Internet of Things (IoT) Definitions and Walkthrough

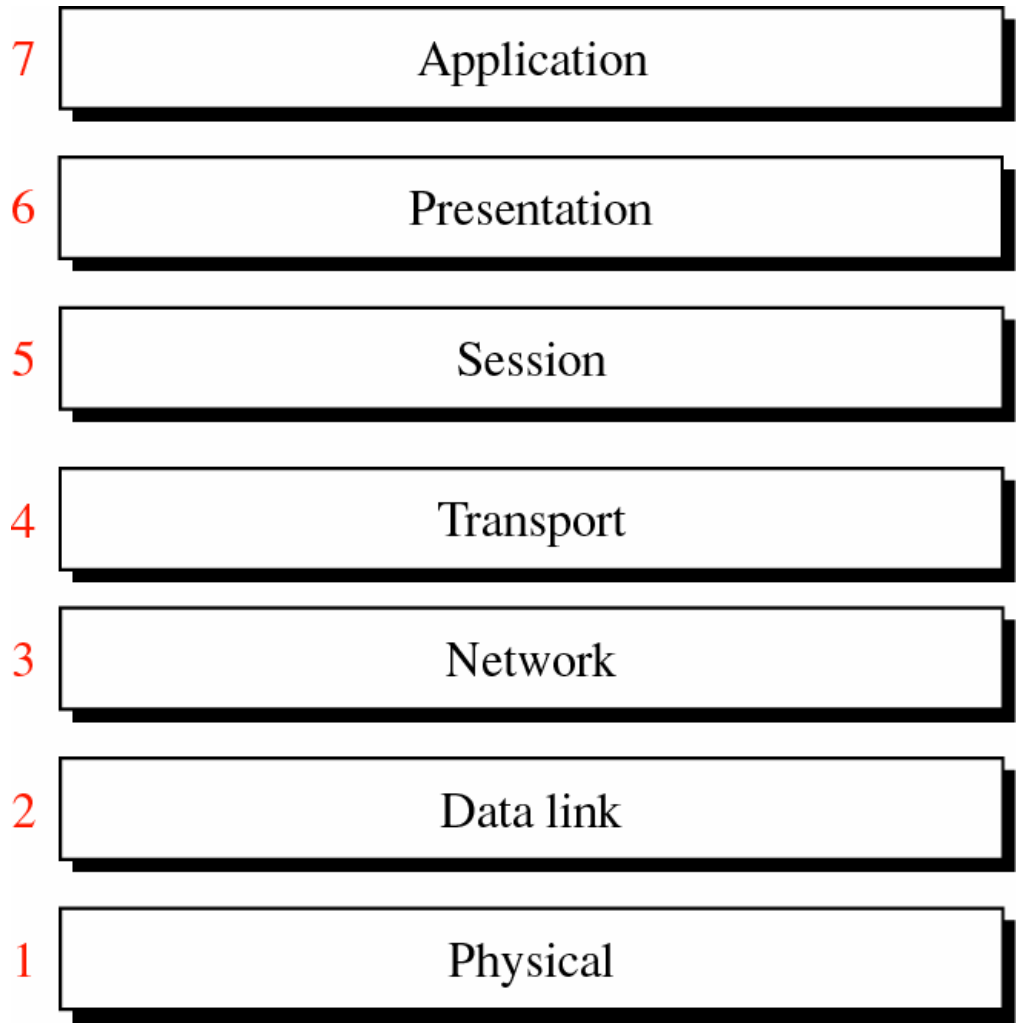
The OSI Model and TCP/IP Protocol Suite

Outline

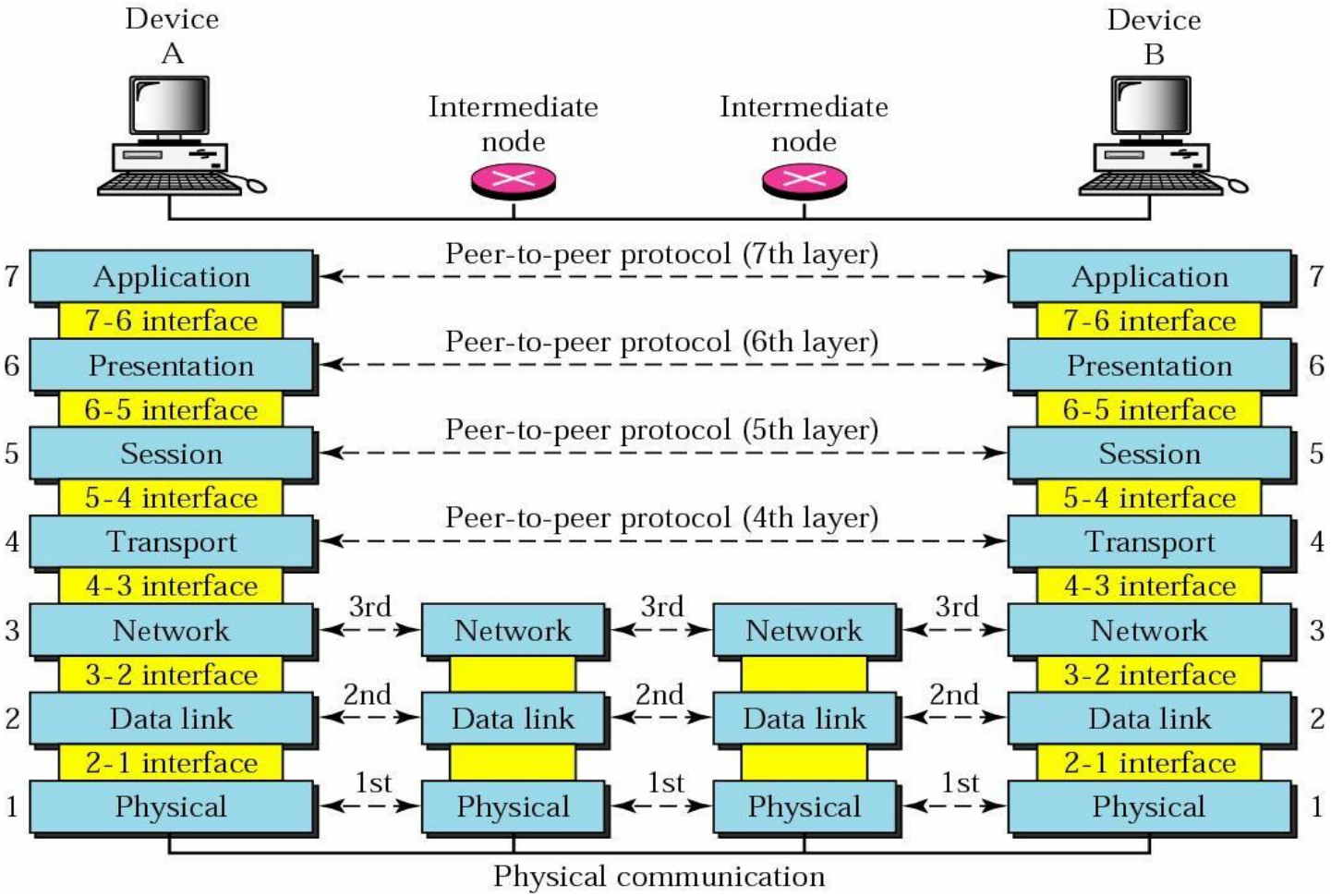
- THE OSI MODEL
- LAYERS IN THE OSI MODEL
- TCP/IP PROTOCOL SUITE
- ADDRESSING

THE Open-Source Interconnect MODEL

OSI Model


7
Application
6
Presentation
5
Session
4
Transport
3
Network
2
Data link
1
Physical

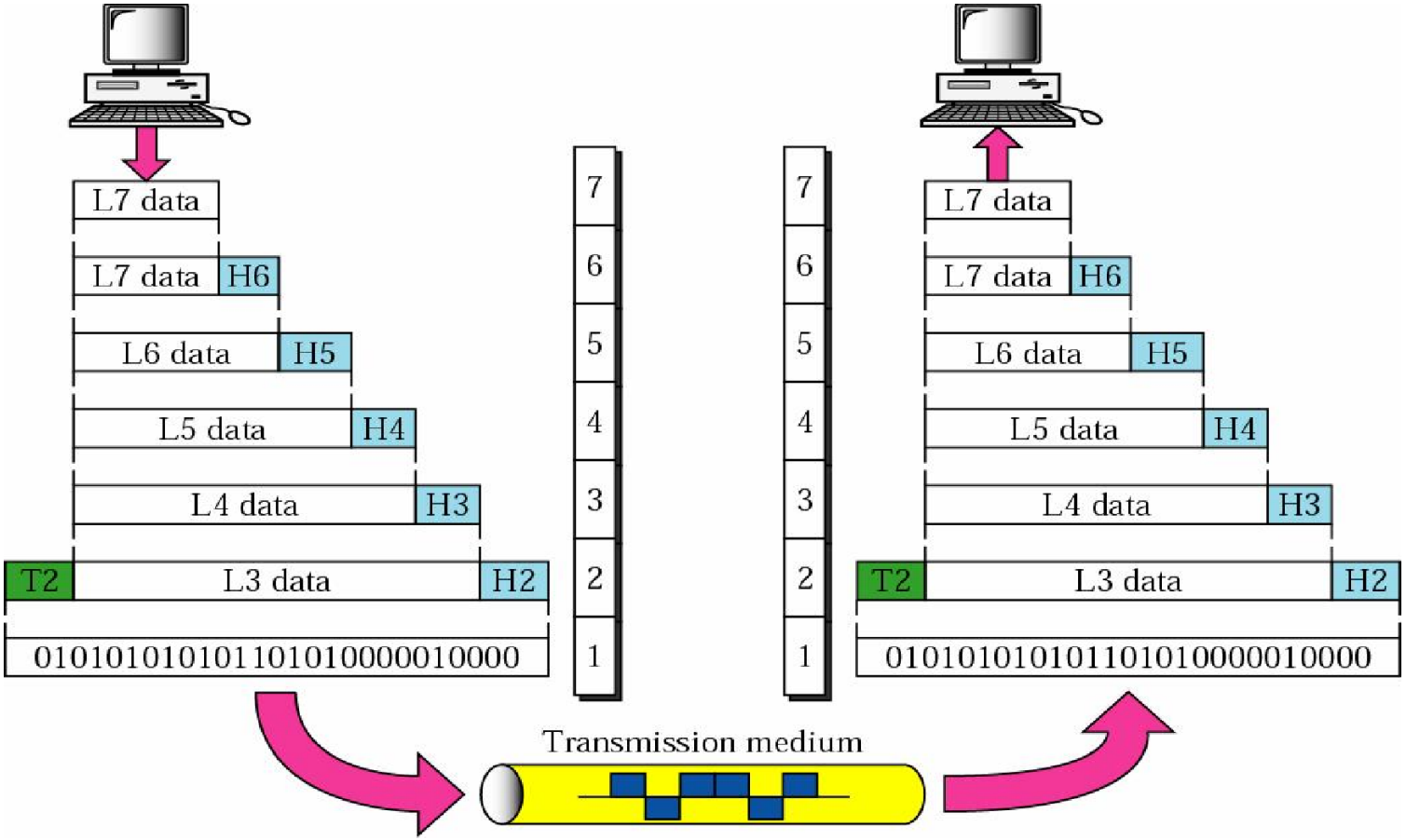
OSI Layers


Device A
Intermediate node
Intermediate node
Device B
Peer-to-peer protocol (7th layer)
Peer-to-peer protocol (6th layer)
Peer-to-peer protocol (5th layer)
Peer-to-peer protocol (4th layer)
7 Application
7-6 interface
6 Presentation
6-5 interface
5 Session
5-4 interface
4 Transport
4-3 interface
3 Network
3-2 interface
2 Data link
2-1 interface
1 Physical
3rd
2nd
1st
Physical communication

Peer-to-Peer Processes

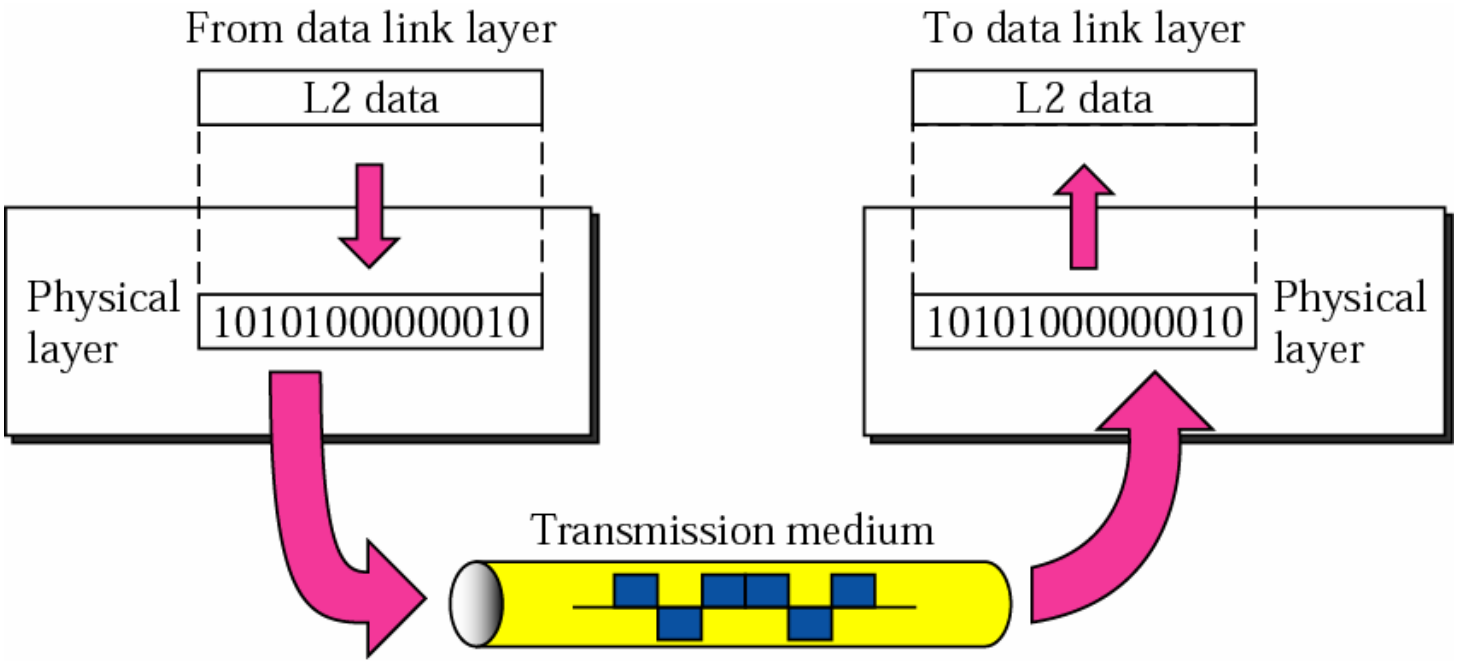
- The processes on each machine that communicate at a given layer are called ***peer-to-peer processes***
- Each layer in the sending device added its own information to the message it receives from the layer just above it
- At the receiving machine, the message is unwrapped layer by layer

An Exchange Using the OSI Model



LAYERS IN THE OSI MODEL

Physical Layer



Physical Layer

- Physical characteristics of interface and media
  - The type of transmission media
- Representation of bits
  - Define the type of ***encoding*** (how 0s and 1s are changed to signals)
- Data rate
  - The number of bits sent each second
- Synchronization of bits
  - The sender and receiver must not only use the same bit rate but must also be synchronized at the bit level

