

OSI Reference Model

CEN 223 - Internet Communication

Assoc. Prof. Dr. Fatih ABUT

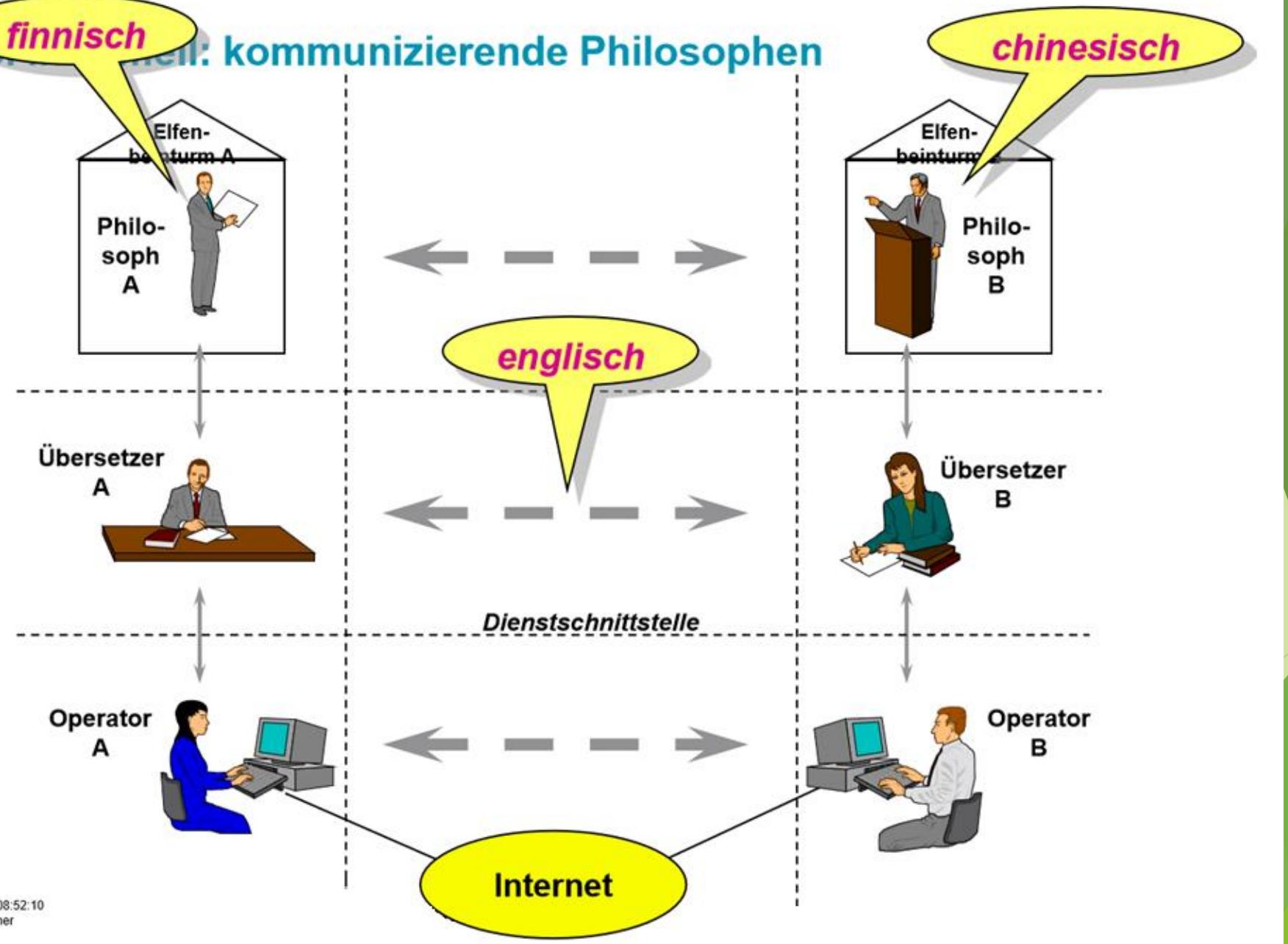
Çukurova University

Department of Computer Engineering

OSI Reference Model

- ▶ OSI Reference Model - internationally standardised network architecture.
- ▶ OSI = *Open Systems Interconnection*: deals with *open systems*, i.e. systems open for communications with other systems.
- ▶ Specified in ISO 7498.
- ▶ Model has 7 layers.

Übersetzungsmittel: kommunizierende Philosophen



OSI Features

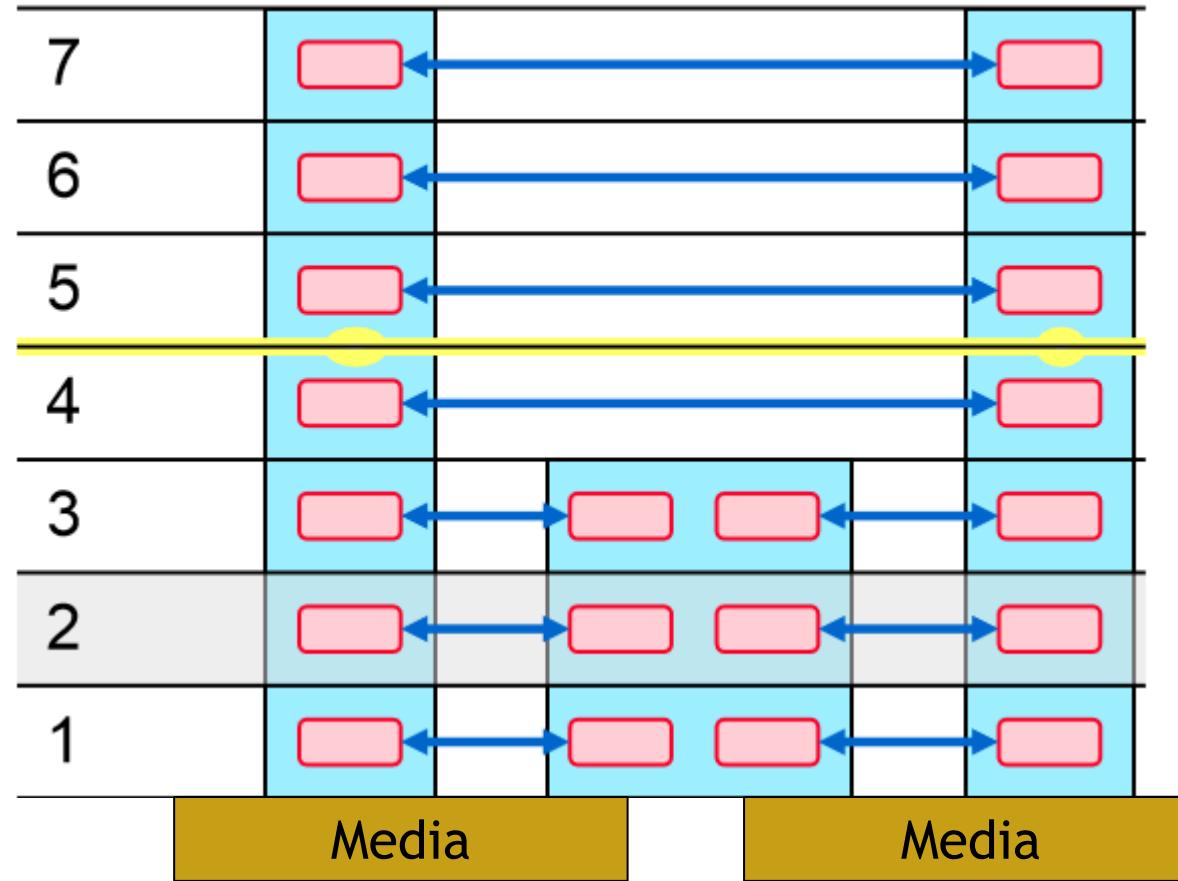
- ▶ Open system standards over the world
- ▶ Rigorously defined structured, hierarchical network model
- ▶ Commonly defined terminology
- ▶ Complete description of the function **(not the implementation!)**
- ▶ Provide standard test procedures

OSI History

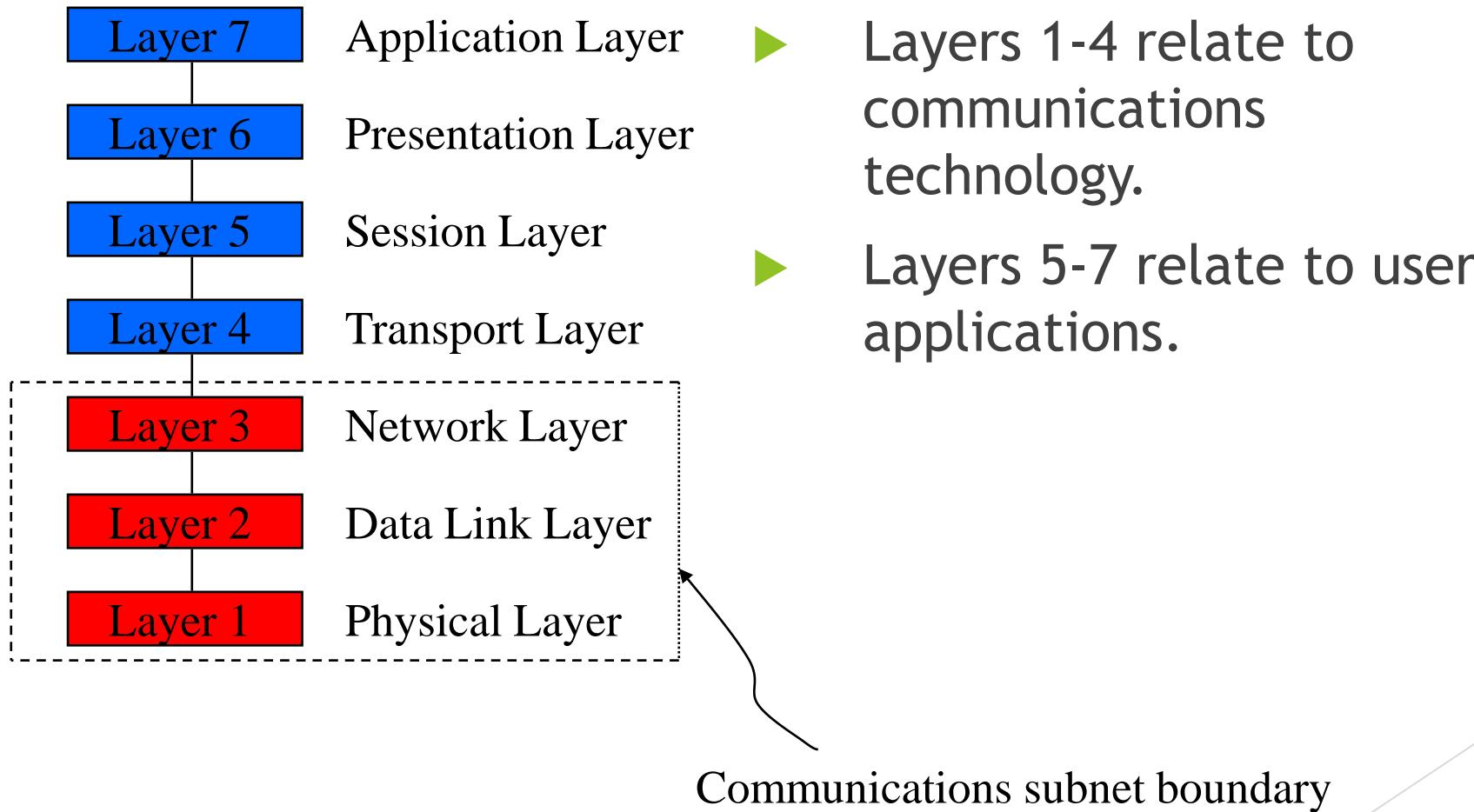
- ▶ In 1978, the International Standard Organization (ISO) began to develop its OSI framework architecture.
- ▶ OSI has two major components: an abstract model of networking, called the Basic Reference Model or seven-layer model, and a set of specific protocols.
- ▶ The concept of a 7 layer model was provided by the work of Charles Bachman, then of Honeywell.

Components of OSI-Model

- ▶ Layers
- ▶ Systems
- ▶ Media
- ▶ Services
- ▶ Entities
- ▶ Protocols



7-Layer OSI Model



Layer 1: Physical Layer

- ▶ Transmits bits from one computer to another
- ▶ Regulates the transmission of a stream of bits over a physical medium.
- ▶ Defines how the cable is attached to the network adapter and what transmission technique is used to send data over the cable. Deals with issues like
 - ▶ The definition of 0 and 1, e.g. how many volts represents a 1, and how long a bit lasts?
 - ▶ Whether the channel is simplex or duplex?
 - ▶ How many pins a connector has, and what the function of each pin is?

Layer 2: Data Link Layer

- Packages raw bits from the Physical layer into frames (logical, structured packets for data).
- Provides reliable transmission of frames
 - It waits for an acknowledgement from the receiving computer.
 - Retransmits frames for which acknowledgement not received
- Access to shared medium

Layer 3: Network Layer

- ▶ Manages addressing/routing of data within the subnet
 - ▶ Addresses messages and translates logical addresses and names into physical addresses.
 - ▶ Determines the route from the source to the destination computer
 - ▶ Manages traffic problems, such as switching, routing, and controlling the congestion of data packets.
- ▶ Routing can be:
 - ▶ Based on static tables
 - ▶ determined at start of each session
 - ▶ Individually determined for each packet, reflecting the current network load.

Layer 4: Transport Layer

- ▶ Manages transmission packets
 - ▶ Repackages long messages when necessary into small packets for transmission
 - ▶ Reassembles packets in correct order to get the original message.
- ▶ Handles error recognition and recovery.
 - ▶ Transport layer at receiving acknowledges packet delivery.
 - ▶ Resends missing packets
- ▶ Flow and congestion control
 - ▶ Avoids congesting the receiver
 - ▶ Avoids congesting the network

Layer 5: Session Layer

- ▶ Allows two applications on different computers to establish, use, and end a session.
 - ▶ e.g. file transfer, remote login
- ▶ Establishes dialog control
 - ▶ Regulates which side transmits, plus when and how long it transmits.
- ▶ Performs *token management* and *synchronization*.

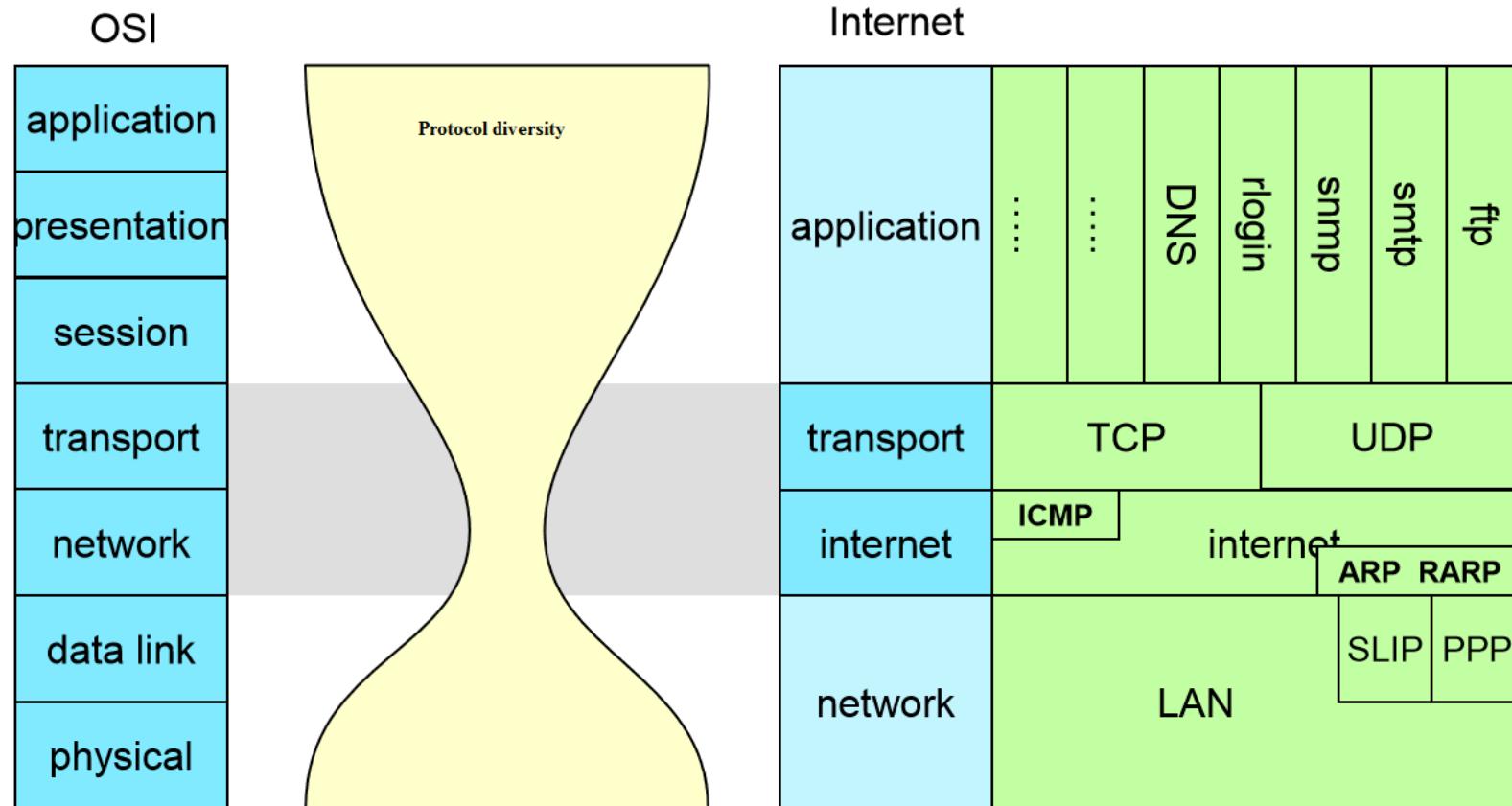
Layer 6: Presentation Layer

- ▶ Related to representation of transmitted data
 - ▶ Translates different data representations from the Application layer into uniform standard format
- ▶ Providing services for secure efficient data transmission
 - ▶ e.g. data encryption, and data compression.

Layer 7: Application Layer

- ▶ Level at which applications access network services.
 - ▶ Represents services that directly support software applications for file transfers, database access, and electronic mail etc.
 - ▶ Offers application-specific functionality but does not contain the applications themselves.

Internet Protocols vs. OSI

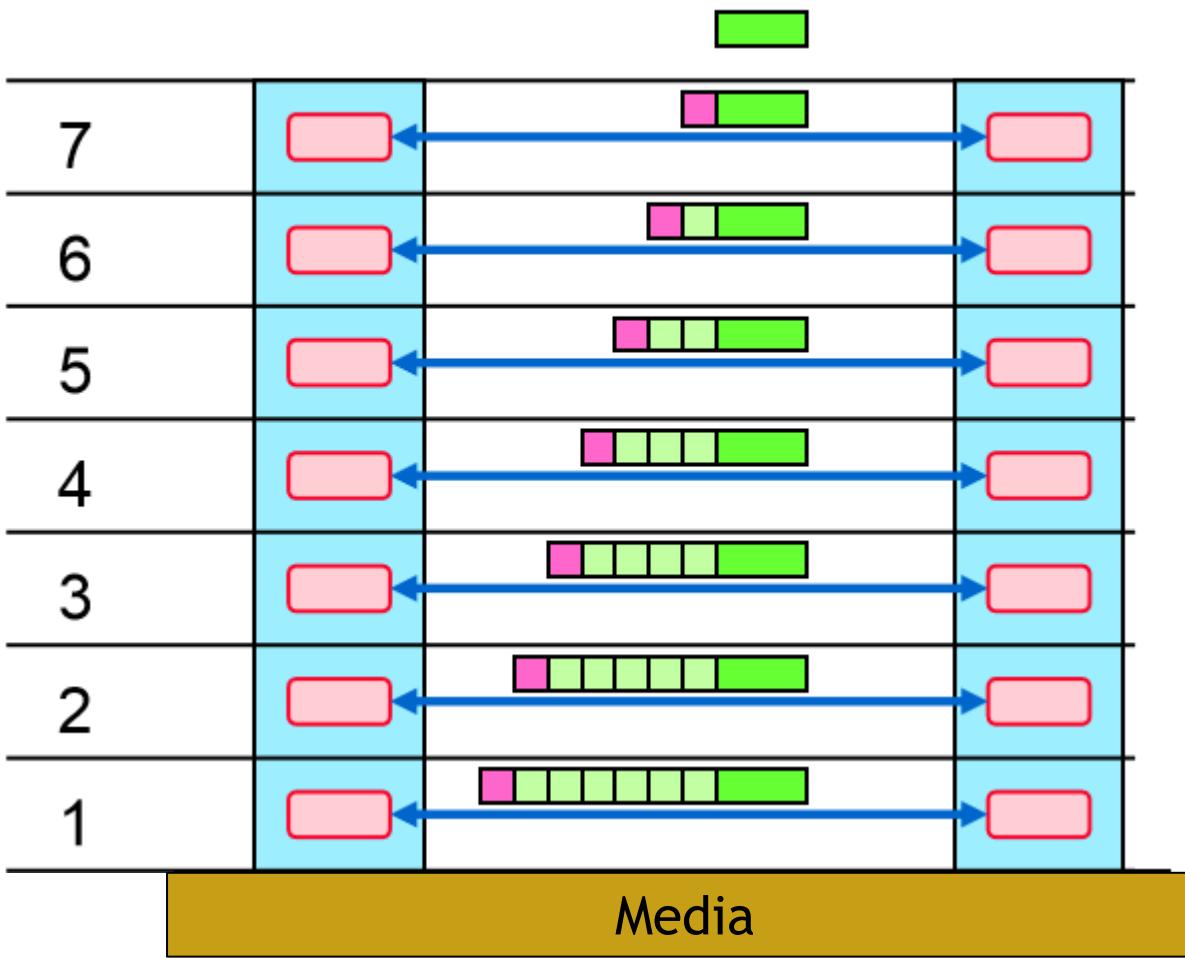


- ▶ Explicit presentation and session layers missing in Internet Protocols
- ▶ Data Link and Network Layers redesigned

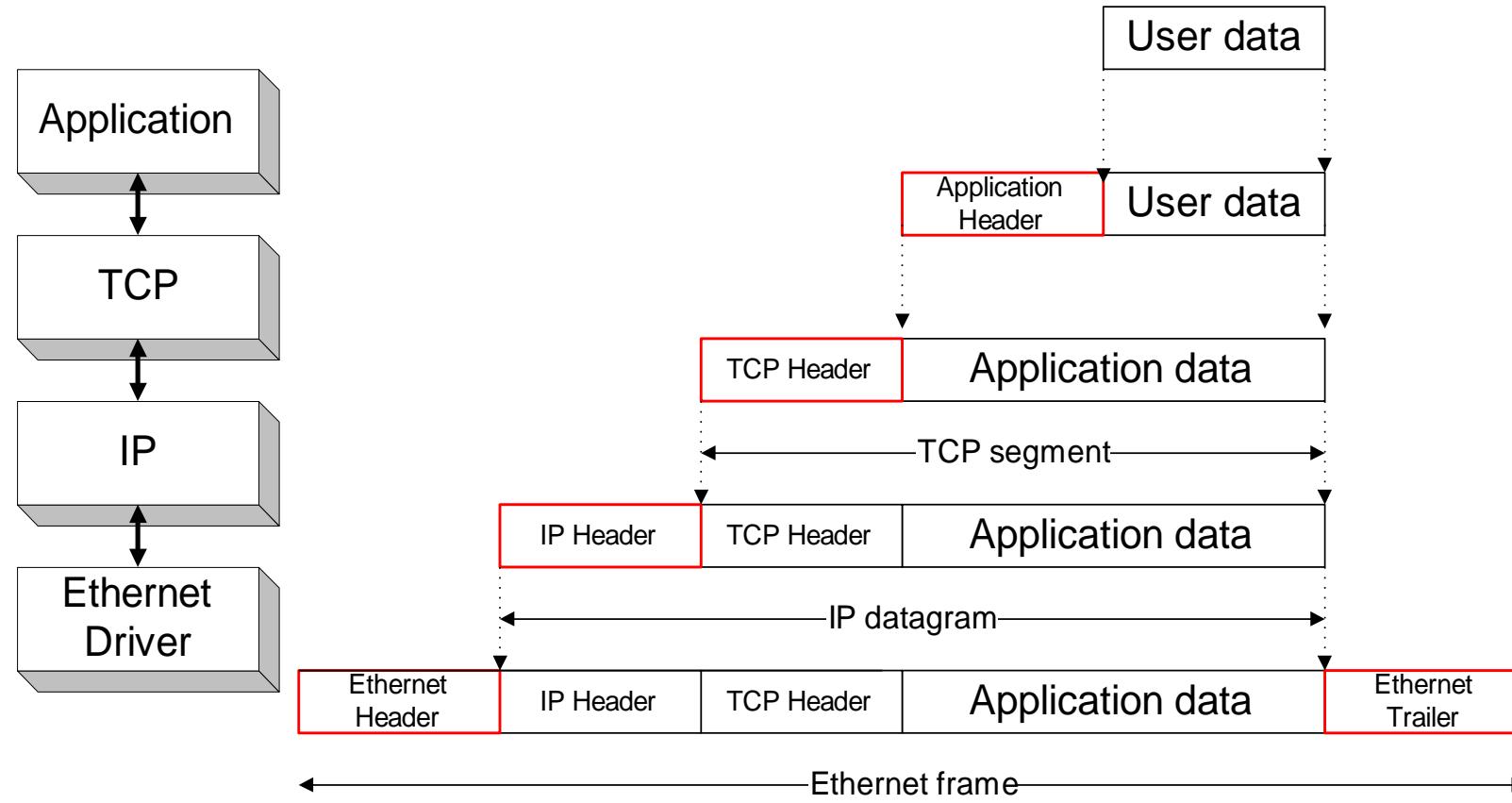
Data Encapsulation in OSI model (1)

- ▶ At each layer in the TCP/IP protocol stack
 - ▶ Outgoing data is packaged and identified for delivery to the layer underneath
- ▶ PDU - Packet Data Unit - the “envelop” information attached to a packet at a particular TCP/IP protocol
 - ▶ e.g. header and trailer
- ▶ Header
 - ▶ PDU’s own particular opening component
 - ▶ Identifies the protocol in use, the sender and intended recipient
 - ▶ The information header is used to assist in any of these tasks: routing of the object, flow control, error detection, error correction, etc.
- ▶ Trailer (or packet trailer)
 - ▶ Provides data integrity checks for the payload

Data Encapsulation in OSI model (2)



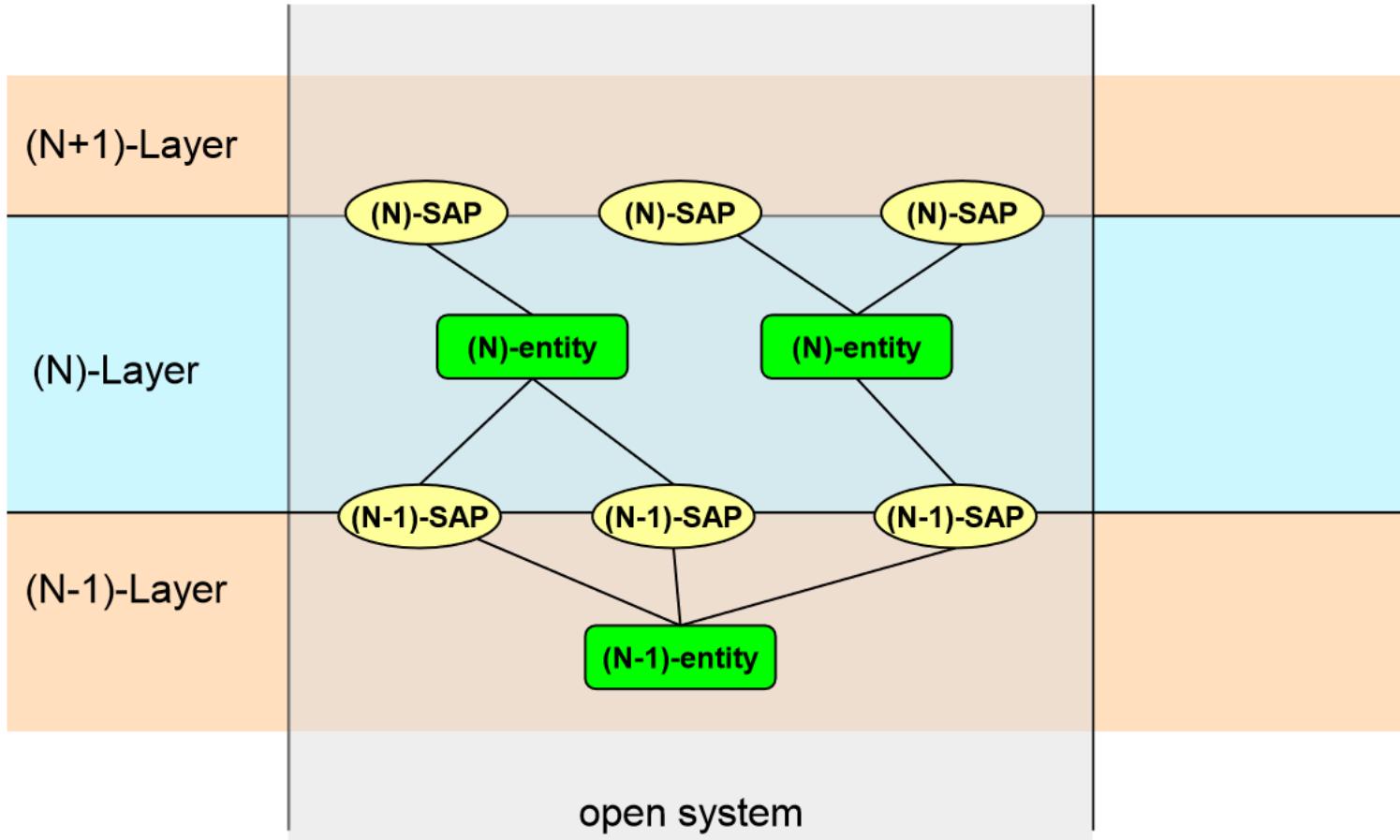
Encapsulation in the TCP/IP Suite



Addressing in the TCP/IP protocol suite

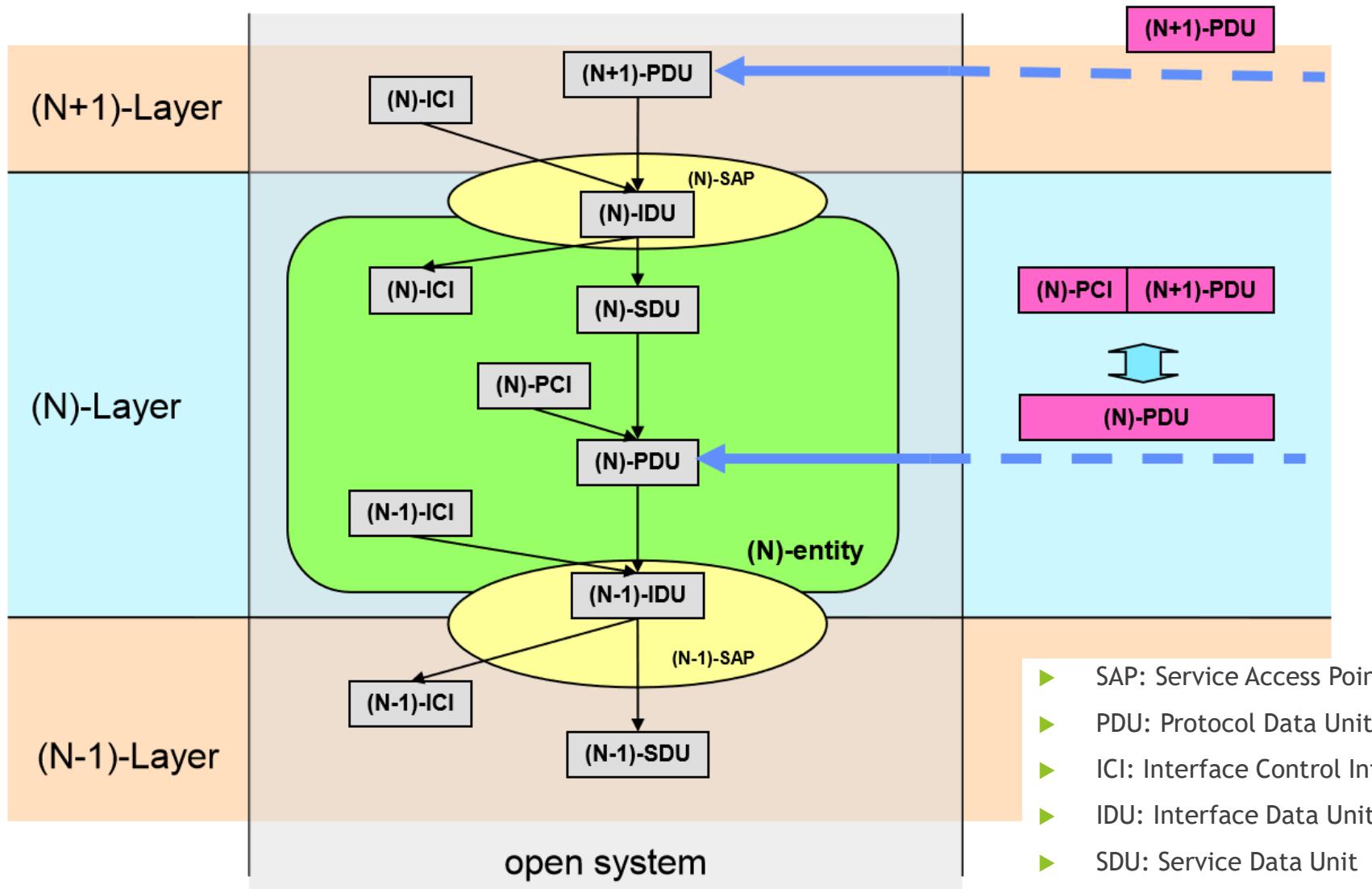
Packet names	Layers	Addresses
Message	Application layer	Names
Segment / User datagram	Transport layer	Port numbers
Datagram	Network layer	Logical addresses
Frame	Data-link layer	Link-layer addresses
Bits	Physical layer	

Layering Principles (1)



► SAP: Service Access Point

Layering Principles (2)



Services and Protocols in the OSI Model

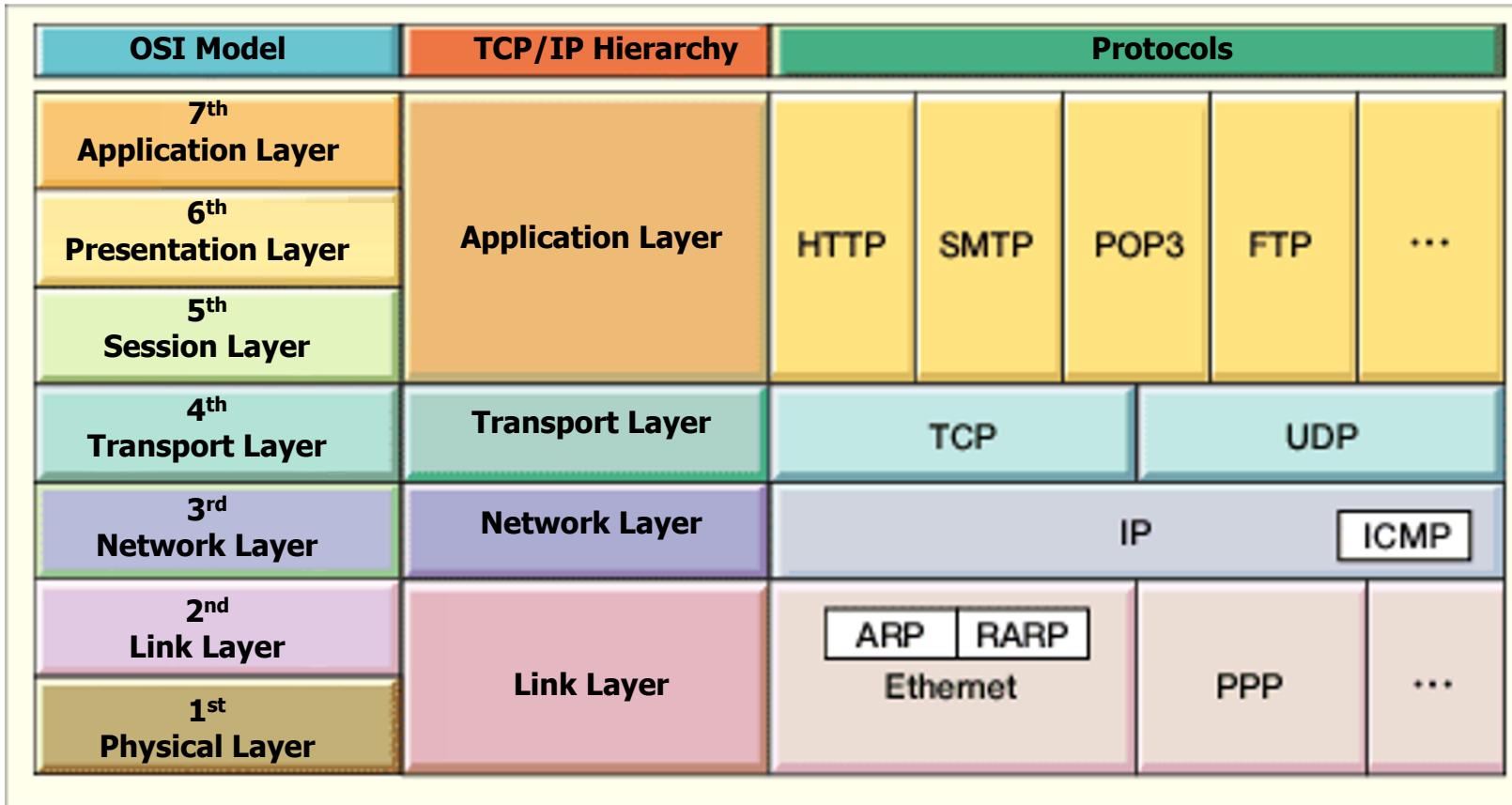
- ▶ Services
 - In OSI model, each layer provide services to layer above, and ‘consumes’ services provided by layer below.
 - Active elements in a layer called *entities*.
 - Entities in same layer in different machines called *peer entities*.

- ▶ Protocols
 - In diplomatic circles, a protocol is the set of rules governing a conversation between people
 - A network protocol is the set of rules governing a conversation between a client and a server
 - -> **Syntax, Semantic and Timing**
 - A standard protocol is described in RFCs (Requests for Comments)

Service vs. Protocol

- ▶ Service = set of primitives provided by one layer to layer above.
- ▶ Service defines what layer can do (but not how it does it).
- ▶ Protocol = set of rules governing data communication between peer entities, i.e. format and meaning of frames/packets.
- ▶ Service/protocol decoupling very important.

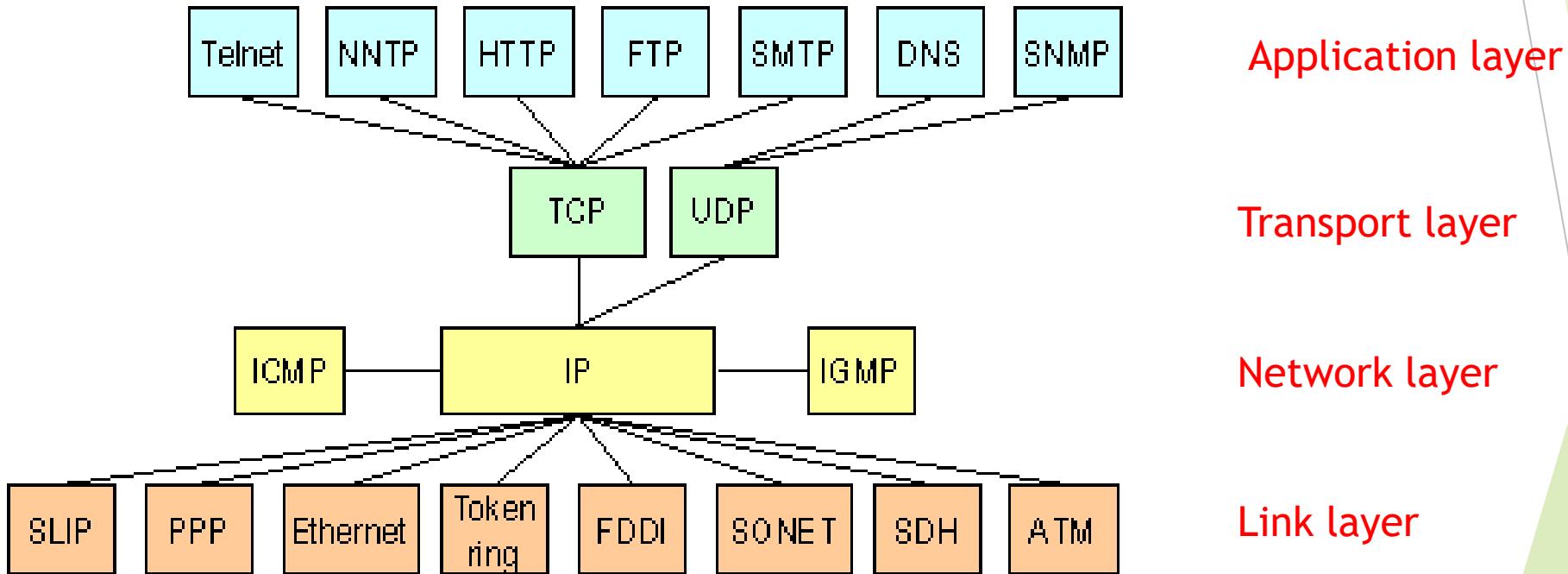
TCP/IP Protocol Stack



- Link Layer : includes device driver and network interface card
- Network Layer : handles the movement of packets, i.e. routing
- Transport Layer : provides a reliable flow of data between two hosts
- Application Layer : handles the details of the particular application

- ▶ A protocol stack is a group of protocols that are running concurrently and are employed for the implementation of network protocol suite
- ▶ Every operating system has a protocol stack built in.
- ▶ The protocol stack used on the internet is TCP/IP.

Some Popular TCP/IP Protocols



Introduction to Ethernet

Presenter:
Assoc.Prof.Dr. Fatih ABUT

Outline

- Ethernet Intro
- CSMA/CD protocol
- Ethernet Framing
- Exponential backoff
- Repeater, Hub, Bridge, Switch

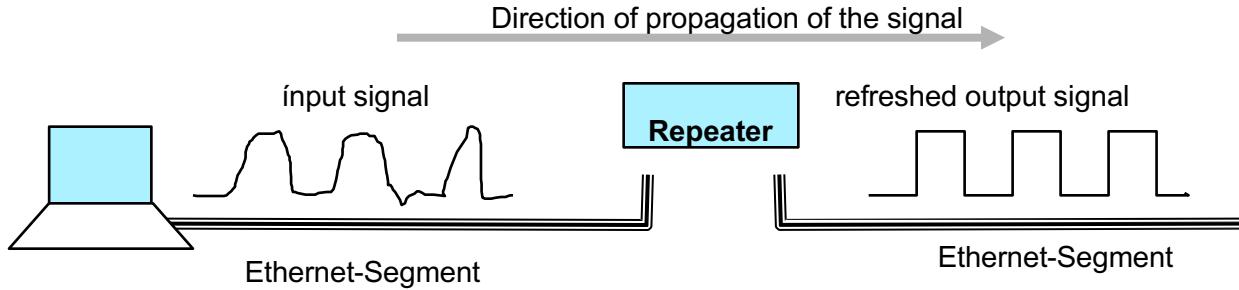
Introduction

- ▶ History
 - ▶ Developed by Bob Metcalfe and others at Xerox PARC in mid-1970s
 - ▶ Standardized by Xerox, DEC, and Intel in 1978
 - ▶ LAN standards define MAC and physical layer connectivity
 - ▶ IEEE 802.3 (CSMA/CD - Ethernet) standard - originally 2Mbps
 - ▶ IEEE 802.3u standard for 100Mbps Ethernet
 - ▶ IEEE 802.3z standard for 1,000Mbps Ethernet
- ▶ CSMA/CD: Ethernet's Media Access Control (MAC) policy
 - ▶ CS = carrier sense
 - ▶ Send only if medium is idle
 - ▶ MA = multiple access
 - ▶ CD = collision detection
 - ▶ Stop sending immediately if collision is detected

Ethernet Overview

- ▶ Most popular packet-switched LAN technology
- ▶ Ethernet is inexpensive, fast and easy to administer!
- ▶ Bandwidths: 10Mbps, 100Mbps, 1Gbps
- ▶ Maximum cabling distance of 100 metres.
 - ▶ The distance can be extended via repeaters.
- ▶ Bus and Star topologies are used to connect hosts
 - ▶ Hosts attach to network via Ethernet transceiver or hub or switch
 - ▶ Detects line state and sends/receives signals
 - ▶ Hubs are used to facilitate shared connections
 - ▶ All hosts on an Ethernet are competing for access to the medium
 - ▶ Switches break this model
- ▶ Problem: Distributed algorithm that provides fair access

Repeater



Functions:

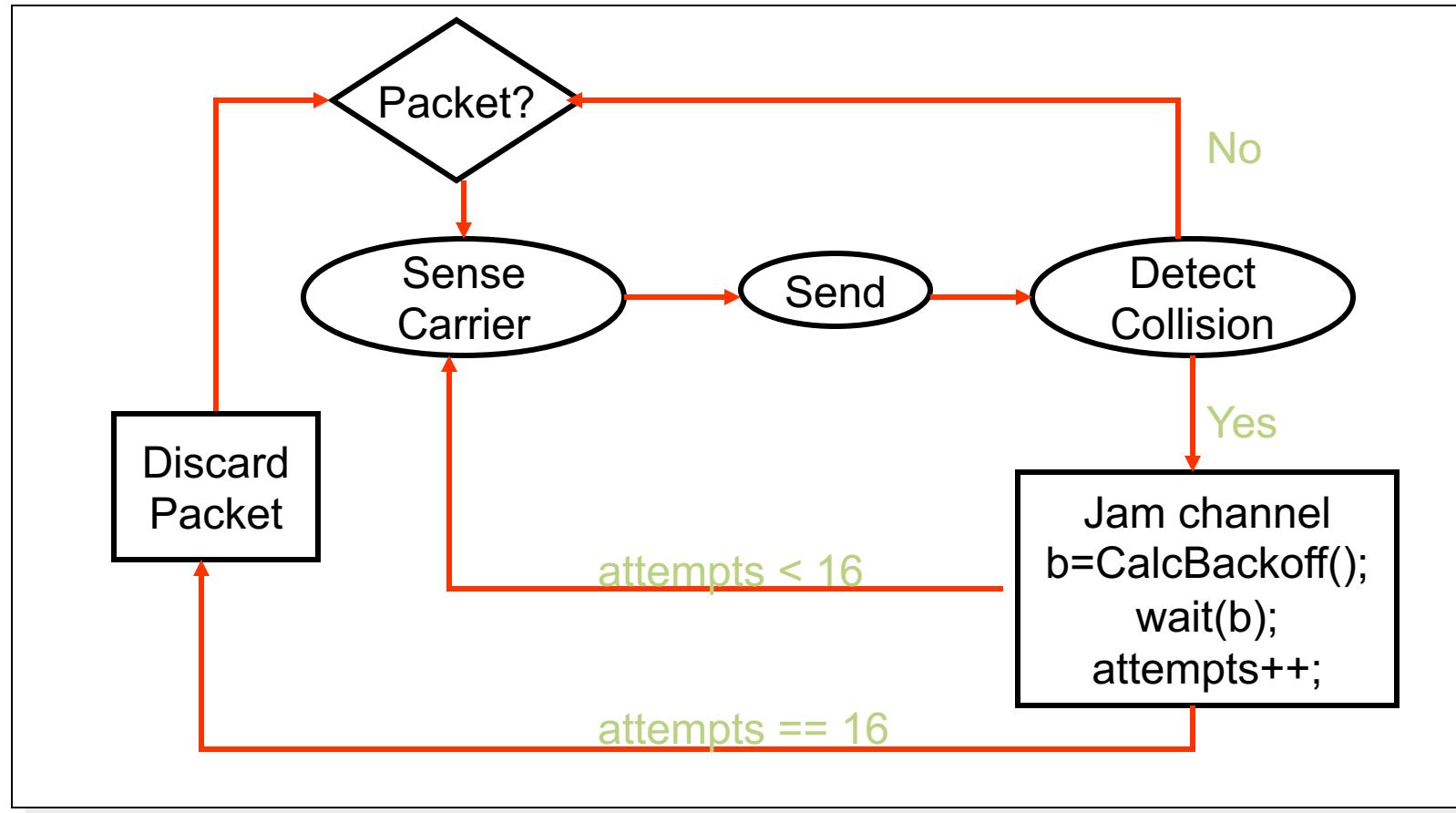
- ▶ timed signal regeneration,
- ▶ Generation or forwarding of a jam signal to signal collisions,
- ▶ Separation of defective cable segments.

→ Collisions are not limited !!

Ethernet's MAC Algorithm

- ▶ Ethernet uses CSMA/CD - listens to line before/during sending
- ▶ If line is idle (no carrier sensed)
 - ▶ send packet immediately
 - ▶ upper bound message size of 1500 bytes
 - ▶ must wait 9.6us between back-to-back frames
- ▶ If line is busy (carrier sensed)
 - ▶ wait until idle and transmit packet immediately
 - ▶ called *1-persistent* sending
- ▶ If collision detected
 - ▶ Stop sending and jam signal
 - ▶ Try again later

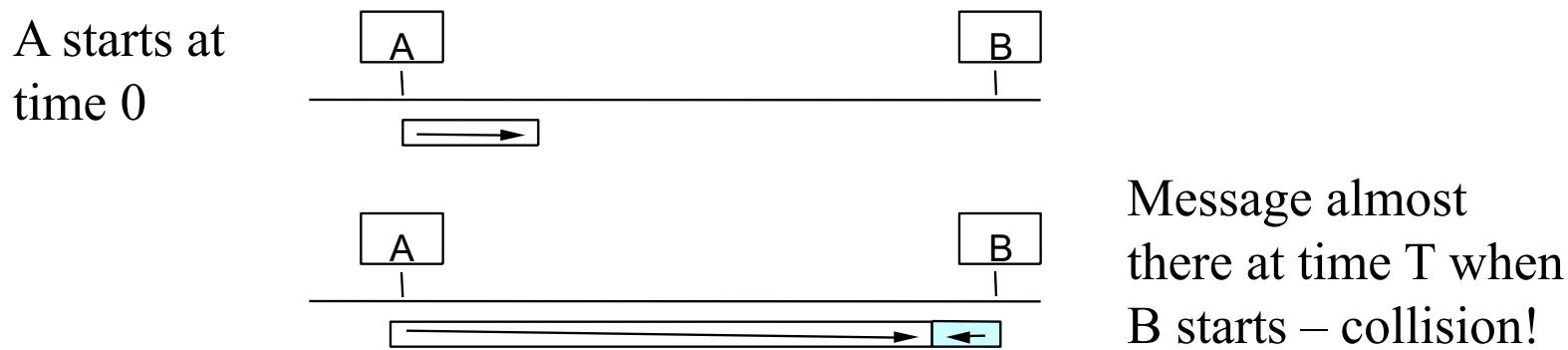
State Diagram for CSMA/CD



Collisions

Collisions are caused when two adaptors transmit at the same time (adaptors sense collision based on voltage differences)

- Both found line to be idle
- Both had been waiting to for a busy line to become idle



How can we be sure A knows about the collision?

$$\frac{\text{Max_Round_Trip_Delay}}{\text{Transmission_time_for_minimal_frames}} > ? \quad 1$$

Binary exponential backoff algorithm

- ▶ The basic waiting time S ("slot time") is the transmission time of a minimum Ethernet frame (64 bytes)
- ▶ Collision counter k counts the number of collisions.
- ▶ The back-off time is calculated as follows
 - ▶ Back-off time = $n * S$,
where n random element from $\{i: 0 \leq i < 2^k \text{ and } i \leq 2^{10}\}$
 - ▶ Example:
 $k = 1: \rightarrow i$ randomly selected from $\{0, 1\} \rightarrow$ back-off time = $0 * S$ or $1 * S$
 $k = 2: \rightarrow i$ randomly selected from $\{0, 1, 2, 3\} \rightarrow$ back-off time = $0 * S, 1 * S, 2 * S$ or $3 * S$
- ▶ If there are more than 16 collisions: Abort (i.e. CSMA / CD "gives up"). An error is reported to the layer above.

Ethernet address (1)

- ▶ Also called "MAC address"
- ▶ Globally unique ID for each device
- ▶ Burnt into ROM, cannot be modified
- ▶ 6 Bytes in which manufacturer, device model and serial number are coded
- ▶ Readable with many auxiliary tools e.g. ipconfig /all, ifconfig

- ▶ More specifically:
 - ▶ A 12-digit hexadecimal number (6-Byte binary number), which is mostly represented by Colon-Hexadecimal notation (e.g., 3C:D9:2B:DA:71:13)
 - ▶ First 6-digits (say 00:40:96) of MAC Address identifies the manufacturer, called as OUI (Organizational Unique Identifier).
 - ▶ The rightmost 6 digits represents **Network Interface Controller**, which is assigned by manufacturer.

CC:46:D6 - Cisco

3C:5A:B4 - Google, Inc.

3C:D9:2B - Hewlett Packard

00:9A:CD - HUAWEI TECHNOLOGIES CO., LTD

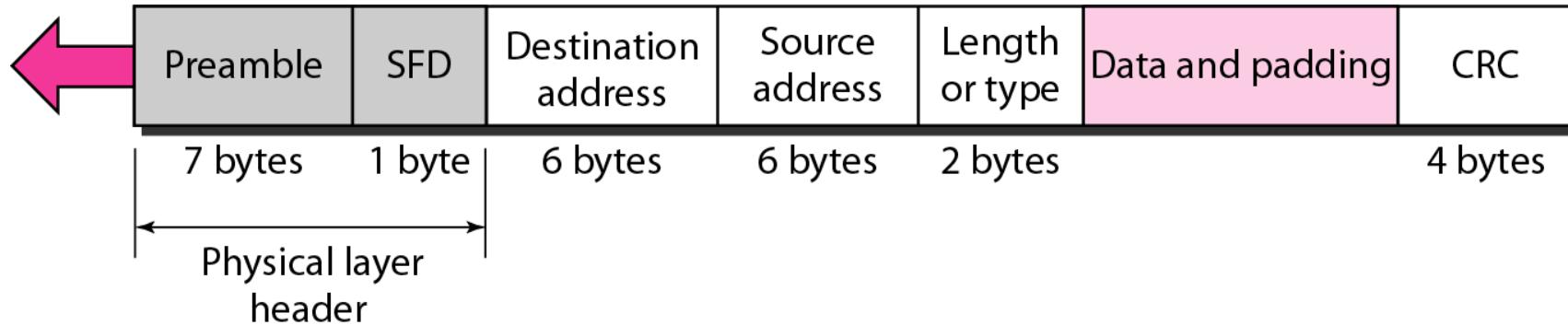
Ethernet address (2)

- ▶ The Layer 2 traffic can be classified as
 - ▶ unicast (one to one),
 - ▶ multicast (one to many), and
 - ▶ broadcast (one to all).
- ▶ Unicasts, Multicasts and Broadcasts are different types of network communication and are required for the normal operation of the network.
- ▶ MAC addresses for broadcast and multicast are given below.
 - ▶ Broadcast Destination MAC address - **FF:FF:FF:FF:FF:FF**
 - ▶ Multicast Destination MAC addresses - **01:00:5E:00:00:00** to **01:00:5E:7F:FF:FF**
- ▶ In case of a broadcast and multicast switch need to forward the Ethernet frame out all its ports.

Ethernet 802.3 MAC frame

Preamble: 56 bits of alternating 1s and 0s.

SFD: Start frame delimiter, flag (10101011)



Ethernet frame

- **Preamble**
Trailer consisting of the bit sequence “0101010101...” serving the bit synchronization of the receiver.
- **SFD (Start Frame Delimiter)**
Start character consisting of the bit pattern “10101011” showing the recipient that the actual information will follow now.
- **DA (Destination Address)**
Evaluated by the recipient’s address filter; only data frames destined for this recipient will be passed on to the communication software.
- **SA (Source Address)**
Sender’s address
- **LEN (Length) or EtherType**
LEN Indicates the length of the subsequent data field in Bytes according to IEEE 802.3. Similarly, EtherType is a two-octet field used to indicate which protocol is encapsulated in the payload of the frame and is used at the receiving end by the data link layer to determine how the payload is processed.

Ethernet frame

- **Data and Pad**

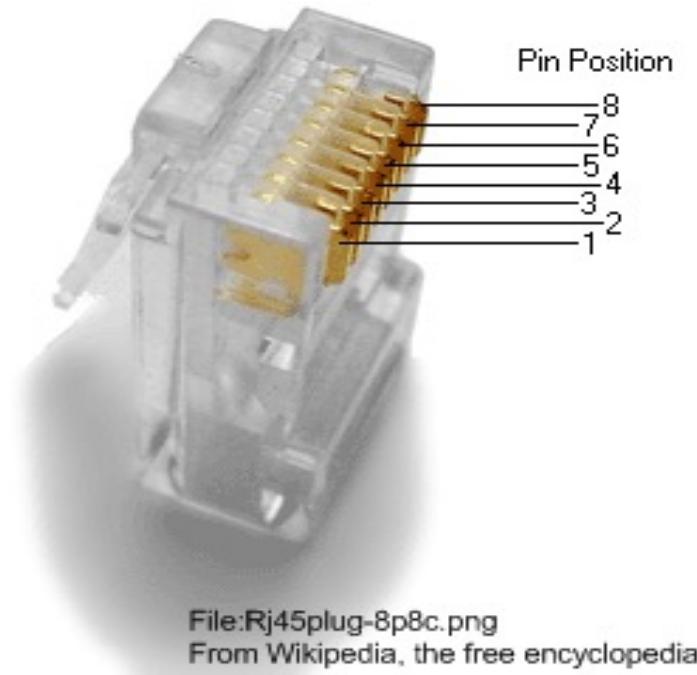
The data field may contain 46 to 1500 user data bytes. Are there less than 46 bytes the Ethernet controller independently adds padding bytes, until the total amount (data + pad) is 46. This minimum length is crucial for the CSMA/CD procedure to work faultlessly. The data field can be used at will, it only has to contain complete bytes.

- **CRC (cyclic redundancy check)**

A check character. It is obtained by taking the rest of the division operation from the formula representing the wide-spread cyclic-redundancy-check procedure. This formula is applied to the bit sequence including the address field through to the padding field. In case of an error the whole frame is ignored, i.e. not passed on to the application program.

Ethernet Cable

Color	Pin (T568B)
White/Orange	1
Orange	2
White/Green	3
Blue	4
White/Blue	5
Green	6
White/Brown	7
Brown	8



- You can use the order of rainbow colors to memorize the order of this wiring.

Ethernet Cabling (1)

- ▶ Straight Through
 - ▶ All order of the wirings is the same as the other side.



Straight-Through Cable

- Host to Switch or hub
- Router to Switch or hub

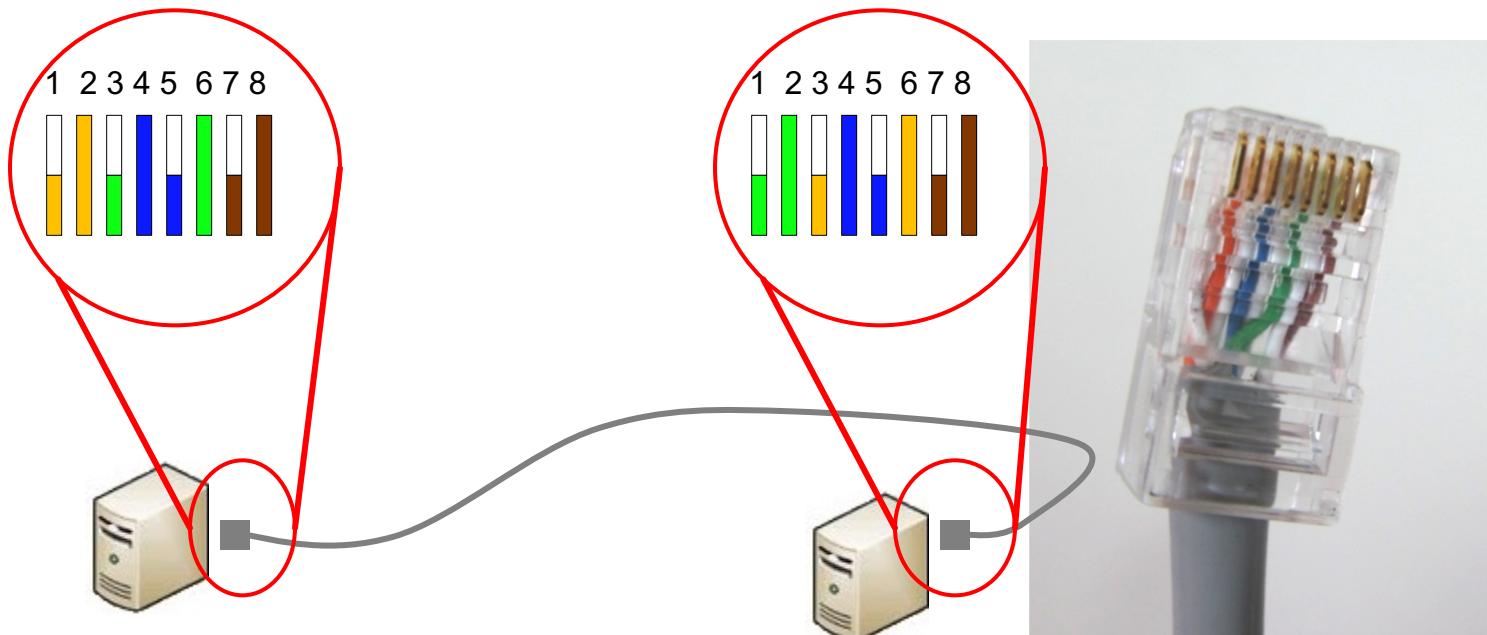
Ethernet Cabling (2)

► Crossover

- We need to change the order of the transmission and receiving wirings.

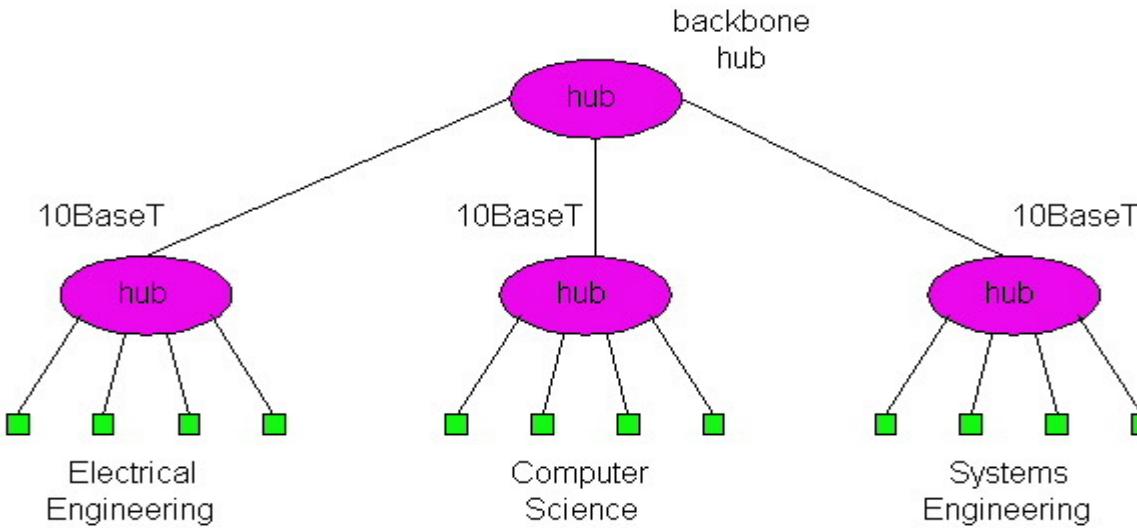
Crossover Cable

- Switch to Switch
- Hub to Hub
- Host to Host



Hubs (Multiport Repeaters) (1)

- ▶ Physical Layer devices: essentially repeaters operating at bit levels: repeat received bits on one interface to all other interfaces
- ▶ Hubs can be arranged in a **hierarchy** (or multi-tier design), with **backbone hub** at its top



Hubs (Multiport Repeaters) (2)

- ▶ Each connected LAN referred to as **LAN segment**
- ▶ Hubs **do not isolate** collision domains: node may collide with any node residing at any segment in LAN
- ▶ Hub Advantages:
 - ▶ simple, inexpensive device
 - ▶ Multi-tier provides graceful degradation: portions of the LAN continue to operate if one hub malfunctions
 - ▶ extends maximum distance between node pairs (100m per Hub)

Bridges (1)

- ▶ **Link Layer devices:** operate on Ethernet frames, examining frame header and selectively forwarding frame based on its destination
- ▶ **Isolates collision domains** resulting in higher total max throughput, and does not limit the number of nodes nor geographical coverage
- ▶ When frame is to be forwarded on segment, bridge uses CSMA/CD to access segment and transmit
- ▶ Transparent: no need for any change to hosts LAN adapters

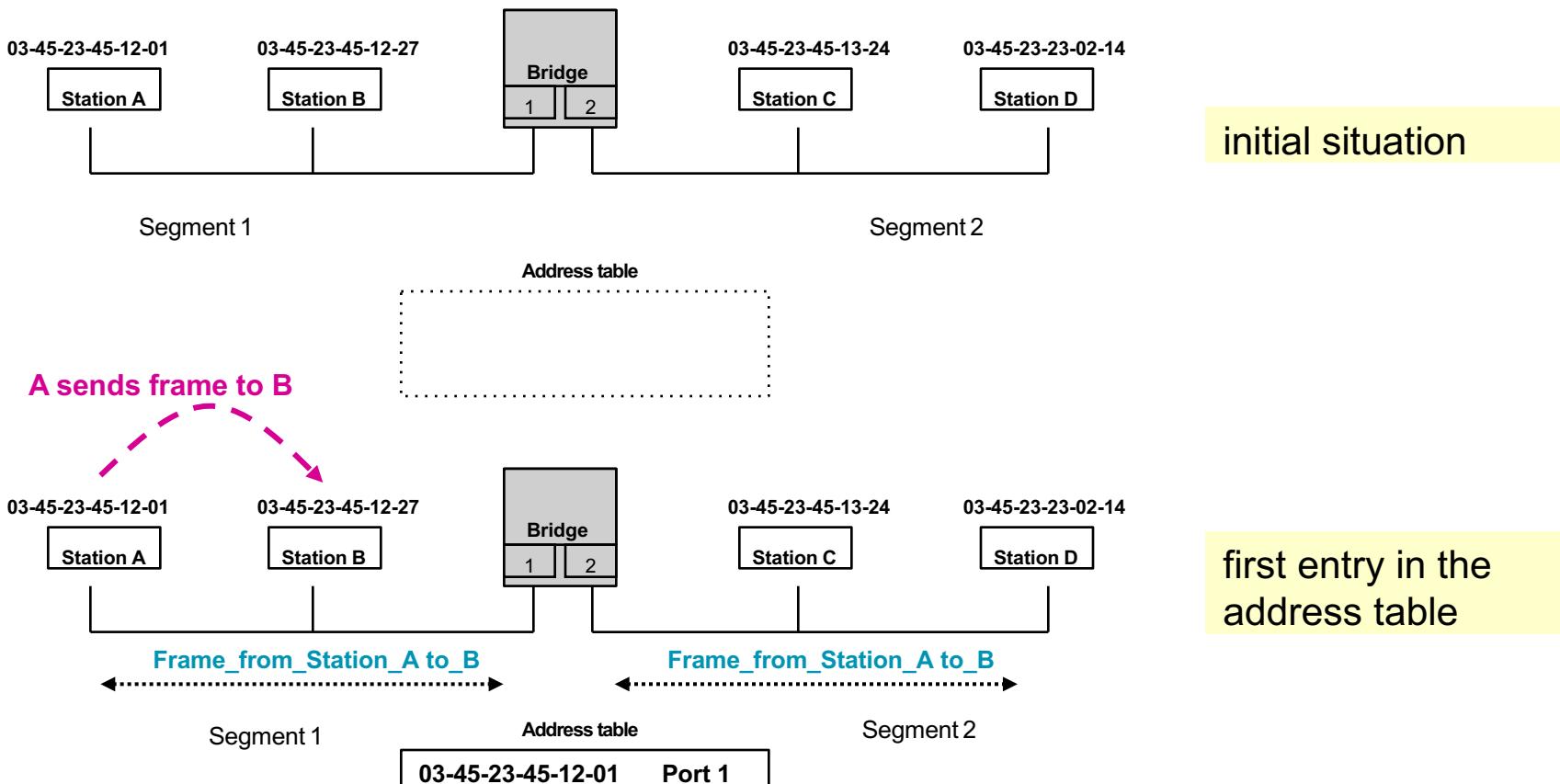
Bridges (2)

- ▶ Bridge advantages:
 - ▶ Isolates collision domains resulting in higher total max throughput, and does not limit the number of nodes nor geographical coverage

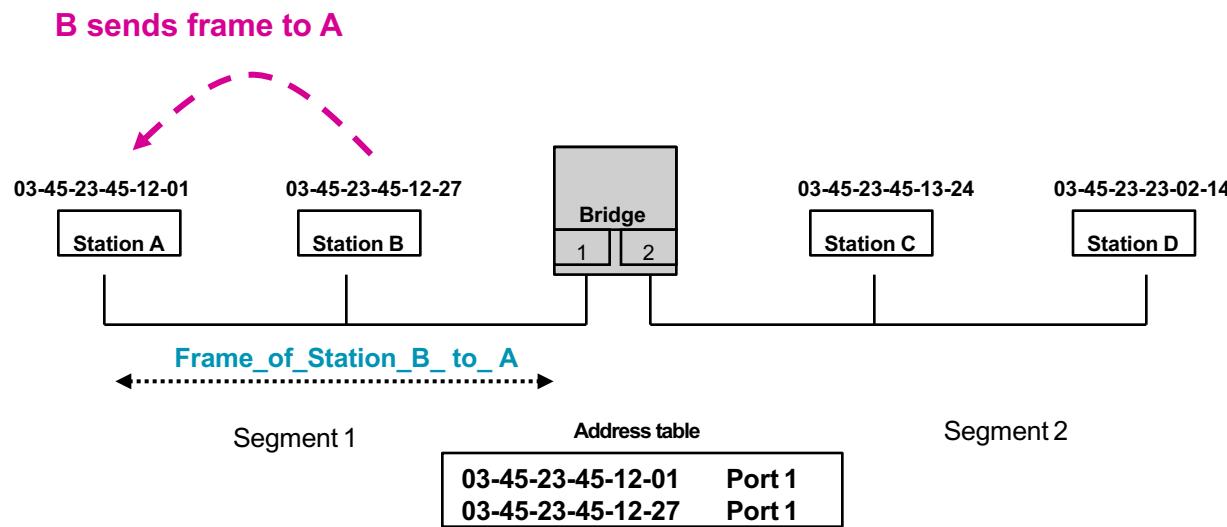
Bridge Filtering

- ▶ Bridges *learn* which hosts can be reached through which interfaces: maintain filtering tables
 - ▶ when frame received, bridge “learns” location of sender: incoming LAN segment
 - ▶ records sender location in filtering table
- ▶ Filtering table entry:
 - ▶ (Node LAN Address, Bridge Interface, Time Stamp)
 - ▶ stale entries in Filtering Table dropped

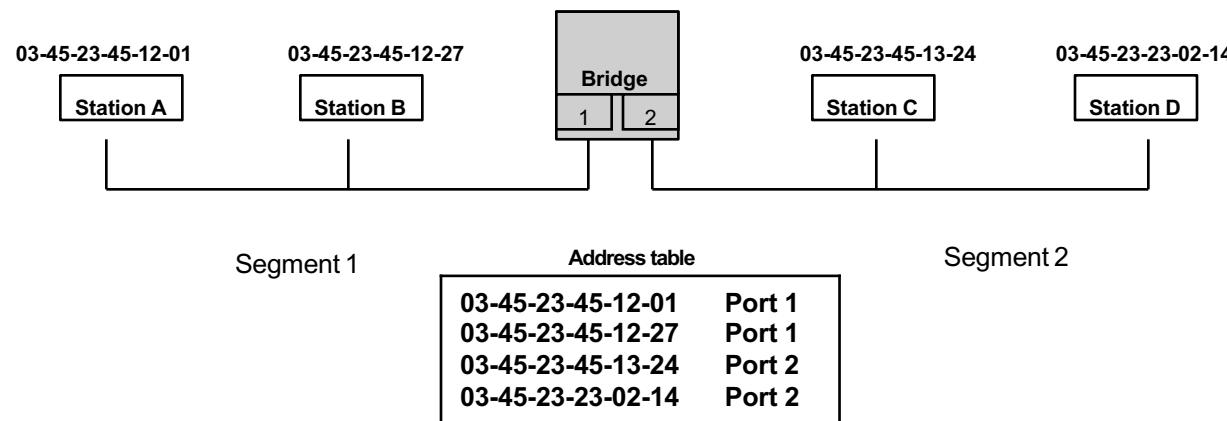
Bridge Learning: example (1)



Bridge Learning: example (2)



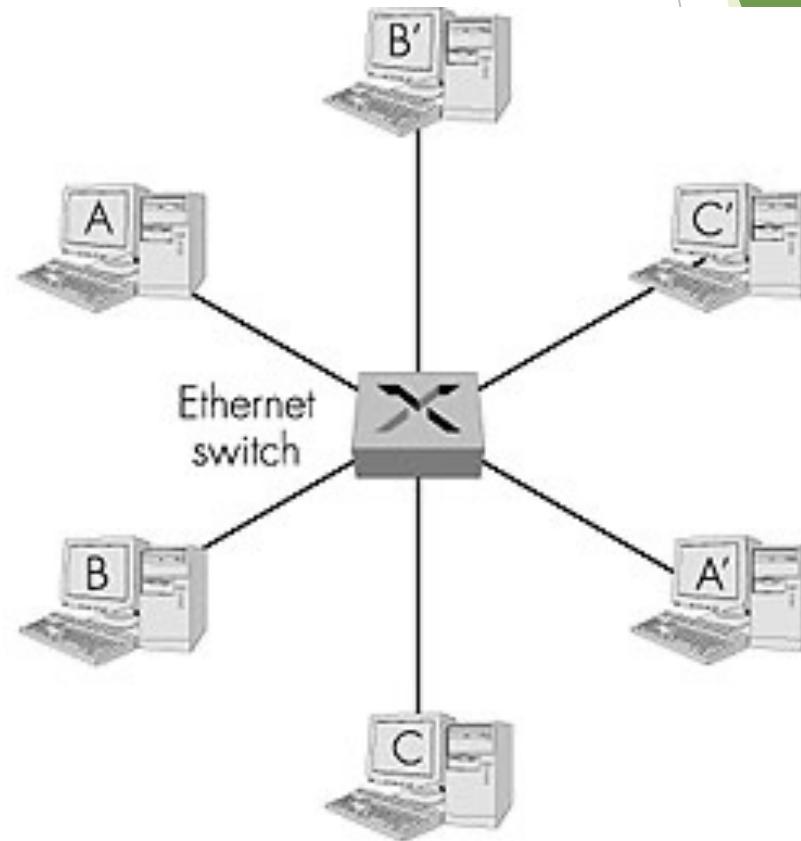
first filtering of a frame



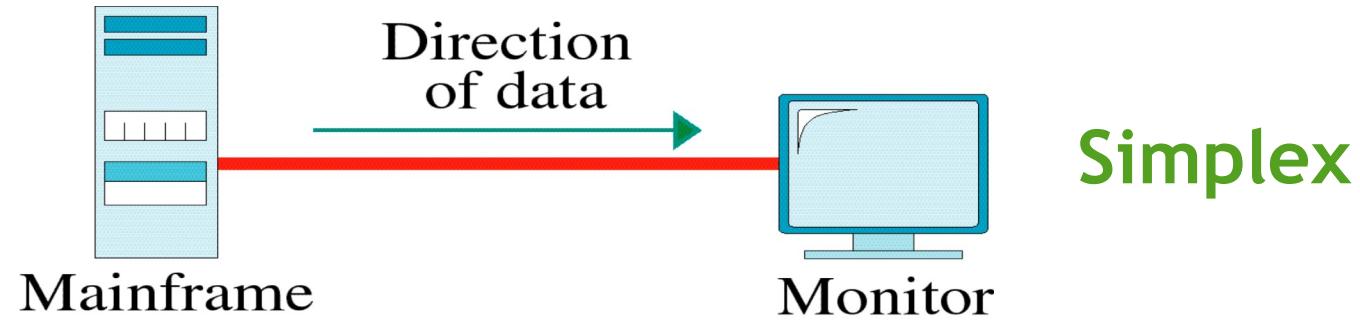
Address table after a complete learning process

Ethernet Switches (Multiport Bridges)

- ▶ Layer 2 (frame) forwarding, filtering using LAN addresses
- ▶ **Switching:** A-to-B and A'-to-B' simultaneously, no collisions
- ▶ Large number of interfaces
- ▶ Often: individual hosts, star-connected into switch
 - ▶ Ethernet, but no collisions!



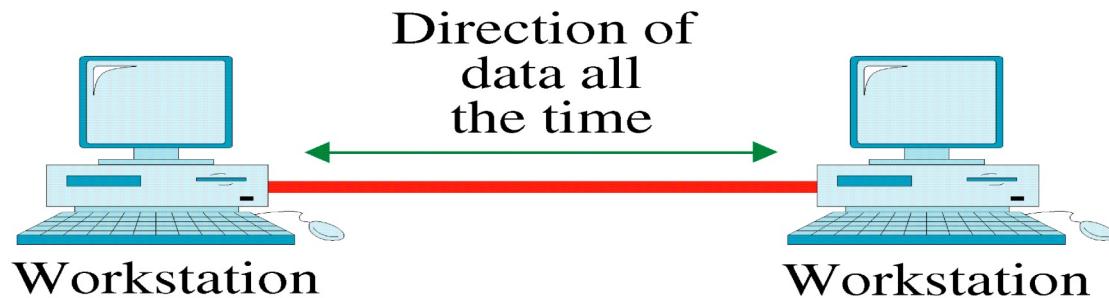
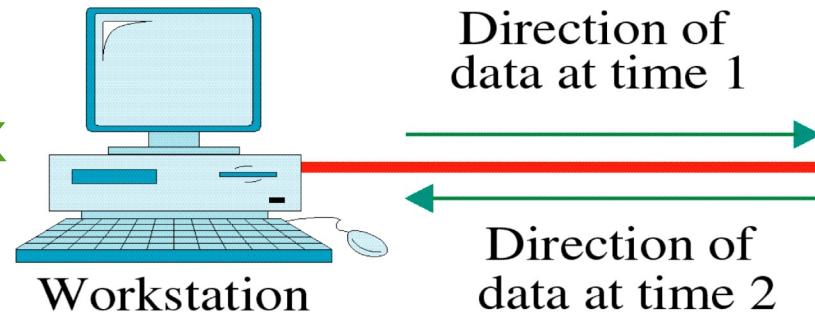
Transmission Mode



Simplex

- Transmission mode of
- ▶ repeater?
 - ▶ hub?
 - ▶ bridge/switch?
 - ▶ Switches have **autonegotiation** for speed mismatches
 - ▶ Modern devices can auto-sense speed and duplex.

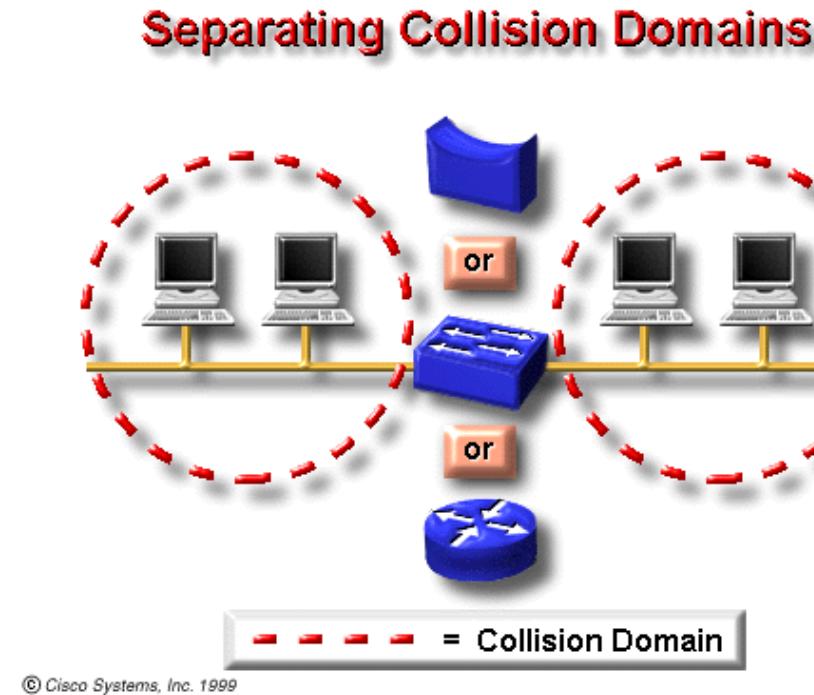
Half-duplex



Full-duplex

Collision Domain

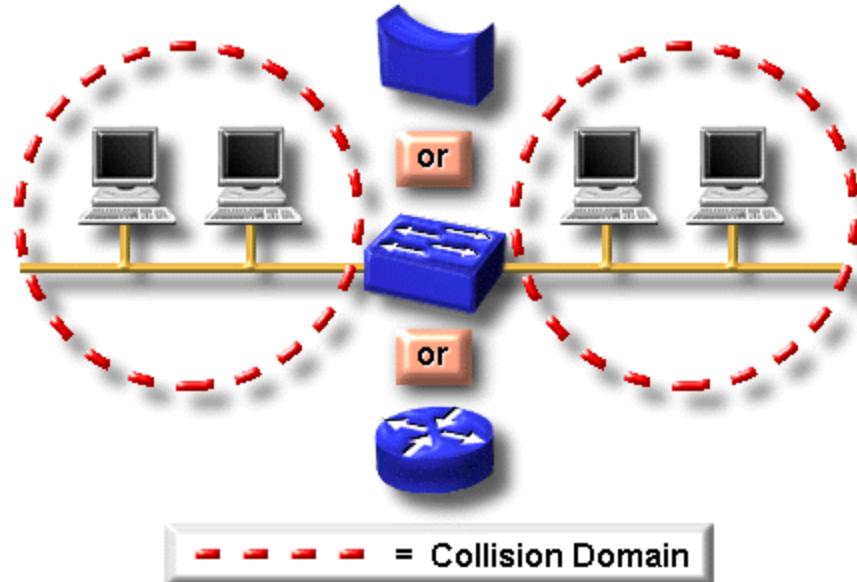
- ▶ Network region in which collisions are propagated.
- ▶ Repeaters and hubs propagate collisions.
- ▶ Bridges, switches and routers do not.



