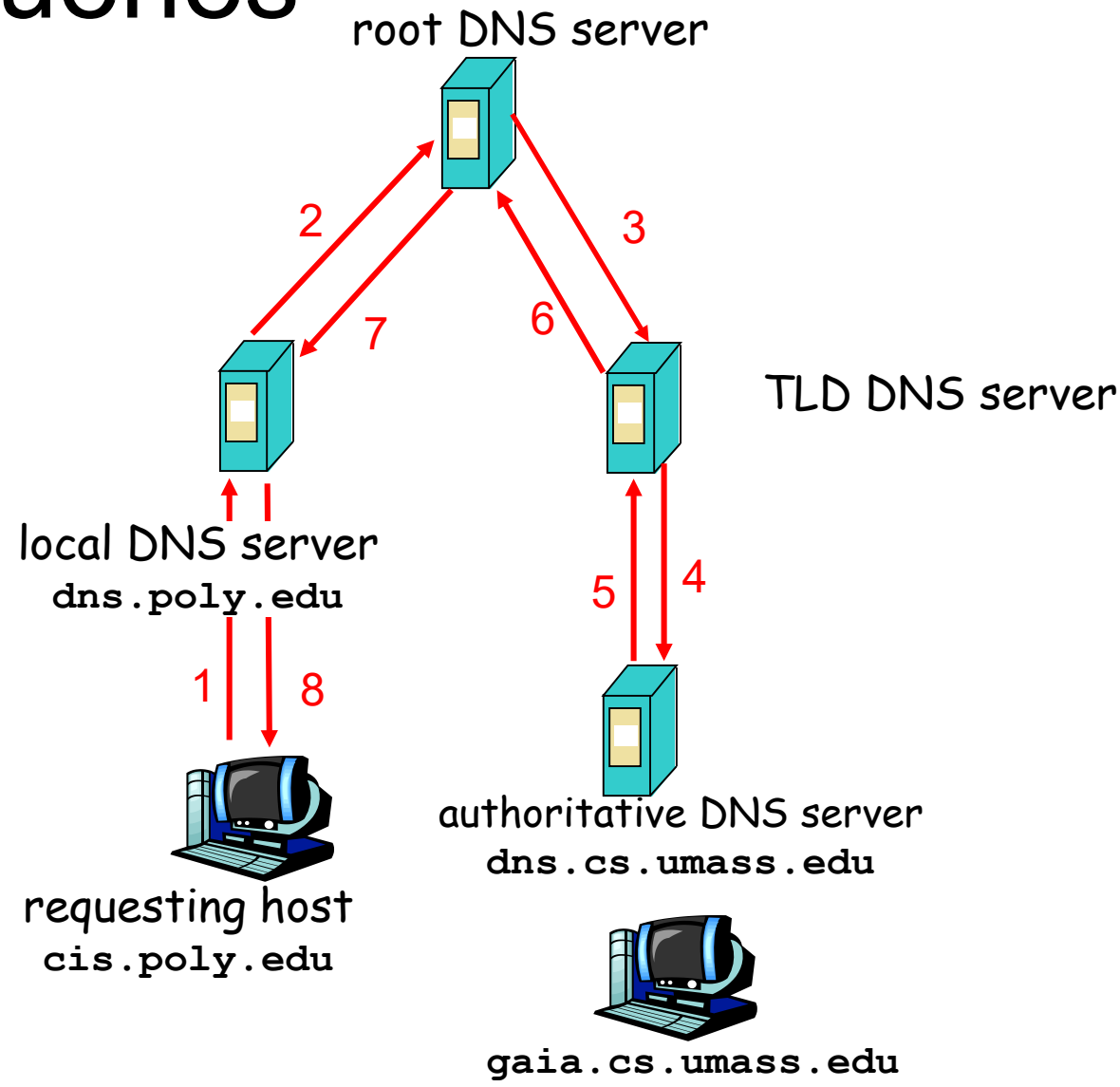
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



Recursive queries

recursive query:

- puts burden of name resolution on contacted name server
- heavy load? But can cache the results



DNS Caching

- How DNS caching works
  - DNS servers cache responses to queries
  - Responses include a "time to live" (TTL) field
  - Server deletes cached entry after TTL expires

- Why caching is effective
  - The top-level servers rarely change
  - Popular sites visited often → local DNS server often has the information cached

Who Knows What?

- Every server knows address of root name server
- Root servers know the address of all TLD servers
- Every node knows the address of children
- An ***authoritative*** DNS server stores name-to-address mappings ("resource records") for all DNS names in the domain that it has authority for
- Therefore, each server:
  - Stores only a subset of the total DNS database (scalable!)
  - Can discover server(s) for any portion of the hierarchy

Benefits of This Approach

- Scalable in names, updates, lookups, users
- Highly available: domains replicate independently
- Extensible: can add TLDs just by changing root database
- Autonomous administration:
  - Each domain manages its own names and servers
  - Can further delegate
  - Can ensure uniqueness of names
  - Can maintain consistency of databases

DNS Goals

- Scaling (names, users, updates, etc.)
  - Yes
- Ease of management (uniqueness of names, etc.)
  - Yes
- Availability and consistency and security
  - Yes
- Performant -- fast lookups
  - ??

DETAILS

DNS Records

- DNS servers store resource records (RRs)
  - RR is (name, value, type, TTL)

- Type = A: *(→ Address)*
  - name = hostname
  - value = IP address
  - (www.cs.ubc.ca, 142.103.6.5, A, TTL)

- Type = NS: *(→ Name Server)*
  - name = domain
  - value = name of dns server for domain
  - (cs.ubc.ca, fs1.ugrad.cs.ubc.ca,NS,TTL)

DNS Records (cont'd)

- Type = MX: *(→ Mail eXchanger)*
  - name = domain in email address
  - value = name(s) of mail server(s)
  - (cs.ubc.ca, mx2.cs.ubc.ca, MX, TTL)
- Type = CNAME: *(→ Canonical NAME)*
  - Name = alias
  - Value is "canonical" name
  - (foo.com, relay1.bar.foo.com, CNAME, TTL) Canonical name relay1.bar.foo.com for alias foo.com
- Type = PTR: *(→ Pointer)*
  - name is reversed IP
  - value is corresponding hostname

DNS Protocol

- Query and Reply messages
  - Both with the same message format
- Client-Server interaction on UDP Port 53
  - Spec supports TCP too, but not always implemented
  - Reliability via repeating requests on timeout
- Resolution is almost always “iterative”

DNS protocol, messages

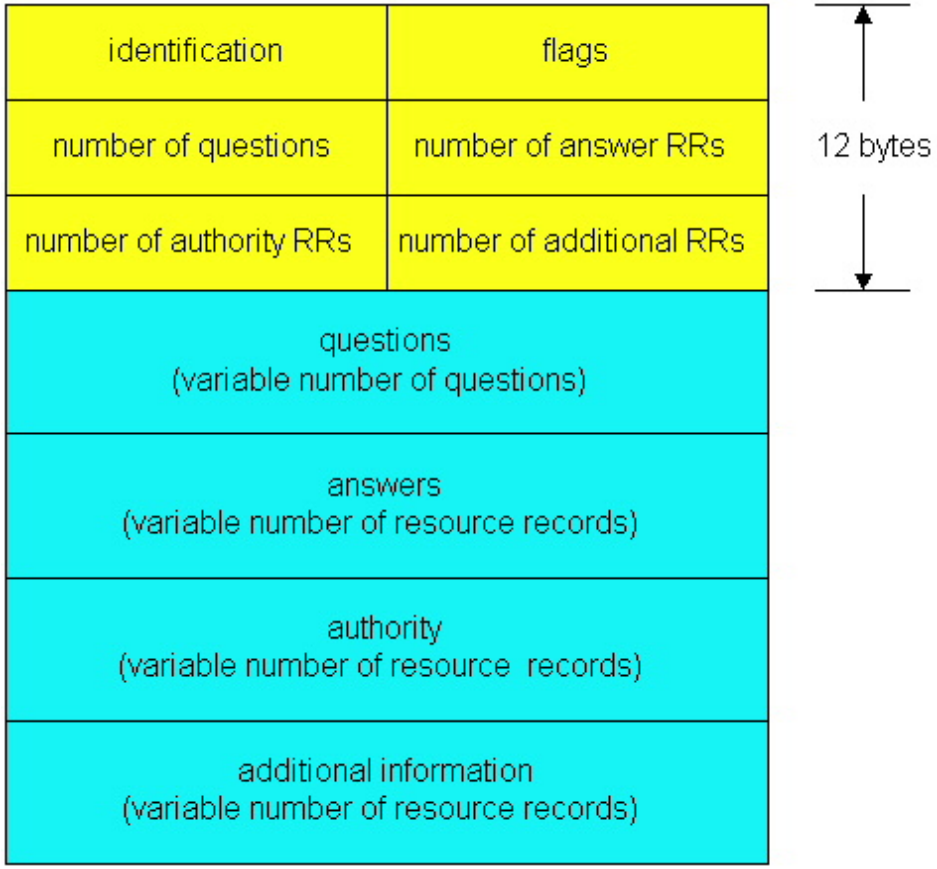
DNS protocol : *query* and *reply* messages, both with same *message format*

msg header

identification: 16 bit # for query, reply to query uses same #

flags:

- query or reply
- recursion desired
- recursion available
- reply is authoritative



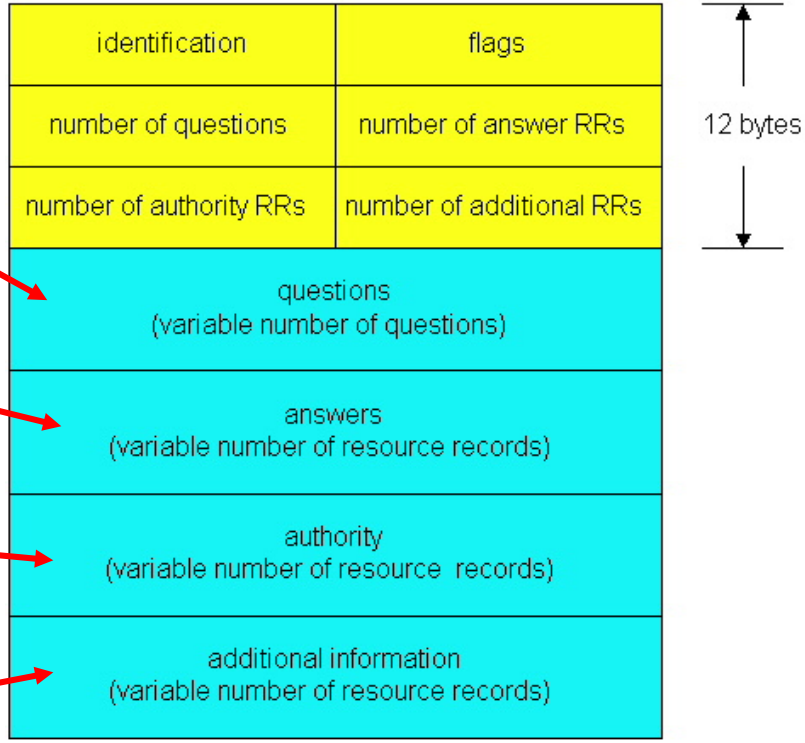
DNS protocol, messages

Name, type fields for a query

Resource Records (RRs) in response to query

Resource records for authoritative servers

Records additional "helpful" info that may be used



Resource Records (RR) returned

- **Questions** are always Name, Type, Class tuples. For Internet applications, the Class is IN, the Type is a valid RR type, and the Name is a fully-qualified domain name, stored in a standard format. Names can't be wildcarded, but Types and Classes can be. In addition, special Types exist to wildcard mail records and to trigger zone transfers. The question is the only section included in a query message; the remaining sections being used for replies.
- **Answers** are RRs that match the Name, Type, Class tuple. If any of the matching records are CNAME pointers leading to other records, the target records should also be included in the answer. There may be multiple answers, since there may be multiple RRs with the same labels.
- **Authority** RRs are type NS records pointing to name servers closer to the target name in the naming hierarchy. This field is completely optional, but clients are encouraged to cache this information if further requests may be made in the same name hierarchy.
- **Additional** RRs are records that the name server believes may be useful to the client. The most common use for this field is to supply A (address) records for the name servers listed in the Authority section.