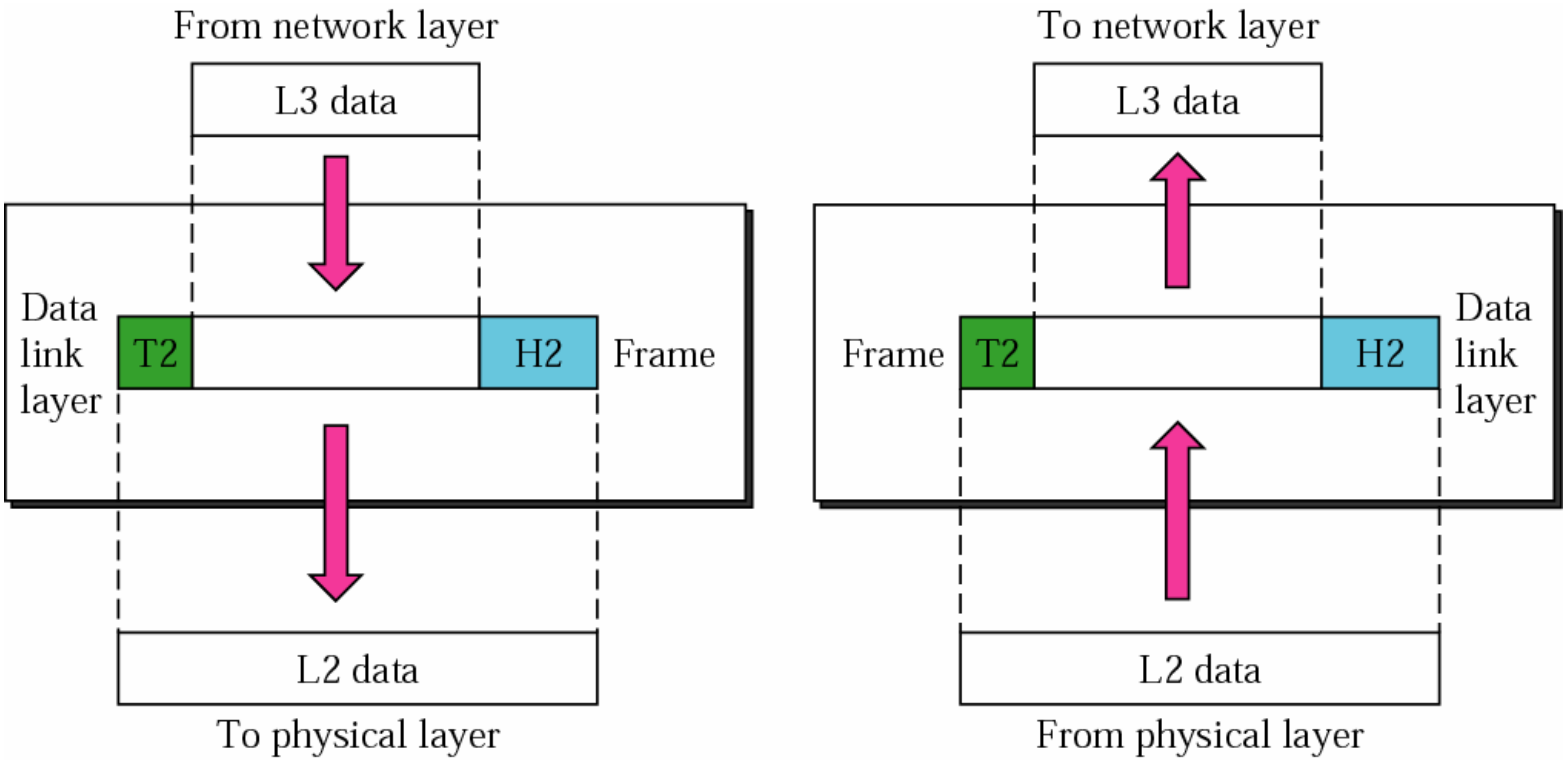
Physical Layer (Cont.)

- Line configuration
  - Point-to-point configuration
  - Multipoint configuration
- Physical topology
  - Define how devices are connected to make a network
  - For example, mesh, star, ring, or bus topology
- Transmission mode
  - Define the direction of transmission between two devices
  - Half-duplex mode
    - two devices can send and receive but not at the same time
  - Full-duplex mode
    - Two devices can send and receive at the same time

Data Link Layer

- Transform the physical layer to a reliable link
- Make the physical layer appear error free to the upper layer

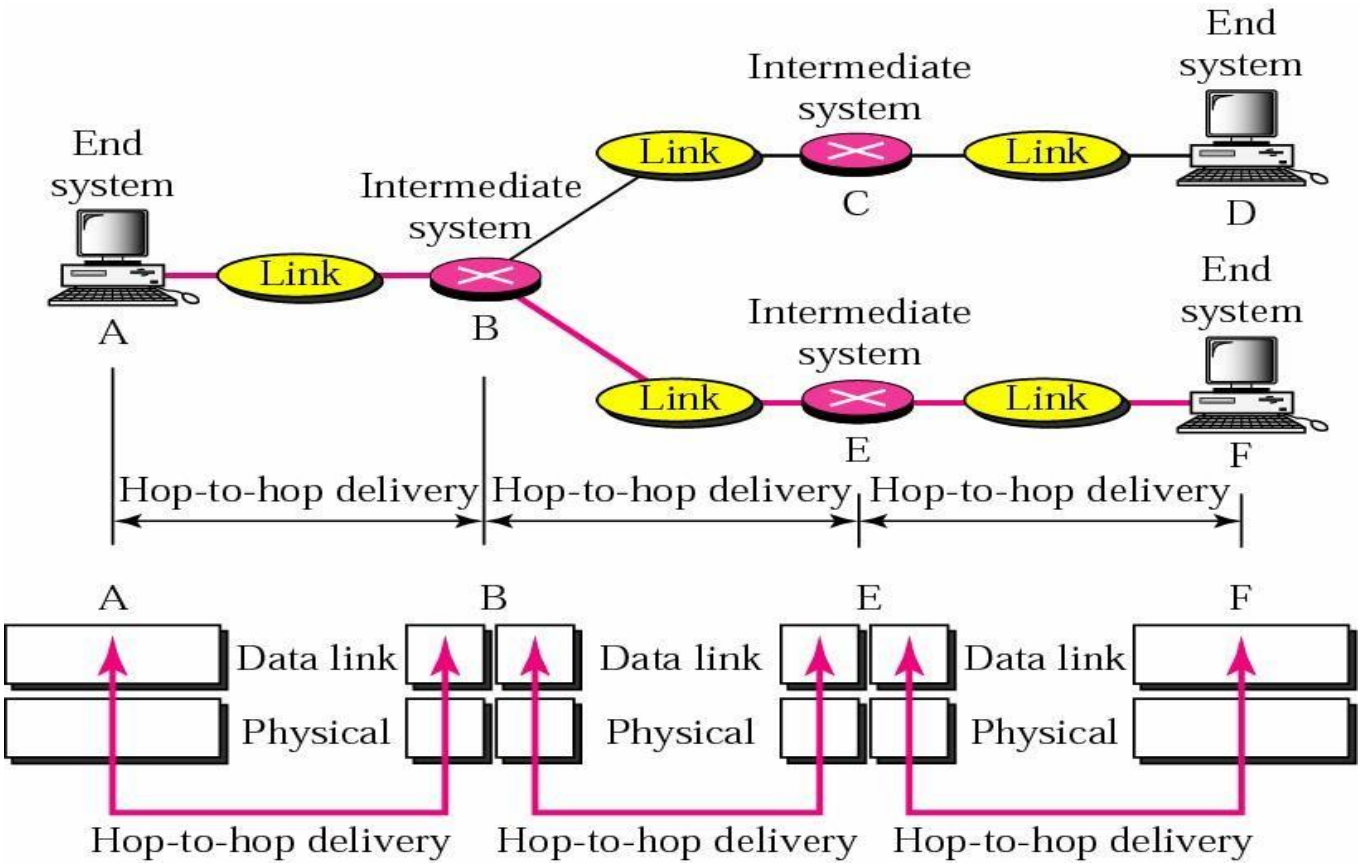
Data Link Layer



Data Link Layer

- Framing
  - Divide the stream of bits into frames
- Physical addressing
  - only one hop
- Flow control
- Error control
  - Detect and retransmit damaged or lost frames
  - Prevent duplication of frames

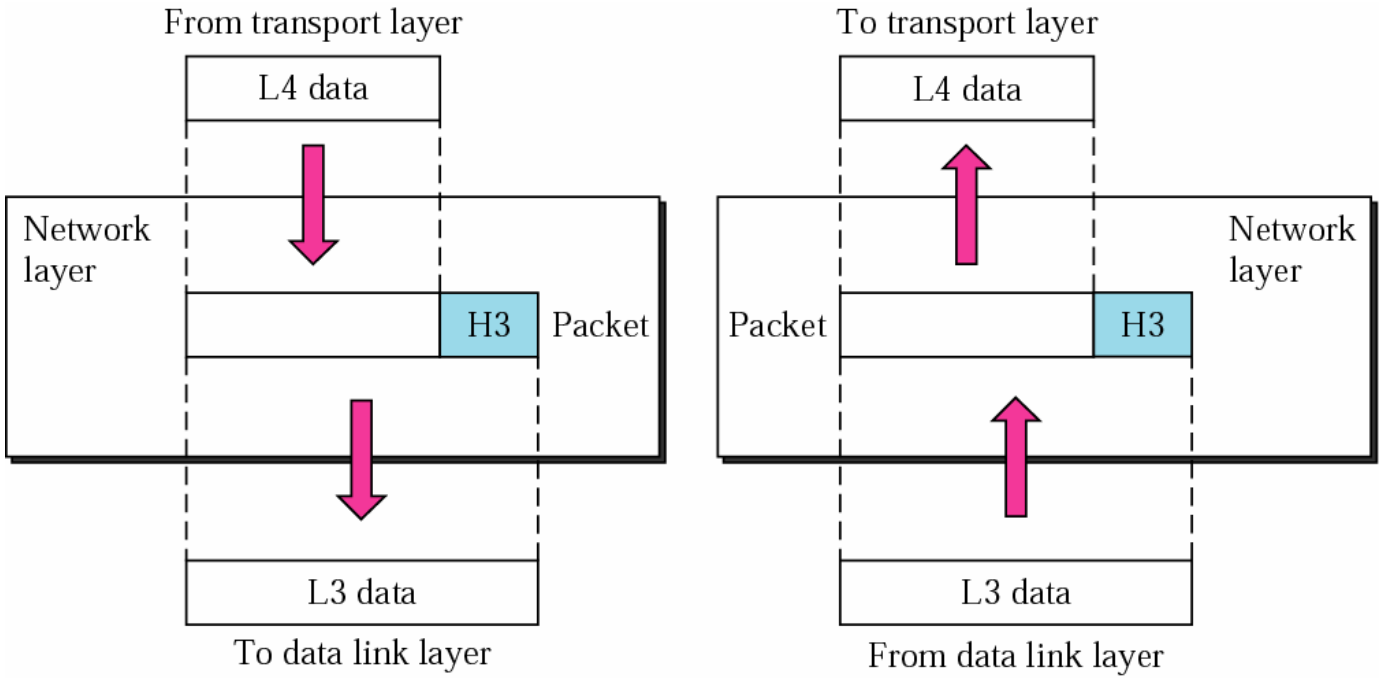
Node-to-Node (Hop-to-Hop) Delivery



Network Layer

- Be responsible for the source-to-destination delivery of a packet possibly across multiple networks (links)
  - Data link layer take care of the delivery of the packet between two systems on the same network (links)
  - Network layer ensures the delivery of a packet from its point of origin to it final destination

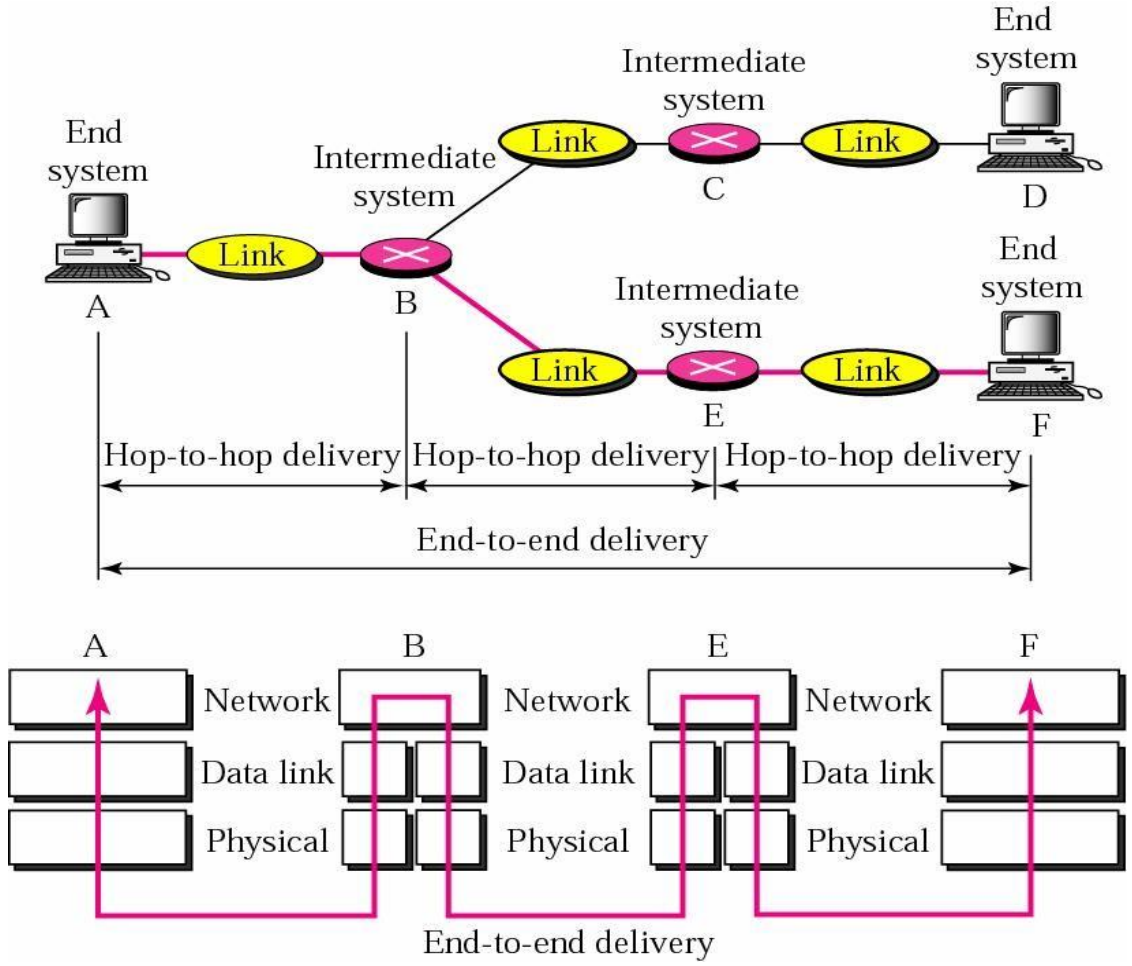
Network Layer



Network Layer

- Logical addressing
  - Physical addressing in the data link layer handles the addressing problem locally
  - If a packet pass the network boundary
    - We need another addressing system to distinguish the source and destination systems
- Routing