Reducing Collisions

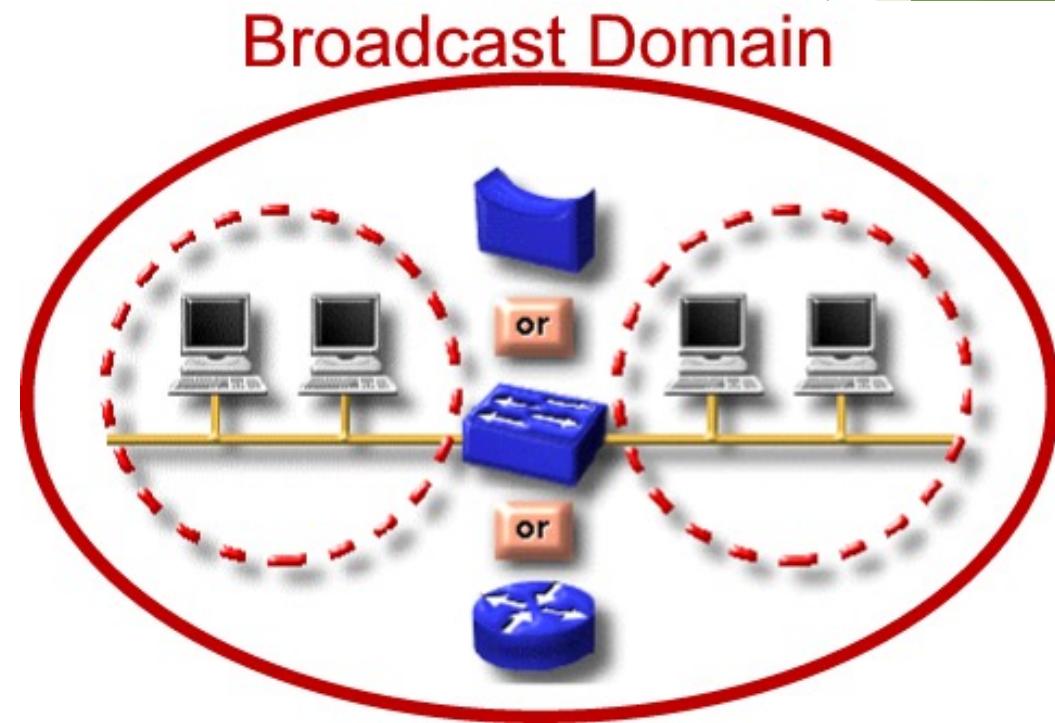
- ▶ Collision frequency can be kept low by breaking the network into segments bounded by:
 - ▶ bridges
 - ▶ switches
 - ▶ routers

Separating Collision Domains



Broadcast Domain

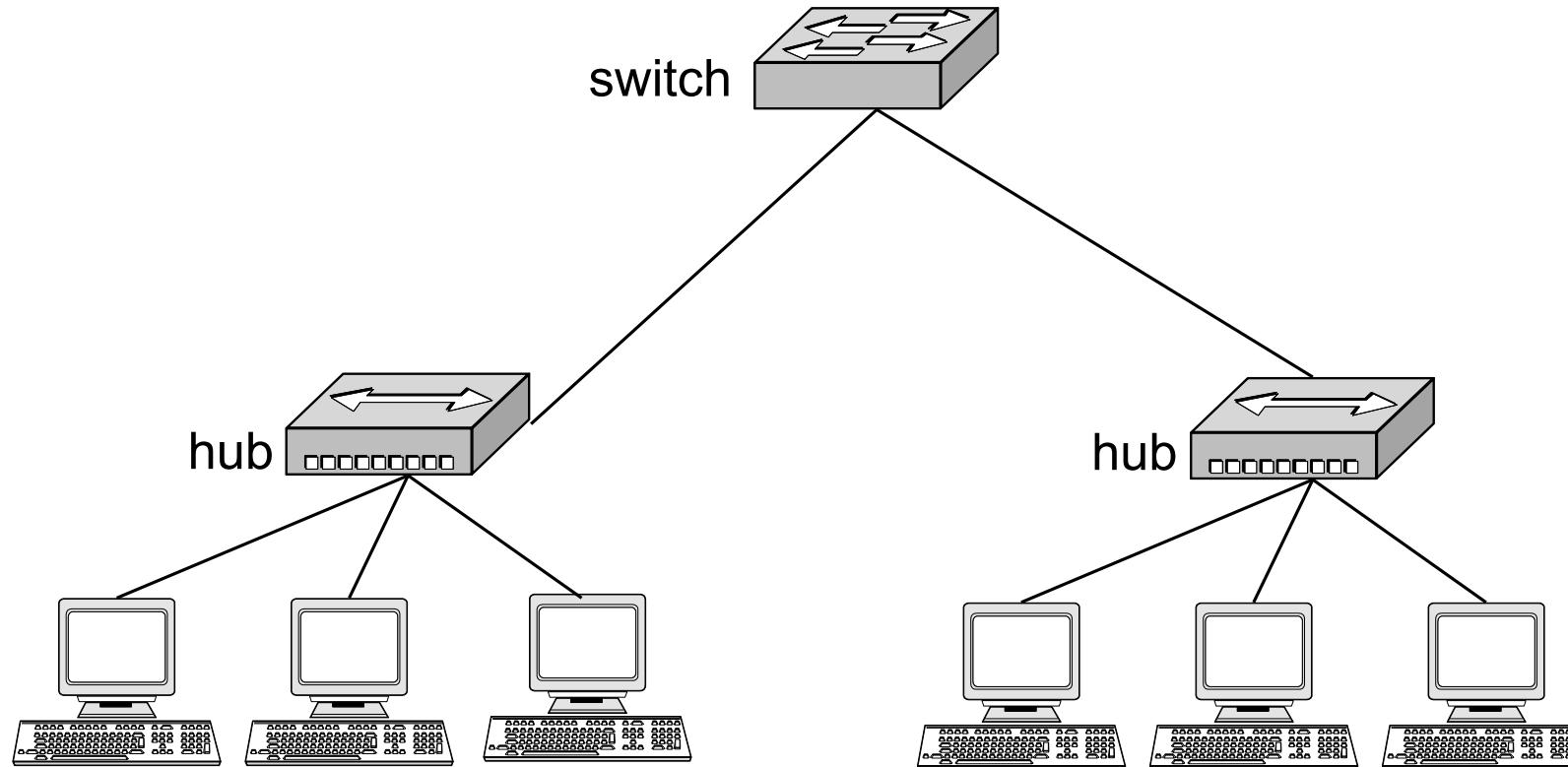
- ▶ Network region in which broadcast frames are propagated.
- ▶ Repeaters, hubs, bridges, & switches propagate broadcasts.
- ▶ Routers either do or don't, depending on their configuration.



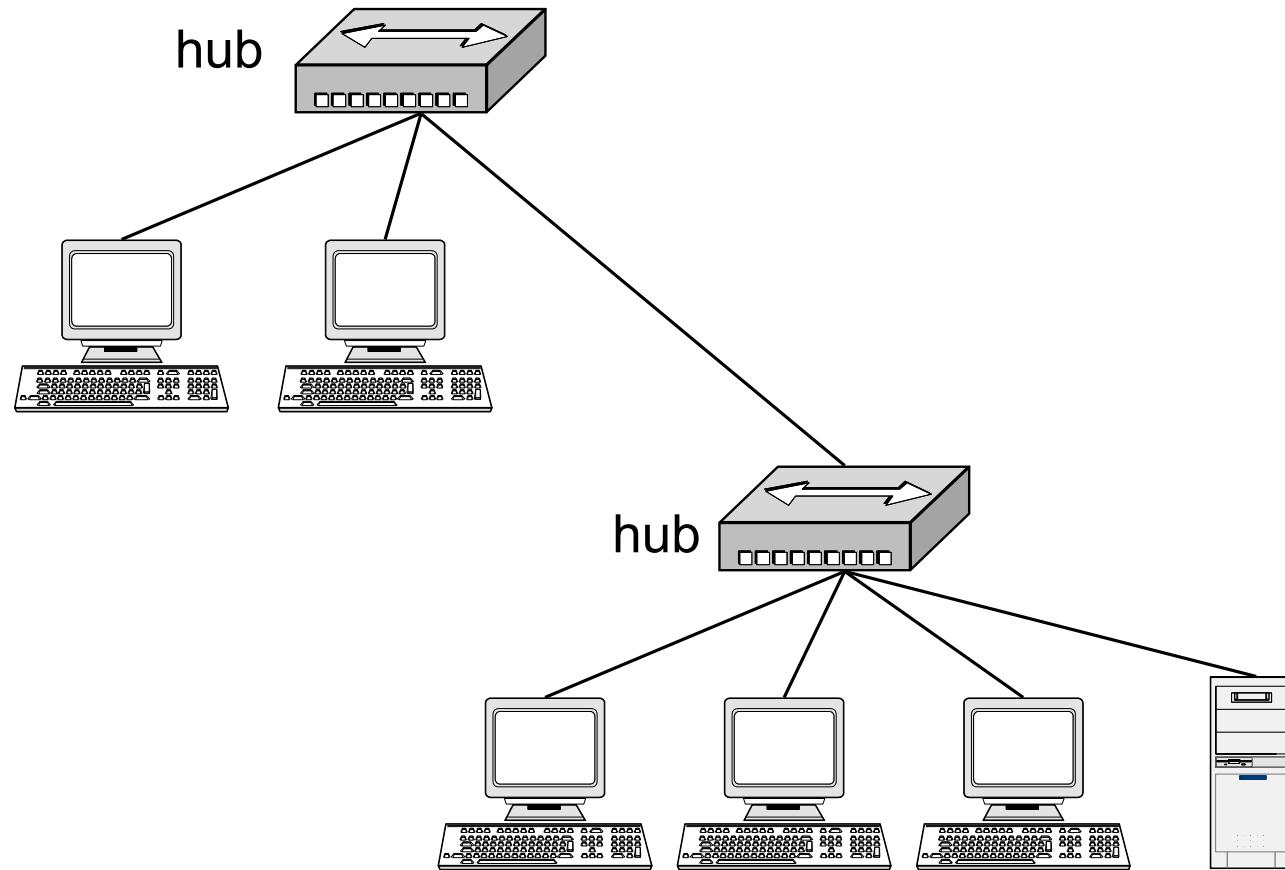
Reducing Broadcasts

- ▶ Broadcasts are necessary for network function.
- ▶ Some devices and protocols produce lots of broadcasts; avoid them.
- ▶ Broadcast frequency can be kept manageable by limiting the LAN size.
- ▶ LANs can then be cross-connected by routers to make a larger internetwork.

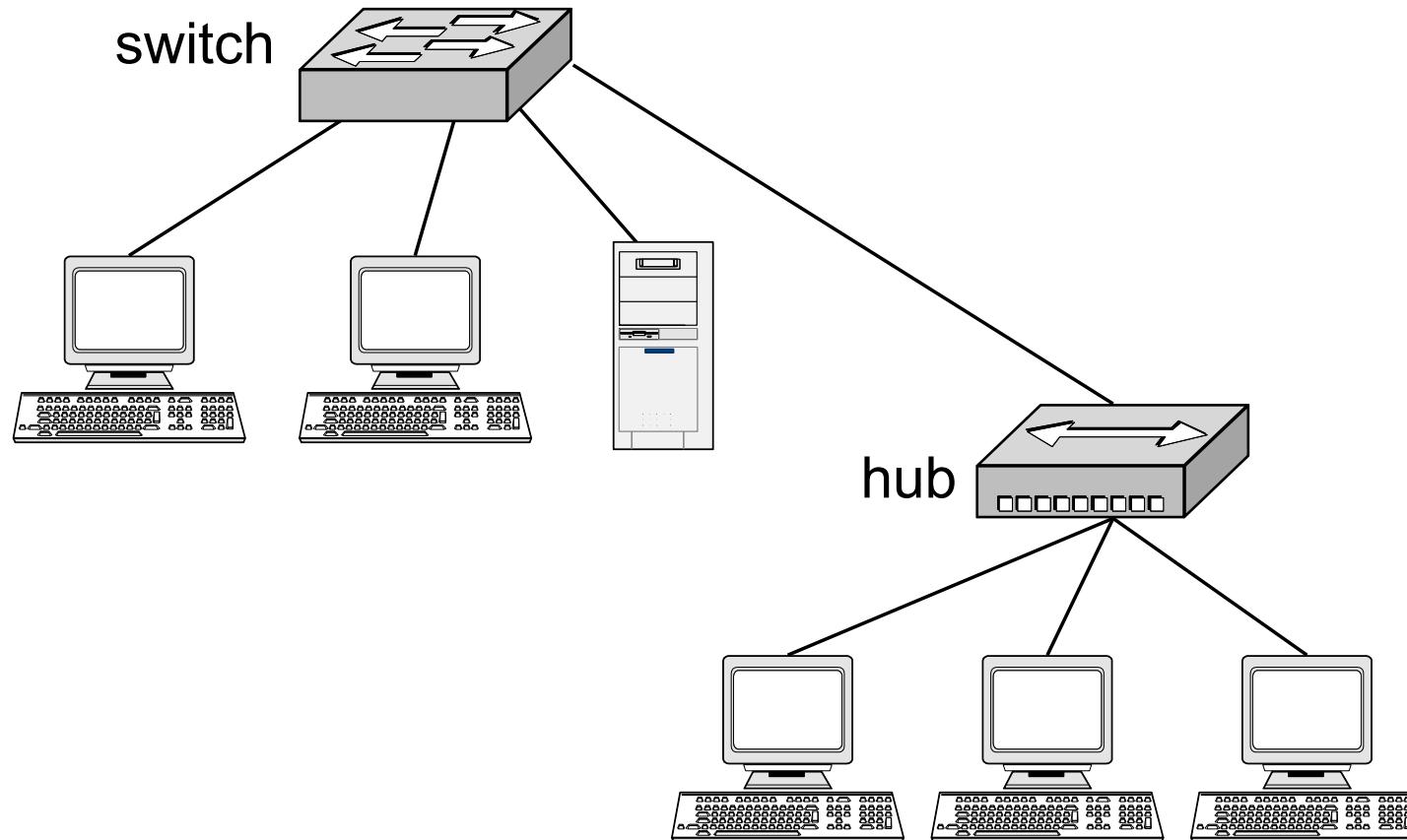
Identify the collision domains & broadcast domains:



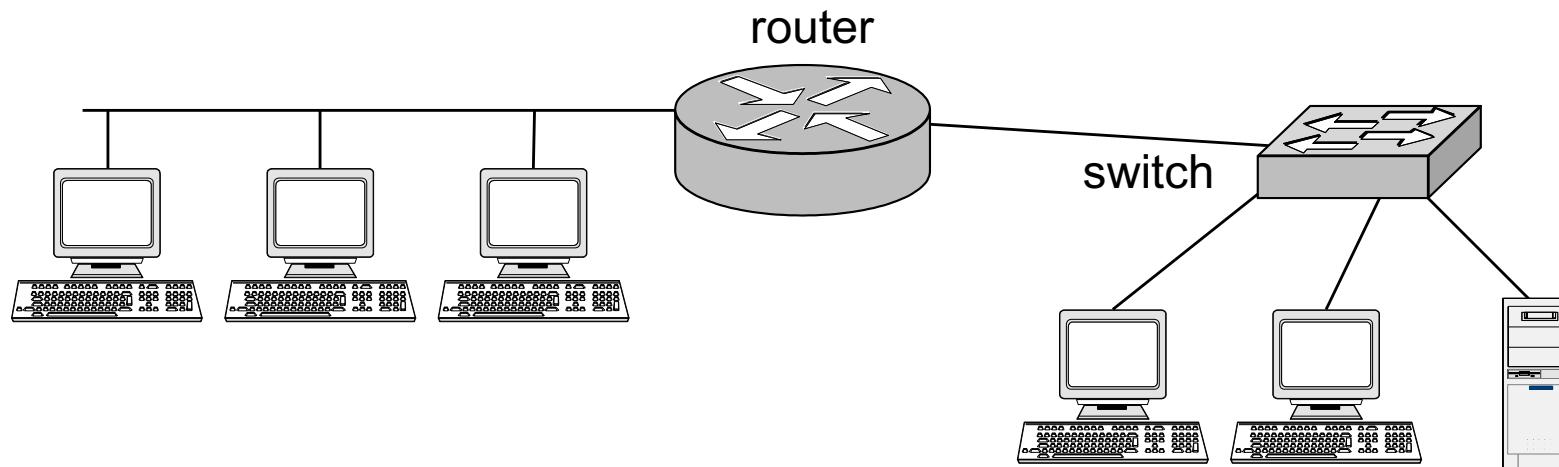
Identify the collision domains & broadcast domains:



Identify the collision domains & broadcast domains:



Identify the collision domains & broadcast domains:

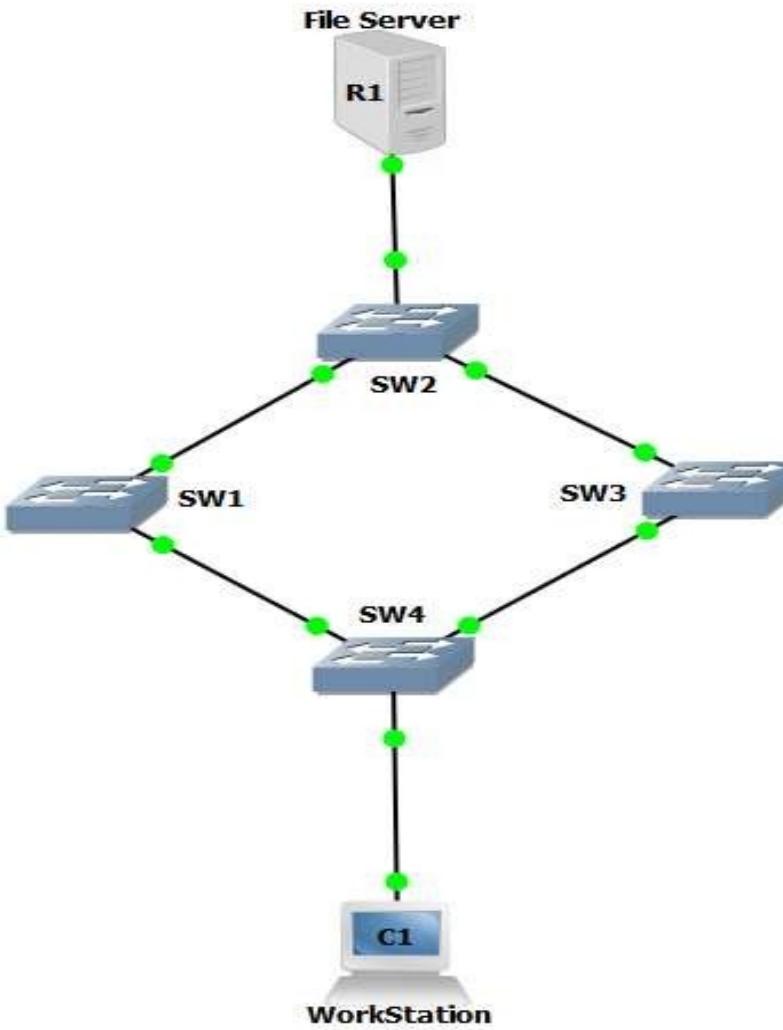


Router connects separate networks.
One broadcast domain per router interface.

Problem: Layer 2 Switching Loop (1)

- ▶ In practical local area networking, it is common that the switches are interconnected for redundancy.
- ▶ When switches are interconnected, the network will not fail completely even one if the connected link fails.
- ▶ When switches are interconnected for redundancy as shown in the topology, another serious network problem can occur, which is known as Layer 2 Switching loop.

Problem: Layer 2 Switching Loop (2)



Problem: Layer 2 Switching Loop (3)

- ▶ A Ethernet frame originating from Workstation to the File Server, first reaches the Switch 4.
- ▶ Switch 4 will forward the packet to all its ports (except the source port) since the MAC address of the destination device (File Server) may not be available in its MAC address table (File Server is attached to Switch 2).

Problem: Layer 2 Switching Loop (4)

- ▶ Both Switch 1 and Switch 3 will receive a copy of the Ethernet frame. Now the Switch 1 and Switch 3 will search for the destination MAC address in its MAC address table and if they fail to find the destination MAC address in their MAC address tables, both the Switches will forward the Ethernet frame to all the ports (except the source port). This may cause the Ethernet frame to reach back the Switch 4 via path Switch 1 - Switch 3 - Switch 4 or Switch 3 - Switch 1 - Switch 4.
- ▶ This may lead to a switching loop and the Ethernet frame will start circulating the network in a loop.

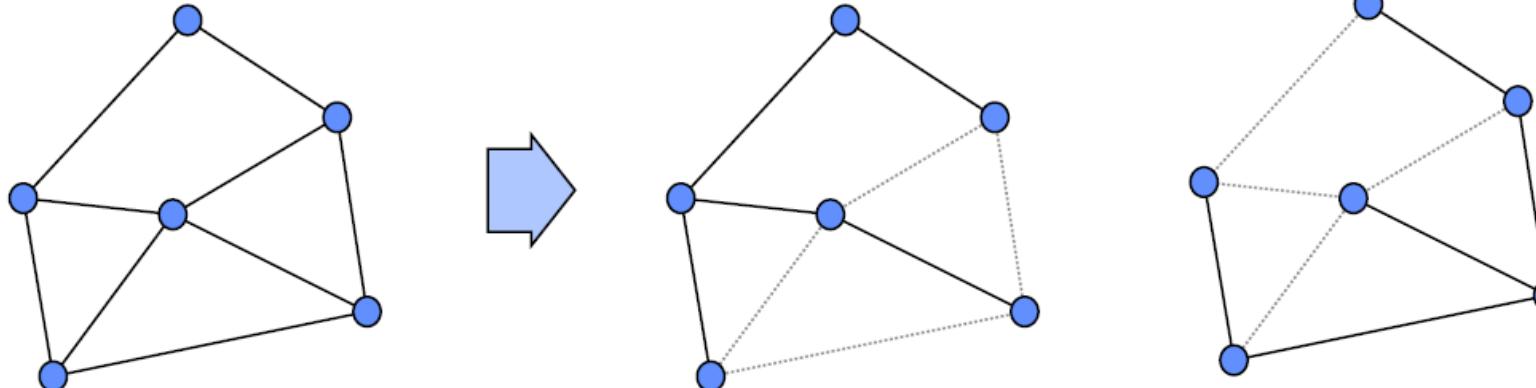
Problem: Layer 2 Switching Loop (4)

- ▶ Another problem is that the File Server can receive multiple copies of the same Ethernet frame arriving via different paths, which leads to additional overhead.
- ▶ Layer 2 Switching loops may cause serious problem to network performance.
- ▶ Layer 2 Switching loops are prevented in networks using **Spanning Tree Protocol**.

Spanning Tree Algorithm

- ▶ Allow a path between every LAN without causing loops (*loop-free environment*)
- ▶ Bridges communicate with special configuration messages (*BPDUs*)
- ▶ Standardized by IEEE 802.1D

Note: redundant paths are good, active redundant paths are bad
(they cause loops)



Libpcap - API

Presenter:

Assoc.Prof.Dr. Fatih ABUT

Agenda

- ▶ Installing libpcap
- ▶ Some C stuff
- ▶ Basic libpcap program
 - ▶ Grab a device to sniff
 - ▶ Filters/Event Loops
 - ▶ Packet structure

Install on Linux

- ▶ gunzip libpcap-0.7.1.tar.gz
- ▶ tar -xvf libpcap-0.7.1.tar
- ▶ cd libpcap-0.7.1
- ▶ ./configure
- ▶ make

- ▶ Or apt-get install libpcap-dev

Avoiding C Gotchas

- ▶ Always declare variables at the beginning of a block (no Java/C++ messiness!!)
- ▶ Nothing ‘new’: Always free what you malloc
`malloc(sizeof(thingYouWantToAllocate));`
- ▶ Always check the return value (no Exceptions!)

```
if (thing_didnt_work()) {  
    fprintf(stderr, "ERROR: thing didn't work\n");  
    exit(-1);  
} /* if (thing_didnt_work) */
```



C cont'd

- ▶ Output is formatted.

```
char person[ ] = "baby";
```

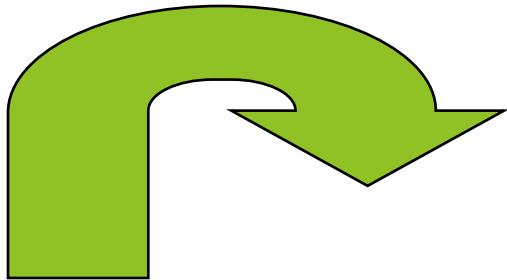
```
printf("give me %d, %s\n", 5, person);
```

%d: int

%x: hex

%s: string

%f: double



Get to the point!

- ▶ Pass by reference explicitly
 - Pass-by-reference prototype

```
int doSomething( Thing * );
```

Choice 1:

```
Thing *t;  
doSomething( t );
```

Choice 2:

```
Thing t;  
doSomething( &t );
```

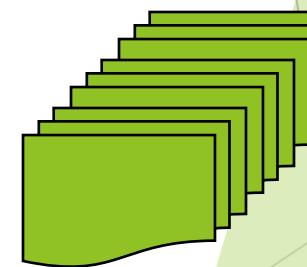
- Arrays are always in reference mode:
`char *` is like `char[0]`

Finally...

- ▶ C is NOT an object-oriented language

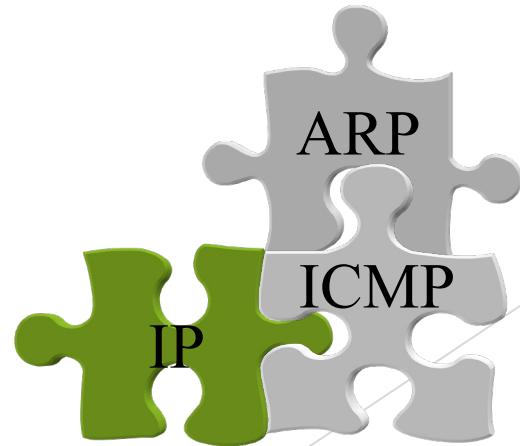
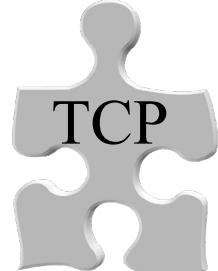
Most frequent data structure is a struct. Under the covers this is an array of contiguous bytes.

```
struct pcap_pkthdr {  
    struct timeval ts; //time stamp  
    bpf_u_int32 caplen; // length of portion present  
    bpf_u_int32 len; //packet length  
}
```



Overview of libpcap

- ▶ What to include and how to compile
- ▶ Going Live
- ▶ Main Event Loop
- ▶ Reading from a packet
- ▶ Filters



What to include and how to compile

- ▶ gcc sniff.c -lpcap -o sniff
- ▶ You must be root or admin
- ▶ Some headers I've used.

```
#include <pcap.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include<netinet/if_ether.h>
```

```
#include<netinet/in.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
#include <arpa/inet.h>
```

For Windows:

```
#include <winsock.h>
```

Getting onto the NIC

```
int main(int argc, char **argv) {  
  
    char *dev;          /* name of the device to use */  
    pcap_t* descr;      /* pointer to device descriptor */  
    struct pcap_pkthdr hdr; /* struct: packet header */  
    const u_char *packet; /* pointer to packet */  
    bpf_u_int32 maskp;   /* subnet mask */  
    bpf_u_int32 netp;    /* ip */  
    char errbuf[PCAP_ERRBUF_SIZE];  
  
    /* ask pcap to find a valid device to sniff */  
    dev = pcap_lookupdev(errbuf);  
    if(dev == NULL) {  
        printf("%s\n",errbuf); exit(1);  
    }  
    printf("DEV: %s\n",dev);
```

Going Live!



```
/* ask pcap for the network address and mask of the device */
pcap_lookupnet(dev,&netp,&maskp,errbuf);

descr = pcap_open_live(dev,BUFSIZ, 0, -1,errbuf);

/* BUFSIZ is max packet size to capture, 0 is promiscous, -1 means
don't wait for read to time out. */

if(descr == NULL)
{
    printf("pcap_open_live(): %s\n",errbuf);
    exit(1);
}
```

Once live, capture a packet.

```
packet = pcap_next(descr, &hdr);
if (packet == NULL)  {
    printf("It got away!\n");
    exit(1);
}
else printf("one lonely packet.\n");
return 0;
} //end main
```

Main Event Loop

```
void my_callback(u_char *useless,const struct
    pcap_pkthdr* pkthdr,const u_char* packet) {
    //do stuff here with packet
}

int main(int argc, char **argv) {
    //open and go live

    pcap_loop(descr,-1,my_callback,NULL);
    return 0;
}
```

What is an ethernet header?

From #include<netinet/if_ether.h>

```
struct ether_header {  
    u_int8_t ether_dhost[ETH_ALEN]; /* 6 bytes destination */  
    u_int8_t ether_shost[ETH_ALEN]; /* 6 bytes source addr */  
    u_int16_t ether_type;           /* 2 bytes ID type */  
} __attribute__((__packed__));
```

Some ID types:

```
#define ETHERTYPE_IP    0x0800 /* IP */  
#define ETHERTYPE_ARP 0x0806 /* Address resolution */
```

NO!

So we may need to swap bytes to read the data.

```
struct ether_header *eptr; /* where does this go? */  
eptr = (struct ether_header *) packet;
```

```
/* Do a couple of checks to see what packet type we have..*/  
if ( ntohs (eptr->ether_type) == ETHERTYPE_IP ) {  
    printf("Ethernet type hex:%x dec:%d is an IP packet\n",  
          ntohs(eptr->ether_type), ntohs(eptr->ether_type));  
  
} else if ( ntohs (eptr->ether_type) == ETHERTYPE_ARP ) {  
    printf("Ethernet type hex:%x dec:%d is an ARP packet\n",  
          ntohs(eptr->ether_type), ntohs(eptr->ether_type));  
}
```

Filter - we don't need to see every packet!

- ▶ Filters are strings. They get “compiled” into “programs”

```
struct bpf_program fp; //where does it go?
```

- ▶ Just before the event loop:

```
if (pcap_compile(descr,&fp,argv[1],0,netp) == -1) {  
    fprintf(stderr,"Error calling pcap_compile\n");  
    exit(1);  
}  
  
if (pcap_setfilter(descr,&fp) == -1) {  
    fprintf(stderr,"Error setting filter\n");  
    exit(1);  
}
```

Some typical filters

`./sniff "dst port 80"`

`./sniff "src host 128.226.121.120"`

`./sniff "less 50"`

(grab all packets less than 50 bytes)

`./sniff "ip proto \udp"`

(must use the escape character, \ , for protocol names)

```
34 #include <stdio.h>
35 #include <pcap.h>
36 #include <stddef.h>
37
38 void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)
39 {
40     return;
41 }
42
43 int main(int argc, char *argv[])
44 {
45     pcap_t *ph; //pcap_handler
46     char *dev;
47     char errbuf[PCAP_ERRBUF_SIZE];
48
49     if (argc != 2) {
50         printf ("Usage: %s DEVICE\n", argv[0]);
51         return 1;
52     }
53
54     dev = argv[1];
55
56     struct bpf_program fp;           /* The compiled filter expression */
57
58     /* The filter expression */ //only Segments with SYN, SYN+ACK or ACK Flags
59     char filter_exp[] = "(tcp[13] == 2 || tcp[13] == 18 ||
60                         tcp[13] == 16) && (port 8080 || port 80)";
61
62     bpf_u_int32 mask;             /* The netmask of our sniffing device */
63     bpf_u_int32 net;              /* The IP of our sniffing device */
64     int num_packets = -1;          /* Number of packets to capture;
65                                   set to -1 to capture as long as an error occur*/
66
67     printf("Device: %s\n", dev);
68     printf("Number of packets to capture: %d\n", num_packets);
69     printf("Filter expression: %s\n", filter_exp);
```

```
71     /* Find the properties for the device */
72     if (pcap_lookupnet(dev, &net, &mask, errbuf) == -1)
73     {
74         fprintf(stderr, "Can't get netmask for device %s\n", dev);
75         net = 0;
76         mask = 0;
77     }
78
79     /* Open the session in promiscuous mode */
80     ph=pcap_open_live(dev, BUFSIZ, 0,1000, errbuf);
81     if (ph == NULL) {
82         fprintf(stderr, "Couldn't open device %s: %s\n", dev, errbuf);
83         return EXIT_FAILURE;
84     }
85
86     /* Compile the filter */
87     if (pcap_compile(ph, &fp, filter_exp, 0, net) == -1)
88     {
89         fprintf(stderr, "Couldn't parse filter %s: %s\n",
90                 filter_exp, pcap_geterr(ph));
91         return EXIT_FAILURE;
92     }
93     /* Apply the filter */
94     if (pcap_setfilter(ph, &fp) == -1)
95     {
96         fprintf(stderr, "Couldn't install filter %s: %s\n",
97                 filter_exp, pcap_geterr(ph));
98         return EXIT_FAILURE;
99     }
100
101    /* Capture packets */
102    printf ("Starting capture...\n");
103    pcap_loop(ph, num_packets, got_packet, NULL);
104
105    /* Close the session */
106    pcap_freecode(&fp);
107    pcap_close(ph);
108
109    printf ("\nCapture complete.\n");
110
111    return(0);
112 }
```

Address Resolution Protocol (ARP) & Internet Control Message Protocol (ICMP)

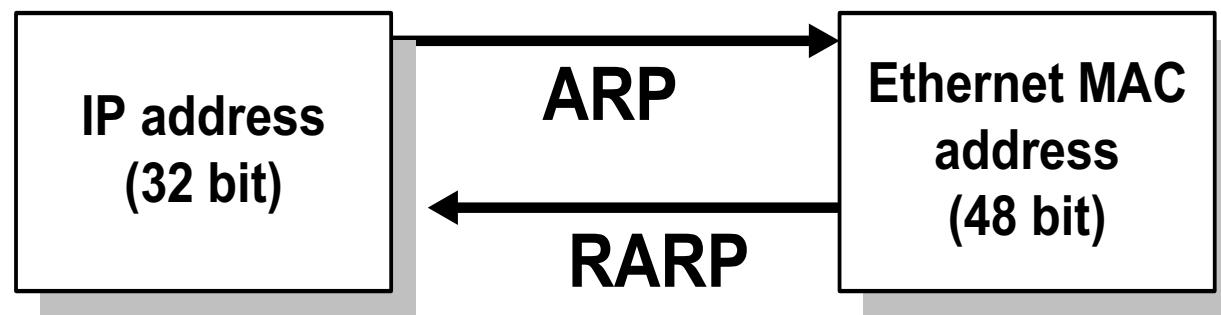
Assoc. Prof. Dr. Fatih ABUT
Department of Computer Engineering
Çukurova University

Overview

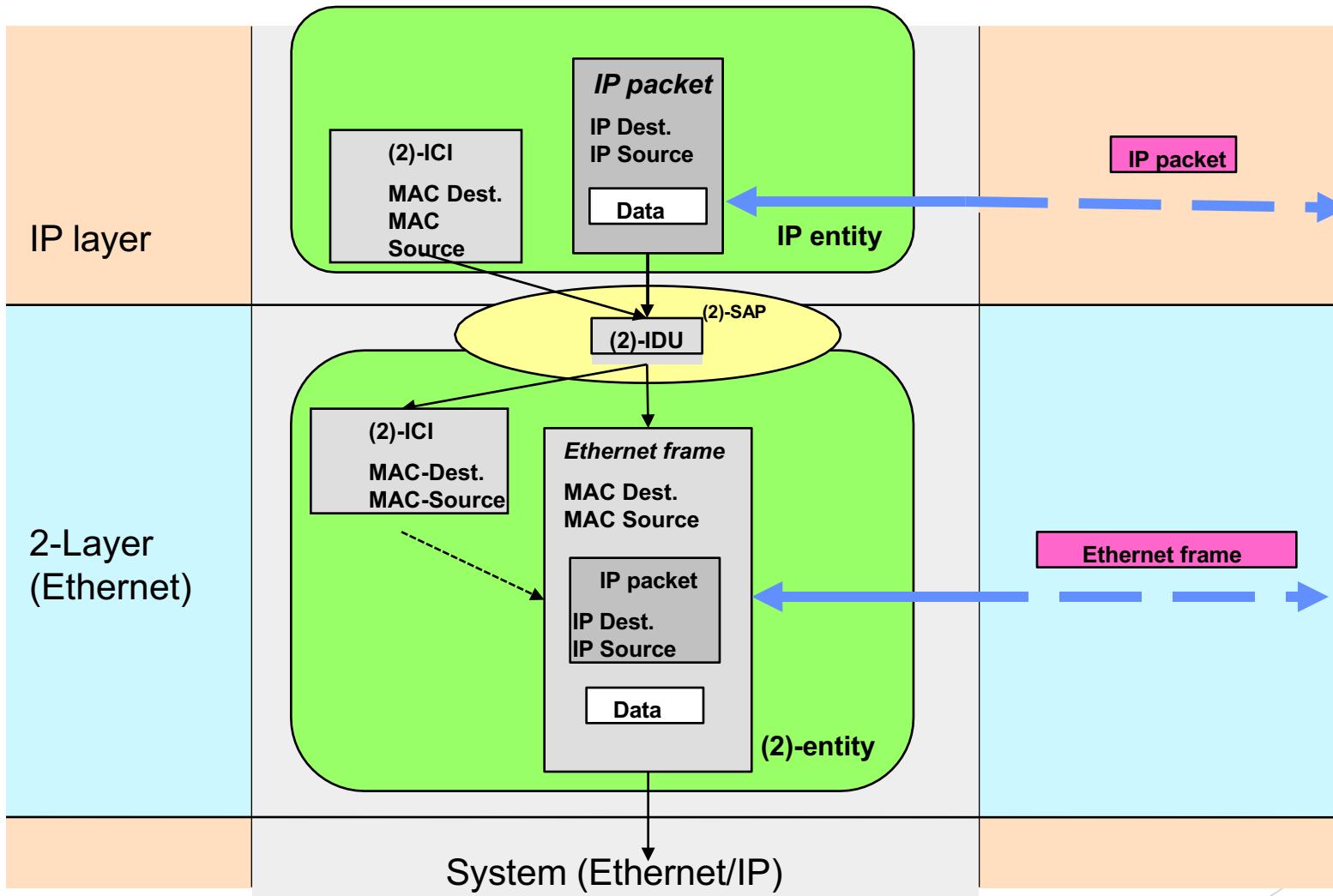
- ▶ The IP (Internet Protocol) relies on several other protocols to perform necessary control and routing functions:
 - ▶ Address resolutions (ARP)
 - ▶ Control functions (ICMP)
 - ▶ Multicast signaling (IGMP)
 - ▶ Setting up routing tables (RIP, OSPF, BGP, PIM, ...)

Address Resolution Protocol (ARP) and Reverse ARP (RARP)

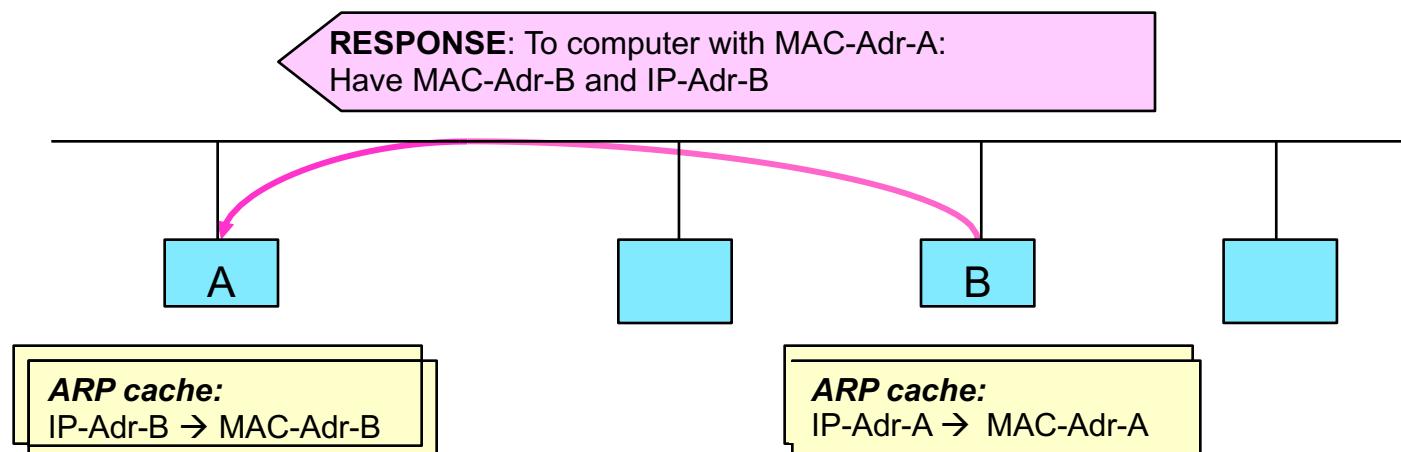
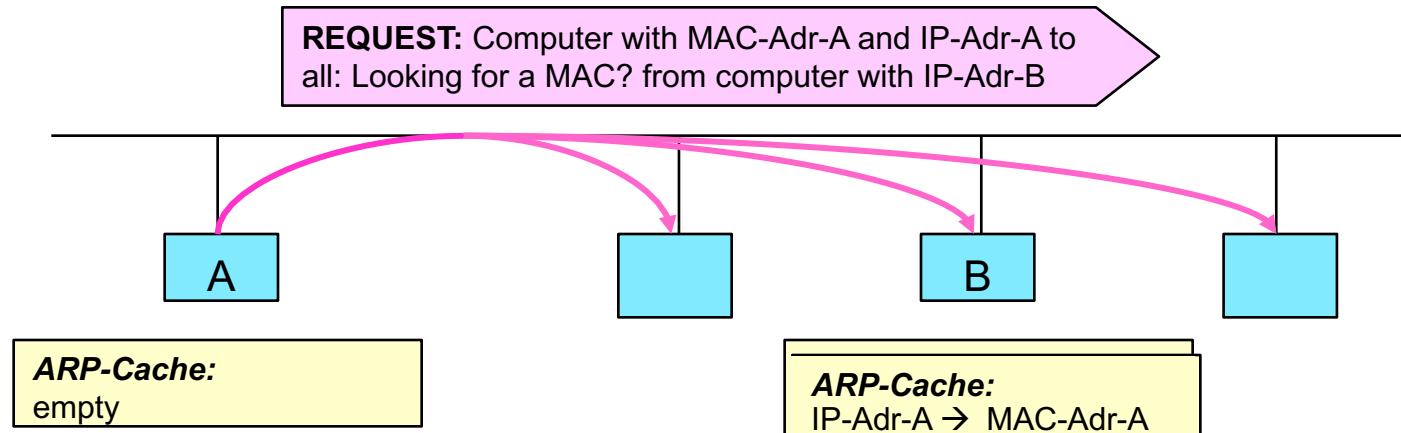
- ▶ Note:
 - ▶ The Internet is based on IP addresses
 - ▶ Data link protocols (Ethernet, FDDI, ATM) may have different (MAC) addresses
- ▶ The ARP and RARP protocols perform the **translation between IP addresses and MAC layer addresses**
- ▶ We will discuss ARP for broadcast LANs, particularly Ethernet LANs



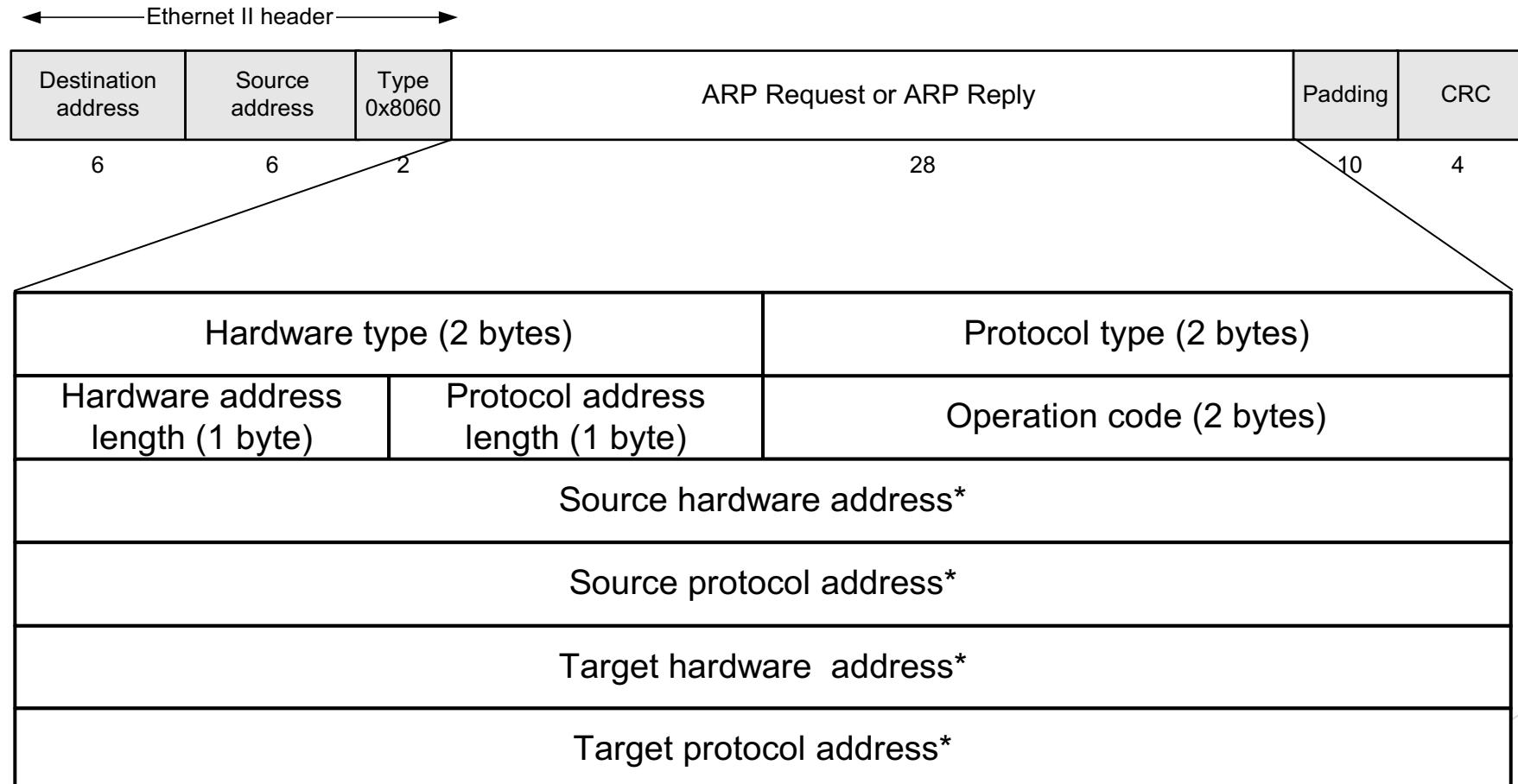
Coupling of the IP layer to the Ethernet layer according to OSI



ARP Protocol



ARP Packet Format (1)



* Note: The length of the address fields is determined by the corresponding address length fields

ARP Packet Format (2)

- ▶ **Hardware type (HTYPE)**
 - This field specifies the network link protocol type. Example: Ethernet is 1.
- ▶ **Protocol type (PTYPE)**
 - This field specifies the internetwork protocol for which the ARP request is intended. For IPv4, this has the value 0x0800. The permitted PTYPE values share a numbering space with those for [EtherType](#).
- ▶ **Hardware length (HLEN)**
 - Length (in [octets](#)) of a hardware address. Ethernet address length is 6.
- ▶ **Protocol length (PLEN)**
 - Length (in octets) of internetwork addresses. The internetwork protocol is specified in PTYPE. Example: IPv4 address length is 4.
- ▶ **Operation**
 - Specifies the operation that the sender is performing: 1 for request, 2 for reply.
- ▶ **Sender hardware address (SHA)**
 - Media address of the sender. In an ARP request this field is used to indicate the address of the host sending the request. In an ARP reply this field is used to indicate the address of the host that the request was looking for.
- ▶ **Sender protocol address (SPA)**
 - Internetwork address of the sender.
- ▶ **Target hardware address (THA)**
 - Media address of the intended receiver. In an ARP request this field is ignored. In an ARP reply this field is used to indicate the address of the host that originated the ARP request.
- ▶ **Target protocol address (TPA)**
 - Internetwork address of the intended receiver.

Example

- ▶ *ARP Request from Argon:*

Source hardware address: 00:a0:24:71:e4:44
Source protocol address: 128.143.137.144
Target hardware address: 00:00:00:00:00:00
Target protocol address: 128.143.137.1

- ▶ *ARP Reply from Router137:*

Source hardware address: 00:e0:f9:23:a8:20
Source protocol address: 128.143.137.1
Target hardware address: 00:a0:24:71:e4:44
Target protocol address: 128.143.137.144

ARP Cache

- ▶ Since sending an ARP request/reply for each IP datagram is inefficient, hosts maintain a cache (ARP Cache) of current entries. The entries expire after 20 minutes.
- ▶ Contents of the ARP Cache:
 - (128.143.71.37) at 00:10:4B:C5:D1:15 [ether] on eth0
 - (128.143.71.36) at 00:B0:D0:E1:17:D5 [ether] on eth0
 - (128.143.71.35) at 00:B0:D0:DE:70:E6 [ether] on eth0
 - (128.143.136.90) at 00:05:3C:06:27:35 [ether] on eth1
 - (128.143.71.34) at 00:B0:D0:E1:17:DB [ether] on eth0
 - (128.143.71.33) at 00:B0:D0:E1:17:DF [ether] on eth0

Things to know about ARP

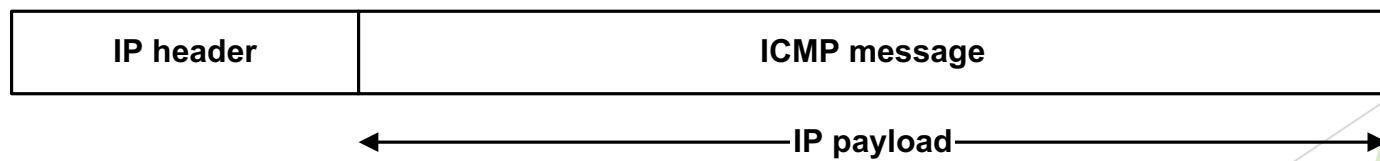
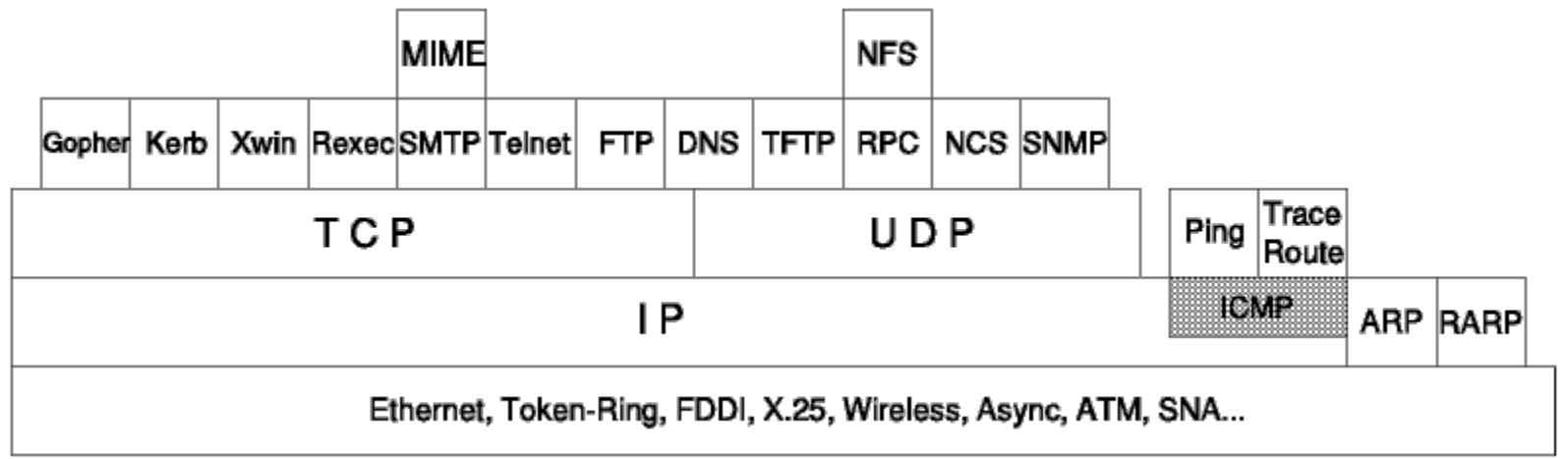
- ▶ What happens if an ARP Request is made for a non-existing host?
Several ARP requests are made with increasing time intervals between requests. Eventually, ARP gives up.

- ▶ What if a host sends an ARP request for its own IP address?
The other machines respond (**gratuitous ARP**) as if it was a normal ARP request.

This is useful for detecting if an IP address has already been assigned.

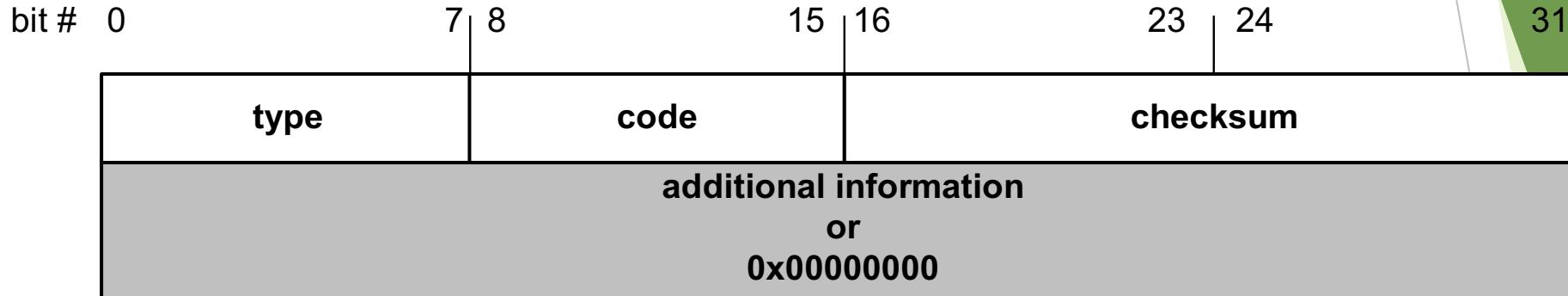
ICMP Overview

- ▶ The **Internet Control Message Protocol (ICMP)** is a helper protocol that supports IP with facility for
 - ▶ Error reporting
 - ▶ Simple queries



- ▶ ICMP messages are encapsulated as IP datagrams

ICMP message format

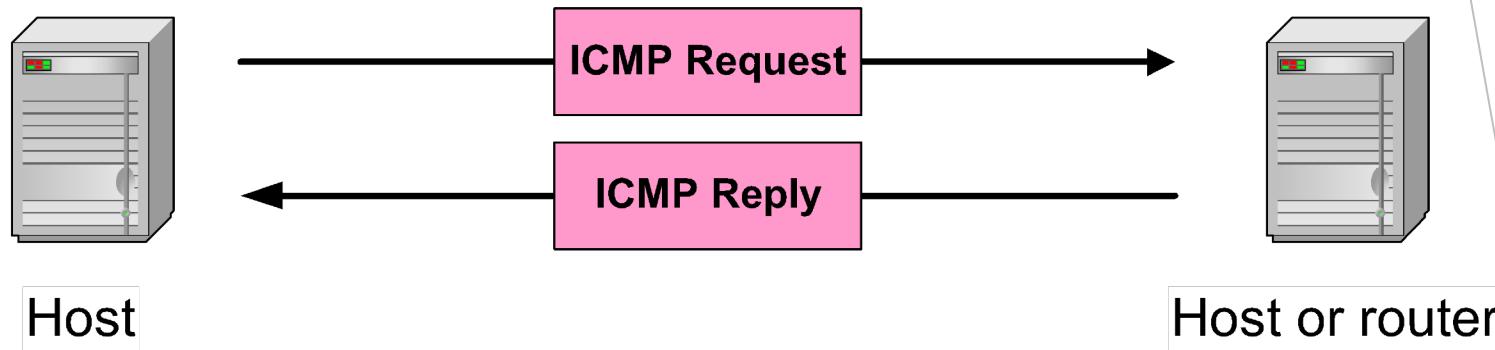


4 byte header:

- **Type (1 byte)**: type of ICMP message
- **Code (1 byte)**: subtype of ICMP message
- **Checksum (2 bytes)**: similar to IP header checksum.
Checksum is calculated over entire ICMP message

If there is no additional data, there are 4 bytes set to zero.
→ each ICMP messages is at least 8 bytes long

ICMP Query message

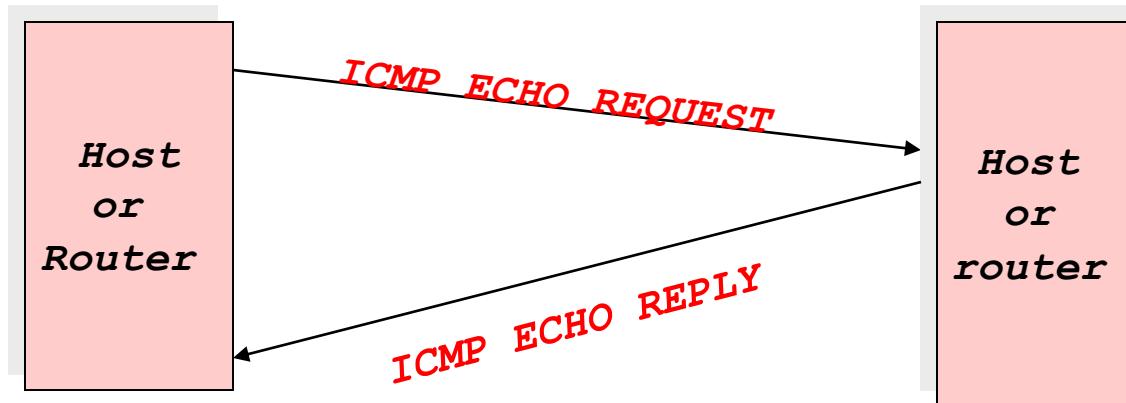


ICMP query:

- **Request** sent by host to a router or host
- **Reply** sent back to querying host

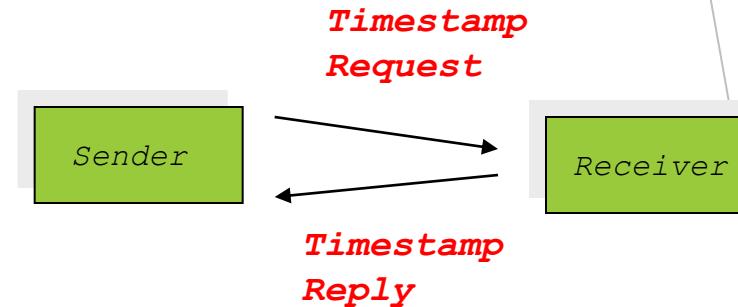
Example of a Query: Echo Request and Reply

- ▶ Ping's are handled directly by the kernel
- ▶ Each Ping is translated into an **ICMP Echo Request**
- ▶ The Ping'ed host responds with an **ICMP Echo Reply**



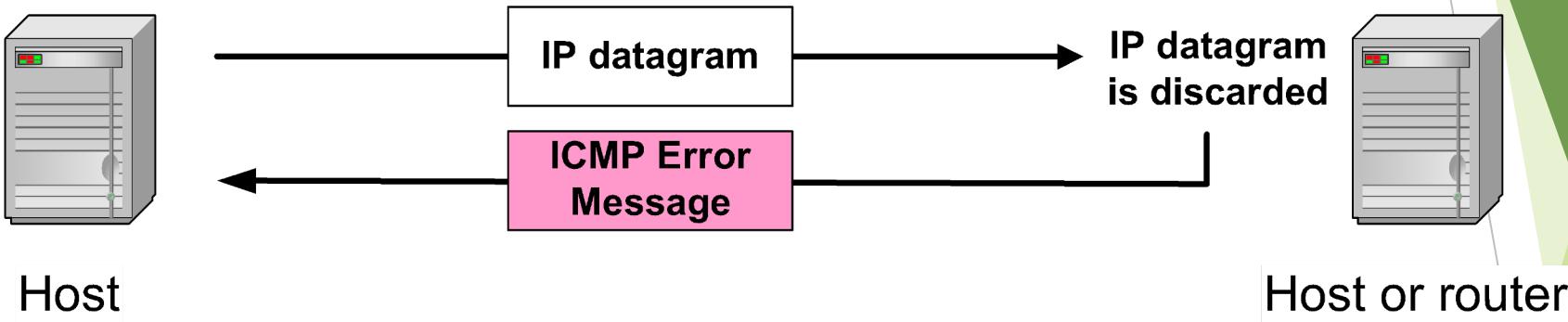
Example of a Query: ICMP Timestamp

- ▶ A system (host or router) asks another system for the current time.
- ▶ Time is measured in milliseconds after midnight UTC (Universal Coordinated Time) of the current day
- ▶ Sender sends a **request**, receiver responds with **reply**



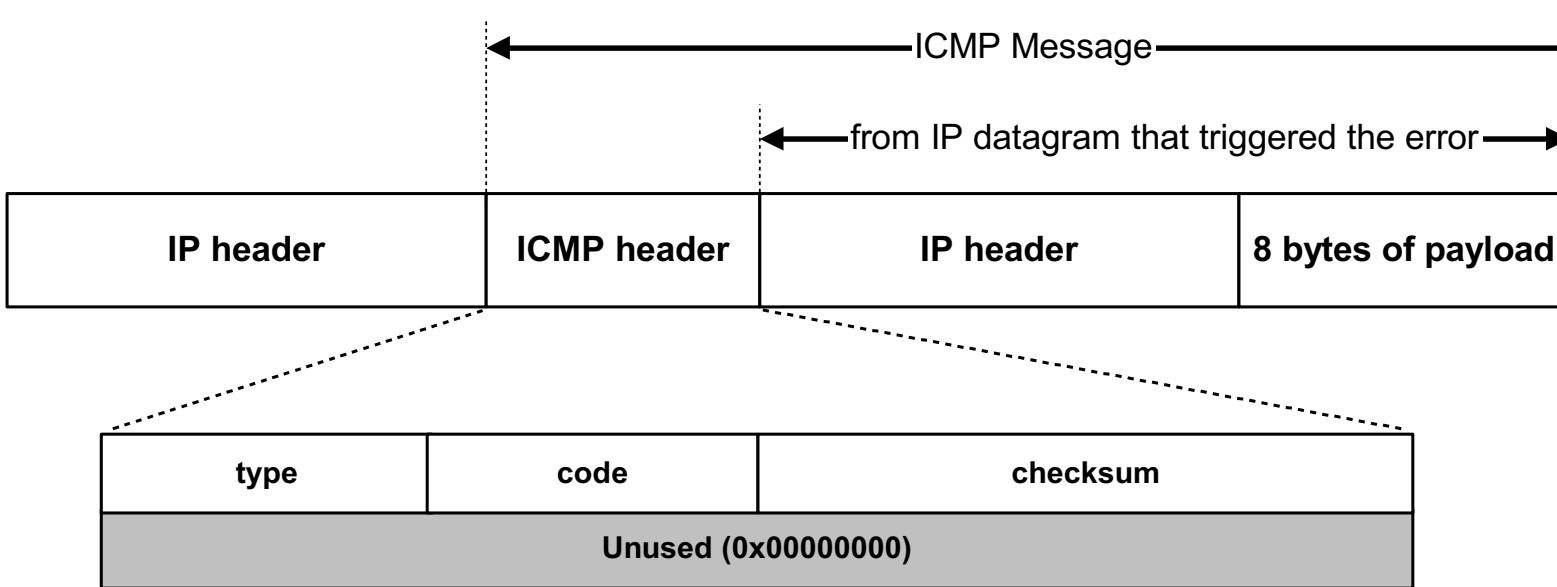
Type (= 17 or 18)	Code (=0)	Checksum
identifier		sequence number
32-bit sender timestamp		
32-bit receive timestamp		
32-bit transmit timestamp		

ICMP Error message



- ICMP error messages report error conditions
- Typically sent when a datagram is discarded
- Error message is often passed from ICMP to the application program

ICMP Error message



ICMP error messages include the complete IP header and the first 8 bytes of the payload (typically: UDP, TCP)

Frequent ICMP Error message

Type	Code	Description	
3	0–15	Destination unreachable	Notification that an IP datagram could not be forwarded and was dropped. The code field contains an explanation.
5	0–3	Redirect	Informs about an alternative route for the datagram and should result in a routing table update. The code field explains the reason for the route change.
11	0, 1	Time exceeded	Sent when the TTL field has reached zero (Code 0) or when there is a timeout for the reassembly of segments (Code 1)
12	0, 1	Parameter problem	Sent when the IP header is invalid (Code 0) or when an IP header option is missing (Code 1)

Some subtypes of the “Destination Unreachable”

Code	Description	Reason for Sending
0	Network Unreachable	No routing table entry is available for the destination network.
1	Host Unreachable	Destination host should be directly reachable, but does not respond to ARP Requests.
2	Protocol Unreachable	The protocol in the protocol field of the IP header is not supported at the destination.
3	Port Unreachable	The transport protocol at the destination host cannot pass the datagram to an application.
4	Fragmentation Needed and DF Bit Set	IP datagram must be fragmented, but the DF bit in the IP header is set.

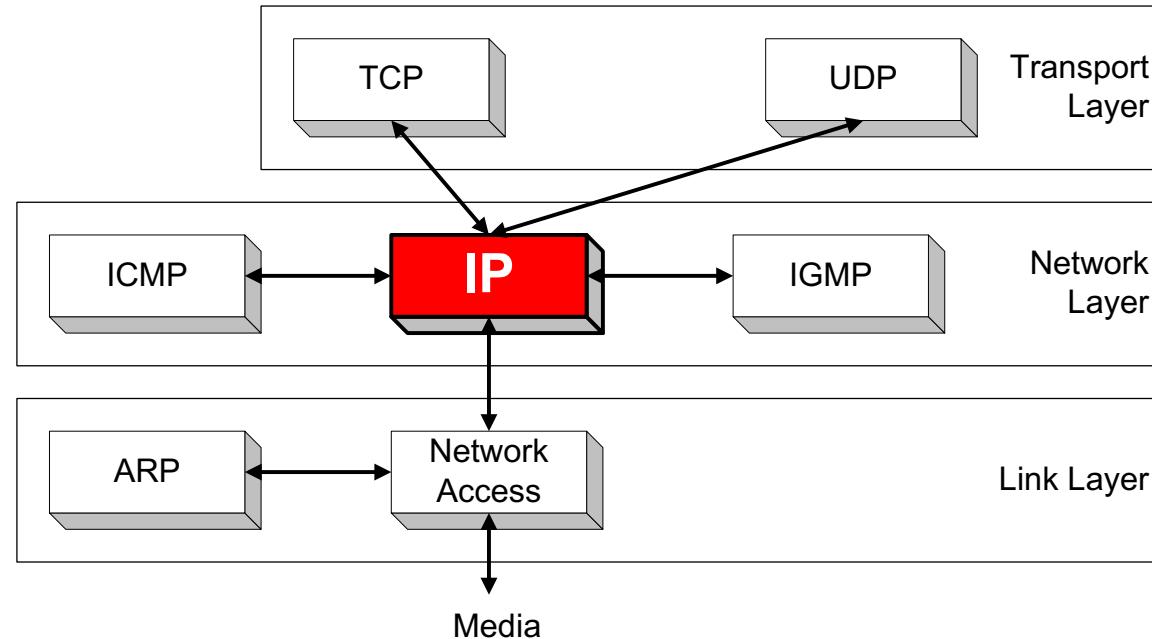
IP - The Internet Protocol

Presenter: Assoc.Prof.Dr. Fatih ABUT

- ▶ **Introduction**
- ▶ **IP Addressing**
- ▶ **Subnetting**
- ▶ **Variable Length Subnet Mask (VLSM)**

Orientation

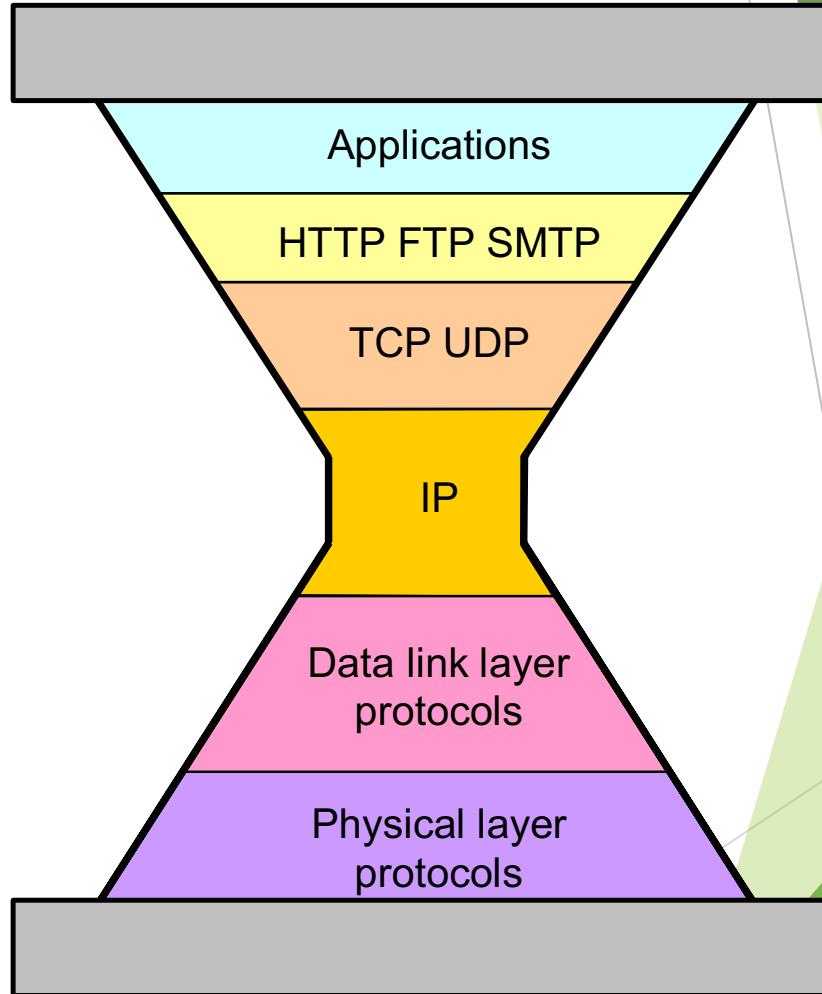
- ▶ IP (Internet Protocol) is a Network Layer Protocol.



- ▶ IP's current version is Version 6 (IPv6).

IP: The waist of the hourglass

- ▶ IP is the **waist of the hourglass** of the Internet protocol architecture
- ▶ Multiple higher-layer protocols
- ▶ Multiple lower-layer protocols
- ▶ Only one protocol at the network layer.



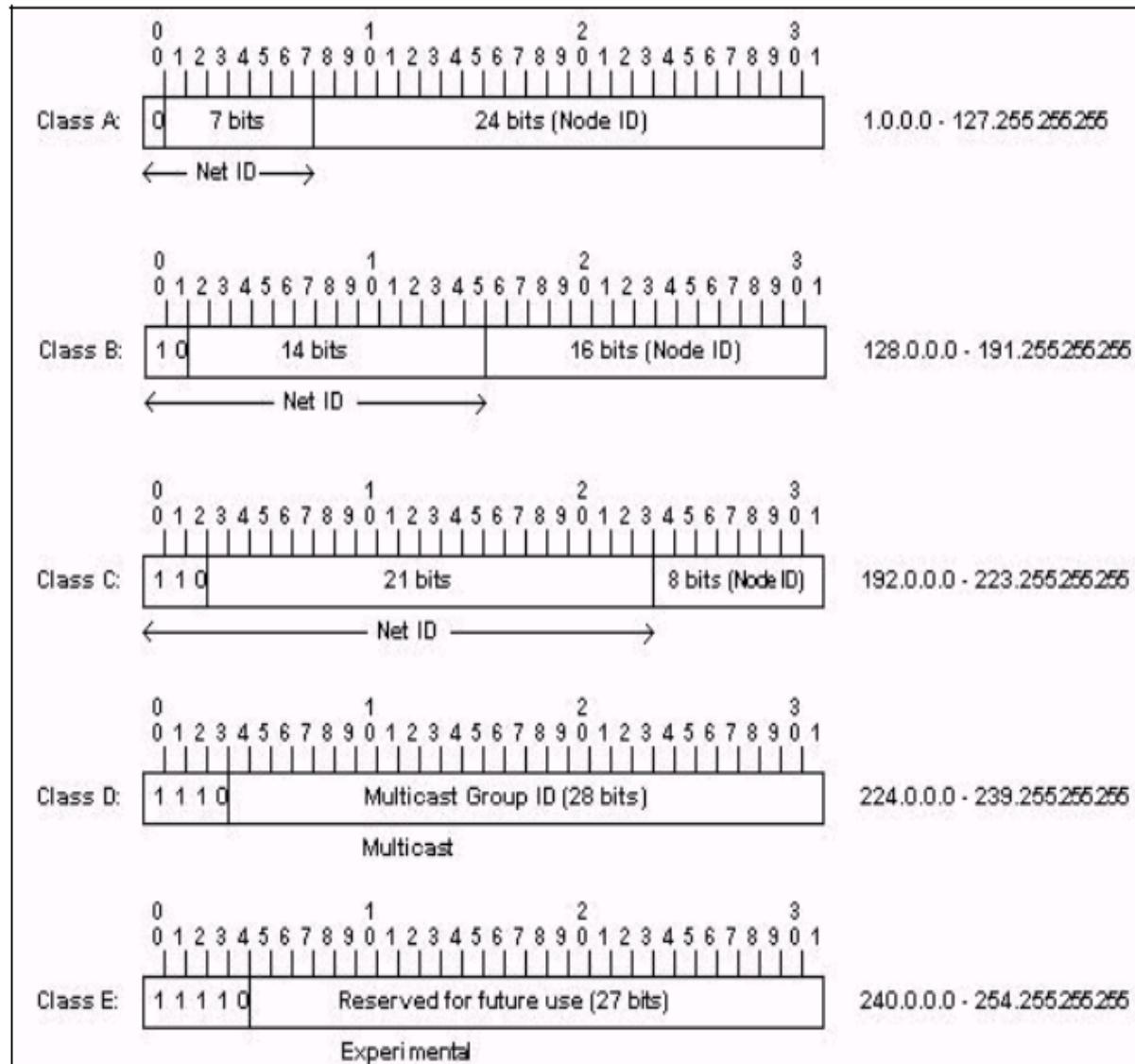
IP Service

- ▶ Delivery service of IP is minimal
- ▶ IP provides an **unreliable connectionless** best effort service (also called: “datagram service”).
 - ▶ **Unreliable:** IP does not make an attempt to recover lost packets
 - ▶ **Connectionless:** Each packet (“datagram”) is handled independently. IP is not aware that packets between hosts may be sent in a logical sequence
 - ▶ **Best effort:** IP does not make guarantees on the service (no throughput guarantee, no delay guarantee,...)
- ▶ Consequences:
 - ▶ Higher layer protocols have to deal with losses or with duplicate packets
 - ▶ Packets may be delivered out-of-sequence

IP Addresses

- An IP address is an address used to uniquely identify a device on an IP network.
- The address is made up of 32 binary bits which can be divisible into a network portion and host portion with the help of a subnet mask.
- 32 binary bits are broken into four octets (1 octet = 8 bits)
- Dotted decimal format (for example, 172.16.81.100)

IP Address Classes (1)



IP Address Classes (2)

- ▶ Class A: The first octet is the network portion. Octets 2, 3, and 4 are for subnets/hosts
- ▶ Class B: The first two octets are the network portion. Octets 3 and 4 are for subnets/hosts
- ▶ Class C: The first three octets are the network portion. Octet 4 is for subnets/hosts

Private Address Range

Address Class	Reserved Address Space
Class A	10.0.0.0 - 10.255.255.255
Class B	172.16.0.0 - 172.31.255.255
Class C	192.168.0.0 - 192.168.255.255

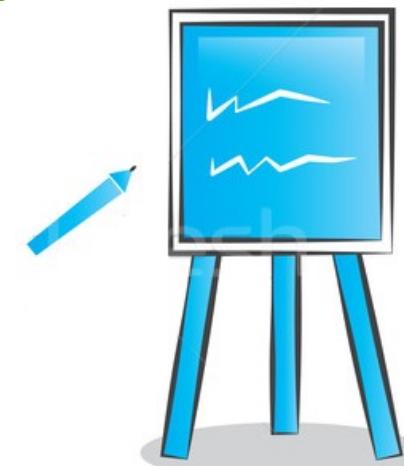
Network/Subnet Masks

- ▶ Distinguishes which portion of the address identifies the network and which portion of the address identifies the node.
- ▶ Default masks:
 - Class A: 255.0.0.0/8
 - Class B: 255.255.0.0/16
 - Class C: 255.255.255.0/24

Subnetting

- ▶ Creates multiple logical networks that exist within a single Class A, B, or C network.
- ▶ If you do not subnet, you will only be able to use one network from your Class A, B, or C network, which is unrealistic
- ▶ Each data link on a network must have a unique network ID, with every node on that link being a member of the same network

Example

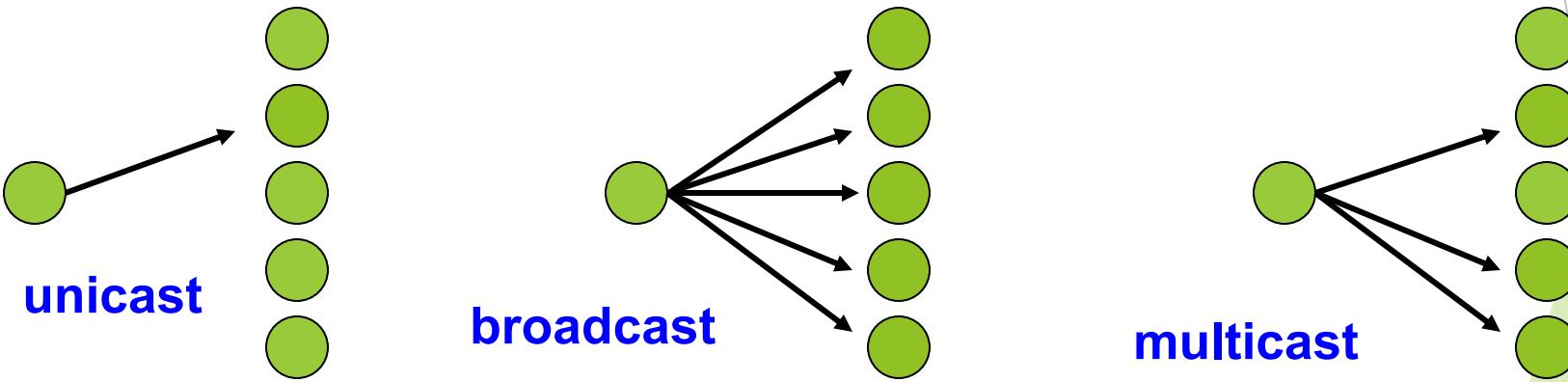


Benefits of Subnetting

- Reduced network traffic
- Optimized network performance
- Simplified management
- Facilitated spanning of large geographical distances

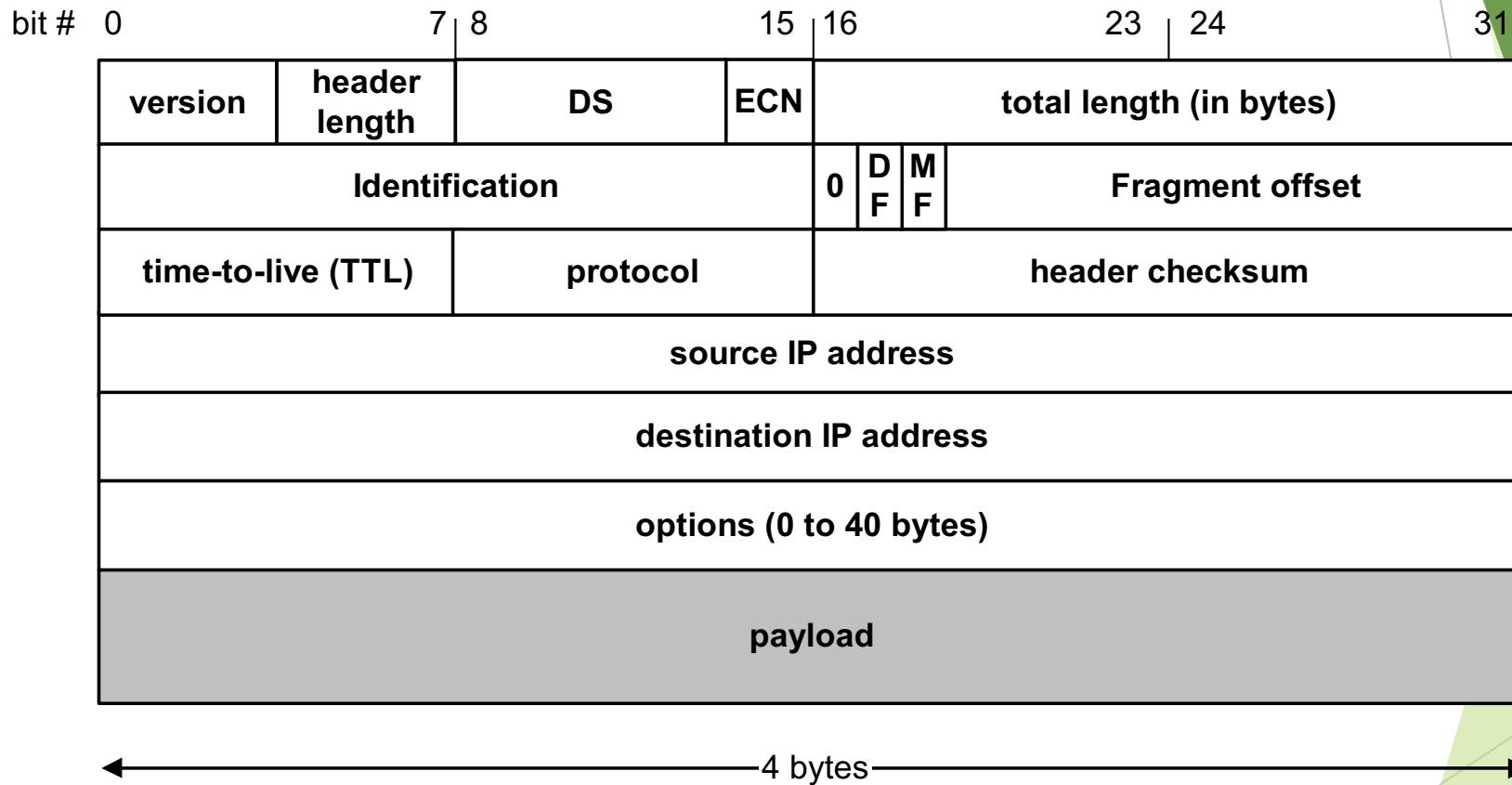
IP Service

- ▶ IP supports the following services:
 - ▶ one-to-one (**unicast**)
 - ▶ one-to-all (**broadcast**)
 - ▶ one-to-several. (**multicast**)



- ▶ IP multicast requires support of other protocols (IGMP, multicast routing)

IP Datagram Format



- ▶ $20 \text{ bytes} \leq \text{Header Size} < 2^4 \times 4 \text{ bytes} = 60 \text{ bytes}$
- ▶ $20 \text{ bytes} \leq \text{Total Length} < 2^{16} \text{ bytes} = 65536 \text{ bytes}$

Fields of the IP Header (1)

- ▶ **Version (4 bits):** current version is 4, next version will be 6.
- ▶ **Header length (4 bits):** length of IP header, in multiples of 4 bytes
- ▶ **DS/ECN field (1 byte)**
 - ▶ This field was previously called as Type-of-Service (TOS) field. The role of this field has been re-defined, but is “backwards compatible” to TOS interpretation
 - ▶ **Differentiated Service (DS) (6 bits):**
 - ▶ Used to specify service level (currently not supported in the Internet)
 - ▶ **Explicit Congestion Notification (ECN) (2 bits):**
 - ▶ New feedback mechanism used by TCP

Fields of the IP Header (2)

- ▶ **Identification (16 bits):** Unique identification of a datagram from a host. Incremented whenever a datagram is transmitted
- ▶ **Flags (3 bits):**
 - ▶ First bit always set to 0
 - ▶ DF bit (Do not fragment)
 - ▶ MF bit (More fragments)

Will be explained later → Fragmentation

Fields of the IP Header (3)

- ▶ **Time To Live (TTL) (1 byte):**

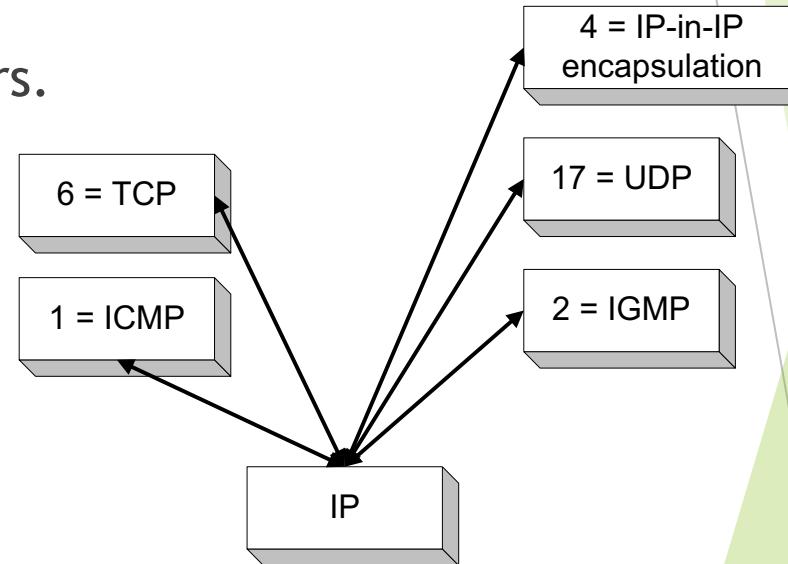
- ▶ Specifies longest paths before datagram is dropped
- ▶ Role of TTL field: Ensure that packet is eventually dropped when a routing loop occurs

Used as follows:

- ▶ Sender sets the value (e.g., 64)
- ▶ Each router decrements the value by 1
- ▶ When the value reaches 0, the datagram is dropped

Fields of the IP Header (4)

- ▶ **Protocol (1 byte):**
 - ▶ Specifies the higher-layer protocol.
 - ▶ Used for demultiplexing to higher layers.