Query to google.com

```
> Queries
> Answers
v Authoritative nameservers
  > google.com: type NS, class IN, ns ns1.google.com
  > google.com: type NS, class IN, ns ns4.google.com
  > google.com: type NS, class IN, ns ns2.google.com
  > google.com: type NS, class IN, ns ns3.google.com
v Additional records
  > ns2.google.com: type A, class IN, addr 216.239.34.10
  > ns1.google.com: type A, class IN, addr 216.239.32.10
  > ns3.google.com: type A, class IN, addr 216.239.36.10
  > ns4.google.com: type A, class IN, addr 216.239.38.10
  > ns2.google.com: type AAAA, class IN, addr 2001:4860:4802:34::a
  > ns1.google.com: type AAAA, class IN, addr 2001:4860:4802:32::a
  > ns3.google.com: type AAAA, class IN, addr 2001:4860:4802:36::a
  > ns4.google.com: type AAAA, class IN, addr 2001:4860:4802:38::a
  [Request In: 228]
  [Time: 0.002698000 seconds]
```

```
000  00 05 9a 3c 7a 00 00 11  22 33 44 55 08 00 45 00   ···<z··· "3DU··E·
010  01 44 15 4b 00 00 3a 11  34 32 89 52 01 02 8e 67   ·D·K··:· 42·R···g
020  1d 71 00 35 f2 92 01 30  1b af e1 7b 81 80 00 01   ·q·5···0 ···{····
030  00 01 00 04 00 08 03 77  77 77 06 67 6f 6f 67 6c   ·······w ww·googl
040  65 03 63 6f 6d 00 00 01  00 01 c0 0c 00 01 00 01   e·com··· ········
050  00 00 00 3a 00 04 d8 3a  c1 44 c0 10 00 02 00 01   ···:···: ·D······
060  00 02 cb 0f 00 06 03 6e  73 31 c0 10 c0 10 00 02   ·······n s1······
070  00 01 00 02 cb 0f 00 06  03 6e 73 34 c0 10 c0 10   ········ ·ns4····
080  00 02 00 01 00 02 cb 0f  00 06 03 6e 73 32 c0 10   ········ ···ns2··
090  c0 10 00 02 00 01 00 02  cb 0f 00 06 03 6e 73 33   ········ ·····ns3
0a0  c0 10 c0 60 00 01 00 01  00 00 66 1f 00 04 d8 ef   ···`···· ··f·····
0b0  22 0a c0 3c 00 01 00 01  00 00 9c 5a 00 04 d8 ef   "··<···· ···Z····
0c0  20 0a c0 72 00 01 00 01  00 03 3d e5 00 04 d8 ef    ··r···· ··=·····
0d0  24 0a c0 4e 00 01 00 01  00 00 e4 4d 00 04 d8 ef   $··N···· ···M····
0e0  26 0a c0 60 00 1c 00 01  00 02 ae 51 00 10 20 01   &··`···· ···Q·· ·
0f0  48 60 48 02 00 34 00 00  00 00 00 00 00 0a c0 3c   H`H··4·· ·······<
100  00 1c 00 01 00 00 ce 48  00 10 20 01 48 60 48 02   ·······H ·· ·H`H·
110  00 32 00 00 00 00 00 00  00 0a c0 72 00 1c 00 01   ·2····· ···r····
120  00 01 91 5a 00 10 20 01  48 60 48 02 00 36 00 00   ···Z·· · H`H··6··
130  00 00 00 00 00 0a c0 4e  00 1c 00 01 00 01 7e 52   ·······N ·····~R
140  00 10 20 01 48 60 48 02  00 38 00 00 00 00 00 00   ·· ·H`H· ·8······
150  00 0a                                              ··
```

Inserting records into DNS

- Example: just created startup “Network Utopia”
- Register name networkuptopia.com at a registrar (e.g., Network Solutions)
  - Need to provide registrar with names and IP addresses of your authoritative name server (primary and secondary)
  - Mechanism to update authoritative records
    - IETF RFC 2136
    - http://www.ietf.org/html.charters/dnsind-charter.html
  - Registrar inserts two RRs into the com TLD server:

```
(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)
```

- Put in authoritative server Type A record for www.networkuptopia.com and Type MX record for networkutopia.com
- How do people get the IP address of your Web site?

Examples

```
$ cat /etc/resolv.conf
# This file was automatically generated by WSL. To stop automatic generation of this file, remove this line.
nameserver 142.103.6.6
nameserver 137.82.1.1
nameserver 137.82.1.2
search cs.ubc.ca wireless.ubc.ca
CPSC-W-WAGNER# ~/bin
$
```

```
$ nslookup www.cs.ubc.ca
Server:         142.103.6.6
Address:        142.103.6.6#53

Name:   www.cs.ubc.ca
Address: 142.103.6.5

CPSC-W-WAGNER# ~/bin
$
```

```
$ dig | more

; <<>> DiG 9.10.3-P4-Ubuntu <<>>
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 53129
;; flags: qr rd ra; QUERY: 1, ANSWER: 13, AUTHORITY: 0, ADDITIONAL: 27

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;.                              IN      NS

;; ANSWER SECTION:
.                       458472  IN      NS      h.root-servers.net.
.                       458472  IN      NS      l.root-servers.net.
.                       458472  IN      NS      i.root-servers.net.
.                       458472  IN      NS      g.root-servers.net.
.                       458472  IN      NS      j.root-servers.net.
.                       458472  IN      NS      m.root-servers.net.
.                       458472  IN      NS      k.root-servers.net.
.                       458472  IN      NS      d.root-servers.net.
.                       458472  IN      NS      f.root-servers.net.
.                       458472  IN      NS      b.root-servers.net.
.                       458472  IN      NS      e.root-servers.net.
.                       458472  IN      NS      c.root-servers.net.
.                       458472  IN      NS      a.root-servers.net.

;; ADDITIONAL SECTION:
a.root-servers.net.     8009    IN      A       198.41.0.4
b.root-servers.net.     8008    IN      A       199.9.14.201
c.root-servers.net.     8009    IN      A       192.33.4.12
d.root-servers.net.     8010    IN      A       199.7.91.13
e.root-servers.net.     8008    IN      A       192.203.230.10
f.root-servers.net.     8009    IN      A       192.5.5.241
--More--
```

http://digwebinterface.com/

Local Machine at Root

```
$ dig @a.root-servers.net www.cs.ubc.ca

; <<>> DiG 9.10.3-P4-Ubuntu <<>> @a.root-servers.net www.cs.ubc.ca
; (2 servers found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 36407
;; flags: qr rd; QUERY: 1, ANSWER: 0, AUTHORITY: 3, ADDITIONAL: 7
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1472
;; QUESTION SECTION:
;www.cs.ubc.ca.                 IN      A

;; AUTHORITY SECTION:
ca.                     172800  IN      NS      c.ca-servers.ca.
ca.                     172800  IN      NS      j.ca-servers.ca.
ca.                     172800  IN      NS      any.ca-servers.ca.

;; ADDITIONAL SECTION:
c.ca-servers.ca.        172800  IN      A       185.159.196.2
j.ca-servers.ca.        172800  IN      A       198.182.167.1
any.ca-servers.ca.      172800  IN      A       199.4.144.2
c.ca-servers.ca.        172800  IN      AAAA    2620:10a:8053::2
j.ca-servers.ca.        172800  IN      AAAA    2001:500:83::1
any.ca-servers.ca.      172800  IN      AAAA    2001:500:a7::2

;; Query time: 44 msec
;; SERVER: 198.41.0.4#53(198.41.0.4)
;; WHEN: Wed Feb 07 08:25:16 STD 2018
;; MSG SIZE  rcvd: 235
```

Whois

Try browsing some sites

198.41.0.4
199.9.14.201
192.33.4.12
199.7.91.13
192.203.230.10
192.5.5.241
192.112.36.4
198.97.190.53
192.36.148.17
192.58.128.30
193.0.14.129
199.7.83.42
202.12.27.33

Website that will give you information about an IP address (whois entry)

https://myip.ms/browse/comp_ip/IPv4_Address_Database

Website listing fake websites (do not visit any Site listed there) :

https://db.aa419.org/fakebankslist.php
WARNING … WARNING

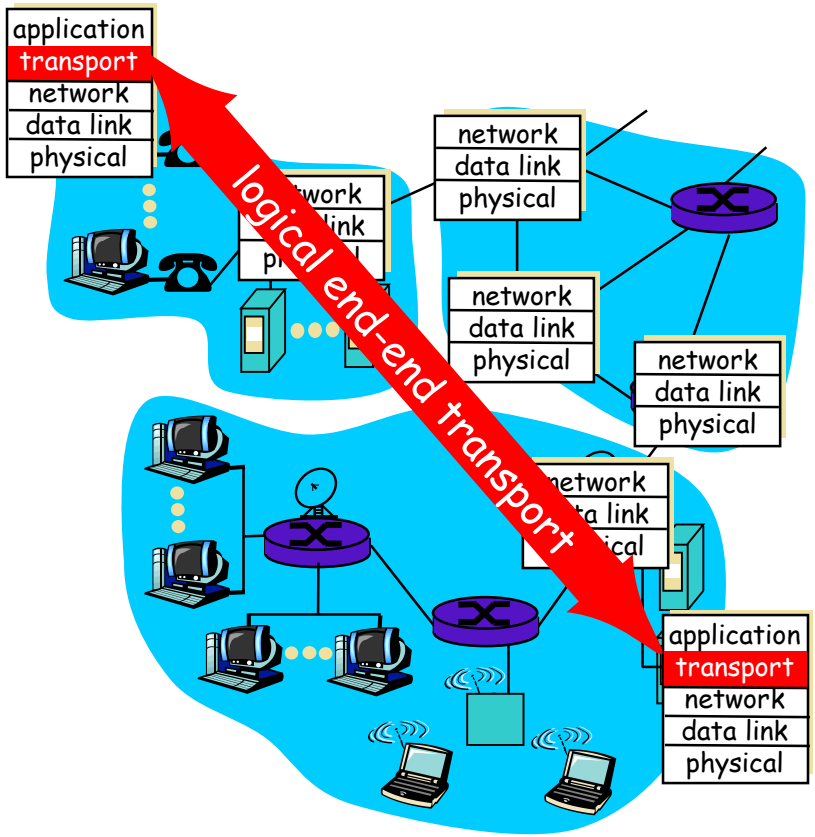
What is needed to set up a company?

Say we wanted to start a new online company called KidsBeSafe

TRANSPORT LEVEL PROTOCOLS

Transport Protocol –main purpose

- provide *logical communication* between app processes running on different hosts
- transport protocols run in end systems
  - send side: breaks app messages into segments, passes to network layer
  - recv side: reassembles segments into messages, passes to app layer
- Multiplexing of communication over the network.



Logical connection between processes on different hosts!