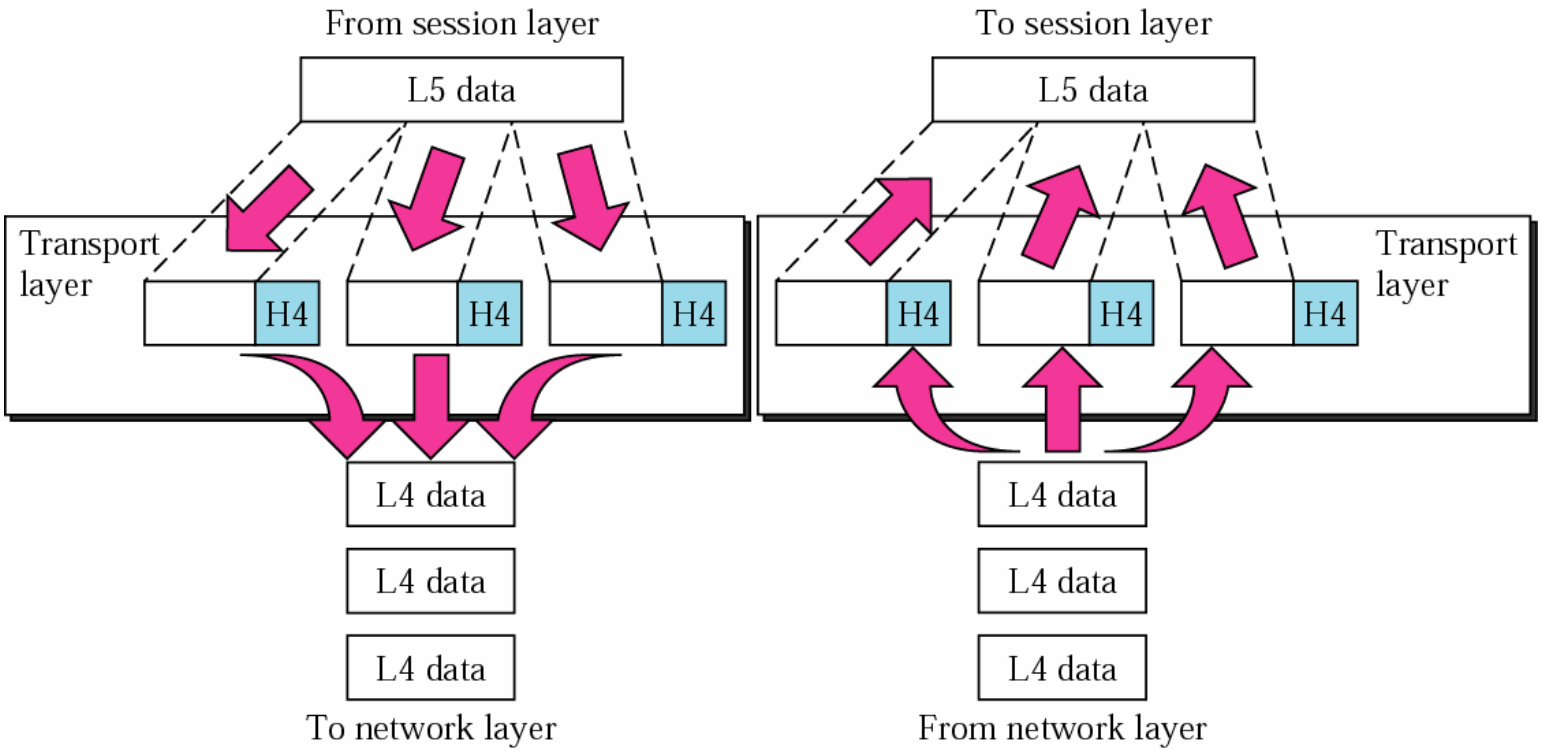
End-to-End Delivery by the Network Layer



Transport Layer

- Be responsible for source-to-destination (end- to-end) delivery of the entire message
  - Network layer only oversees the end-to-end delivery of individual packets
- Oversee both error control and flow control at the source-to-destination level

Transport Layer



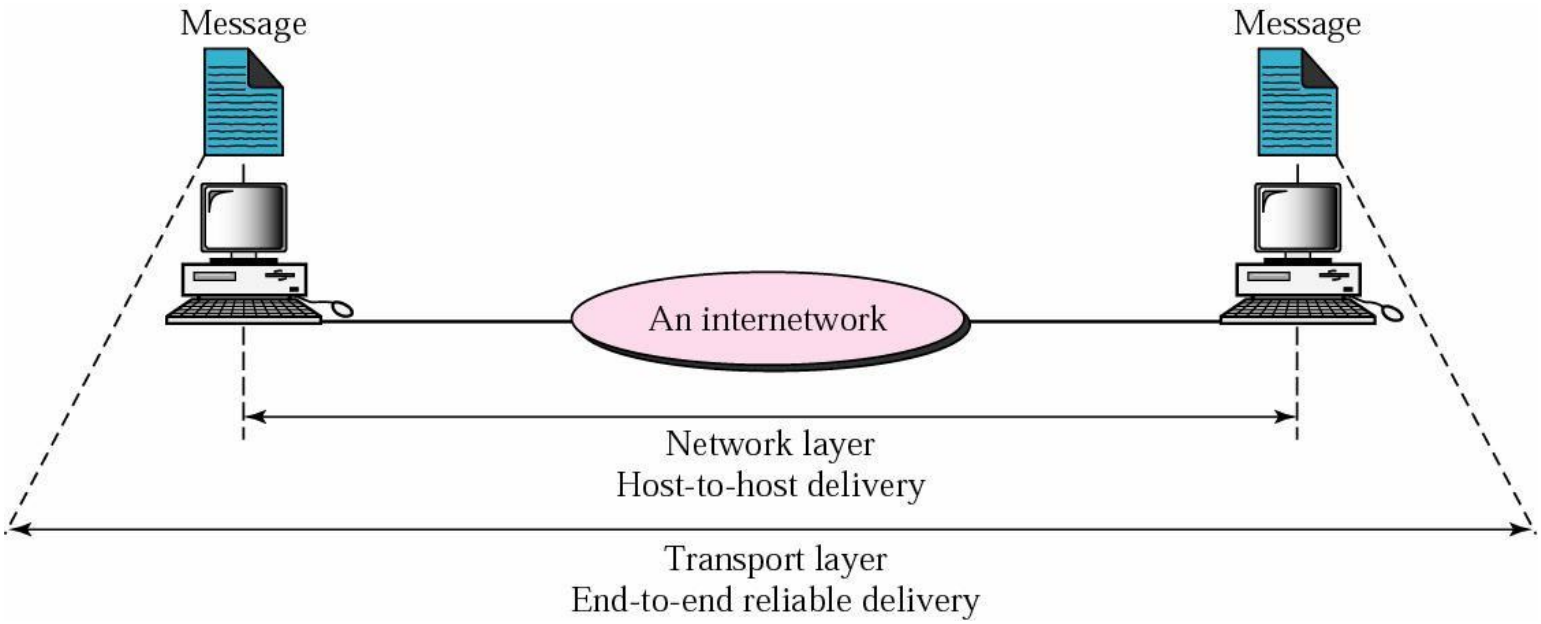
Transport Layer

- Service-point addressing
  - Service-point address: port address
  - The network layer gets each packet to the correct computer
  - The transport layer gets the entire message to the correct process on that computer
- Segmentation and reassembly

Transport Layer (Cont.)

- Connection control
  - Connectionless
  - Connection-oriented
- Flow control
  - End to end flow control
  - The flow control, however, at the data link layer is across a single link
- Error control
  - End to end error control that makes sure that the entire message arrives without error (damage, loss, or duplication)

Reliable End-to-End Delivery of a Message

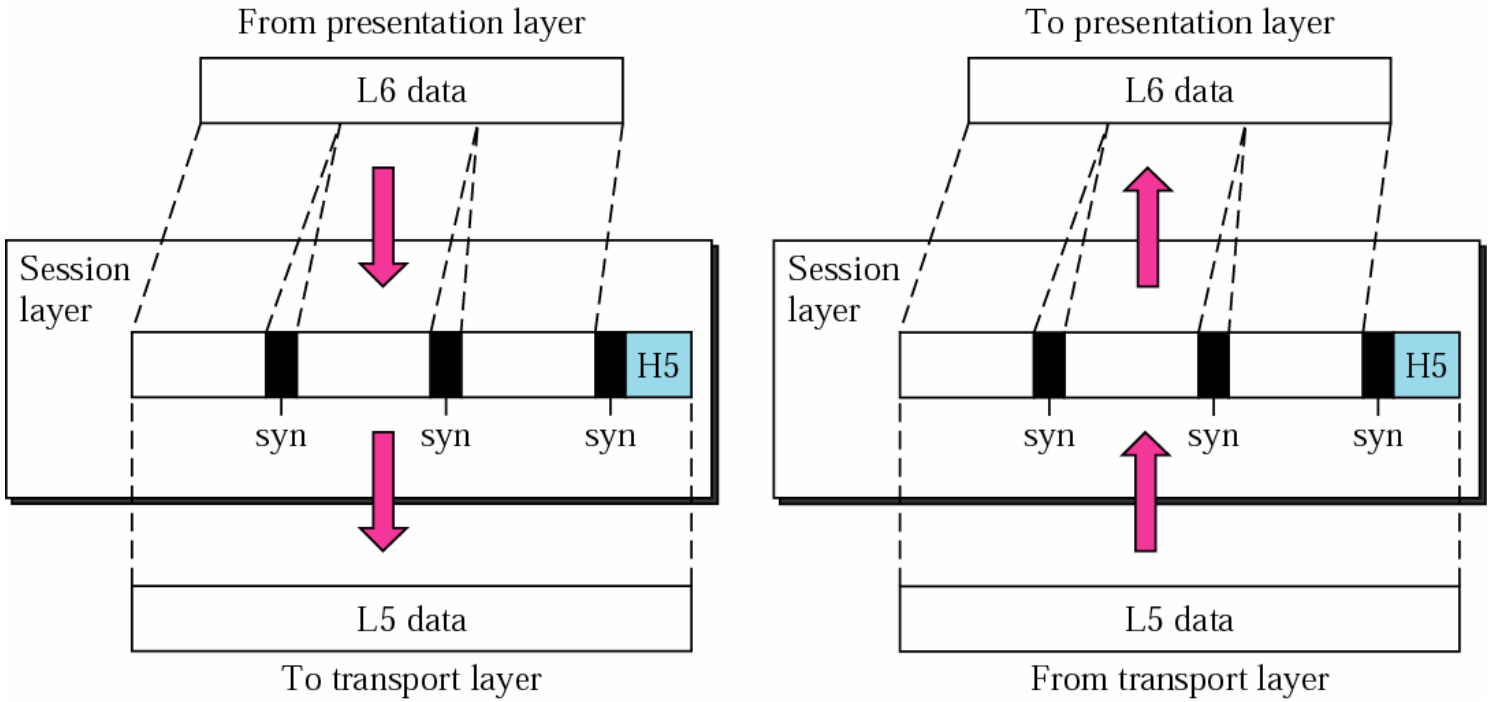


end-to-end delivery by the transport layer

Session Layer

- Session layer is the network dialog controller
  - It establishes, maintains, and synchronizes the interaction between communication systems

Session Layer



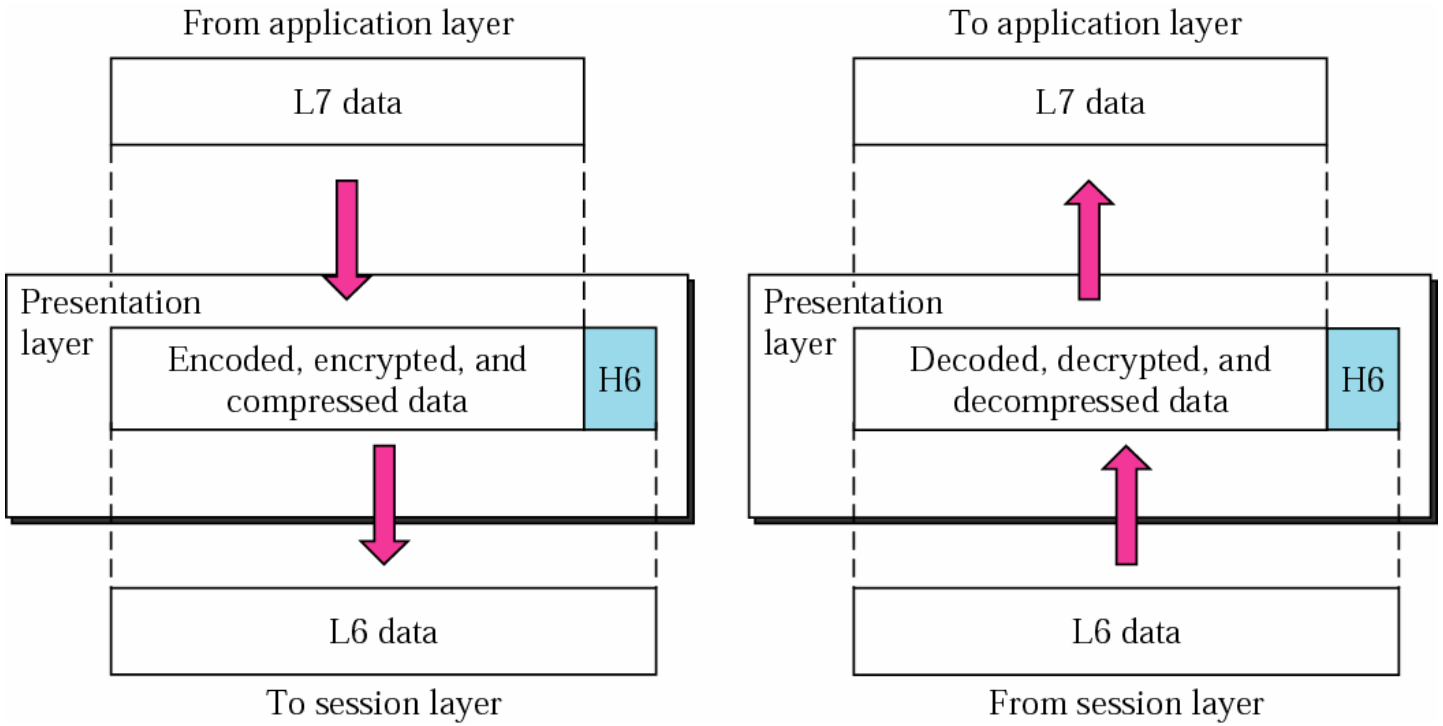
Session Layer

- Dialog control
  - Allow two systems to enter a dialog
  - Allow the communication between two processes in
    - Half-duplex
    - Full-duplex
- Synchronization
  - Allow a process to add checkpoints (synchronization points) into a stream of data

Presentation Layer

- Be concerned with the syntax and semantics of the information exchanged between two systems