- ▶ **Header checksum (2 bytes):** A simple 16-bit long checksum which is computed for the header of the datagram.

Fields of the IP Header (5)

- ▶ **Options:**
 - ▶ Security restrictions
 - ▶ Record Route: each router that processes the packet adds its IP address to the header.
 - ▶ Timestamp: each router that processes the packet adds its IP address and time to the header.
 - ▶ (loose) Source Routing: specifies a list of routers that must be traversed.
 - ▶ (strict) Source Routing: specifies a list of the only routers that can be traversed.
- ▶ **Padding:** Padding bytes are added to ensure that header ends on a 4-byte boundary

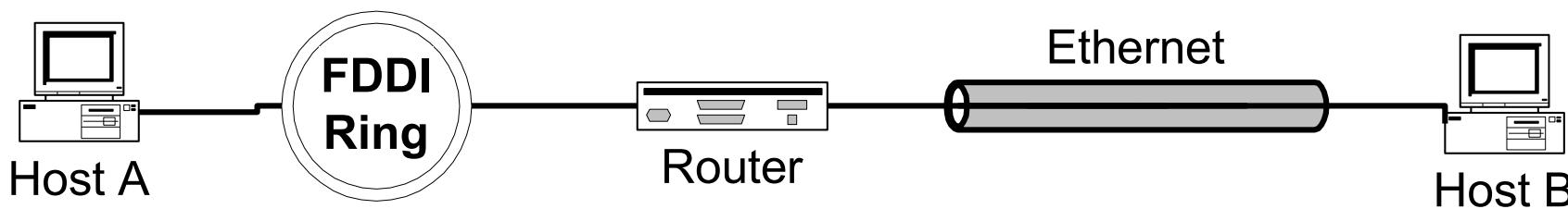
Maximum Transmission Unit (MTU)

- ▶ Maximum size of IP datagram is 65535, but the data link layer protocol generally imposes a limit that is much smaller
- ▶ Example:
 - ▶ Ethernet frames have a maximum payload of 1500 bytes
 - IP datagrams encapsulated in Ethernet frame cannot be longer than 1500 bytes
- ▶ The limit on the maximum IP datagram size, imposed by the data link protocol is called **maximum transmission unit (MTU)**
- MTUs for various data link protocols:

Ethernet:	1500	FDDI:	4352
802.3:	1492	ATM AAL5:	9180
802.5:	4464	PPP:	negotiated

IP Fragmentation

- What if the size of an IP datagram exceeds the MTU?
IP datagram is fragmented into smaller units.
- What if the route contains networks with different MTUs?



MTUs:

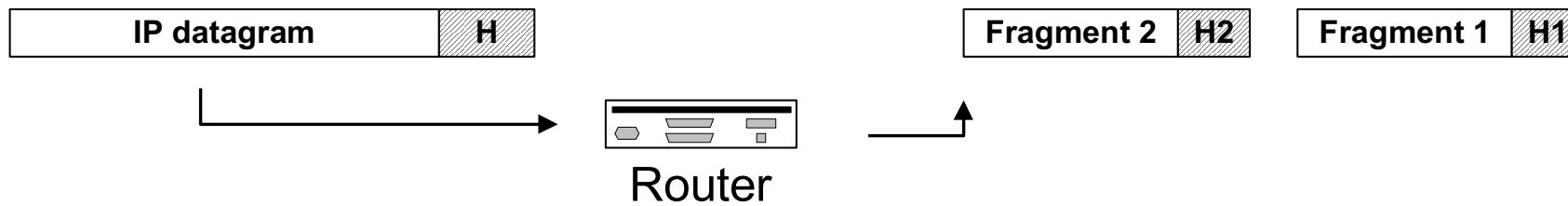
FDDI: 4352

Ethernet: 1500

- **Fragmentation:**
 - IP router splits the datagram into several datagram
 - Fragments are reassembled at receiver

Where is Fragmentation done?

- ▶ Fragmentation can be done at the sender or at intermediate routers
- ▶ The same datagram can be fragmented several times.
- ▶ Reassembly of original datagram is only done at destination hosts !!



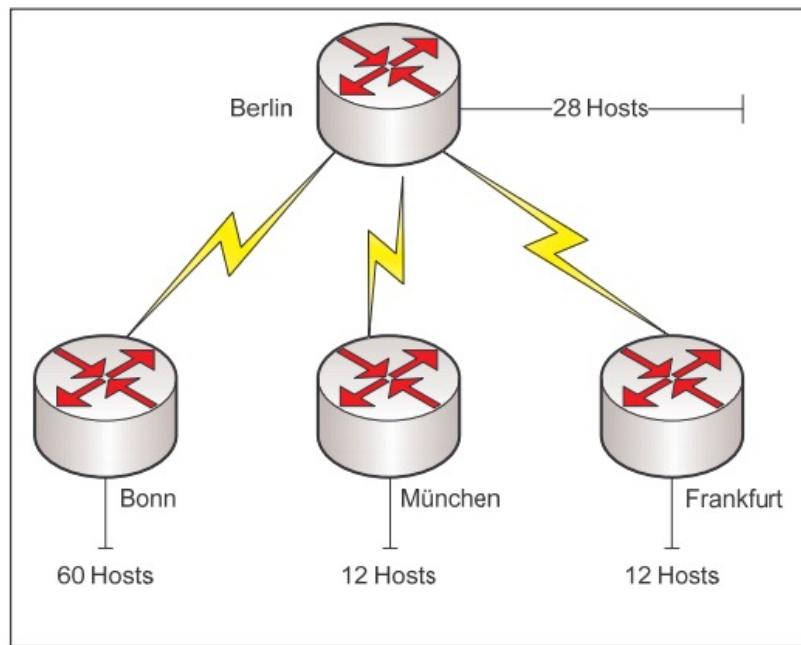
What's involved in Fragmentation?

version	header length	DS	ECN	total length (in bytes)			
Identification				0	D	M	F
time-to-live (TTL)		protocol		Fragment offset			
				header checksum			

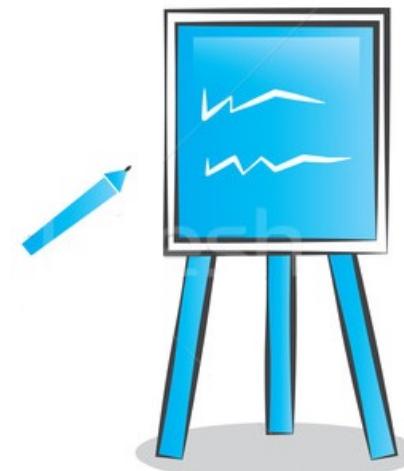
- ▶ The following fields in the IP header are involved:
 - ▶ **Identification** : When a datagram is fragmented, the identification is the same in all fragments
 - ▶ **Flags**
 - ▶ DF bit is set: Datagram cannot be fragmented and must be discarded if MTU is too small
 - ▶ MF bit set: This datagram is part of a fragment and an additional fragment follows this one
 - ▶ **Fragment offset:** Offset of the payload of the current fragment in the original datagram
 - ▶ **Total length:** Total length of the current fragment

VLSM Subnetting

- ▶ Using classful subnetting wastes IP addresses
- ▶ The number of hosts involved in each subnet must of same size
- ▶ Create a number of subnet masks that suit our needs more efficiently than a classful subnetting scheme could



Example

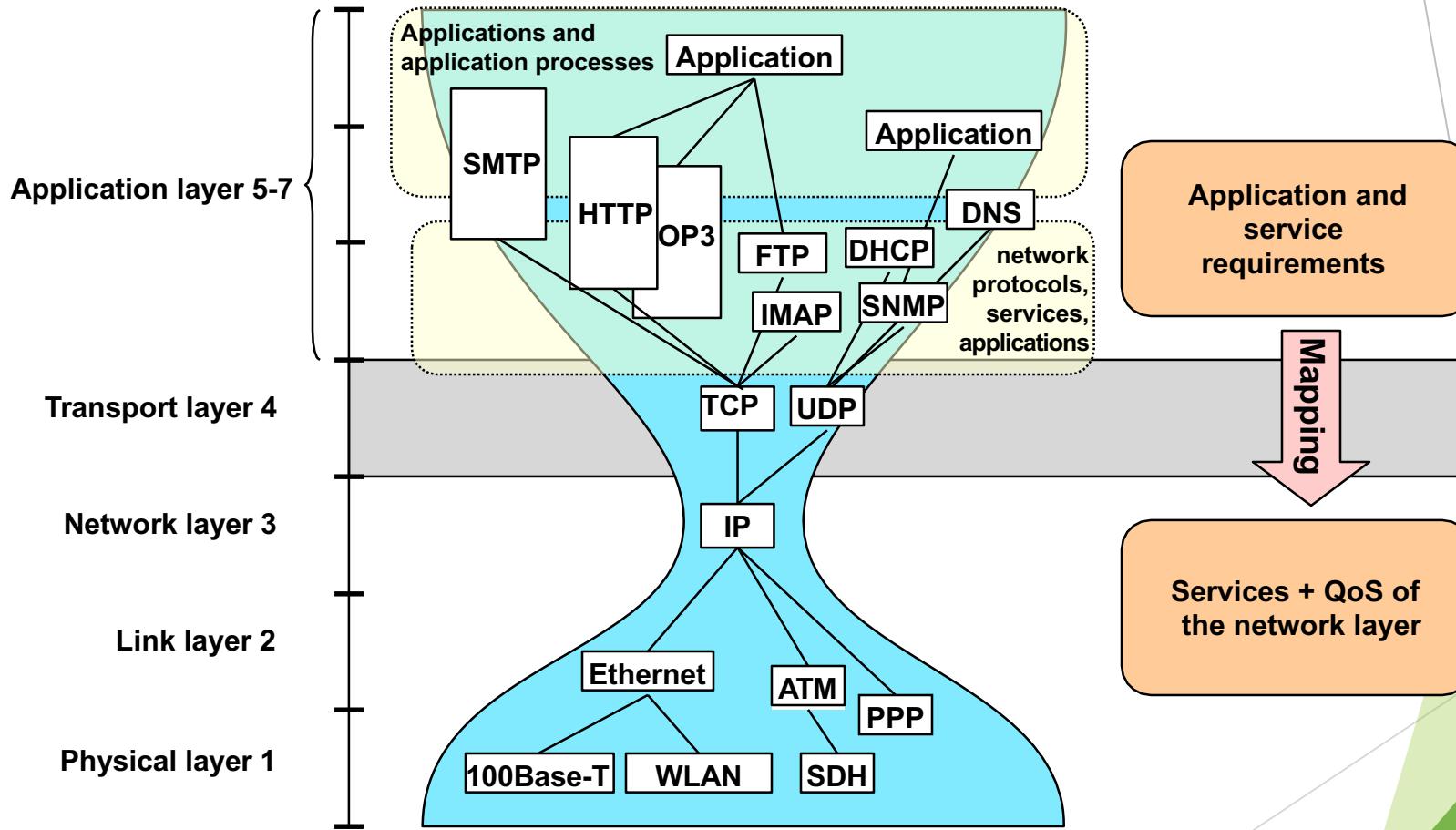


UDP & TCP Basics (Part I)

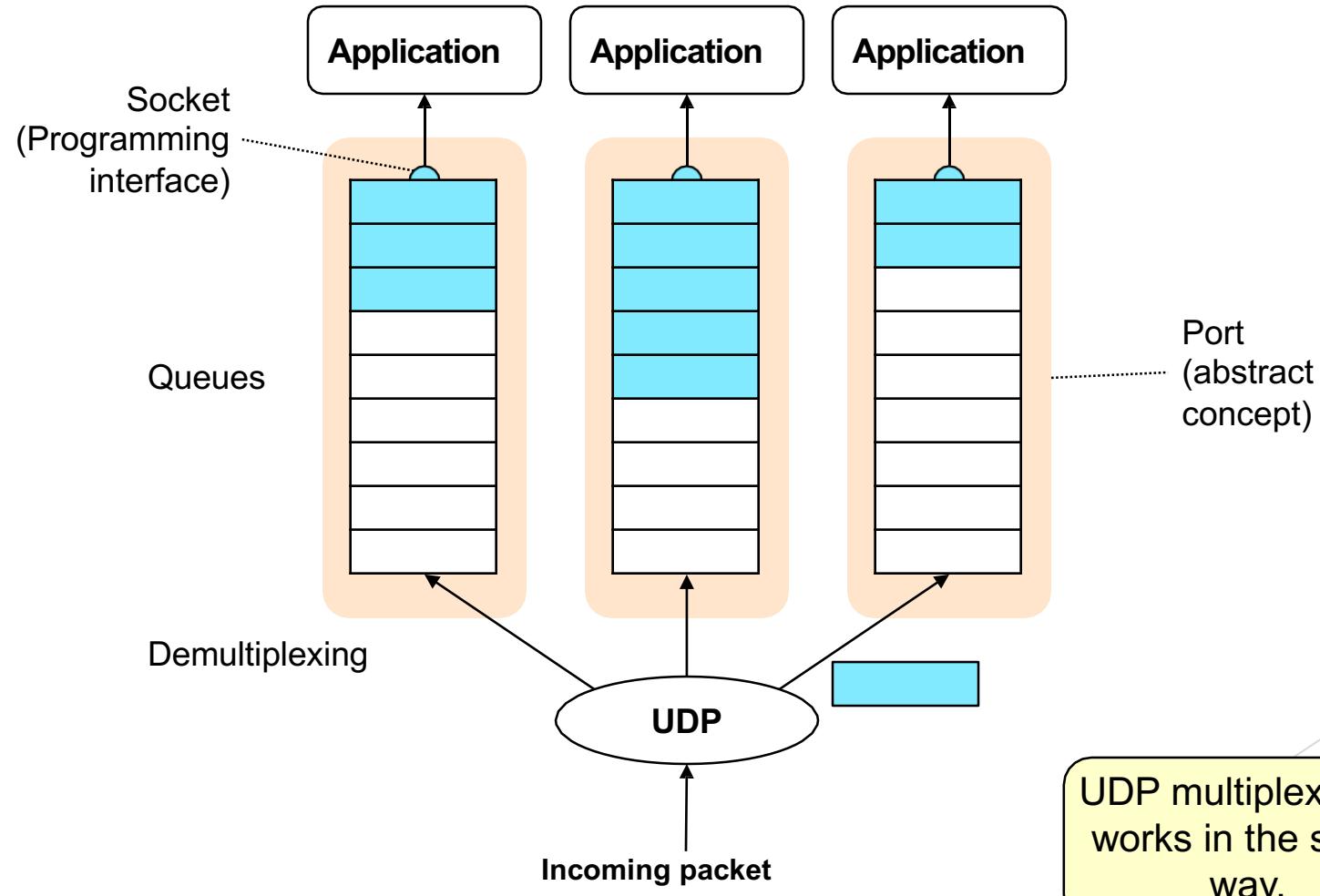
CEN 223 - Internet Communication

Assoc. Prof. Dr. Fatih ABUT
Department of Computer Engineering
Çukurova University

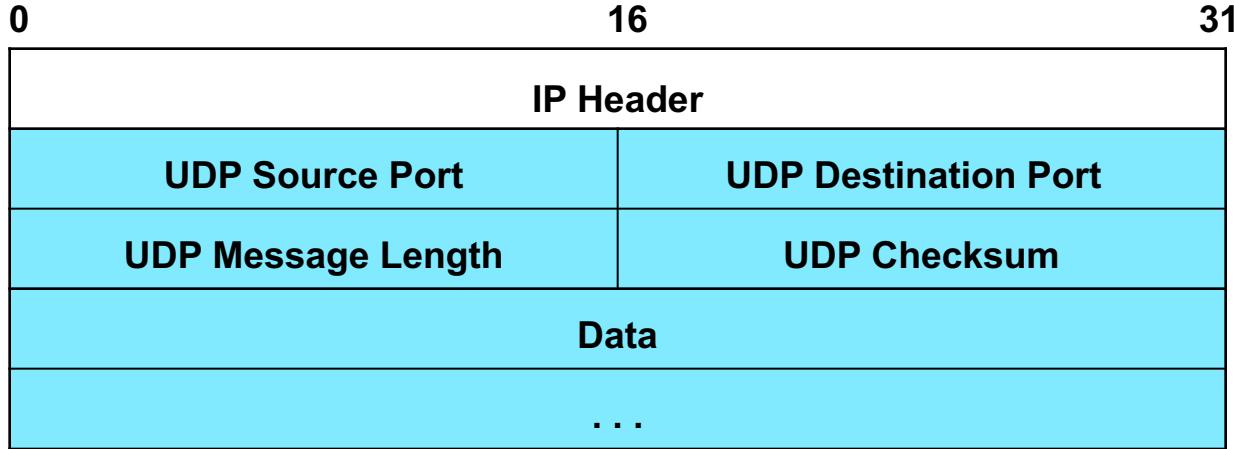
End-to-end protocols: classification + task definition



UDP Demultiplexing (Ports und Sockets)



UDP Header



- Source Port:** Contains the port of the sending host;
Optional specification, i.e., if specified, this is the port to which a possible response should go
- Destination Port:** Destination port where the datagram is sent
- Message Length:** Indicates the total length of the datagram
- Checksum:** Check sum via UDP header and IP pseudo header (protocol number + IP addresses)
Ensures that the correct port and host have been reached.

Problem: Reliable transport performance over an unreliable network layer

Properties of the IP network layer::

- Dividing the information into packages
- Maximum packet sizes
- Loss of packets in transit
- Receipt of corrupt packets
- Multiple transmission of packets
- Rearranging the order of the packets
- Changing packet transmission times
- Storage capacity of the network

IP is (only) a
packet-oriented
best-effort network

What should TCP do?

TCP provides a reliable
bidirectional byte stream

There is a lot to
be done for TCP

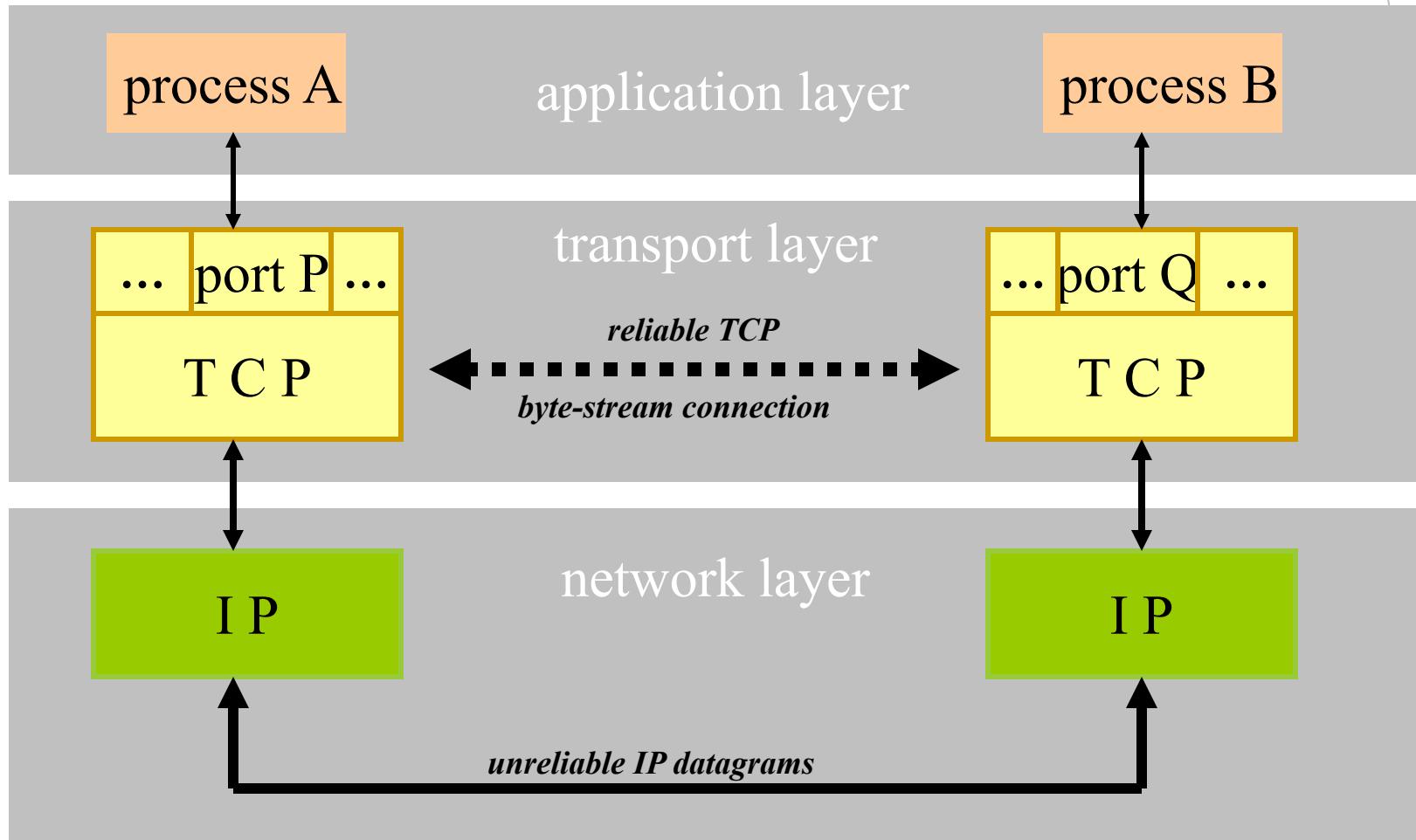
Introduction to TCP

- ▶ Point-to-point connection
- ▶ Streaming interface
 - ▶ Byte-oriented
 - ▶ No boundaries for fragments/segments
- ▶ Reliable, connection-oriented data transfer
 - **Three phases:** connection establishment, user data transmission, connection disconnection
 - Reliable connection establishment (\rightarrow **three-way handshake**)
 - Reliable user data transmission (\rightarrow **retransmission**)
 - Reliable disconnection
- ▶ Full duplex
- ▶ Flow and congestion controlled
- ▶ Protocol implemented entirely at the ends

Benefits of TCP

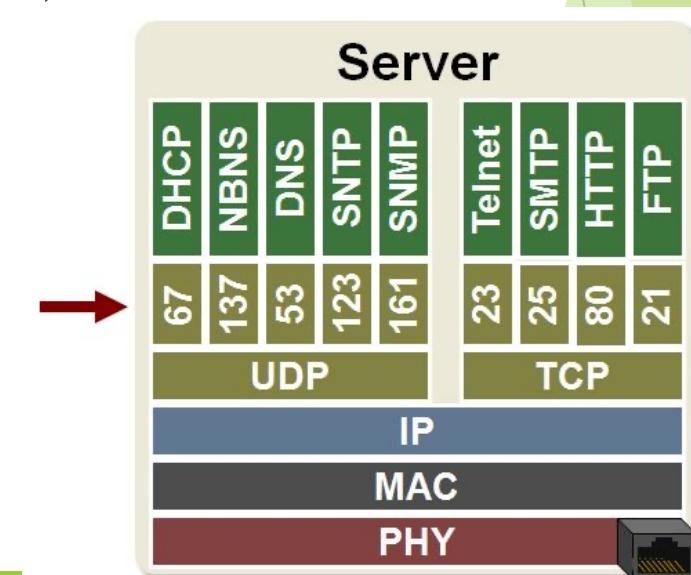
- ▶ Communication is ‘transparently reliable’
- ▶ Data is delivered in the proper sequence
- ▶ An application programmer does not need to worry about issues such as:
 - Lost or delayed packets
 - Timeouts and retransmissions
 - Duplicated packets
 - Packets arriving out-of-sequence
 - Flow and congestion control

Overview of port concept



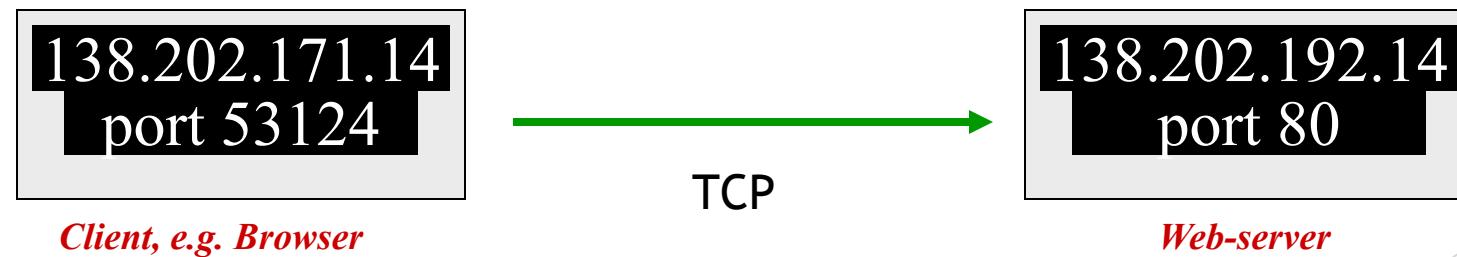
Knowing what port number to use

- ▶ Popular applications have well-known ports
 - ▶ E.g., port 80 for Web and port 25 for e-mail
 - ▶ See <http://www.iana.org/assignments/port-numbers>
- ▶ Well-known vs. ephemeral (temporary) ports
 - ▶ Server has a well-known port (e.g., port 80)
 - ▶ Between 0 and 1023 (requires root to use)
 - ▶ Client picks an unused temporary port
 - ▶ Between 1024 and 65535

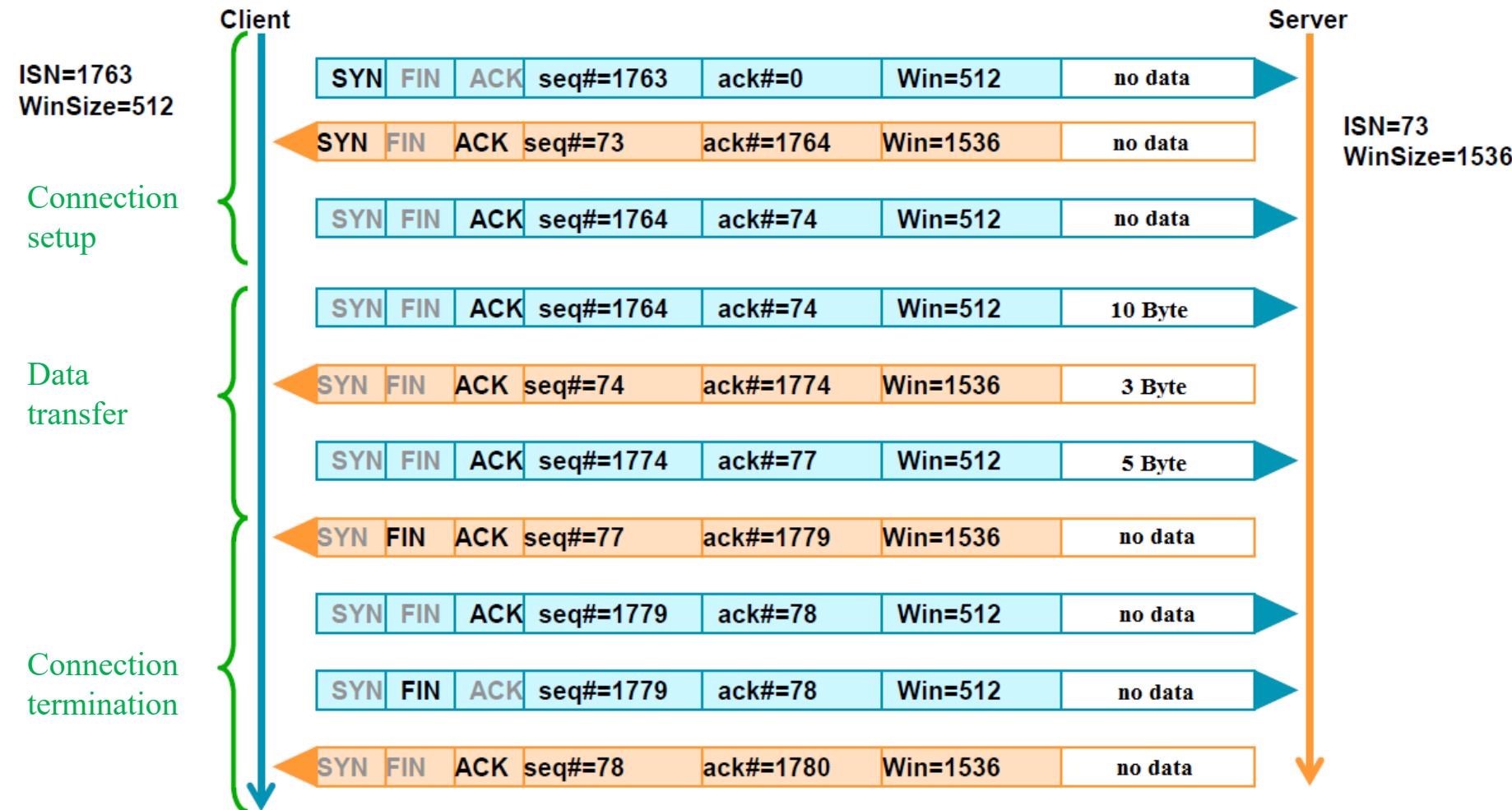


What is a ‘connection’?

- ▶ "5 tuple" uniquely identifies traffic between hosts:
 - ▶ An IP-address and port-number for the ‘client’
 - ▶ An IP-address and port-number for the ‘server’
 - ▶ + underlying transport protocol (e.g., TCP or UDP)



TCP - Normal Procedure (Simplified)



Connection Establishment (cont)

Three way handshake:

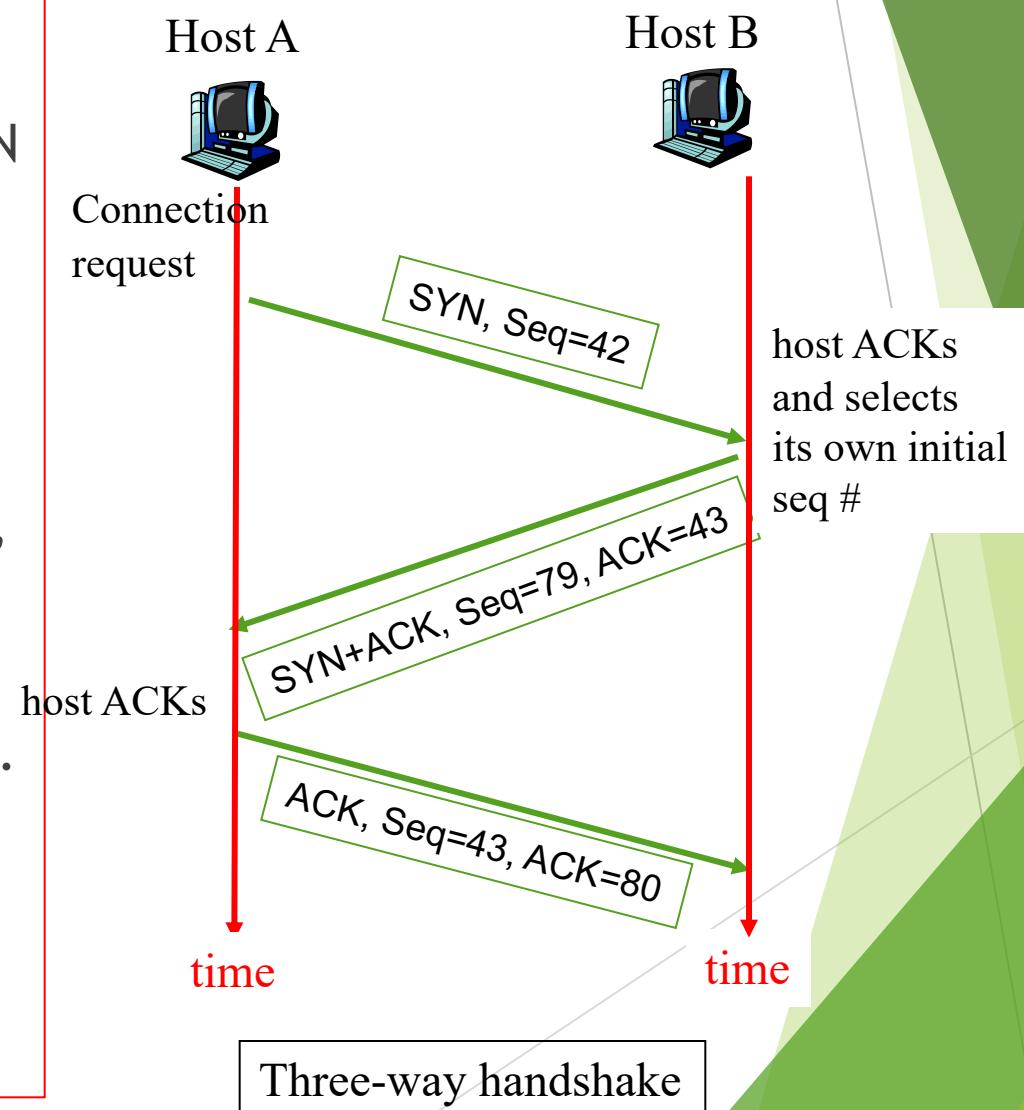
Step 1: client host sends TCP SYN segment to server

- ▶ specifies a **random** initial seq #
- ▶ no data

Step 2: server host receives SYN, replies with SYNACK segment

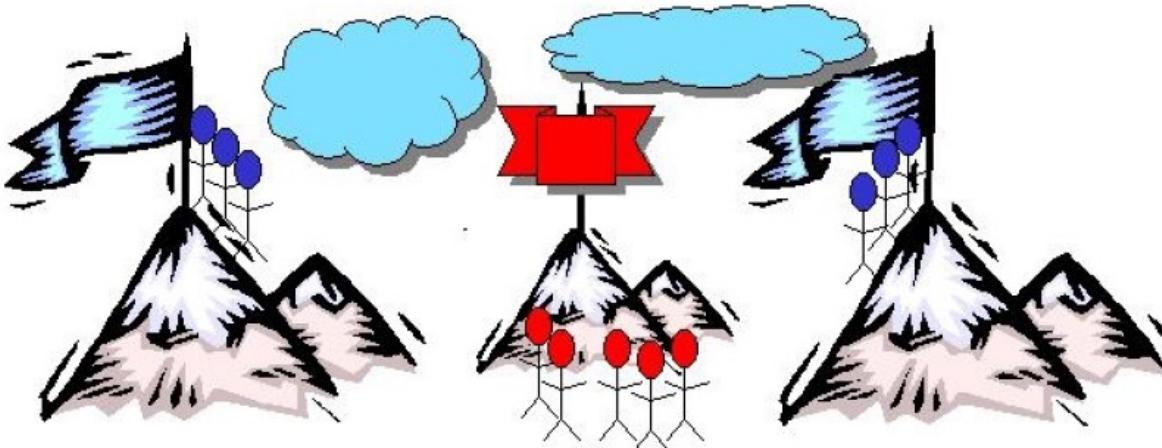
- ▶ server allocates buffers
- ▶ specifies server initial seq. #

Step 3: client receives SYNACK, replies with ACK segment, which may contain data

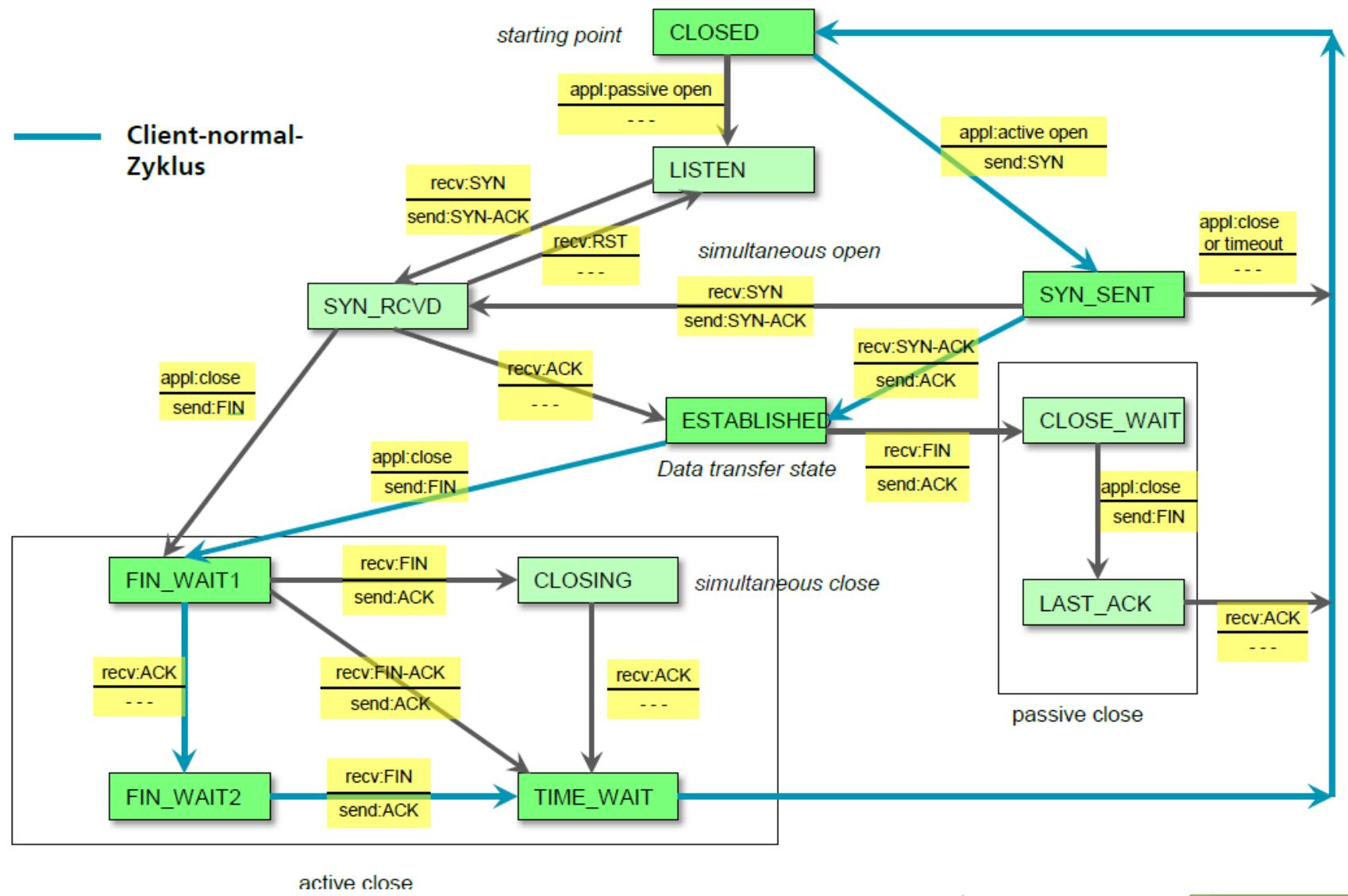


Two-Army Problem

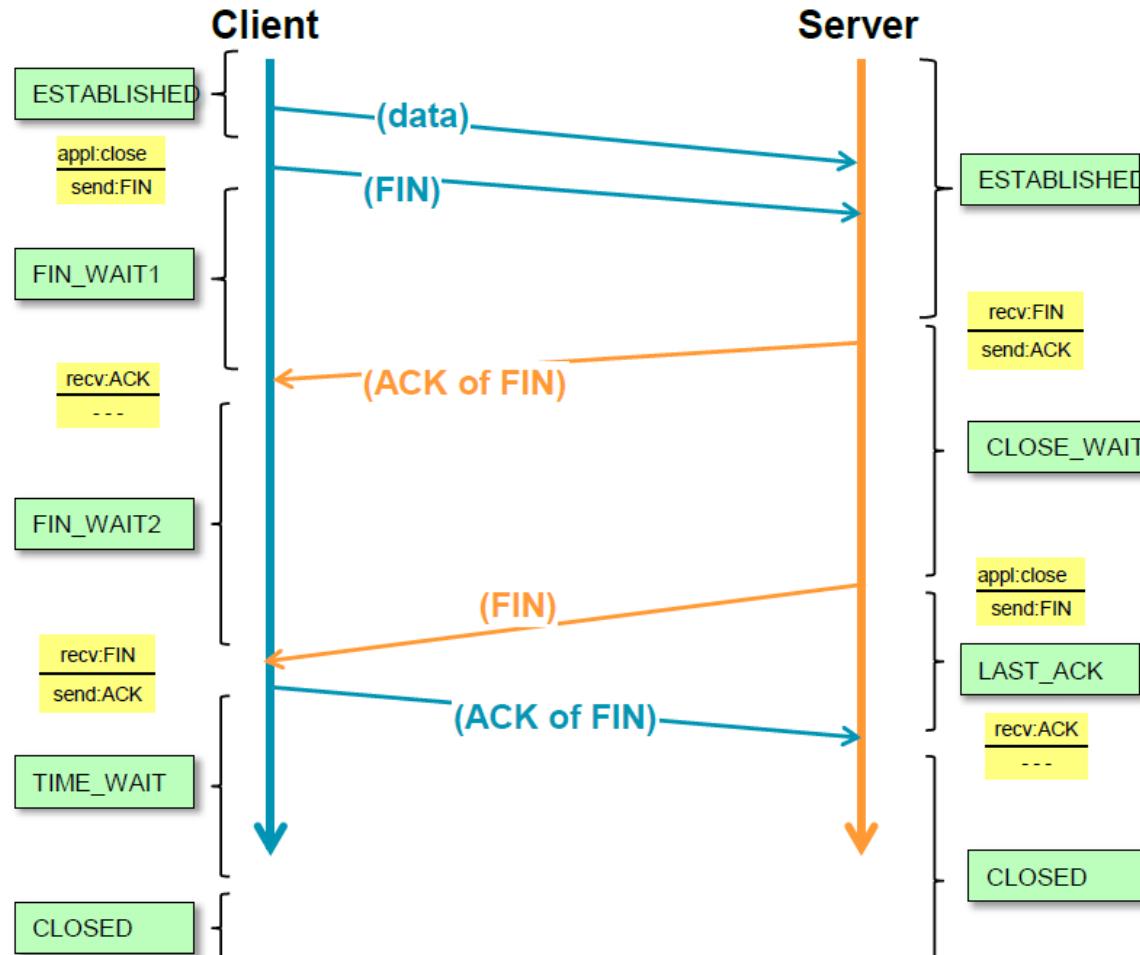
- ▶ Imagine that one army (e.g., red army) is encamped in a valley. On both the surrounding hill sides is opposite army (e.g., blue army). The red army is larger than either of the blue armies alone, but together they are larger than the red army.
- ▶ If either blue army attacks by itself, it will be defeated, but if the blue armies attack simultaneously, they will be victorious.
- ▶ The question is: Does a protocol exist that allows the blue armies to win?



TCP Connection Management



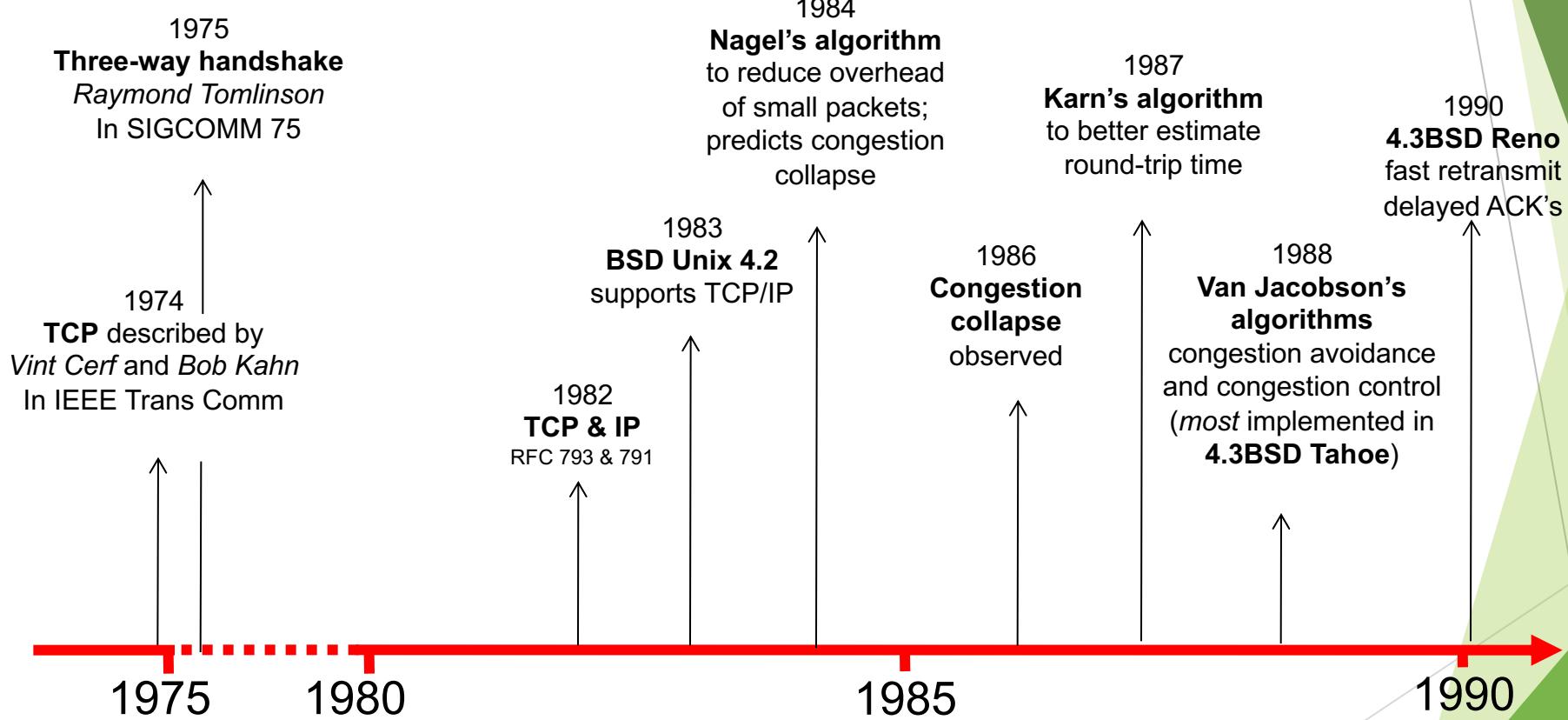
Regular TCP Connection Termination



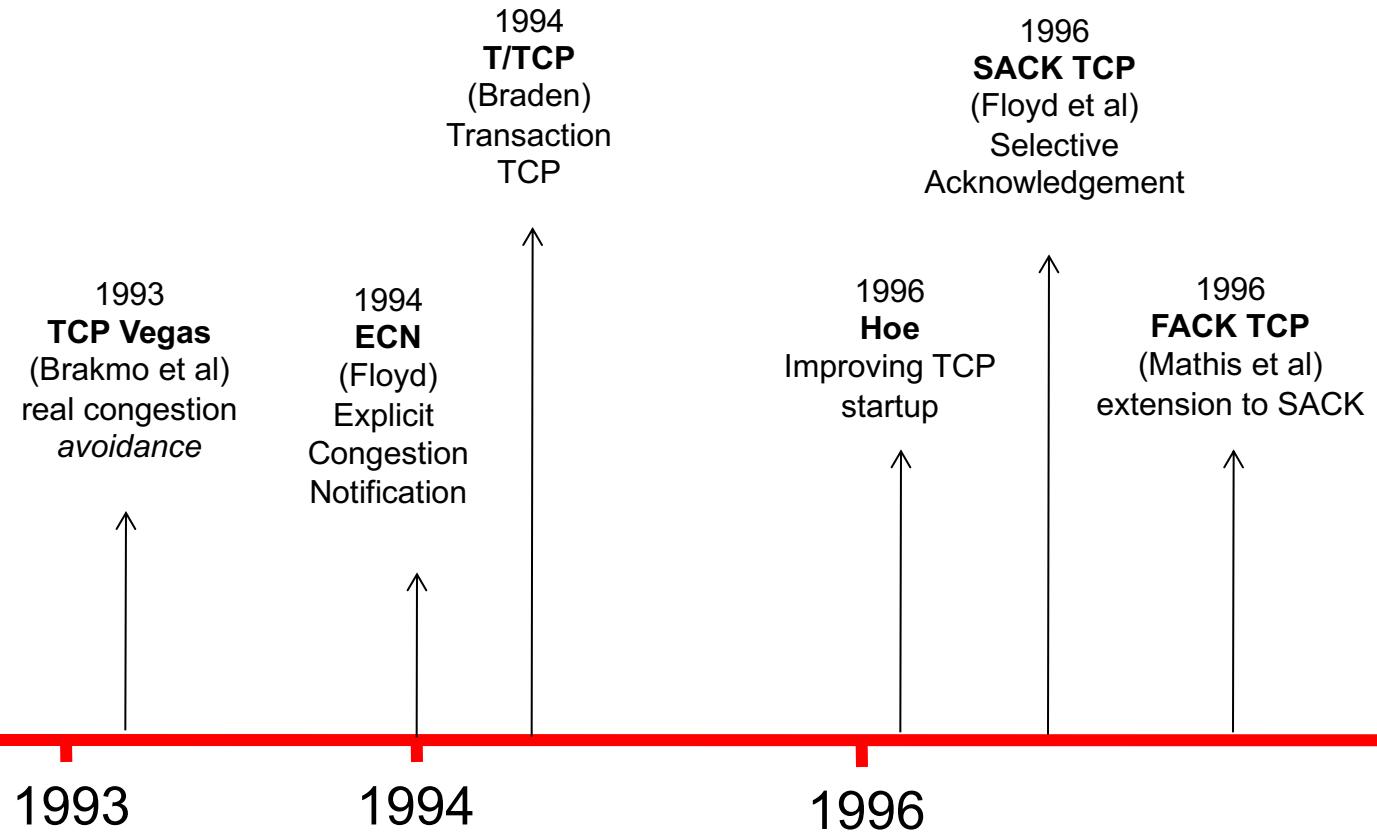
Terminating a connection - basic types and problem definition

- ▶ Basic types
 - **Reset:** The connection is terminated by an "asynchronous" side. (Data loss problem)
 - **Unilateral closing:** The connection signals to the other side that it is not sending any further data. (Problem point: the other side can send further data)
 - **Asymmetrical disconnection:** One of the two computers closes its connection on one side and disconnects the entire connection at the same time.
 - **Symmetrical disconnection:** Either of the two sides clears "their" unidirectional connection as soon as both sides have agreed to clear the connection (-> TCP approach)
- ▶ Problem with the last ACK: How do I safely terminate a connection?
 - Is a three-way handshake enough?
 - Key question: Is the last PDU important for the protocol?)
- ▶ Use of timers

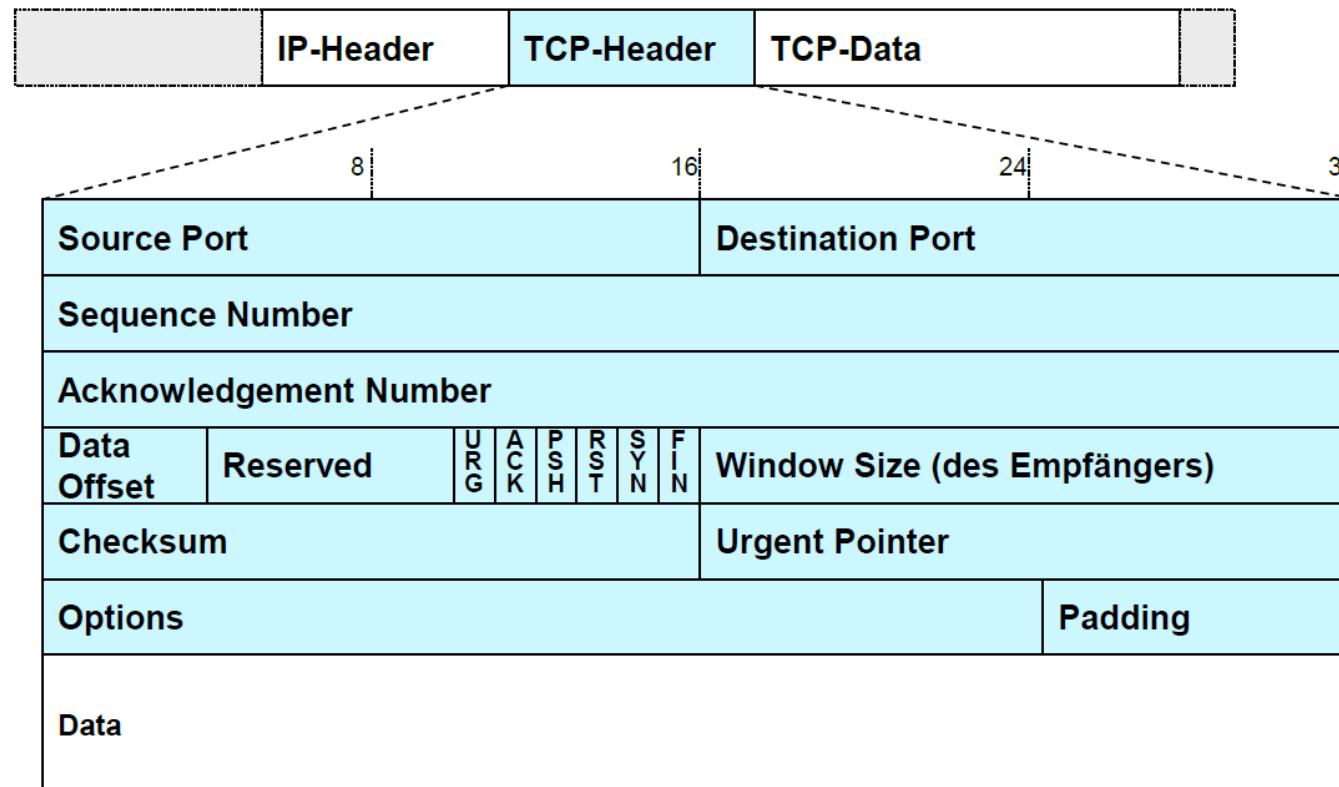
Evolution of TCP



TCP Through the 1990s



TCP Header



Structure of the TCP header

- **Source Port:** 16-bit field for local TCP user (application)
- **Destination Port:** 16-bit field for remote TCP user (application)
- **Sequence number:** Position of the current block in the current stream.
Points to the first new byte.
- **Acknowledgment Number:** Next sequence number expected.
Previous numbers were received correctly.
- **Data Offset:** Length of the TCP header in 32-bit blocks
- **Flags:**
 - URG Urgent-Pointer-Flag (for priority data)
 - ACK Acknowledgement-Flag (confirmation)
 - PSH Push-Flag (direct forwarding to application)
 - RST Reset-Flag (Termination of the connection due to an error)
 - SYN Synchronization-Flag (Synchronization process)
 - FIN Final-Flag (proper disconnection)

Structure of the TCP header (Cont.)

- **Window Size:** The buffer size still available on the receiving side, important information for the transmitter, is used for flow control
- **Urgent Pointer:** Pointer to the end of urgent data
- **Ckecksum:** Check sum of headers, data and a pseudo header to protect against misdirected segments
- **Options:** MSS (Maximum Segment Size) when establishing a connection
- **Padding:** Fill data for 32 bit limit

TCP Retransmission and Flow Control (Part II)

CEN 223 - Internet Communication

Assoc. Prof. Dr. Fatih ABUT
Department of Computer Engineering
Çukurova University

What TCP does for you?

- ▶ Stream-based in-order delivery
 - ▶ Segments are ordered according to sequence numbers
 - ▶ Only consecutive bytes are delivered
- ▶ Reliability
 - ▶ Missing segments are detected (ACK is missing) and retransmitted
- ▶ Flow control
 - ▶ Receiver is protected against overload (window based)
- ▶ Congestion control
 - ▶ Network is protected against overload (window based)
 - ▶ Protocol tries to fill available capacity
- ▶ Connection handling
 - ▶ Explicit establishment + teardown
- ▶ Full-duplex communication
 - ▶ e.g., an ACK can be a data segment at the same time (piggybacking)

TCP Retransmission Strategy

- ▶ TCP relies on positive acknowledgements
 - ▶ Retransmission on timeout
- ▶ Timer associated with each segment as it is sent
- ▶ If timer expires before acknowledgement, sender must retransmit

- ▶ TCP uses an *adaptive retransmission algorithm* because internet delays are so variable
- ▶ *Round trip time* of each connection is recomputed every time an acknowledgment arrives
- ▶ Timeout value is adjusted accordingly

Retransmit Timeout Mechanism (1)

- ▶ RTO timer value difficult to determine:
 - ▶ too high \Rightarrow bad in case of msg-loss!
 - ▶ too short \Rightarrow risk of false alarms!
 - ▶ General consensus: too short is worse than too long; use conservative estimate
- ▶ Calculation: measure RTT (Seg# ... ACK#)
- ▶ Original suggestion in RFC 793 (1981)
 - ▶ Exponentially Weighed Moving Average (EWMA)
 - ▶ $SRTT_{new} = \alpha * SRTT_{old} + (1-\alpha) * RTT_{current}$
 - ▶ $RTO = \min(UBOUND, \max(LBOUND, \beta * SRTT))$

SRTT: Smoothed Round Trip Time

α : Smoothing factor (typically 0.8-0.9)

β : Variance factor (typically 2)

Retransmit Timeout Mechanism (2)

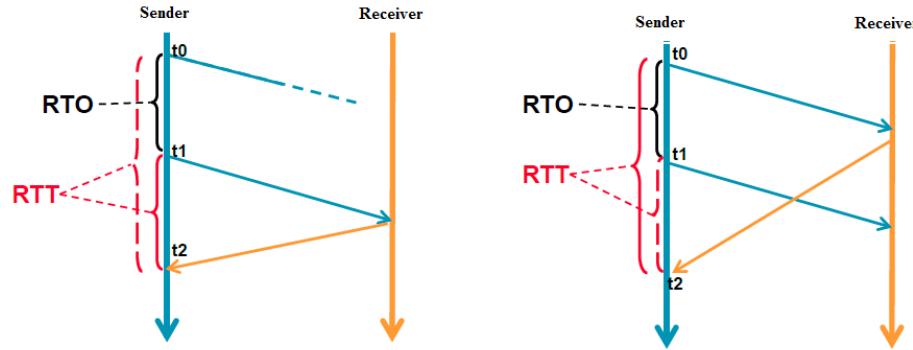
- ▶ Depending on variation, this RTO may be too small or too large; thus, final algorithm includes deviation
- ▶ Key observation:
 - ▶ Original smoothed RTT can't keep up with wide/rapid variations in RTT
- ▶ Jacobson/Karel's Retransmission Timeout (1988)
 - ▶ $SRTT_{new} = \alpha * SRTT_{old} + (1-\alpha) * RTT_{current}$
 - ▶ $SDEV_{new} = \beta * SDEV_{old} + (1 - \beta) * [abs(RTT_{current} - SRTT_{new})]$
 - ▶ $RTO_{new} = SRTT_{new} + \gamma * SDEV_{new}$

SDEV: Smoothed deviation

β : Smoothing factor for standard deviation (typically 0.8-0.9)

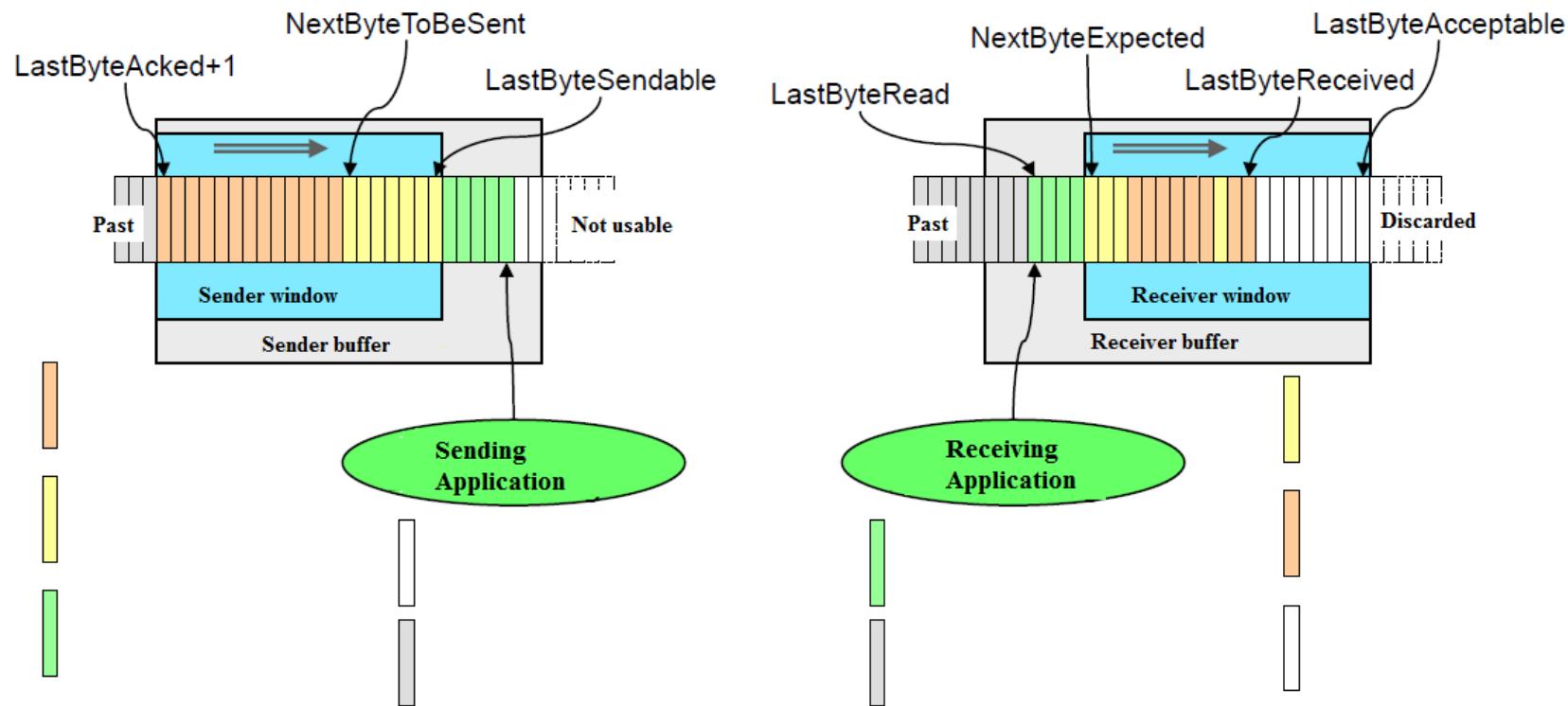
γ : Adjustment factor (typically 4)

RTO calculation



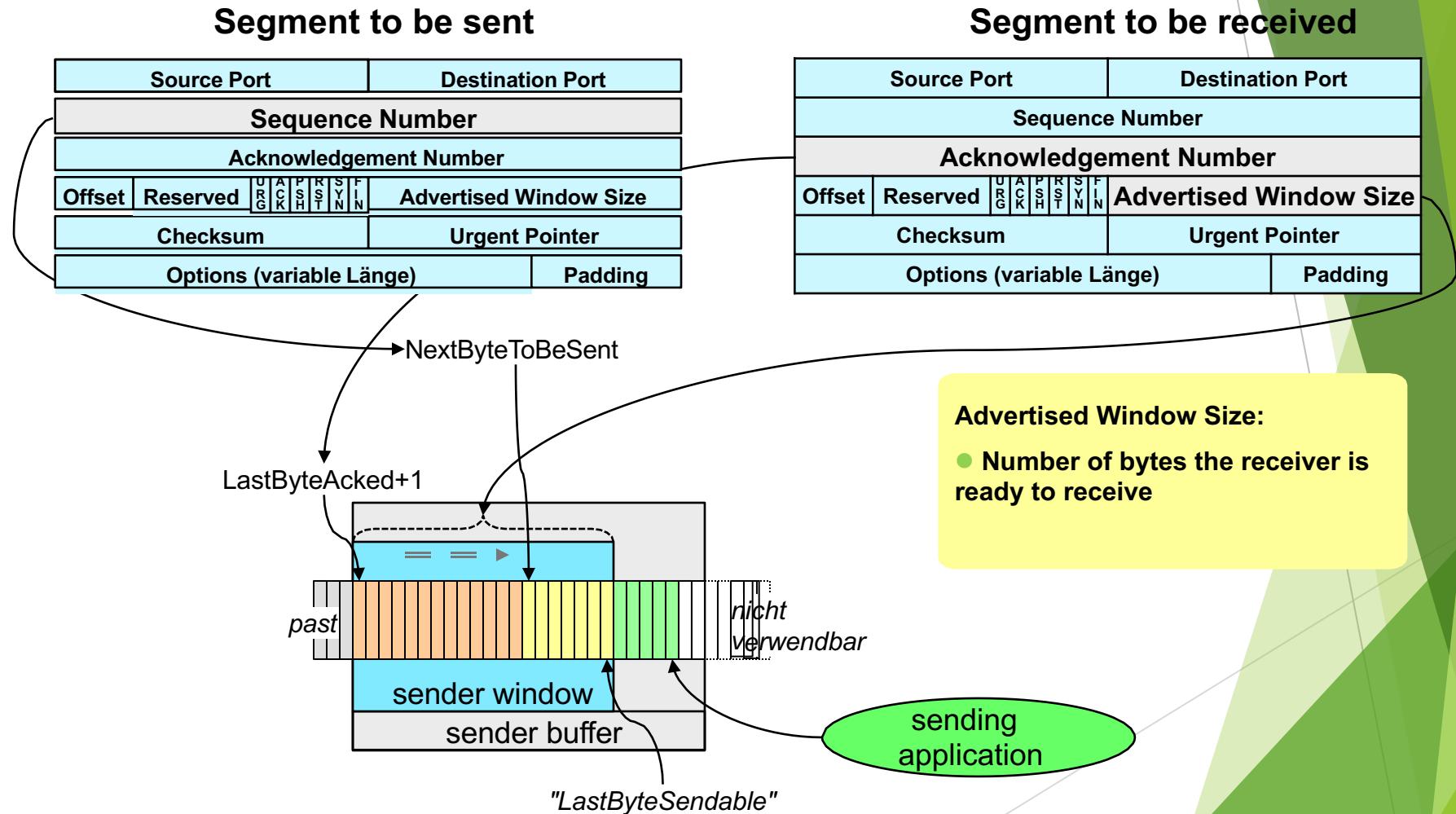
- ▶ Problem: retransmission ambiguity
 - ▶ Segment #1 sent, no ACK received → segment #1 retransmitted
 - ▶ Incoming ACK #2: cannot distinguish whether original or retransmitted segment #1 was ACKed
 - ▶ Thus, cannot reliably calculate RTO!
- ▶ Solution [Karn/Partridge]: ignore RTT values from retransmits
 - ▶ Problem: RTT calculation especially important when loss occurs; sampling theorem suggests that RTT samples should be taken more often
- ▶ Solution: Timestamps option
 - ▶ Sender writes current time into packet header (option)
 - ▶ Receiver reflects value
 - ▶ At sender, when ACK arrives, $RTT = (\text{current time}) - (\text{value carried in option})$

Sliding Window Management

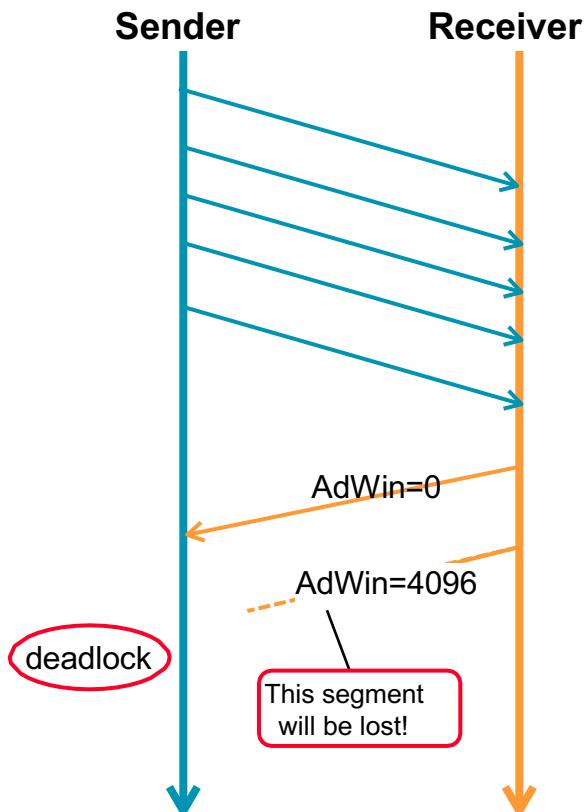


- ▶ Receiver “grants” credit (receiver window, rwnd)
 - ▶ sender restricts sent data with window
- ▶ Receiver buffer not specified
 - ▶ i.e. receiver may buffer reordered segments (i.e. with gaps)

TCP flow control - Segments on the sender side



TCP Flow Control - "Zero advertised window size"



The solution of the problem:

- Sender uses a persistent timer to periodically send a zero window probe segment as soon as the receiver window is closed.

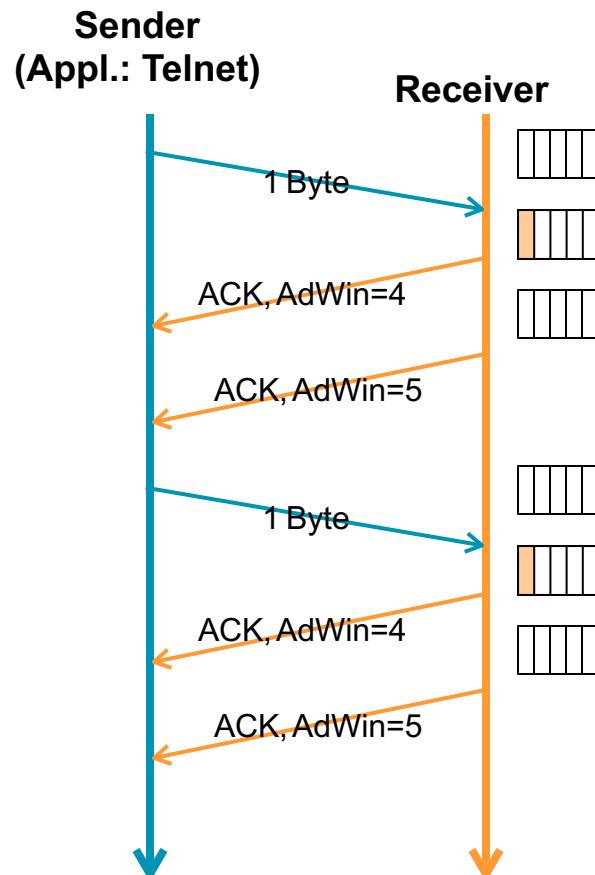
(Algorithm: Exponential back-off algorithm: start value 1.5 seconds, doubling after each Ack until limit, e.g., 60 s, is reached.

→ 1,5 3 6 12 24 48 60 60 60 60 60)

- Probe segment does not contain any user data. Specification explicitly allows sample segments to be sent even after the receiver window is closed.

(Note: As long as the receiver cannot process the sample segment, the ACK contains the sequence number of the last accepted byte)

TCP Flow Control - "Small Packet Problem"



Problem:

- **Sending 2 bytes creates 240 bytes of overhead**
(2 data segments of 40 bytes each,
2 acknowledgments of 40 bytes each,
2 window updates of 40 bytes each)

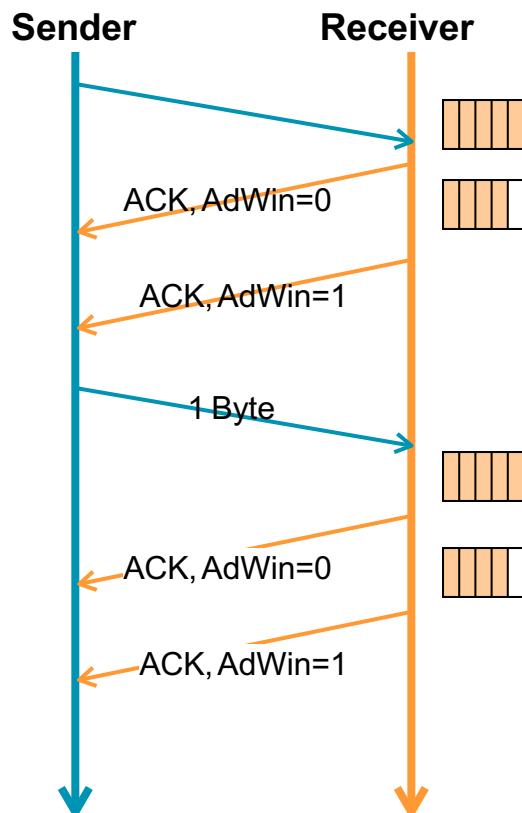
Solutions:

- **Receiver side: delayed Acknowledgement:** Delay of confirmation and window updates by 200 ms (Idea behind: "Piggyback", "Collecting ACKs")
- **Sender side: Nagle's algorithm (RFC 896, 1984):** If data is used byte by byte, only send the first byte, collect bytes and send them in a segment as soon as MSS is reached, or the first byte is confirmed.

Little influence with IP over LAN, but with WAN.

Problems with interactive applications (Nagle's algorithm leads to jerky work e.g. with X-Window). Therefore Nagle's algorithm can be switched off via sockets.

TCP Flow Control - "Silly Window Syndrome (SWS) RFC 813"



Problem:

- Byte-wise processing on the receiver side:
The sender is animated to send small segments.

Solution:

- Sender side:**
(1) Avoid sending small amounts of data.
(2) Nagle's algorithm.
- Receiver side:**
Window updates only for a larger amount (e.g.
if 30% of the receive buffer or 2 MSS are free)

Nagle's Algorithm (1)

- ▶ If there is data to send but the window is open less than MSS, then we may want to wait some amount of time before sending the available data
 - ▶ If we wait too long, then we hurt interactive applications
 - ▶ If we don't wait long enough, then we risk sending a bunch of tiny packets and falling into the silly window syndrome
- ▶ The solution is to introduce a timer and to transmit when the timer expires

Nagle's Algorithm (2)

When the application produces data to send

if both the available data and the window \geq MSS

 send a full segment

else /* window < MSS */

 if there is unACKed data in flight

 buffer the new data until an ACK arrives

 else

 send all the new data now

TCP Acknowledgements (RFC 1122, RFC 2581)

Event	Reaction TCP receiver
Arrival of a directly following segment, no gap, all previous segments have already been confirmed.	delayed Acknowledgement: Wait up to 200 ms to see whether a new segment comes, if none comes by then, an ACK must be sent.
Arrival of a directly following segment, no gap, segment waiting for confirmation	Sending a cumulative ACK: Confirmation of several segments with a single acknowledgment
Arrival of an out-of-order segment larger than NextByteExpected (there is a gap).	Sending a duplicate ACK: Repeated sending of the last ACK (= beginning of the gap)
Arrival of a segment that completely or partially fills a gap (so that part of the byte stream is completed).	Immediate Acknowledgement: Immediate sending of an ACK

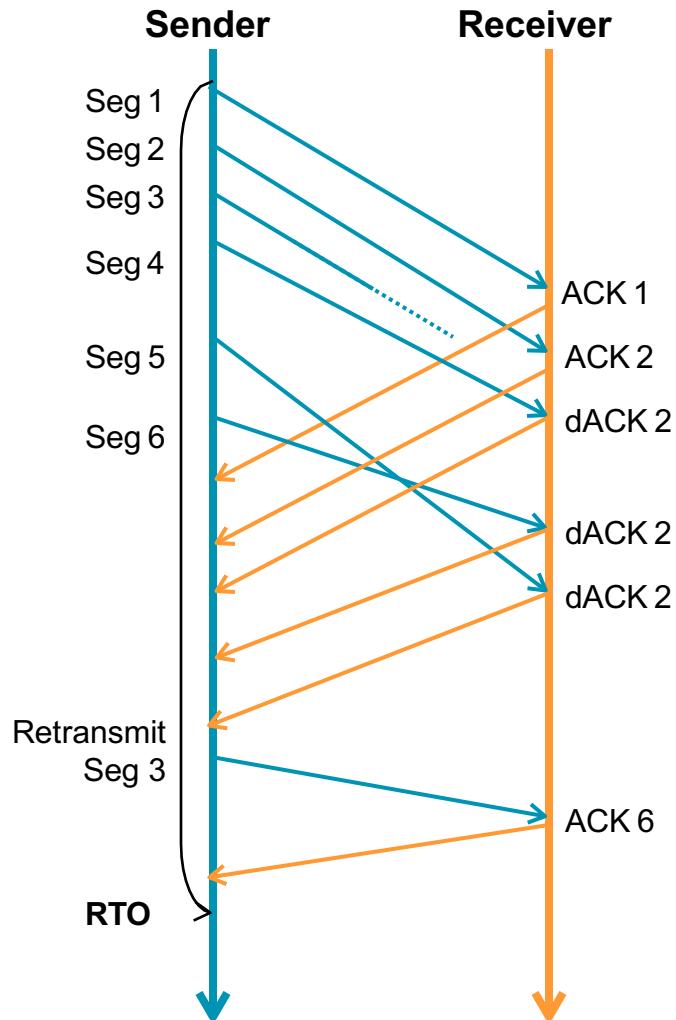
Fast Retransmit (1)

- ▶ Coarse timeouts remained a problem, and **Fast retransmit** was added with TCP Tahoe.
- ▶ Since the receiver responds every time a packet arrives, this implies the sender will see duplicate ACKs.
- ▶ Basic Idea: *use **duplicate ACKs** to signal lost packet.*

Fast Retransmit

Upon receipt of **three** duplicate ACKs, the TCP Sender retransmits the lost packet.

Fast Retransmit (2)



Problem:

- Rough setting of the RTO leads to waiting times in the event of packet loss

Solution:

- In the event of a triple duplicate ACK, retransmission of the missing package without waiting for the retransmission timer to expire

New Problem:

- A lost packet is a sign of network congestion. Therefore, after Fast Retransmit, a reaction to network overload is necessary.