Transport layer provides multiplexing + demultiplexing by using different ports

Transport versus Network layer

- *network layer:* logical communication between hosts (ends at the interface)
- *transport layer:* logical communication between processes (ends at the application)

Household analogy:

12 kids sending letters to 12 kids

- processes = kids
- app messages = letters in envelopes
- hosts = houses
- transport protocol = Ann and Bill
- network-layer protocol = postal service

TRANSPORT LEVEL SERVICES

List of Possible Services

Partial delivery

Reliable delivery

Reliable means that it either works or doesn't work and provides an error message

Flow control

Local

Congestion control

Global

Signaling

Ordered delivery

Bidirectional

Unidirectional

Connection oriented

Connection-less

Segmentation

Data bundling

Stream-oriented

Message-oriented

Multicast

Non-duplication

TCP is friendly; will back off, send less messages if there's too much conjestion

List of Possible Services

You do not have to memorize this list, many of the services on this list will be covered in M6 and by the end of term you will know more about them. But there are others like multicast, signaling, data bundling that we will NOT discuss and you will not be asked about. The list is good to show you why TCP/UDP are not always the best suited to applications.

Partial delivery -> delivers corrupted packets (see in M6)
Reliable delivery -> if delivered, correct (in M6)
Flow control -> stop/go, prevents overrunning
Congestion control -> plays nice, does not overload buffers at routers
Signaling -> can send signals, delivered before data (we will not discuss this)

List of Possible Services

Ordered delivery -> delivers segments in-order (in M6)
Bidirectional -> data flows both ways
Unidirectional -> data can flow only one way
Connection oriented -> first makes a connection between ends
Connection-less -> no connection between ends
Segmentation -> cut the data stream into segments for IP
Data bundling -> optimization to send full packets (Nagle –not covered)
Stream-oriented -> sends a stream of bytes
Message-oriented -> send messages
Multicast -> can multicast on LAN
Non-duplication -> handle duplicate packets (in M6)

Flow control and congestion control

- Local link(s)
- Overrun the destination
- TCP (advertised windows)
- Control ingress into network
- Stop & go on links
- HW and protocols

- Global network
- overwhelming network buffers
- TCP (back-off)
- Restrict ingress to network
- HW and protocols

APPLICATION REQUIREMENTS

What transport service does an app need?

Data loss

- some apps (e.g., audio) can tolerate some loss
- other apps (e.g., file transfer, telnet) require 100% reliable data transfer

Timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be “effective”

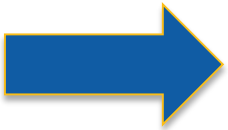
Bandwidth

- some apps (e.g., multimedia) require minimum amount of bandwidth to be “effective”
- other apps (“elastic apps”) make use of whatever bandwidth they get

Service Characteristics

General Properties:

- Bandwidth
- Latency
- Jitter
- Loss
- Use corrupt or partial packets
- Tolerate out of order, duplicate



Quality of Service:

- Bandwidth
- Constant bit rate
- Variable bit rate
- Real-time
- Non-real-time

Internet apps: application, transport protocols

Should the app use TCP or UDP?

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	
remote terminal access	Telnet [RFC 854]	
Web	HTTP [RFC 2616]	
file transfer	FTP [RFC 959]	
streaming multimedia	proprietary (e.g. RealNetworks)	
Internet telephony	proprietary (e.g., Skype)	

Question: a lot of apps today use https and json. Good? websockets RFC 6455

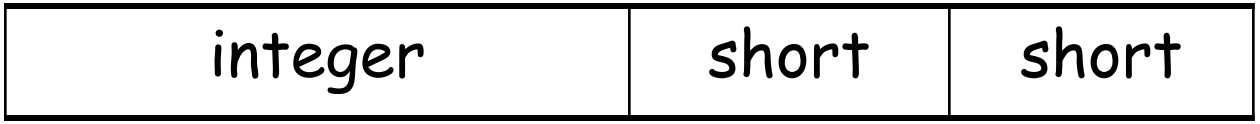
Transport service requirements of common apps

Application	Data loss	Bandwidth	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
instant messaging	no loss	elastic	yes and no

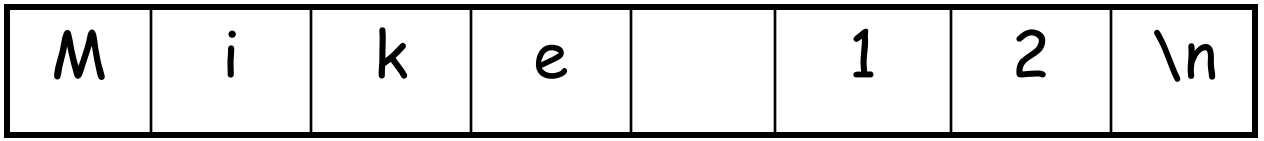
BYTES ON THE WIRE (presentation layer)

Message Composition

- Message composed of fields
  - Fixed-length fields

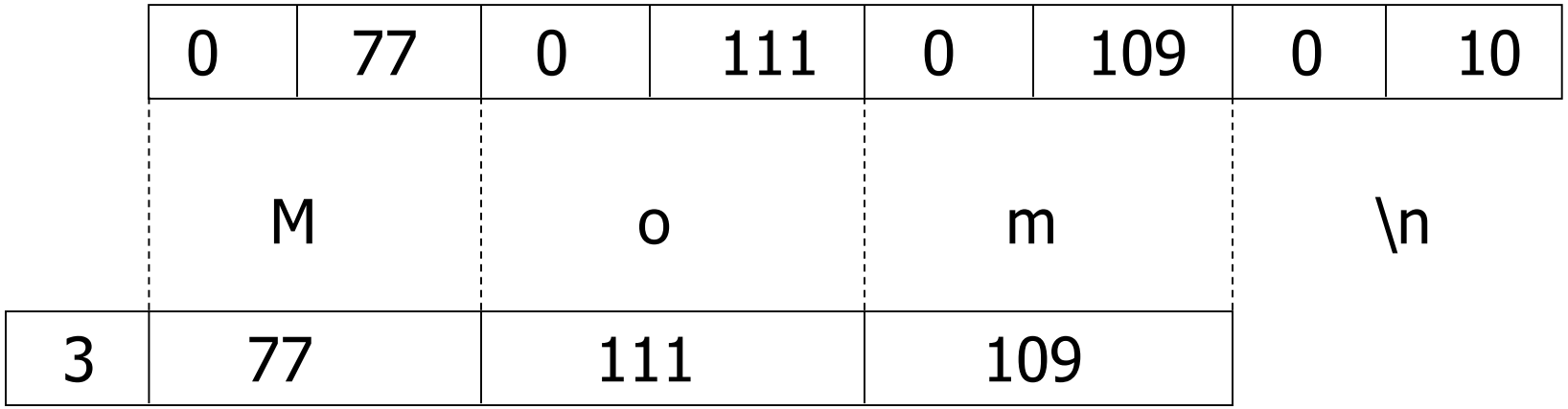


  - Variable-length fields



Primitive Types

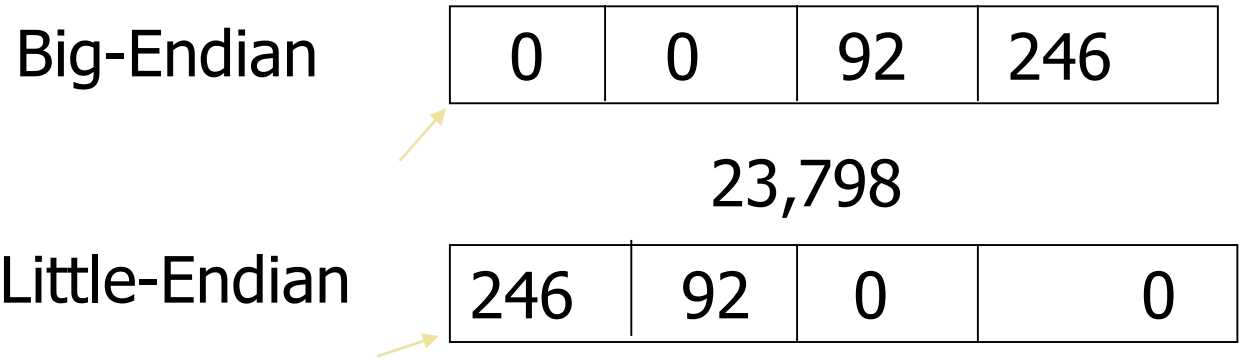
- String
  - Character encoding: ASCII, Unicode, UTF
  - Delimit: length versus termination character



Primitive Types

- Integer
  - Native representation

4-byte two's-complement integer

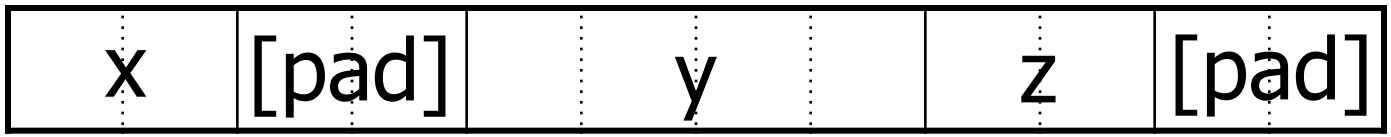


  - Network byte order (Big-Endian!)
    - Use for multi-byte, binary data exchange
    - htonl(), htons(), ntohl(), ntohs()

Padding differences (Beware!)

- Architecture alignment restrictions
- Compiler pads **`structs`** to accommodate

```
struct tst {
  short x;
  int y;
  short z;
};
```

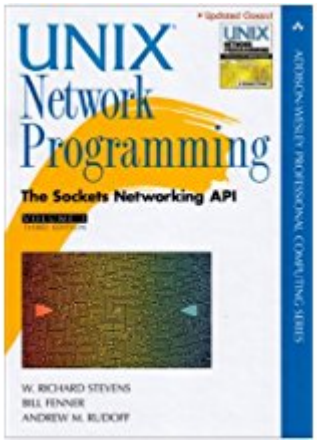


- Problem: Alignment restrictions vary
- Solution:
  1. Rearrange **`structs`** members
  2. Serialize **`structs`** by-member to include padding

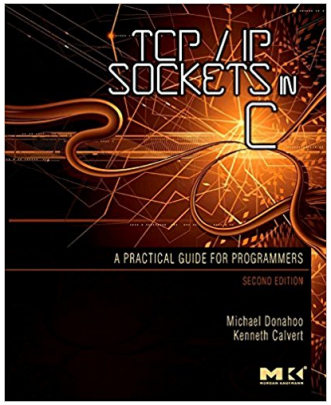
TCP and UDP

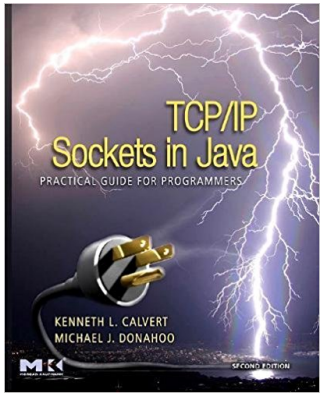
Resources

BIBLE



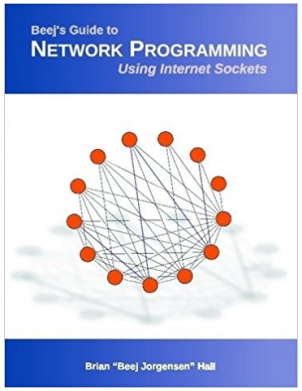
Unix Network Programming, Volume 1: The Sockets Networking API (3rd Edition)





TCP/IP Sockets in C, Second Edition: Practical Guide for Programmers (The Morgan Kaufmann Practical Guides Series) 2nd Edition

FREE



Beej's Guide to Network Programming

Application-layer protocol defines

- Types of messages exchanged, e.g. request and response messages
- Syntax of message types: what fields in messages and how fields are delineated
- Semantics of the fields, ie, meaning of information in fields
- Rules for when and how processes send and respond to messages

Public-domain protocols:

- defined in RFCs
- allows for interoperability
- eg, HTTP, SMTP

Proprietary protocols:

- eg, Whatsapp, telegraph, etc.