Presentation Layer



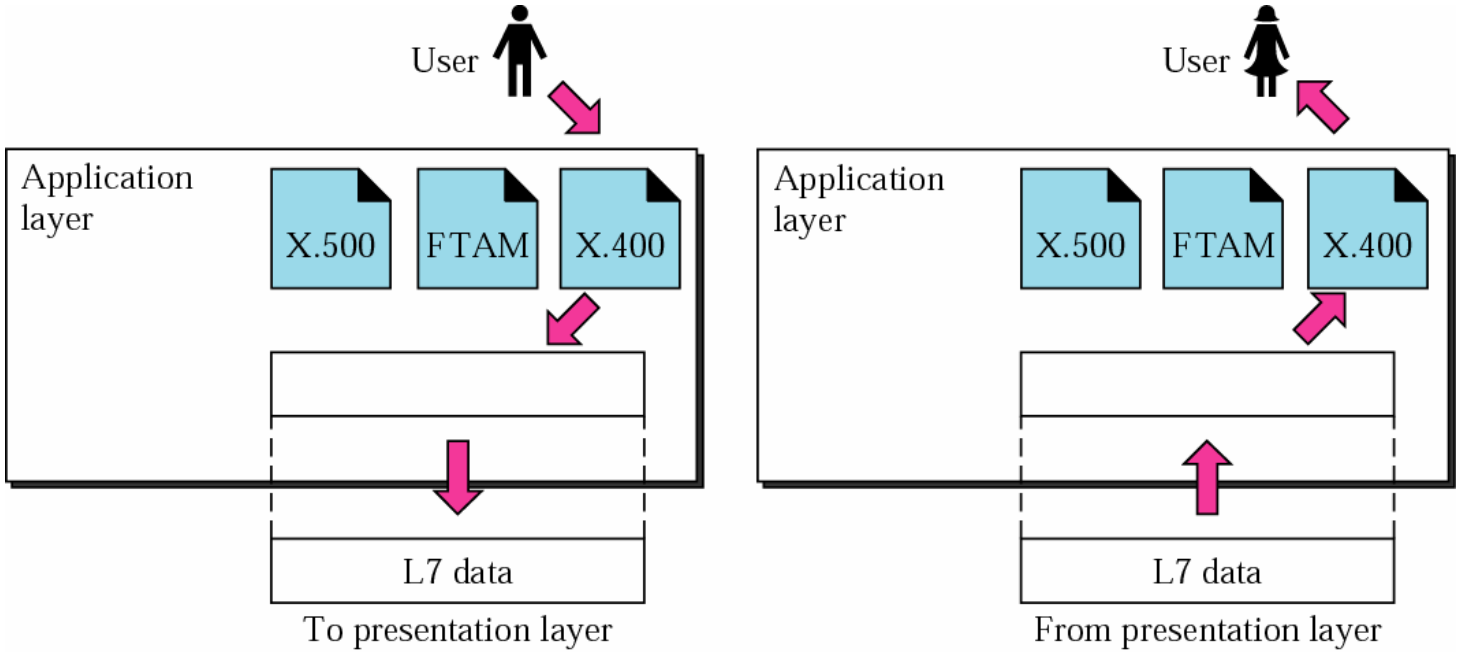
Presentation Layer

- Translation
  - Different computers use different encoding systems
  - The presentation layer is responsible for interoperability between these different encoding methods
- Encryption
- Compression

Application Layer

- The application layer enables the user, whether human or software, to access the network

Application Layer



Application Layer

- Specific services provided by the application layer
  - Network virtual terminal
  - File transfer, access, and management (FTAM)
  - Mail services
  - Directory services

Summary of Layers

OSI (Open Source Interconnection) Model

Layers	Purpose / Function	Protocol Used	Fundamental Data Type
7. Application	User Application Layer: browser, ftp, app, etc. (remote file access, resource sharing, LDAP, SNMP)	SMTP FTP	Data
6. Presentation	Syntax Layer: encrypt, compress (optional) (data encrypt/decrypt, codec, translation)	JPEG, ASCII, ROT13	Data
5. Session Layer	Synchronization & Logical Port Routing (session establishment, start & terminate, security, logging, name recognition)	RPC, NFS, NetBIOS	Data
4. Transport Layer	TCP: Host to Host & Flow Control (end to end connections & reliability, message segmentation, acknowledgment, session multiplexing)	TCP / UDP	TCP: Segments UDP: Datagrams
3. Network Layer	Packets: IP Address (path determination, logical addressing, routing, traffic control, frame segmentation, subnet management)	IP, IPX, ICMP	Packets
2. Data Link Layer	Data Frames: MAC address, packet (physical addressing, Media Access Control, LLC. frame error checking, sequencing and reordering)	PPP/SLIP	Frames
1. Physical Layer	Physical Device: Cables, fibers, RF spectrum (data encoding, media attachment, baseband/broadband, signaling, binary transmission)	Coax, fiber, wireless	Bits / Signals

TCP/IP PROTOCOL SUITE

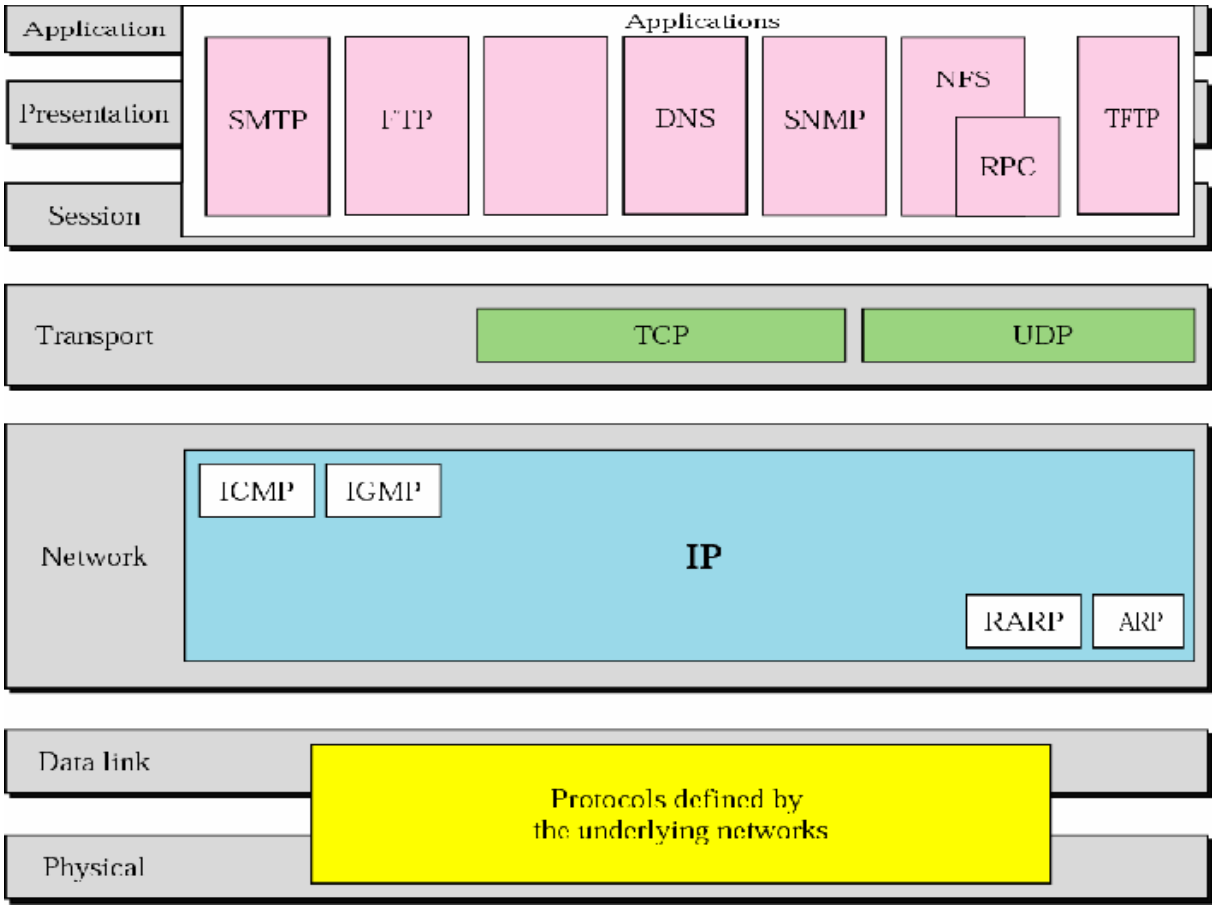
TCP/IP Protocol Suite

- TCP/IP protocol suite was developed prior to the OSI model and is made of five layers
  - Physical, data link, network, transport, and application layers
- The first four layers correspond to the first four layer of the OSI model

TCP/IP Protocol Suite

- The three topmost layers is OSI model are represented in TCP by a single layer called the *application layer*

TCP/IP and OSI Model



Physical and Data Link Layers

- TCP/IP does not define any specific protocol
- TCP/IP supports all of the standard and proprietary protocols

Network Layer

- Internetworking Protocols (IP)
- Four supporting protocols
  - ARP
  - RARP
  - ICMP
  - IGMP

Internetworking Protocol (IP)

- The packets in IP are called *datagrams*
- Unreliable and connectionless datagram protocol
- Best-effort delivery services
  - Provide no error checking or tracking

Address Resolution Protocol (ARP)

- Associate an IP address with the physical address
  - Data in LAN are transmitted by the physical address

Reverse Address Resolution Protocol (RARP)

- Discover a host's internet address via its physical address
- Used when
  - A computer is first connected to the network
  - When a diskless computer is booted

Internet Control Message Protocol (ICMP)

- A mechanism used to send notification of datagram problems back to the sender
- ICMP sends query and error reporting messages

Internet Group Message Protocol (IGMP)

- Facilitate the simultaneous transmission of a message to a group of recipients

Transport Layer

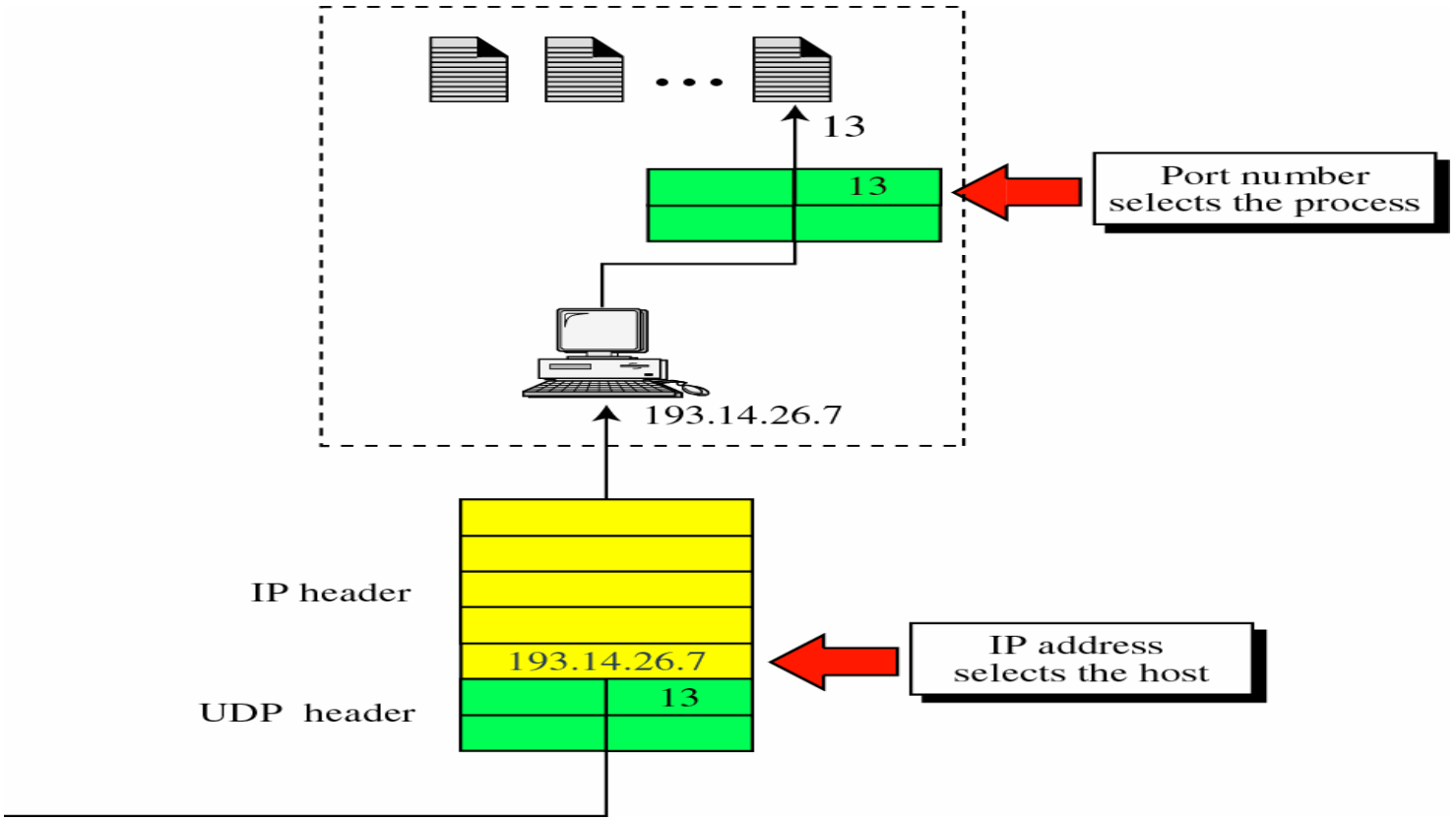
- TCP/UDP
  - A process-to-process protocol
- IP
  - A host-to-host protocol

User Datagram Protocol (UDP)

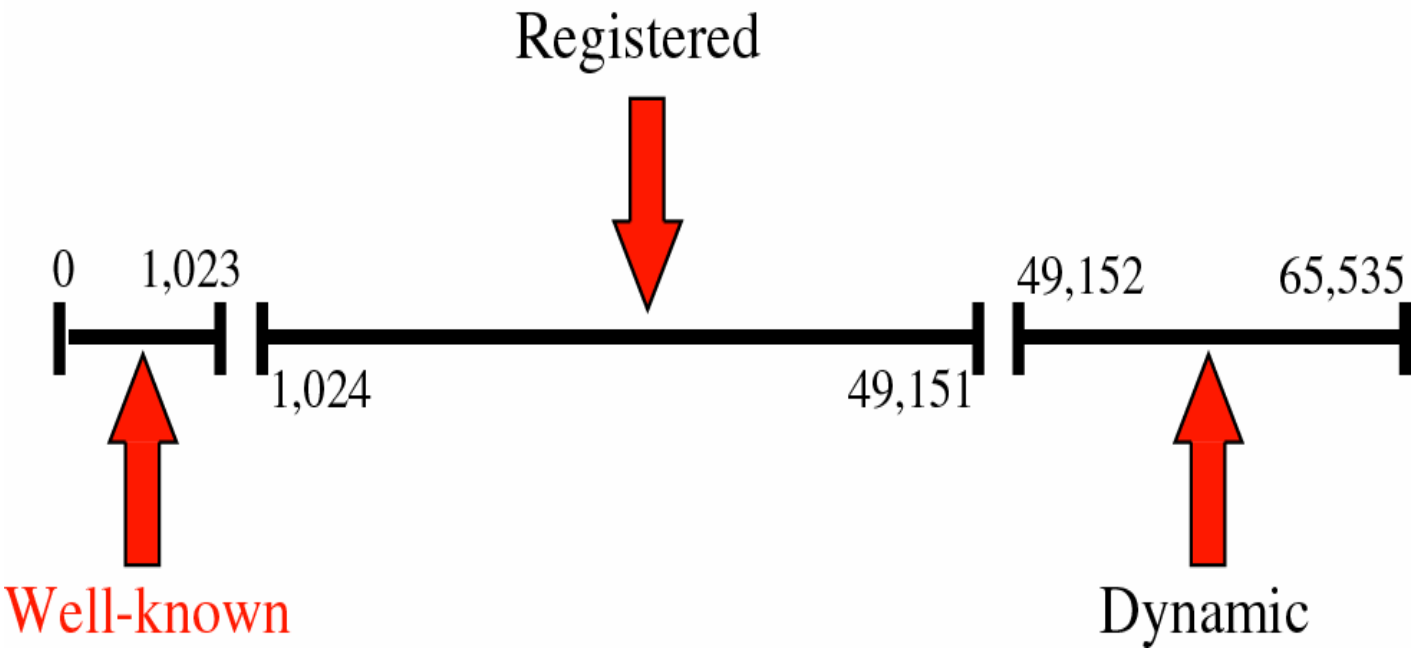
User Datagram Protocol (UDP)