→ Congestion Control

Note:

Fast Recovery: Algorithm to get data flow after Fast Retransmit

SACK (Selective Acknowledgement, RFC2018): Confirmation of the segments actually received

TCP Congestion Control (Part III)

CEN 223 - Internet Communication

Assoc. Prof. Dr. Fatih ABUT
Department of Computer Engineering
Çukurova University

Contents

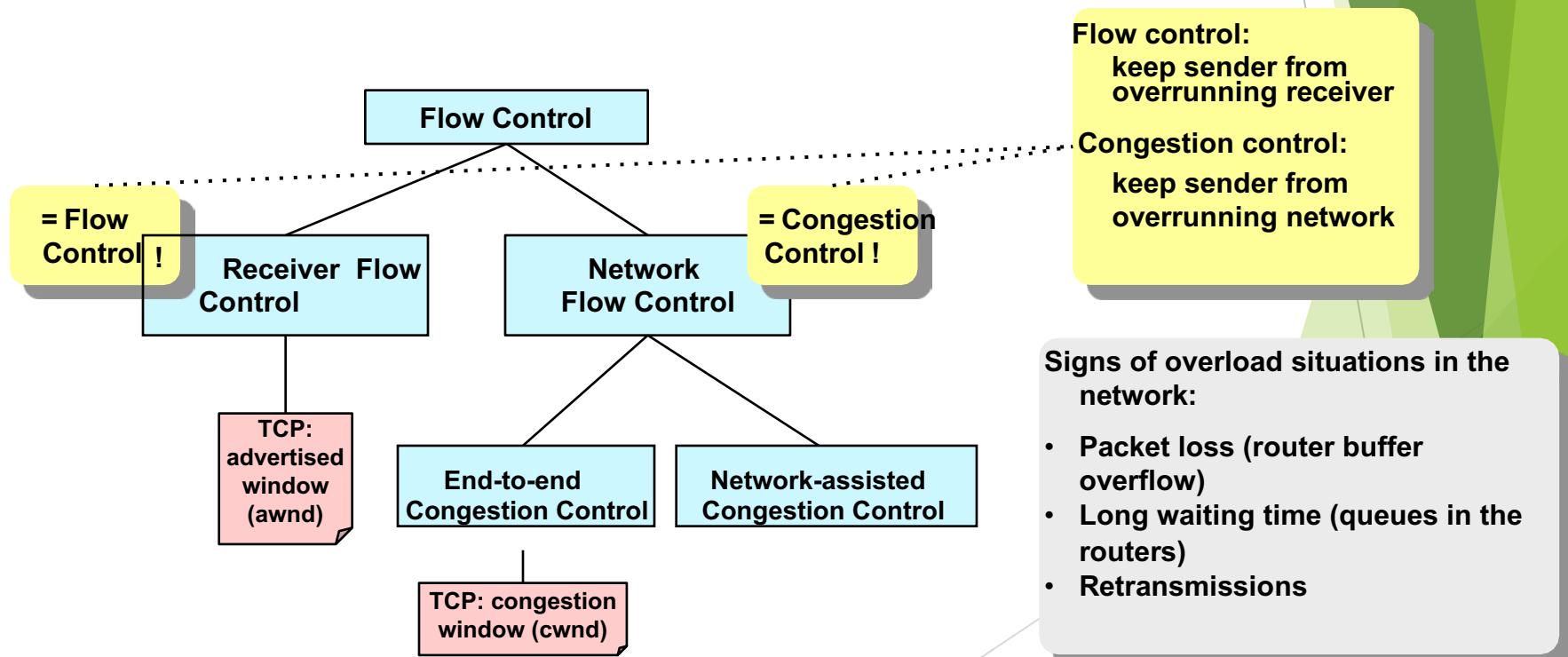
- ▶ Introduction
- ▶ Slow-start
- ▶ Congestion avoidance
 - ▶ Additive Increase
 - ▶ Multiplicative Decrease
- ▶ Fast retransmit
- ▶ Fast recovery
- ▶ TCP Fairness

Definition of terms: congestion and flow control

Definition (according to Steven Low):

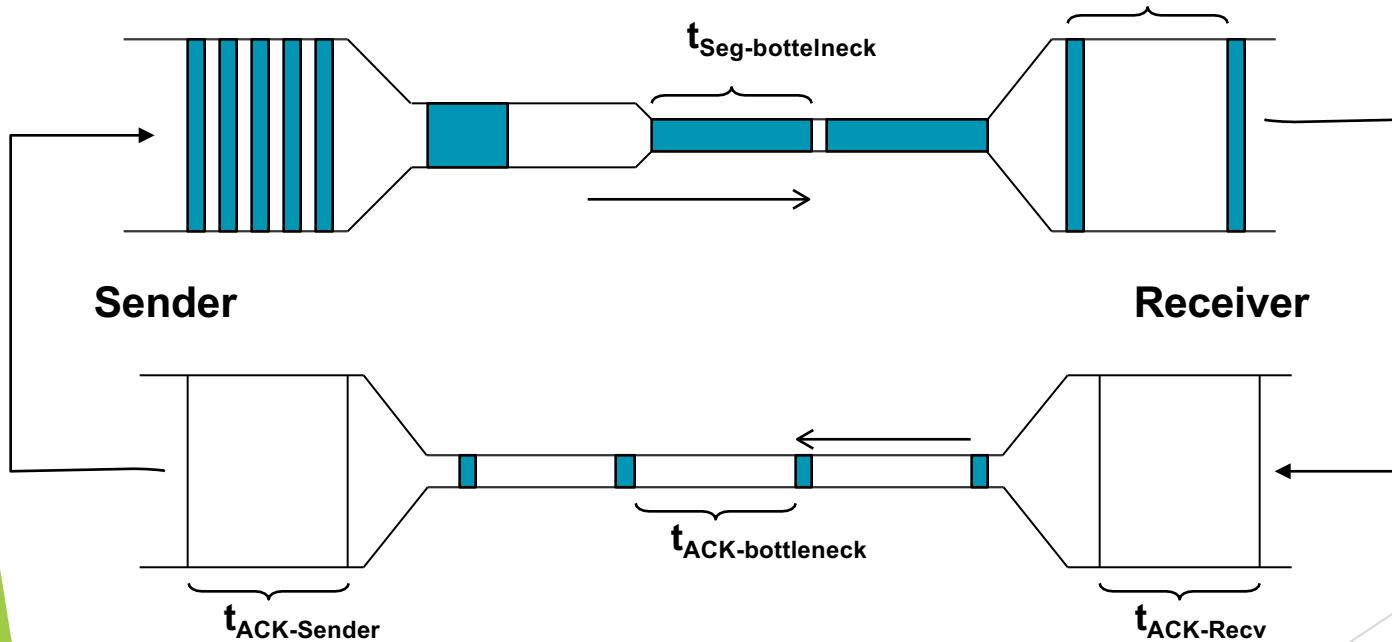
With flow control we refer to all methods for the efficient use of bandwidth (of a connectionless network)

With window flow control we refer to flow control mechanisms that are based on window sizes (= buffer sizes).



Principle: Self-Clocking

Equilibrium through "Conservation of Packets"



TCP
Selfclocking
→ Adaptation of the transmission rate to the slowest connection part

Key question:
Congestion Avoidance:

How is Equilibrium ensured?

TCP Congestion Control

TCP tries to achieve the following goals through congestion control:

- (1) high usage of the network (-> high equilibrium)
- (2) as no overload as possible (-> stable equilibrium)
- (3) fair distribution of the available bandwidth (-> fair equilibrium)

TCP uses two flow control mechanisms for this purpose:

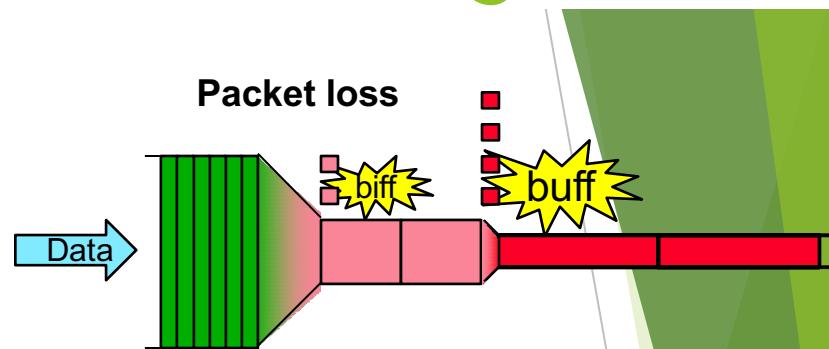
- Receiver Flow Control
 - Avoidance of overload at the receiver
 - Controlled by the receiver
 - Window size used: AdvertisedWindow (awnd)
- Network Flow Control
 - Collects information about the resilience of the network (loss, delay, hints) -> (1)
 - Avoid overloading the network -> (2)
 - Controlled by the transmitter
 - Window size used: CongestionWindow (cwnd)

Window effectively used: $\text{wnd} = \min(\text{awnd}, \text{cwnd})$

Critical phases, equilibrium and algorithms

Start of the TCP transmission

- **Slow Start (Build an equilibrium)**



Maintaining Equilibrium

- **Setting the RTO (Jacobson/Karel Algorithm)**
- **Maximize utilization (→ Additive Increase)**

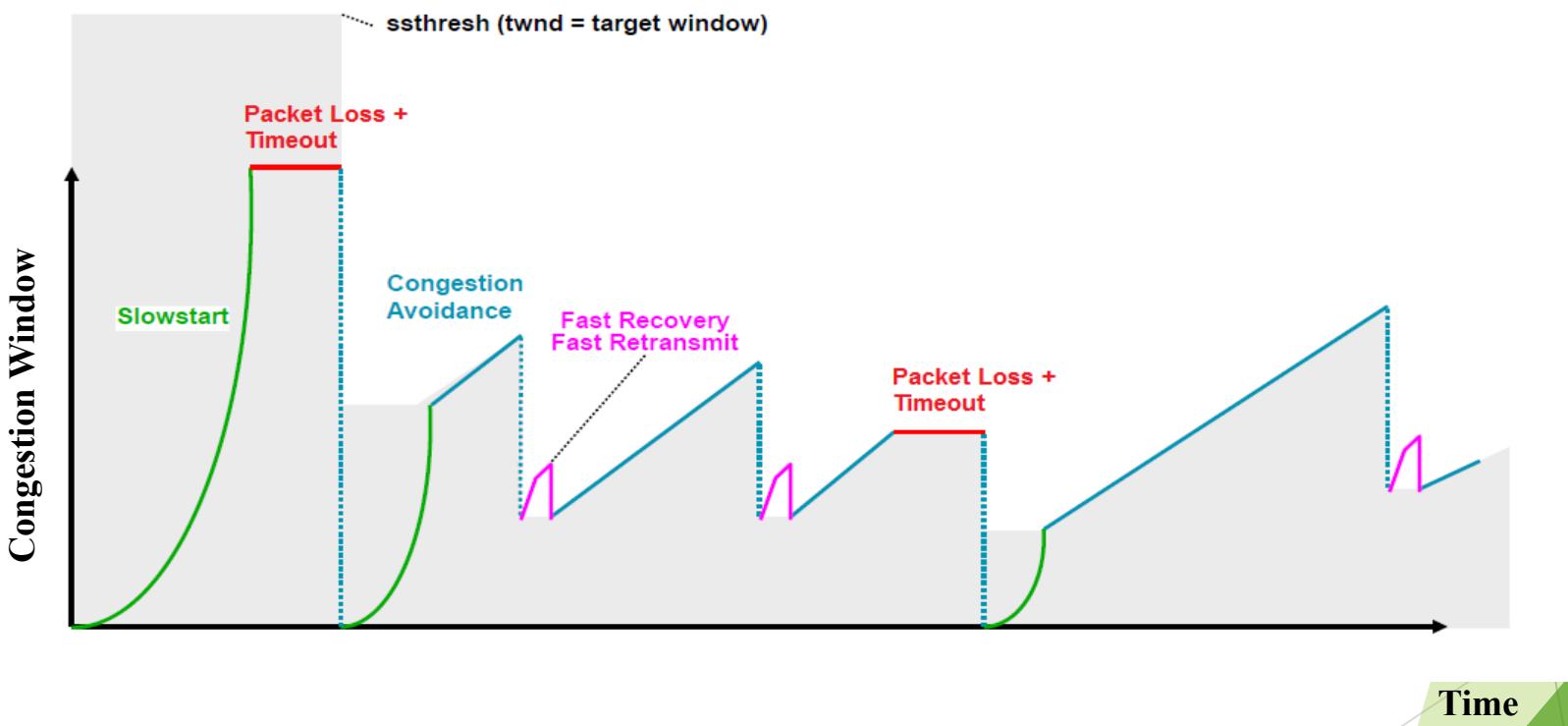
Response to signs of overload

- **Stabilization of the situation (→ Multiplicative Decrease, Karn-Algo.)**
- **Ensuring a continuous, uninterrupted flow of data (→ Fast Retransmit, →Fast Recovery,)**

Fairness of resource allocation

- **Automatically (?) through AIMD (= Additive Increase / Multiplicative Decrease)**

Typical “Sawtooth” Pattern



Slow Start (1)

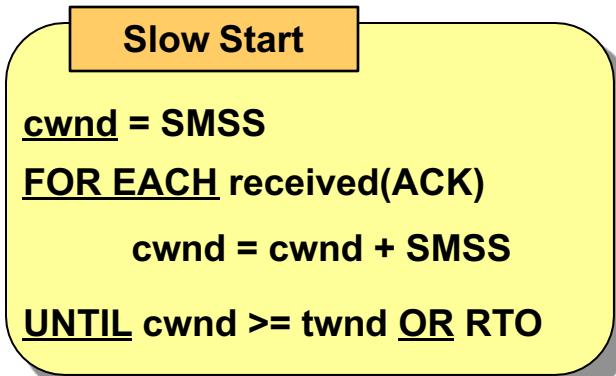
The aim of slow start :

- Rapid introduction of the TCP connection to the Equilibrium strategy:
- Progressive testing of the load capacity of the route between the sender and receiver (up to the overload)
- Packet losses (-> retransmission time out) are evaluated as an indicator of an overload situation. (This is critical for mobile TCP -> low transmission rate)
- Calming of the network after the occurrence of an overload situation (emptying the router queues at the bottleneck)

Application:

- When starting a connection
- After the retransmission timer has expired
- After a long passive TCP phase

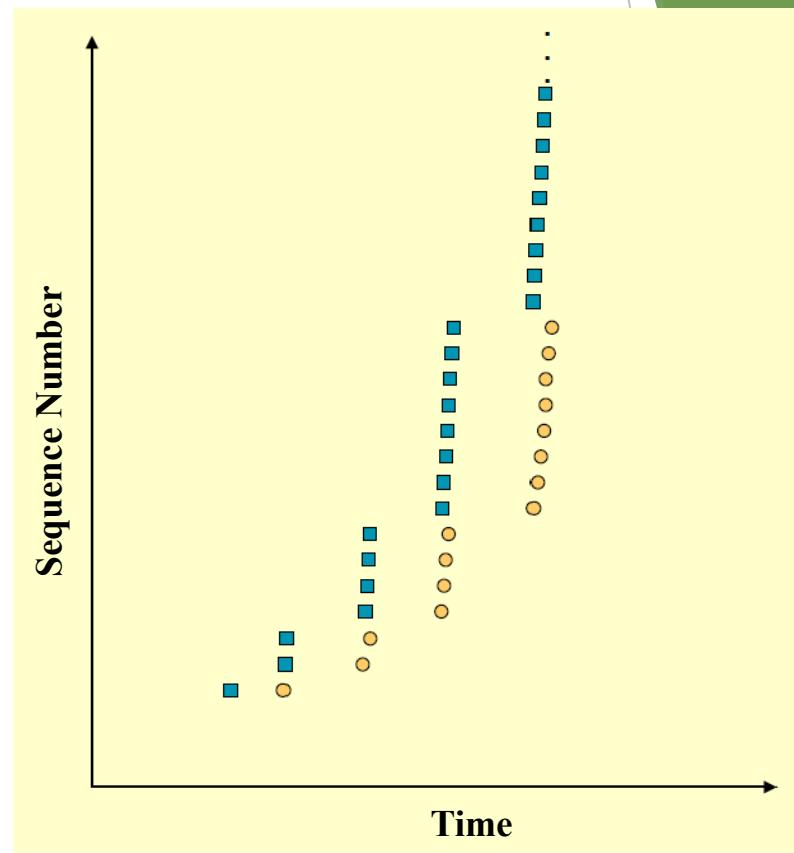
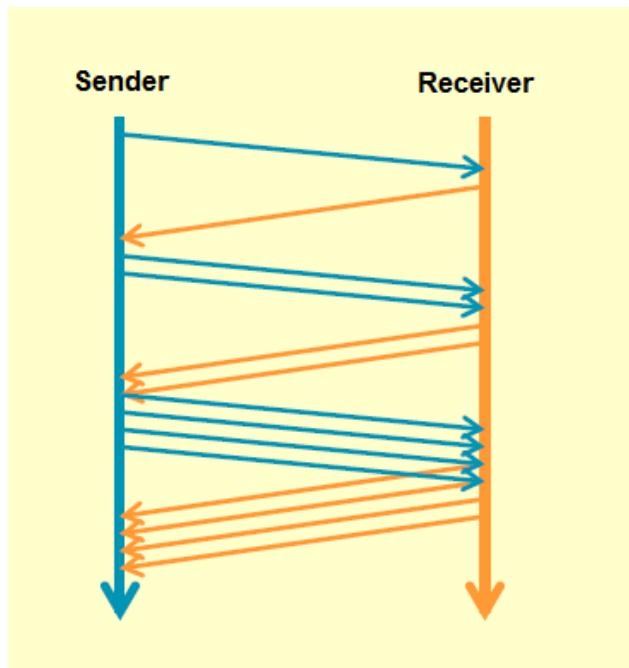
Slow Start (2)



- SMSS Sender Maximum Segment Size
largest segment that a transmitter can send
(536 bytes according to RFC 1122, 1024 bytes for most implementations).
- The start value for cwnd can be set to up to 2 SMSS (according to RFC 2581)
- For the first slow start, the target window ($twnd = ssthresh = \text{slow start threshold}$) can be set as high as you want (e.g. to awnd)
- (approximate) growth of cwnd as a function of time measured in RTT:

$$cwnd(t) = \sum_{i=0}^t 2^i = 2^{t+1} - 1$$

Slow Start (3)



Congestion Avoidance

Aim:

- Best possible utilization of the available bandwidth (greedy), but avoid overload if possible.

Strategy:

- Continuous linear increase in the cwnd as long as no overload situation occurs (adding 1 SMSS per RTT)
→ Additive Increase
- Halving the cwnd in the event of an overload situation. An overload situation is assumed, e.g. after the retransmission timer has expired.
→ Multiplicative Decrease
- AIMD = Additive Increase / Multiplicative Decrease

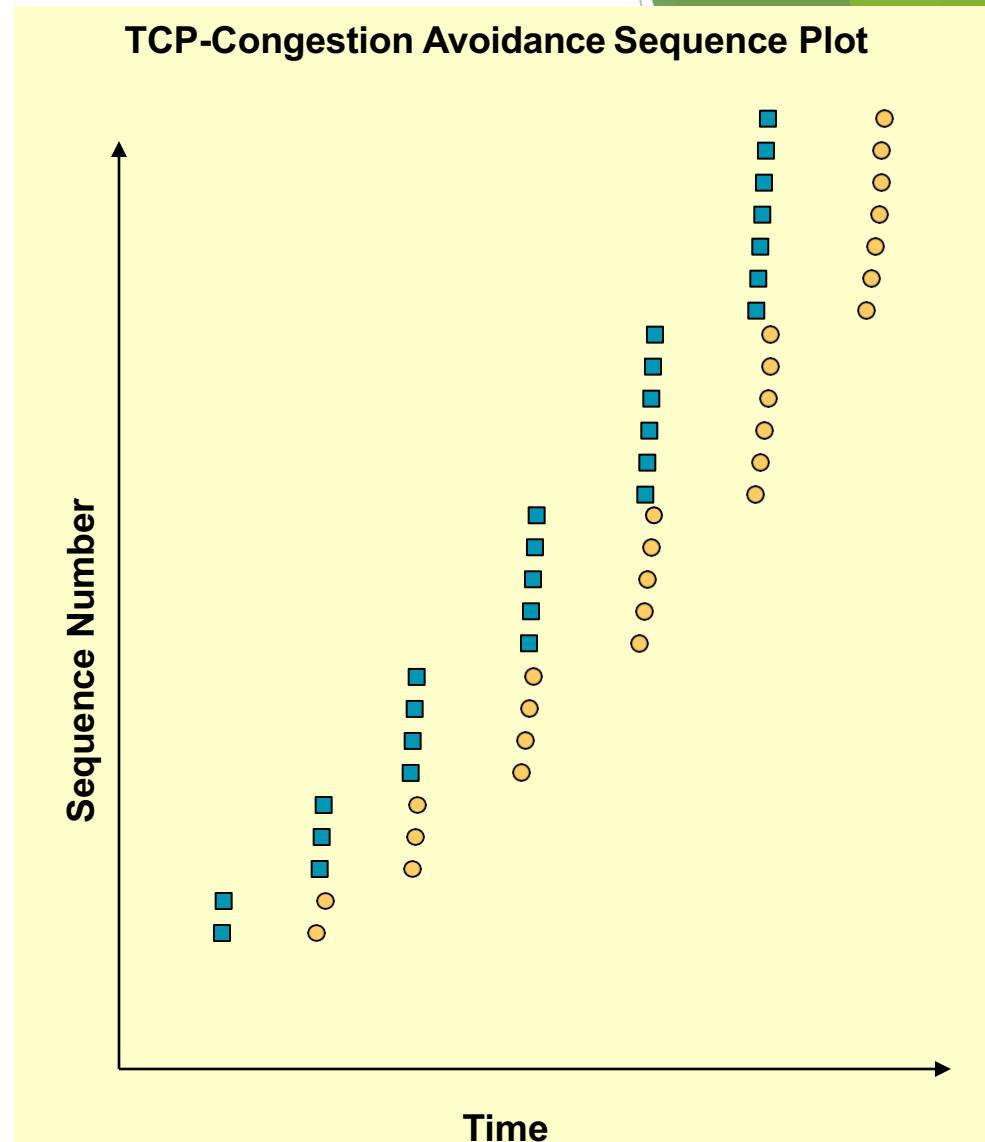
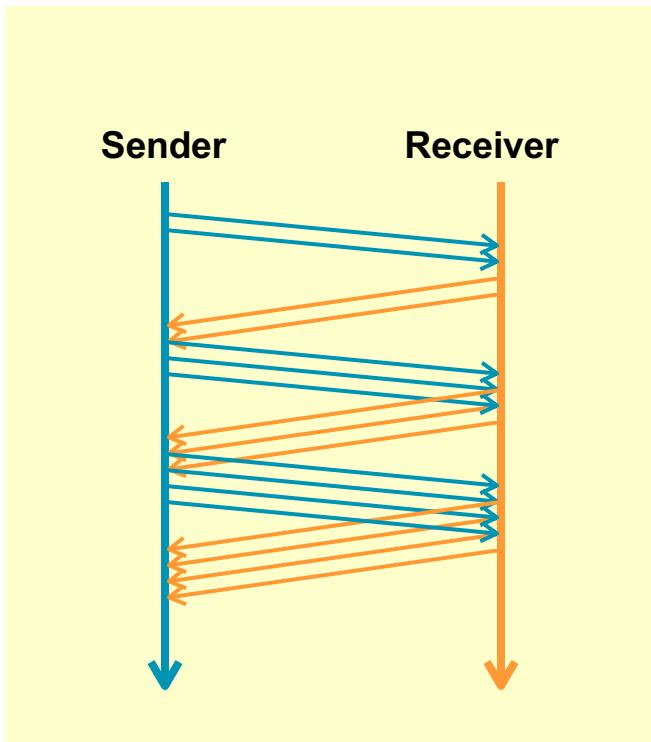
Congestion Avoidance Algorithm

Congestion Avoidance

```
UNTIL LossEvent  
FOR EACH received(ACK)  
    cwnd = cwnd + SMSS * (SMSS/cwnd)  
ENDFOR  
ENDUNTIL  
twnd := cwnd/2  
  
IF intelligent  
    Perform Fast Retransmit or Fast Recovery  
ELSE  
    cwnd :=1  
    Perform SlowStart  
ENDIF
```

Loss Event can be triggered by three
dupACKs or RTO

Illustration of the Congestion Avoidance Algorithm



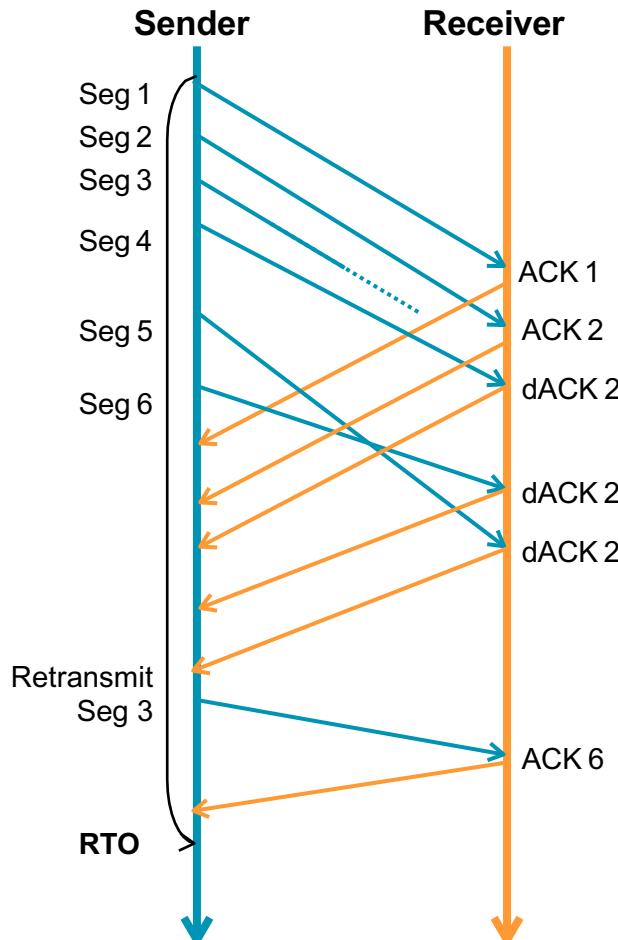
Fast Retransmit (1)

- ▶ Coarse timeouts remained a problem, and **Fast retransmit** was added with TCP Tahoe.
- ▶ Since the receiver responds every time a packet arrives, this implies the sender will see duplicate ACKs.
- ▶ **Basic Idea:** *use **duplicate ACKs** to signal lost packet.*

Fast Retransmit

Upon receipt of **three** duplicate ACKs, the TCP Sender retransmits the lost packet.

Fast Retransmit (2)



Problem:

- Rough setting of the RTO leads to waiting times in the event of packet loss

Solution:

- In the event of a triple duplicate ACK, retransmission of the missing package without waiting for the retransmission timer to expire

New Problem:

- A lost packet is a sign of network congestion. Therefore, after Fast Retransmit, a reaction to network overload is necessary.

→ Congestion Control

Note:

Fast Recovery: Algorithm to get data flow after Fast Retransmit
SACK (Selective Acknowledgement, RFC2018): Confirmation of the segments actually received

Fast Recovery (1)

- ▶ **Fast recovery** was added with TCP Reno.
- ▶ **Basic idea:** When **fast retransmit** detects three duplicate ACKs, start the recovery process from congestion avoidance region and use ACKs in the pipe to pace the sending of packets.

Fast Recovery

After Fast Retransmit, half **cwnd and commence recovery from this point using linear additive increase ‘primed’ by left over ACKs in pipe.**

Fast Recovery (according to RFC 2581)

Fast Recovery

AS-SOON-AS #dACK=3

$\text{twdn} := 0.5 * \text{cwndn}$ -- multiplicative decrease

$\text{cwnd} := \text{twnd} + 3 * \text{SMSS}$ -- inflating

`send(LostSegment)`

END AS-SOON-AS

FOR-EACH-ADDITIONAL dACK

$\text{cwnd} := \text{cwnd} + \text{SMSS}$

END-FOR-EACH ADDITIONAL

AS-SOON-AS regular-ACK

$\text{cwnd} := \text{twnd}$ -- deflating

END AS-SOON-AS

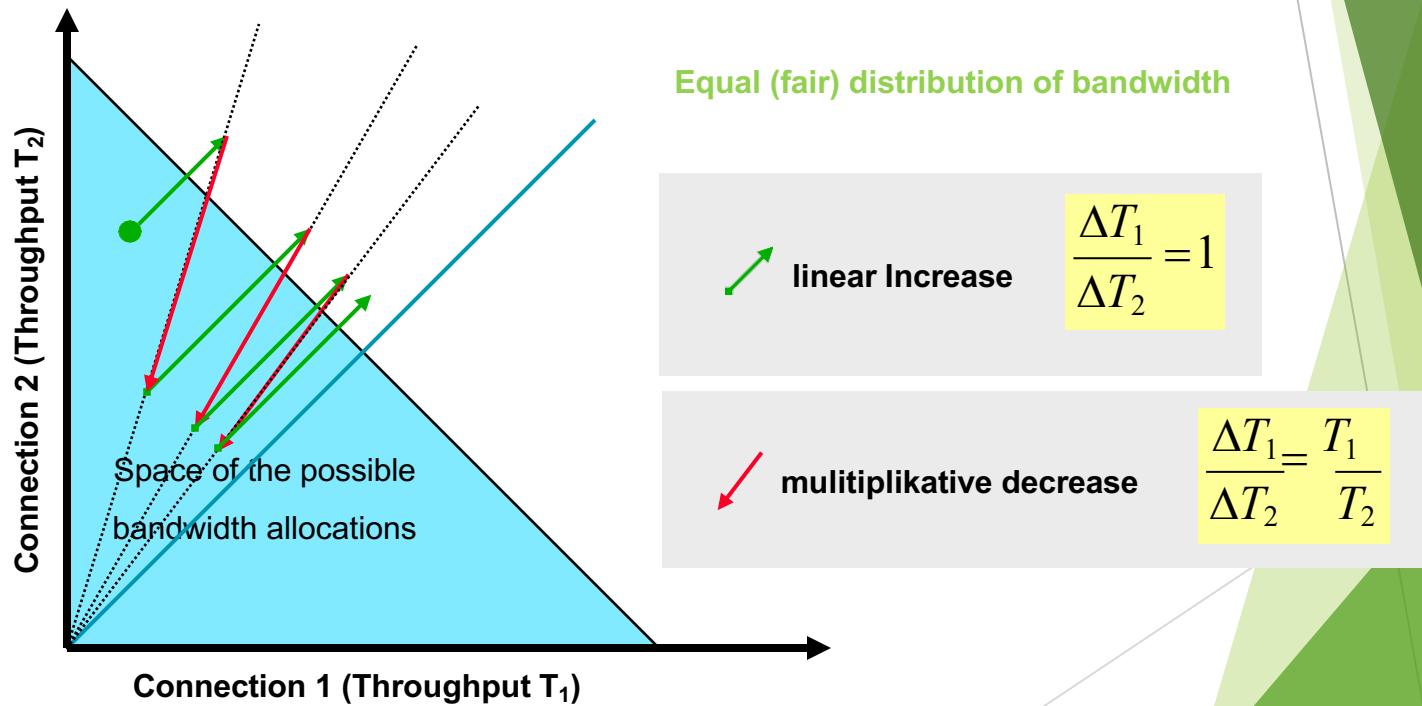
Strategy:

Additional opening of the cwnd in accordance with the "Equilibrium concept" is intended to control continuous uninterrupted shutdown of the pipe to the new setpoint value of the twnd.

TCP Fairness

Two simultaneous TCP sessions

- Additive Increase → Slope 1
- Multiplicative Decrease → Proportional slope



UDP: User Datagram Protocol [RFC 768]

- ▶ “bare bones”, “best effort” transport protocol
- ▶ ***connectionless:***
 - ▶ no handshaking between UDP sender, receiver before packets start being exchanged
 - ▶ each UDP segment handled **independently** of others
- ▶ Just provides multiplexing/demultiplexing

Pros:

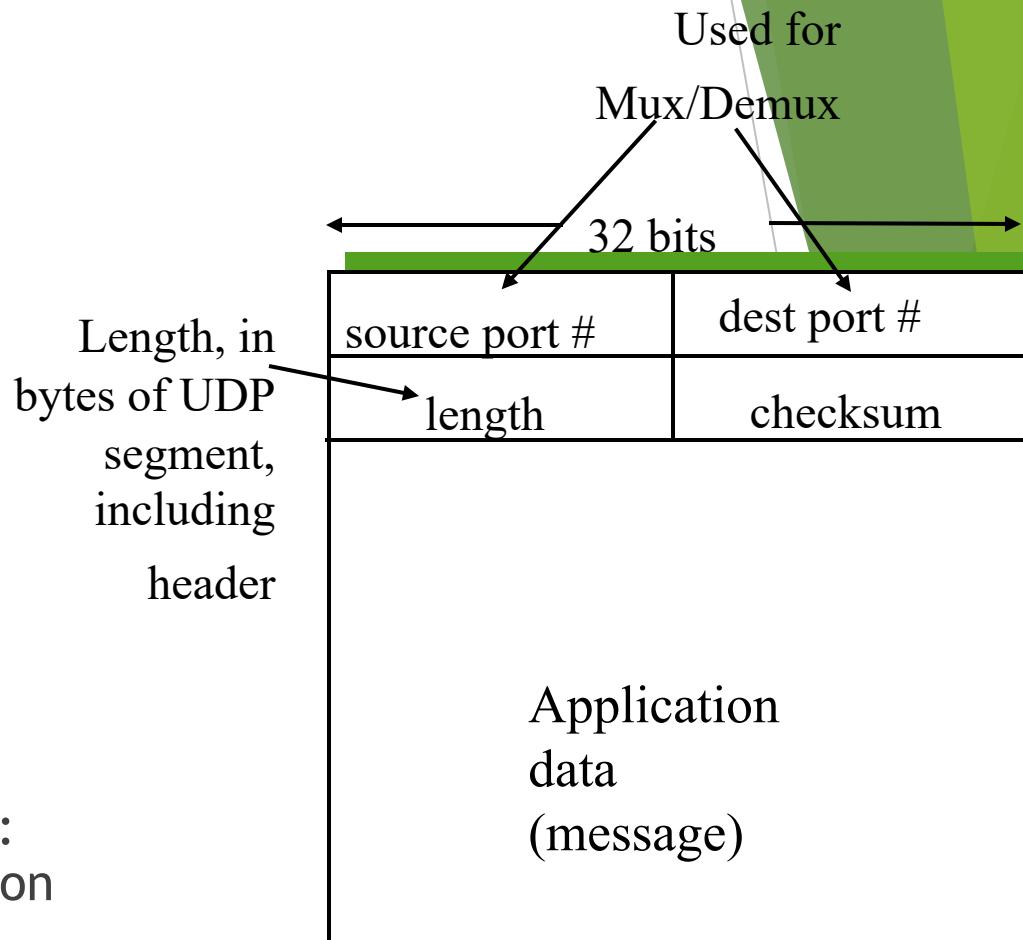
- ▶ No connection establishment
 - ▶ No delay to start sending/receiving packets
- ▶ Simple
 - ▶ no connection state at sender, receiver
- ▶ Small segment header
 - ▶ Just 8 bytes of header

Cons:

- ▶ “best effort” transport service means, UDP segments may be:
 - ▶ lost
 - ▶ delivered out of order to app
- ▶ **no congestion control:** UDP can blast away as fast as desired

UDP more

- ▶ 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

TCP Sockets (Part IV)

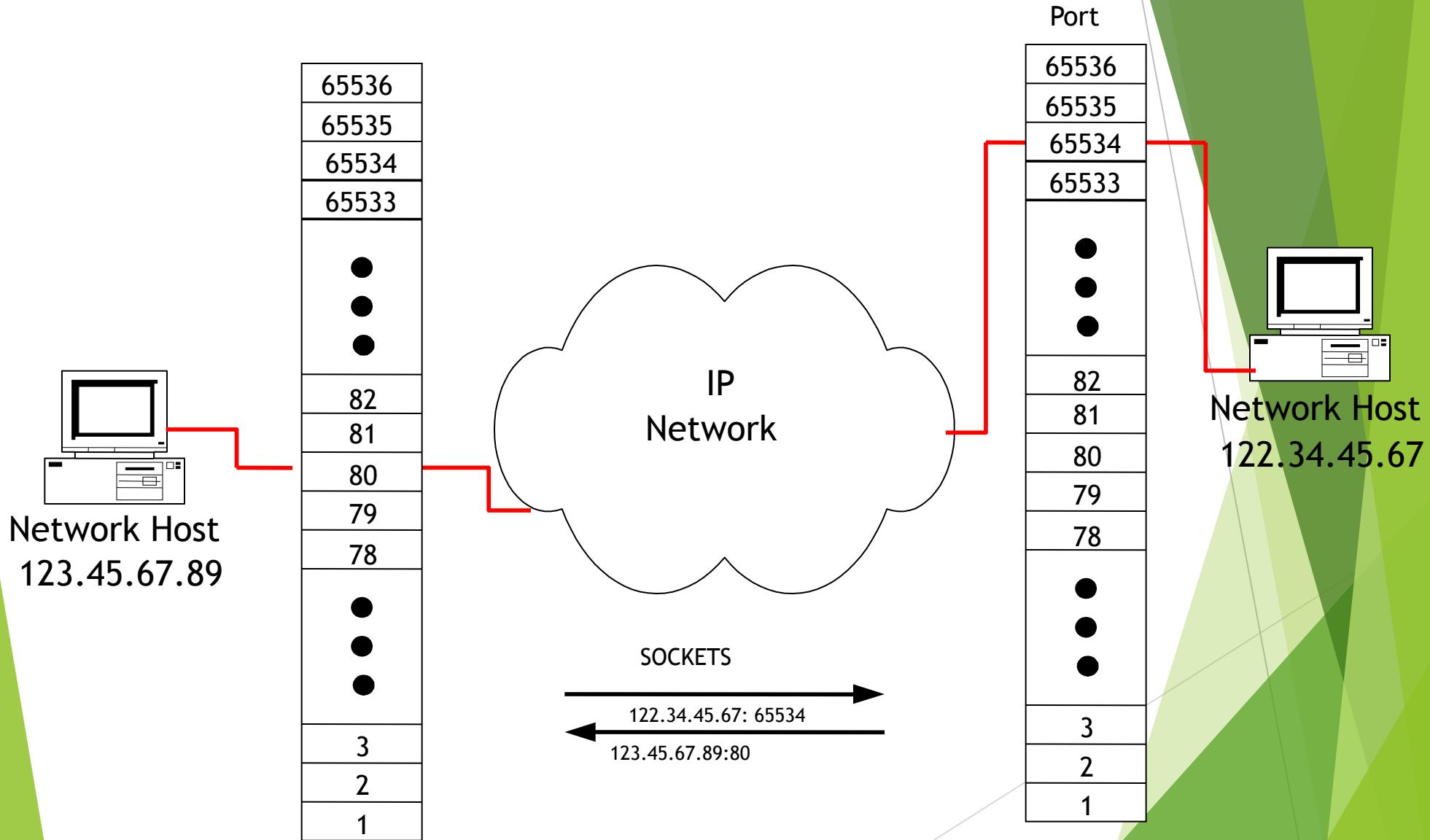
CEN 223 - Internet Communication

Assoc. Prof. Dr. Fatih ABUT
Department of Computer Engineering
Çukurova University

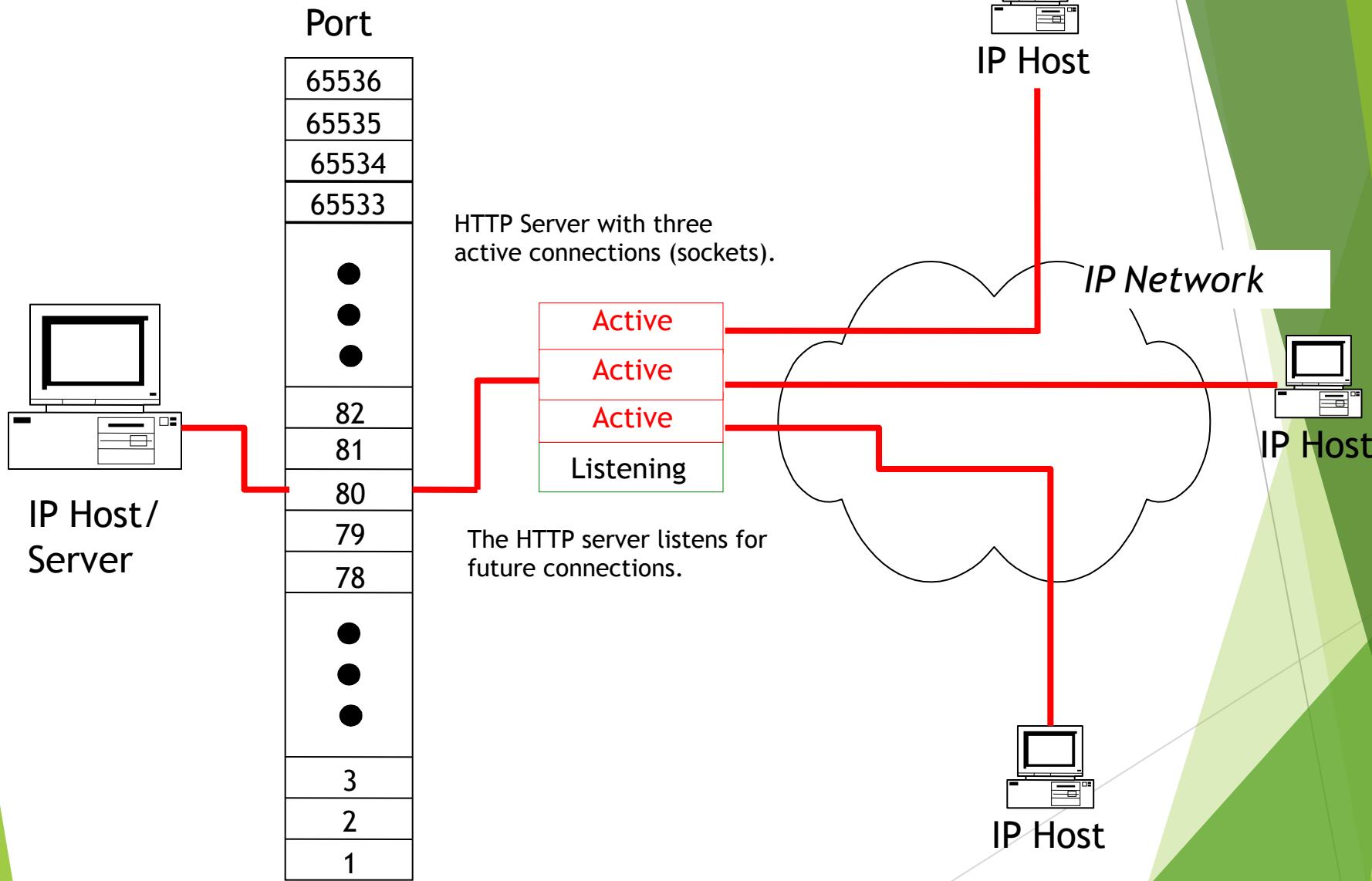
Client Server Communication (1)

- ▶ The transport protocols TCP and UDP were designed to enable communication between network applications
 - Internet host can have several servers running.
 - usually has only one physical link to the “rest of the world”
 - When packets arrive how does the host identify which packets should go to which server?
 - Ports
 - ports are used as logical connections between network applications
 - ▶ 16 bit number (65536 possible ports)
 - demultiplexing key
 - ▶ identify the application/process to receive the packet
 - TCP connection
 - source IP address and source port number
 - destination IP address and destination port number
 - the combination **IP Address : Port Number** pair is called a **Socket**

Client Server Communication (2)



Client Server Communication (3)



Connections

- ▶ A **socket address** is the triplet:

$$\{protocol, local-IP, local-port\}$$

- ▶ example,

$$\{tcp, 130.245.1.44, 23\}$$

- ▶ A **connection** is the 5-tuple that completely specifies the two end-points that comprise a connection:

$$\{protocol, local-IP, local-port, remote-IP, remote-port\}$$

- ▶ example:

$$\{tcp, 130.245.1.44, 23, 130.245.1.45, 1024\}$$

Socket Domain Families

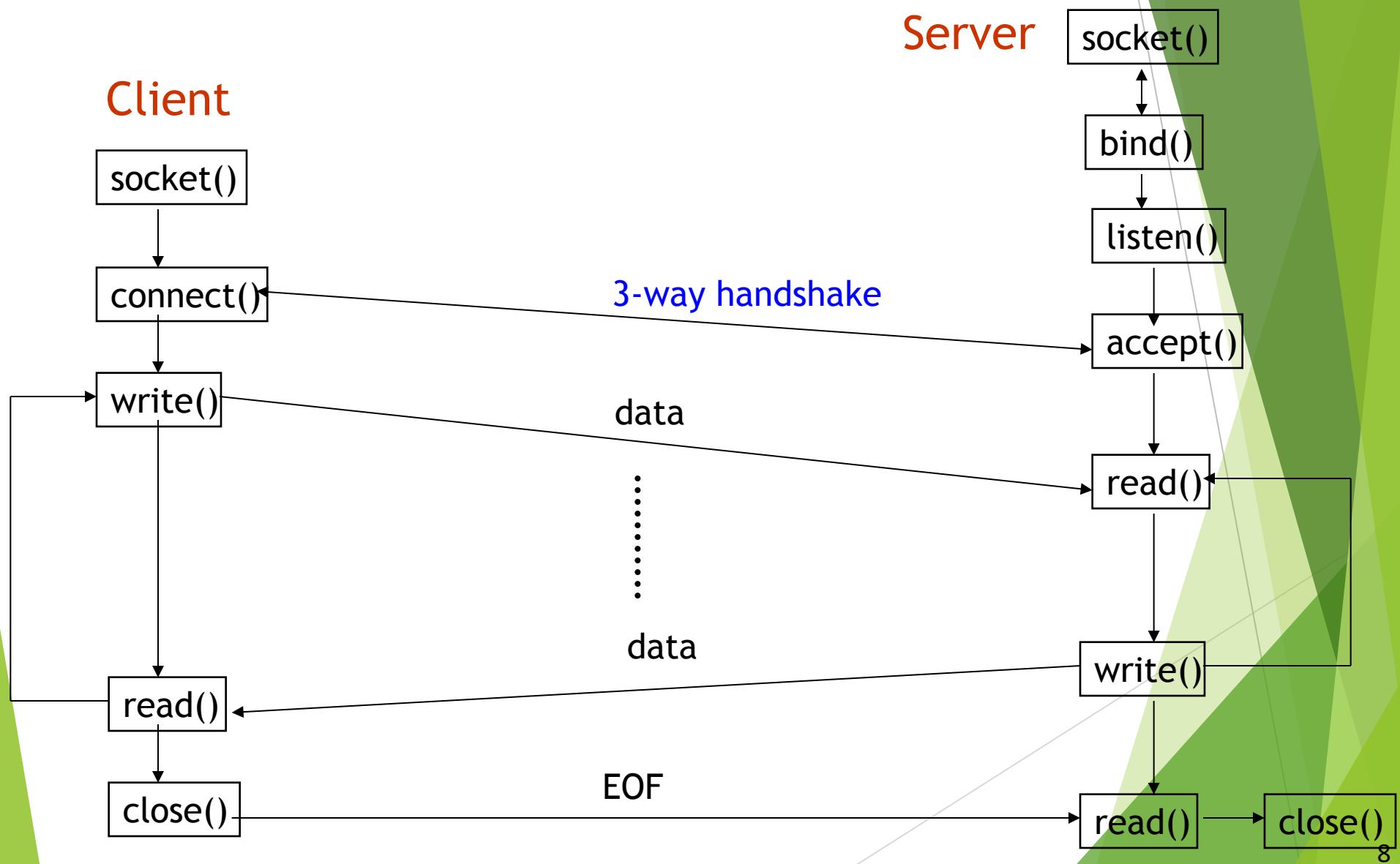
- ▶ There are several significant socket domain families:
 - ▶ Internet Domain Sockets (AF_INET)
 - ▶ implemented via IP addresses and port numbers
 - ▶ Unix Domain Sockets (AF_UNIX)
 - ▶ implemented via filenames (similar to IPC “named pipe”)

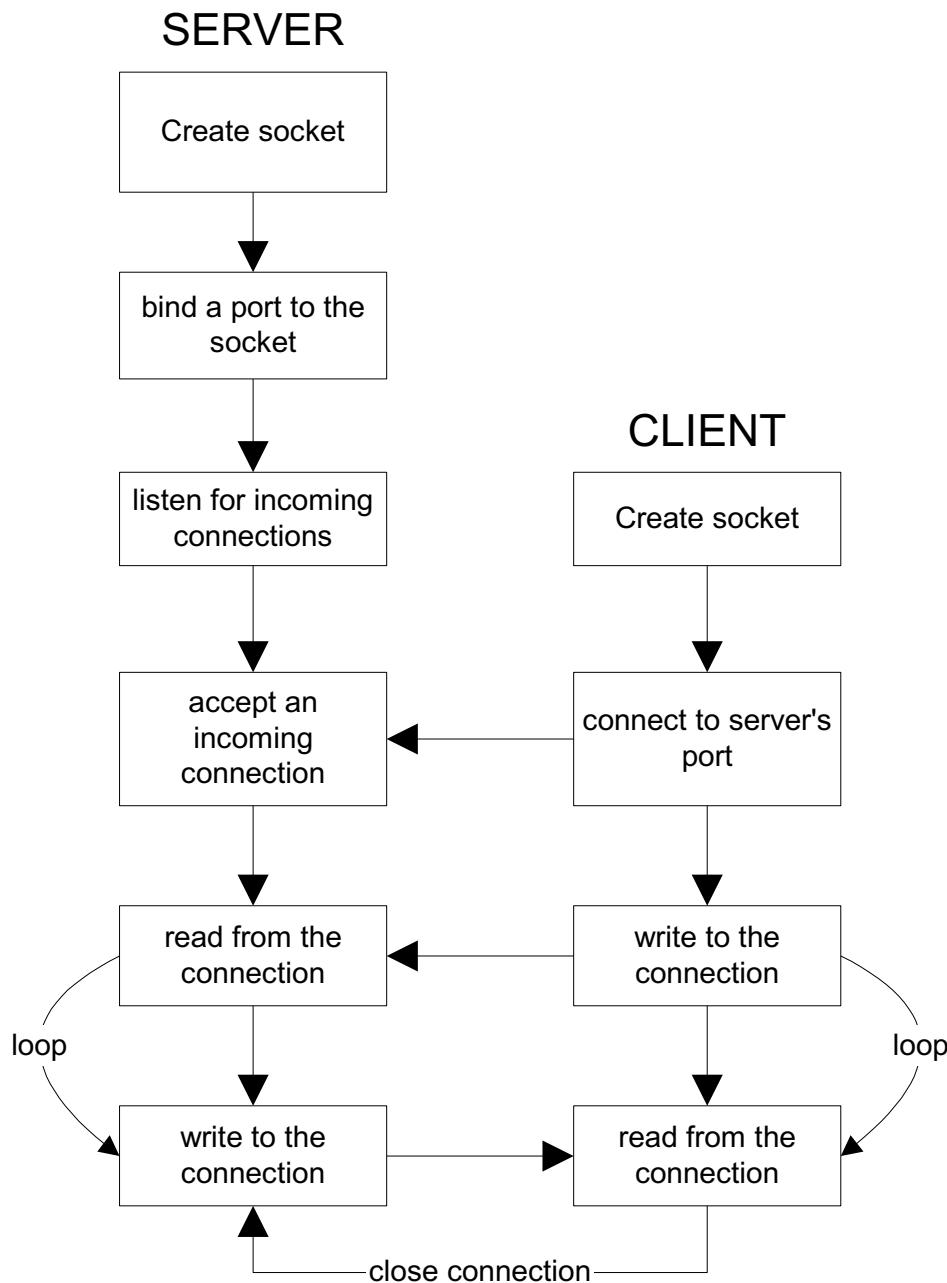
Creating a Socket

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```

- ▶ **domain** is one of the *Protocol Families* (AF_INET, AF_UNIX, etc.)
- ▶ **type** defines the communication protocol semantics, usually defines either:
 - ▶ SOCK_STREAM: connection-oriented stream (TCP)
 - ▶ SOCK_DGRAM: connectionless, unreliable (UDP)
- ▶ **protocol** specifies a particular protocol, just set this to 0 to accept the default

Stream Socket Transaction (TCP Connection)

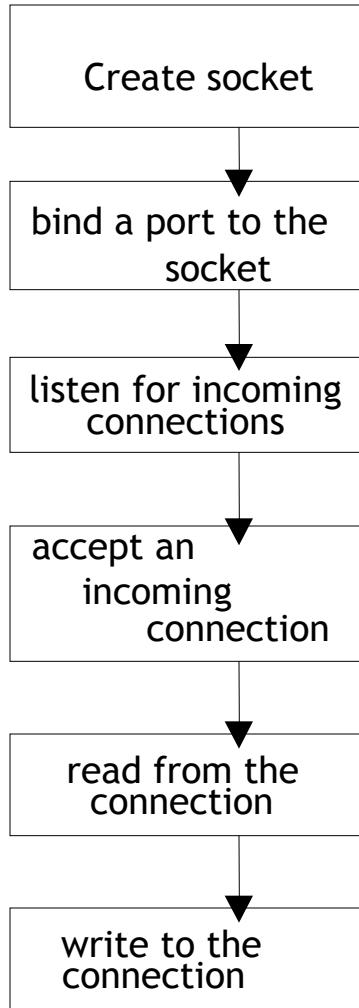




- Connection-oriented socket connections
- Client-Server view

Server-Side Socket Details

SERVER



```
int socket(int domain, int type, int protocol)  
sockfd = socket(AP_INET, SOCK_STREAM, 0);
```

```
int bind(int sockfd, struct sockaddr *server_addr, socklen_t length)  
bind(sockfd, &server, sizeof(server));
```

```
int listen( int sockfd, int num_queued_requests)  
listen( sockfd, 5);
```

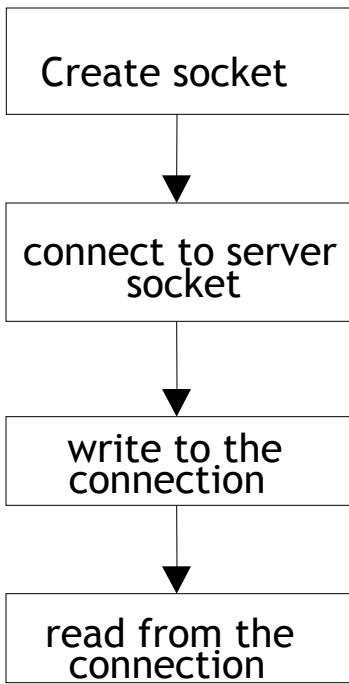
```
int accept(int sockfd, struct sockaddr *incoming_address, socklen_t length)  
newfd = accept(sockfd, &client, sizeof(client)); /* BLOCKS */
```

```
int read(int sockfd, void * buffer, size_t buffer_size)  
read(newfd, buffer, sizeof(buffer));
```

```
int write(int sockfd, void * buffer, size_t buffer_size)  
write(newfd, buffer, sizeof(buffer));
```

Client-Side Socket Details

CLIENT



```
int socket(int domain, int type, int protocol)  
sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

```
int connect(int sockfd, struct sockaddr *server_address, socklen_t len)  
connect(sockfd, &server, sizeof(server)); /* Blocking */
```

```
int write(int sockfd, void * buffer, size_t buffer_size)  
write(sockfd, buffer, sizeof(buffer));
```

```
int read(int sockfd, void * buffer, size_t buffer_size)  
read(sockfd, buffer, sizeof(buffer));
```

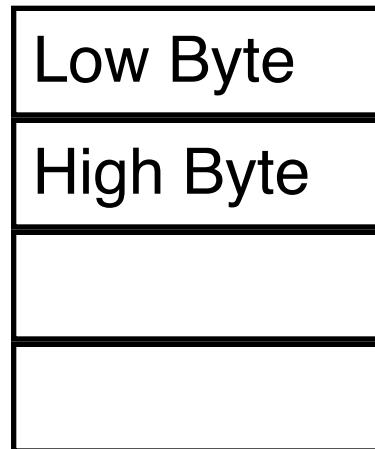
Closing a Socket Session

- ▶ `int close(int socket);`
 - ▶ closes read/write IO, closes socket file descriptor
- ▶ `int shutdown(int socketfd, int mode);`
 - ▶ where mode is:
 - ▶ 0: no more receives allowed “r”
 - ▶ 1: no more sends are allowed “w”
 - ▶ 2: disables both receives and sends (but doesn’t close the socket, use close() for that) “rw”

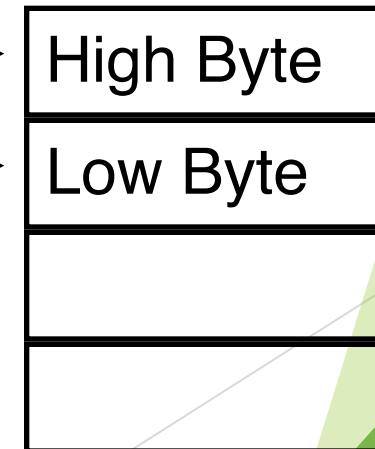
Byte Ordering

- ▶ Different computer architectures use different byte ordering to represent/store multi-byte values (such as 16-bit/32-bit integers)
- ▶ 16 bit integer:

Little-Endian (Intel)



Big-Endian (RISC-Sparc)



Byte Order and Networking

- ▶ Suppose a Big Endian machine sends a 16 bit integer with the value 2:

```
0000000000000010
```

- ▶ A Little Endian machine will understand the number as 512:

```
0000001000000000
```

- ▶ How do two machines with different byte-orders communicate?
 - ▶ Using **network byte-order**
 - ▶ Network byte-order = big-endian order
- ▶ Host byte order
 - ▶ Big Endian (IBM mainframes, Sun SPARC) or
 - ▶ Little Endian (x86)

Network Byte Order

- ▶ Conversion of application-level data is left up to the presentation layer.
- ▶ Lower-level layers communicate using a fixed byte order called *network byte order* for all control data.
- ▶ TCP/IP mandates that *big-endian* byte ordering be used for transmitting protocol information
- ▶ All values stored in a `sockaddr_in` must be in network byte order.
 - ▶ `sin_port` a TCP/IP port number.
 - ▶ `sin_addr` an IP address.

Network Byte Order Functions

- ▶ Several functions are provided to allow conversion between host and network byte ordering,
- ▶ Conversion macros (`<netinet/in.h>`)
 - ▶ to translate 32-bit numbers (i.e. IP addresses):
 - ▶ `unsigned long htonl(unsigned long hostlong);`
 - ▶ `unsigned long ntohl(unsigned long netlong);`
 - ▶ to translate 16-bit numbers (i.e. Port numbers):
 - ▶ `unsigned short htons(unsigned short hostshort);`
 - ▶ `unsigned short ntohs(unsigned short netshort);`

Creating a TCP socket

```
int socket(int family,int type,int proto);  
  
int mysockfd;  
mysockfd = socket (AF_INET, SOCK_STREAM, 0);  
if (mysockfd<0) {  
    /* ERROR */  
}
```

Binding to well known address

```
int mysockfd;  
  
int err;  
  
struct sockaddr_in myaddr;  
  
  
mysockfd = socket(AF_INET, SOCK_STREAM, 0);  
myaddr.sin_family = AF_INET;  
myaddr.sin_port = htons( 80 );  
myaddr.sin_addr = htonl( INADDR_ANY );  
  
  
err= bind(mysockfd, (sockaddr *) &myaddr,  
          sizeof(myaddr));
```

Converting Between IP Address formats

► From ASCII to numeric

- ▶ “130.245.1.44” → 32-bit network byte ordered value
- ▶ `inet_aton(...)` with IPv4
- ▶ `inet_pton(...)` with IPv4 and IPv6

► From numeric to ASCII

- ▶ 32-bit value → “130.245.1.44”
- ▶ `inet_ntoa(...)` with IPv4
- ▶ `inet_ntop(...)` with IPv4 and IPv6
- ▶ **Note** - `inet_addr(...)` obsolete
 - ▶ cannot handle broadcast address “255.255.255.255” (0xFFFFFFFF)

IPv4 Address Conversion

```
int inet_aton( char *, struct in_addr *);
```

Convert ASCII dotted-decimal IP address to network byte order 32 bit value. Returns 1 on success, 0 on failure.

```
char *inet_ntoa(struct in_addr);
```

Convert network byte ordered value to ASCII dotted-decimal (a string).

Establishing a passive mode TCP socket

► Passive mode:

- ▶ Address already determined.
- ▶ Tell the kernel to accept incoming connection requests directed at the socket address.
 - ▶ *3-way handshake*
- ▶ Tell the kernel to queue incoming connections for us.

listen()

```
int listen( int mysockfd, int backlog);
```

mysockfd **is** the TCP socket (already bound to an address)

backlog **is** the number of incoming connections the kernel should be able to keep track of (queue for us).

listen() returns -1 on error (otherwise 0).

Accepting an incoming connection

- ▶ Once we call `listen()`, the O.S. will queue incoming connections
 - ▶ Handles the 3-way handshake
 - ▶ Queues up multiple connections.
- ▶ When our application is ready to handle a new connection, we need to ask the O.S. for the next connection.

accept()

```
int accept( int mysockfd,  
            struct sockaddr* cliaddr,  
            socklen_t *addrlen);
```

mysockfd **is the passive mode TCP socket.**

cliaddr **is a pointer to *allocated* space.**

addrlen **is a *value-result* argument**

- ▶ must be set to the size of cliaddr
- ▶ on return, will be set to be the number of used bytes in cliaddr.

- ▶ accept() return value
 - ▶ accept() returns a new socket descriptor (positive integer) or -1 on error.
 - ▶ After accept returns a new socket descriptor, I/O can be done using the read() and write() system calls.

Terminating a TCP connection

- ▶ Either end of the connection can call the `close()` system call.
- ▶ If the other end has closed the connection, and there is no buffered data, reading from a TCP socket returns 0 to indicate EOF.

Client Code

- ▶ TCP clients can call `connect()` which:
 - ▶ takes care of establishing an endpoint address for the client socket.
 - ▶ don't need to call `bind` first, the O.S. will take care of assigning the local endpoint address (TCP port number, IP address).
 - ▶ Attempts to establish a connection to the specified server.
 - ▶ *3-way handshake*

connect()

```
int connect( int sockfd,  
             const struct sockaddr *server,  
             socklen_t addrlen);
```

sockfd is an already created TCP socket.

server contains the address of the server (IP Address and TCP port number)

connect() returns 0 if OK, -1 on error

Reading from a TCP socket

```
int read( int fd, char *buf, int max);
```

- ▶ By default `read()` will block until data is available.
- ▶ reading from a TCP socket may return less than max bytes (whatever is available).

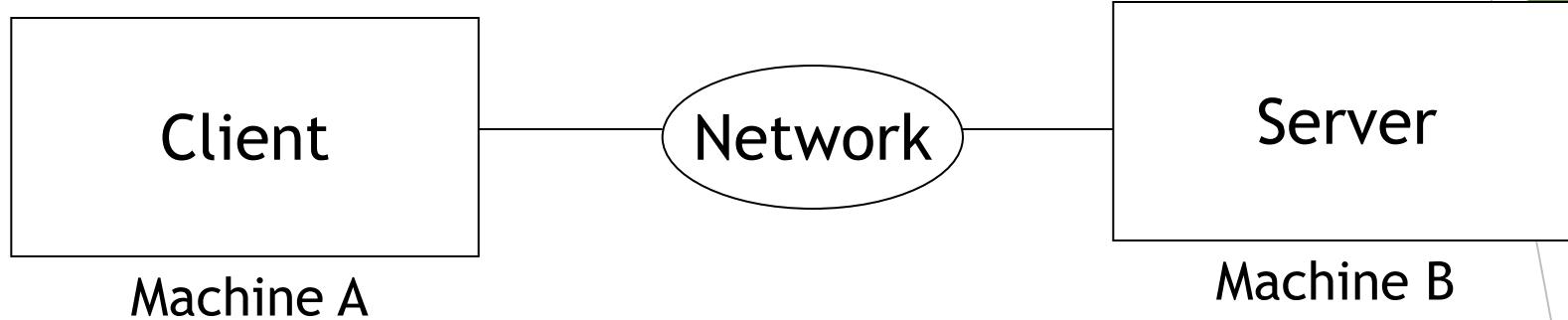
Writing to a TCP socket

```
int write( int fd, char *buf, int num);
```

- ▶ write might not be able to write all num bytes (on a nonblocking socket).
- ▶ Other functions (API)
 - ▶ readn(), writen() and readline() - see man pages definitions.

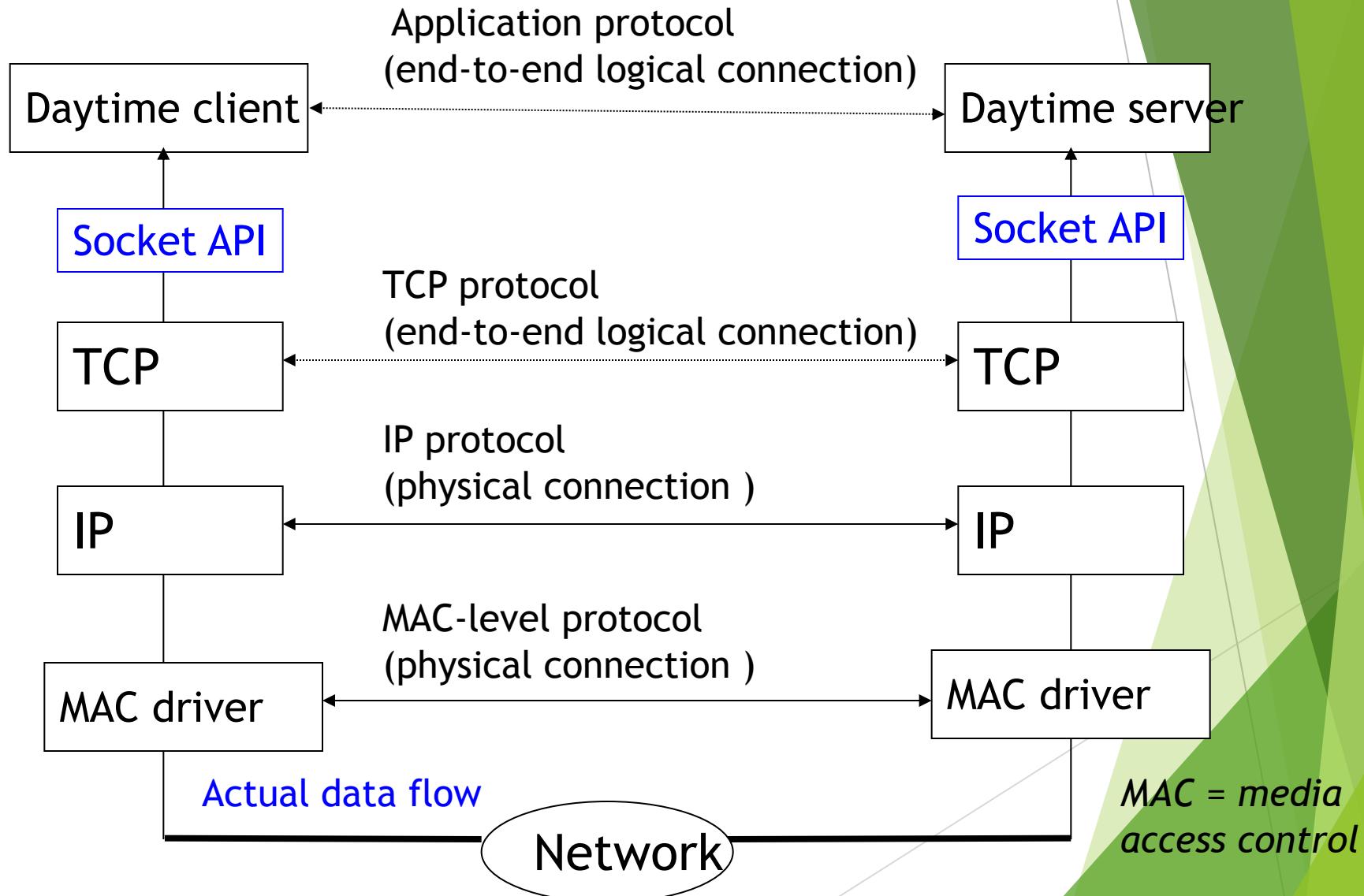
Example

Client Server communication



- Web browser and server
- FTP client and server
- Telnet client and server

Example - Daytime Server/Client



Daytime client

```
#include      "unp.h"
int main(int argc, char **argv)
{
    int sockfd, n;
    char recvline[MAXLINE + 1];
    struct sockaddr_in servaddr;

    if( argc != 2 )err_quit("usage : gettime <IP address>");

    /* Create a TCP socket */
    if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        err_sys("socket error");

    /* Specify server's IP address and port */
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(13); /* daytime server port */
    if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0)
        err_quit("inet_pton error for %s", argv[1]);
```

- Connects to a daytime server
 - Retrieves the current date and time
- % gettime 130.245.1.44
Thu Sept 05 15:50:00 2002

Daytime client

```
/* Connect to the server */

if (connect(sockfd, (SA *) &servaddr, sizeof(servaddr)) < 0)
    err_sys("connect error");

/* Read the date/time from socket */

while ( (n = read(sockfd, recvline, MAXLINE)) > 0) {
    recvline[n] = 0;           /* null terminate */
    printf("%s", recvline);

}

if (n < 0) err_sys("read error");

close(sockfd);

}
```

Daytime Server

```
#include      "unp.h"
#include      <time.h>

int main(int argc, char **argv)
{
    int      listenfd, connfd;
    struct sockaddr_in servaddr;
    char    buff[MAXLINE];
    time_t   ticks;

    /* Create a TCP socket */
    listenfd = socket(AF_INET, SOCK_STREAM, 0);

    /* Initialize server's address and well-known port */
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family      = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port        = htons(13); /* daytime server */

    /* Bind server's address and port to the socket */
    Bind(listenfd, (SA *) &servaddr, sizeof(servaddr));
```

- **Waits for requests from Client**
- **Accepts client connections**
- **Send the current time**
- **Terminates connection and goes back waiting for more connections.**

Daytime Server

```
/* Convert socket to a listening socket */
listen(listenfd, LISTENQ);

for ( ; ; ) {
    /* Wait for client connections and accept them */
    connfd = accept(listenfd, (SA *) NULL, NULL);

    /* Retrieve system time */
    ticks = time(NULL);
    sprintf(buff, sizeof(buff), "%.24s\r\n", ctime(&ticks));

    /* Write to socket */
    write(connfd, buff, strlen(buff));

    /* Close the connection */
    close(connfd);
}
```

HTTP: The Hypertext Transfer Protocol

CEN 223 - Internet Communication

Assoc. Prof. Dr. Fatih ABUT
Department of Computer Engineering
Çukurova University

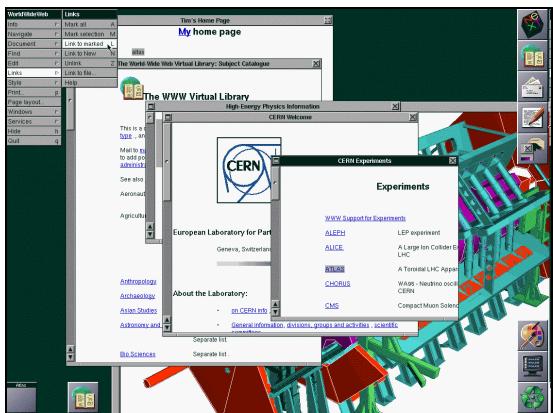
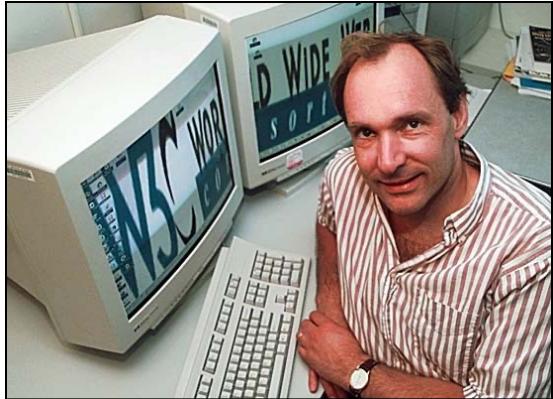
HTTP Overview

- HTTP is based on the TCP protocol.
- The web server uses the TCP port 80 by default.
- HTTP is a stateless request / response protocol.
- HTTP is text oriented
- Typical methods are
 - GET: Retrieval of the document identified in the URL
 - HEAD: Retrieval of meta information about the document identified in the URL
 - ...
- An HTTP server only needs to implement the GET and HEAD methods.
- Design goals of HTTP are:
 - **Simplicity:** The protocol should be easy to implement.
 - **Efficiency:** The protocol should only consume a few resources (We are currently moving away from this.)
 - **Performance:** The protocol should be as fast as possible.

Design Criteria:

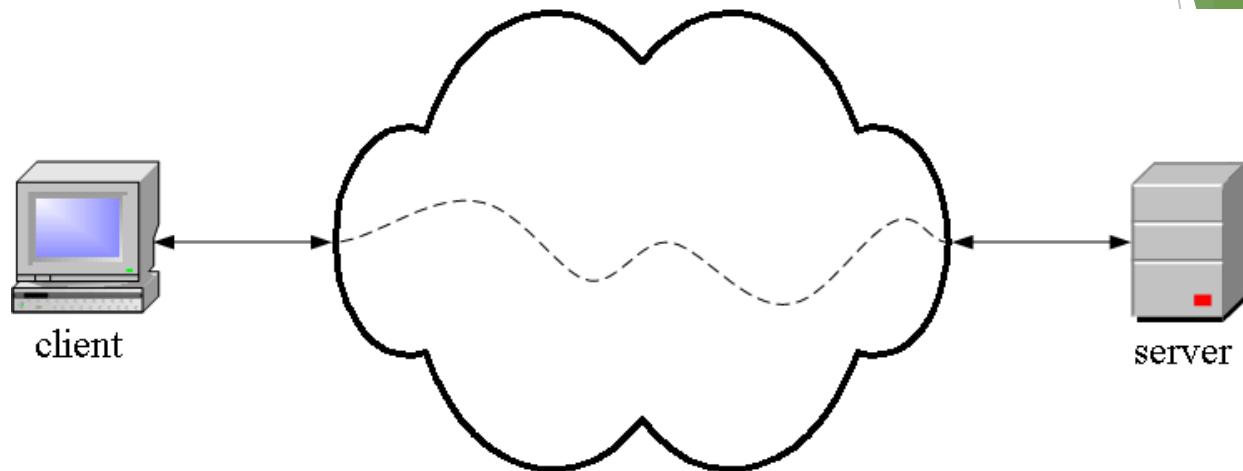
- Simplicity
- Efficiency
- Performance

Tim Berners-Lee



Tim Berners-Lee was knighted by Queen Elizabeth for his invention of the World Wide Web. He is shown here, along with the first picture posted on the Web and a screen shot from an early version of his Web browser.

HTTP is an application layer protocol



- ▶ The Web client and the Web server are application programs
- ▶ Application layer programs do useful work like retrieving Web pages, sending and receiving email or transferring files
- ▶ Lower layers take care of the communication details
- ▶ The client and server send messages and data without knowing anything about the communication network

The application layer is boss - the top layer

Layer	Function
Application	Do useful work like Web browsing, email, and file transfer
Lower layers	Handle communication between the client and server

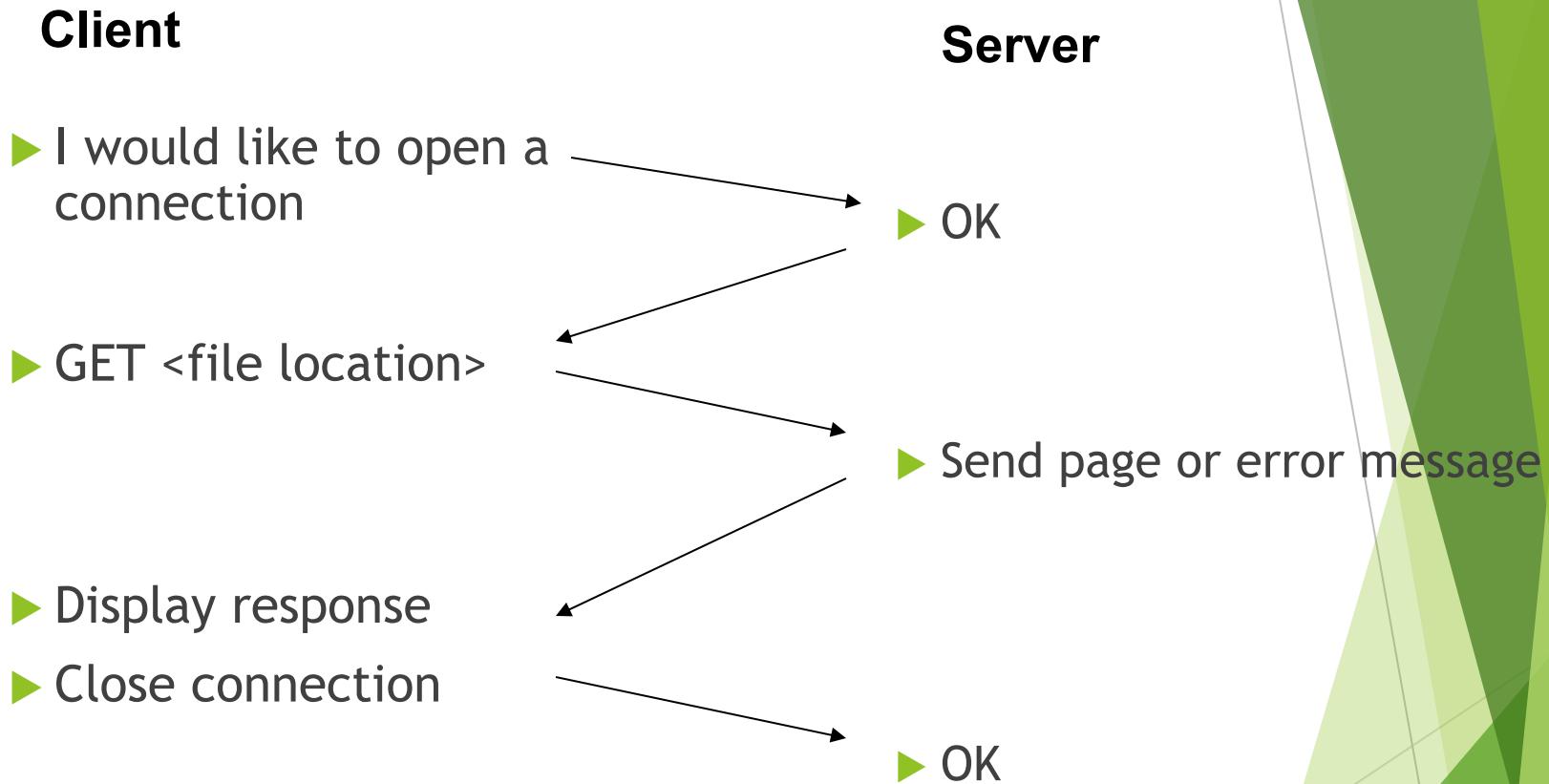
► **Your boss says:** Send this package to Miami -- I don't care if you use Federal Express, UPS, or any other means. Also, let me know when it arrives or if it cannot be delivered for some reason.

► **The application program says:** Send this request to the server -- I don't care how you do it or whether it goes over phone lines, radio, or anything else about the details. Just send the message, and let me know when it arrives or if it cannot be delivered for some reason.

Many application layer protocols are used on the Internet, HTTP is only one

Protocol	Application
HTTP: Hypertext Transfer	Retrieve and view Web pages
FTP: File Transfer	Copy files from client to server or from server to client
SMTP: Simple Mail Transport	Send email
POP: Post Office	Read email

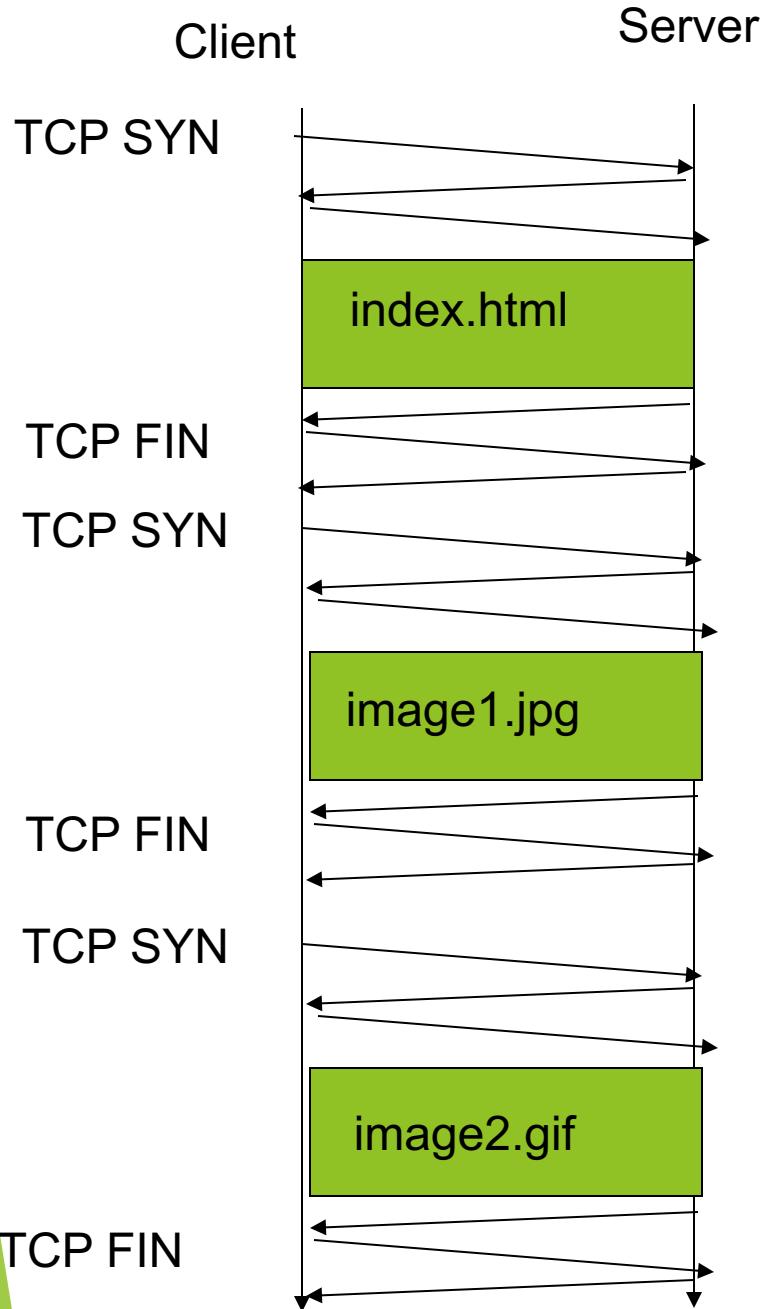
An HTTP conversation



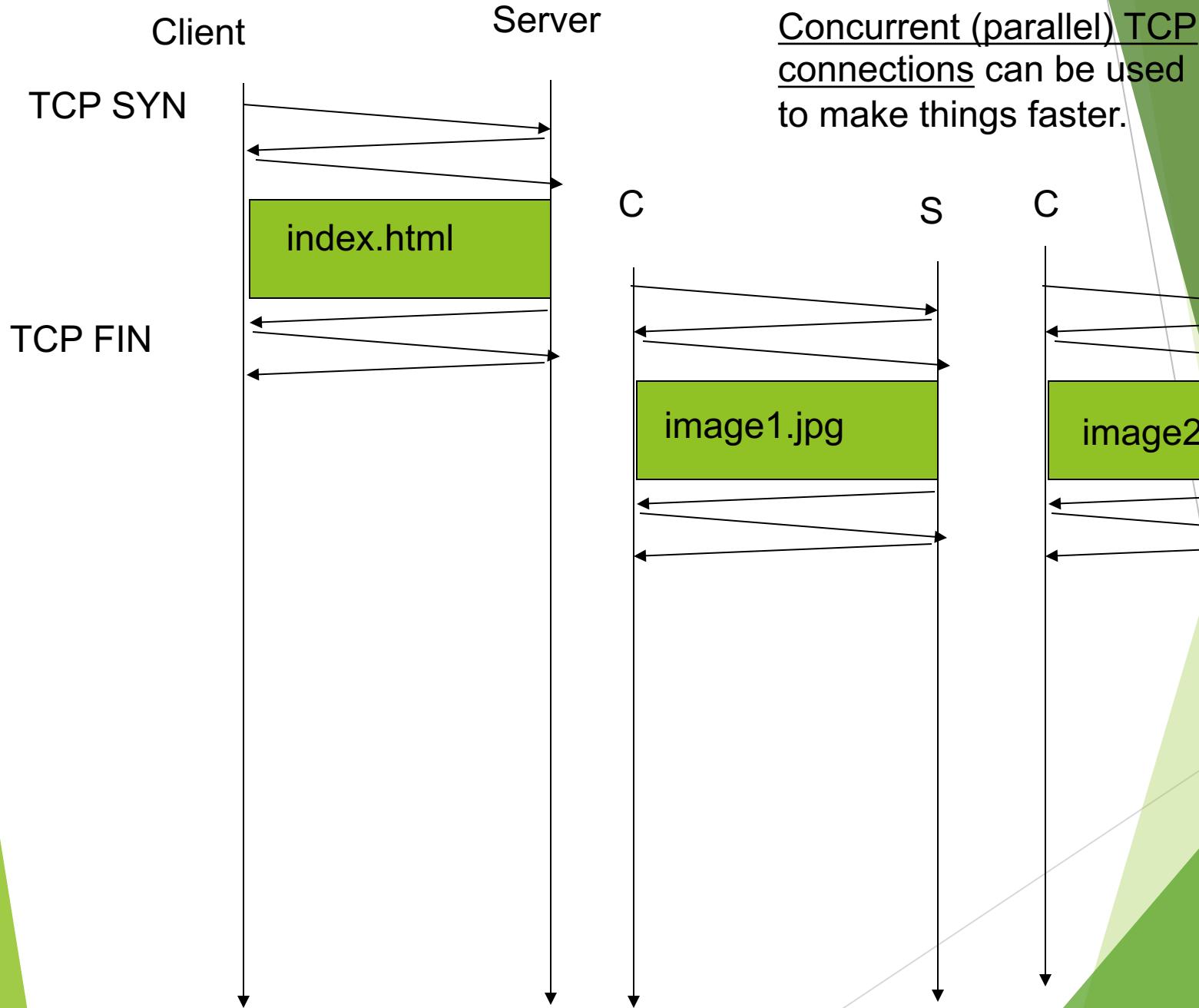
HTTP is the set of rules governing the format and content of the conversation between a Web client and server

Possible Communication Models for HTTP

1. The “classic” approach: HTTP/1.0
2. HTTP over multiple, concurrent TCP connections
3. Persistent-HTTP: HTTP/1.1
4. HTTP Pipelining
5. Transaction-TCP (T/TCP)
6. HTTP over UDP-based protocols



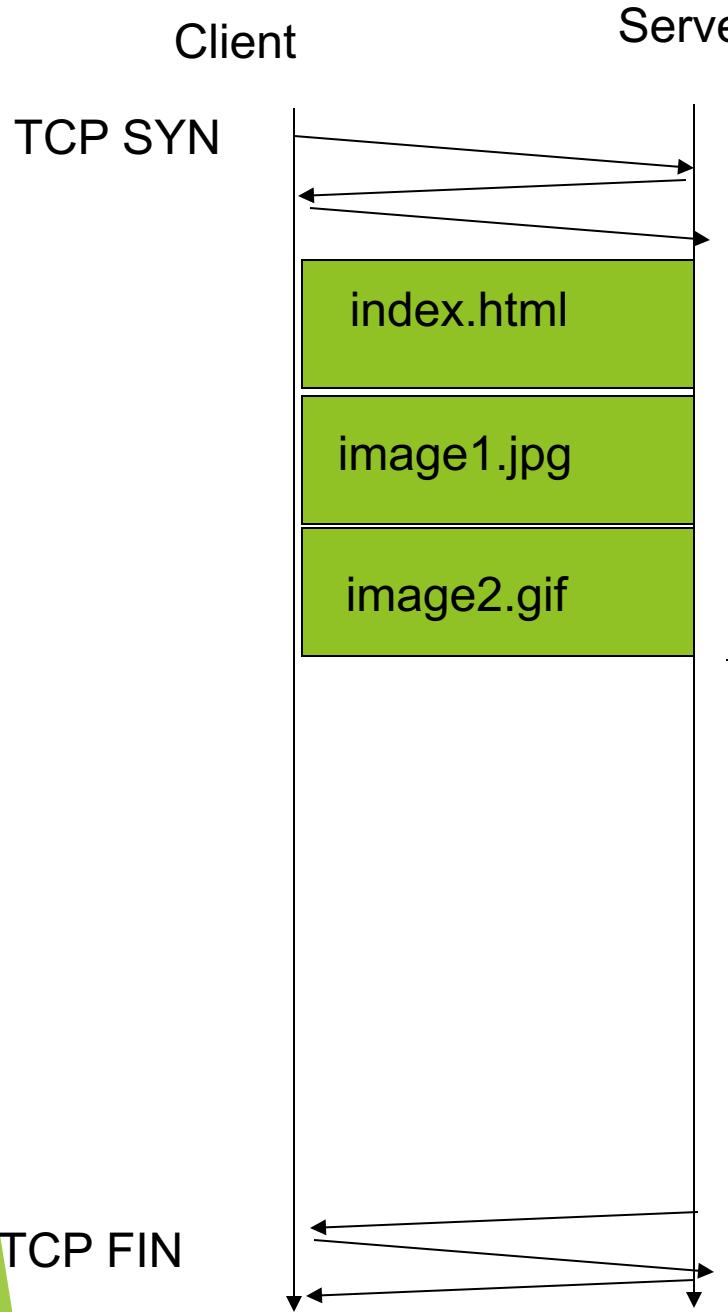
The “classic” approach in HTTP/1.0 is to use one HTTP request per TCP connection, serially.



Concurrent (parallel) TCP connections can be used to make things faster.



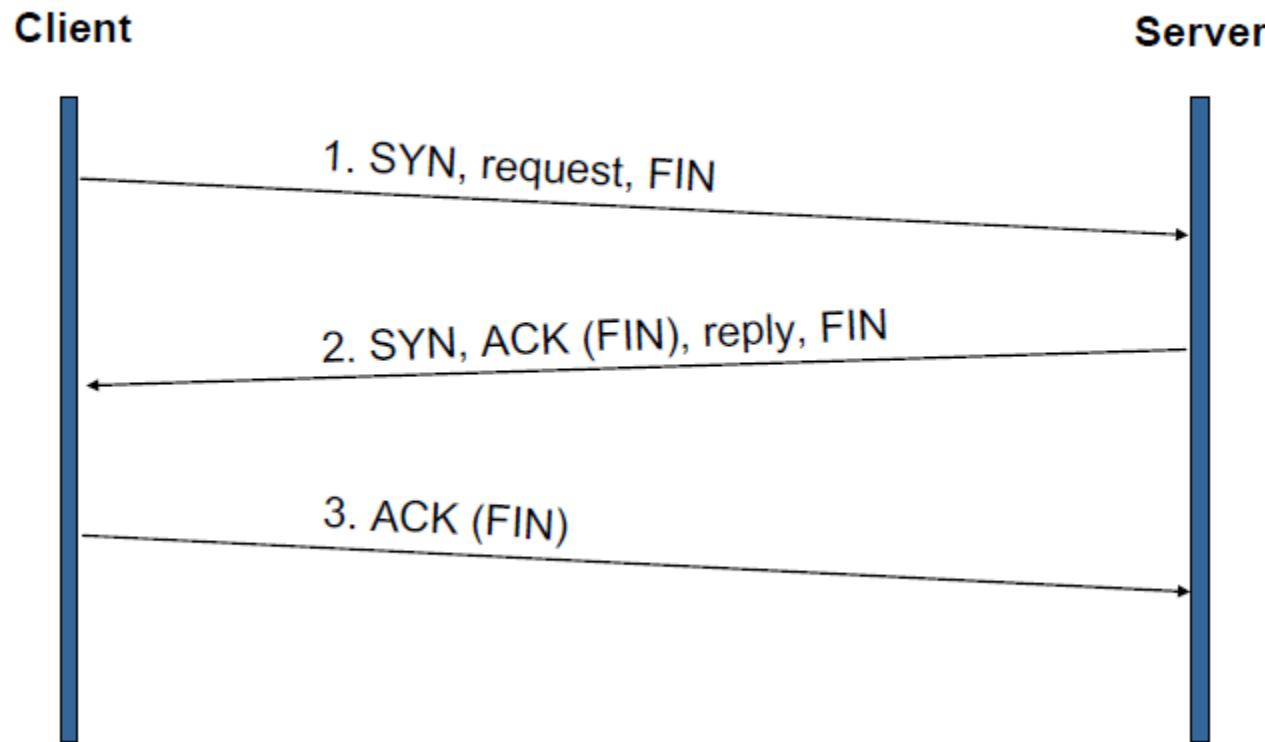
The “persistent HTTP” approach can re-use the same TCP connection for multiple HTTP transfers, one after another, serially. Amortizes TCP overhead, but maintains TCP state longer at server.



The “pipelining” feature in HTTP/1.1 allows requests to be issued asynchronously on a persistent connection. Requests must be processed in proper order. Can do clever packaging.

Problem: Head of line blocking

Transaction-TCP



Performance Effects of HTTP/1.1

	HTTP/1.0	HTTP/1.1 Persistent	HTTP/1.1 Pipeline
Max simultaneous sockets	6	1	1
Total number of sockets used	40	1	1
Packets from client to server	226	70	25
Packets from server to client	271	153	58
Total number of packets	497	223	83
Total elapsed time [secs]	1.85	4.13	3.02

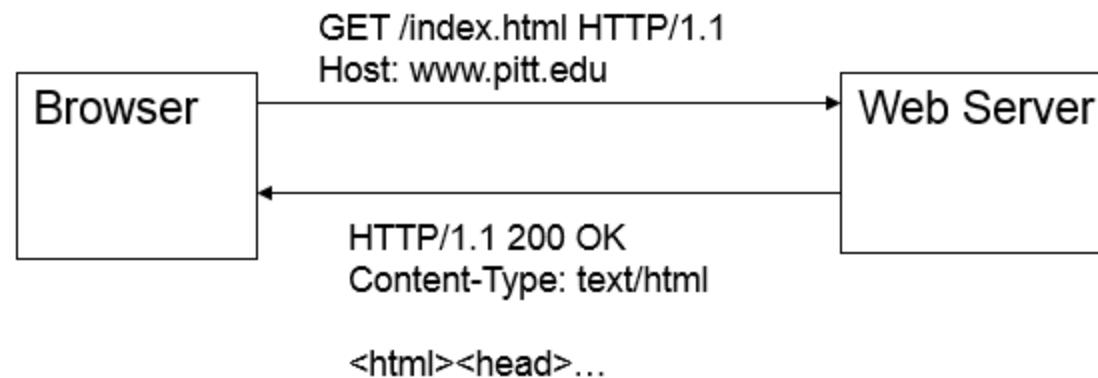
Nielsen, Hendrik Frystyk, Gettys, Jim, Baird-Smith, Anselm, Prud'hommeaux, Eric, Håkon Wium Lie, Lilley, Chris: Network Performance Effects of HTTP/1.1, CSS1, and PNG, W3C, NOTE 24-June 1997.

HTTP over UDP

- What is the motivation?
- What are the advantages?
- What are the disadvantages?
- What are the consequences?

HTTP Messages (1)

- ▶ HEAD
- ▶ GET
- ▶ POST
- ▶ PUT
- ▶ DELETE
- ▶ TRACE
- ▶ OPTIONS
- ▶ CONNECT
- ▶ PATCH



HTTP methods (2)

Method	Description
GET	<p>Request for a resource. The resource is specified in the URL. GET is supported by all browsers.</p> <p>Example: GET /index.html</p>
POST	<p>Used to send data to the server. Main application online forms. The format of the data is described in the Content-Type: attribute. The data is transferred in the body.</p> <p>Example: Content-Type: application/x-www-form-urlencoded param1=value1&param2=value2</p>
HEAD	<p>Returns only the headers that would have been used in the case of a GET request. The client can then decide how to proceed.</p> <p>Example: HEAD /downloads/video.mpeg HTTP/1.1</p>

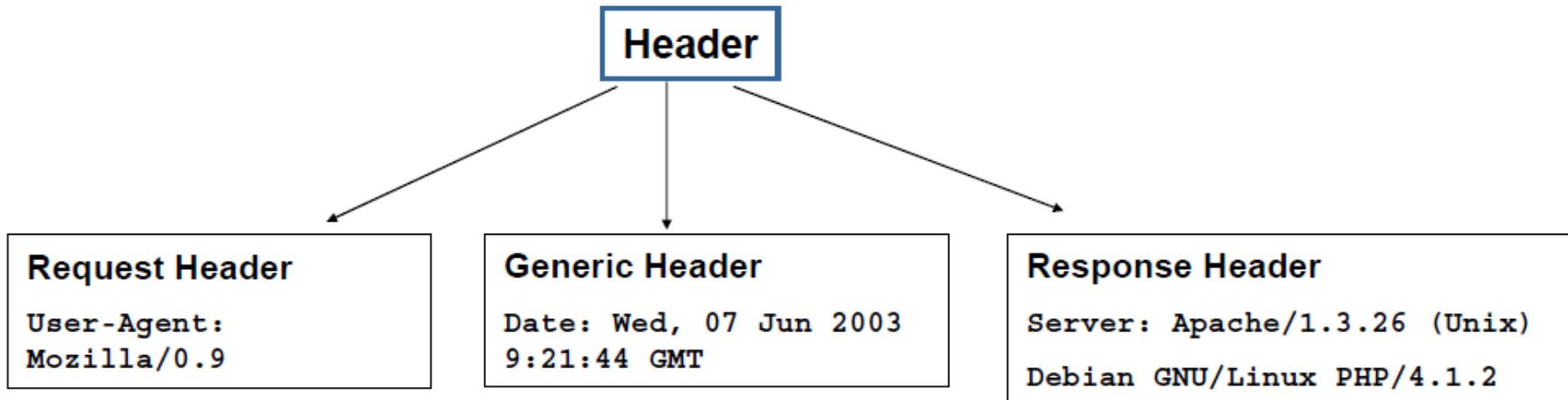
HTTP methods (3)

Method	Description
PUT	Create a new resource or replace an existing one. The data is transferred in the body. Successful answer (code 200 or 204) does not contain a body. The creation of a new resource is indicated by the code 201 (Created). Example: PUT /new.html HTTP/1.1
OPTIONS	Asks the server for the permitted HTTP methods. A special URL or the asterisk * can be specified for the entire server. The options can be found in the Response header Allow. The OPTIONS method is often deactivated. Example: OPTIONS http://bmb.cu.edu.tr HTTP/1.1
DELETE	Deletes the specified resource. Used for API servers. Is switched off on content-oriented servers.
TRACE	Loop back test. The request is sent back as it was received.

HTTP special methods

Method	Description
PATCH	<p>While the PUT method completely replaces a resource, a resource can be specifically modified with PATCH. The modifications specified in the body must be applied to the resource entirely or not at all ("atomic"). The modifications can, for example, be specified with the correct syntax in the body using a JSON PATCH (RFC6902).</p> <p>The body of the response contains a representation of the changed resource.</p>
CONNECT	<p>CONNECT establishes a tunnel to the requested resource. After that, data can be exchanged "blindly" via the tunnel until the tunnel is closed. CONNECT should only be used for requests to a proxy.</p>

HTTP Messages (2)



HTTP Messages (3)

Informational (1xx): 100 Continue

Successfull (2xx): 200 OK

Redirection (3xx) 302 Found

Client Error (4xx) 400 Bad Request

Server Error (5xx) 500 Internal Server Error

HTTP Response Status Codes

There are a variety of status codes. The most important are:

- **HTTP Status Code 200 – OK**
- **HTTP Status Code 301 - Moved Permanently:**

Resource has a different URL. This can be found in the location header.

- **HTTP Status Code 302 – Moved Temporarily**
- **HTTP Status Code 403 – Forbidden**
- **HTTP Status Code 404 - Not Found:**

Forwarding to an error page or to a most relevant page. The redirect to the homepage is bad, as it seems confusing.

- **HTTP Status Code 500 – Internal Server Error:**
Server crashed.
- **HTTP Status Code 503 - Service Unavailable:**

Server is overloaded. It is helpful here to provide information about when the server is likely to be available again.

HTTP Messages - Example