Transport Level Protocols

- TCP
- UDP
- List of all IETF standardized transport protocols:

UDP	UDP-Lite	TCP	Multipath TCP	SCTP

QUIC transport over UDP

Transport Protocols

Feature	UDP	UDP-Lite	TCP	Multipath TCP	SCTP	DCCP
Packet header size	8 bytes	8 bytes	20–60 bytes	50–90 bytes	12 bytes[b]	12 or 16 bytes
Typical data-packet overhead	8 bytes	8 bytes	20 bytes	?? bytes	44–48+ bytes[c]	12 or 16 bytes
Transport-layer packet entity	Datagram	Datagram	Segment	Segment	Datagram	Datagram
Connection-oriented	No	No	Yes	Yes	Yes	Yes
Reliable transport	No	No	Yes	Yes	Yes	No
Unreliable transport	Yes	Yes	No	No	Yes	Yes
Preserve message boundary	Yes	Yes	No	No	Yes	Yes
Delivery	Unordered	Unordered	Ordered	Ordered	Ordered / Unordered	Unordered
Data checksum	Optional	Yes	Yes	Yes	Yes	Yes
Checksum size	16 bits	16 bits	16 bits	16 bits	32 bits	16 bits
Partial checksum	No	Yes	No	No	No	Yes
Path MTU	No	No	Yes	Yes	Yes	Yes
Flow control	No	No	Yes	Yes	Yes	No
Congestion control	No	No	Yes	Yes	Yes	Yes
Explicit Congestion Notification	No	No	Yes	Yes	Yes	Yes
Multiple streams	No	No	No	No	Yes	No
Multi-homing	No	No	No	Yes	Yes	No
Bundling / Nagle	No	No	Yes	Yes	Yes	No

Wikipedia: https://en.wikipedia.org/wiki/Transport_layer

Internet transport protocols services (multiplexing and communication)

TCP service:

- *connection-oriented:* setup required between client and server processes
- *reliable transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *does not provide:* timing, minimum bandwidth guarantees

UDP service:

- unreliable data transfer between sending and receiving process
- does not provide: connection setup, reliability, flow control, congestion control, timing, or bandwidth guarantee

Q: why bother? Why is there a UDP?

SOCKETS

Socket

- Logical Communication Endpoint in a process

```
mysock = socket( family, type, protocol )
```

```
mysock = socket( PF_INET, SOCK_DGRAM, 0 )
```

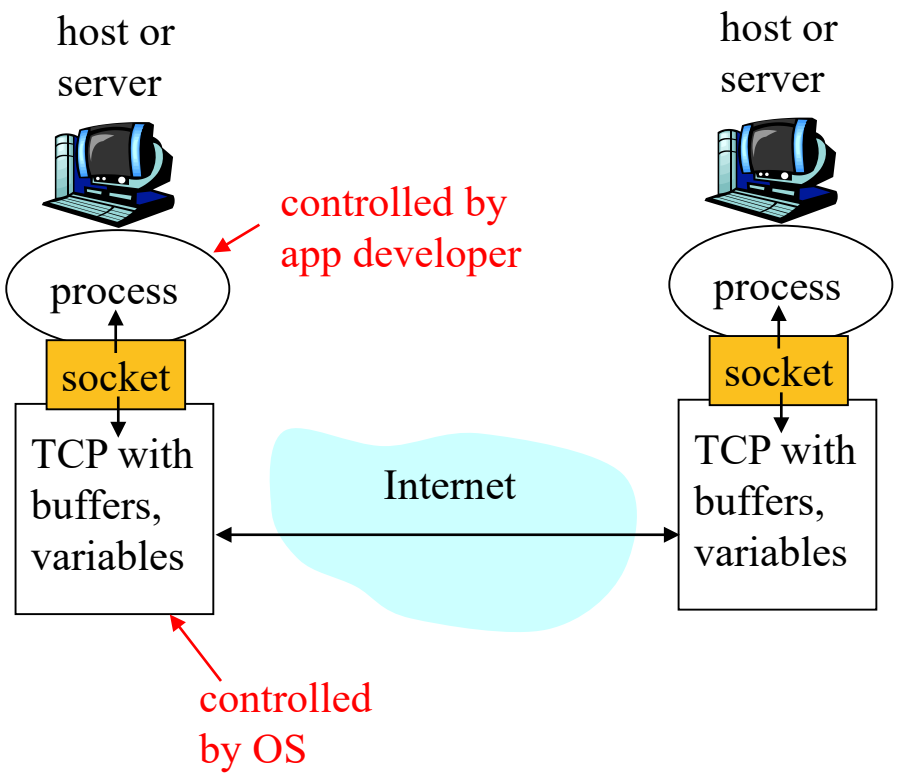
mysock is a file descriptor

Local Port
Remote Port

Port addresses communication endpoint => socket

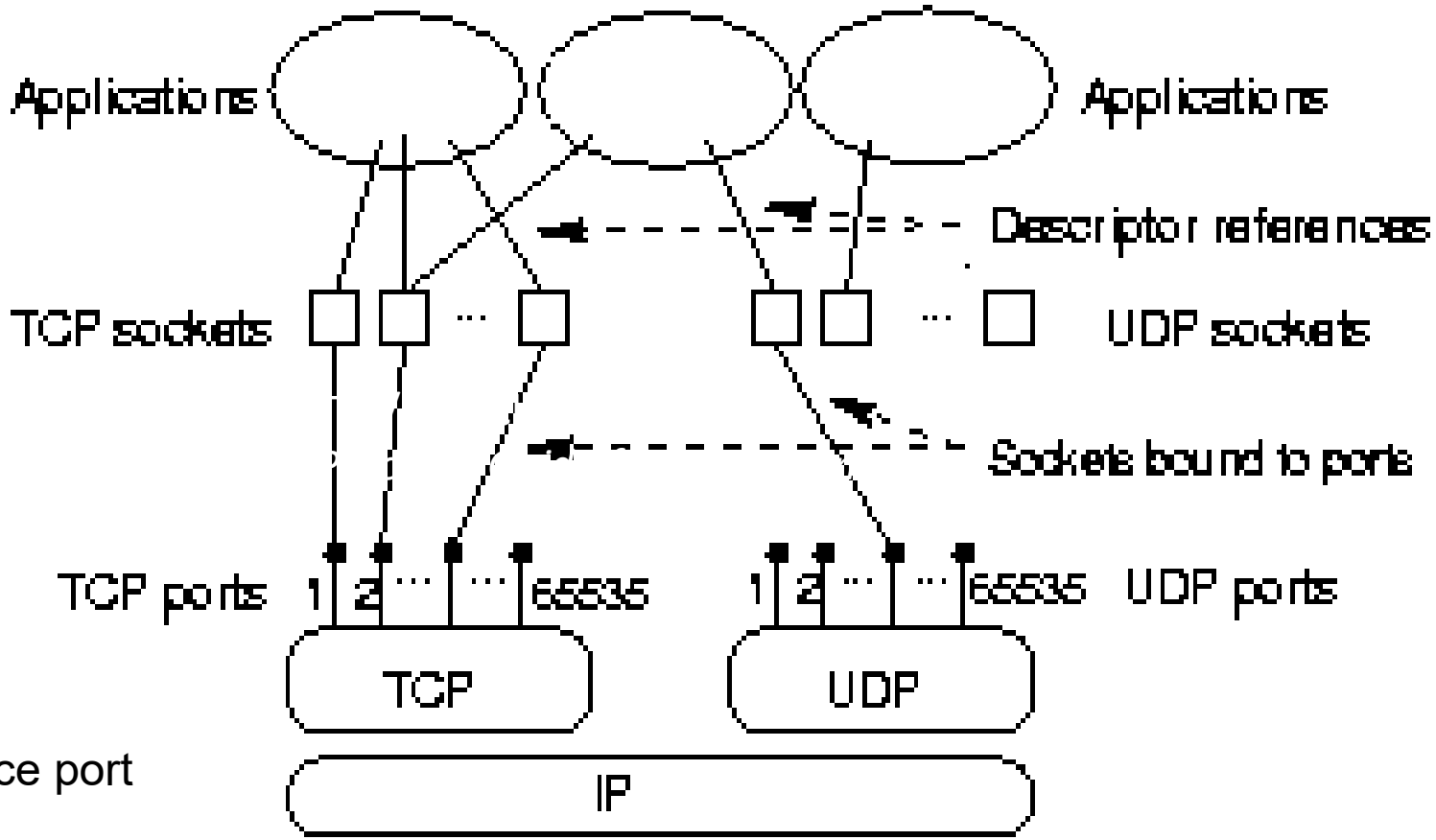
Sockets

- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door which brings message to socket at receiving process



API: (1) choice of transport protocol; (2) ability to fix a few parameters (options)

Sockets, communication endpoints



source port

INET Sockets

- INET-specific sockets

	Family	Type	Protocol
TCP	PF_INET	SOCK_STREAM	IPPROTO_TCP
UDP		SOCK_DGRAM	IPPROTO_UDP

- Socket reference
  - File (socket) descriptor in UNIX
  - Socket handle in WinSock

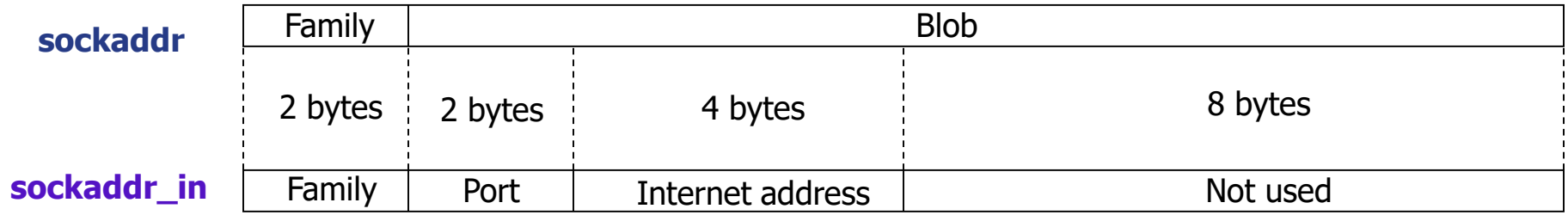
Generic

```
struct sockaddr
{
  unsigned short sa_family;    /* Address family (e.g., AF_INET) */
  char sa_data[14];            /* Protocol-specific address information */
};
```

IP Specific

```
struct sockaddr_in
{
   unsigned short sin_family;  /* Internet protocol (AF_INET) */
   unsigned short sin_port;    /* Port (16-bits) */
   struct in_addr sin_addr;    /* Internet address (32-bits) */
   char sin_zero[8];           /* Not used */
};

struct in_addr
{
  unsigned long s_addr;    /* Internet address (32-bits) */
};
```