- Add only
  - Port numbers
  - Checksum error control
  - length

IP Addresses Versus Port Numbers



IANA Ranges


Registered
0
1,023
1,024
49,151
49,152
65,535
Well-known
Dynamic

Well-Known Ports for UDP

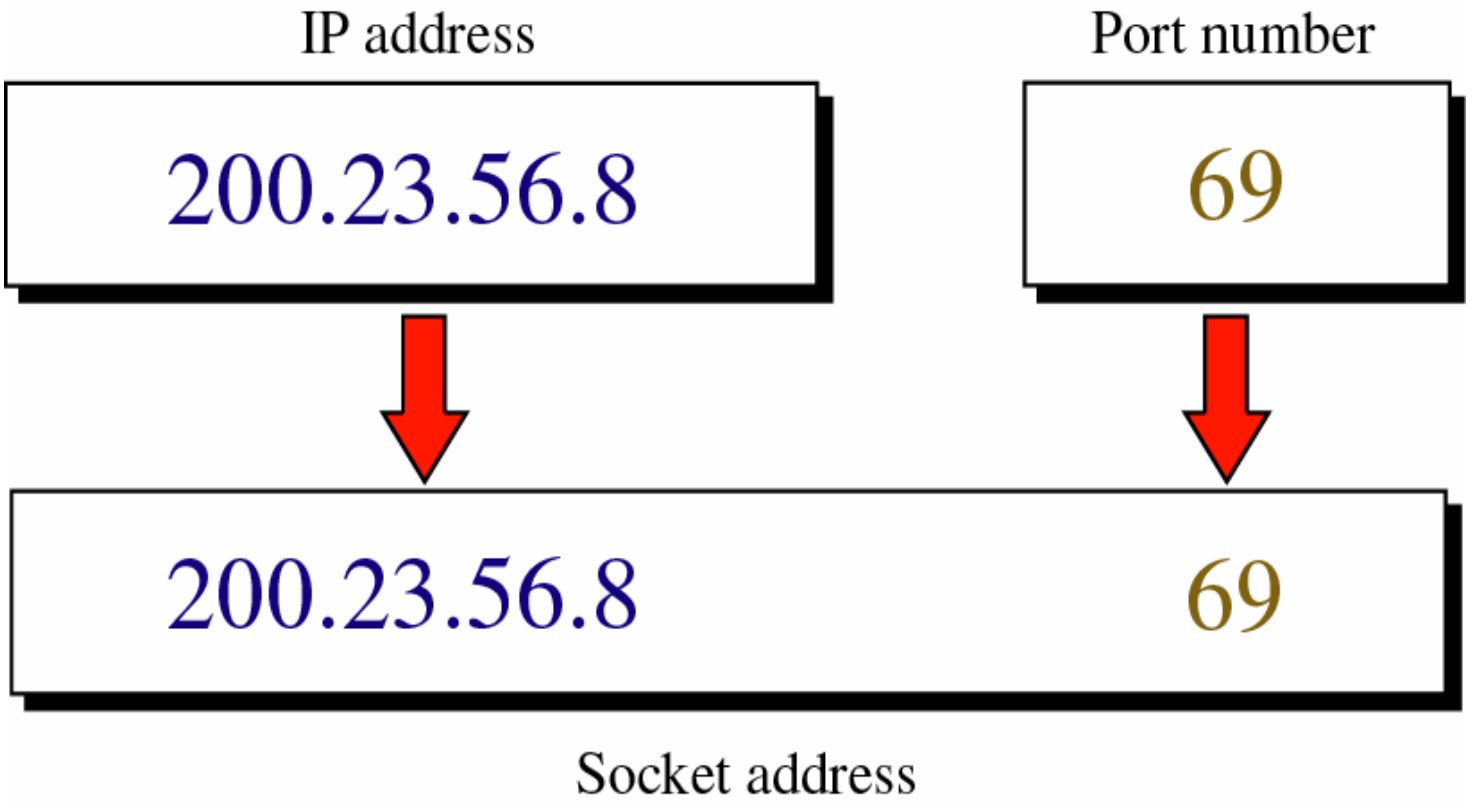
Port	Protocol	Description
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Return the data and the time
17	Quote	Returns a quote of a day
19	Chargen	Return a string of characters
53	Nameserver	Domain Name Service
67	Bootps	Server port to download bootstrap information
68	Bootpc	Client port to download bootstrap information

Socket Addresses

- UDP needs two identifiers
  - The IP address
  - The port number

- Socket address
  - The combination of an IP address and a port number

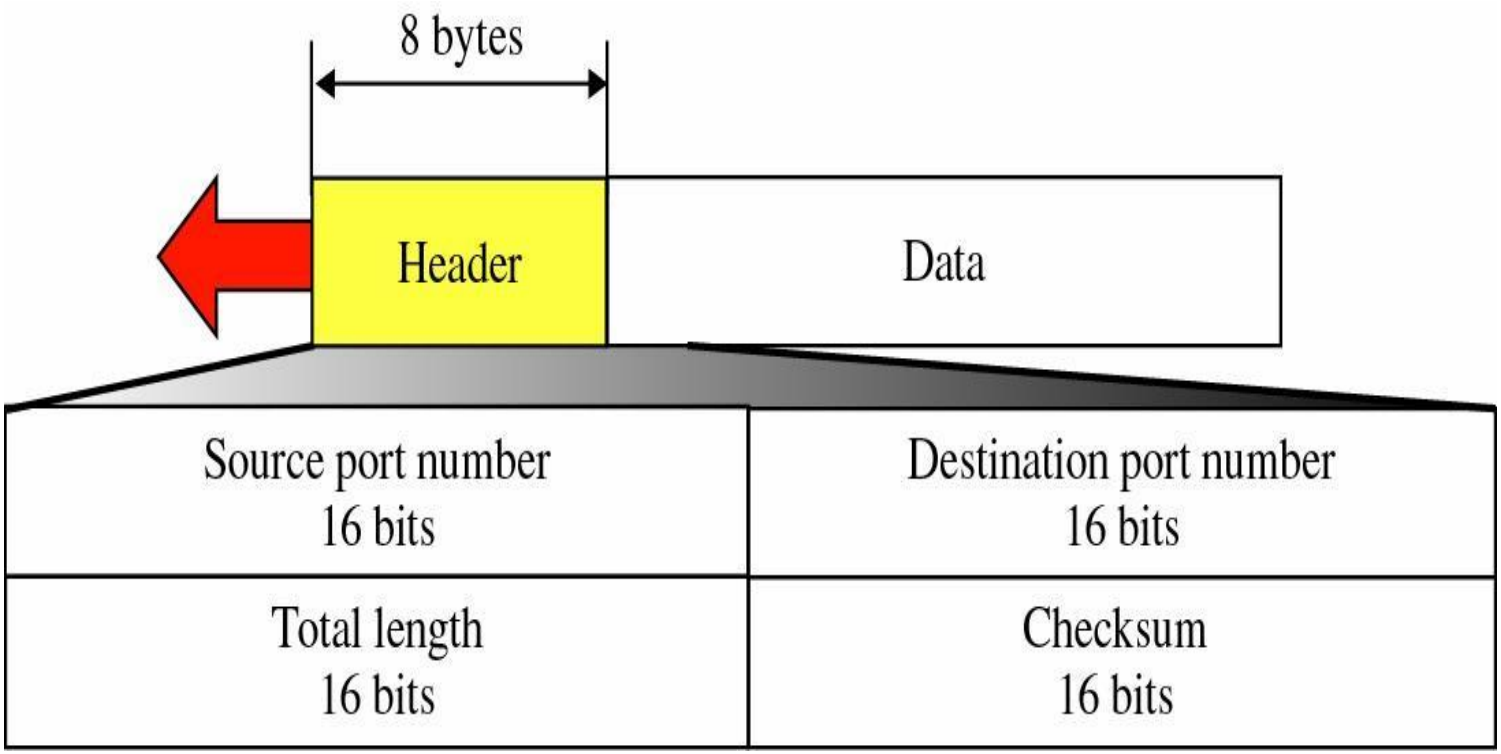
Socket Addresses



User Datagram

- UDP packets: called *user datagrams*
  - Have a fixed-size header of 8 bytes
- The fields are
  - Source port number: 16-bits: 0~65535
    - Usually chosen by the UDP software
  - Destination port number: 16-bits: 0~65535
  - Length: 16 bits:
    - Total length (*header plus data*) of the user datagram
    - Minimum length must be 8 bytes (contains only header)
  - Checksum: 16 bits
    - Detect errors over the entire user datagram (*header plus data*)

User Datagram Format



UDP Operation

- Connectionless Services
- Flow and Error Control
- Encapsulation and Decapsulation
- Queuing

Connectionless Services

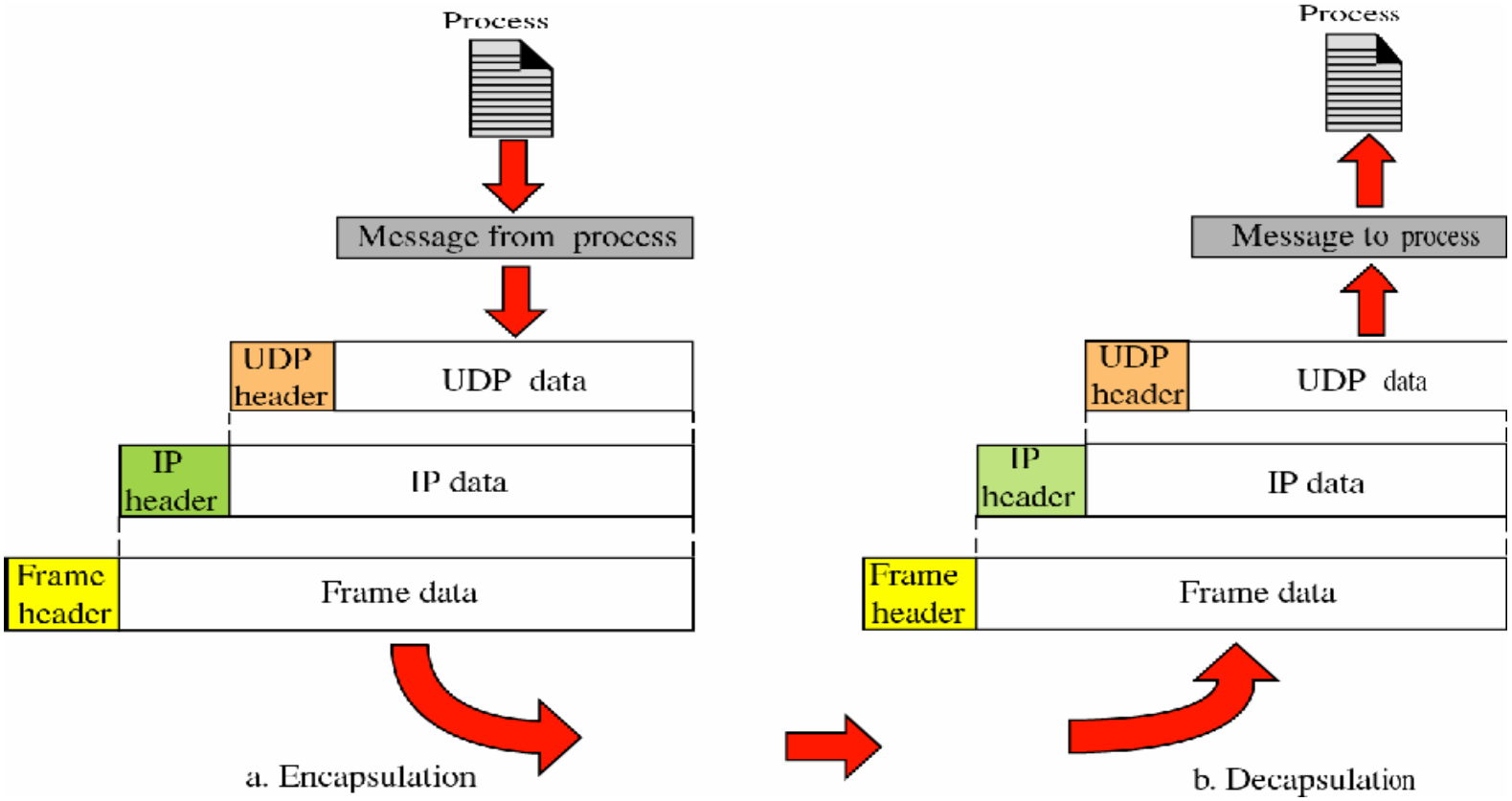
- Each user datagram is an independent datagram
  - Even coming from the same source process
- The user datagram is not numbered