```
ousystem@ux-2s02 /mnt/www/ousystem 5# telnet www2 80
Trying 194.95.66.20...
Connected to ux-2s05.inf.fh-rhein-sieg.de.
Escape character is '^]'.
GET / HTTP/1.0
```

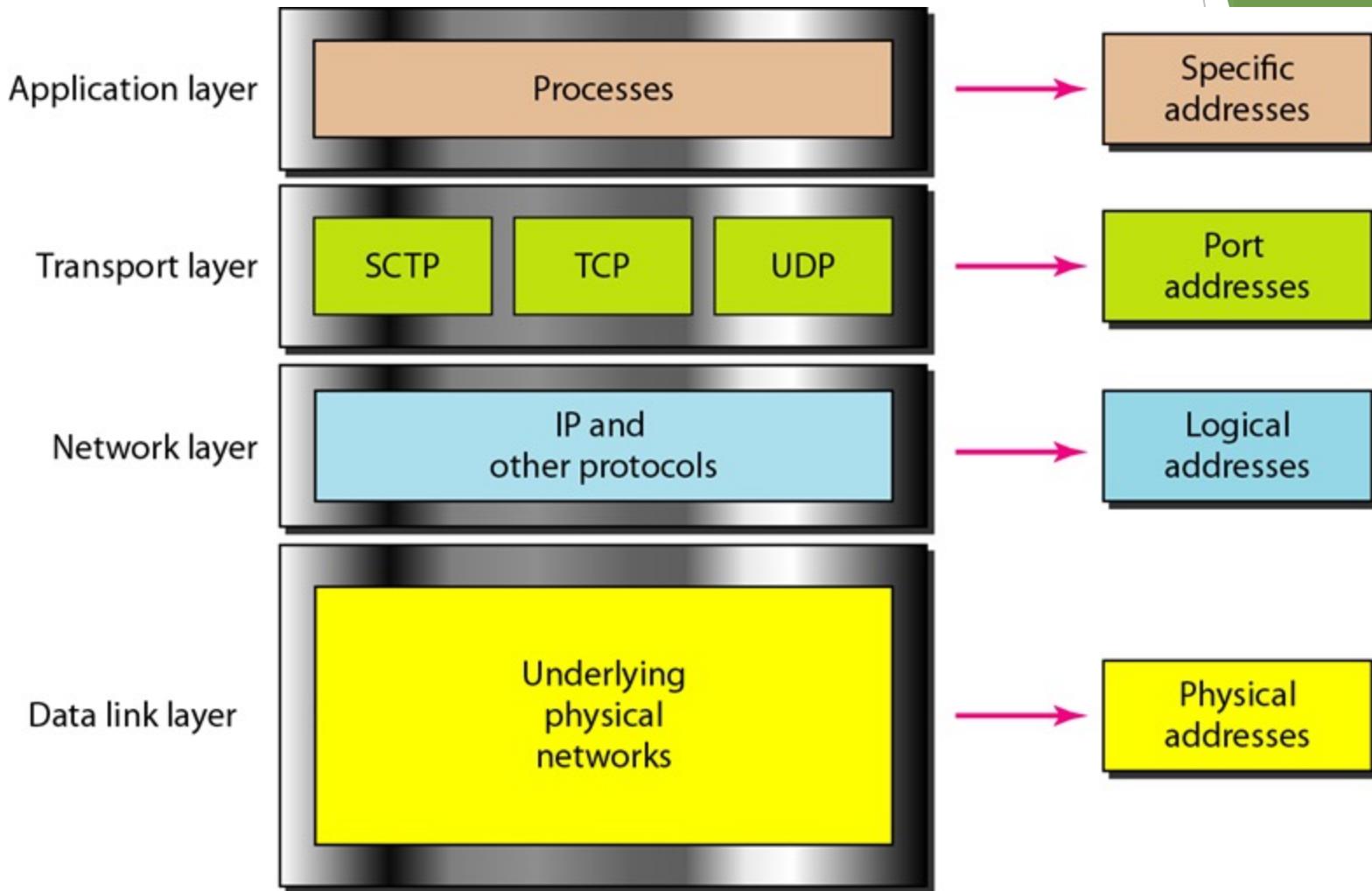
HTTP/1.1 200 OK

Date: Fri, 05 Dec 2003 10:09:52 GMT
Server: Apache/1.3.26 (Unix) Debian GNU/Linux PHP/4.1.2 mod_ssl/2.8.9 OpenSSL/0.9.6g
Last-Modified: Wed, 27 Feb 2002 15:12:43 GMT
ETag: "b42d-45f-3c7cf76b"
Accept-Ranges: bytes
Content-Length: 1119
Connection: close
Content-Type: text/html; charset=iso-8859-1

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<HTML>
<HEAD>
```

Relationship of layers and addresses in TCP/IP

HTTP Addressing?



Identification of resources

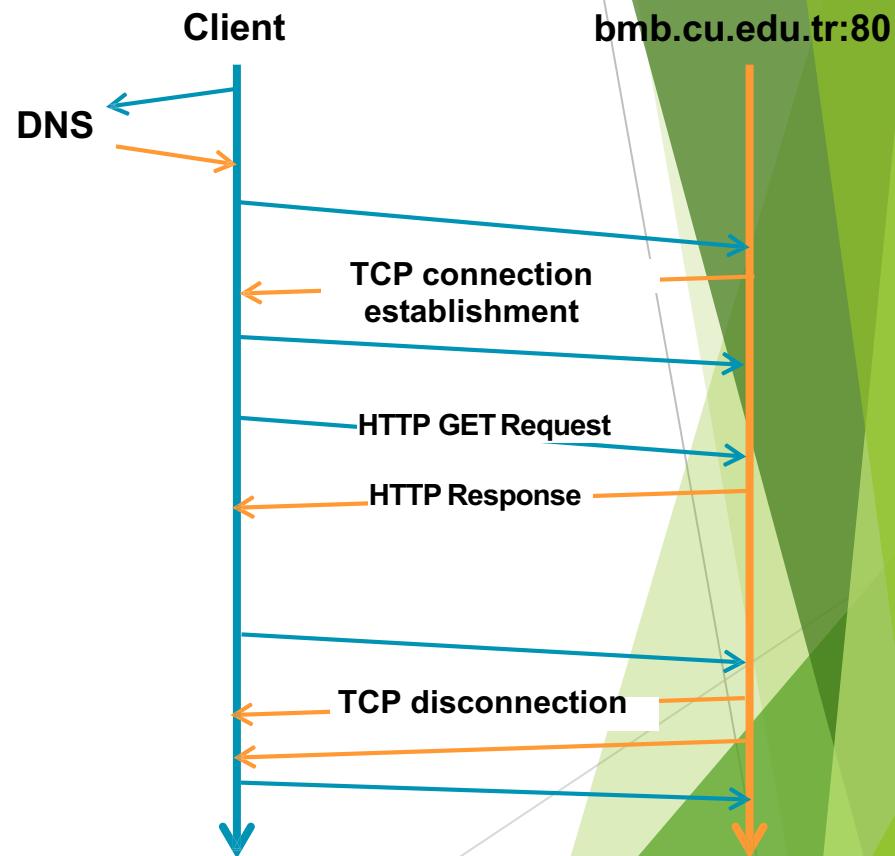
`http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument`

Component	Name	Other examples, remark
http	Protocol	https, mailto, ftp, file
www.example.com	Authority	192.34.23.8
80	Port	Can be omitted or other port numbers
path/to/myfile.html	Path	Path to the resource
key1=value1&key2=value2	Query	Syntax is not fixed, server specific
SomewhereInTheDocument	Fragment	Pointer for referring to a particular fragment in the resource.

Basic sequence of an HTTP request response

Example: Access to <http://bmb.cu.edu.tr:80/>

1. DNS resolution
2. Establishing a TCP connection to 193.140.54.17:80
3. Send HTTP GET request messages as bytestreams to 193.140.54.17:80.
4. Send HTTP response messages as bytestreams to the client.
5. Termination of the TCP connection by the client or server
6. Termination of the TCP connection by the server or client



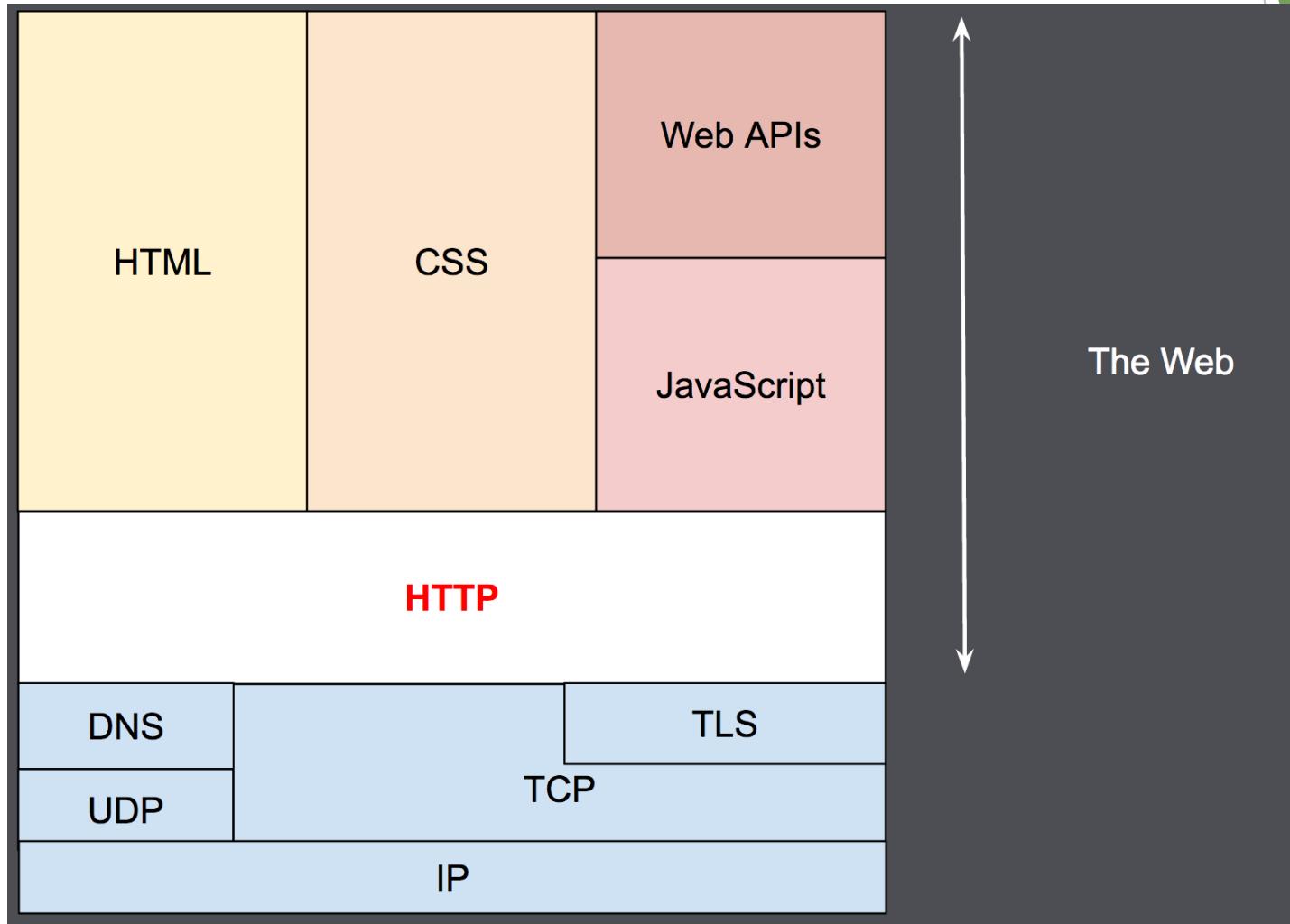
HTTP History

- ▶ 1991 - HTTP 0.9 defined, supports only hypertext, linked to port 80 (no images)
 - ▶ Tim Berners-Lee starts World Wide Web project at CERN
 - ▶ Goal: Simplicity, just get an ASCII page from the server
- ▶ 1992 - HTTP 1.0, supports images, scripts as well
 - ▶ Goal: Transfer of any file format between client and server
- ▶ 1994 - CERN and MIT agree to set up WWW Consortium
- ▶ 1997 - HTTP 1.1, supports open ended extensions
 - ▶ Goal: Performance optimization, caching, pipelining.
- ▶ 2000 - Roy Fielding: Doctoral thesis "The REST-API was born (REST = REpresentational State Transfer)
- ▶ 2012 - First HTTP 2.0 based on SPDY (Google) HTTP 2.0. RFC 7540.
- ▶ 2015 - Concept: Performance optimization through multiplexing (parallel requests), stream prioritization, server push, "must" TLS.
- ▶ 2018 - Hypertext Transfer Protocol (HTTP) over QUIC UDP-based HTTP
- ▶ 2019 - HTTP-over-QUIC goes over to HTTP/3.

HTTP vs HTML

- ▶ HTML: hypertext **markup language**
 - ▶ Definitions of tags that are added to Web documents to control their appearance
- ▶ HTTP: hypertext transfer **protocol**
 - ▶ The rules governing the conversation between a Web client and a Web server

Web technologies and OSI



References

IETF: *RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1*,
<https://datatracker.ietf.org/doc/html/rfc2616>, June 1999.

IETF: *RFC 7230 - Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*, <https://datatracker.ietf.org/doc/html/rfc7230>, June 2014.

IETF: *RFC 7230 - Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*,
<https://datatracker.ietf.org/doc/html/rfc7231>, June 2014.

Mozilla and individual contributors: *HTTP*, <https://developer.mozilla.org/en-US/docs/Web/HTTP> (abgerufen am 13.05.2021)

David Gourley, Brian Totty, Marjorie Sayer, Anshu Aggarwal, Sailu Reddy: *HTTP: The Definitive Guide*, O'Reilly Media, Inc., 2002

Domain Name System (DNS) & E-Mail

CEN 223 - Internet Communication

Assoc. Prof. Dr. Fatih ABUT
Department of Computer Engineering
Çukurova University



Domain Name System (DNS) was proposed

in 1983 by Paul Mockapetris

Host Names vs. IP addresses

- ▶ Host names
 - ▶ Mnemonic name appreciated by humans
 - ▶ Variable length, alpha-numeric characters
 - ▶ Provide little (if any) information about location
 - ▶ Examples: www.cnn.com and ftp.eurocom.fr
- ▶ IP addresses
 - ▶ Numerical address appreciated by routers
 - ▶ Fixed length, binary number
 - ▶ Hierarchical, related to host location
 - ▶ Examples: 64.236.16.20 and 193.30.227.161

“What's the IP address of mail.cis.udel.edu?”

“128.4.40.12”

“What's the name of the host at 128.4.40.12?”

“mail.cis.udel.edu”

Separating Naming and Addressing

- ▶ Names are easier to remember
 - ▶ www.cnn.com vs. 64.236.16.20
- ▶ Addresses can change underneath
 - ▶ Move www.cnn.com to 173.15.201.39
 - ▶ E.g., renumbering when changing providers
- ▶ Name could map to multiple IP addresses
 - ▶ www.cnn.com to multiple replicas of the Web site
- ▶ Map to different addresses in different places
 - ▶ Address of a nearby copy of the Web site
 - ▶ E.g., to reduce latency, or return different content
- ▶ Multiple names for the same address
 - ▶ E.g., aliases like ee.mit.edu and cs.mit.edu

Strawman Solution: Central Server

- ▶ Central server
 - ▶ One place where all mappings are stored
 - ▶ All queries go to the central server
- ▶ Many practical problems
 - ▶ Single point of failure
 - ▶ High traffic volume
 - ▶ Distant centralized database
 - ▶ Single point of update
 - ▶ Does not scale

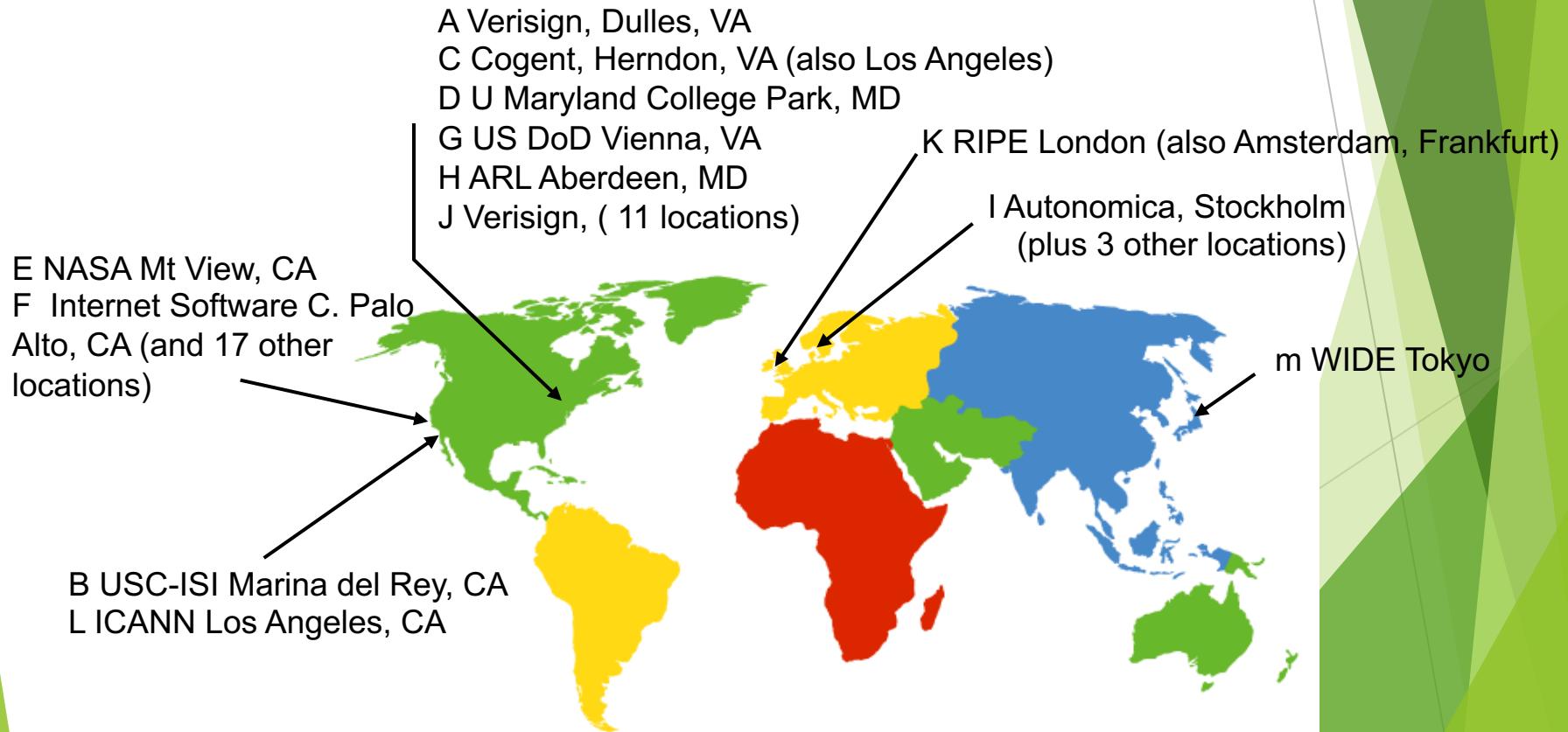
Need a distributed, hierarchical collection of servers

Domain Name System (DNS)

- ▶ Properties of DNS
 - ▶ Hierarchical name space divided into zones
 - ▶ Distributed over a collection of DNS servers
- ▶ Hierarchy of DNS servers
 - ▶ Root servers
 - ▶ Top-level domain (TLD) servers
 - ▶ Authoritative DNS servers
- ▶ Performing the translations
 - ▶ Local DNS servers
 - ▶ Resolver software

DNS Root Servers

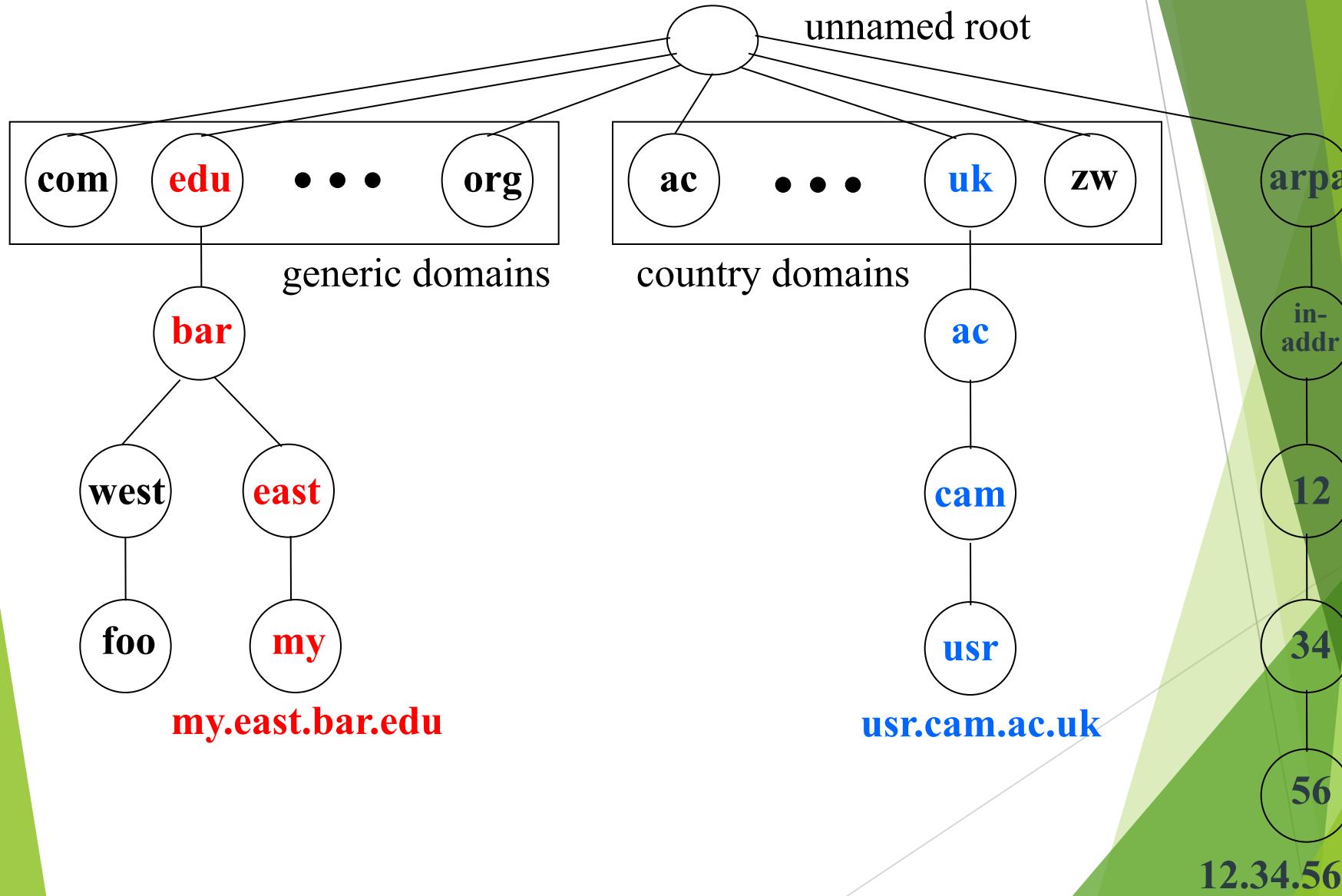
- ▶ 13 root servers (see <http://www.root-servers.org/>)
- ▶ Labeled A through M



TLD and Authoritative DNS Servers

- ▶ Top-level domain (TLD) servers
 - ▶ Generic domains (e.g., com, org, edu)
 - ▶ Country domains (e.g., uk, fr, ca, jp)
 - ▶ Typically managed professionally
 - ▶ Network Solutions maintains servers for “com”
 - ▶ Educause maintains servers for “edu”
- ▶ Authoritative DNS servers
 - ▶ Provide public records for hosts at an organization
 - ▶ For the organization’s servers (e.g., Web and mail)
 - ▶ Can be maintained locally or by a service provider

Distributed Hierarchical Database

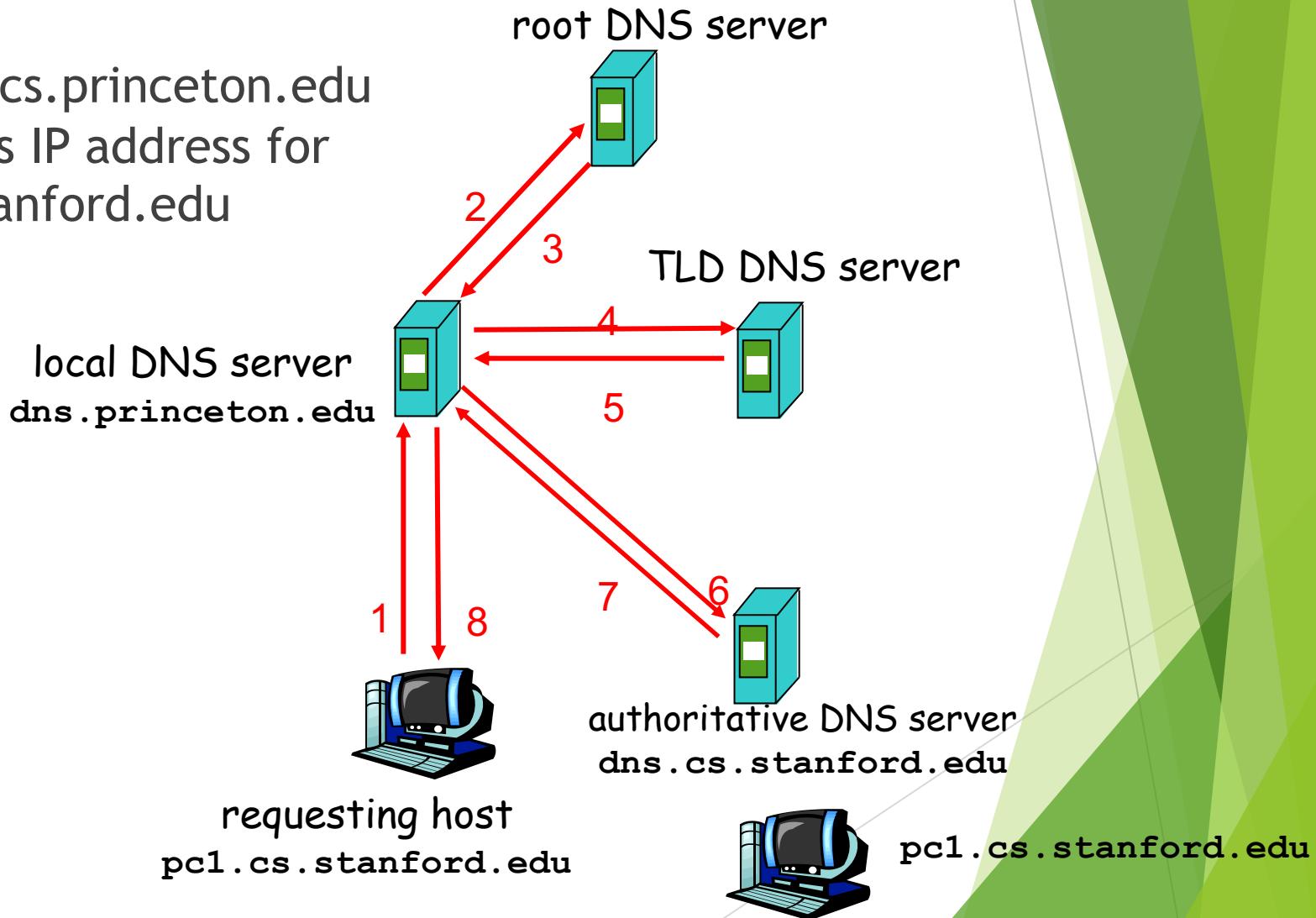


Using DNS

- ▶ Local DNS server (“default name server”)
 - ▶ Usually near the end hosts who use it
 - ▶ Local hosts configured with local server (e.g.,
`/etc/resolv.conf`) or learn the server via DHCP
- ▶ Client application
 - ▶ Extract server name (e.g., from the URL)
 - ▶ Do `gethostbyname()` to trigger resolver code
- ▶ Server application
 - ▶ Extract client IP address from socket
 - ▶ Optional `gethostbyaddr()` to translate into name

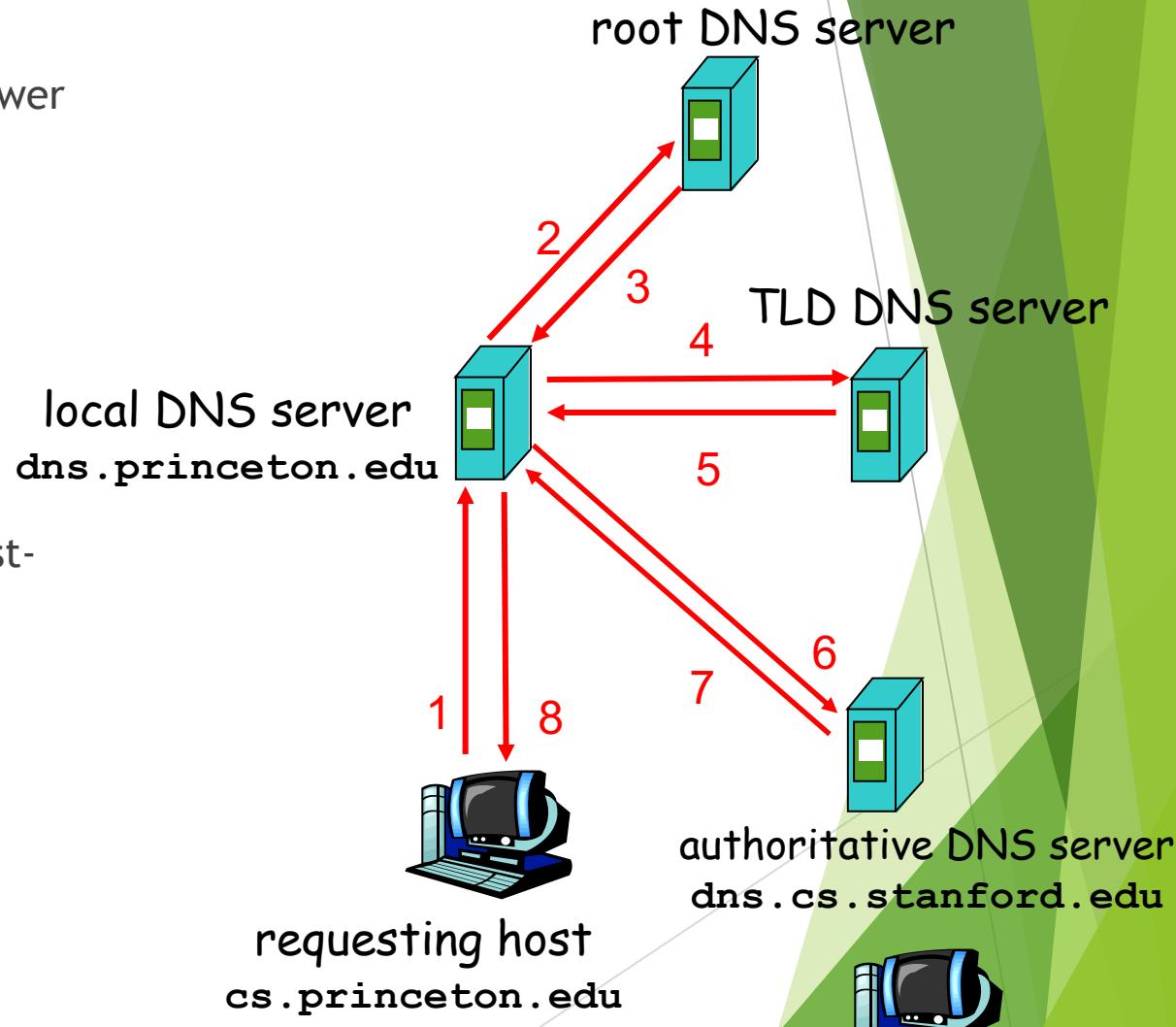
Example

Host at `cs.princeton.edu`
wants IP address for
`cs.stanford.edu`



Recursive vs. Iterative Queries

- ▶ Recursive query
 - ▶ Ask server to get answer for you
 - ▶ E.g., request 1 and response 8
- ▶ Iterative query
 - ▶ Ask server who to ask next
 - ▶ E.g., all other request-response pairs



DNS Caching

- ▶ Performing all these queries take time
 - ▶ And all this before the actual communication takes place
 - ▶ E.g., 1-second latency before starting Web download
- ▶ Caching can substantially reduce overhead
 - ▶ The top-level servers very rarely change
 - ▶ Popular sites (e.g., www.cnn.com) visited often
 - ▶ Local DNS server often has the information cached
- ▶ How DNS caching works
 - ▶ DNS servers cache responses to queries
 - ▶ Responses include a “time to live” (TTL) field
 - ▶ Server deletes the cached entry after TTL expires

Negative Caching

- ▶ Remember things that don't work
 - ▶ Misspellings like www.cnn.comm and www.cnna.com
 - ▶ These can take a long time to fail the first time
 - ▶ Good to remember that they don't work
 - ▶ ... so the failure takes less time the next time around

DNS Resource Records

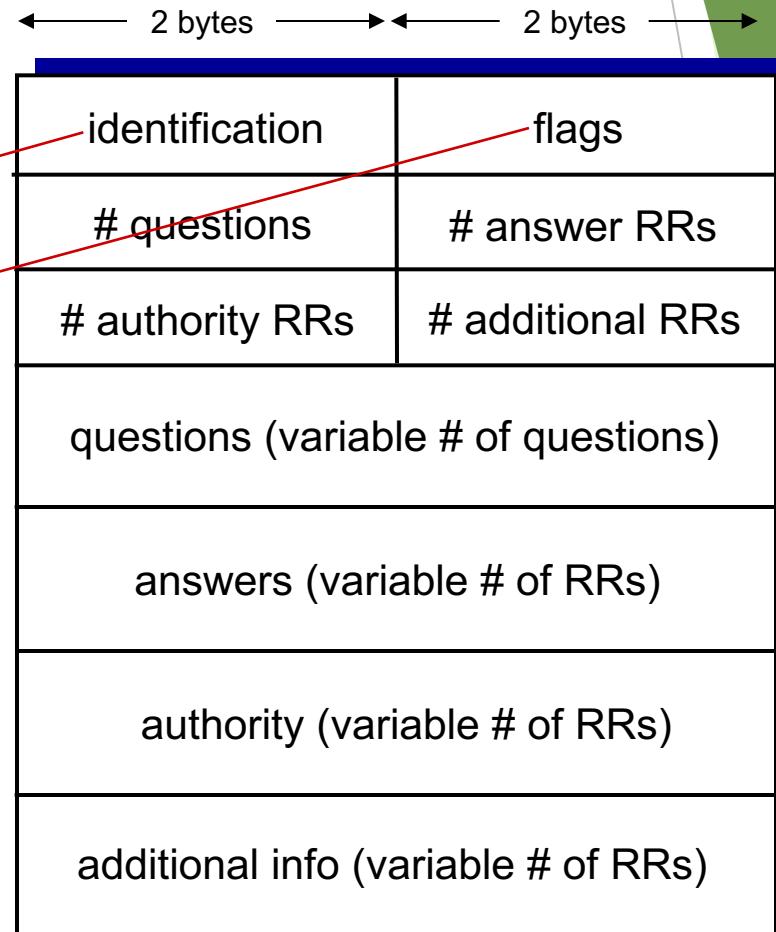
DNS: distributed db storing resource records (**RR**)

RR format: `(name, value, type, ttl)`

- Type=A
 - name is hostname
 - value is IP address
- ▶ Type=NS
 - ▶ name is domain (e.g. foo.com)
 - ▶ value is hostname of authoritative name server for this domain
- Type=CNAME
 - name is alias name for some “canonical” (the real) name
www.ibm.com is really servereast.backup2.ibm.com
 - value is canonical name
- Type=MX
 - value is name of mailserver associated with name

DNS protocol, messages

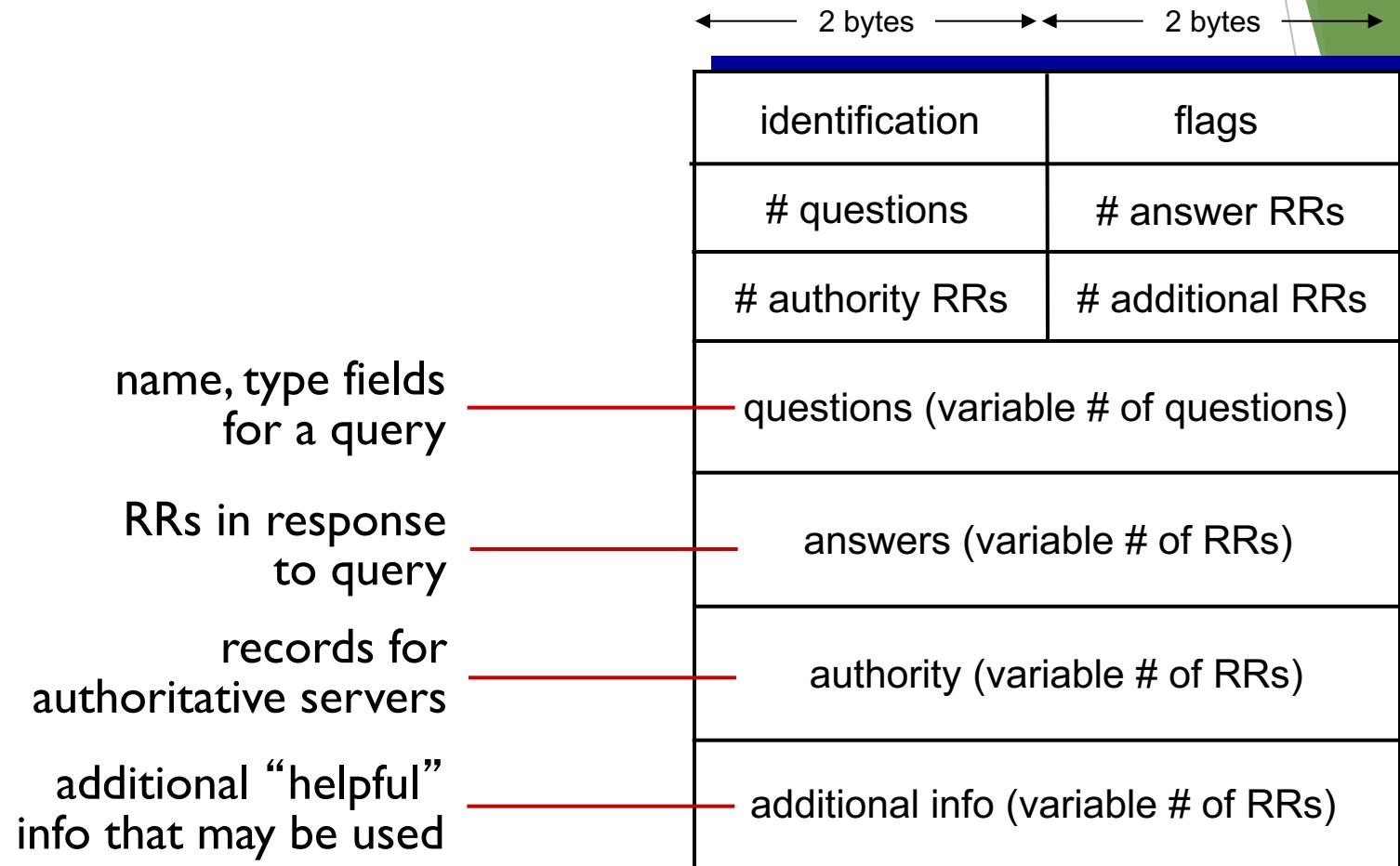
- ▶ *query* and *reply* messages, both with same *message format*



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



Reliability

- ▶ DNS servers are replicated
 - ▶ Name service available if at least one replica is up
 - ▶ Queries can be load balanced between replicas
- ▶ UDP used for queries
 - ▶ Need reliability: must implement this on top of UDP
- ▶ Try alternate servers on timeout
 - ▶ Exponential backoff when retrying same server
- ▶ Same identifier for all queries
 - ▶ Don't care which server responds

Inserting Resource Records into DNS

- ▶ Example: just created startup “FooBar”
- ▶ Register foobar.com at Network Solutions
 - ▶ Provide registrar with names and IP addresses of your authoritative name server (primary and secondary)
 - ▶ Registrar inserts two RRs into the com TLD server:
 - ▶ (foobar.com/, dns1.foobar.com, NS)
 - ▶ (dns1.foobar.com, 212.212.212.1, A)
- ▶ Put in authoritative server dns1.foobar.com
 - ▶ Type A record for www.foobar.com
 - ▶ Type MX record for foobar.com

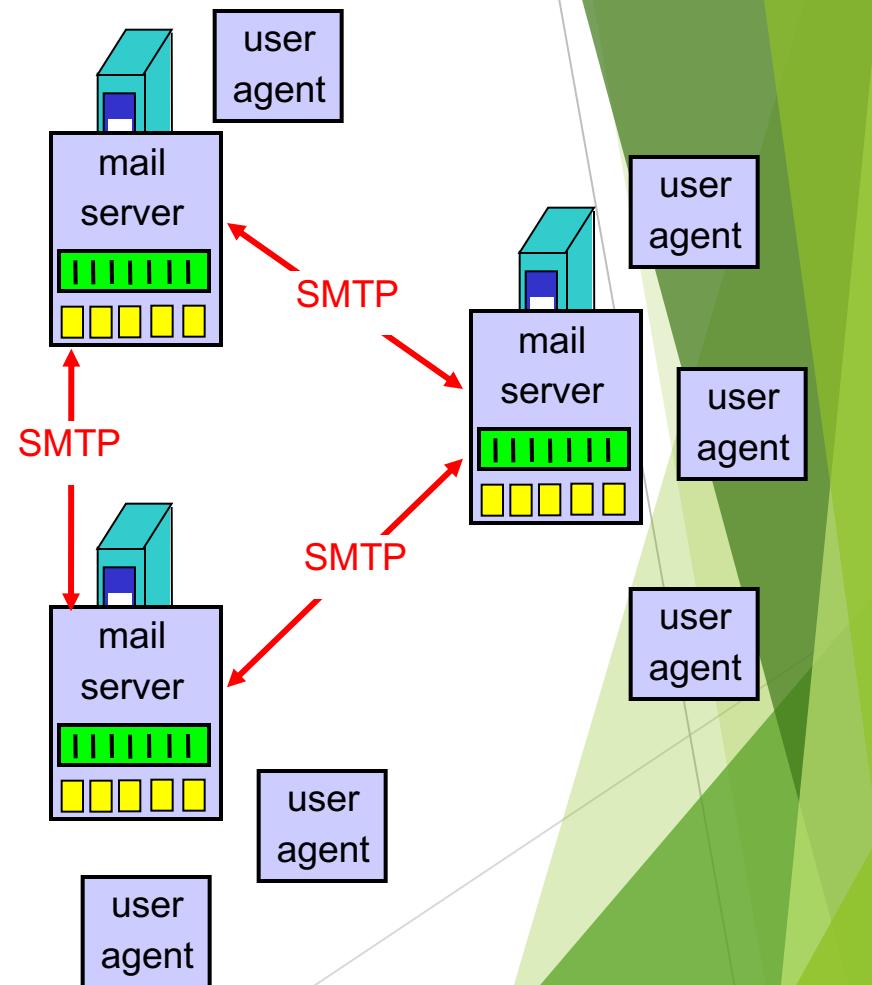


Electronic Mail (e-Mail)

- ▶ Three major components:
 - ▶ user agents
 - ▶ mail servers
 - ▶ simple mail transfer protocol: SMTP

User Agent

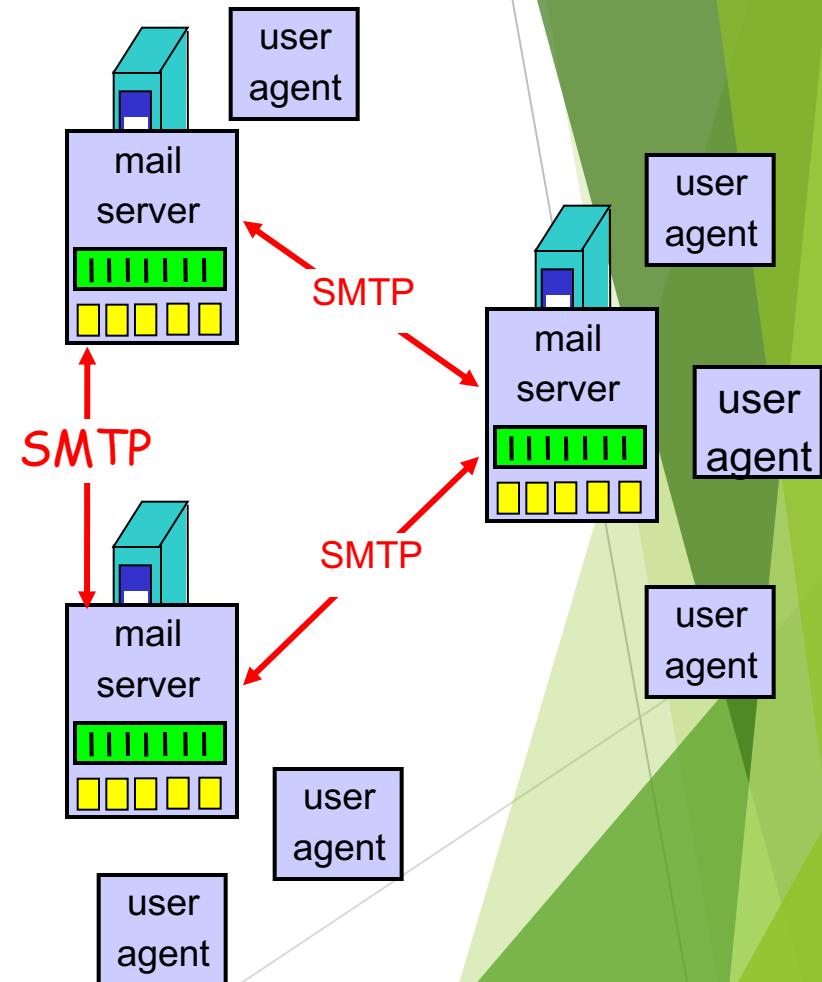
- ▶ a.k.a. “mail reader”
- ▶ composing, editing, reading mail messages
- ▶ e.g., Outlook, Mozilla Thunderbird, Apple Mail
- ▶ outgoing, incoming messages stored on server



Electronic Mail: mail servers

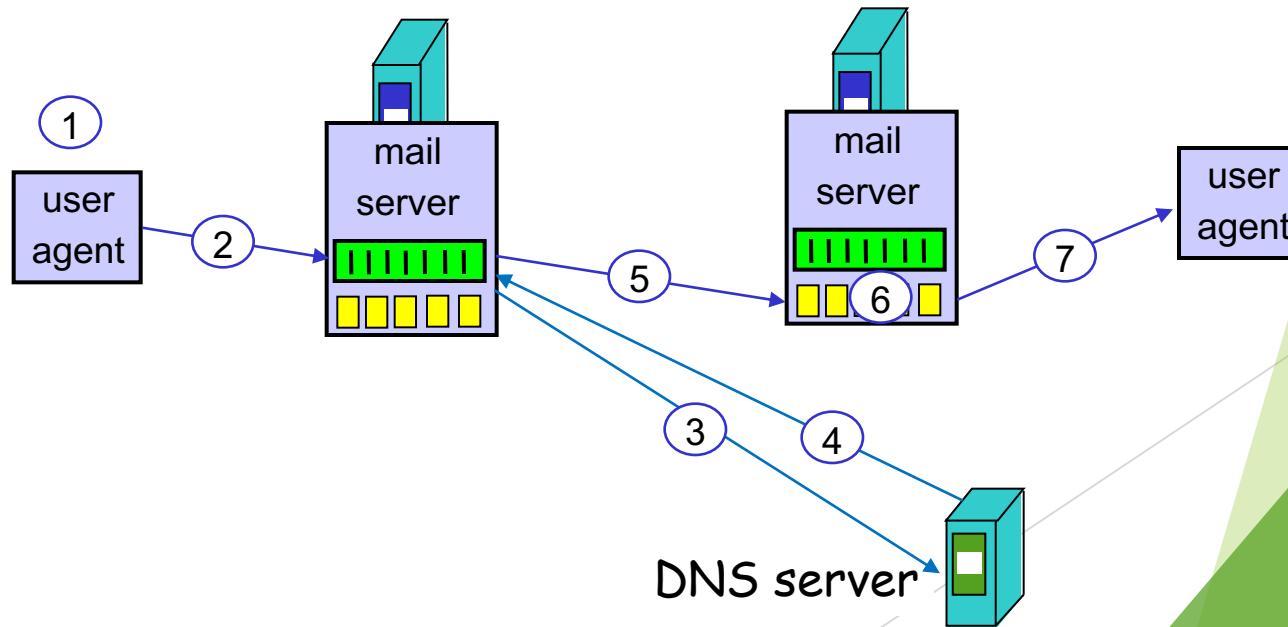
► Mail Servers

- **mailbox** contains incoming messages for user
- **message queue** of outgoing (to be sent) mail messages
- **SMTP protocol** between mail servers to send email messages
 - ❖ client: sending mail server
 - ❖ “server”: receiving mail server



Scenario: Alice sends message to Bob

- 1) Alice uses UA to compose message and "to" bob@someschool.edu
- 2) Alice's UA sends message to her mail server; message placed in message queue
- 3) The part before the @ sign is the local part of the address, often the username of the recipient, and the part after the @ sign is a domain name. The mail server looks up this domain name in the Domain Name System to find the mail exchange servers accepting messages for that domain (i.e., someschool.edu).
- 4) The DNS server responds with an MX record listing the mail exchange servers for that domain.
- 5) Client side of SMTP opens TCP connection with Bob's mail server using the resolved IP address and sends Alice's message over the TCP connection
- 6) Bob's mail server places the message in Bob's mailbox
- 7) Bob invokes his user agent to read message



Electronic Mail: SMTP [RFC 2821]

- ▶ uses TCP to reliably transfer email message from client to server, port 25
- ▶ direct transfer: sending server to receiving server
- ▶ three phases of transfer
 - ❖ handshaking (greeting)
 - ❖ transfer of messages
 - ❖ closure
- ▶ command/response interaction
 - ❖ **commands:** ASCII text
 - ❖ **response:** status code and phrase
- ▶ messages must be in 7-bit ASCII

Try SMTP interaction for yourself:

- ▶ `telnet servername 25`
- ▶ see 220 reply from server
- ▶ enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

above lets you send email without using
email client (reader)

Trace it - does your mail data go in the clear?

Sample SMTP interaction

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

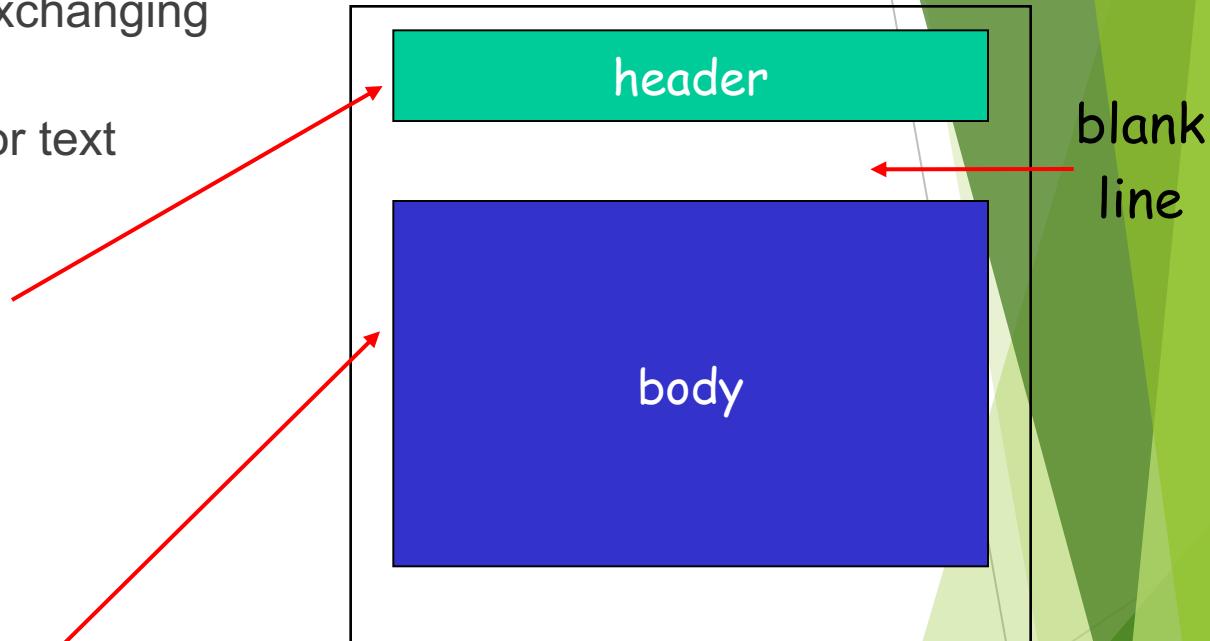
Mail message format

- ▶ SMTP: protocol for exchanging email msgs
- ▶ RFC 822: standard for text message format:
- header lines, e.g.,

- To:
- From:
- CC:
- Subject:
- Date:

different from SMTP commands!

- body
 - the “message”, ASCII characters only



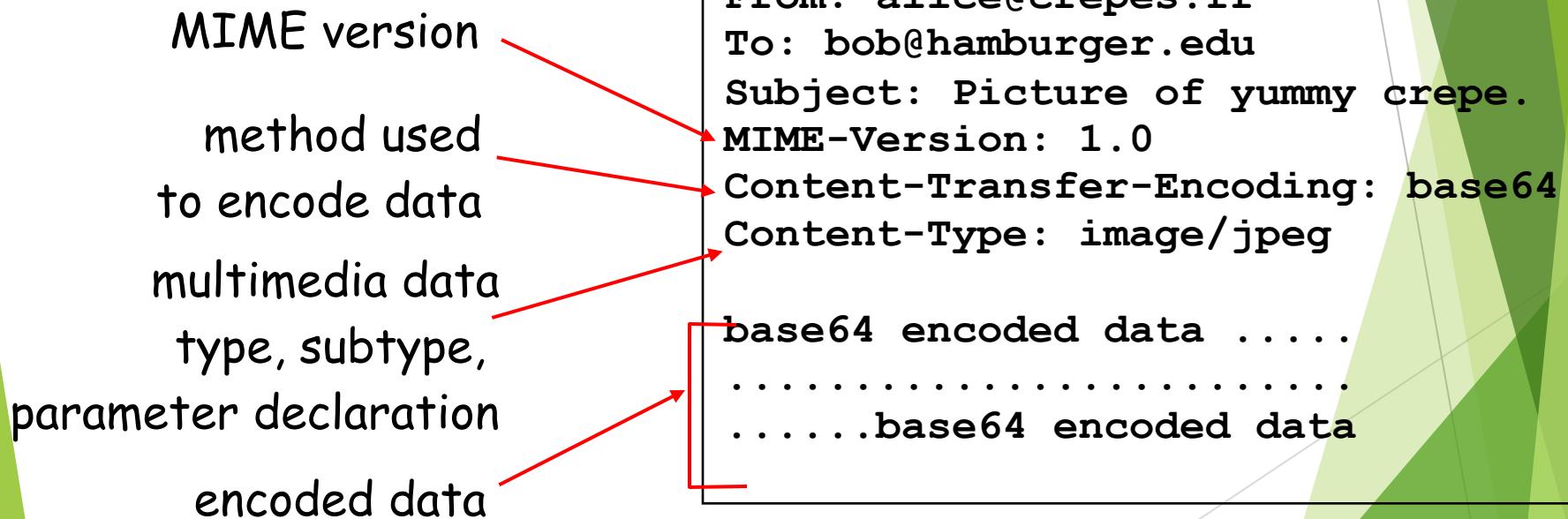
Message format: multimedia extensions

- ▶ MIME: multimedia mail extension, RFC 2045, 2056
- ▶ additional lines in msg header declare MIME content type

MIME version
method used
to encode data
multimedia data
type, subtype,
parameter declaration
encoded data

From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data
.....
.....base64 encoded data



MIME types: Extensible

Content-Type: type/subtype; parameters

Text

- ▶ example subtypes:
plain, html

Image

- ▶ example subtypes:
jpeg, gif

Audio

- ▶ example subtypes:
basic (8-bit mu-law encoded), **32kadpcm** (32 kbps coding)

Video

- ▶ example subtypes:
mpeg, quicktime

Application

- ▶ other data that must be processed by reader before “viewable”
- ▶ example subtypes:
msword, octet-stream

Multipart Type

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=98766789
```

--98766789

```
Content-Transfer-Encoding: quoted-printable
Content-Type: text/plain
```

Dear Bob,

Please find a picture of a crepe.

--98766789