NETWORK ADDRESS TRANSLATION

NAT: Network Address Translation

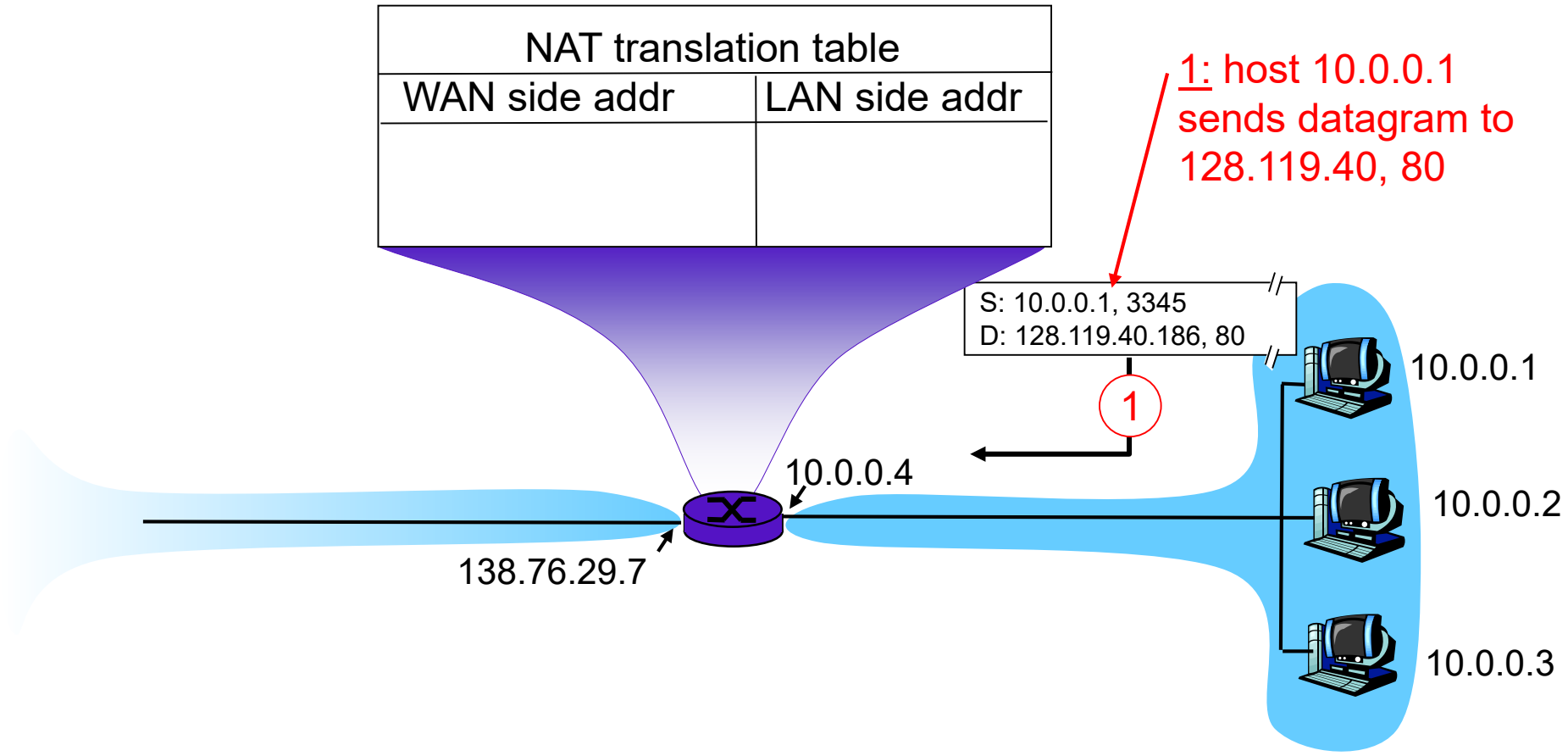
- Motivation: local network uses just one IP address as far as outside word is concerned:
  - no need to be allocated range of addresses from ISP: - just one IP address is used for all devices
  - can change addresses of devices in local network without notifying outside world
  - can change ISP without changing addresses of devices in local network
  - devices inside local net not explicitly addressable, visible by outside world (a security plus).

NAT: Network Address Translation

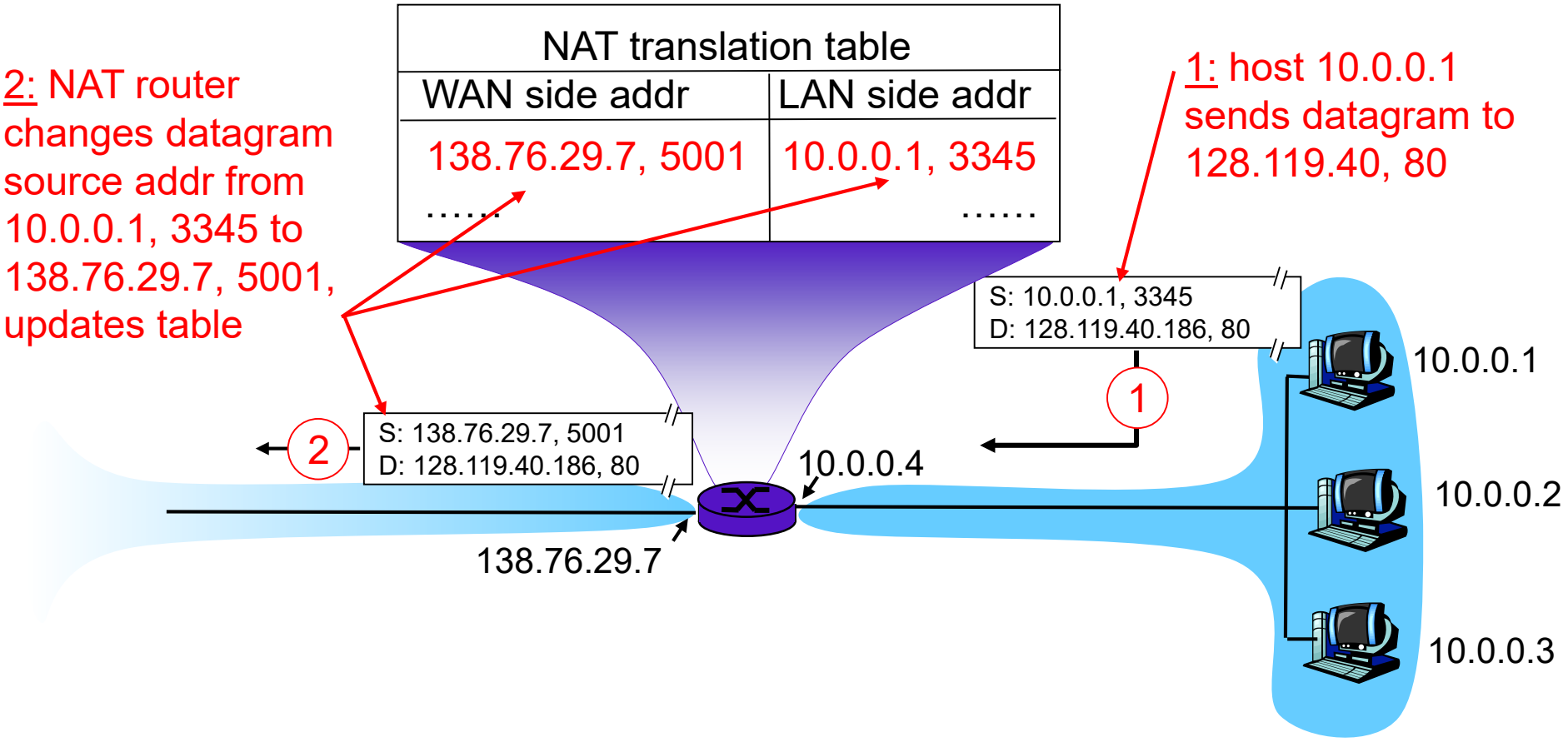
Implementation: NAT router must:

- *outgoing datagrams: replace* (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #)
  - . . . remote clients/servers will respond using (NAT IP address, new port #) as destination addr.
- *remember (in NAT translation table)* every (source IP address, port #) to (NAT IP address, new port #) translation pair
- *incoming datagrams: replace* (NAT IP address, new port #) in dest fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table