- No connection establishment and no connection termination

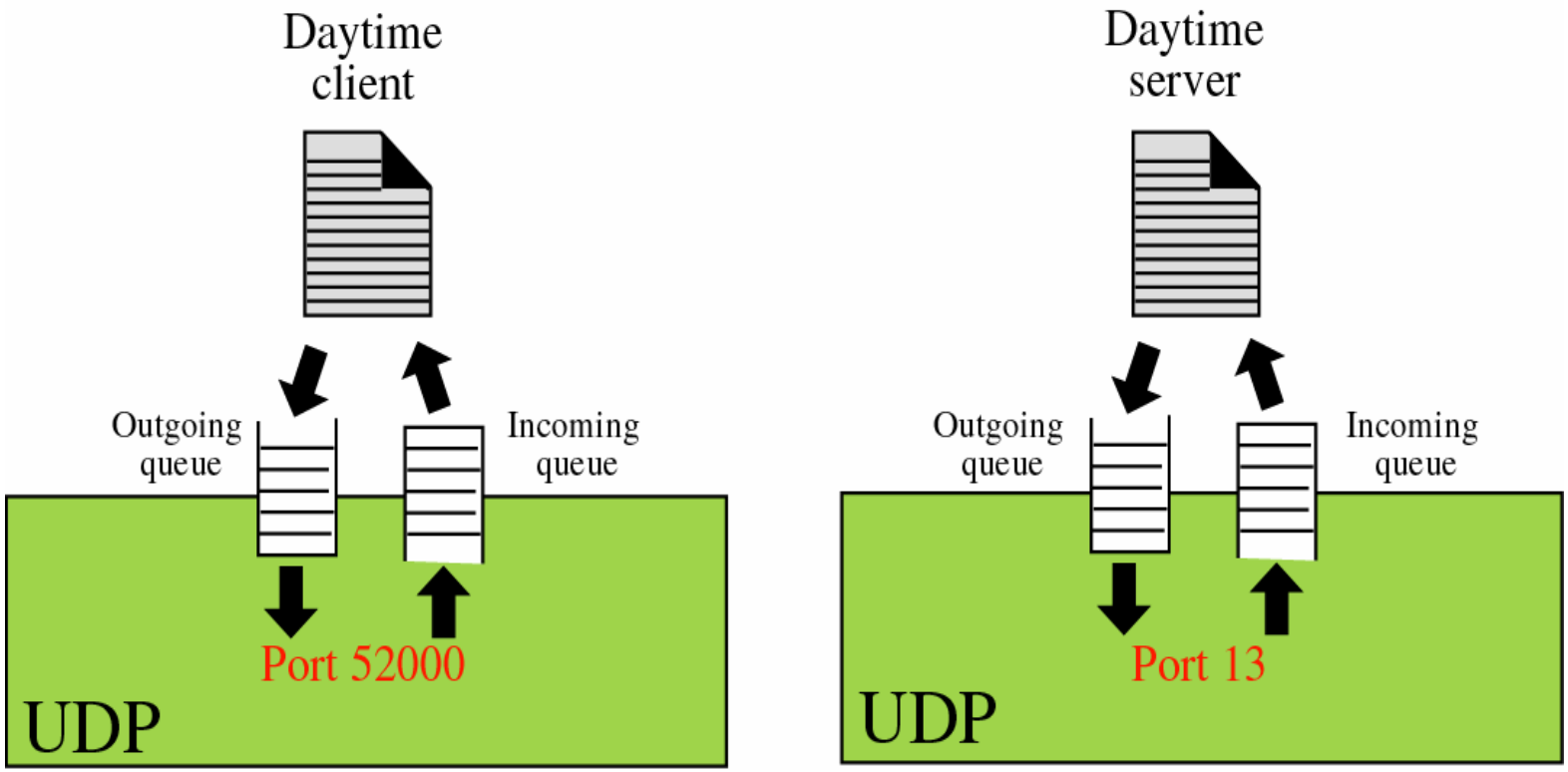
Flow and Error Control

- No flow control
  - The receiver may overflow with incoming messages
- No error control except for the checksum
  - The sender does not know if a message has been lost or duplicated

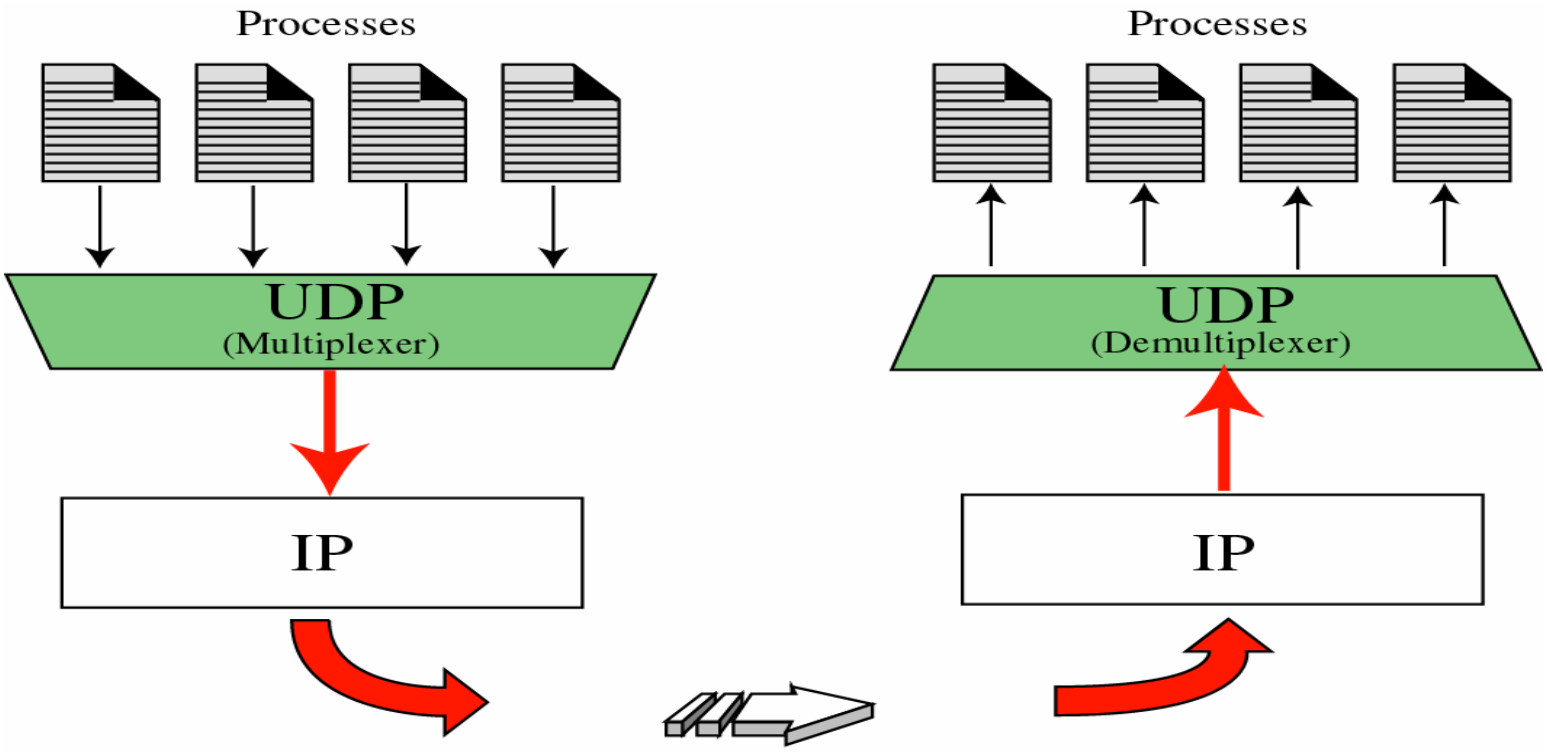
Encapsulation and Decapsulation



Queues in UDP



Multiplexing and Demultiplexing



Uses of UDP

- For a process that require simple request-response communication with little concern for flow and error control
- For a process with internal flow and error-control
  - TFTP (Trivial File Transfer Protocol)
- For muticasting and broadcasting
  - Embedded in UDP but not in the TCP software
- For some management processes
- For some route updating protocol
  - RIP (Routing Information Protocol)

Transmission Control Protocol (TCP)

Introduction

- TCP
  - Like UDP, TCP provides process-to-process communication using Port numbers
  - A connection-oriented protocol
    - Create a virtual connection between two TCPs to send data
  - Add flow and error-control mechanisms at the transport layer
    - For flow control: TCP uses a *sliding window protocol*
    - For error control: TCP uses the *acknowledge packet*, *time-out*, and *retransmission* mechanisms

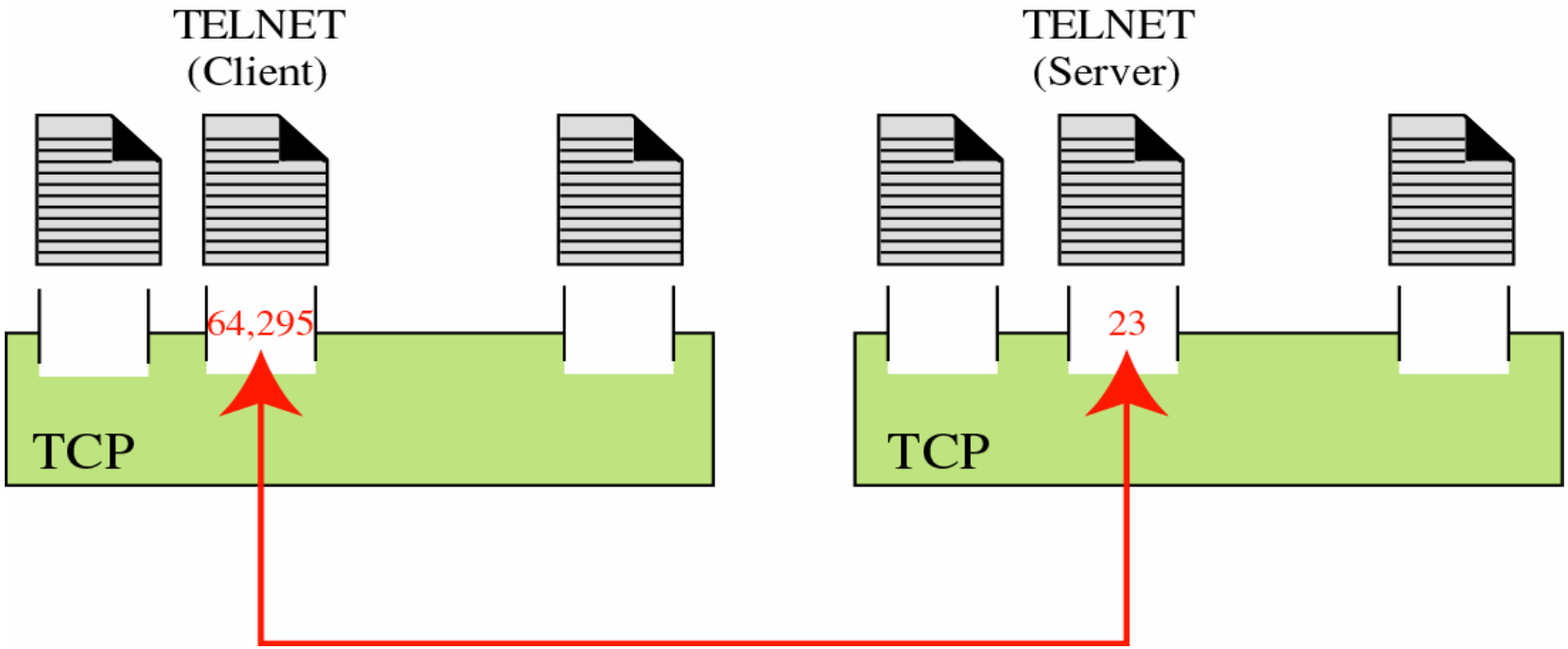
TCP Services

- TCP provides services to the processes at the application layer
  - Process-to-Process Communication
  - Stream Delivery Service
  - Full-Duplex Service
  - Connection-Oriented Service
  - Reliable Service

Port Number

- Client's port number
  - Chosen randomly by the TCP software running on the local host
  - Called *ephemeral(temporary) port number*
- Server's port number
  - Define itself with a port number
  - Called *well-known port numbers*

Port Numbers



Well-Know Ports Used by TCP

Port	Protocol	Description
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytimes	Return the data and the time
17	Quote	Return a quote of the day
19	Chargen	Return a string of characters
20	FTP, Data	File Transfer Protocol (data connection)
21	FTP, Data	File Transfer Protocol (control connection)

Example 1

In UNIX, the well-known ports are stored in a file called /etc/services. Each line in this file gives the name of the server and the well-known port number. We can use the grep utility to extract the line corresponding to the desired application. The following shows the ports for FTP.

```
$ grep  ftp   /etc/services

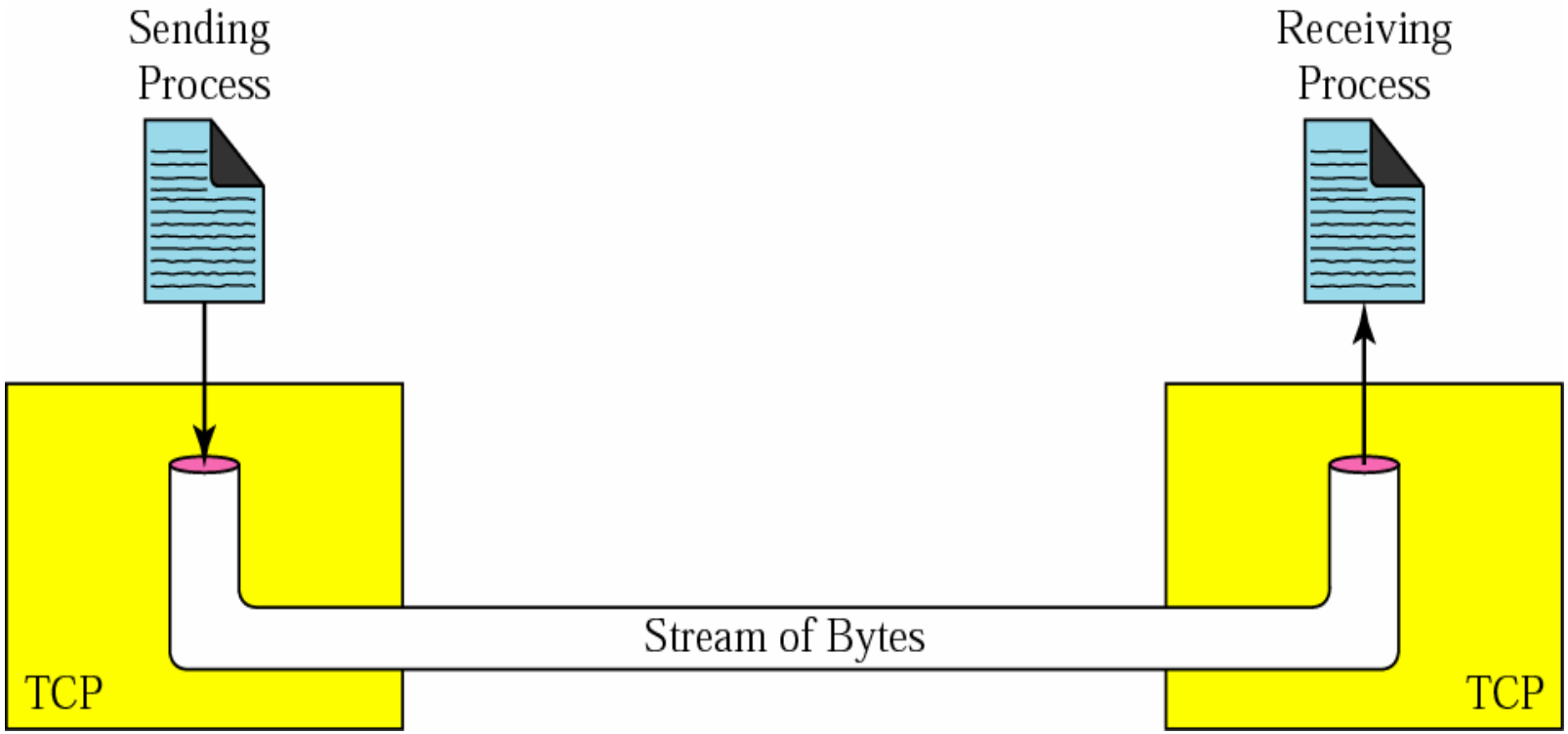
ftp-data        20/tcp
ftp-control     21/tcp
```