```
Content-Transfer-Encoding: base64
Content-Type: image/jpeg
```

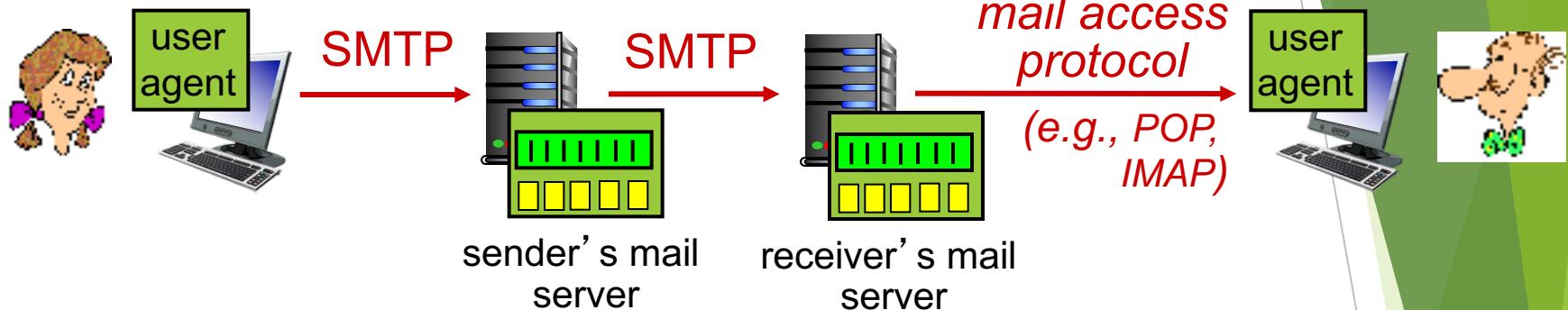
base64 encoded data

.....

.....base64 encoded data

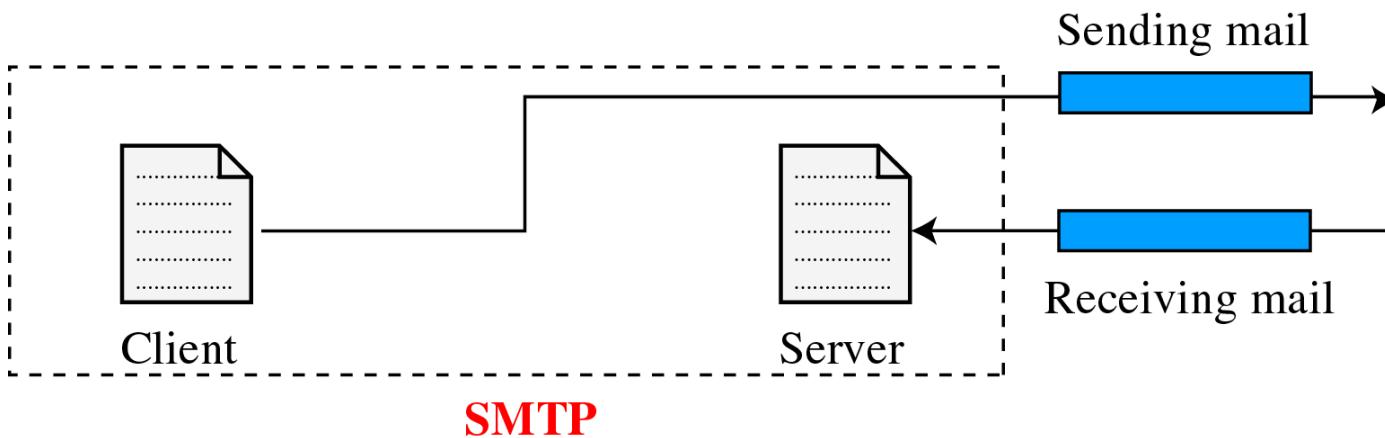
--98766789--

Mail access protocols

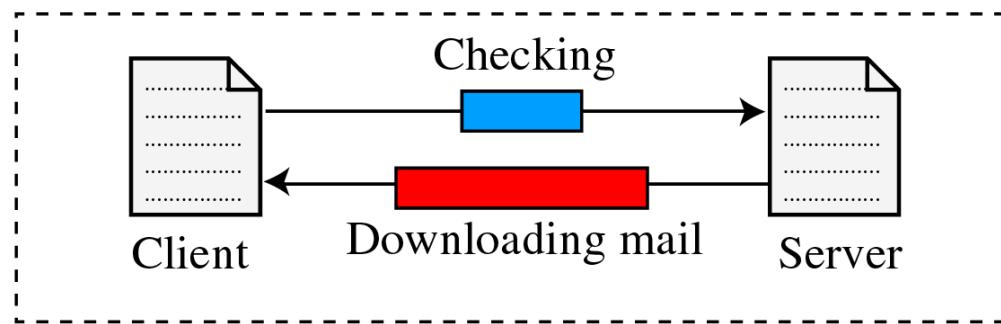


- ▶ **SMTP:** delivery/storage to receiver's server
- ▶ Mail access protocol: retrieval from server
 - ▶ **POP:** Post Office Protocol [RFC 1939]: authorization, download
 - ▶ **IMAP:** Internet Mail Access Protocol [RFC 1730]: more features, including manipulation of stored msgs on server
 - ▶ **HTTP:** Gmail, Hotmail, Yahoo! Mail, etc.

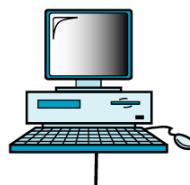
POP3 and SMTP



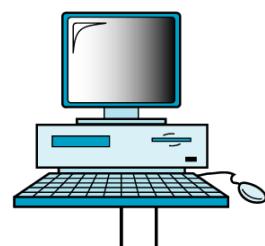
SMTP



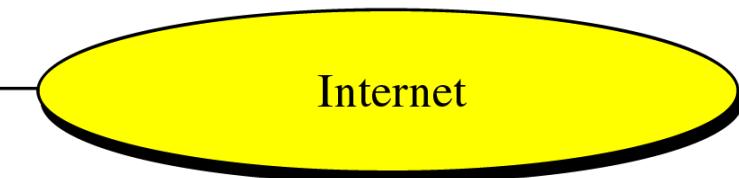
POP3



Desktop



Mail server



Internet

POP3 protocol

- ▶ **authorization phase**
- client commands:
 - ❖ **user**: declare username
 - ❖ **pass**: password
- server responses
 - ❖ **+OK**
 - ❖ **-ERR**

transaction phase, client:

- **list**: list message numbers
- **retr**: retrieve message by number
- **dele**: delete
- **quit**

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on

C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

IMAP - Commands

- ▶ login
- ▶ list
- ▶ status
- ▶ examine
- ▶ select
- ▶ create, delete, rename
- ▶ fetch
- ▶ store
- ▶ close
- ▶ expunge
- ▶ copy
- ▶ idle
- ▶ lsub, subscribe, unsubscribe
- ▶ logout
- ▶ capability, getquotaroot, getacl

POP3 (more) and IMAP

► More about POP3

- ▶ Previous example uses “download and delete” mode.
- ▶ Bob cannot re-read e-mail if he changes client
- ▶ “Download-and-keep”: copies of messages on different clients
- ▶ POP3 is stateless across sessions

► IMAP

- ▶ Keep all messages in one place: the server
- ▶ Allows user to organize messages in folders
- ▶ IMAP keeps user state across sessions:
 - ▶ names of folders and mappings between message IDs and folder name

SMTP vs. HTTP

- ▶ SMTP uses persistent connections
- ▶ SMTP requires message (header & body) to be in 7-bit ASCII

comparison with HTTP:

- ▶ HTTP: pull
- ▶ SMTP: push
- ▶ both have ASCII command/response interaction, status codes
- ▶ HTTP: each object encapsulated in its own response msg
- ▶ SMTP: multiple objects sent in multipart msg

VoIP - Basics and Protocols

CEN 223 - Internet Communication

Assoc. Prof. Dr. Fatih ABUT

Department of Computer Engineering

Çukurova University

What is VoIP?

Transmission of **voice** and **signaling**

via data networks

using Internet Protocol (IP),

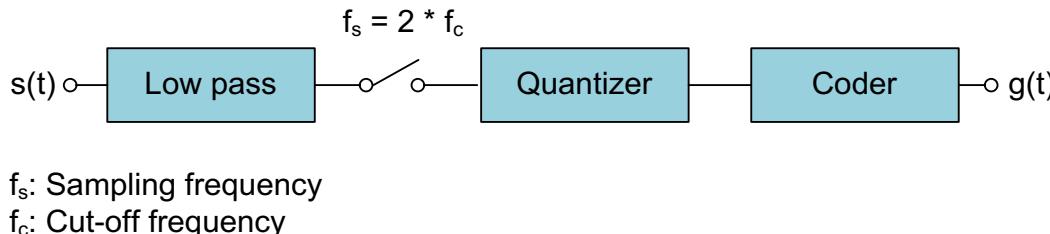
i.e. via so-called **IP networks** (Internet,
Intranets)

Basics of how it works

- ▶ We digitalize audio with an ADC (analog to digital converter)
- ▶ Transmit it over a digital network (Internet)
- ▶ Transform it again in analog format with DAC (digital to analog converter)
- ▶ In other words, digitalizing voice in data packets, sending them and reconverting them in voice at destination
- ▶ Why digitize?
 - ▶ We can compress it
 - ▶ We can route it
 - ▶ We can convert it to a new better format (more compressed with similar or no quality loss to the human ear as the previous format)

Analog to Digital Conversion (1)

- ▶ This is made by hardware. Today every sound card allows you convert with 16 bit a band of 22050 Hz.
- ▶ The objective is that a VoIP terminal ultimately generates a digital signal from an analog sound signal.
- ▶ As input, the VoIP terminal receives a sound signal $s(t)$, which is of an analog nature, and provides at the output a signal $g(t)$, which is in binary form.



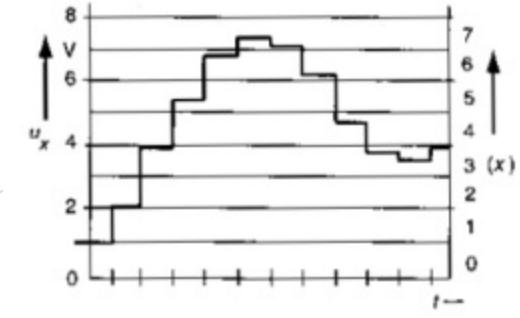
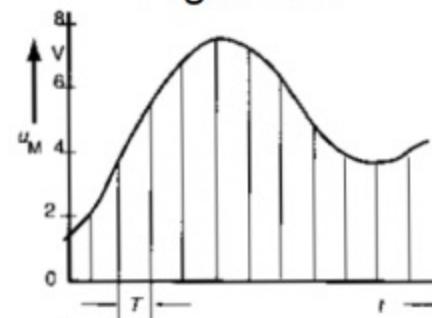
Conversion of an analog signal into a digital signal.

Analog to Digital Conversion (2)

- ▶ If an analog signal $s(t)$ arrives, first a band limitation takes place. Particularly, the analog signal is narrowed in its frequency spectrum, provided that it can be reconstructed as clearly as possible afterward.
- ▶ This band limitation is caused by an element called a low pass. This maximum band-limited signal, also referred to as the cut-off frequency f_c , is determined by speech intelligibility.
- ▶ The conventional telephony has a **cutoff frequency of 4 kHz**.

Analog to Digital Conversion (3)

- ▶ The steps to be taken to convert this band-limited signal $s(t)$ into a binary stream $g(t)$ can be outlined as follows:
 - **Sampling:** The first step is to convert the signal from a time- and value-continuous form into a time-discrete form. For this purpose, the analog signal is sampled at twice the cutoff frequency f_c .
 - **Quantization:** Thereafter, the sampled signal must be quantized to convert it from the time-discrete, but value-continuous form into a time- and value-discrete form.
 - **Coding and compression:** The time- and value-discrete signal must finally be coded and, if necessary, compressed.

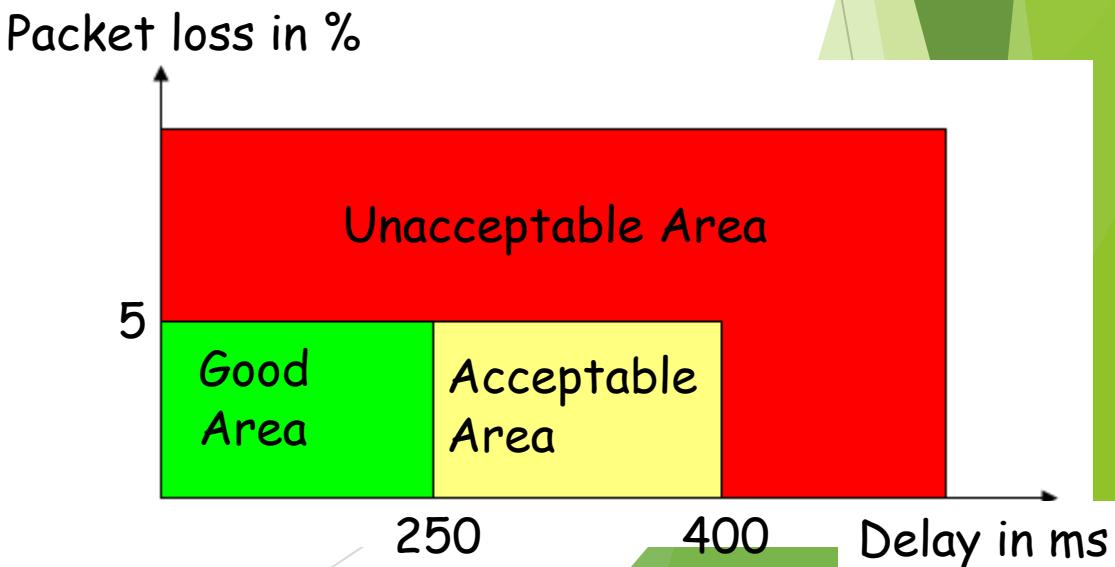


Compression Algorithms

- ▶ Voice bandwidth is 4 kHz, so sampling bandwidth has to be 8 kHz (**according to Nyquist-Shannon sampling theorem**)
- ▶ We represent each sample with 8 bit (having 256 possible values).
- ▶ Throughput is $8000 \text{ Hz} * 8 \text{ bit} = 64 \text{ kbit/s}$, as a typical digital phone line.
- ▶ PCM, Pulse Code Modulation, Standard ITU-T G.711
- ▶ (North America) mu-law and (Europe) a-law variants code analog signal on a logarithmic scale using 12 or 13 bits instead of 8 bits

Requirements for VoIP networks

- ▶ Requirements from the user's point of view
 - ▶ Voice quality
 - ▶ Availability and Reachability
 - ▶ Eavesdropping security
 - ▶ Performance characteristics
 - ▶ Flexibility
- ▶ Requirements from a technical point of view
 - ▶ Delay
 - ▶ Jitter
 - ▶ Echo
 - ▶ Packet losses
 - ▶ Distortions



VoIP Signaling (1)

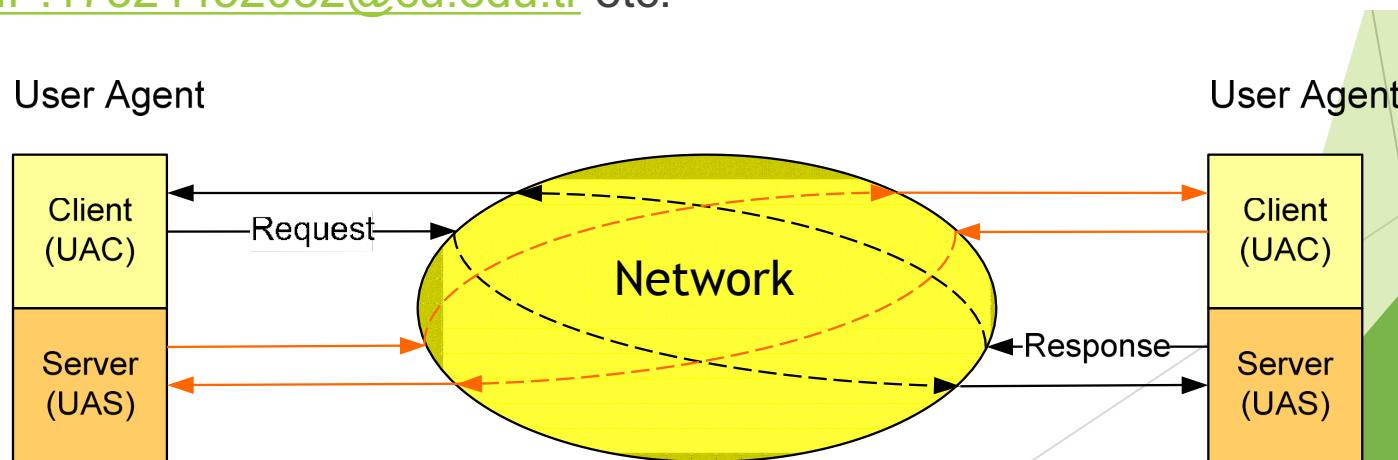
- ▶ Signaling
 - ▶ Session Initiation Protocol (SIP), [IETF]
 - ▶ H323, [ITU]
 - ▶ Skinny Client Control Protocol (SCCP), [Cisco]
 - ▶ Inter Asterisk Exchange Protocol (IAX™), [Digium]
- ▶ Protocols for the transmission of multimedia / voice data
 - ▶ Real Time Protocol (RTP) and
 - ▶ Real Time Control Protocol (RTCP), [IETF]

VoIP Signaling (2)

- ▶ Components
 - ▶ User Agent
 - ▶ „Hard“ or „softphone“ acting as participant equipment
 - ▶ User Agent Client (UAC) and User Agent Server (UAS)
 - ▶ Proxy
 - ▶ Routing for signaling messages
 - ▶ stateless or statefull)
 - ▶ Redirect Server
 - ▶ Registrar Server
 - ▶ Participant management
 - ▶ Participant mobility
 - ▶ Location Server
 - ▶ Mapping of permanent to temporary addresses

Session Initiation Protocol (SIP)

- ▶ SIP: Protocol of the application layer, RFC 3261 - 269 pages
- ▶ SIP supports the establishment and operation of so-called sessions for multimedia communication (**voice, video, text**) between Internet users
- ▶ Two or more user agents are involved in a session
- ▶ SIP supports the use of a network infrastructure for registering and localizing participants, receiving connection requests, etc. => Proxy server
- ▶ **SIP uses naming similar to e-mail** (SIP:badri@ceng.cu.edu.tr, SIP:17324452082@cu.edu.tr etc.)



SIP uses a request / response model (like HTTP)

Session Initiation Protocol (SIP) (2)

SIP Facets of Establishing and Terminating
Multimedia Communications

User Location

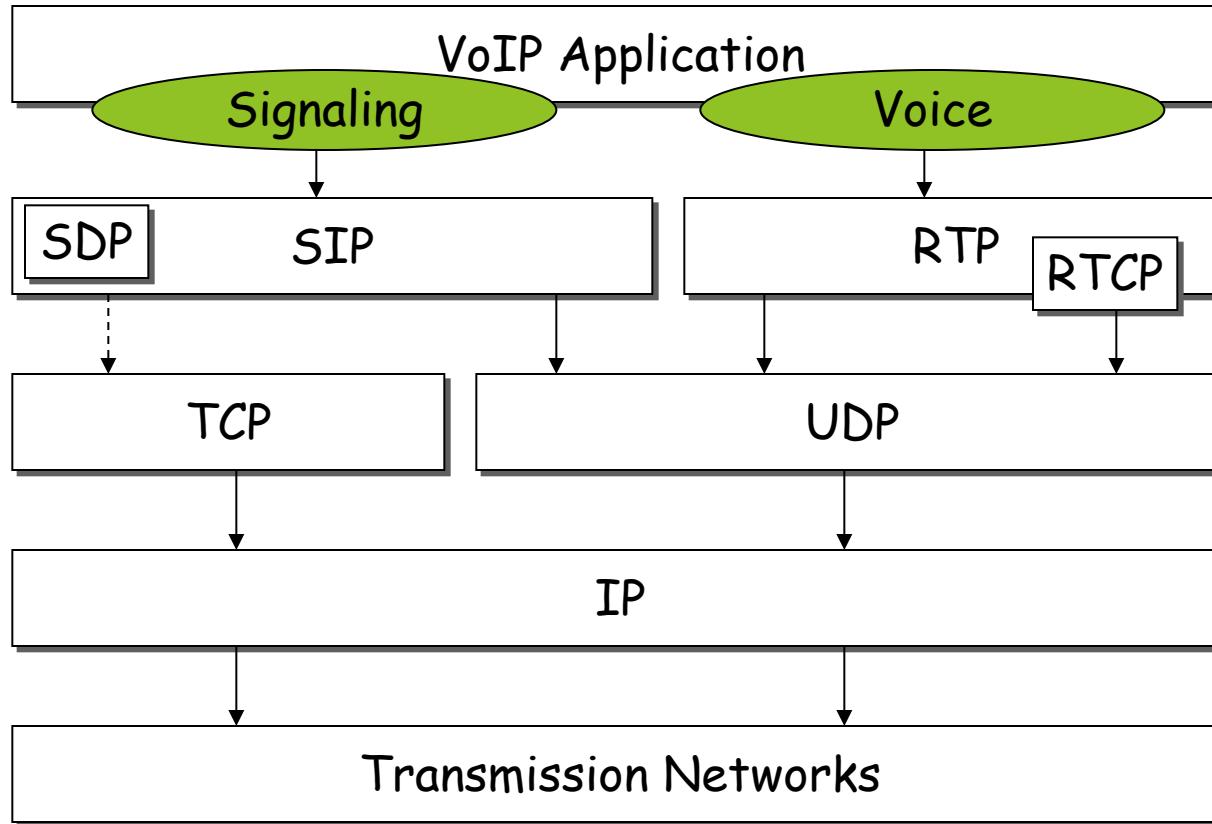
User Availability

User Capabilities

Session Setup

Session Management

Session Initiation Protocol (SIP) (3)



Classification of SIP and RTP

SIP: Session Initiation Protocol

SDP: Session Description Protocol

RTP: Real Time Transport Protocol

RTCP: Real Time Control Protocol

SIP Request Messages

Request method	Purpose
INVITE	Initiate a call.
ACK	Confirm the final response to an INVITE.
BYE	Terminate a call.
CANCEL	Cancel searches and “ringing”
OPTIONS	Communicate features supported
REGISTERED	Register a client with a location service.

SIP Response Messages

Status-Code	Category	Example information
1xx	Informational	trying, ringing, call is being forwarded, queued
2xx	Success	OK
3xx	Redirection	Moved permanently, moved temporarily, etc
4xx	Client error	Bad request, unauthorized, not found, busy, etc
5xx	Server error	Server error, not implemented, bad gateway, etc
6xx	Global failure	Busy everywhere, does not exist anywhere, etc.

100 Trying

180 Ringing

181 Call is being Forwarded

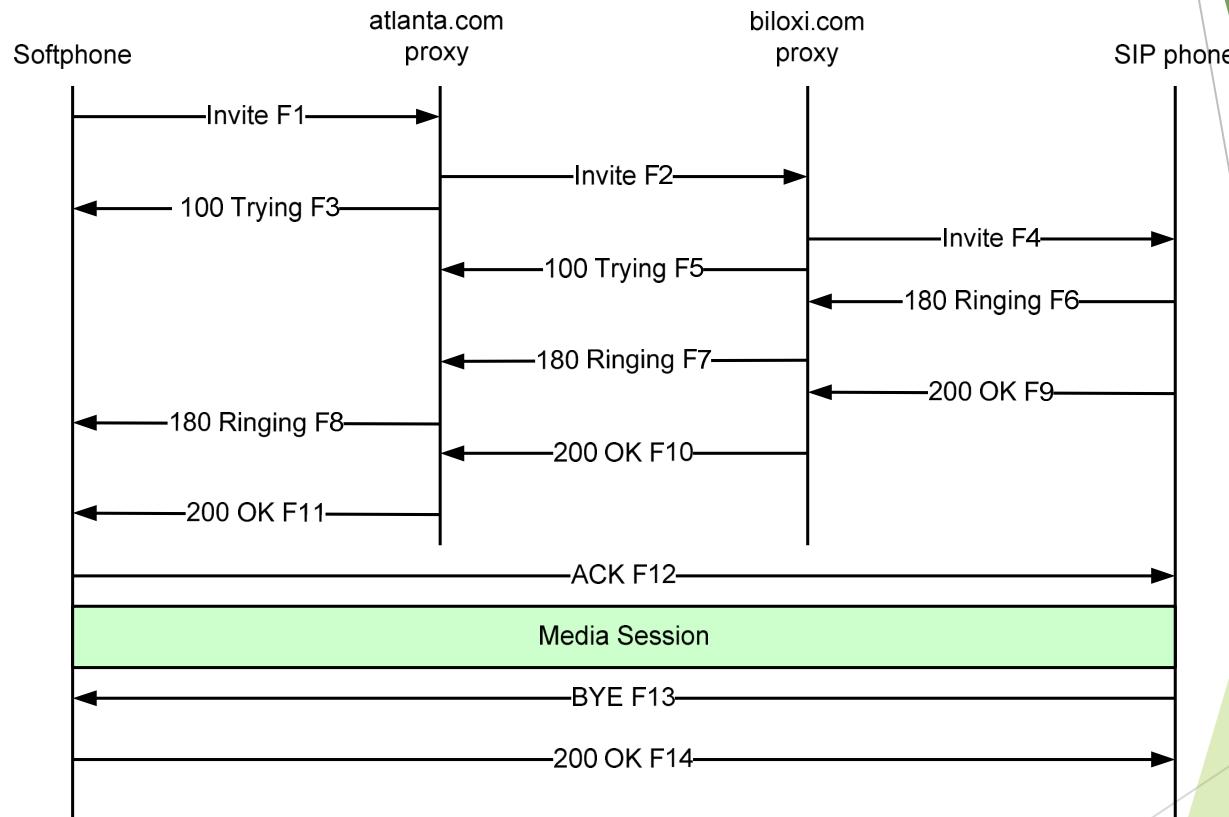
182 Queued

200 OK

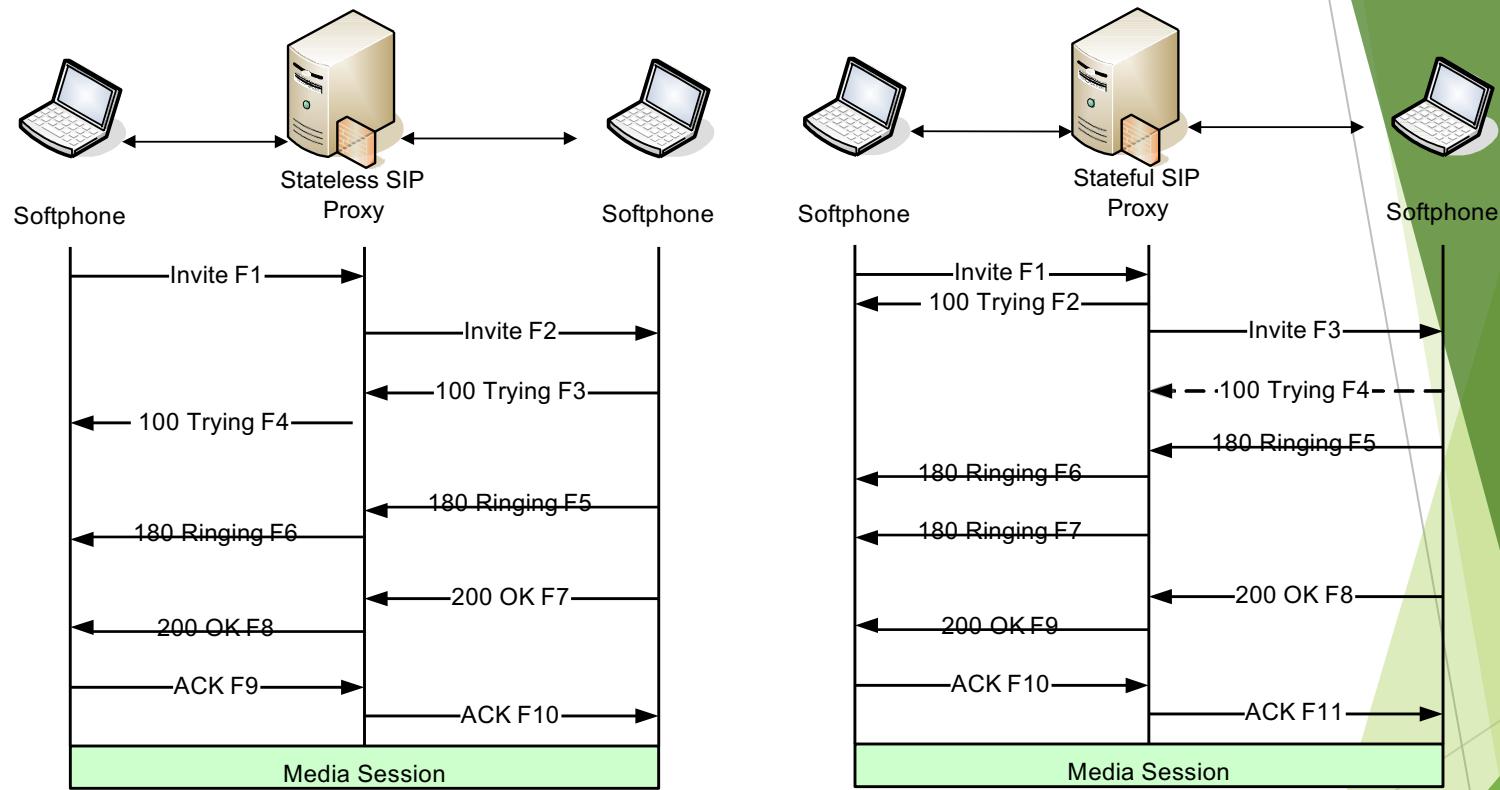
301 Moved Permanently

302 Moved Temporarily

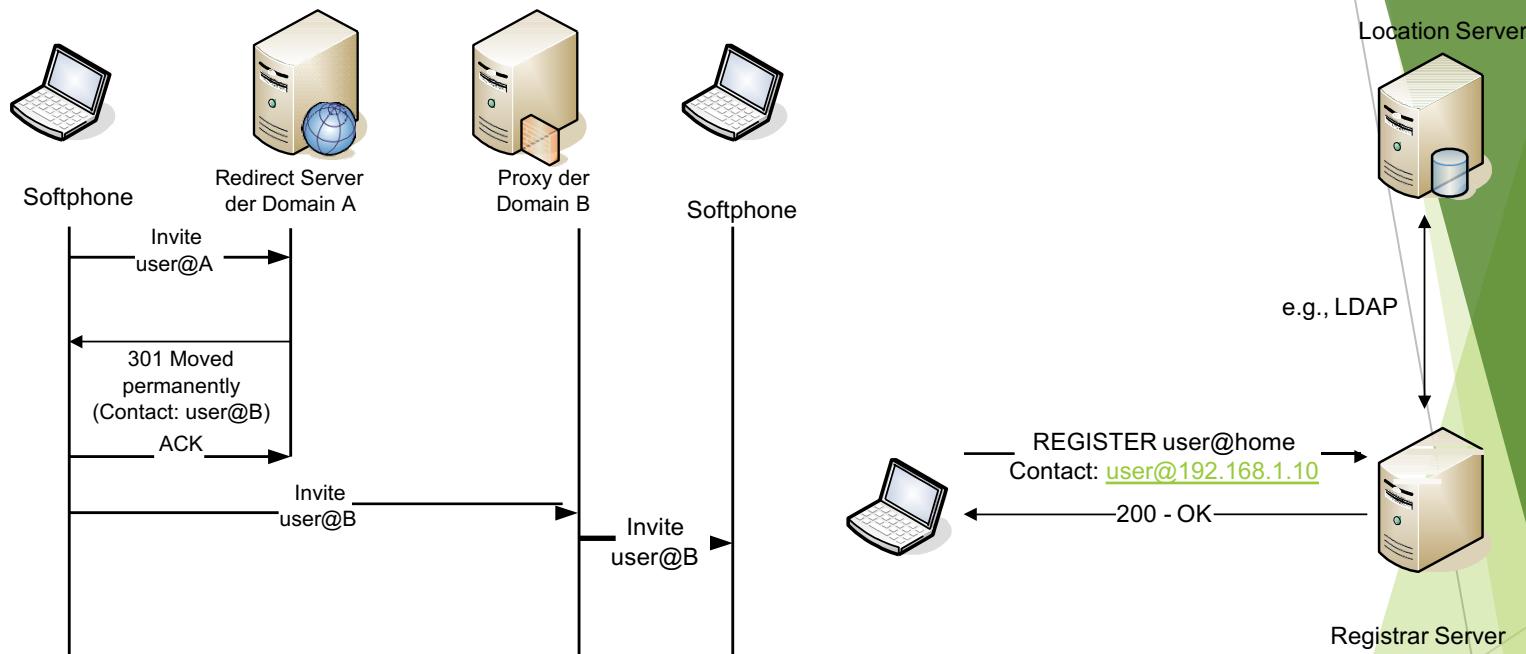
Example: Setting up and triggering a "session"



SIP Network Elements (1)



SIP Network Elements (2)



SIP: Registration & Discovery

Register Request:

Request URI:

domain of the location service
(e.g. sip@chicago.com)

To:

contains address-of-record
(SIP URI)

From:

address-of-record of the person
responsible for the registration

Call-ID:

UAC should use only one CALL-ID
for all registrations to a
particular registrar

Cseq:

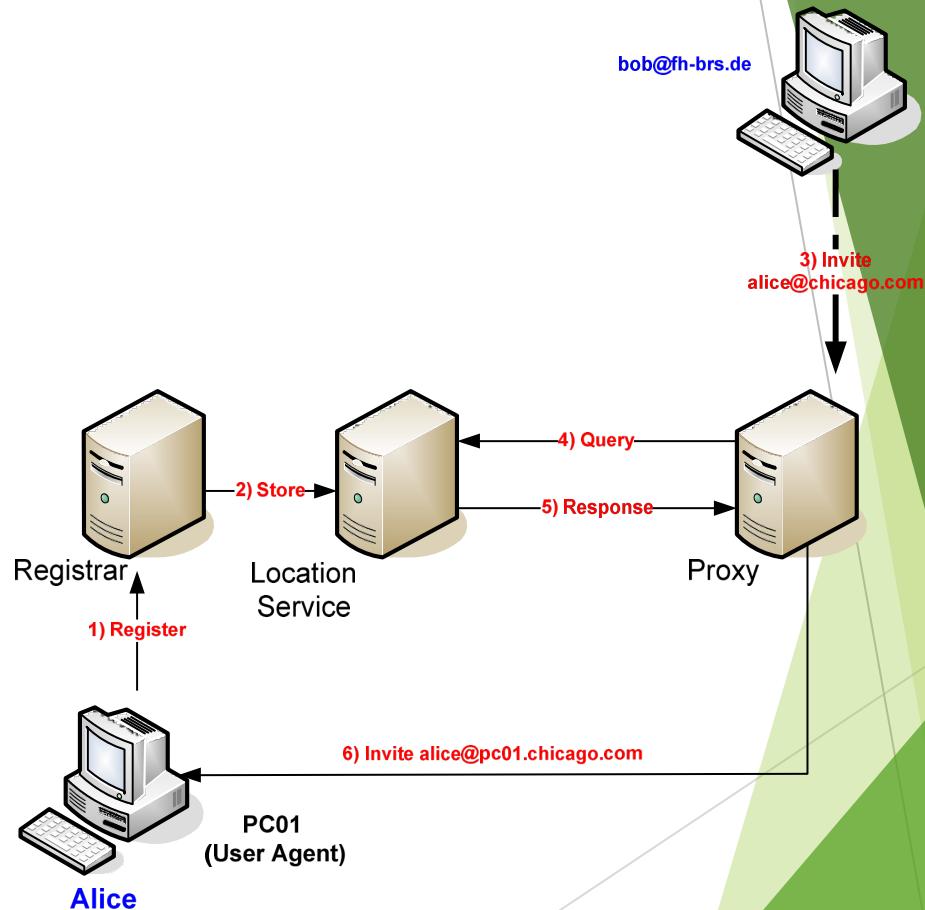
>> sequence number <<

Contact (optional):

address bindings

Expires (optional):

expiration for the binding



SIP - Message

Session Initiation Protocol

```
Request-Line: INVITE sip:10000@sipgate.de SIP/2.0
Method: INVITE
[Resent Packet: False]
```

Message Header

```
Via: SIP/2.0/UDP 192.168.178.102:5060;rport;branch=z9hG4bK60846
Max-Forwards: 70
To: <sip:10000@sipgate.de>
From: <sip:user@sipgate.de>;tag=z9hG4bK66687806
Call-ID: 841479524486@192.168.178.102
CSeq: 1 INVITE
Contact: <sip:user@192.168.178.102
Expires: 3600
User-Agent: mjsip stack 1.6
Content-Length: 159
Content-Type: application/sdp
```

Message body