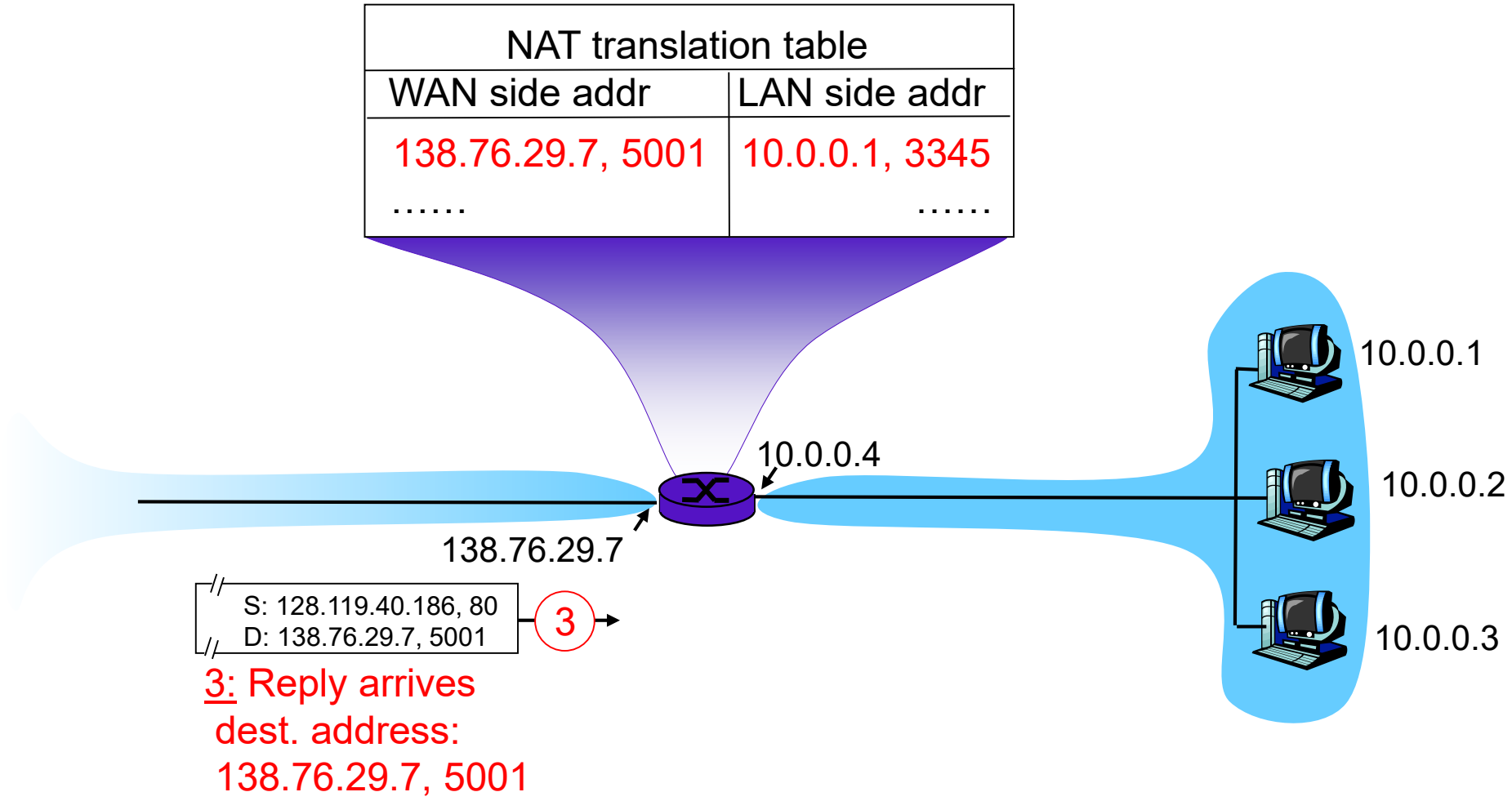
NAT: Network Address Translation



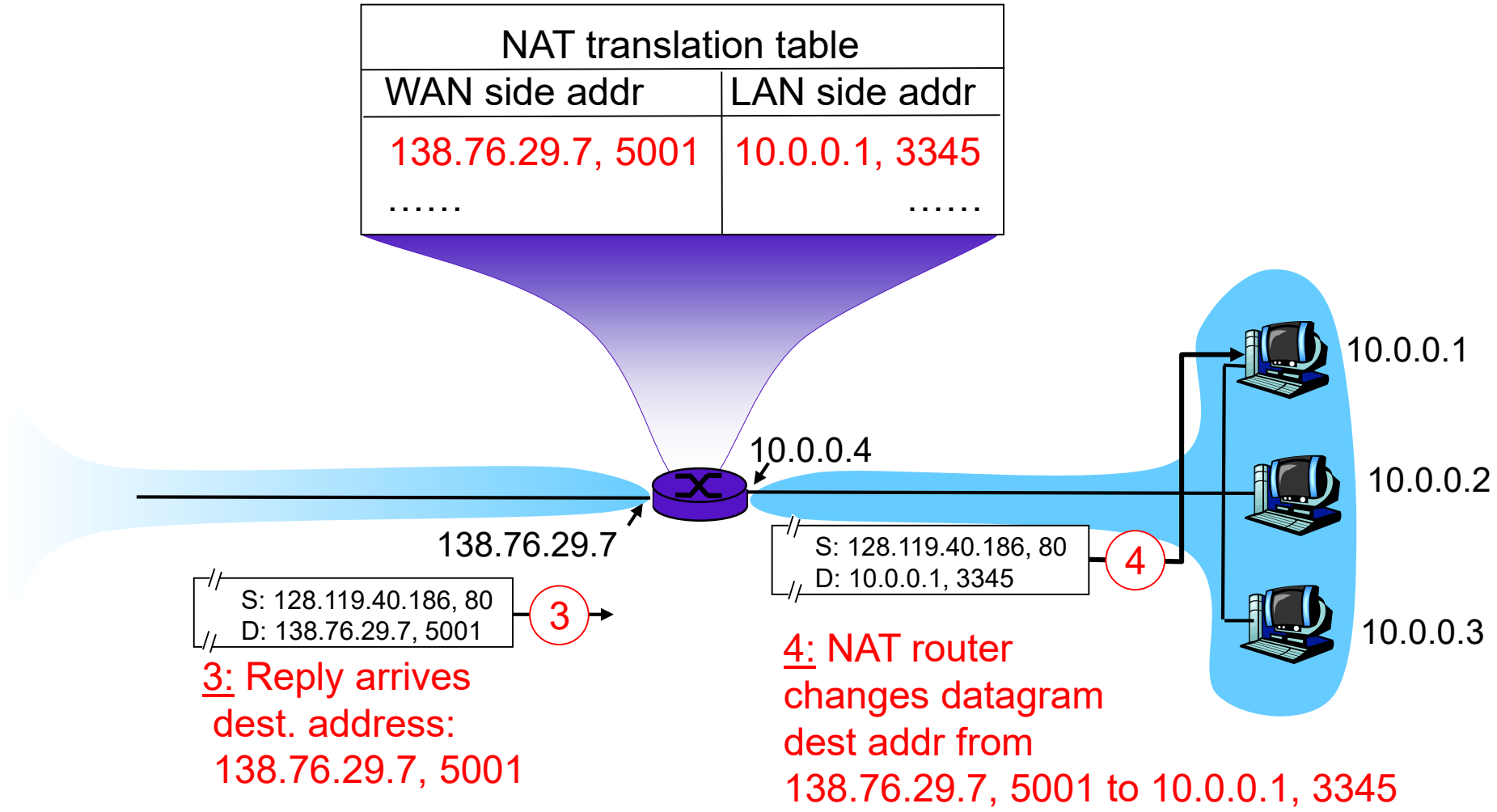
NAT: Network Address Translation



NAT: Network Address Translation



NAT: Network Address Translation



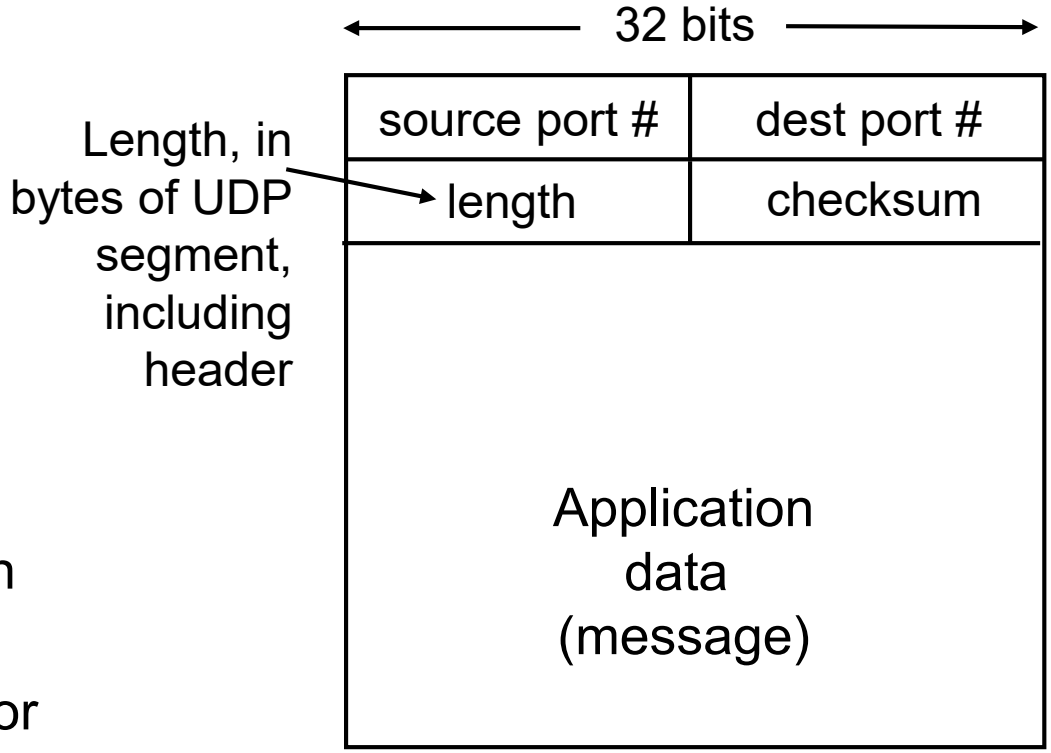
NAT: Network Address Translation

- 16-bit port-number field:
  - 60,000 simultaneous connections with a single LAN-side address!
- NAT is controversial:
  - routers should only process up to layer 3
  - violates end-to-end argument
    - NAT possibility must be taken into account by app designers, eg, P2P applications
  - address shortage should instead be solved by IPv6

UDP

UDP Protocol Format

- often used for streaming multimedia apps
  - loss tolerant
  - rate sensitive
- other UDP uses
  - DNS
  - SNMP
- reliable transfer over UDP: add reliability at application layer
  - application-specific error recovery!



UDP segment format

Socket programming *with UDP*

UDP provides unreliable transfer of groups of bytes ("datagrams") between client and server

UDP: no "connection" between client and server

- no handshaking
- sender explicitly attaches IP address and port of destination to each packet
- server must extract IP address, port of sender from received packet
- Basically UDP only provides multiplexing/demultiplexing

UDP: transmitted data may be received out of order, or lost

UDP Client/Server Interaction

Server starts by getting ready to receive client connections…

Client

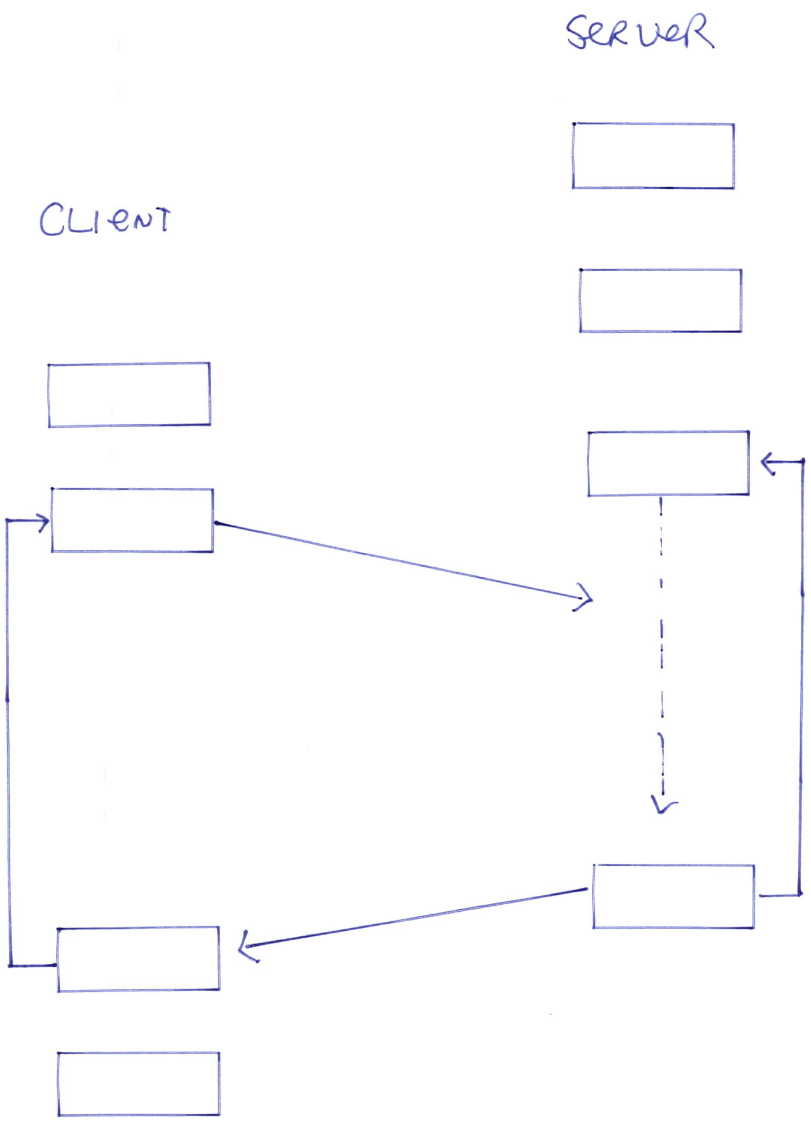
1. Create a UDP socket
2. Repeatedly
   a. sendto a message
   b. recvfrom a reply
3. Close the connection

Server

1. Create a UDP socket
2. Bind a port to socket
3. Repeatedly
   a. Receive a message
   b. Send a message

Why isn't there a close for the server?