Stream Delivery Service

- UDP treats each chunk independently
  - No any connection between the chucks

- In contrast, TCP allow the data be delivered/received as a stream of bytes

Stream Delivery

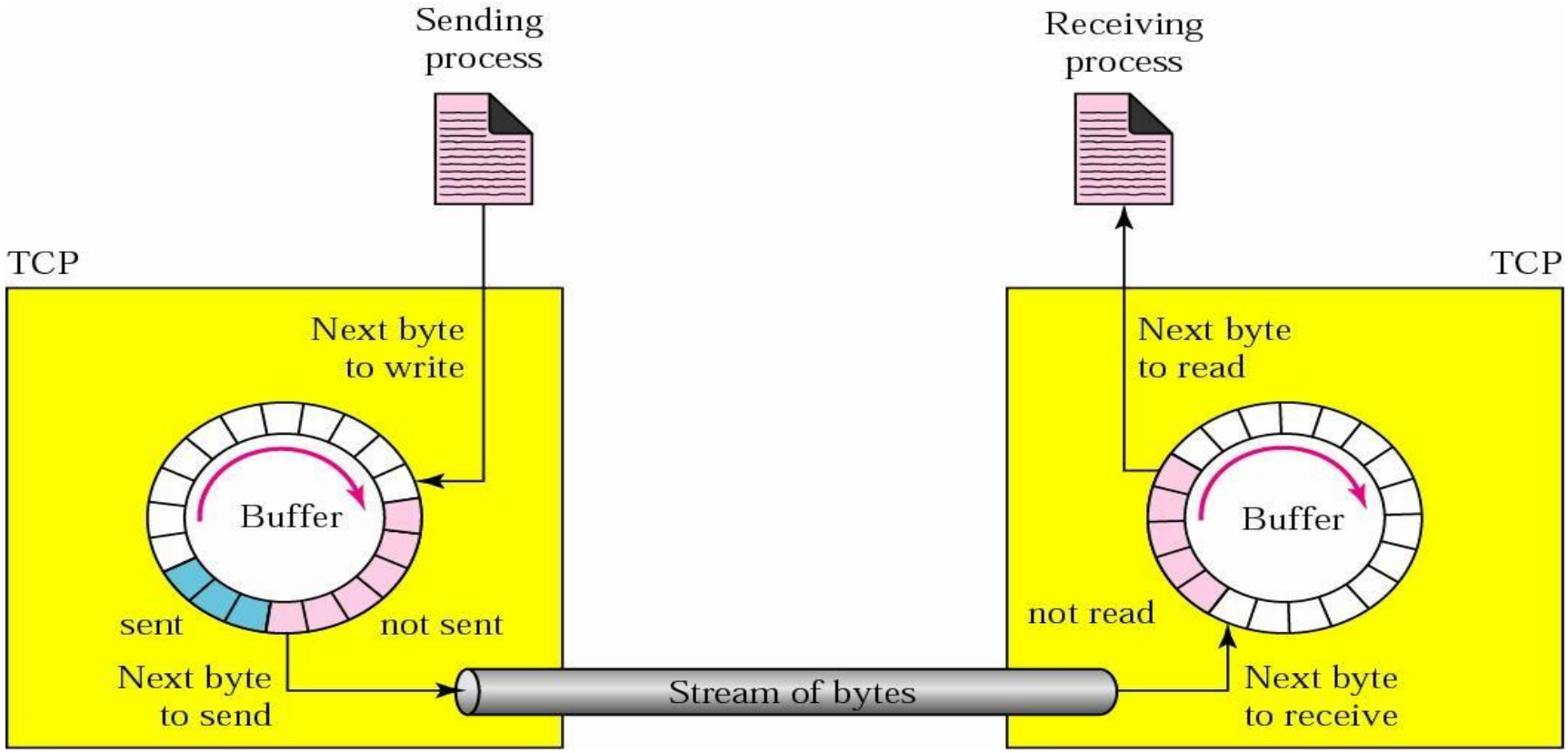


Stream Delivery Service (Cont.)

- However, the sending and receiving speed may not be the same
  - TCP needs buffers for storage

- Two buffers in TCP
  - Sending buffer and receiving buffer, one for each connection
  - Also used in flow- and error-control mechanisms

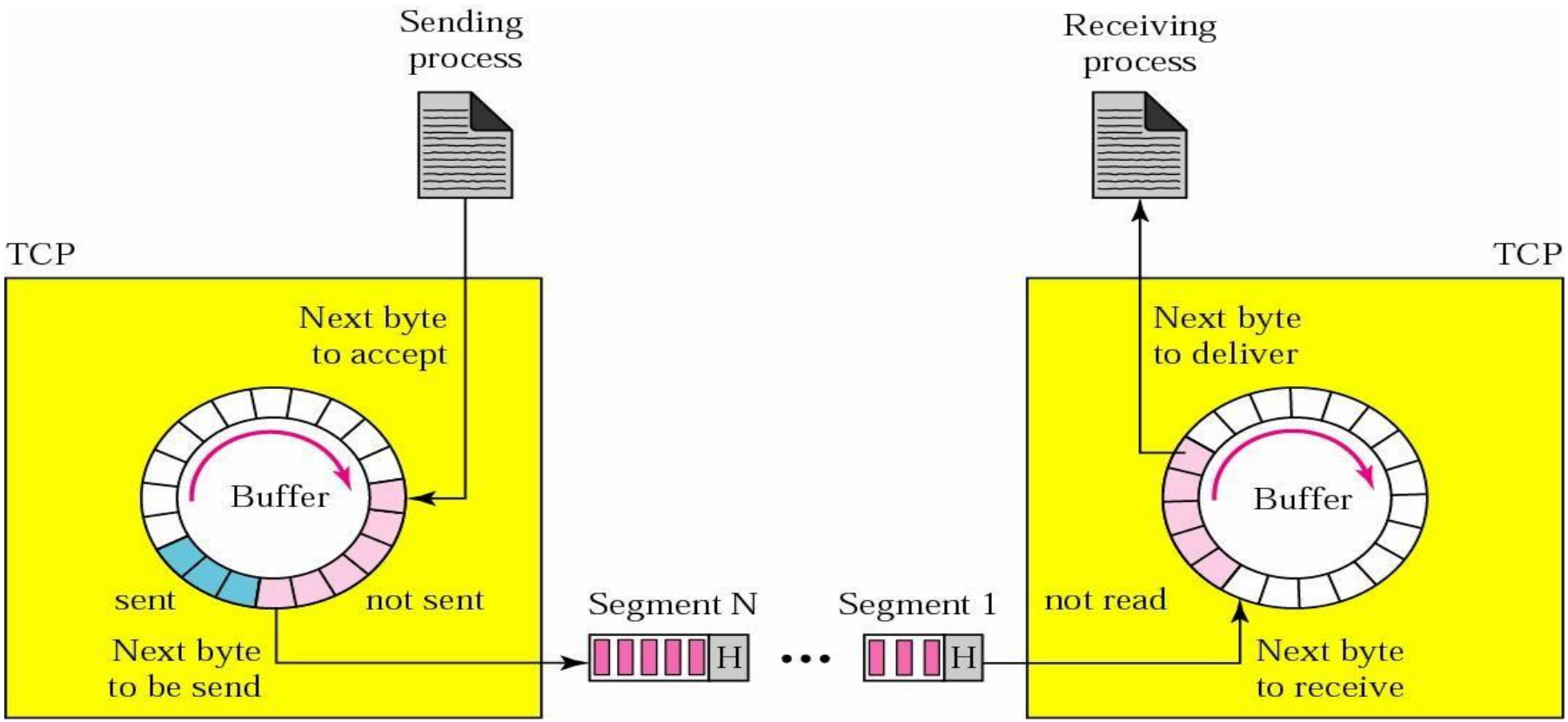
Sending and Receiving Buffers



Segments

- TCP groups a number of bytes together into a packet called a *segment*
  - A TCP packet is called a ***segment***
  - TCP adds a header to each segment
  - Then, the segments are encapsulated in an IP datagram

TCP Segments



Full-Duplex Communication

- TCP offers full-duplex service
  - Data can flow in both directions at the same time
  - Each TCP has a sending and receiving buffer and segments are sent in both direction

Connection-Oriented Service

- TCP is a connection-oriented protocol
  - However, the connection is virtual, not a physical connection
  - Each TCP segment may use a different path to reach the destination

Reliable Service

- TCP uses an *acknowledge mechanism* to check the safe and sound arrival of data

Numbering Bytes

- Although TCP use segments for transmission and reception
  - There is no field for a segment number in the segment header, i.e., TCP header

- TCP uses *sequence number* and *acknowledgement number* to keep track of the segment being transmitted or received
  - Notably, these two fields refer to the *byte number*, not the *segment number*

Byte Numbers

- TCP numbers all data bytes that are transmitted in a connection
- The numbering does not necessarily start from 0
  - *It starts randomly*
  - Between *0* and *2^32 – 1* for the number of the first byte
  - Byte numbering is used for *flow* and *error* control

The bytes of data being transferred in each connection are numbered by TCP.

The numbering starts with a randomly generated number.

Sequence Number

- TCP assigns a sequence number to each segment that is being sent

- The sequence number for each segment is *the number of the first byte* carried in that segment

The value of the sequence number field in a segment defines the number of the first data byte contained in that segment.

Example 2

Suppose a TCP connection is transferring a file of 5000 bytes. The first byte is numbered 10001.

What are the sequence numbers for each segment if data is sent in five segments, each carrying 1000 bytes?

Solution

- ***The following shows the sequence number for each segment:***

Segment 1 ➡ ***Sequence Number: 10,001 (range: 10,001 to 11,000)***

Segment 2 ➡ ***Sequence Number: 11,001 (range: 11,001 to 12,000)***

Segment 3 ➡ ***Sequence Number: 12,001 (range: 12,001 to 13,000)***

Segment 4 ➡ ***Sequence Number: 13,001 (range: 13,001 to 14,000)***

Segment 5 ➡ ***Sequence Number: 14,001 (range: 14,001 to 15,000)***

Example

- Imagine a TCP connection is transferring a file of 6000 bytes.
  - The first byte is numbered 10010

- What are the sequence numbers for each segment if data is sent in five segments with
  - The first four segments carrying 1,000 bytes
  - The last segment carrying 2,000 bytes?

Solution

The following shows the sequence number for each segment:

Segment 1 ➔ 10,010 (10,010 to 11,009)

Segment 2 ➔ 11,010 (11,010 to 12,009)

Segment 3 ➔ 12,010 (12,010 to 13,009)

Segment 4 ➔ 13,010 (13,010 to 14,009)

Segment 5 ➔ 14,010 (14,010 to 16,009)

Acknowledgment Number

- Communication in TCP is full duplex
  - Both parties can send and receive data at the same time in a connection
- Each party numbers the bytes, usually with a different starting byte number
  - *Sequence number*: the number of the first byte carried by the segment
  - *Acknowledgment number*: the number of the next byte that *the party expects to receive*

Acknowledgment Number (Cont.)

- Acknowledgment number is *cumulative*
- For example, if a party uses 5,643 as an acknowledgment number
  - It has received all bytes from the beginning up to 5,642
  - Note that, *this does not mean that the party has received 5642 bytes*
    - The first byte number does not have to start from 0

The value of the acknowledgment field in a segment defines the number of the next byte a party expects to receives. The acknowledgment number is cumulative.

Flow Control

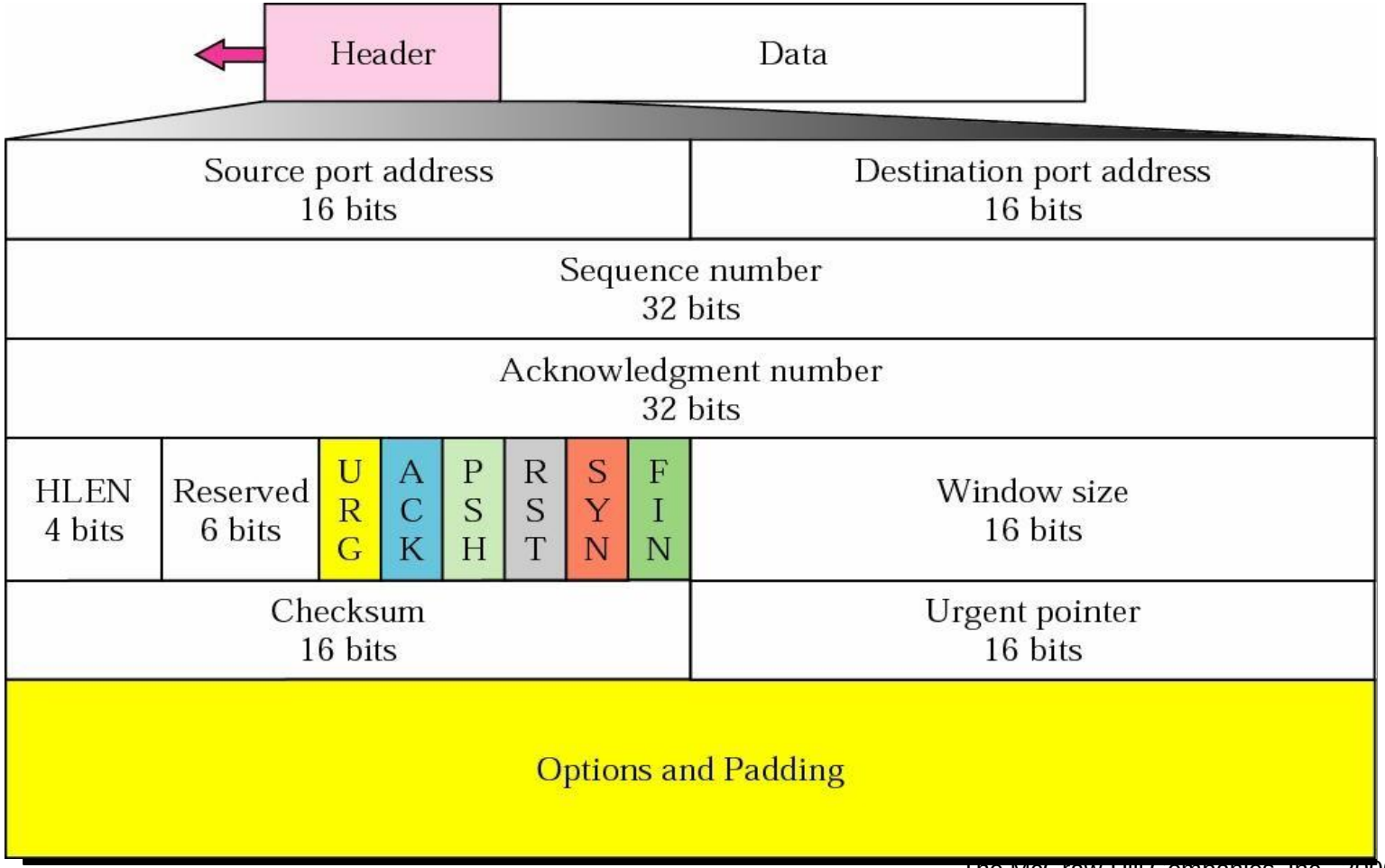
- The receiver controls how much data are to be sent by the sender
  - Prevent the receiver from being overwhelmed with data

- The numbering system allow TCP to use a ***byte-oriented flow control***

Error Control

- TCP implements an error control mechanism
  - To provide reliable service
  - Also byte-oriented

TCP Segment Format



The McGraw-Hill Companies, Inc., 2000

TCP Segment Format

- Source port address: 16-bit
- Destination port address: 16-bit
- Sequence number: 32-bit
  - The first byte number in this segment
  - In connection establishment, each party randomly generate an ***initial sequence number (ISN)***
    - Usually different in each direction
- Acknowledgment number: 32-bit
  - The byte number that the receiver expects
    - If received byte number *x*, ack. number is *x*+1
  - *Acknowledgment and data can be piggybacked together*

TCP Segment Format (Cont.)