```
Session Description Protocol
Session Description Protocol Version (v): 0
Owner/Creator, Session Id (o): <sip:user@sipgate.de> 0 0 IN IP4 192.168.178.102
Session Name (s): Session SIP/SDP
Connection Information (c): IN IP4 192.168.178.102
Time Description, active time (t): 0 0
Media Description, name and address (m): audio 21000 RTP/AVP 0
Media Attribute (a): rtpmap:0 PCMU/8000
```

► Free Software Clients

- Ekiga (Unix, Windows)
- Sipgate (Windows)
- Kphone (Unix)

Real-Time Transport Protocol (RTP)

- ▶ Why not TCP?
 - ▶ TCP offers reliable, in-sequence data transfer
 - ▶ TCP embeds flow/error/congestion control
- ▶ Why not UDP?
 - ▶ Missing sequence control and not reliable!
- ▶ **RTP:** transport protocol for multimedia traffic. It carries real-time data.
 - ▶ Application level framing; integrated layer processing
 - ▶ Usually RTP/UDP/IP, or RTP/IP
 - ▶ RTP does NOT guarantee real-time itself!
- ▶ **RTP Control Protocol (RTCP):** It monitors the quality of service and conveys information about the participants.

RTP Applications (1)

► Simple Multicast Audio Conference

- ▶ The audio-conferencing application used by each conference participant sends audio data **in small chunks of, say, 20 ms duration.**
- ▶ Each chunk of audio data is preceded by **an RTP header**; RTP header and data are in turn contained **in a UDP packet.**
- ▶ The RTP header indicates what **type of audio encoding** (such as PCM, ADPCM or LPC) is contained in each packet.
- ▶ RTP header contains **timing information** and a **sequence number** that allow the receivers to reconstruct the timing produced by the source.
- ▶ The sequence number can also be used by the receiver to estimate how many packets are being lost.
- ▶ The audio application in the conference periodically multicasts a **reception report** plus the name of its user on the RTCP port. The reception report indicates how well the current speaker is being received.
- ▶ A site sends the RTCP **BYE packet** when it leaves the conference.

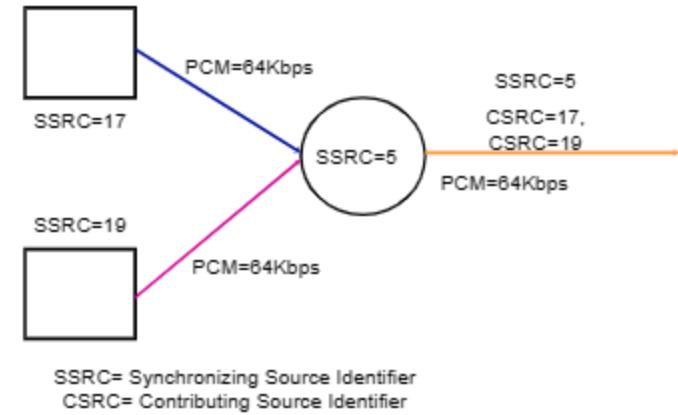
RTP Applications (2)

► Audio and Video Conference

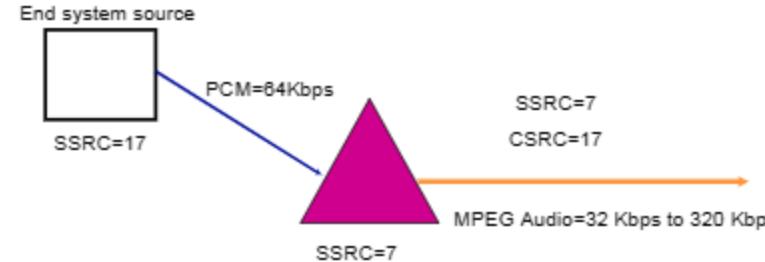
- ▶ Audio and video media are transmitted **as separate RTP session** and RTCP packets are transmitted for each medium using two different UDP port pairs and/or multicast addresses.
- ▶ There is no direct coupling at the RTP level between the audio and video sessions, except that a user participating in both sessions should use the same distinguished (canonical) name in the RTCP packets for both so that the sessions can be associated.
- ▶ Despite the separation, **synchronized playback of a source's audio and video can be achieved using timing information carried in the RTP packets for both sessions.**

RTP Translators and Mixers

- ▶ A mixer combines several media stream into a one new stream (with possible new encoding)
 - ▶ reduced bandwidth networks (video or telephone conference)
 - ▶ appears as new source, with own identifier

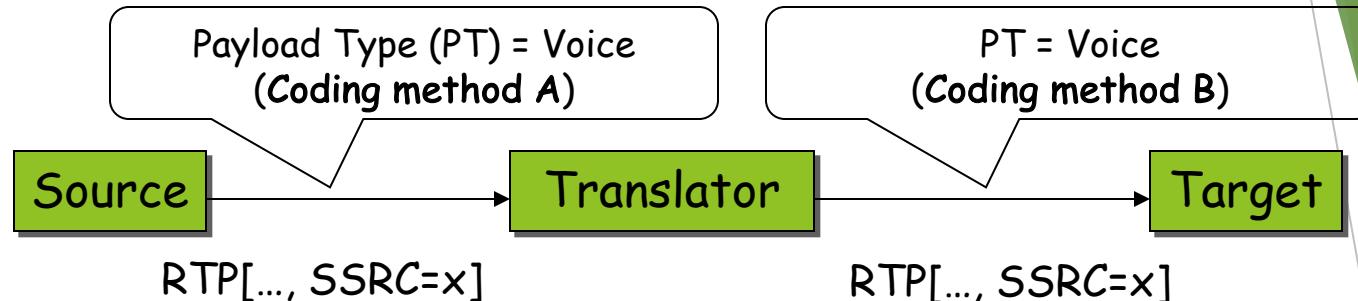


- ▶ A translator changes encoding from one format to another single media stream
 - ▶ may convert encoding



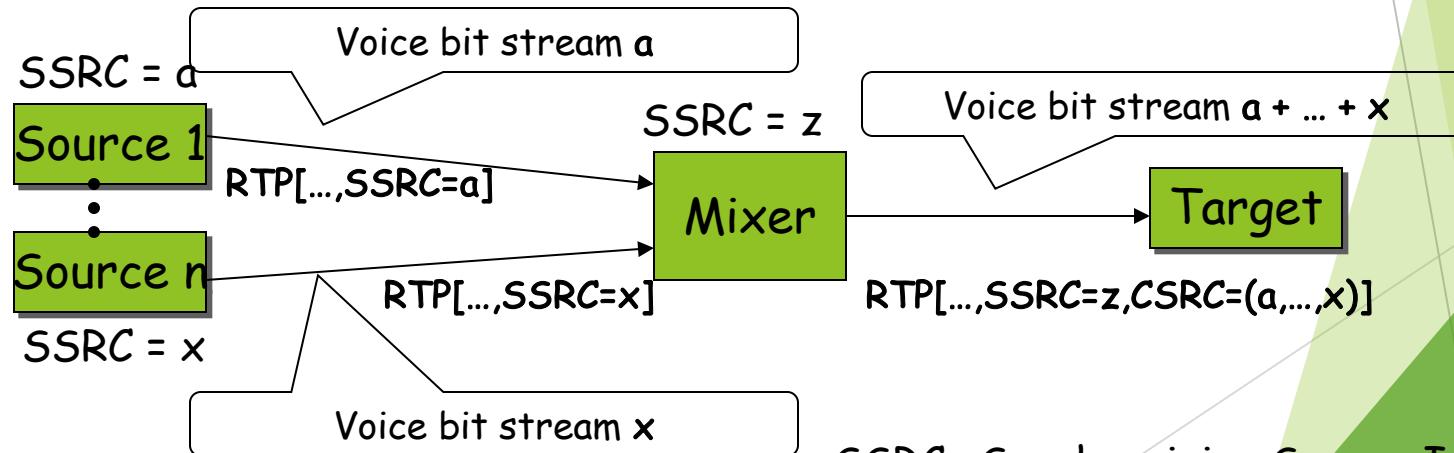
Translator und Mixer

Translator



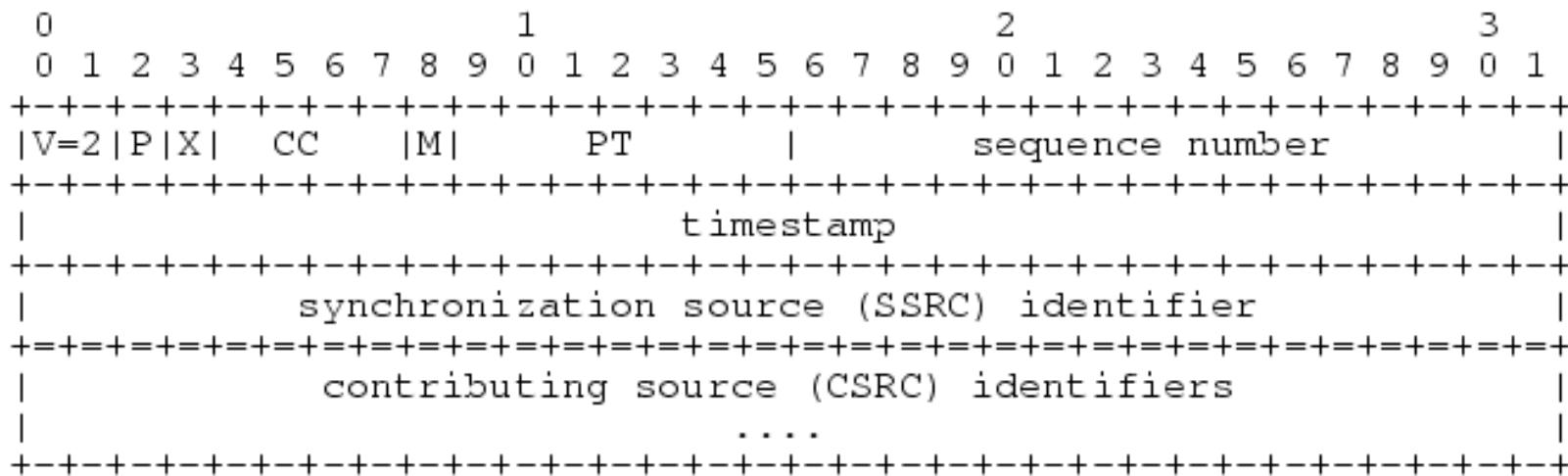
A - Law
μ - Law

Mixer



SSRC= Synchronizing Source Identifier
CSRC= Contribution Source Identifier

Real-Time Transport Protocol



V: Version

P: Padding

X: Extension

CC: Contribution Source Identifier Count

M: Marker

RTP Data Transfer Protocol (2): Synchronization

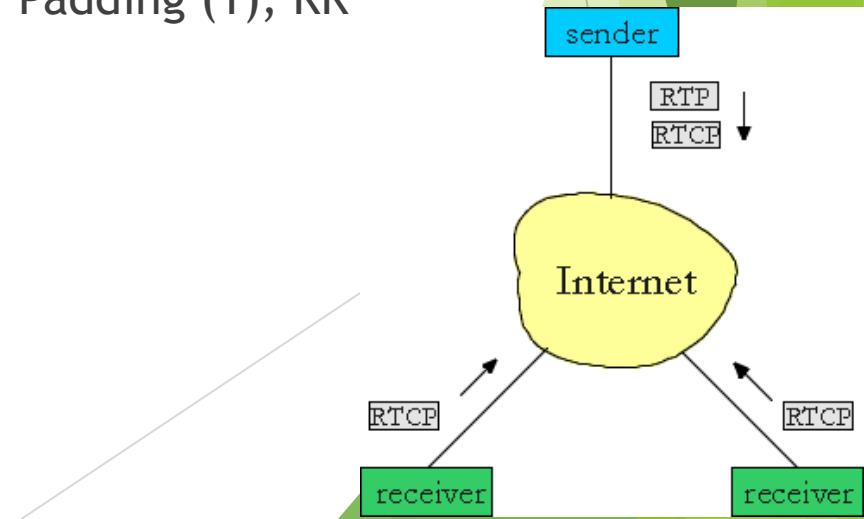
- ▶ Synchronization Source (SSRC)
- ▶ Sequence number (packet count)
 - ▶ gap in sequence #: packet loss
- ▶ Timestamp (sample count)
 - ▶ e.g., audio@8KHz, 20ms samples/packet
 - ▶ timestamp increment per packet: 160
 - ▶ 90KHz used for video
 - ▶ a video frame may be encapsulated in a few packets
 - ▶ gap in timestamp: silence

RTP Data Transfer Protocol (3): RTP profiles

- ▶ Media specific (e.g., audio)
 - ▶ Marker: e.g., the start of a talk spurt
 - ▶ Payload Type: e.g., specific audio codec
 - ▶ PT=0: uPCM 64Kbps; PT=3: GSM 13Kbps
 - ▶ timestamp: e.g., sampling rate, 8KHz PCM
 - ▶ packet size: e.g., about 20ms samples in PCM
 - ▶ packets independent as much as possible: ALF
 - ▶ other issues: e.g., mixed audio channels

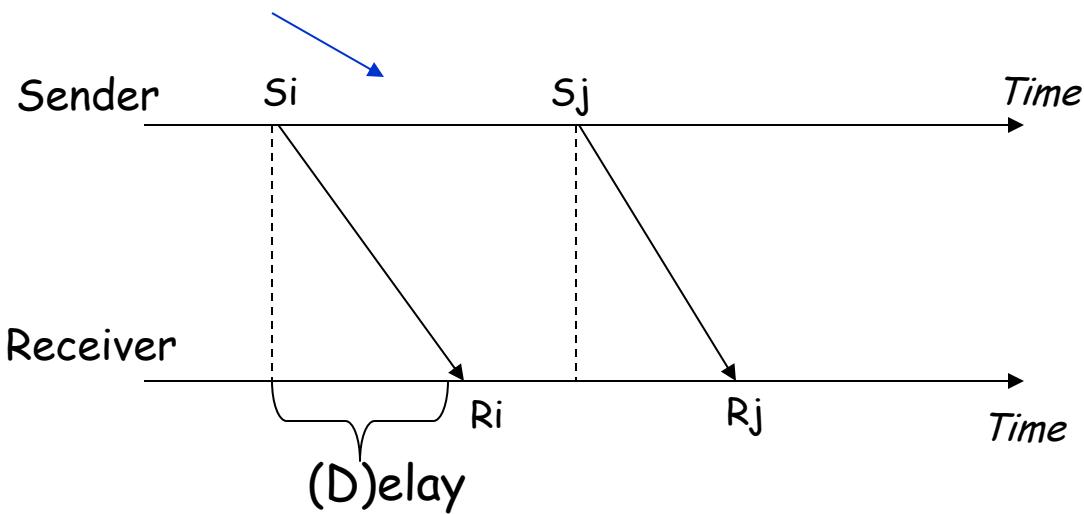
Real-Time Control Protocol (RTCP)

- ▶ RTCP: RTP's control companion
 - ▶ Purpose: feedback control information
 - ▶ for flow/error/congestion/quality control
 - ▶ Two consecutive UDP ports for RTP and RTCP
 - ▶ Sender report: offer sending/reception statistics
 - ▶ Receiver report: offer reception statistics
- ▶ RTCP common header: Version (2-bit); Padding (1); RR count (5); Type (8); Length (16)



Real-Time Control Protocol (RTCP)

- ▶ An integral part of RTP
- ▶ Current properties of the RTP packet stream, e.g. jitter, packet loss
→ QoS monitoring

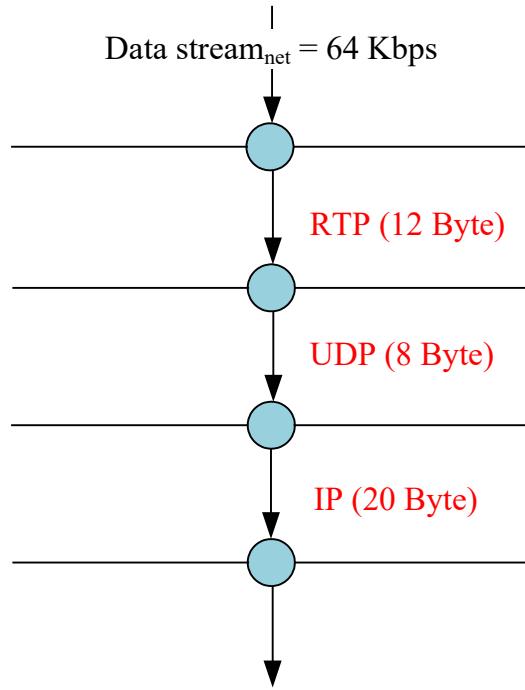


- **Inter-arrival time ($D(i,j)$)** is the time difference between the arrival times of two consecutive packets
- **Jitter ($J(i)$)** is defined as a statistical variance of the RTP data packet inter-arrival time

$$D(i,j) = (R_j - R_i) - (S_j - S_i) = (R_j - S_j) - (R_i - S_i)$$

$$J(i) = J(i-1) + (|D(i-1,i)| - J(i-1))/16$$

Typical VoIP Traffic Overhead



$$f_c = 4 \text{ kHz}$$

$$f_s = 8 \text{ kHz}$$

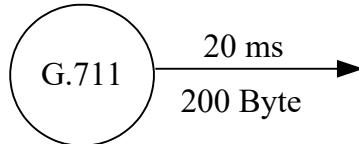
Coding: A-law à 8 bit per sample value

$$\Rightarrow \text{Data stream}_{\text{net}} = 64 \text{ kbps}$$

\Rightarrow Packetization time: 20 ms

\Rightarrow Packet size = 160 Byte (*without header*)

\Rightarrow Packet size = 160 Byte + 40 Byte = 200 Byte (*including header*)



Additional overhead caused by TCP/IP protocol stack regarding VoIP

\Rightarrow The overhead of 40 Byte in relation to 160 Byte, which one would only have to transfer in the classical telephony, shows the immense overhead caused by VoIP.

Advantages of VoIP

- ▶ Reduced cost
 - ▶ Don't have to pay for line usage
- ▶ Mobility
 - ▶ Can use same identity (same number or user name) from anywhere in the world
- ▶ Other data exchange and synchronization
 - ▶ Can exchange data such as images, graphs, videos and contact info while talking

Disadvantages of VoIP

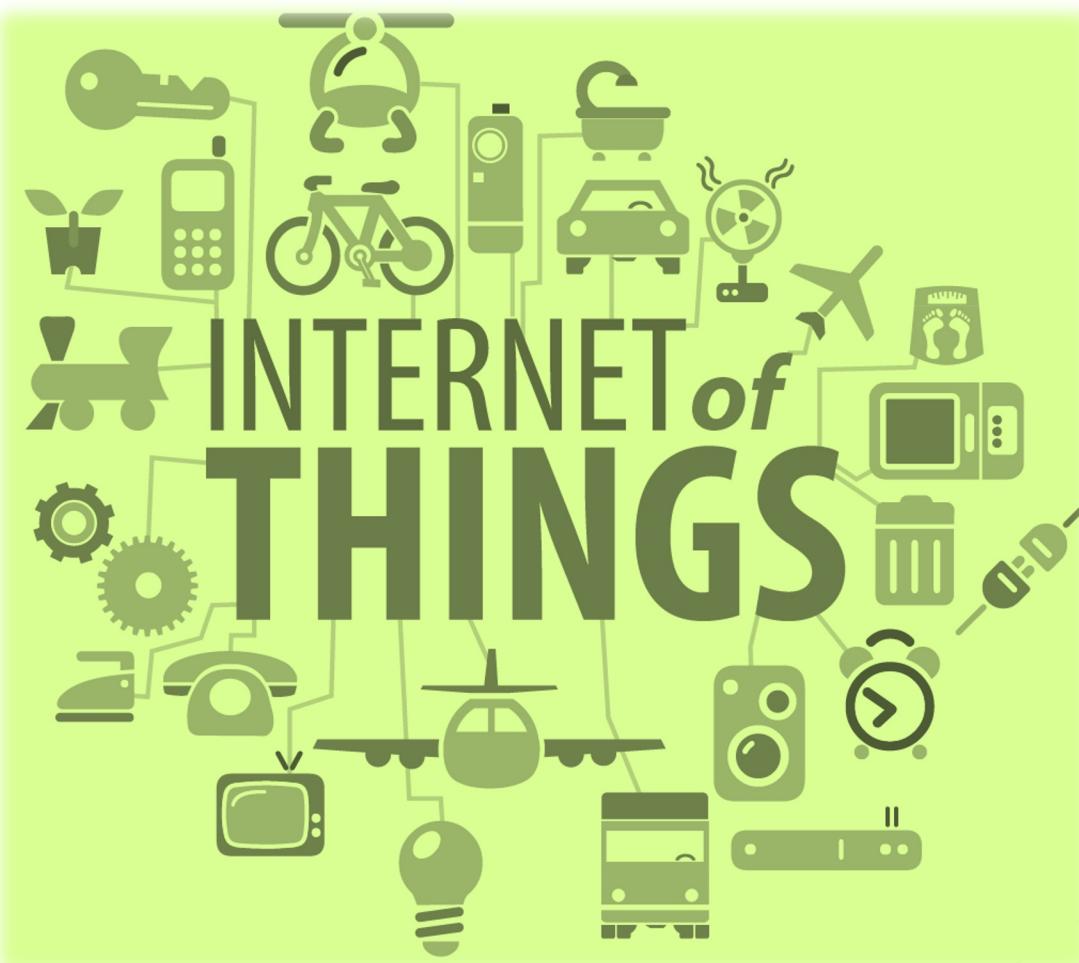
- ▶ Privacy and security etc is complicated
- ▶ Internet services don't work during power outages
 - ▶ Regular phone uses low DC current and an AC current during ring. 99% uptime even during power outage and/or natural disaster.
- ▶ Some devices such as traditional faxes most likely won't work
- ▶ Emergency calls?

Internet of Things (IoT)

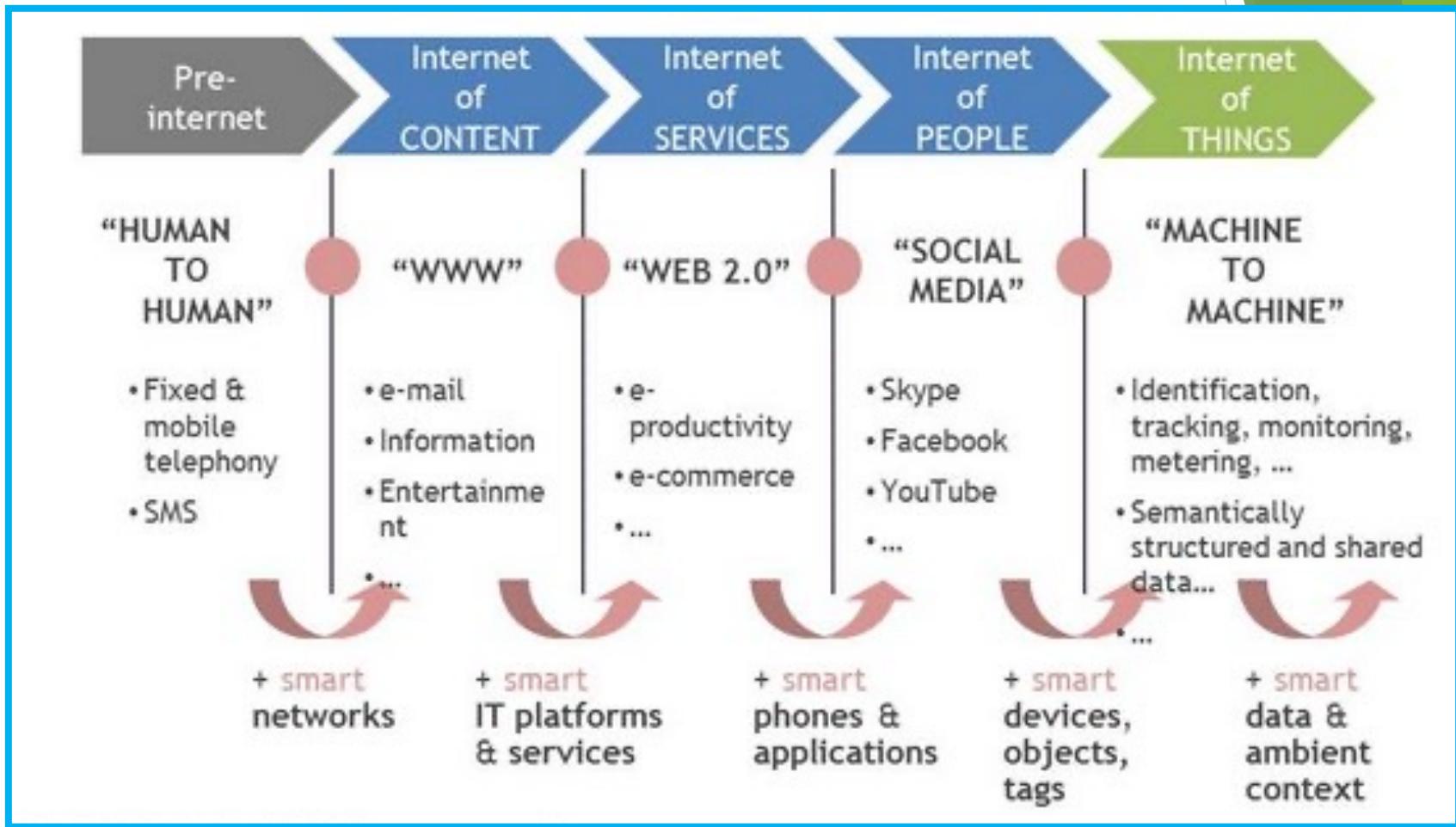
CEN 223 - Internet Communication

Assoc. Prof. Dr. Fatih ABUT
Department of Computer Engineering
Çukurova University

What is IoT?

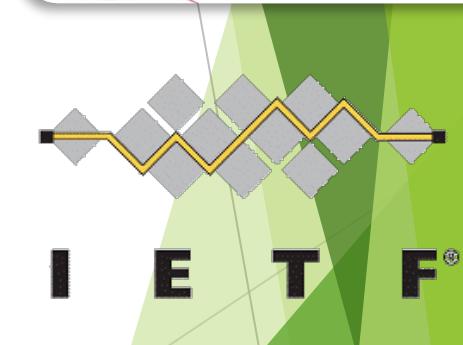


Evolution of IoT



IoT Definition

- ▶ According to EU: A global network infrastructure, linking physical and virtual objects through the exploitation of data capture and communication capabilities
- ▶ According to ITU: A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on, existing and evolving, interoperable information and communication technologies [ITU-T Y.2060]
- ▶ According to IETF: IoT is a world-wide network of interconnected objects uniquely addressable, based on standard communication protocols
- ▶ According to IEEE: A network of items each embedded with sensors which are connected to the Internet.

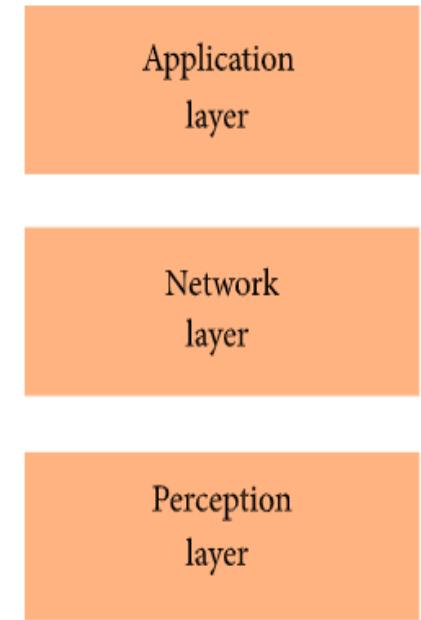


IoT is.... (According to Wikipedia)

The Internet of Things (IoT) is the network of physical devices, vehicles, home appliances, and other items embedded with electronics, software, sensors, actuators, and connectivity which enables these things to connect and exchange data, creating opportunities for more direct integration of the physical world into computer-based systems, resulting in efficiency improvements, economic benefits, and reduced human exertions.

Architecture of IoT

- The perception layer is the physical layer, which has sensors for sensing and gathering information about the environment. It senses some physical parameters or identifies other smart objects in the environment.
- The network layer is responsible for connecting to other smart things, network devices, and servers. Its features are also used for transmitting and processing sensor data.
- The application layer is responsible for delivering application specific services to the user. It defines various applications in which the Internet of Things can be deployed.

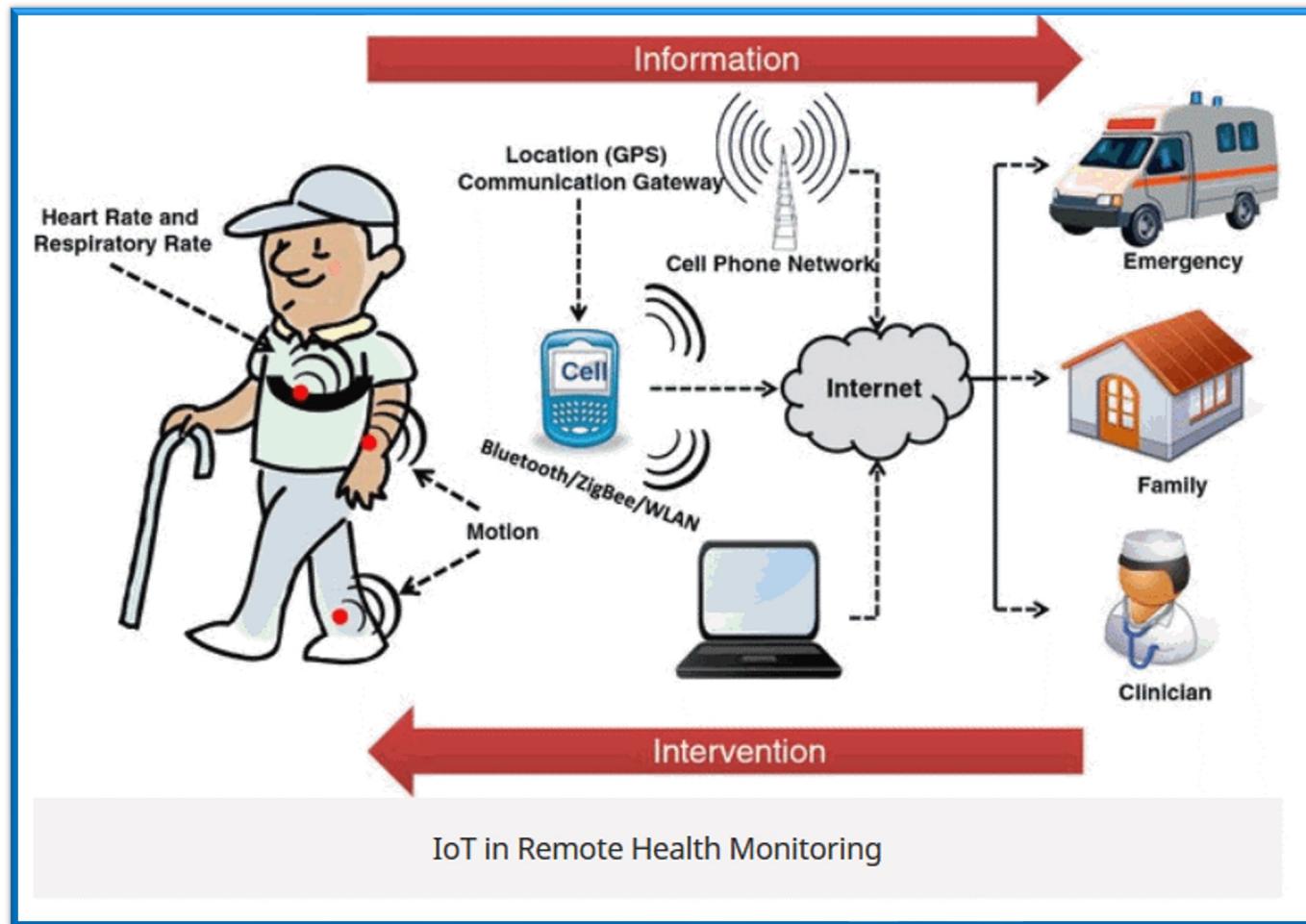


Note: The application layer is responsible for data formatting and presentation. The application layer in the Internet is typically based on HTTP. However, HTTP is not suitable in resource constrained environments. Many alternate protocols have been developed for IoT environments such as CoAP (Constrained Application Protocol) and MQTT (Message Queue Telemetry Transport).

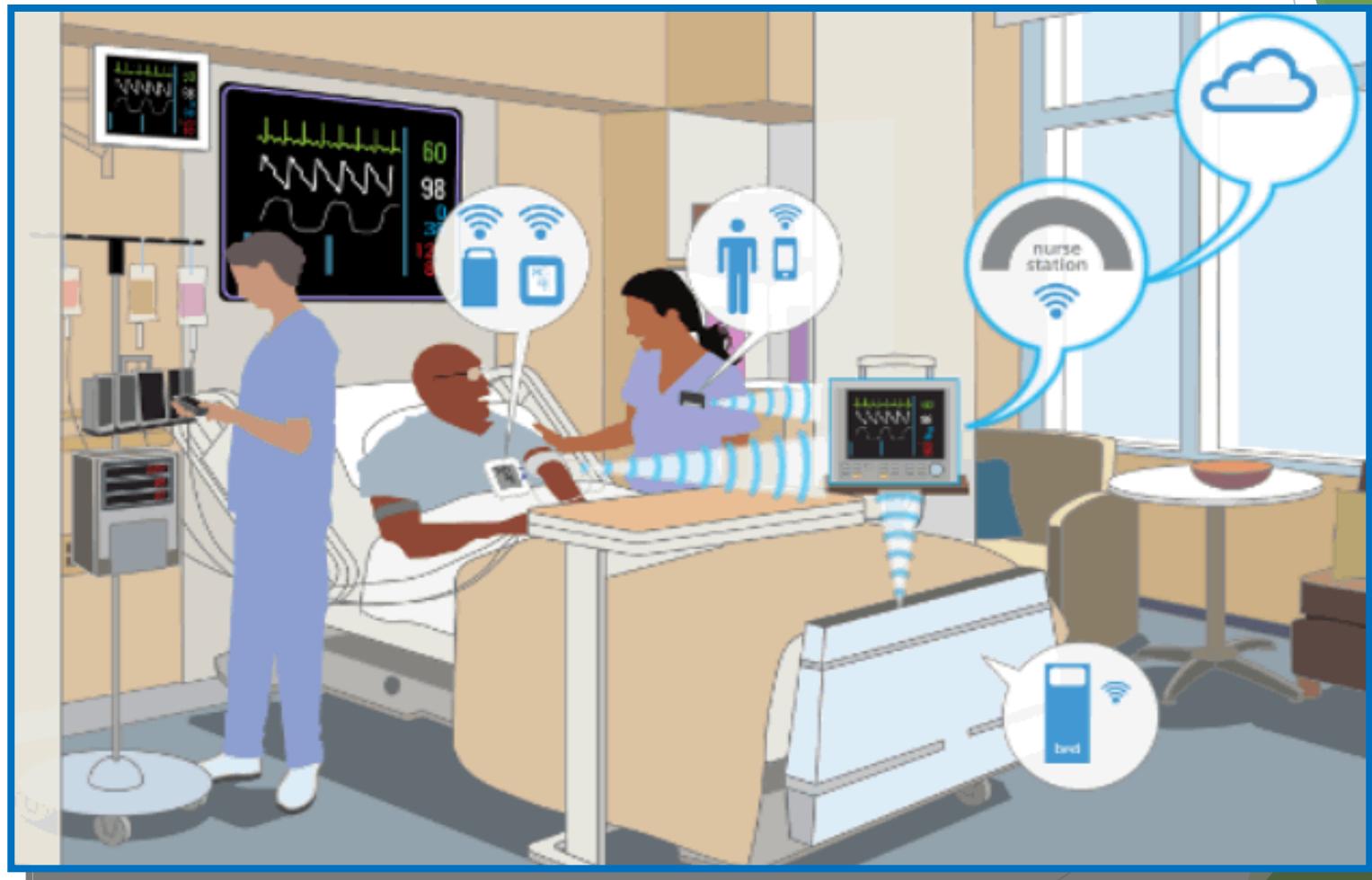
Examples of IoT Applications

- ▶ The IoT is bringing about new generation of applications such as smart
 - ▶ factories, cities, and homes,
 - ▶ grids and power plants,
 - ▶ automotive and transportation industry,
 - ▶ aerospace and aviation,
 - ▶ healthcare,
 - ▶ education and
 - ▶ agriculture, to name just a few.

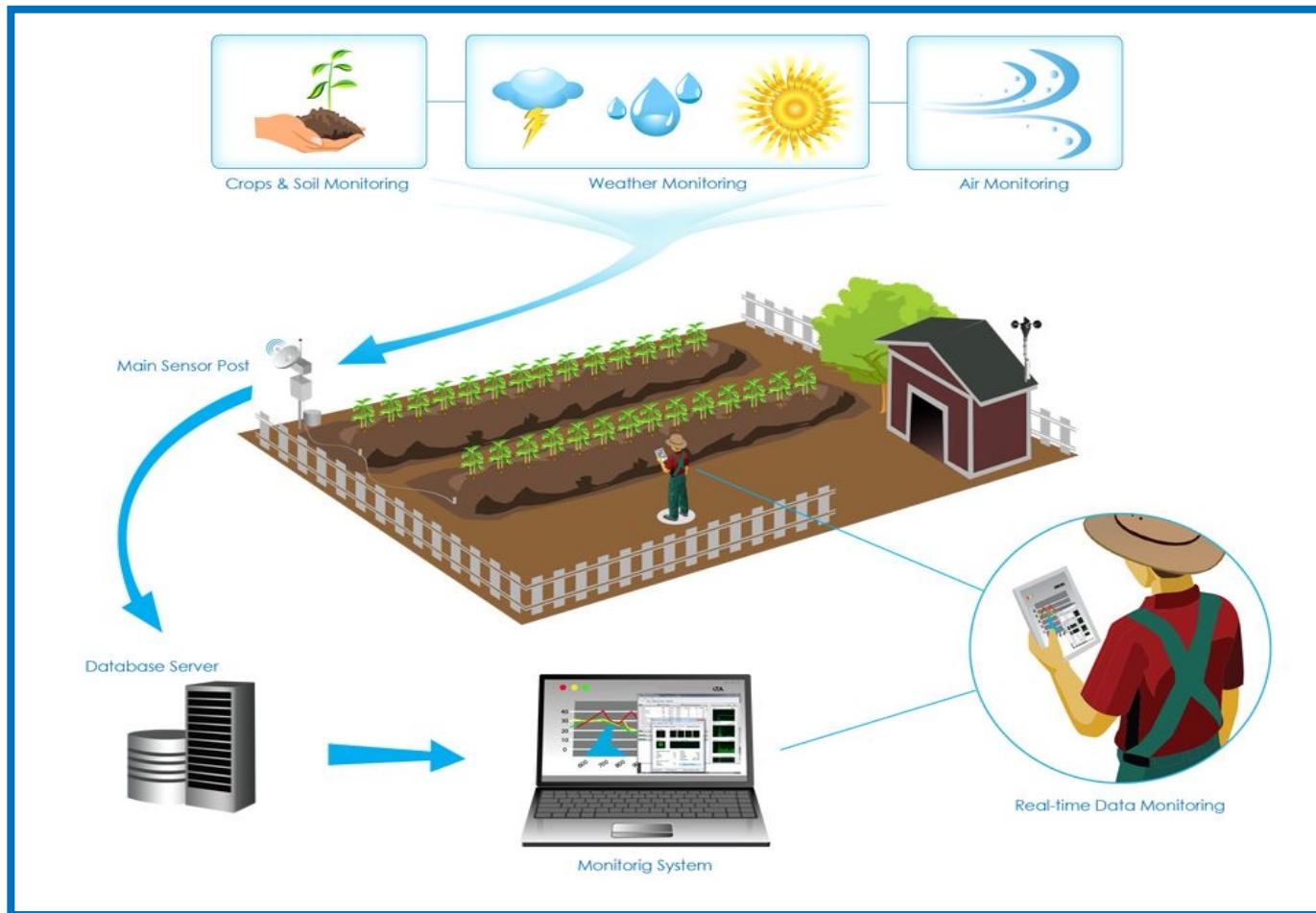
IoT in Health



IoT in Health (2)



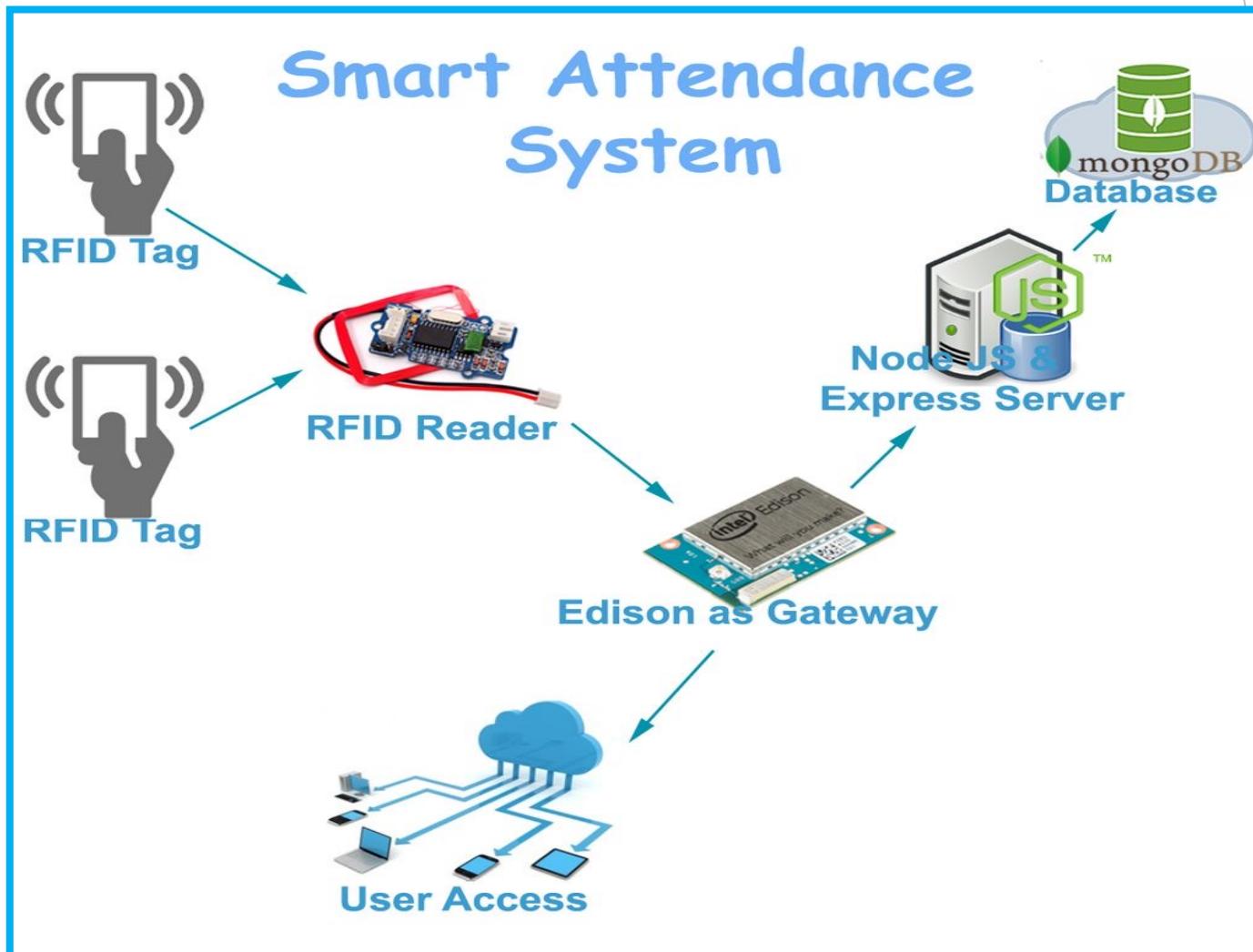
IoT in Agriculture



IoT in Agriculture (2)



IoT in Education



IoT in Education (2)

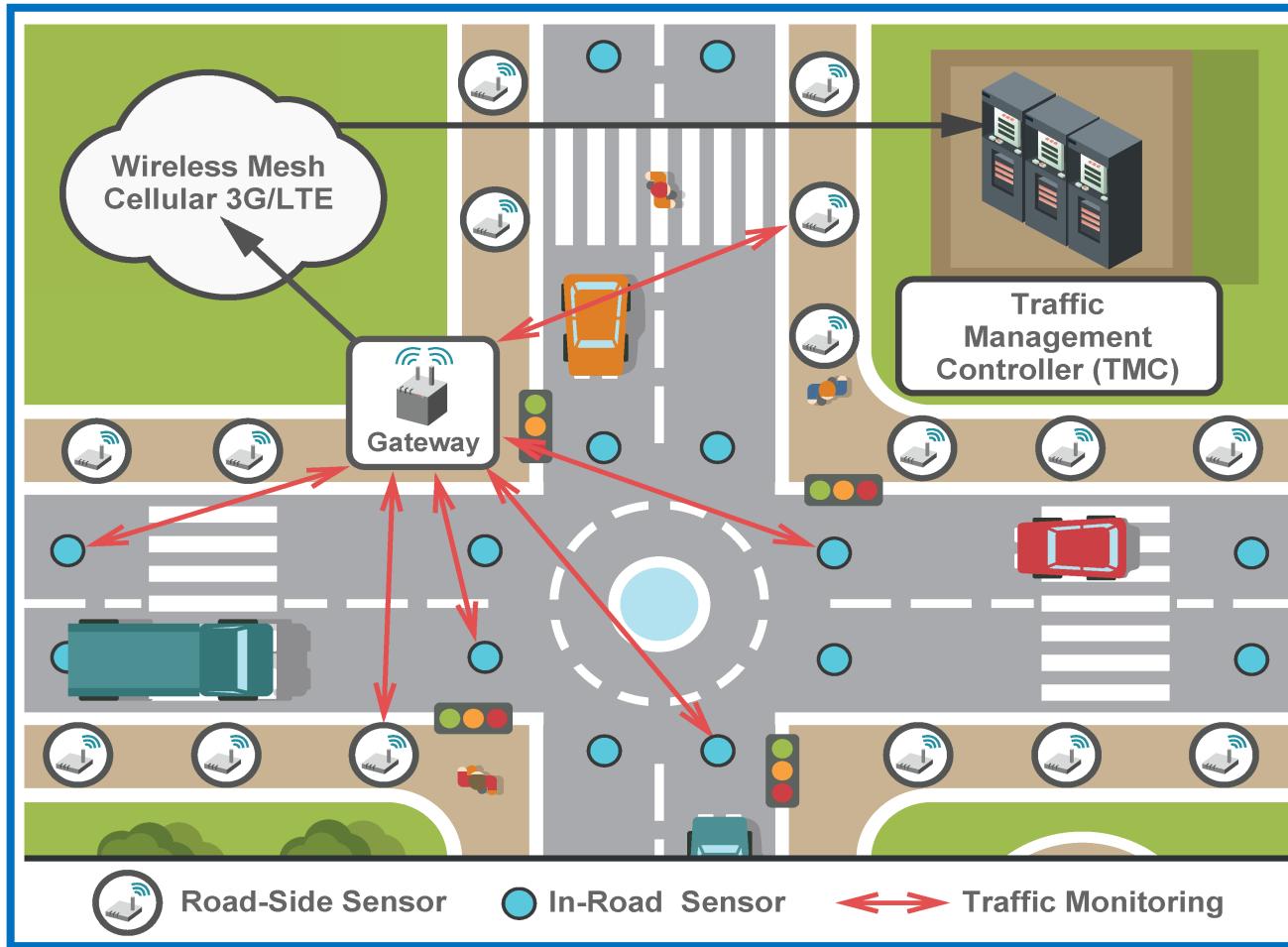
Model – III in operation



IoT in Traffic



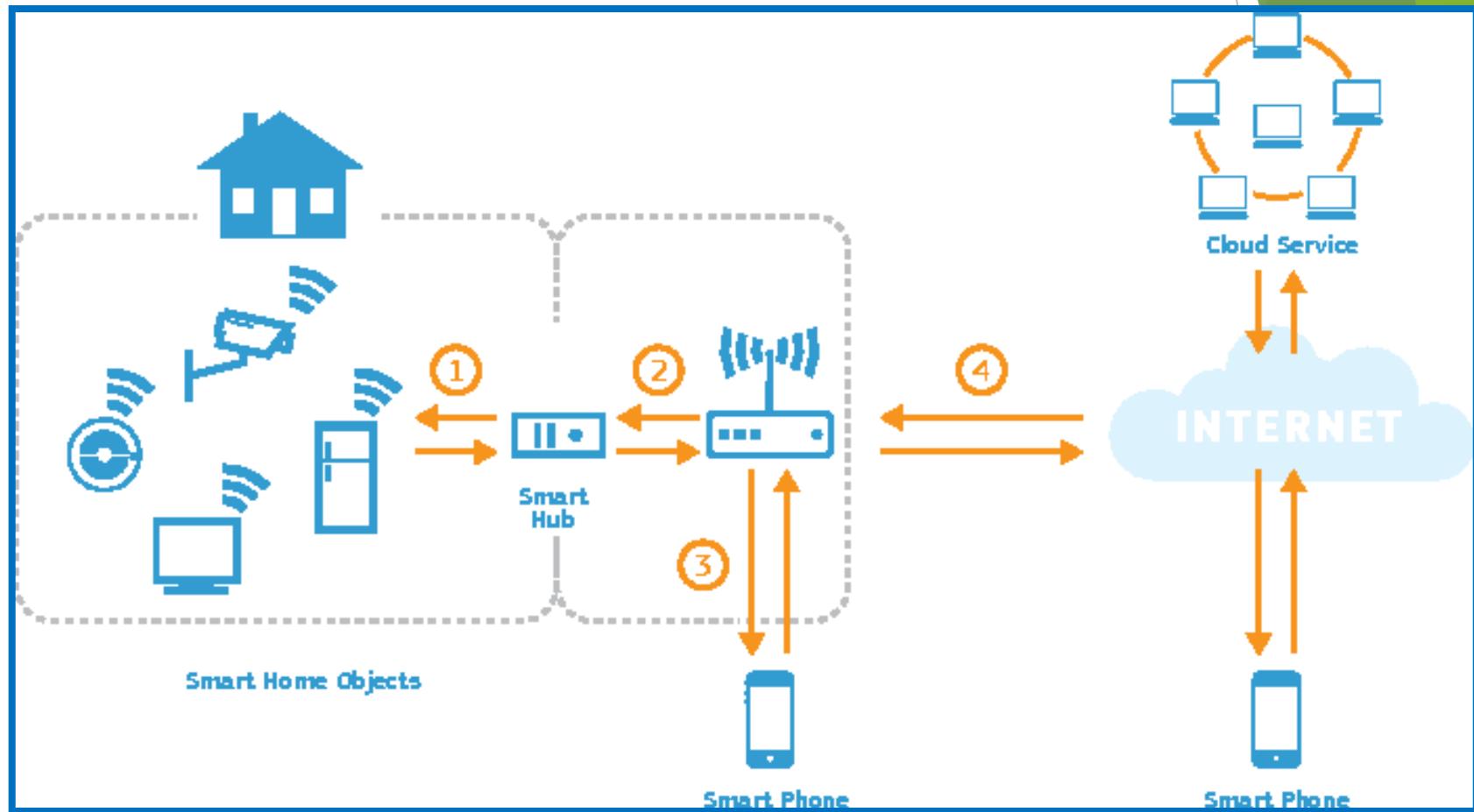
IoT in Traffic (2)



IoT in Smart Home



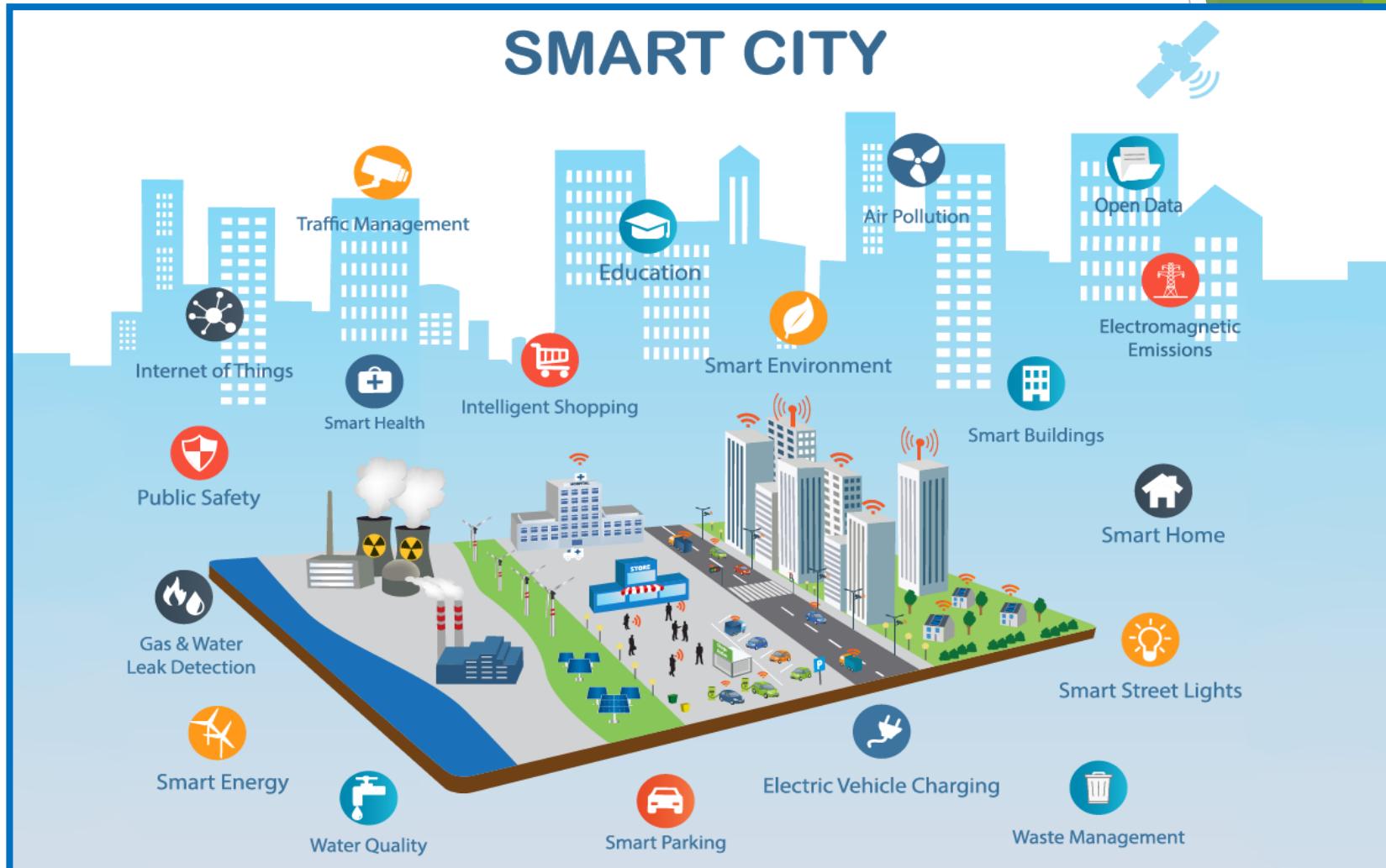
IoT in Smart Home



IoT in Retail



IoT in Smart Cities

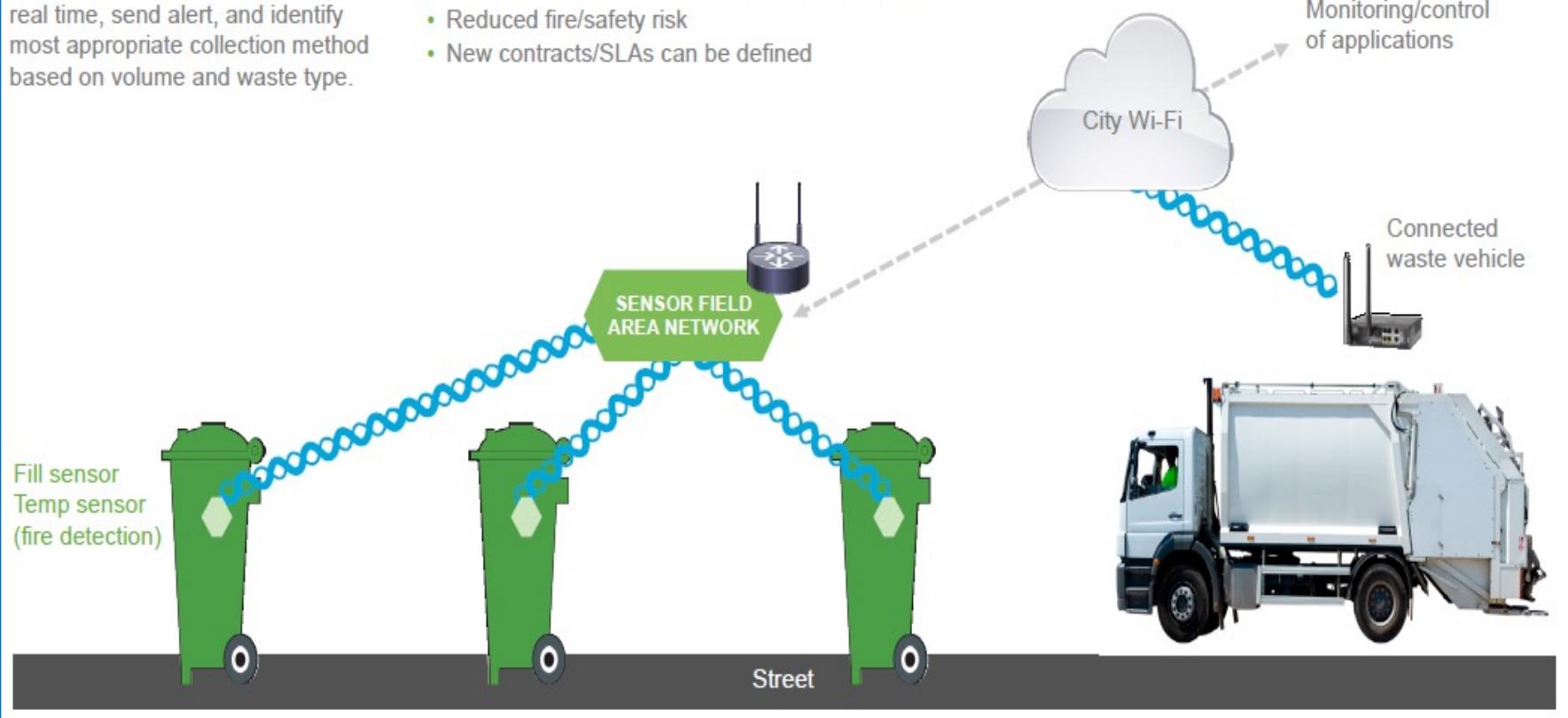


IoT Based Waste Collection

Sensors deployed in recycling containers monitor waste levels in real time, send alert, and identify most appropriate collection method based on volume and waste type.

Benefits include:

- Waste collection consumes less cost and carbon
- Reduced fire/safety risk
- New contracts/SLAs can be defined

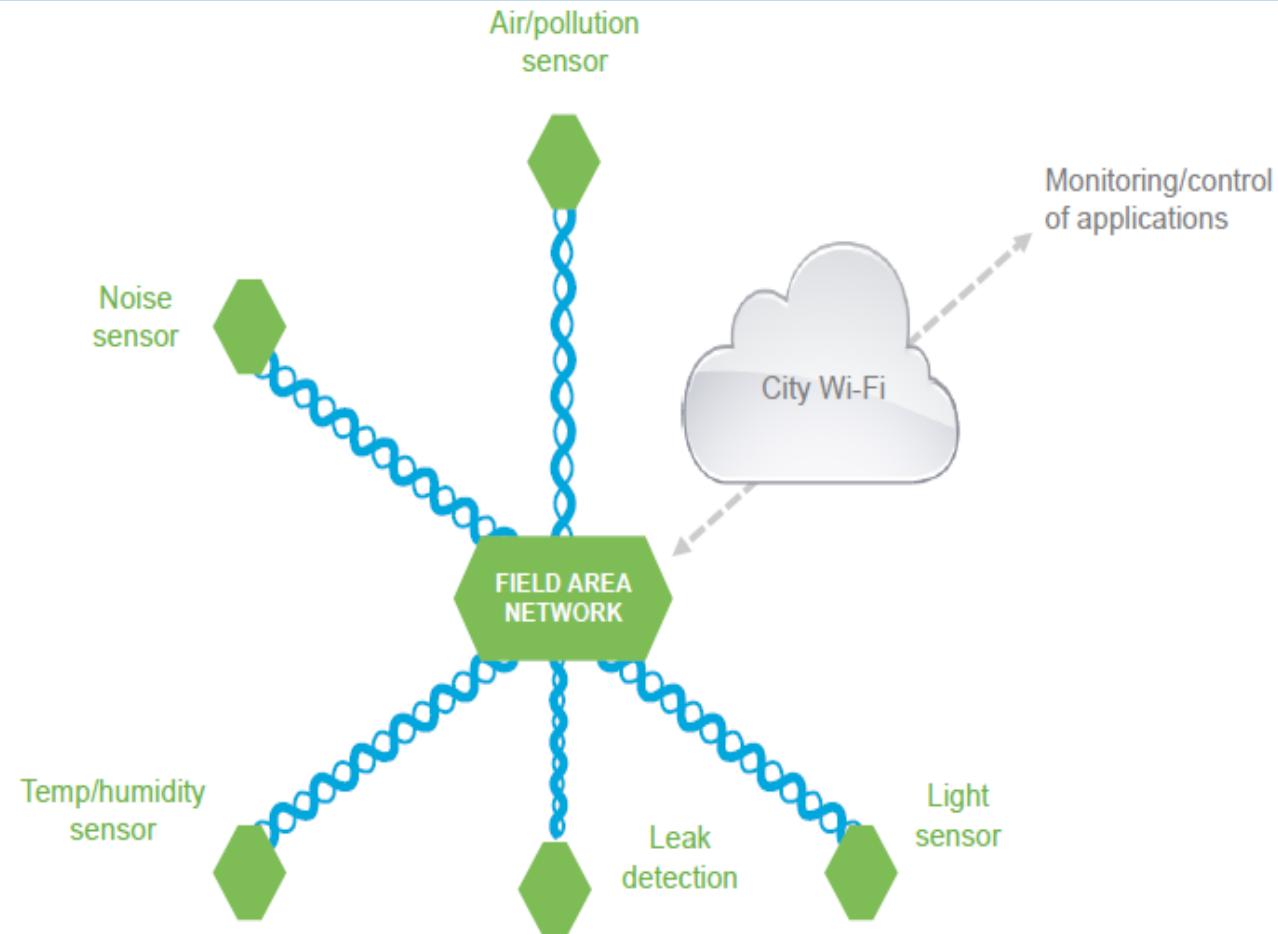


IoT Based Pollution Control

Installation of environment sensors:
air, light, humidity, noise, etc.

Benefits include:

- Leverages parking sensor infrastructure
- Provides valuable data for improving analytics applications and forecasting



IoT Application Layer Protocols

- ▶ Due to constrained
 - ▶ energy,
 - ▶ computation,
 - ▶ memory,
 - ▶ hardware resources, and
 - ▶ communication capacities

of these physical IoT devices, a new generation of protocols and algorithms are being developed and standardized

- ▶ CoAP with default CoAP CC
- ▶ CoCoA

Why Not HTTP?

- ▶ Because HTTP requires connection as it runs on TCP, and HTTP is document centric.
- ▶ But in IoT all we want to send is small data, not documents.
 - ▶ HTTP generates a problem which is low data rate and high energy consumption (3-step handshake),
 - ▶ while CoAP is a connectionless as it runs on UDP, and it is meant for M2M communication by design.

CoAP (RFC 7252)

- ▶ The Constrained Application Protocol (CoAP) has been designed by the Internet Engineering Task Force (IETF) for the needs of IoT application layer communication.
- ▶ The CoAP is a specialized web transfer protocol for use with these constrained physical devices.
- ▶ There are various types of CoAP Messages
 - ▶ Confirmable Message (CON)
 - ▶ Non-confirmable Message (NCON)
 - ▶ Acknowledgement Message (ACK)
 - ▶ Reset Message (RST)

Some CoAP Message Types

- ▶ Confirmable Message (CON)
 - ▶ Some messages require an acknowledgement. These messages are called "Confirmable". When no packets are lost, each Confirmable message elicits exactly one return message of type Acknowledgement or type Reset.