UDP CLIENT / SERVER

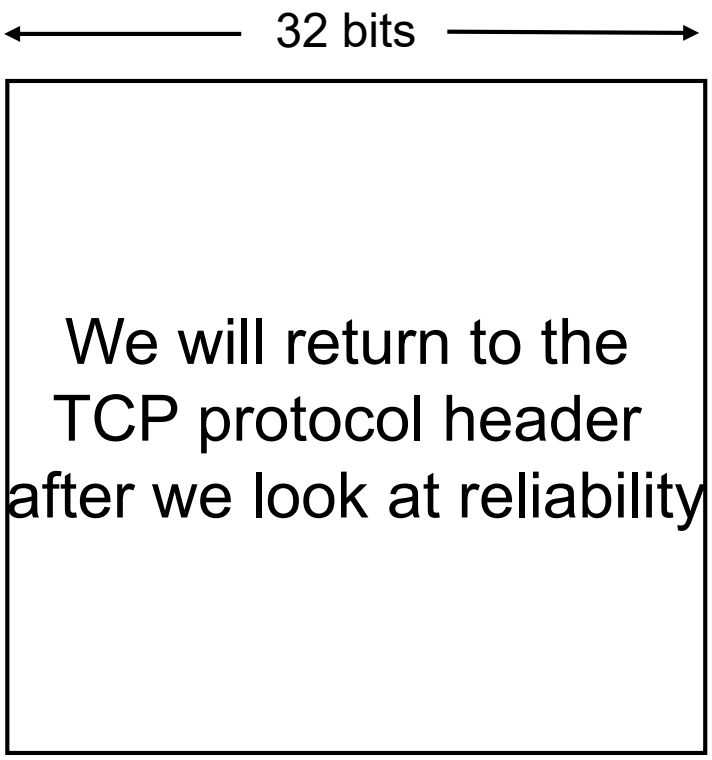


UDP Tidbits

- There is a direct one-one relationship between a send and receive.
- Receive the max of what you asked or the size of the message (there is a peek routine).
- Beware of deadlock
- multi-homed hosts (bind to inaddr_any)
- multicast and broadcast

TCP

TCP Protocol Format



TCP segment format

Socket programming *with TCP*

TCP provides reliable transfer of a stream of bytes ("broken into datagrams") between client and server

TCP: creates a connection between client and server

- Need for initial handshaking
- sender receiver automatically attach src/dest IP address and port to each datagram

TCP: transmitted data always received in-order with no loss data (only best effort --- when it can't deliver data it fails)

TCP Client/Server Interaction

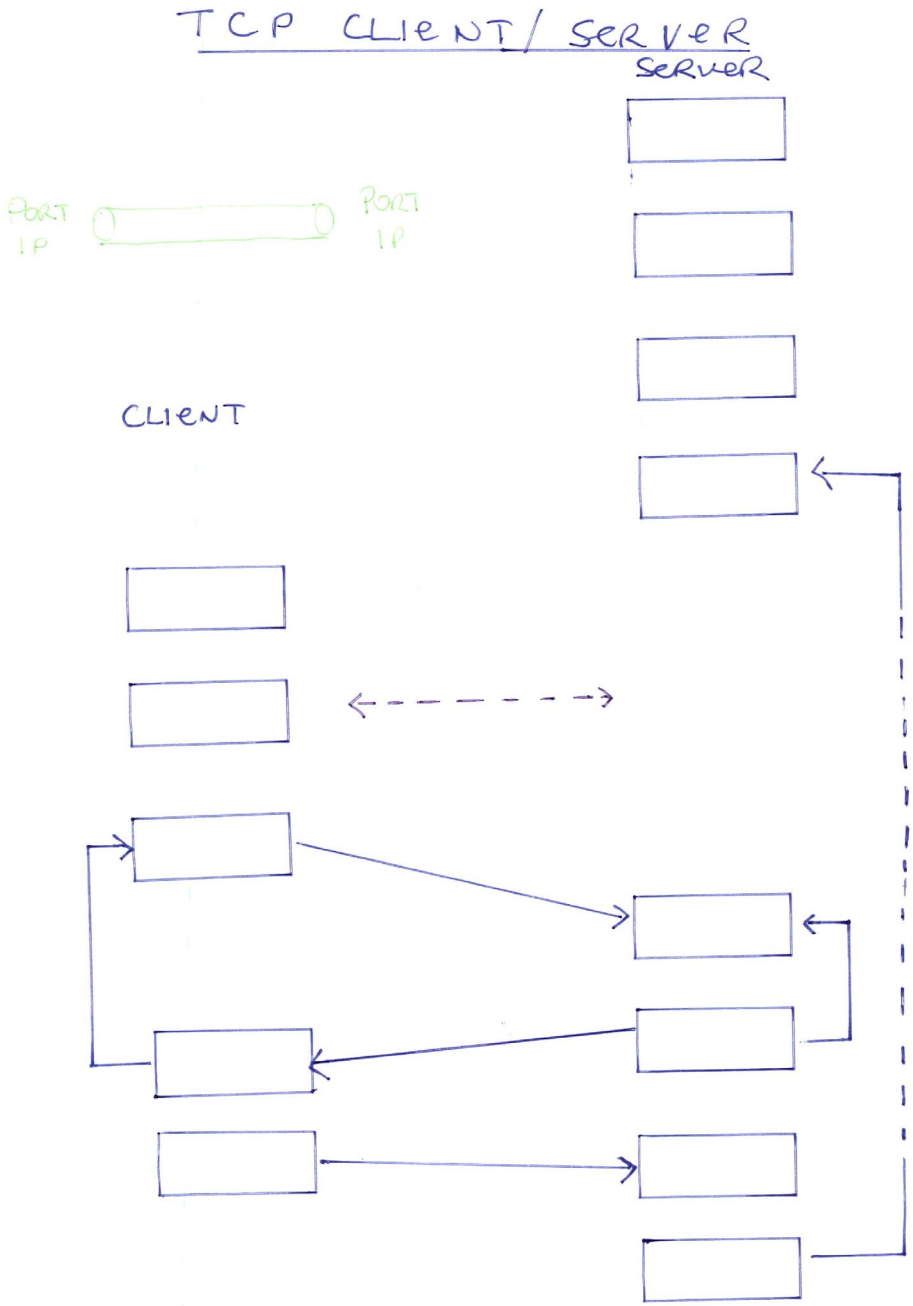
Server starts by getting ready to receive client connections…

Client

1. Create a TCP socket
2. Establish connection (connect)
3. Communicate
4. Close the connection

Server

1. Create a TCP socket
2. Assign a port to socket (bind)
3. Set socket to listen
4. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection


TCP CLIENT/SERVER
SERVER
PORT
IP
PORT
IP
CLIENT

TCP Sockets and Ports (mux/demux)

- Socket addresses (communication endpoints)
- Source/Destination port and Source/Destination IP addresses uniquely defines a TCP connection
- What happens where there are more than one interface (IP addresses – inaddr_any)
- Only one set of ports for all interfaces

Socket Calls

- Socket()
- Connect()
- Bind()
- Listen()
- Accept()
- Sendto(), Recvfrom()
- Read(), Write()

Client-side connecting

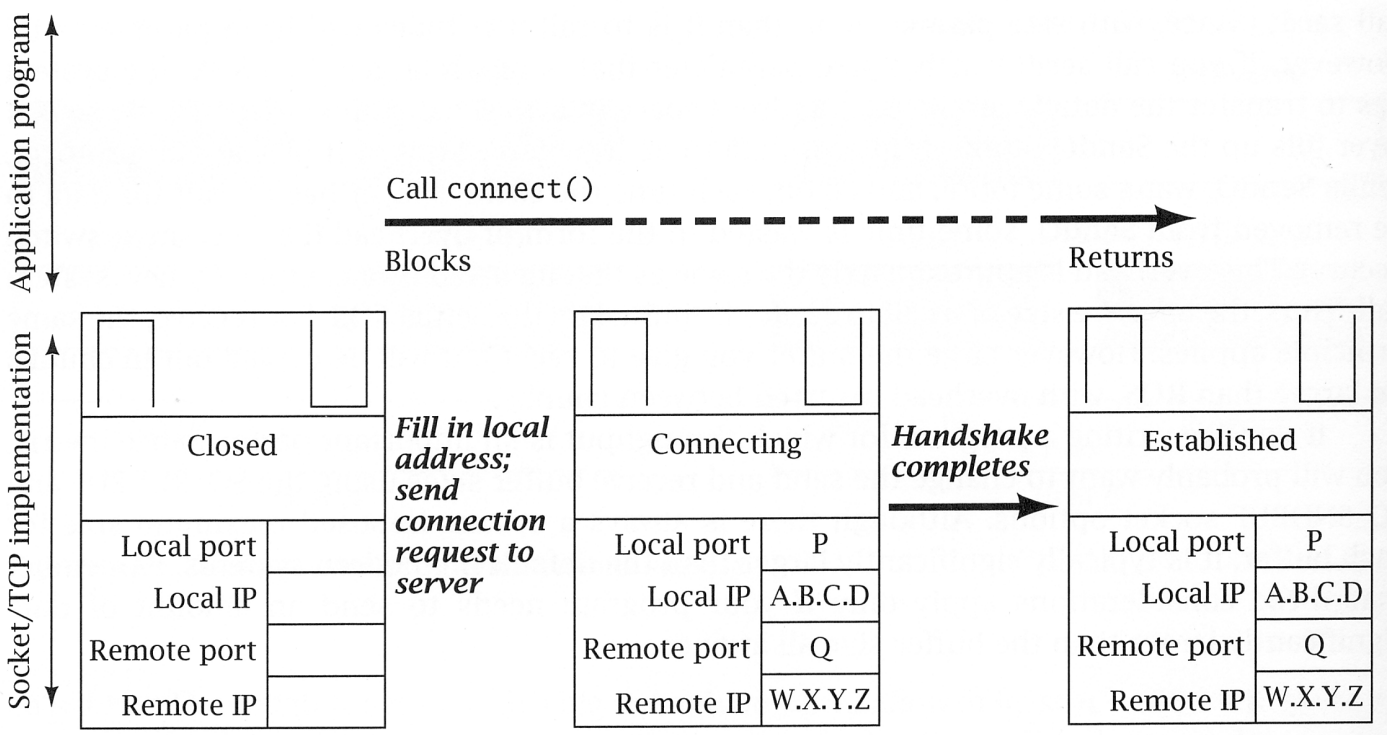


Figure 6.6: Client-side connection establishment.

Server-side binding to a port

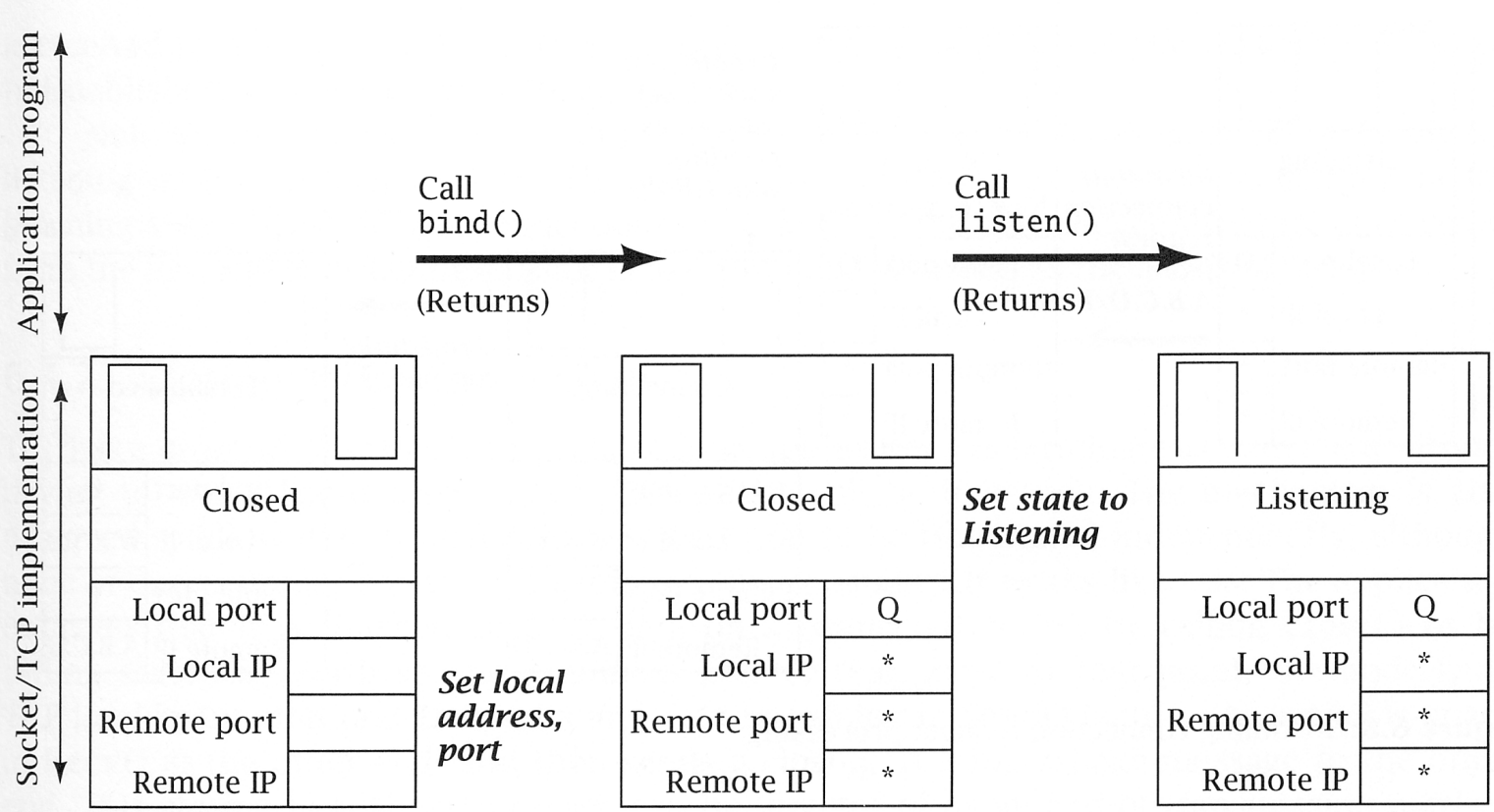


Figure 6.7: Server-side socket setup.