- ▶ Non-confirmable Message (NCON)
 - ▶ Some other messages do not require an acknowledgement.
- ▶ Acknowledgement Message (ACK)
 - ▶ An Acknowledgement message acknowledges that a specific Confirmable message arrived.

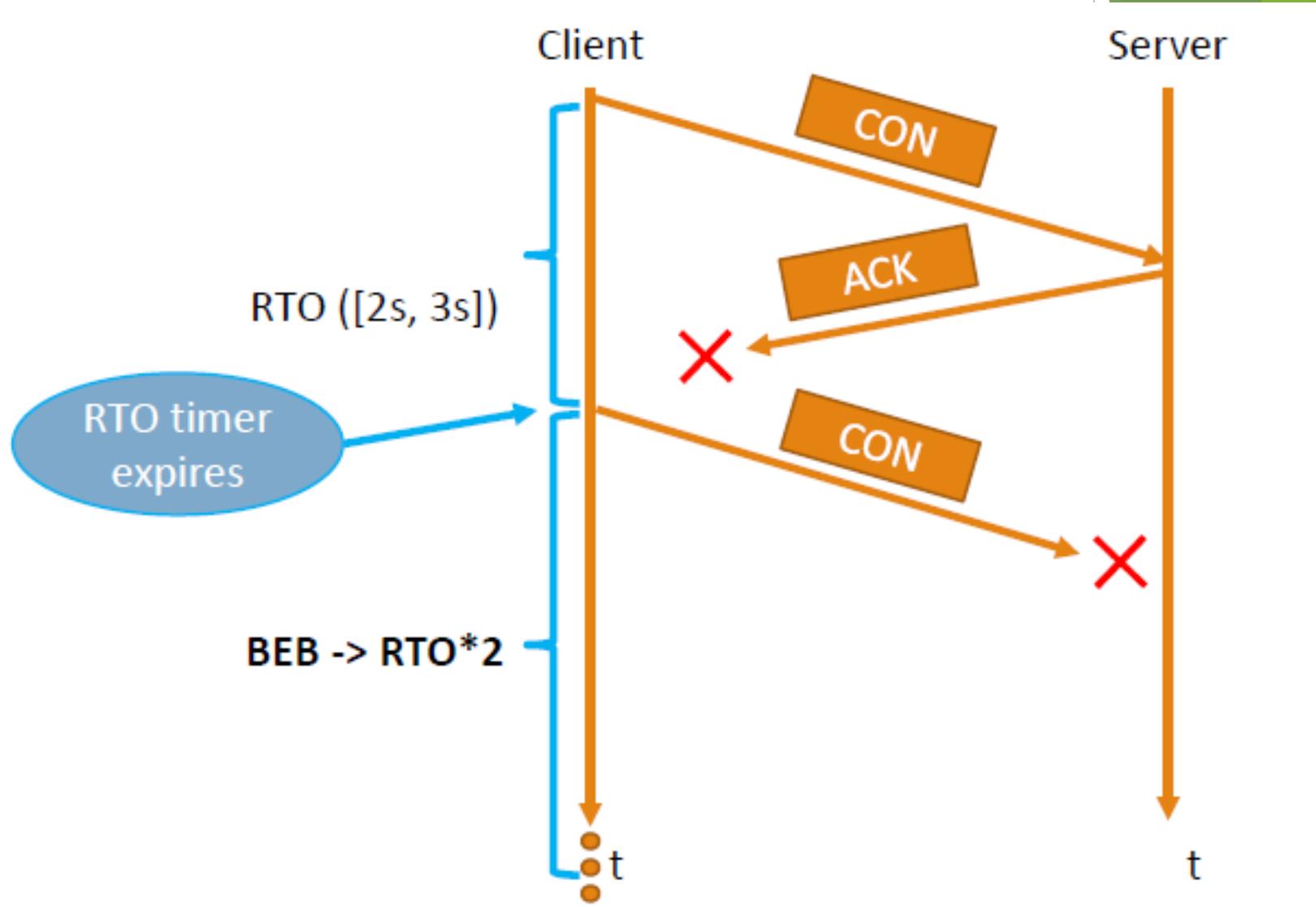
Congestion in IoT networks

- ▶ CoAP is based on UDP to better fit the requirements of constrained physical devices.
 - ▶ UDP is a very simple and light-weight transport layer protocol that does not handle congestion within the network.
- ▶ The phenomenon of congestion in IoT networks is also a major problem.
 - ▶ When the queuing and storing capacities of physical devices forming the IoT network are exceeded or generated traffic within IoT network gets close to the network capacity, network congestion is inevitably observed.
 - ▶ Typical effect of network congestion results with queuing delay and packet loss.
 - ▶ Congestion decreases the network utilization. This is known as congestive collapse.
 - ▶ Congestion control and avoidance (CCA) mechanisms are required to avoid congestive collapse.

Default CoAP Congestion Control (CC)

- ▶ Thus, the core CoAP specification offers a basic conservative default CoAP congestion control (CC) mechanism based on retransmission timeout (RTO) with binary exponential backoff (BEB).
- ▶ The sender retransmits the Confirmable message at exponentially increasing intervals, until it receives an acknowledgement (or Reset message) or runs out of attempts.
[RFC7252]

Default CoAP CC (2)



Default CoAP CC

► CONSERVATIVE

- Default CoAP CC is insensitive to network conditions.
- «Further congestion control optimizations and considerations are expected in the future, ...»

[RFC7252]

CoAP Congestion Control Advanced (CoCoA)

- ▶ For each destination endpoint, CoCoA maintains two RTO estimators:
 - ▶ The strong RTO estimator gathers RTT information (RTT_{strong}) by measuring only when no retransmissions occurred.
 - ▶ The weak RTO estimator, which takes RTT values from retransmitted requests (RTT_{weak}) using the time between sending the initial request and obtaining the reply.

CoCoA (2)

- ▶ The following formulas apply when obtaining a new RTT measurement (RTT_{X_new}), where RTTVar is the round-trip time variation and X stands for strong or weak accordingly:
- ▶ $RTTVar_X = (1 - \beta) \times RTTVar_X + \beta \times |RTT_X - RTT_{X_new}|$
- ▶ $RTT_X = (1 - \alpha) \times RTT_X + \alpha \times RTT_{X_new}$
 $\beta = 1/8 \quad , \alpha = 1/4$
- ▶ $RTO_X = RTT_X + K_X \times RTTVar_X$
 $K_{strong} = 4, K_{weak} = 1$
- ▶ $RTO_{overall} = \gamma_X \times RTO_X + (1 - \gamma_X) \times RTO_{overall}$
 $\gamma_{strong} = 0.5, \gamma_{weak} = 0.25$

CoCoA (3)

- ▶ $RTO_{overall}$ is then used to determine the initial RTO (RTO_{init}) of CON transmission

VARIABLE BACKOFF FACTOR (VBF)

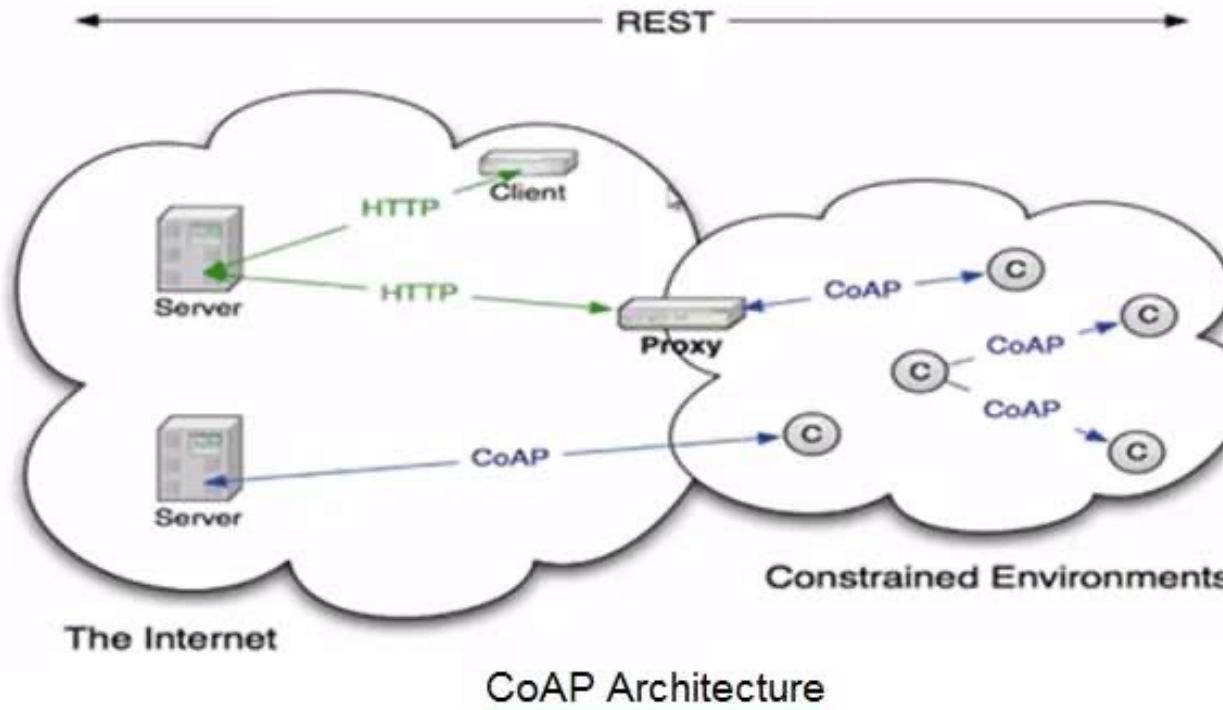
- ▶ The introduction of the VBF is an important change to the backoff behavior used by default CoAP CC and CoCoA, replacing the BEB applied to retransmissions. Instead of doubling the previous RTO value ($RTO_{previous}$) to obtain the RTO applied to the next retransmission (RTO_{new}), it is multiplied by a variable factor:

$$\blacktriangleright \quad RTO_{new} = RTO_{previous} \times VBF$$

$$VBF(RTO_{init}) = \begin{cases} 3, & RTO_{init} < 1s \\ 2, & 1s \leq RTO_{init} \leq 3s \\ 1.5, & RTO_{init} > 3s \end{cases}$$

URI Scheme of COAP

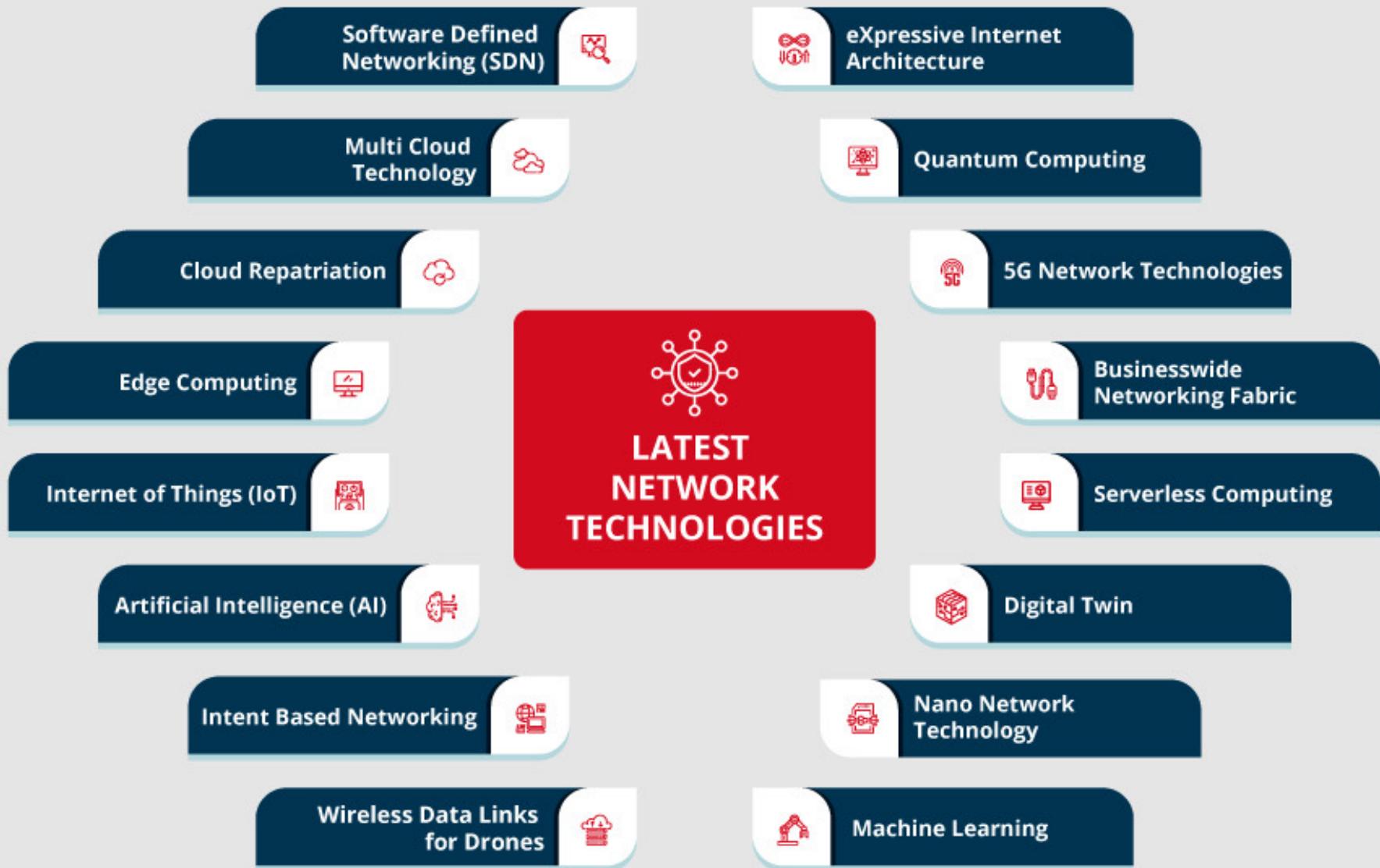
- ▶ COAP similar to HTTP. Just like URI scheme for HTTP, COAP have URI scheme also written like `coap://`.



Some Challenges of IoT

- ▶ Privacy
- ▶ Interoperability
- ▶ Security
- ▶ Quality of Service
- ▶ Data Encryption and Key Management
- ▶ Network Issues (Traffic Congestion)





Summary

CEN 223 - Internet Communication

Assoc. Prof. Dr. Fatih ABUT
Department of Computer Engineering
Çukurova University

An HTTP conversation

Client

- ▶ I would like to open a connection

Web Server

- ▶ OK

- ▶ GET <file location>

- ▶ Send page or error message

- ▶ Display response

- ▶ Close connection

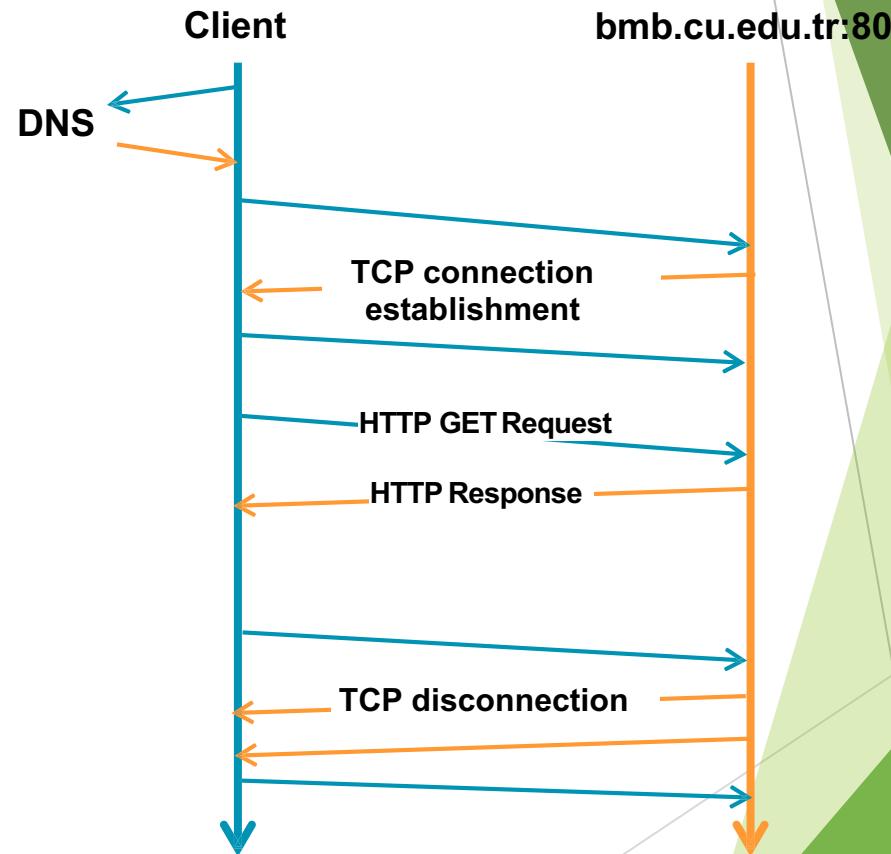
- ▶ OK

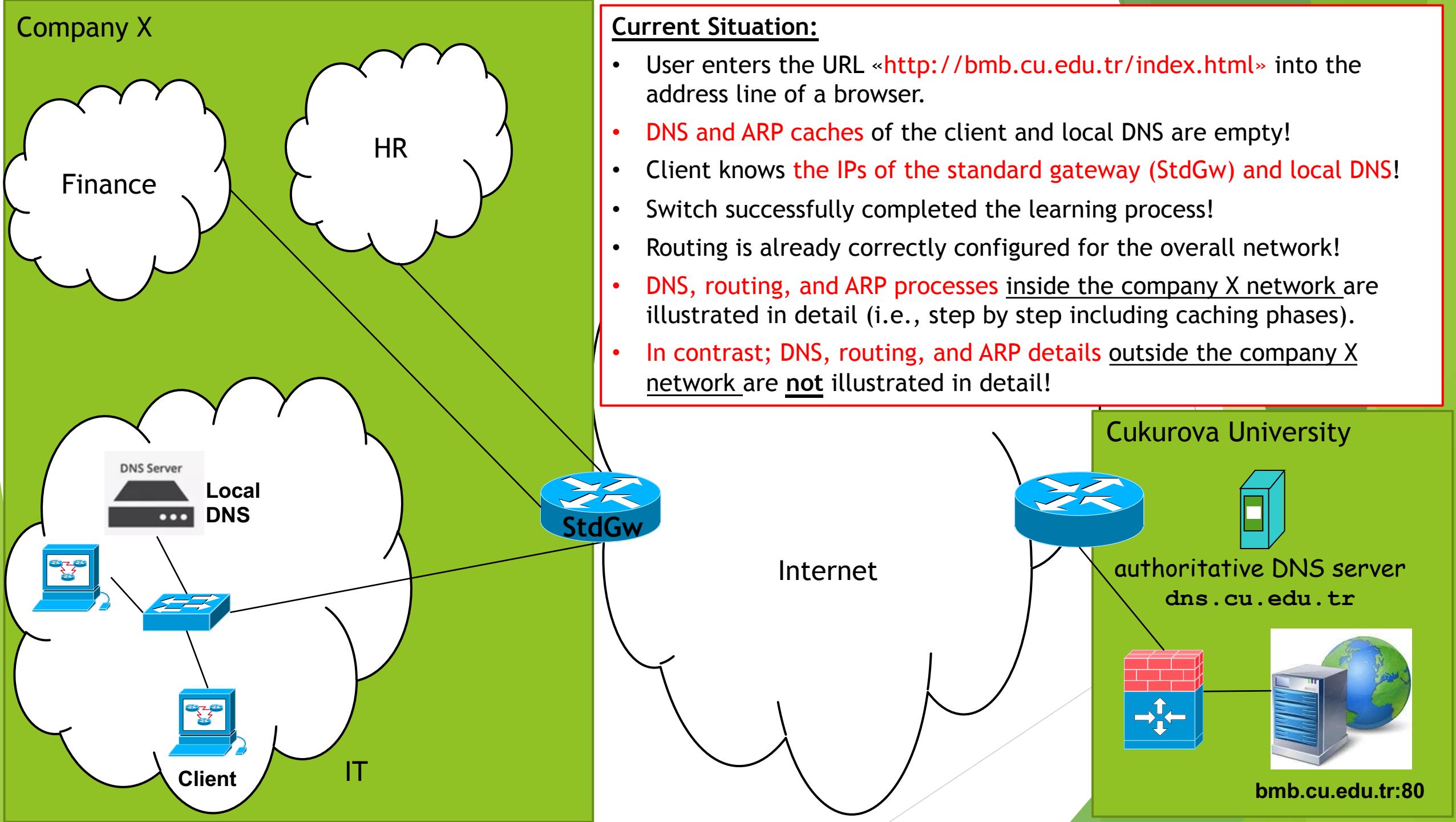
HTTP is the set of rules governing the format and content of the conversation between a Web client and server

Basic sequence of an HTTP request response

Example: Access to <http://bmb.cu.edu.tr:80/>

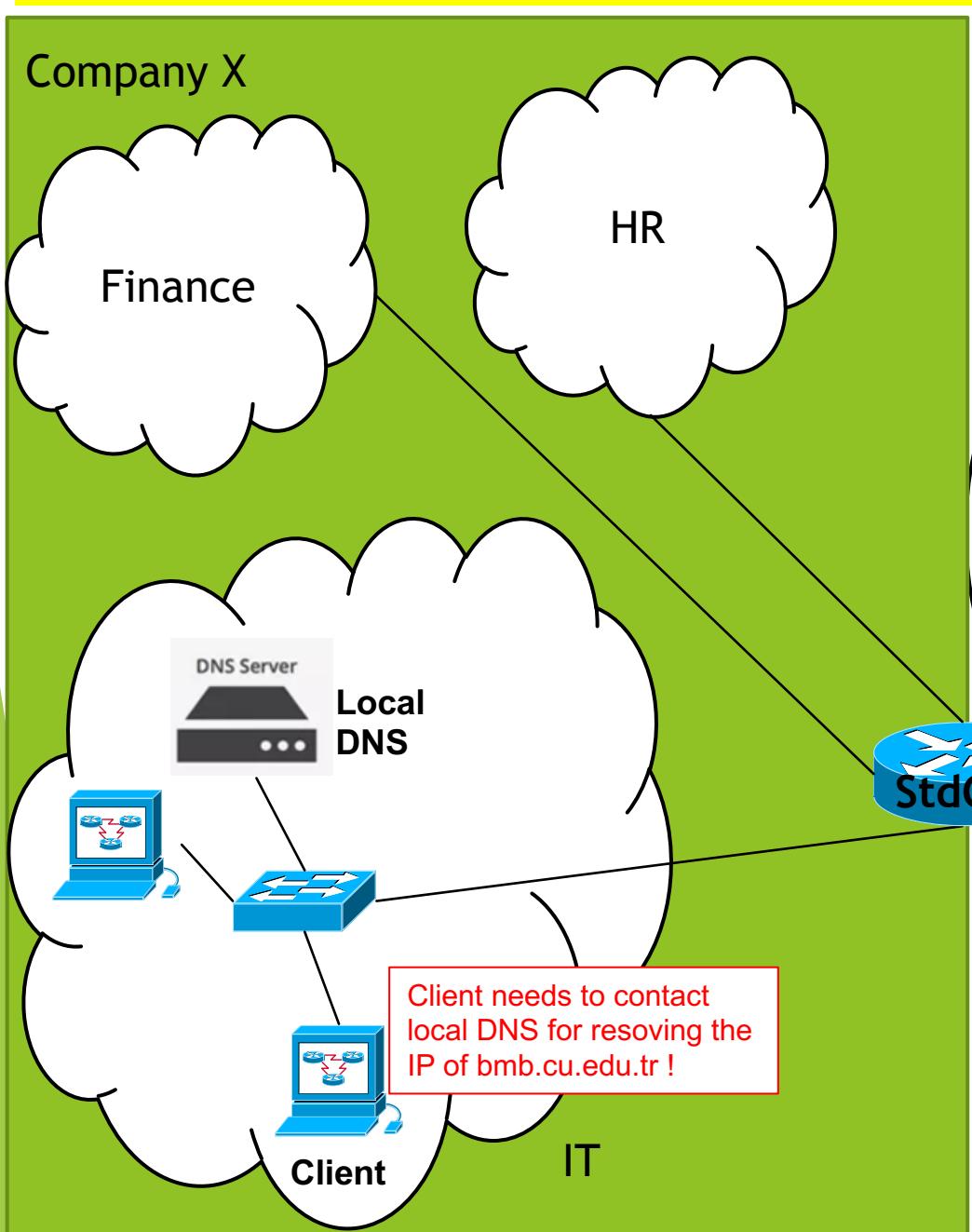
1. DNS resolution
2. Establishing a TCP connection to 193.140.54.17:80
3. Send HTTP GET request messages as bytestrome to 193.140.54.17:80.
4. Send HTTP response messages as bytestromes to the client.
5. Termination of the TCP connection by the client or server
6. Termination of the TCP connection by the server or client



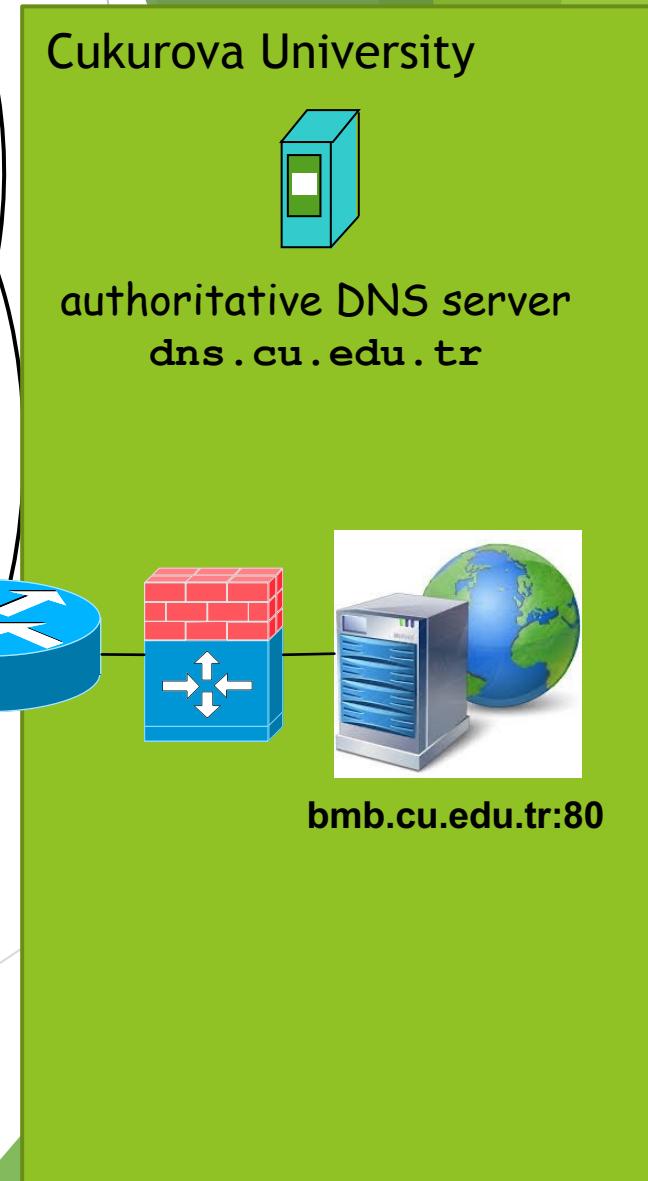


Step 1: Client wants to communicate with local DNS for resolving the IP of bmb.cu.edu.tr!

Check whether local DNS is in the same or different subnet!

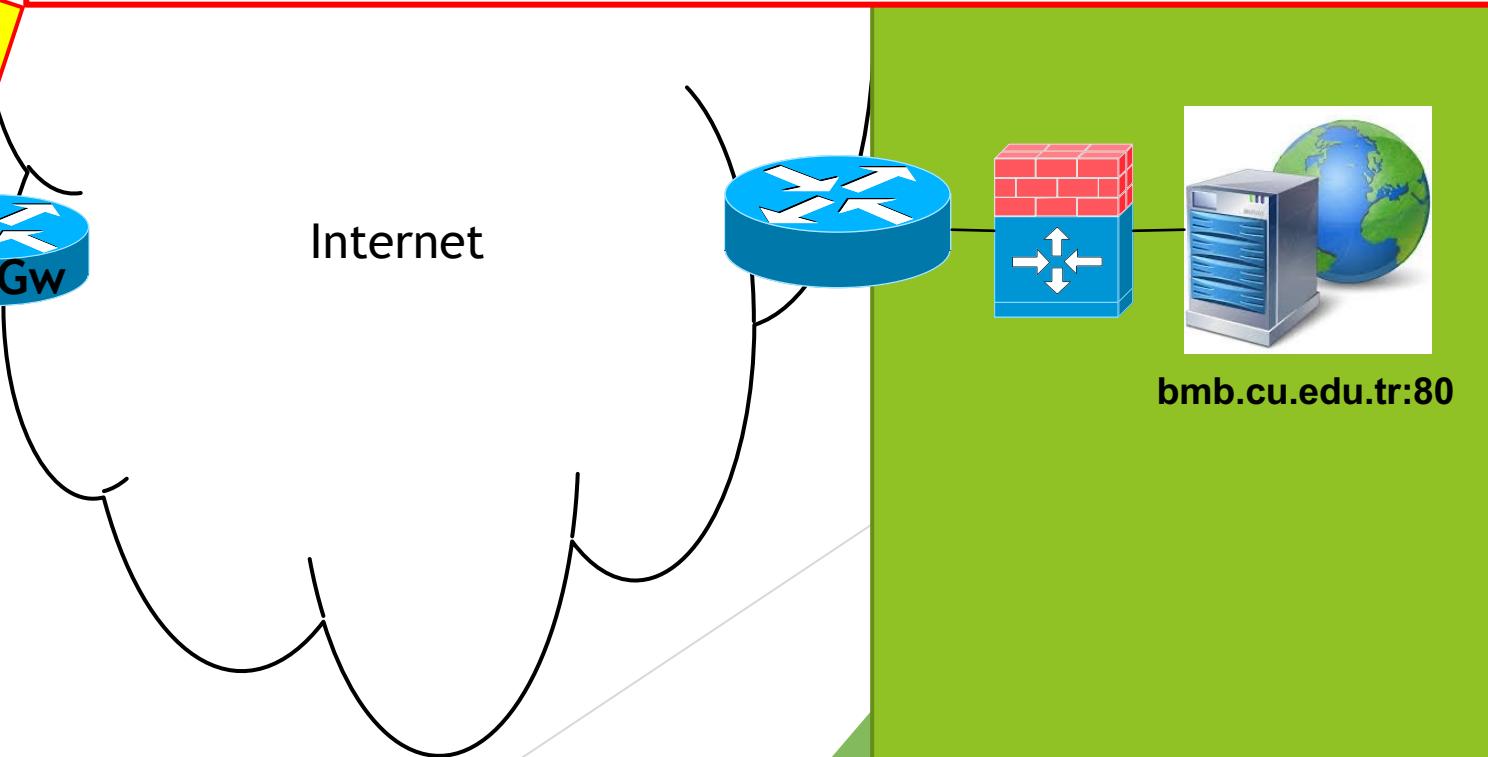
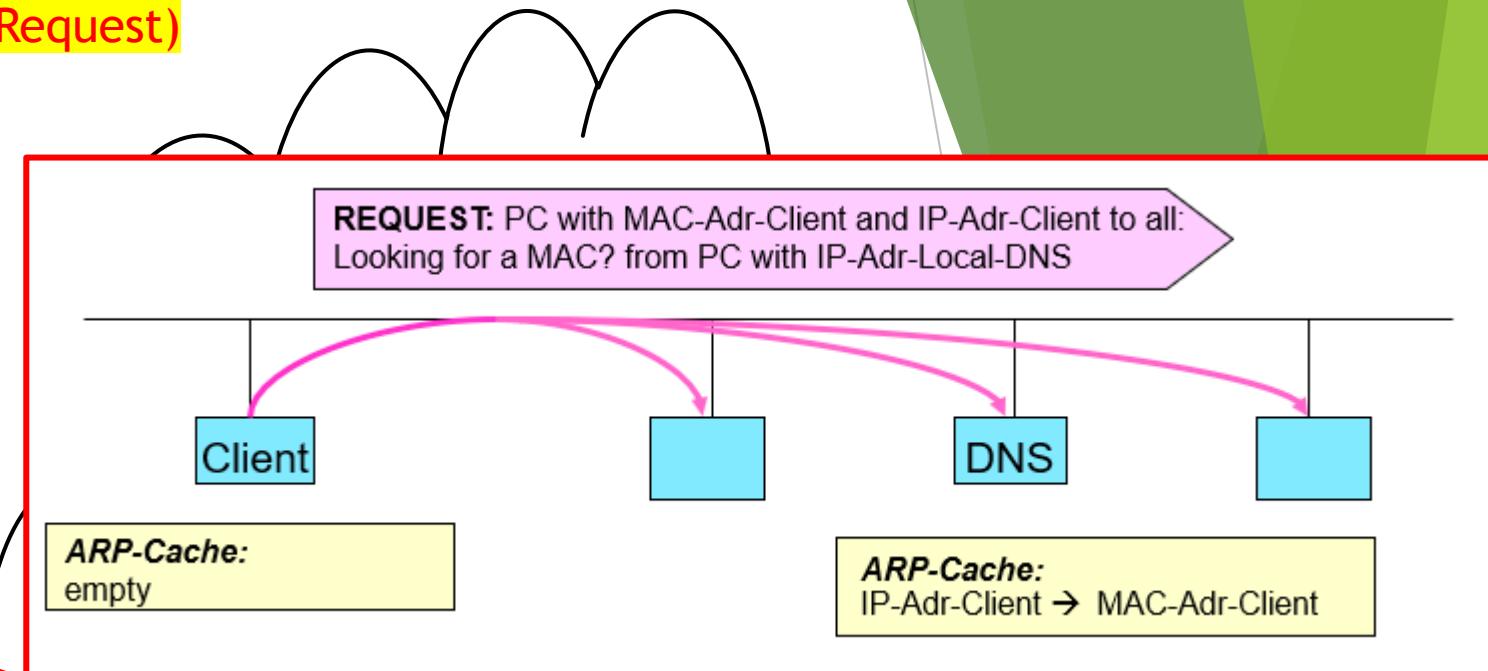
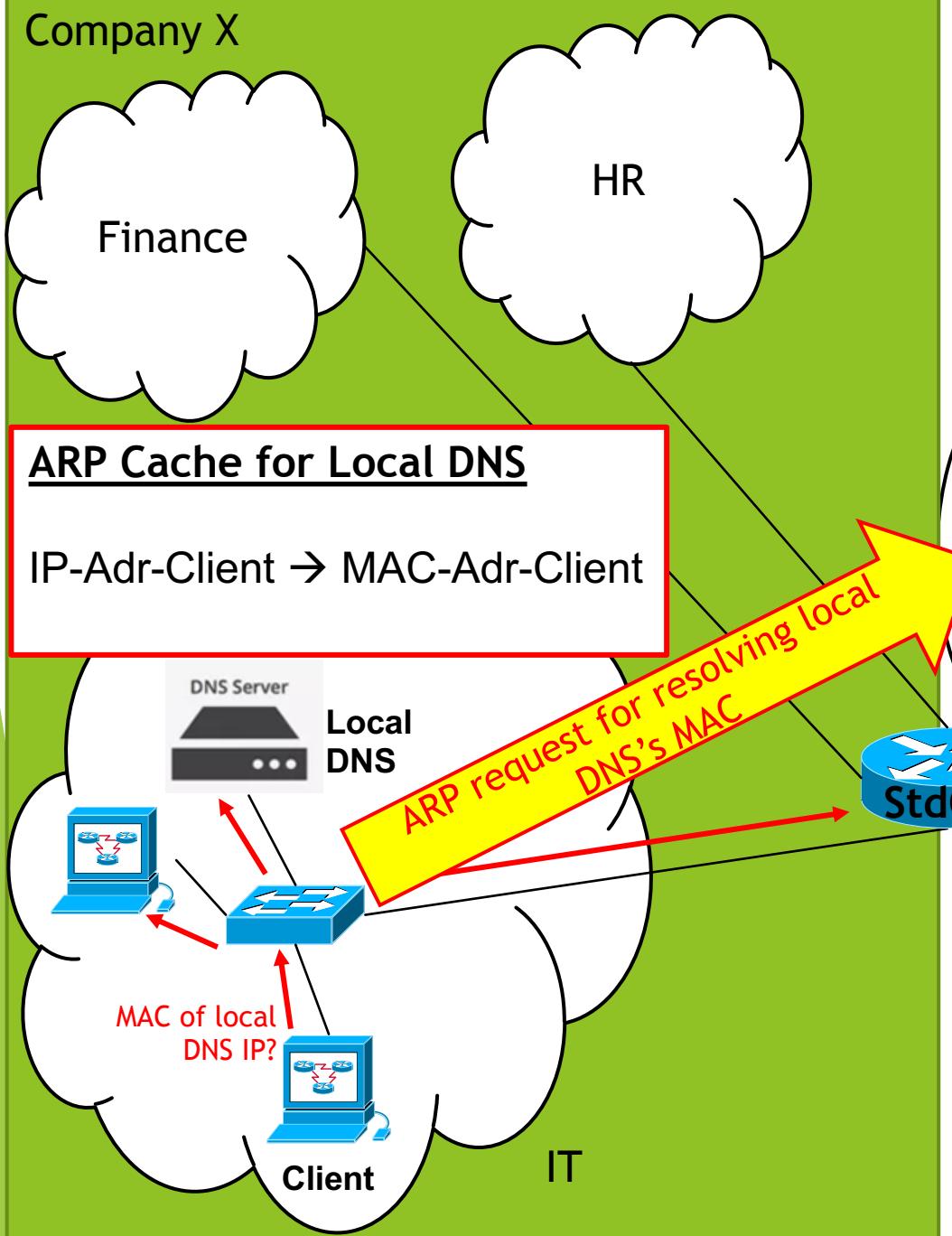


- Find out whether local DNS server is in the same subnet?
 - Applying the logical «and» operation for local DNS IP and own subnet masks
- In this case: Same subnet -> No routing is necessary
- ARP cache of client does not include the MAC of local DNS.
- Find out the MAC address of local DNS in the local subnet using ARP request (broadcast with FF:FF:FF:FF:FF)

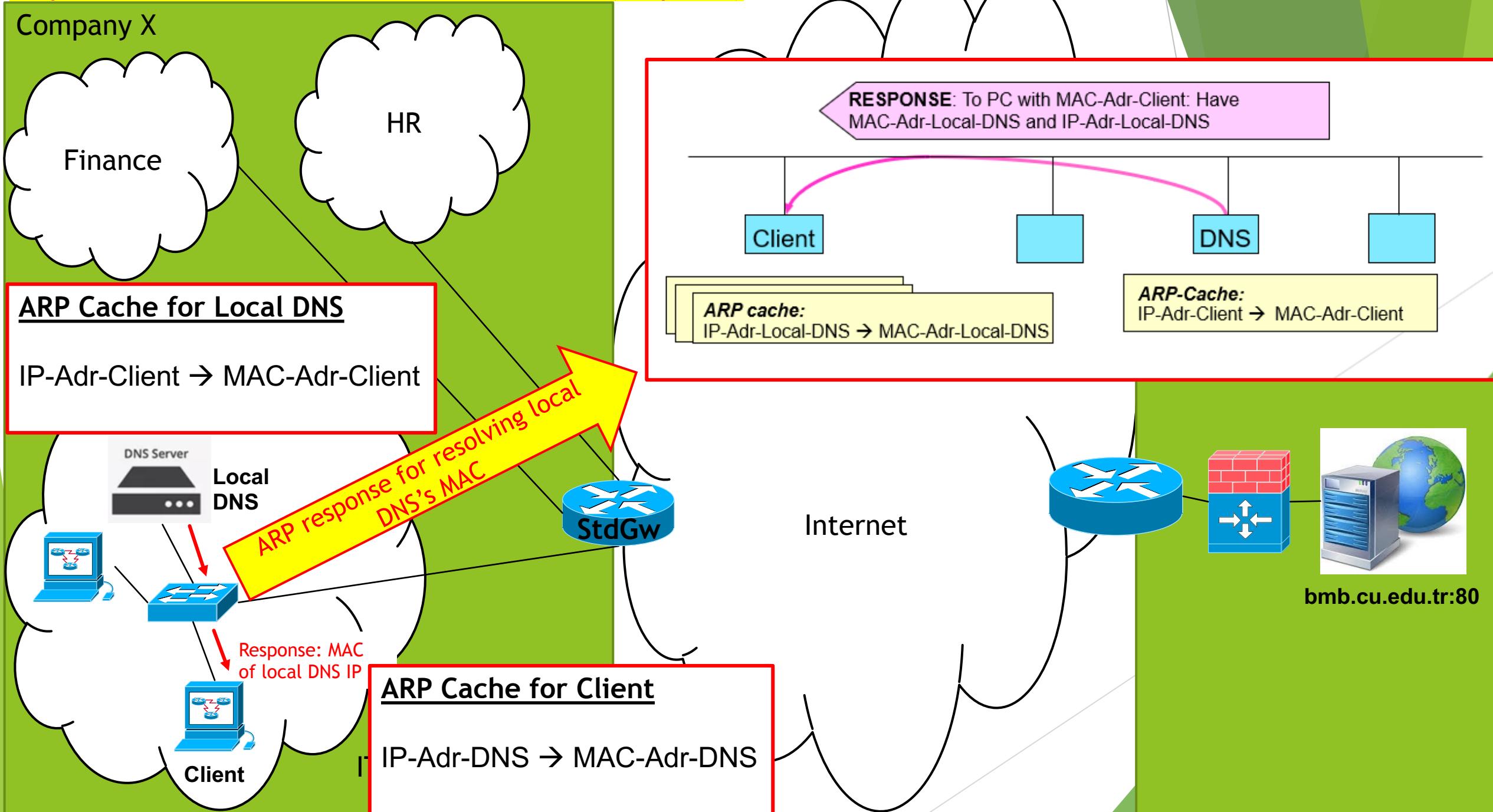


Step 2: Client resolves the MAC of local DNS IP (Request)

Company X

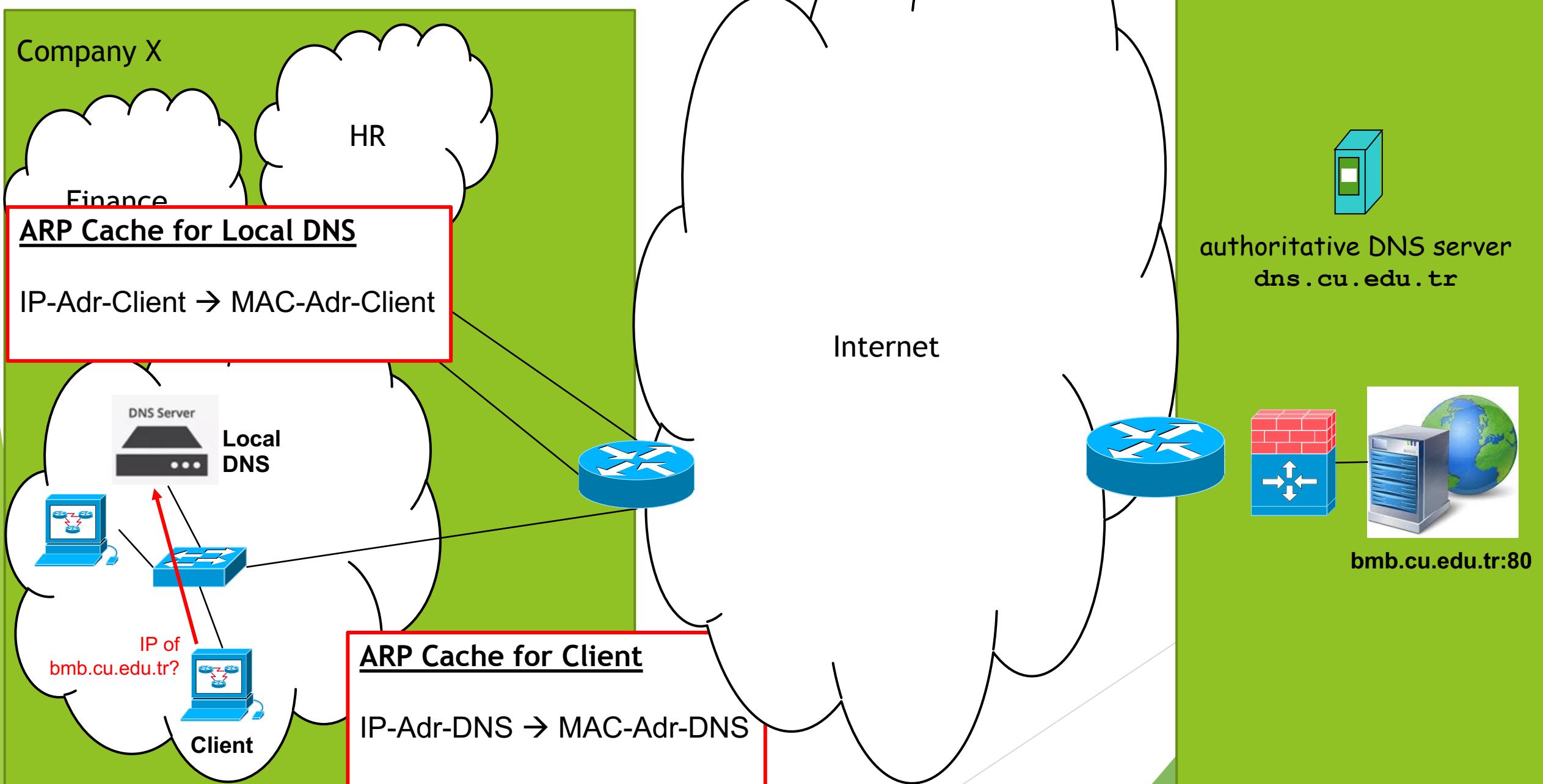


Step 2: Client resolves the MAC of local DNS IP (Response)



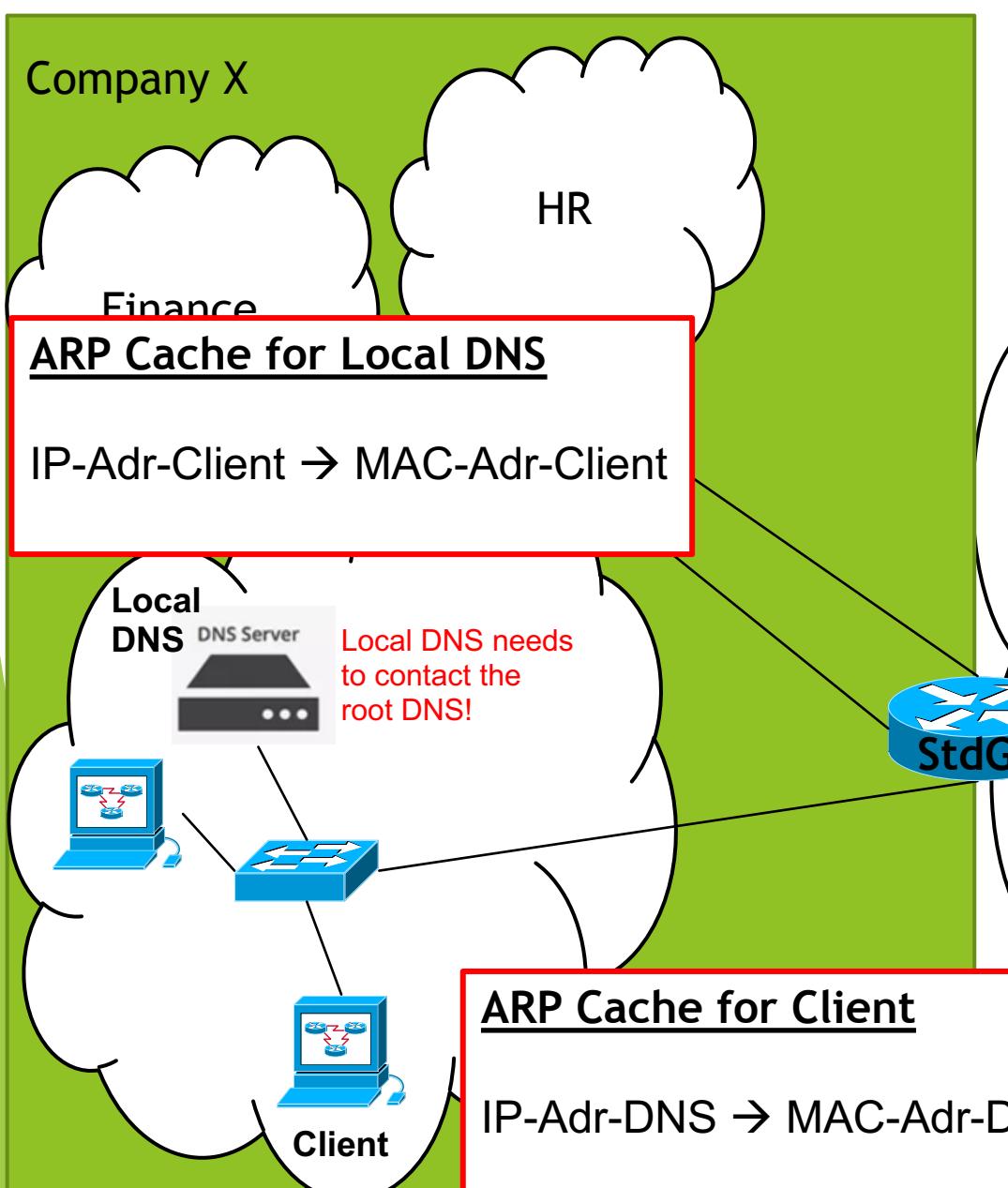
Step 3: Client asks the local DNS for resolving the IP of bmb.cu.edu.tr!

Cukurova University



Step 4: Local DNS needs to communicate with root DNS for resolving the IP of bmb.cu.edu.tr!

Check whether root DNS is in the same or different subnet!



- Find out whether **root DNS IP** is in the same subnet?
 - Applying the logical «and» operation for dest. IP and subnet masks
- In this case: different subnet -> **routing is necessary**
- ARP cache of local DNS doesn't include the MAC of StdGw.
- Find out the **MAC address of standard gateway (StdGw)** using ARP request (broadcast with FF:FF:FF:FF:FF)



authoritative DNS server
dns.cu.edu.tr



bmb.cu.edu.tr:80

Step 4: Client resolves the MAC of StdGw (Request)

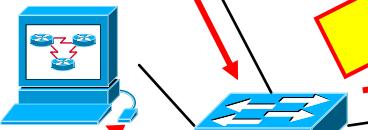
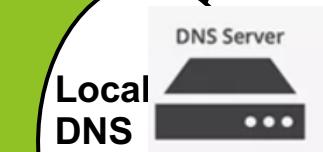
Company X

Finance

HR

ARP Cache for Local DNS

IP-Adr-Client → MAC-Adr-Client



MAC of
StdGw IP?

ARP request for resolving StdGw's MAC

StdGw

ARP Cache for Client

IP-Adr-DNS → MAC-Adr-DNS

REQUEST: PC with MAC-Adr-Local-DNS and IP-Adr-Local-DNS to all: Looking for a MAC? from PC with IP-Adr-StdGw

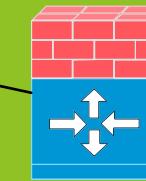
DNS

ARP-Cache:
empty

StdGw

ARP-Cache:
IP-Adr-Local-DNS → MAC-Adr-Local-DNS

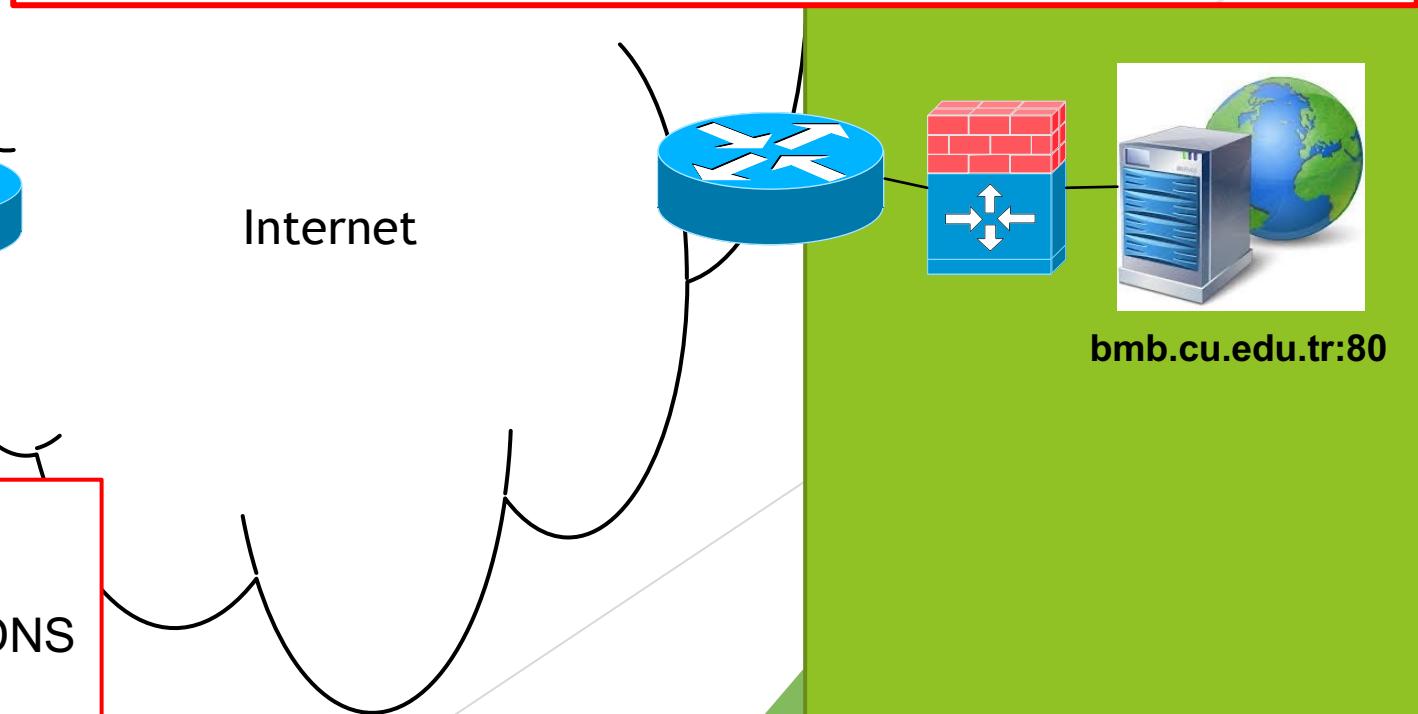
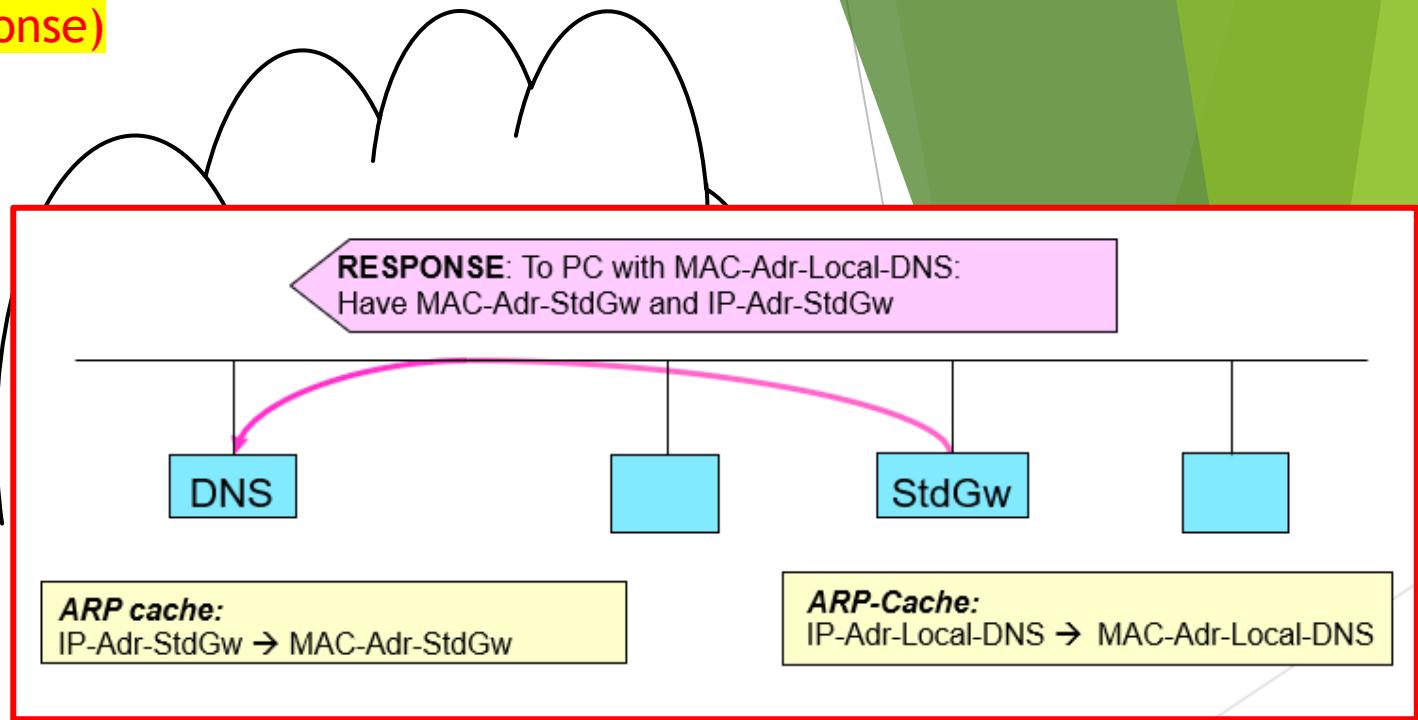
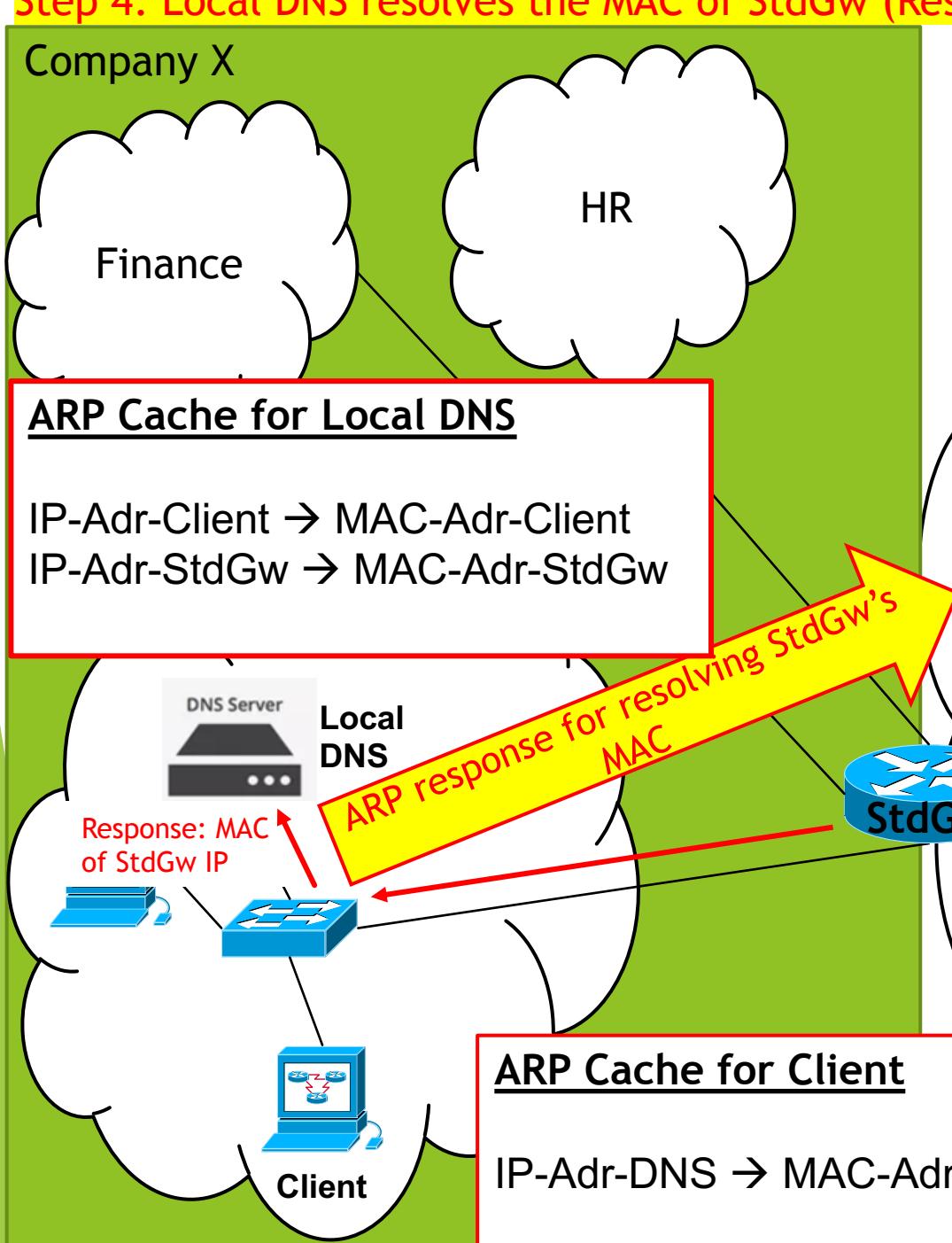
Internet



bmb.cu.edu.tr:80

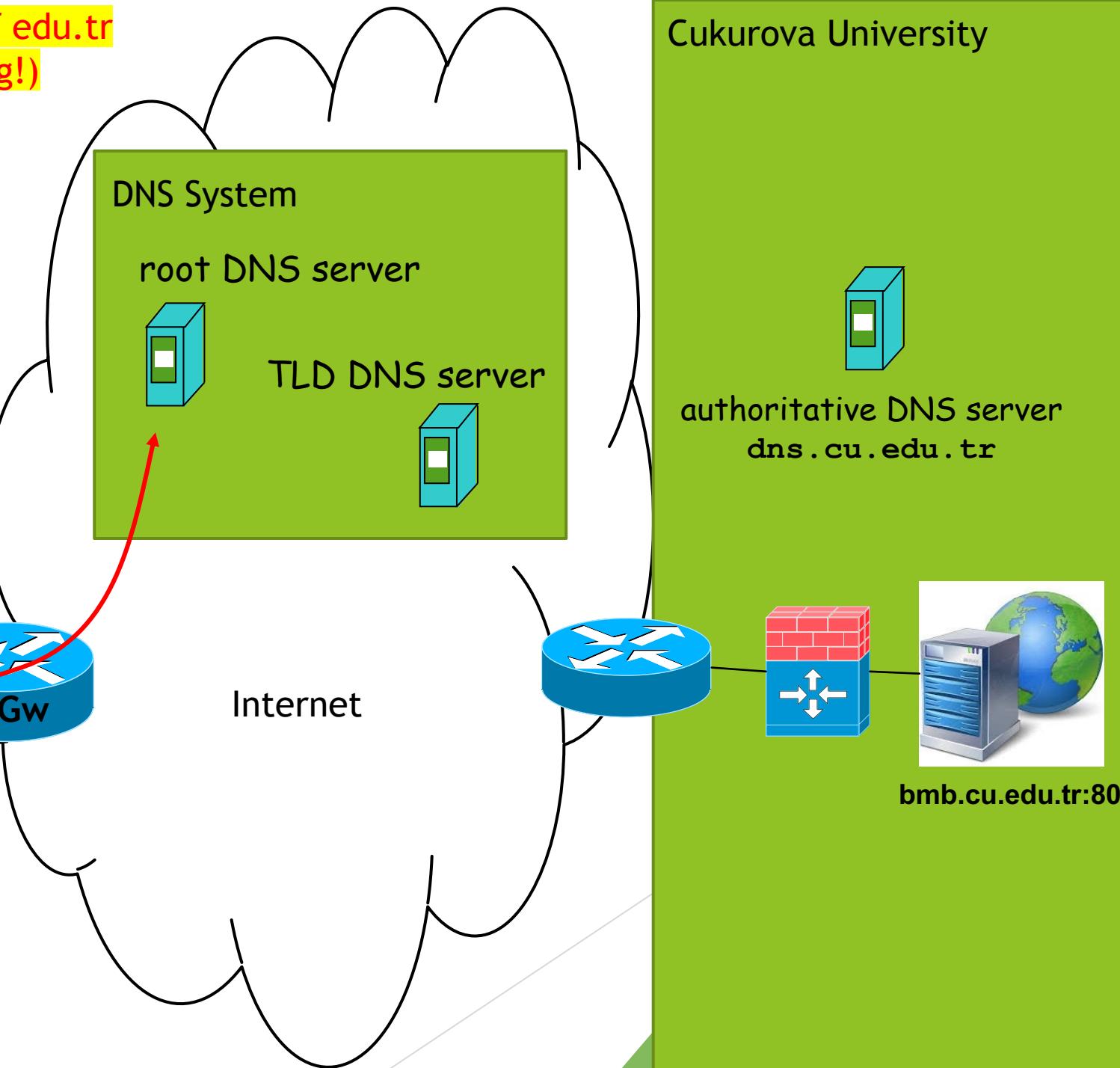
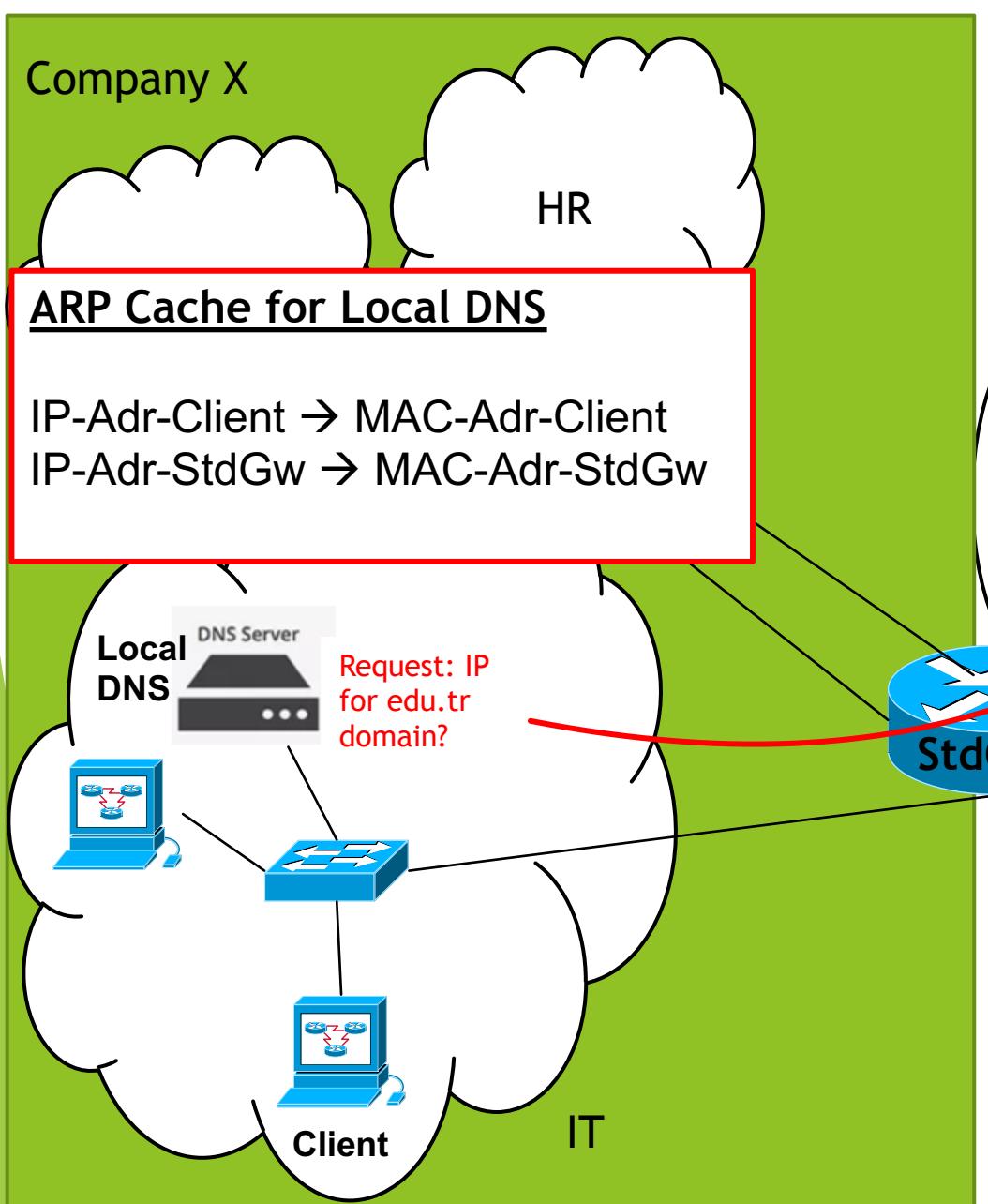
Step 4: Local DNS resolves the MAC of StdGw (Response)

Company X



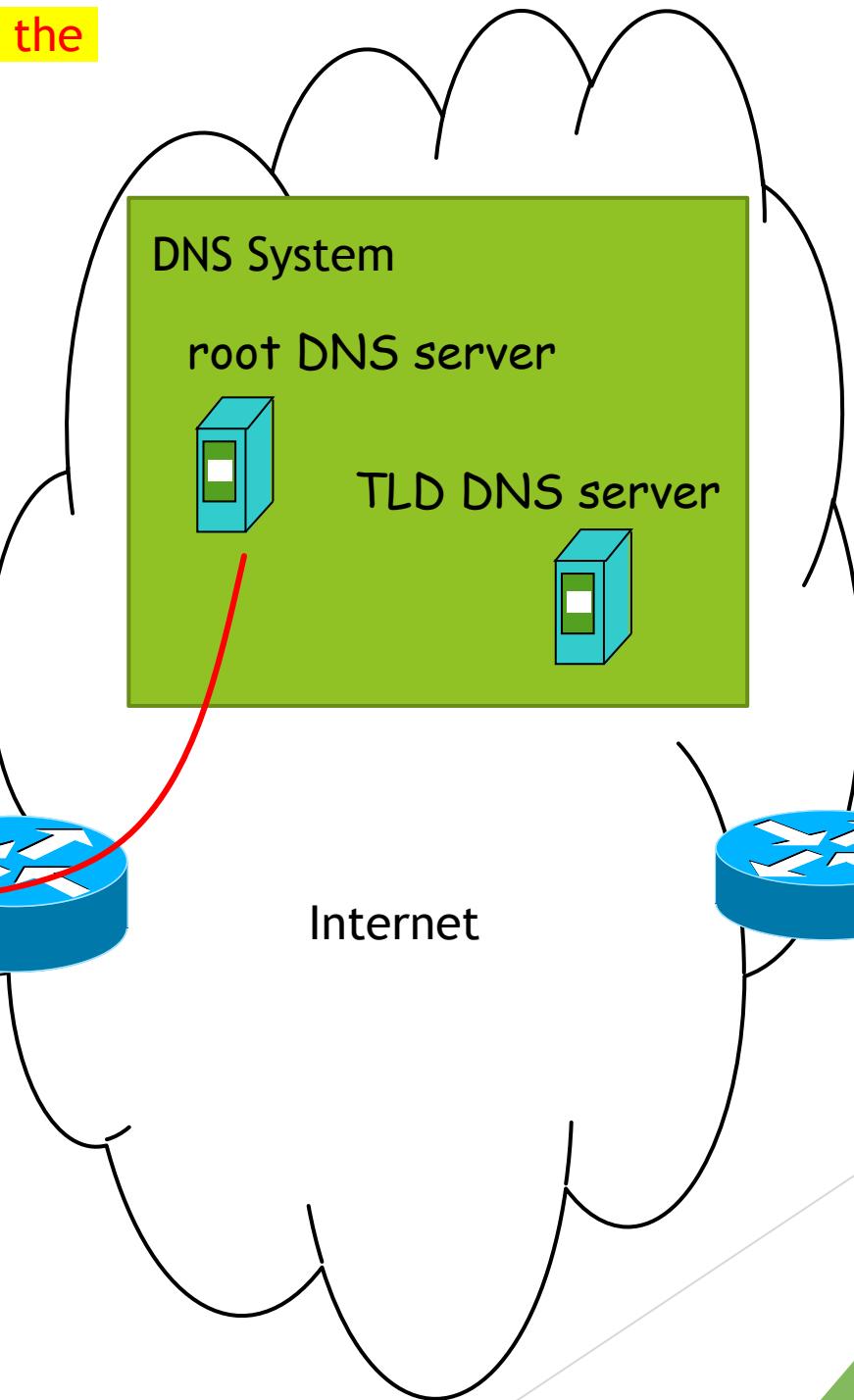
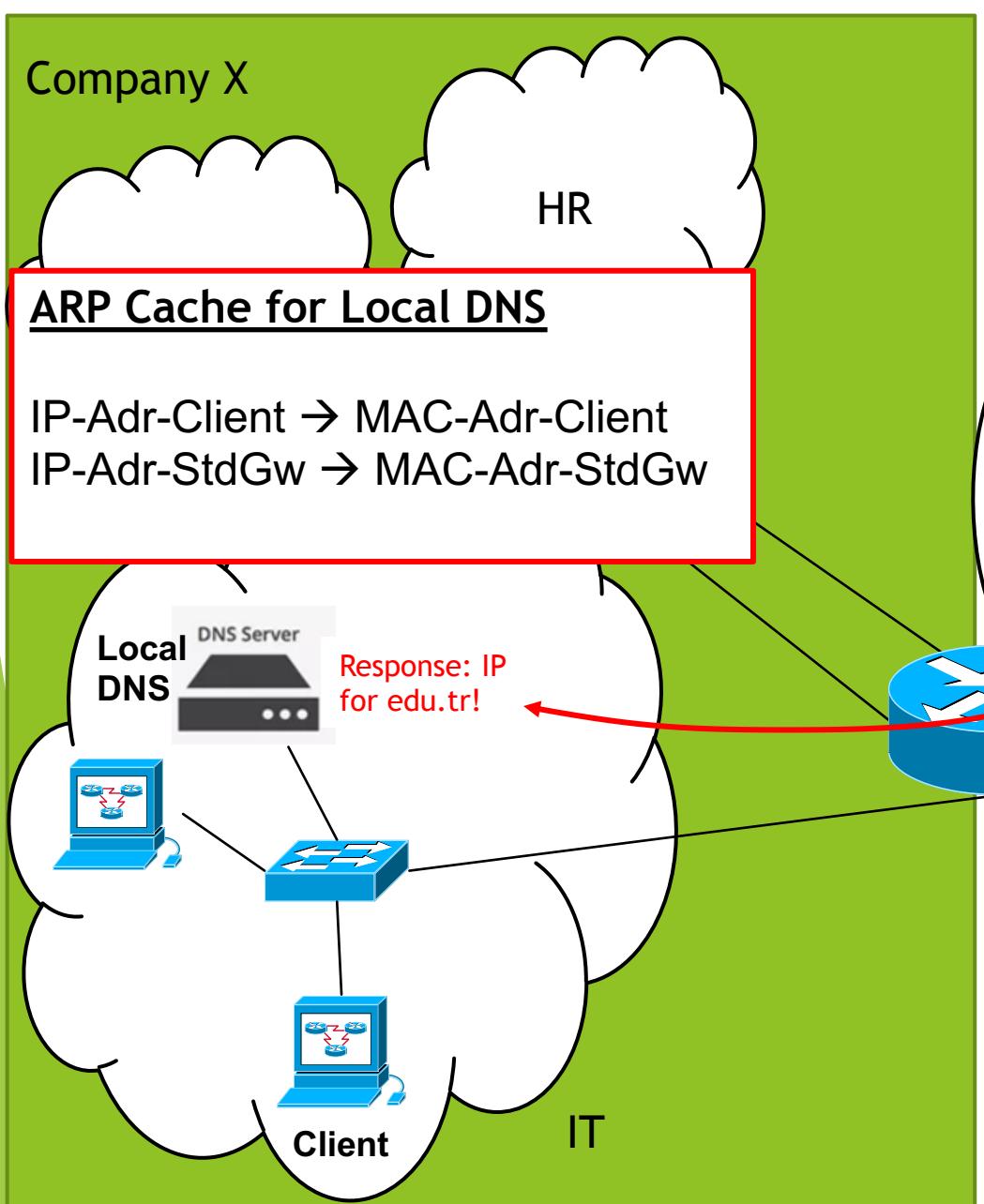
Step 5: Local DNS asks the root DNS for the IP of edu.tr over the StdGw! (StdGw takes the rest of routing!)

Cukurova University



Step 5: Local DNS receives the IP of edu.tr from the root DNS !

Cukurova University



Step 6: Local DNS needs to communicate with TLD DNS for resolving the IP of cu.edu.tr domain.

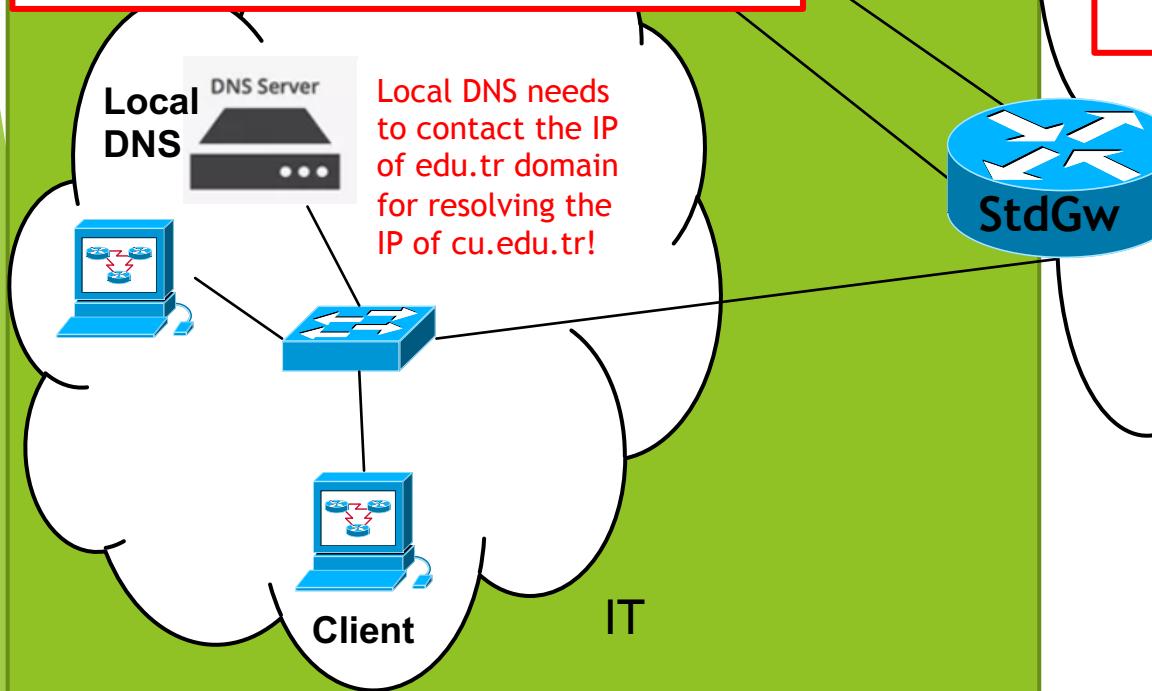
Check whether TLD DNS is in the same or different subnet!

Company X

HR

ARP Cache for Local DNS

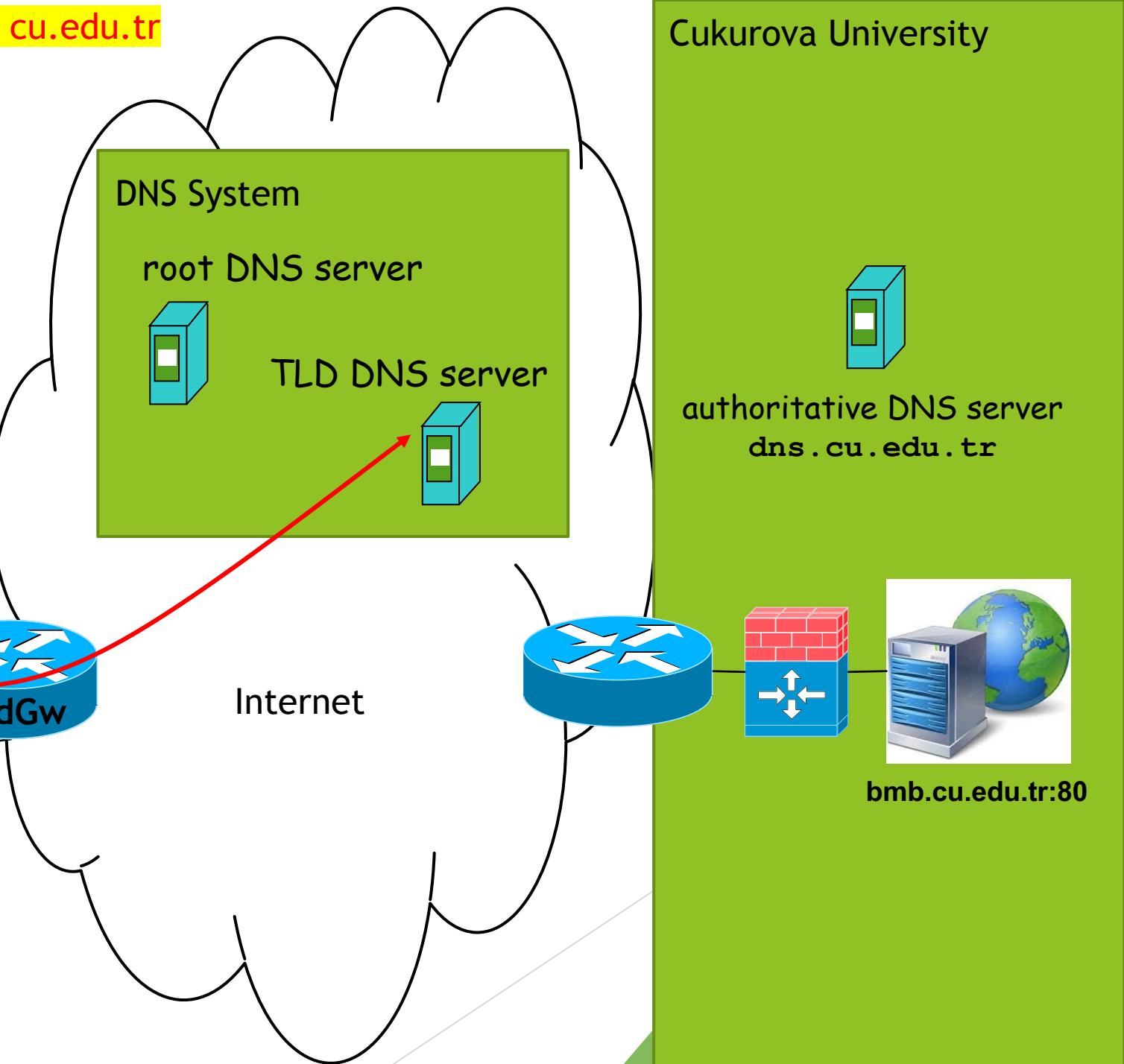
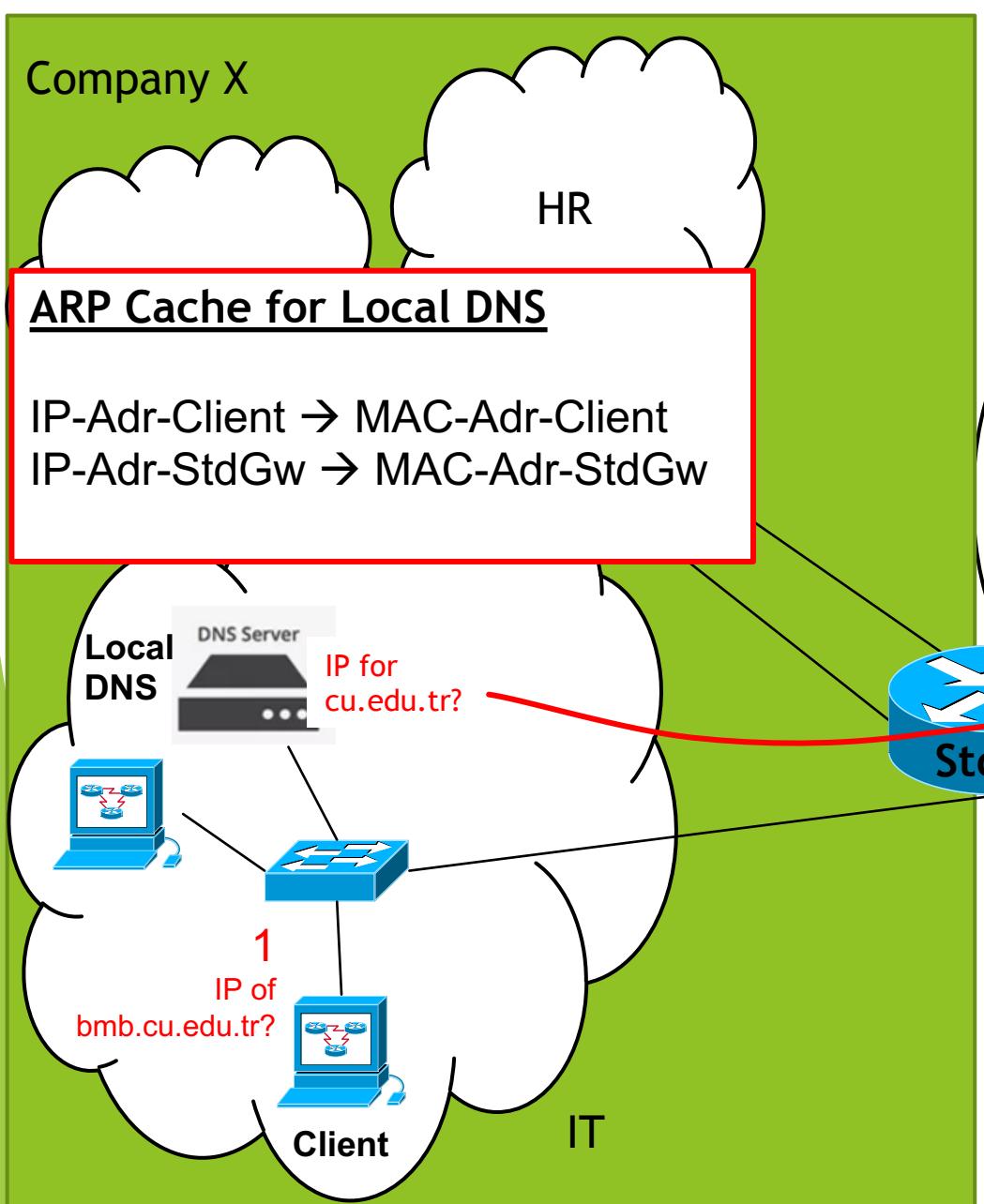
IP-Adr-Client → MAC-Adr-Client
IP-Adr-StdGw → MAC-Adr-StdGw



- Find out whether **TLD DNS IP (i.e., edu.tr)** is in the same subnet?
 - Applying the logical «and» operation for dest. IP and subnet masks
- In this case: different subnet -> **routing is necessary**
- Switch the request directly (!) to the previously learned MAC address of standard gateway (StdGw). The rest of routing is performed by the StdGw.

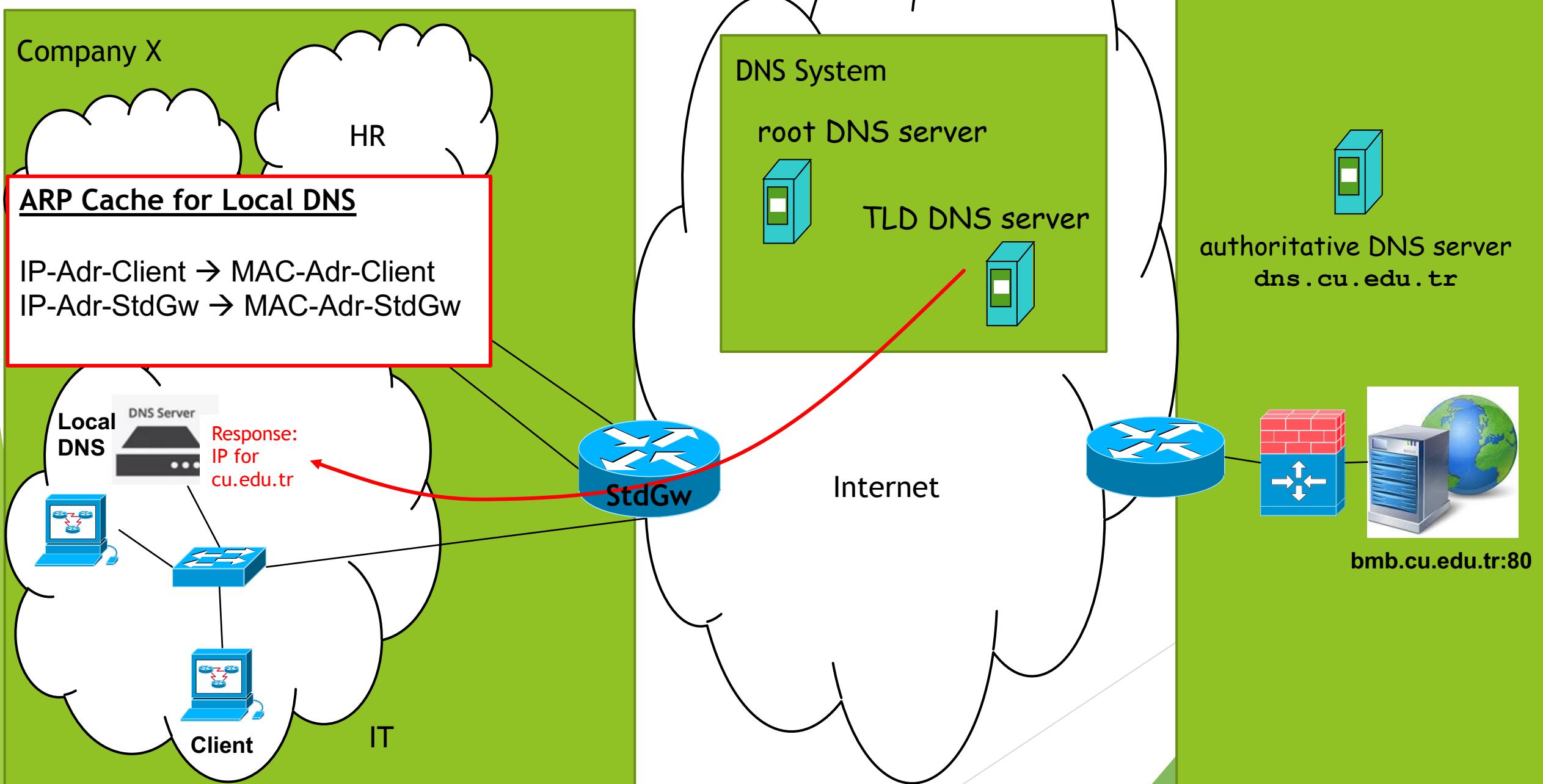
Step 6: Local DNS asks the TLD DNS for the IP of cu.edu.tr over the StdGw.

Cukurova University



Step 7: Local DNS receives from the TLD DNS the IP of cu.edu.tr!

Cukurova University



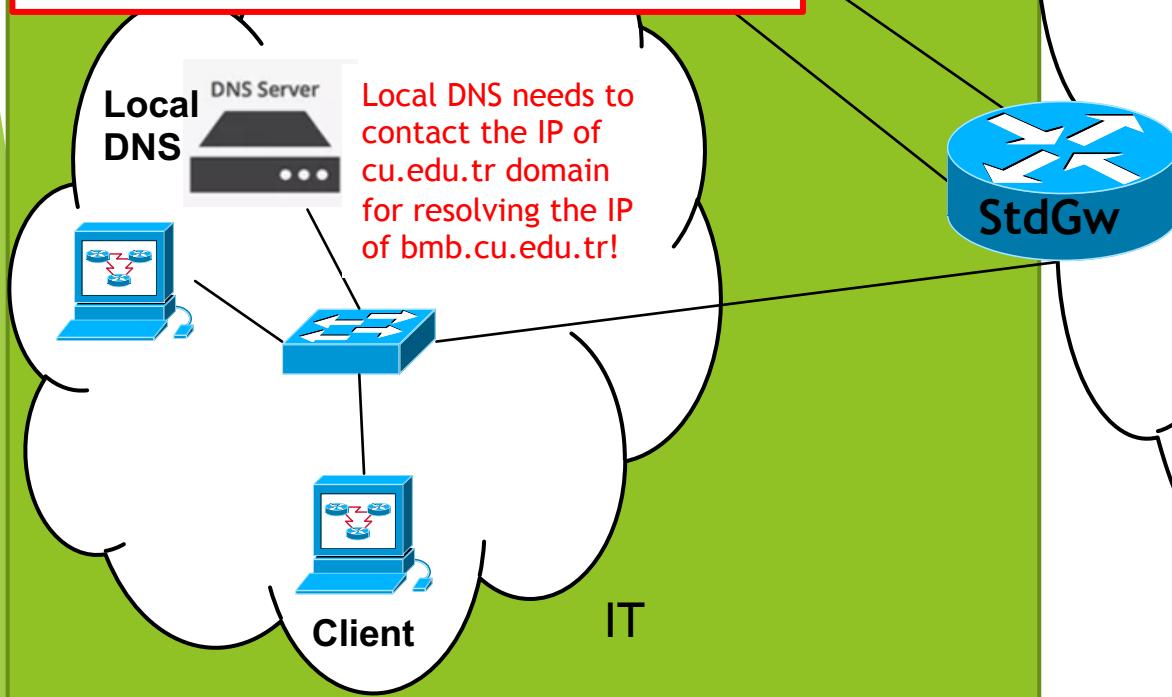
Step 8: Local DNS needs to communicate with Autoritative DNS for resoving the IP of **bmb.cu.edu.tr**. Check whether Autoritative DNS is in the same or different subnet!

Company X

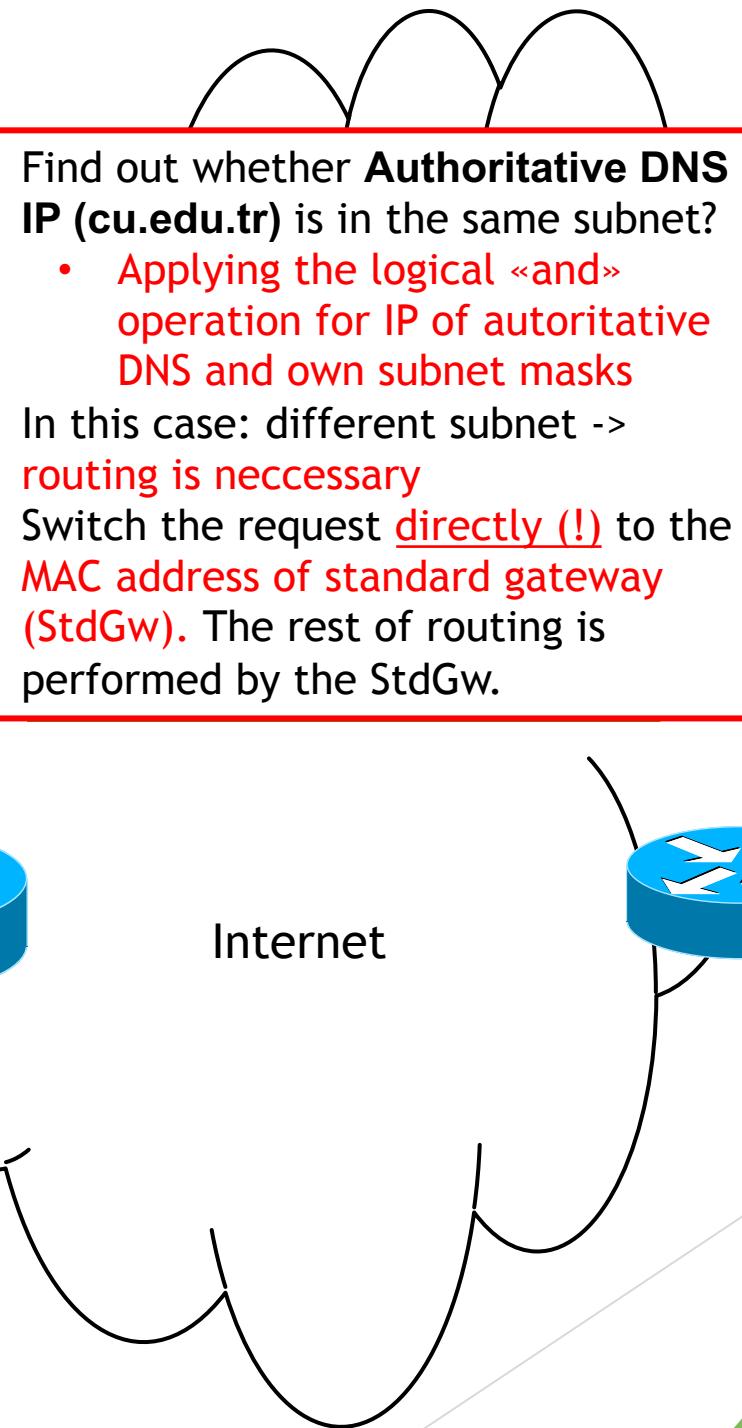


ARP Cache for Local DNS

IP-Adr-Client → MAC-Adr-Client
IP-Adr-StdGw → MAC-Adr-StdGw



- Find out whether **Autoritative DNS IP (cu.edu.tr)** is in the same subnet?
 - Applying the logical «and» operation for IP of autoritative DNS and own subnet masks
- In this case: different subnet -> **routing is necessary**
- Switch the request directly (!) to the **MAC address of standard gateway (StdGw)**. The rest of routing is performed by the StdGw.



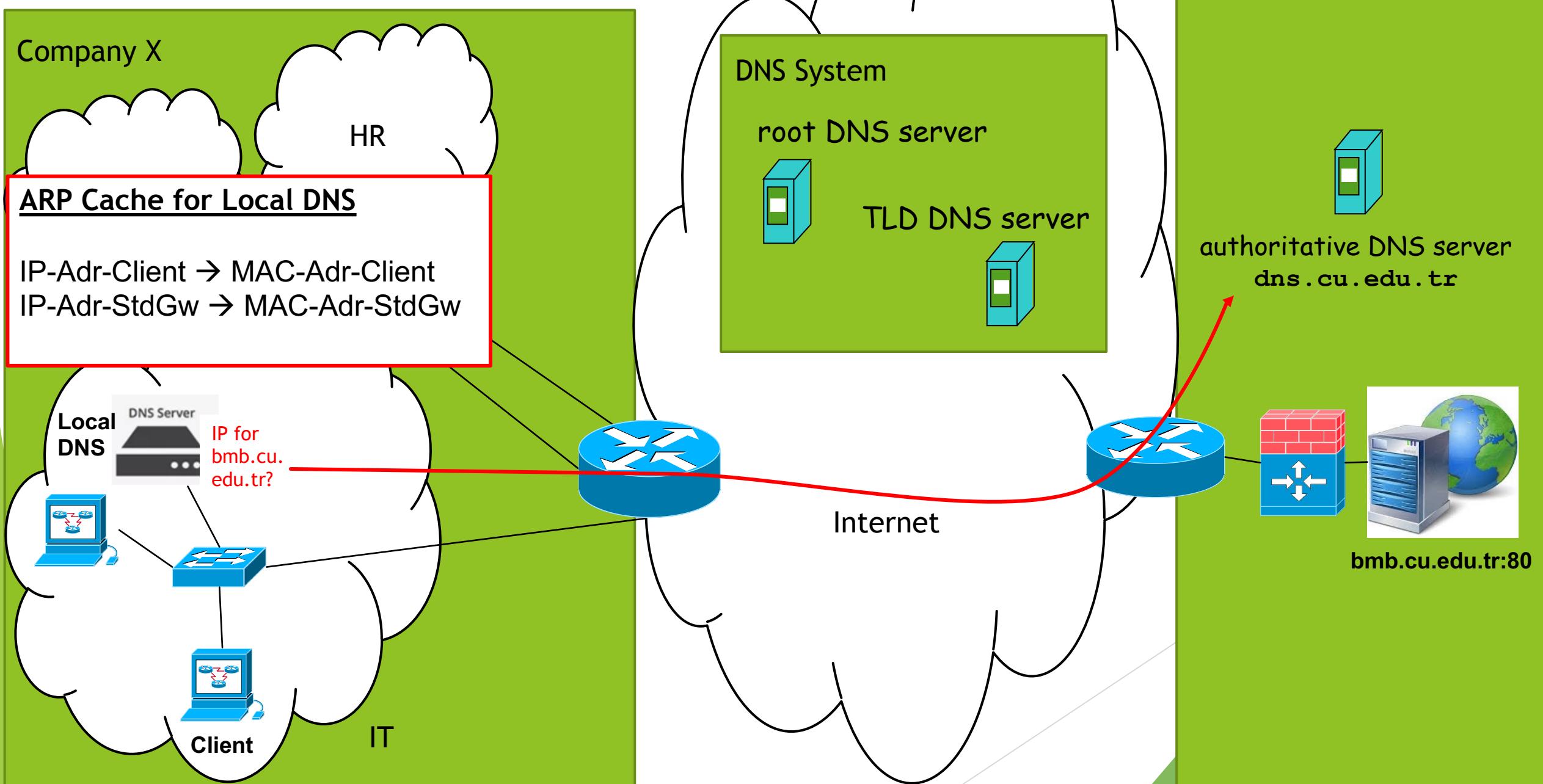
authoritative DNS server
dns.cu.edu.tr



bmb.cu.edu.tr:80

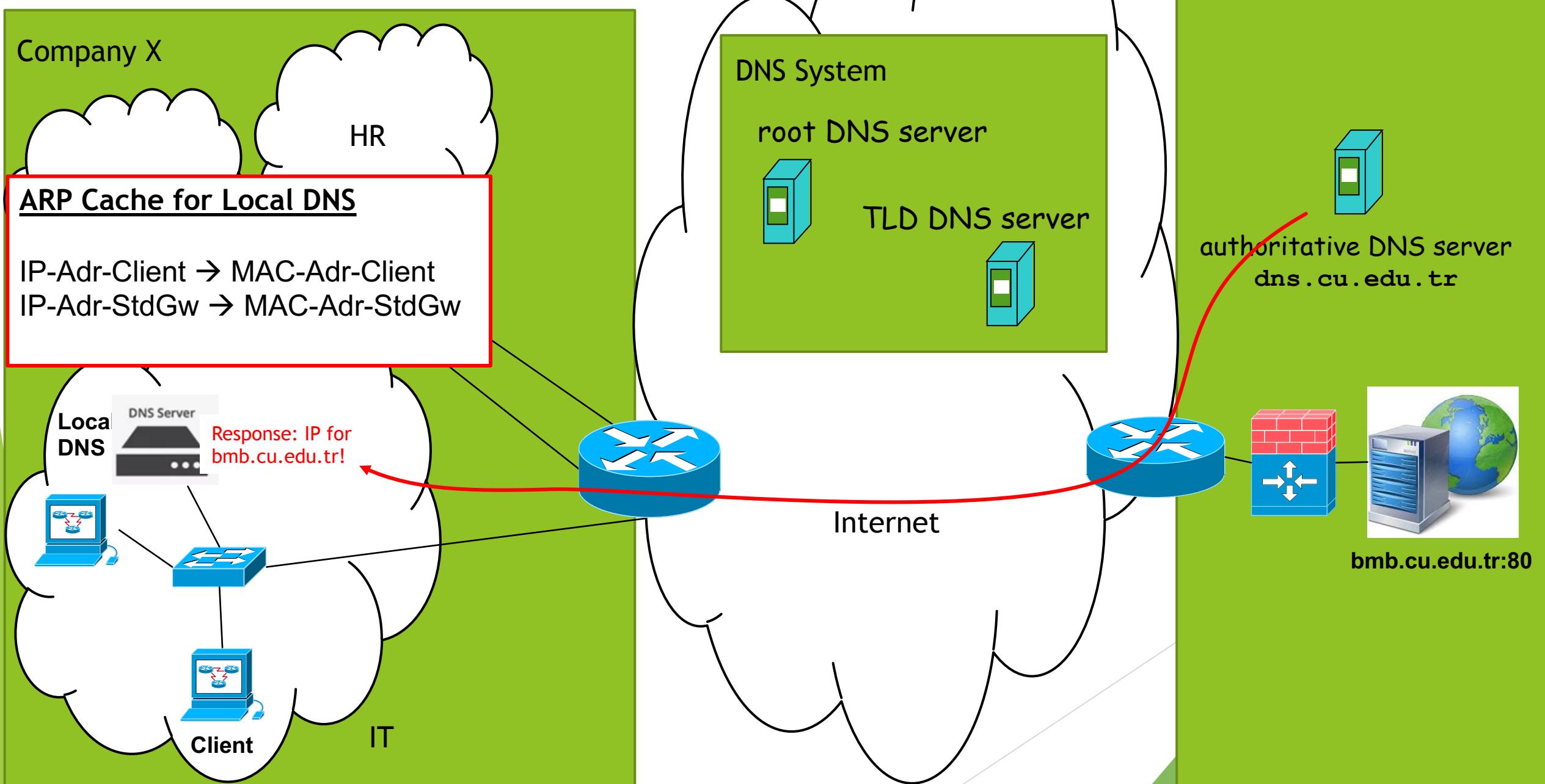
Step 8: Local DNS asks the Autoritative DNS for the IP of bmb.cu.edu.tr over the StdGw.

Cukurova University



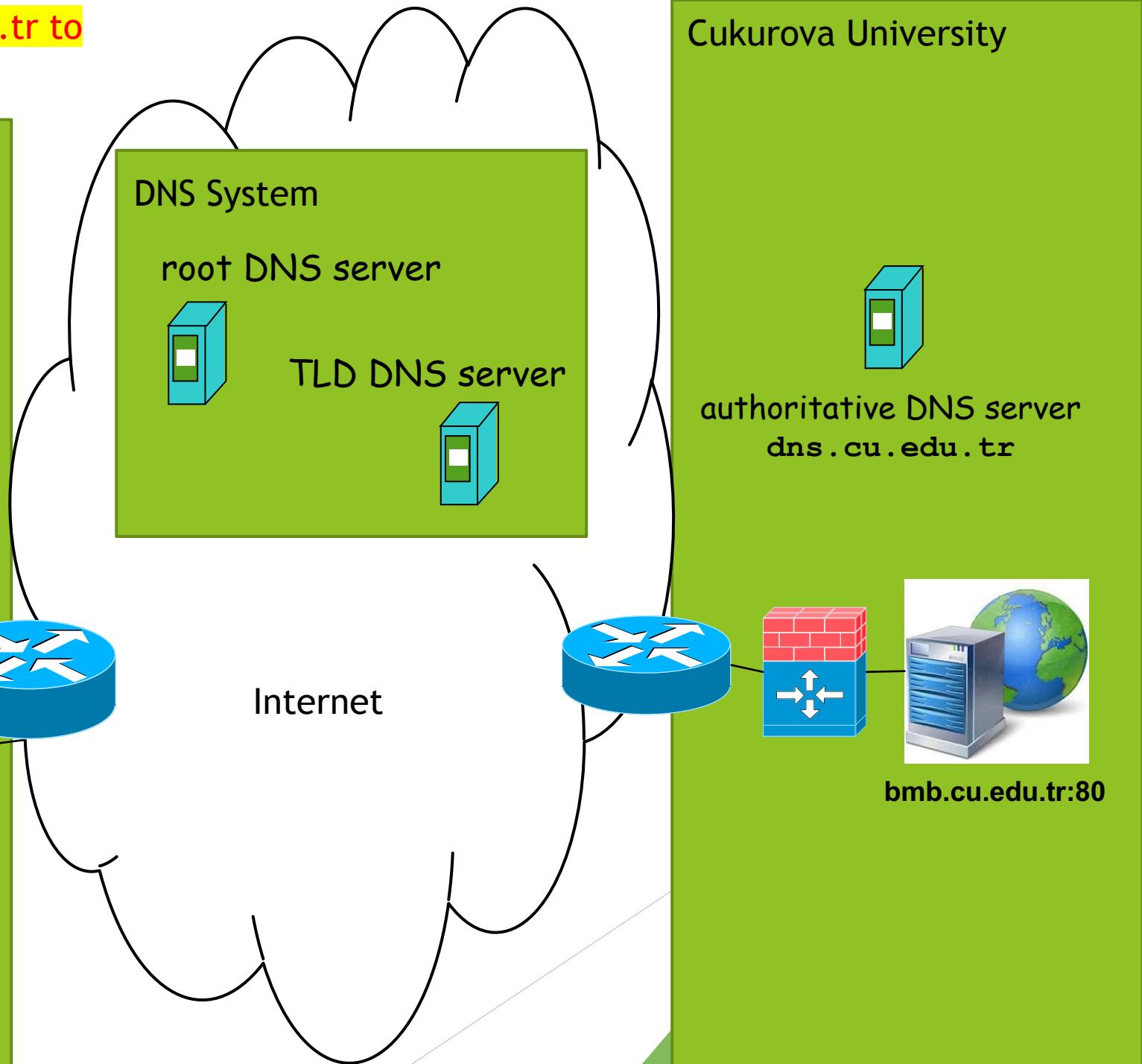
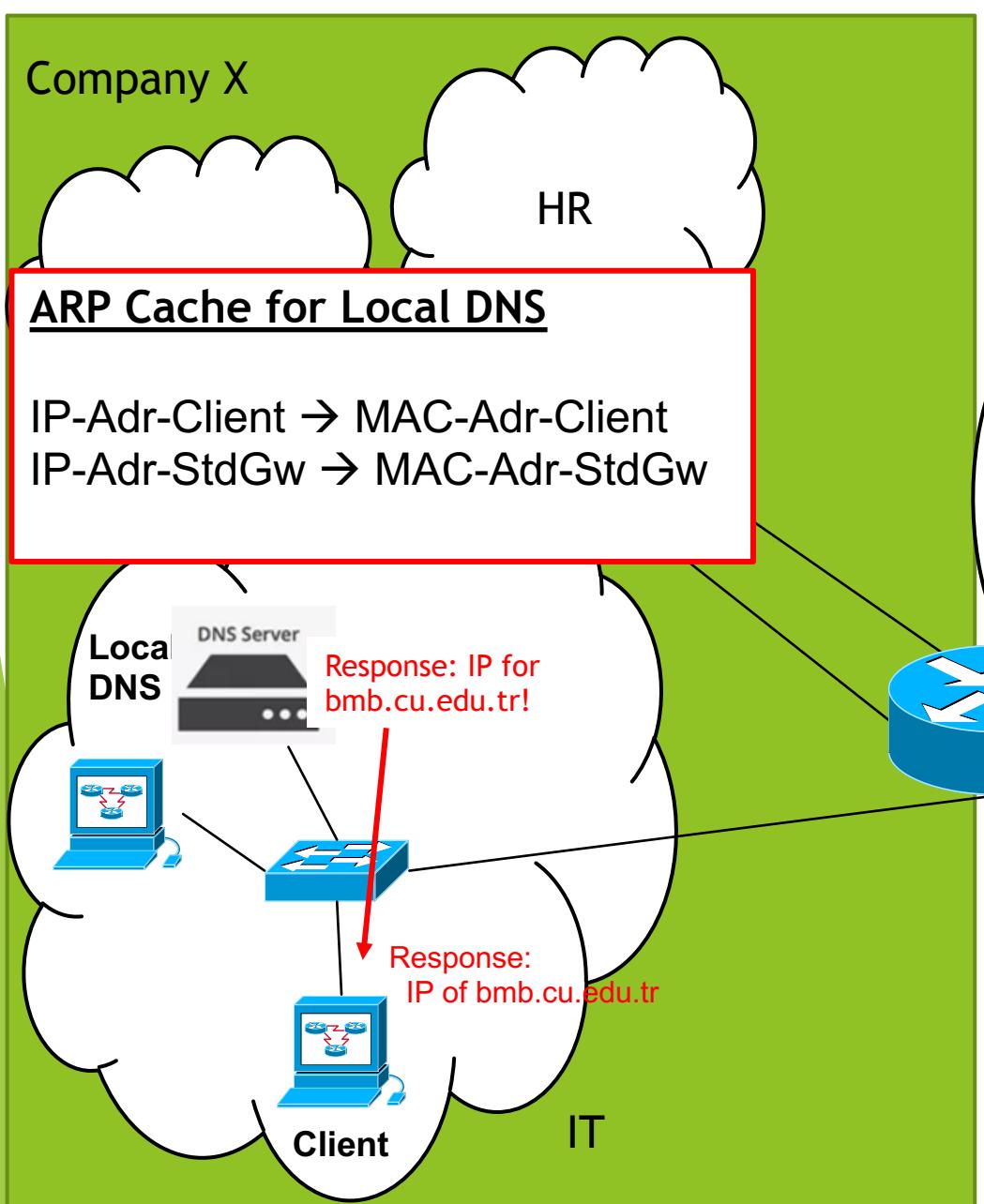
Step 9: Local DNS receives the IP of bmb.cu.edu.tr from the Autoritative DNS !

Cukurova University



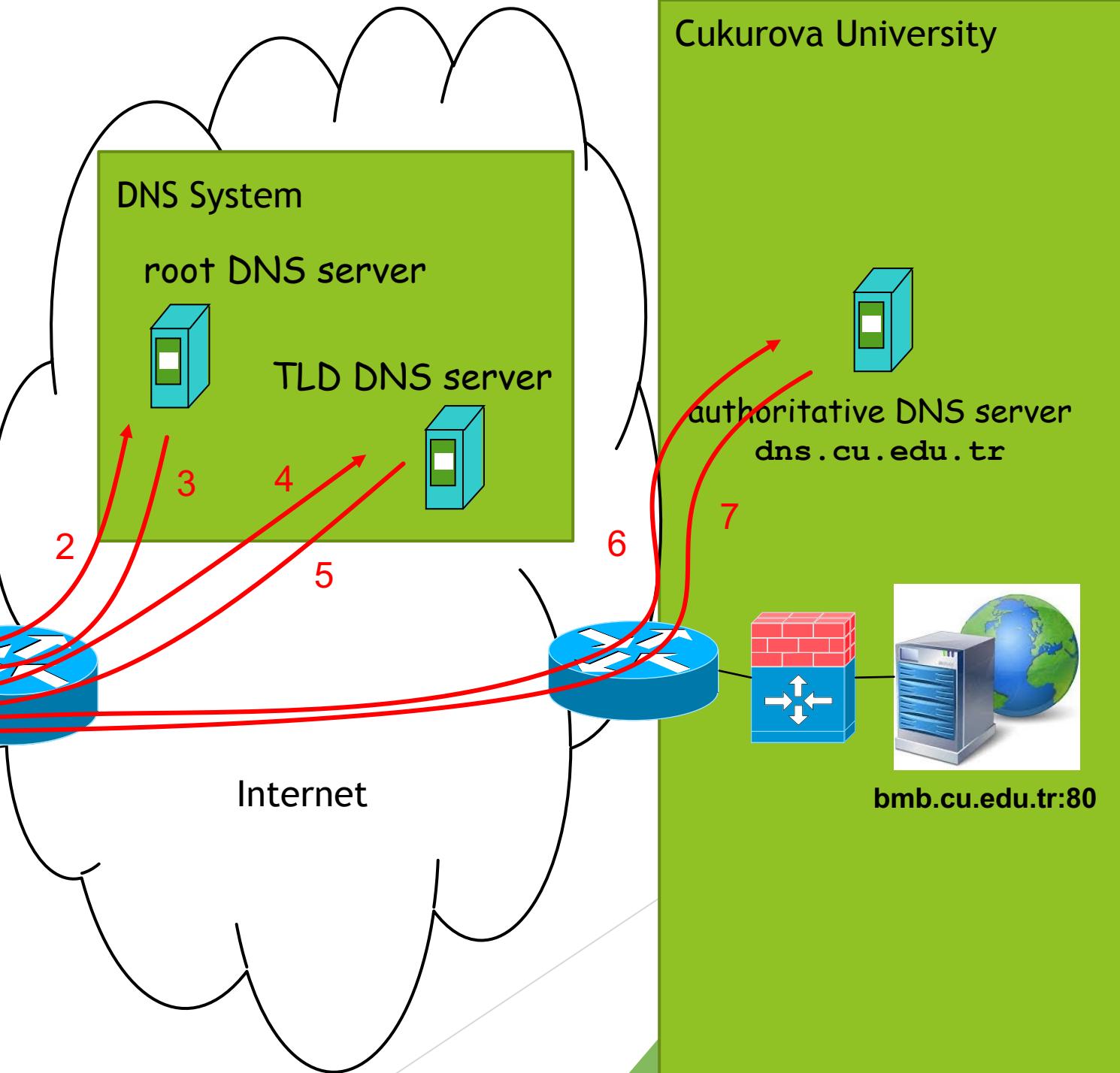
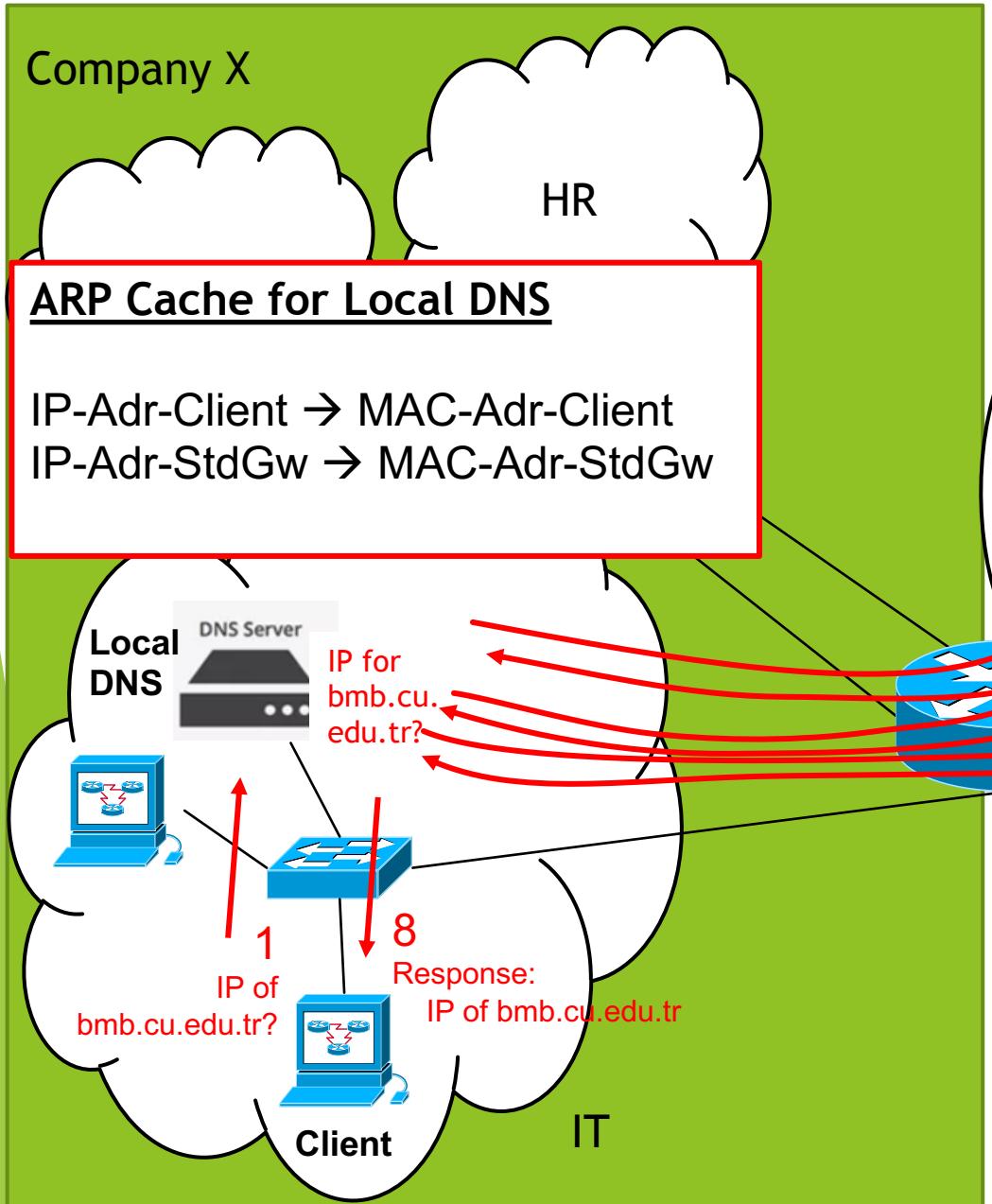
Step 10: Local DNS sends the IP of bmb.cu.edu.tr to the client.

Cukurova University



Summary for DNS resolution

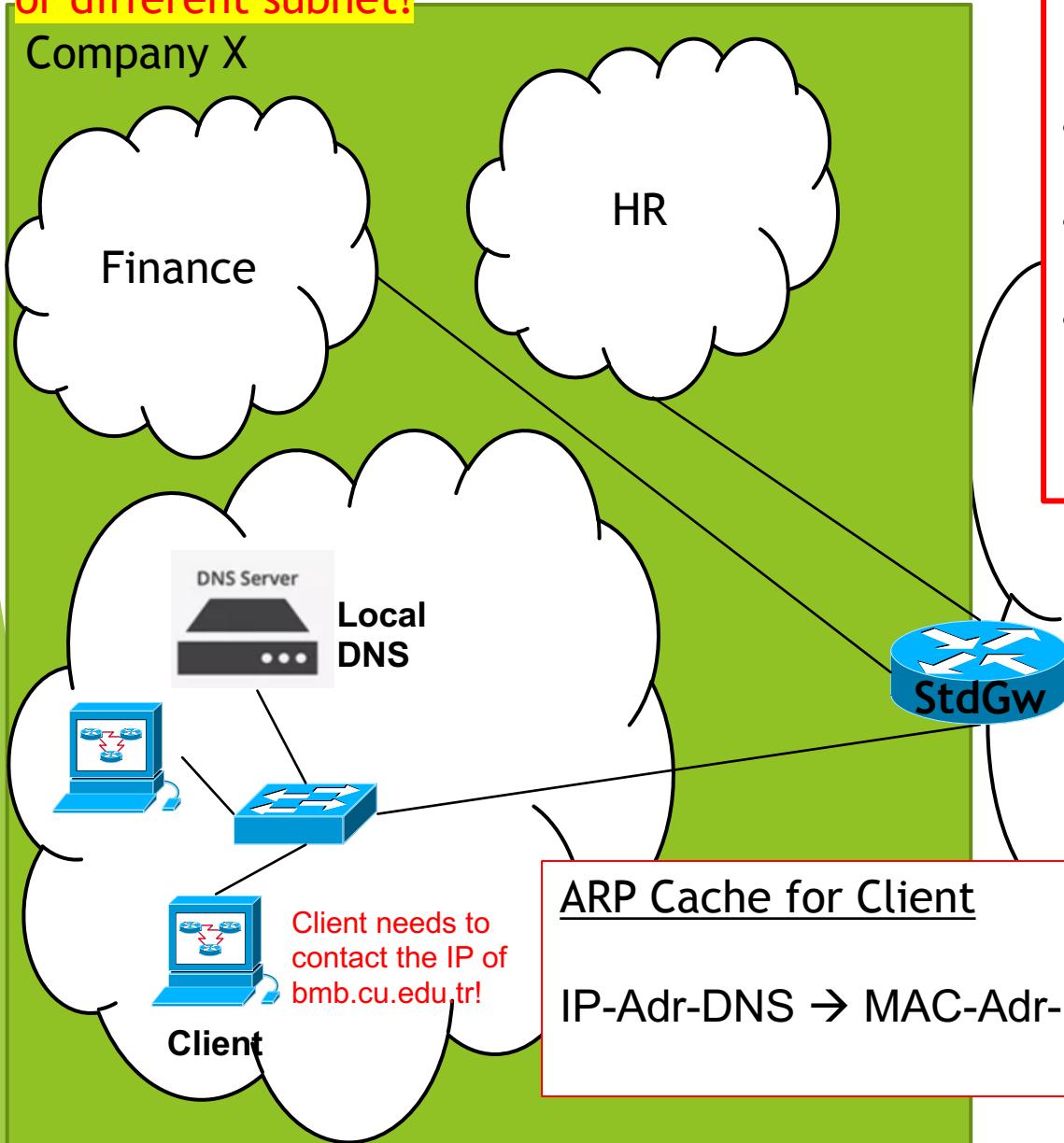
Cukurova University



Step 11: Client wants to establish a TCP connection to the IP of bmb.cu.edu.tr.

Check whether IP of bmb.cu.edu.tr is in the same or different subnet!

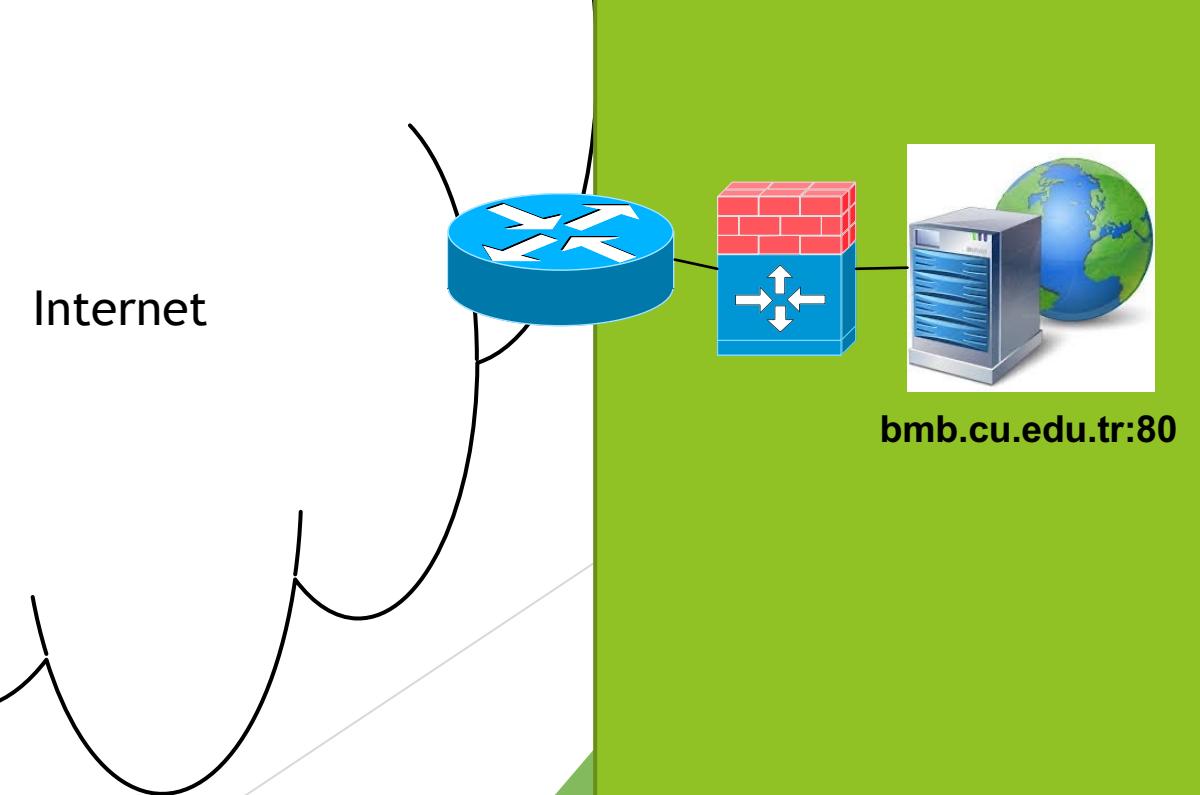
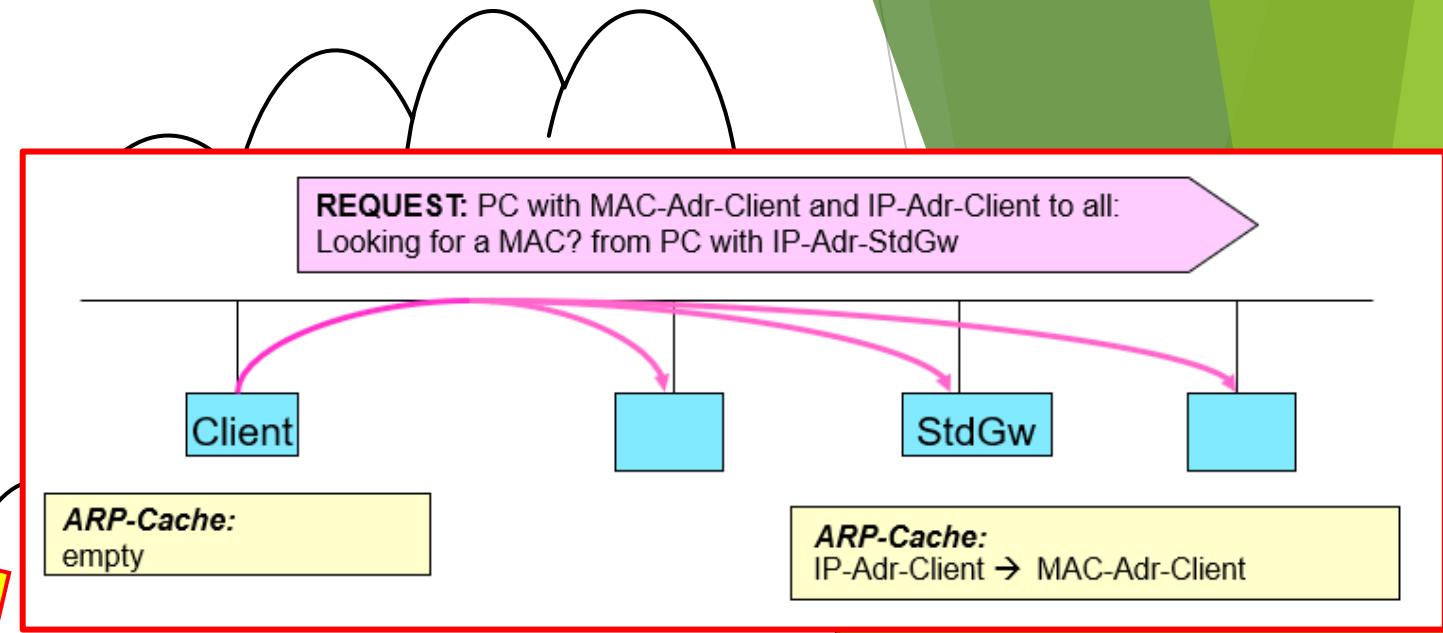
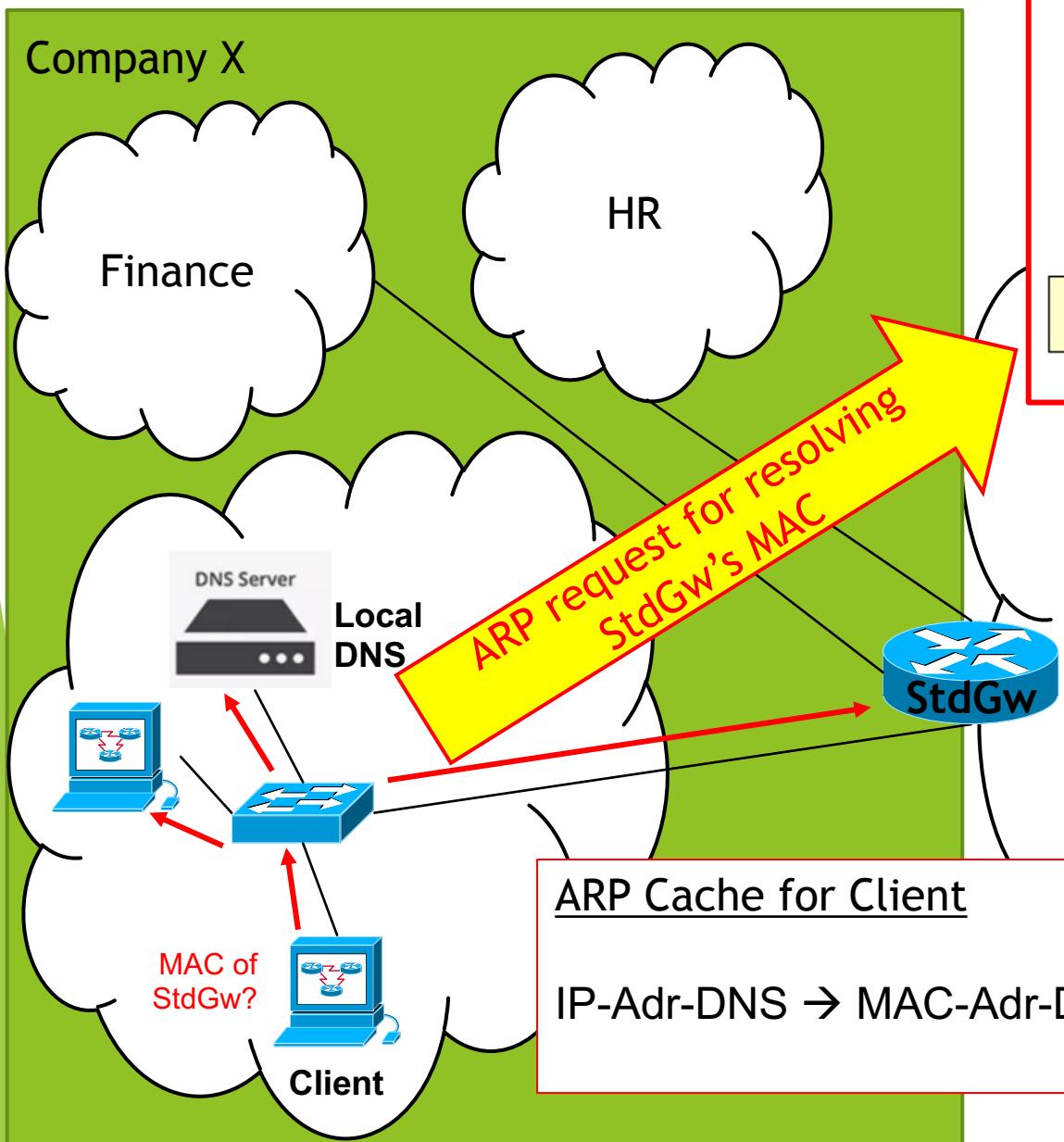
Company X



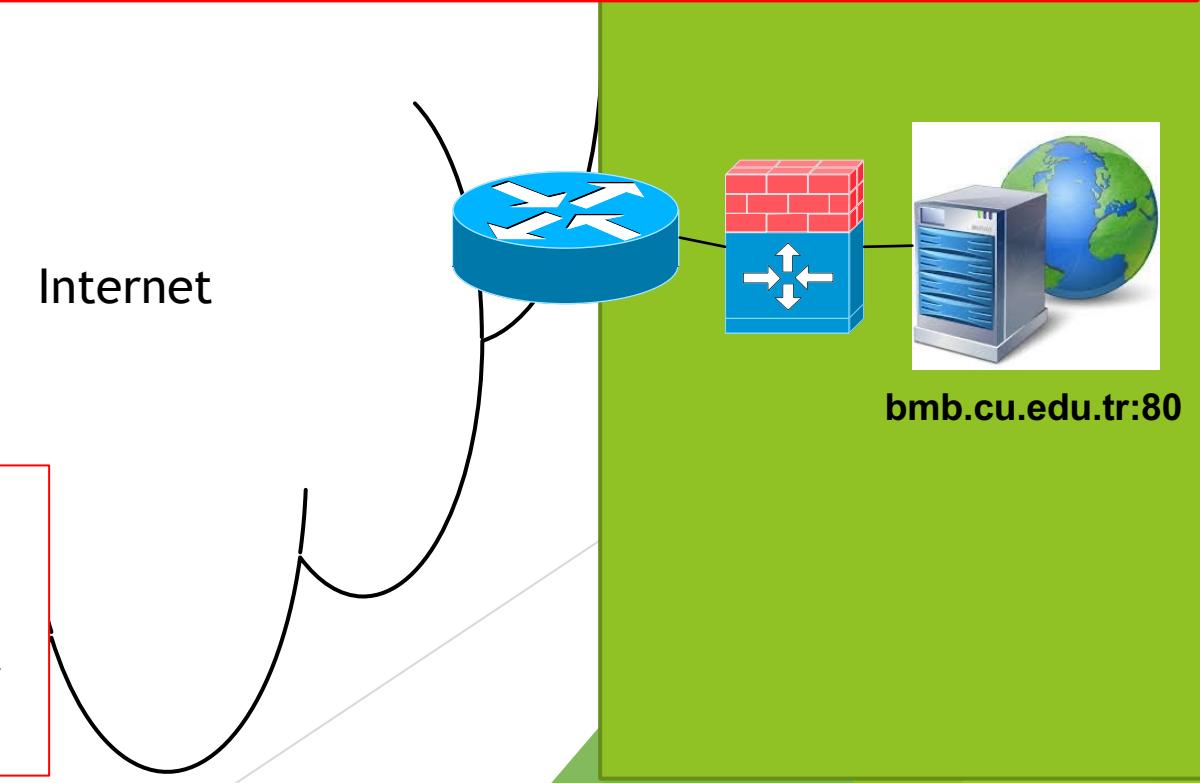
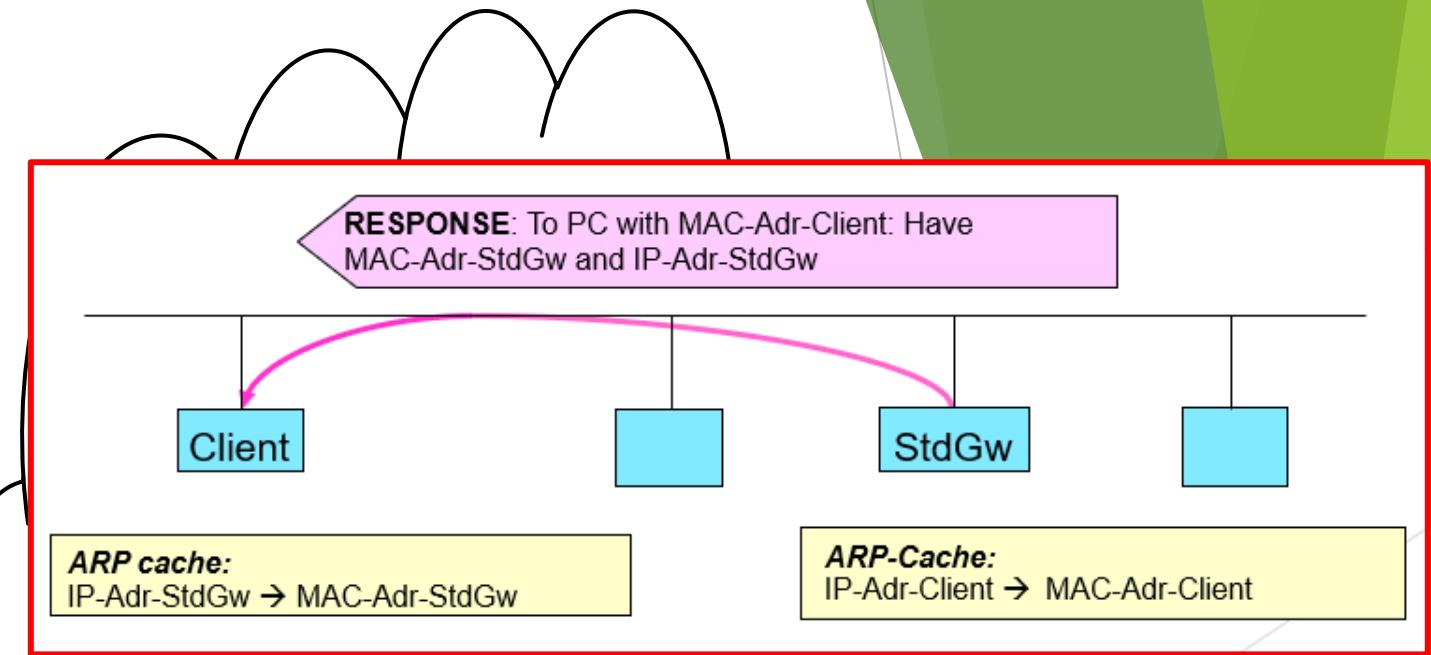
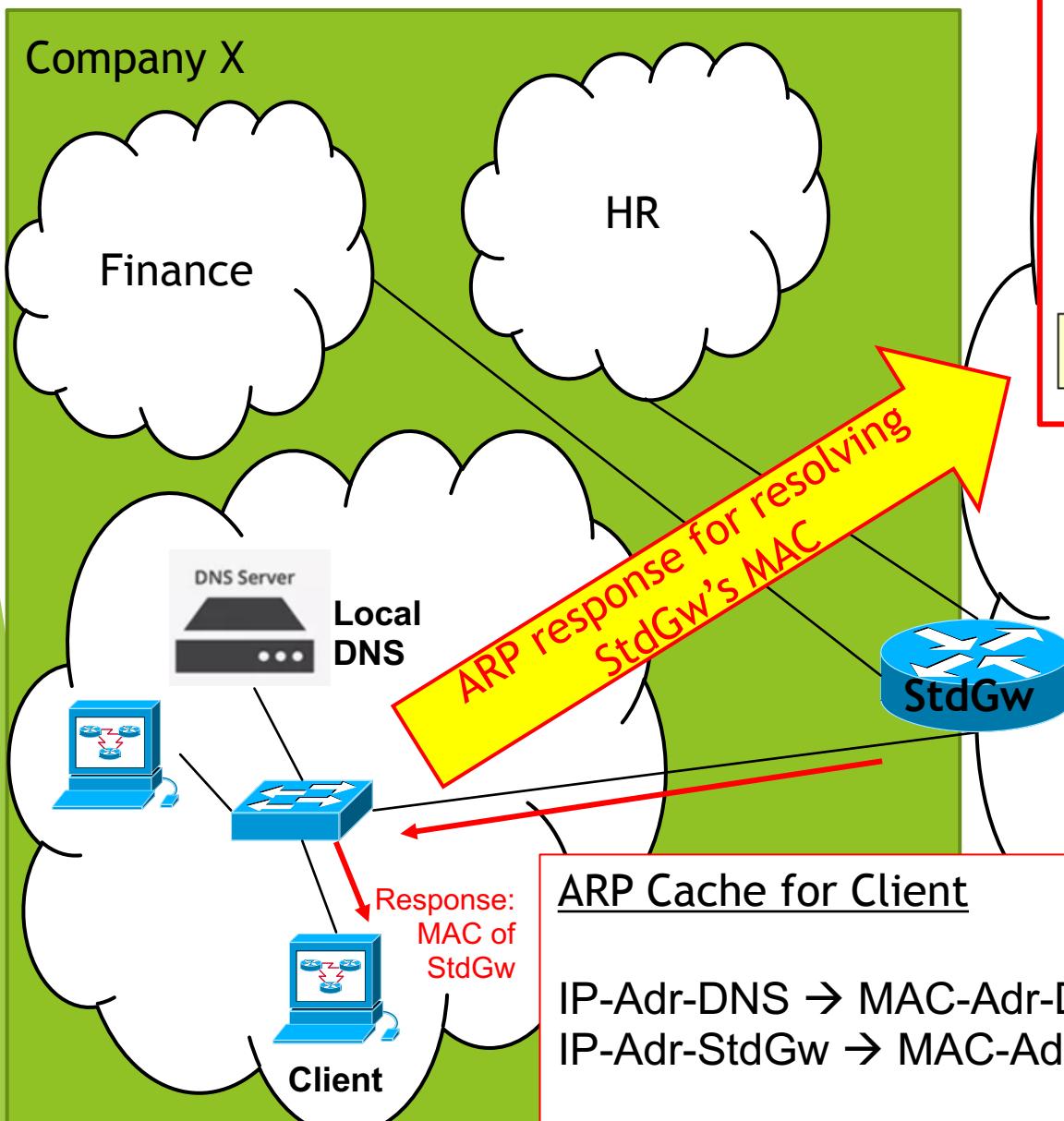
- Find out whether **IP of bmb.cu.edu.tr** is in the same subnet?
 - Applying the logical «and» operation for dest. IP and subnet masks
- In this case: different subnet -> **routing is necessary**
- ARP cache of client doesn't include the MAC of StdGw.
- Find out the **MAC address of standard gateway (StdGw)** using ARP request (broadcast FF:FF:FF:FF:FF:FF)



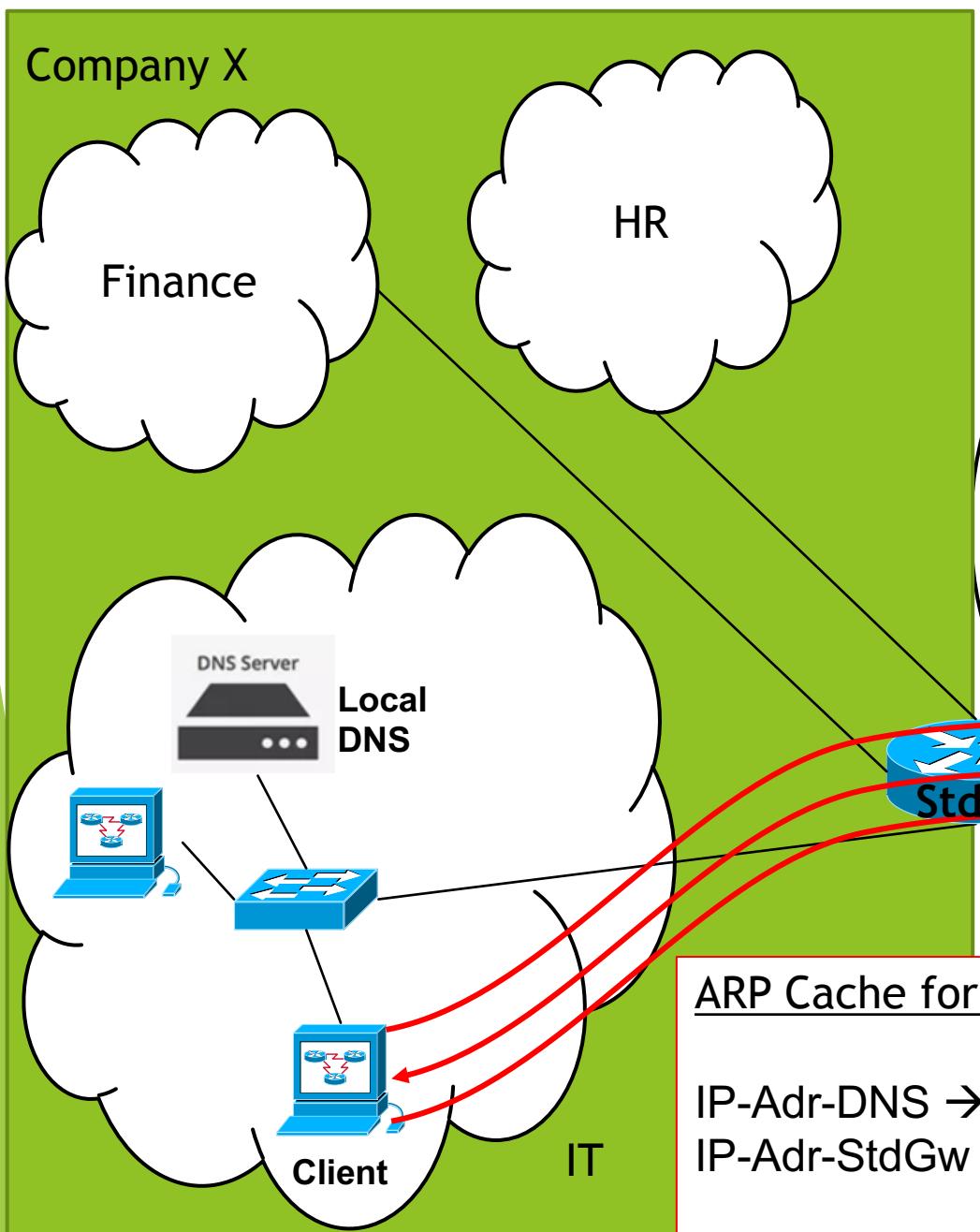
Step 11: Client resolves the MAC of StdGw IP (Request)



Step 12: Client resolves the MAC of StdGw IP (Response)



Step 13: Client establishes a TCP connection to the IP of bmb.cu.edu.tr over the StdGw.



Note:

- For simplicity reasons, the steps for deciding whether the destination IP is in the same subnet or not, are not illustrated anymore in detail!
- But keep in mind that this inspection (i.e., whether switching or routing is required) is performed for every outgoing packet!

1) SYN

2) SYN / ACK

3) ACK

Internet

ARP Cache for Client

IP-Adr-DNS → MAC-Adr-DNS

IP-Adr-StdGw → MAC-Adr-StdGw

Cukurova University

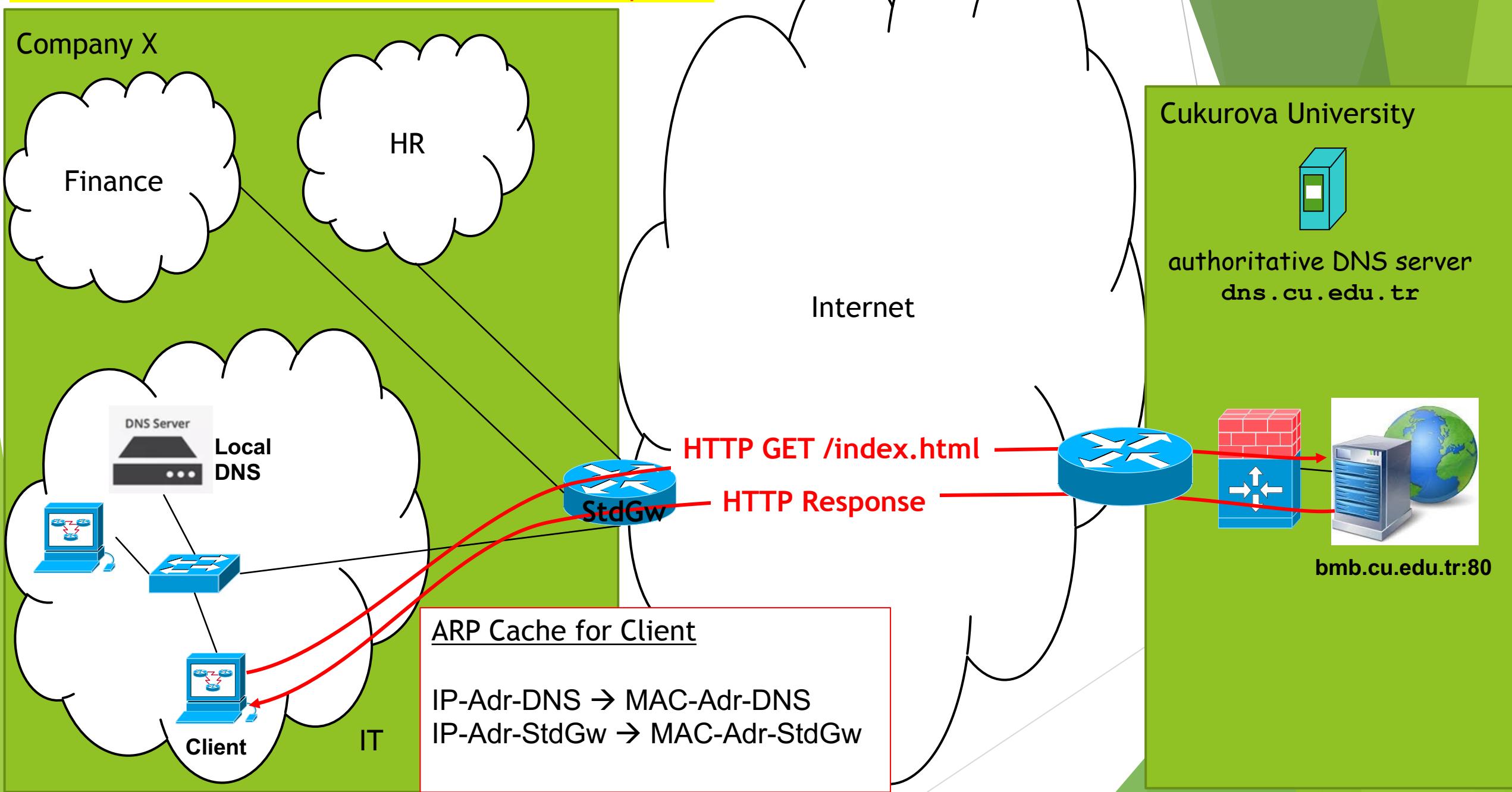


authoritative DNS server
dns.cu.edu.tr

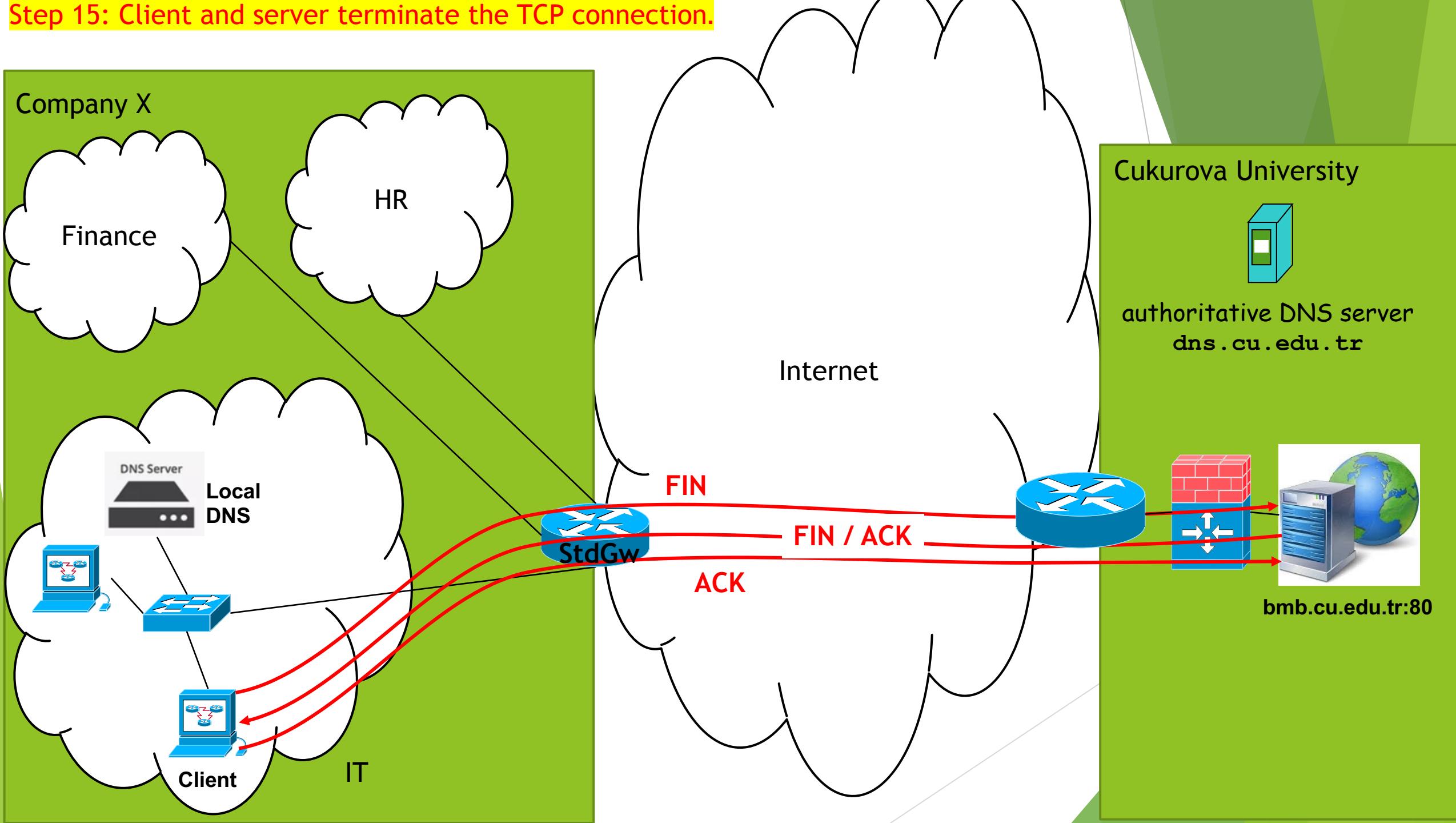


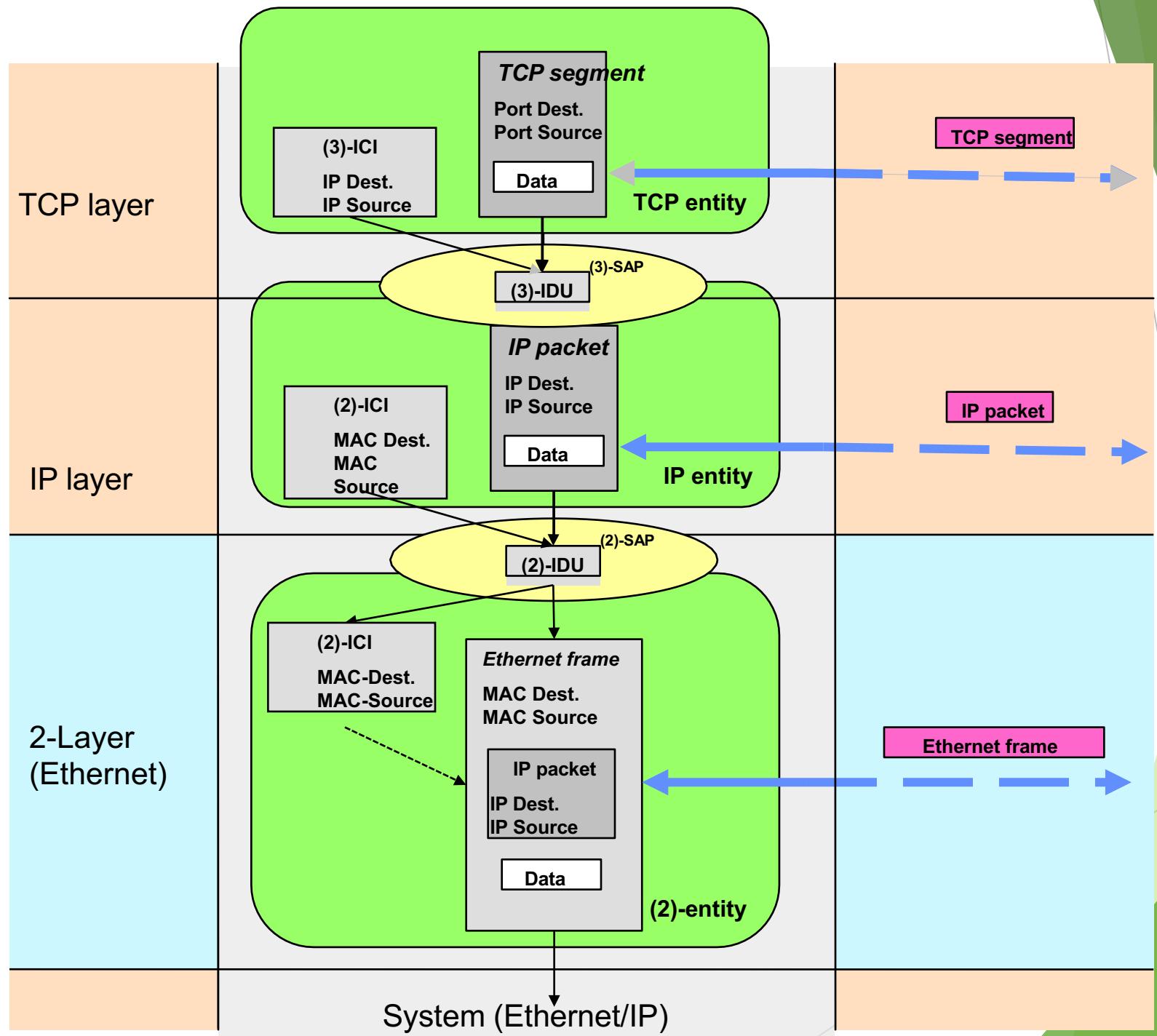
bmb.cu.edu.tr:80

Step 14: Client sends a HTTP GET request to the IP of bmb.cu.edu.tr with URL index.html over the established TCP connection and receives the response.



Step 15: Client and server terminate the TCP connection.





Overview of Application Layer Protocols