Pocket Socket Book – Donahoo and Calvert

Server-side listening and connecting

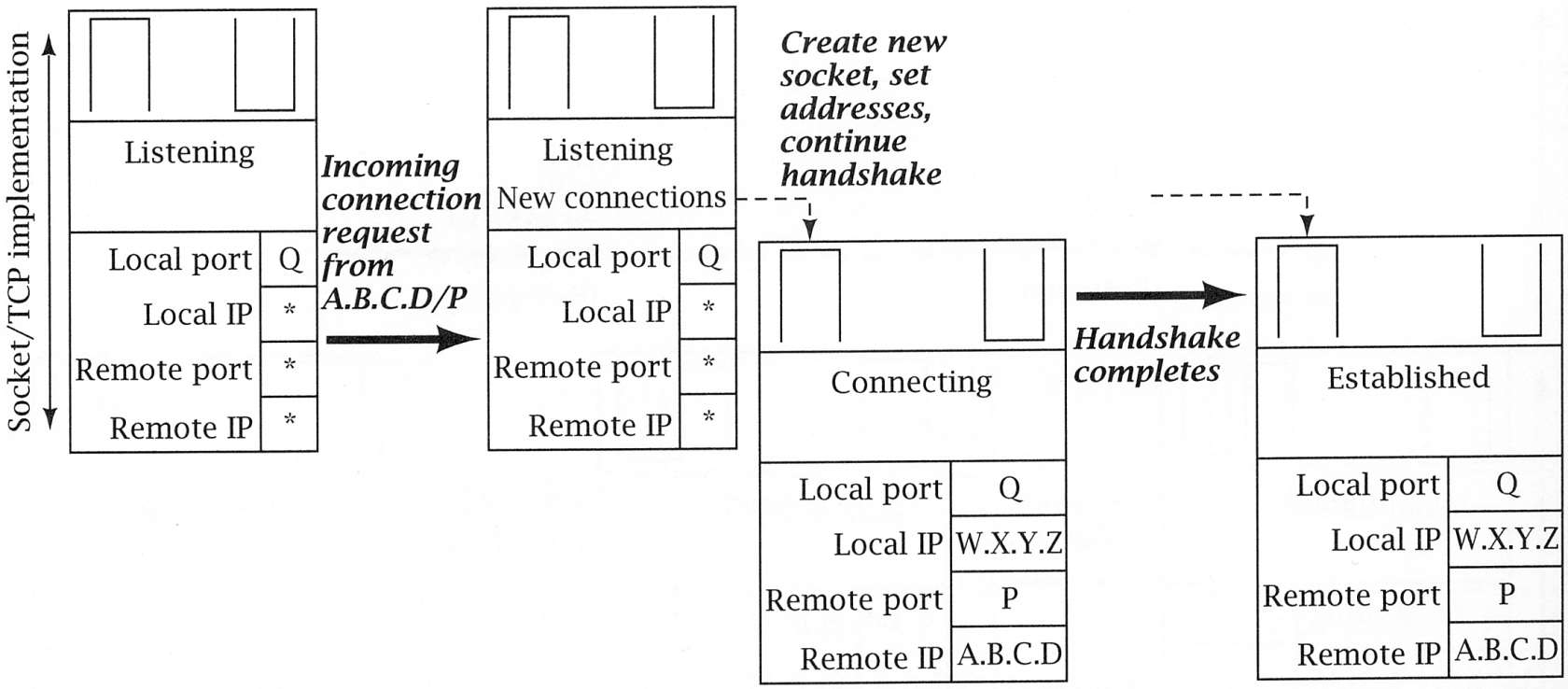


Figure 6.8: Incoming connection request processing.

Server-side accept() processing

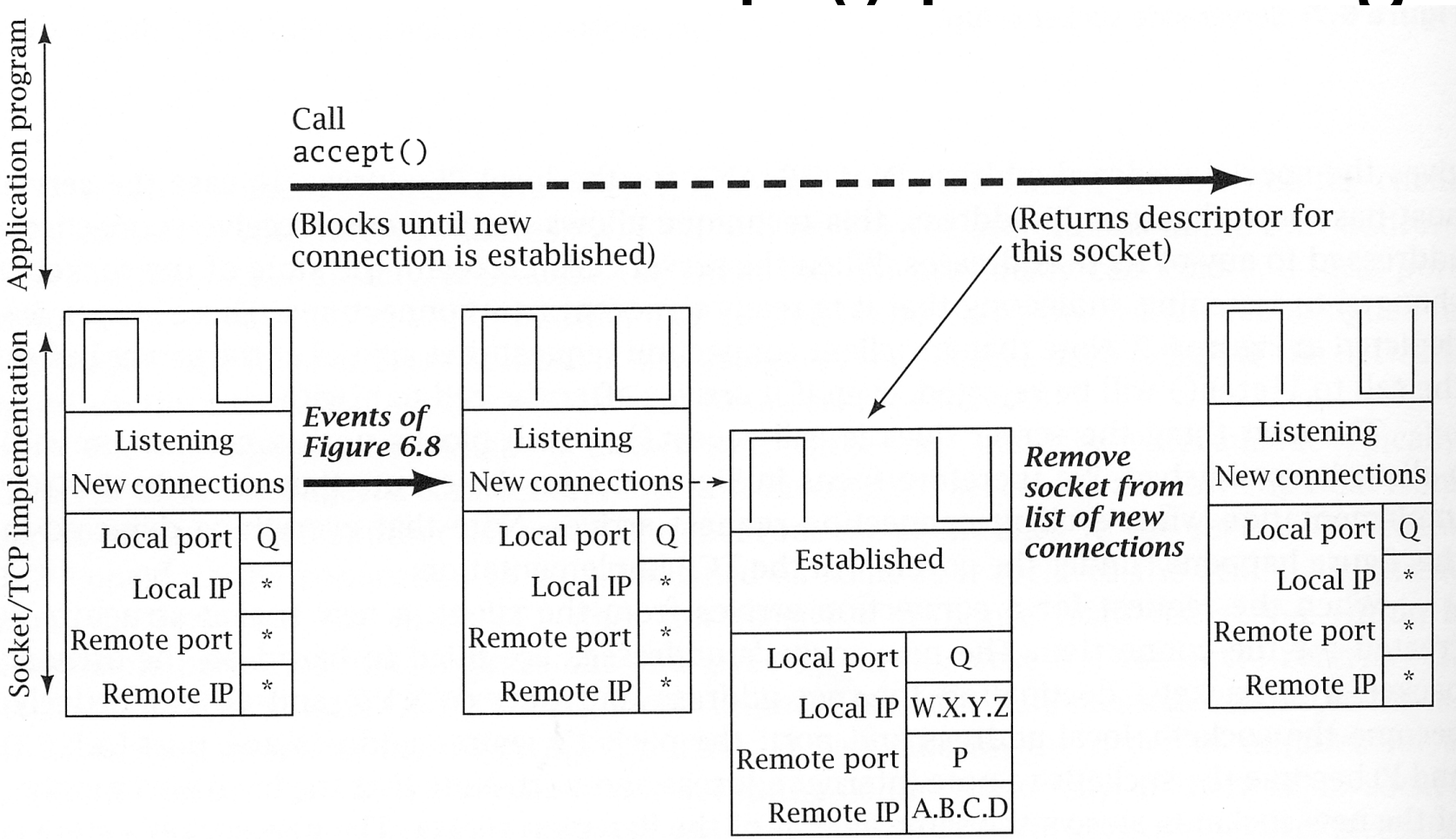


Figure 6.9: `accept()` processing.

Multiple matching ports

What happens on 172.16.1.9:56789 destination 10.1.2.3:99?

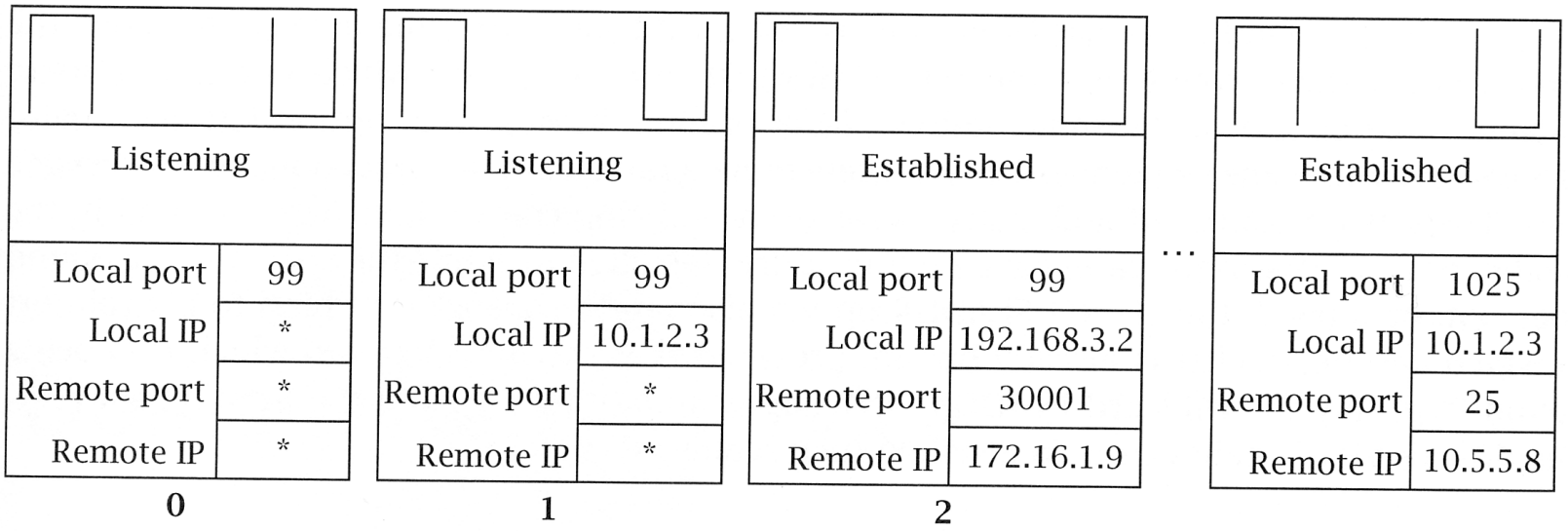


Figure 6.12: Demultiplexing with multiple matching sockets.