- Header length: 4-bit
  - The number of *4-byte words* in the TCP header
  - Value of this field is between 5 and 15
    - TCP header is between *20-60* bytes
- Reserved: 6-bit
    - Reserved for future use
- Control: 6-bits

Control Field

URG: Urgent pointer is valid
ACK: Acknowledgment is valid
PSH: Request for push
RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: Terminate the connection

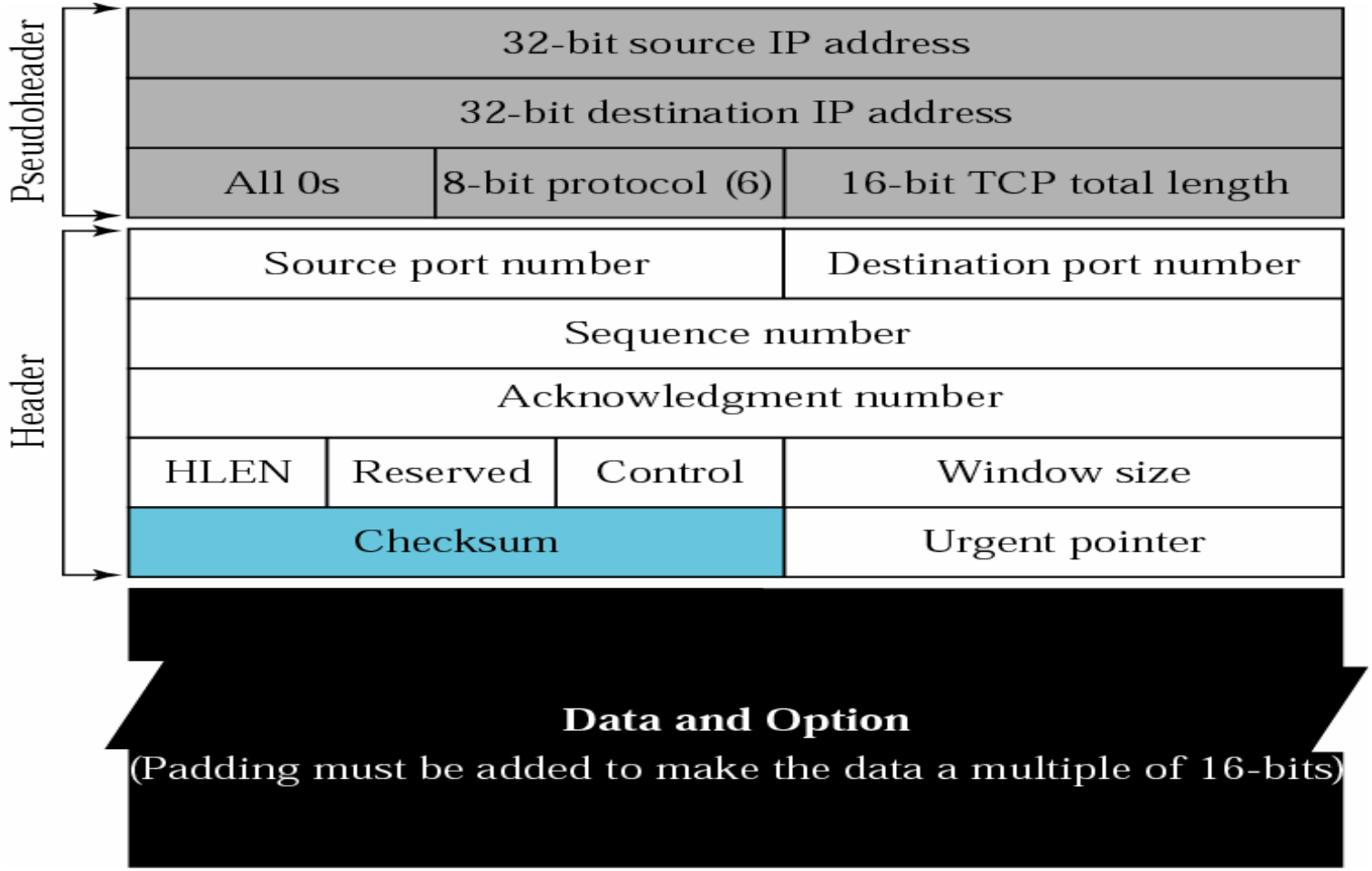
TCP Segment Format (Cont.)

- Control
  - URG: urgent pointer is valid
  - ACK: acknowledgment field is valid
  - PSH: push the data
  - RST: the connection must be reset
  - SYN: Synchronization sequence numbers during connection
  - FIN: terminate the connection

TCP Segment Format (Cont.)

- Window size: 16-bit
  - Define the size of the receiving window, in bytes
    - Determined by the receiver
  - The maximum window size is 2^16 = 65535
- Checksum: 16-bit
  - Follow the same procedure as UDP
  - Checksum for TCP is mandatory (UDP is optional)
- Urgent pointer: 16-bit
  - Valid only if the urgent bit is set
  - Used when the segment contains urgent data
- Options: 0~40 bytes

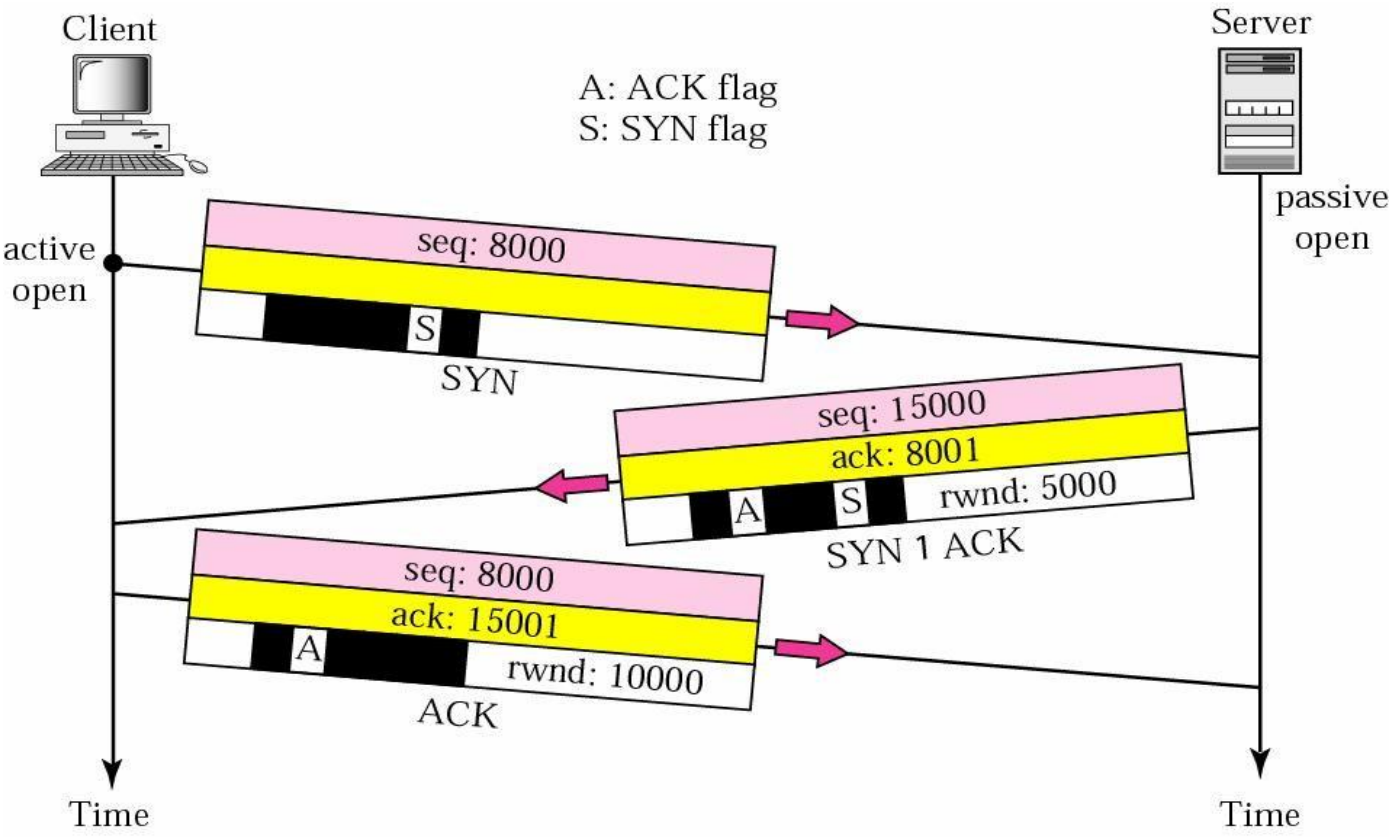
Pseudoheader added to the TCP datagram



A SYN segment cannot carry data, but it consumes one sequence number.

A SYN + ACK segment cannot carry data, but does consume one sequence number.

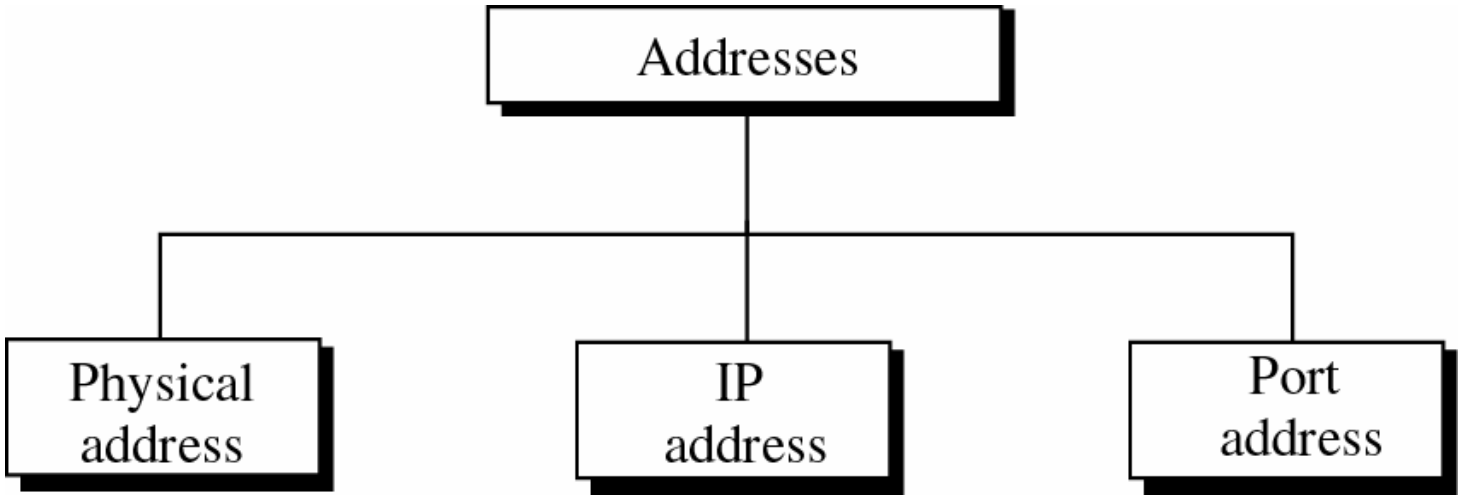
Three-way Handshaking



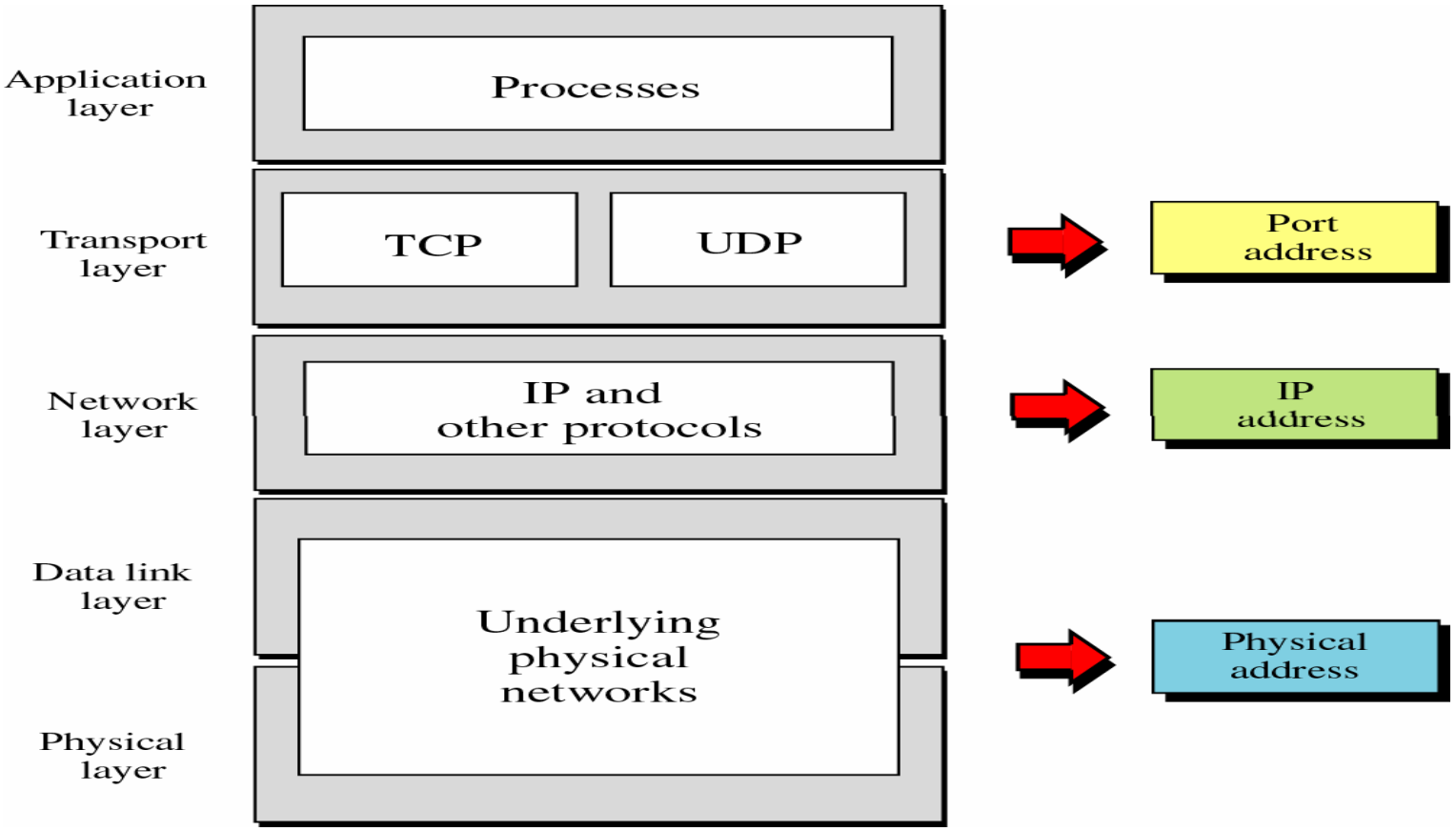
An ACK segment, if carrying no data, consumes no sequence number.

ADDRESSING

Addresses in TCP/IP



Relationship of Layers and Addresses in TCP/IP



Physical Address

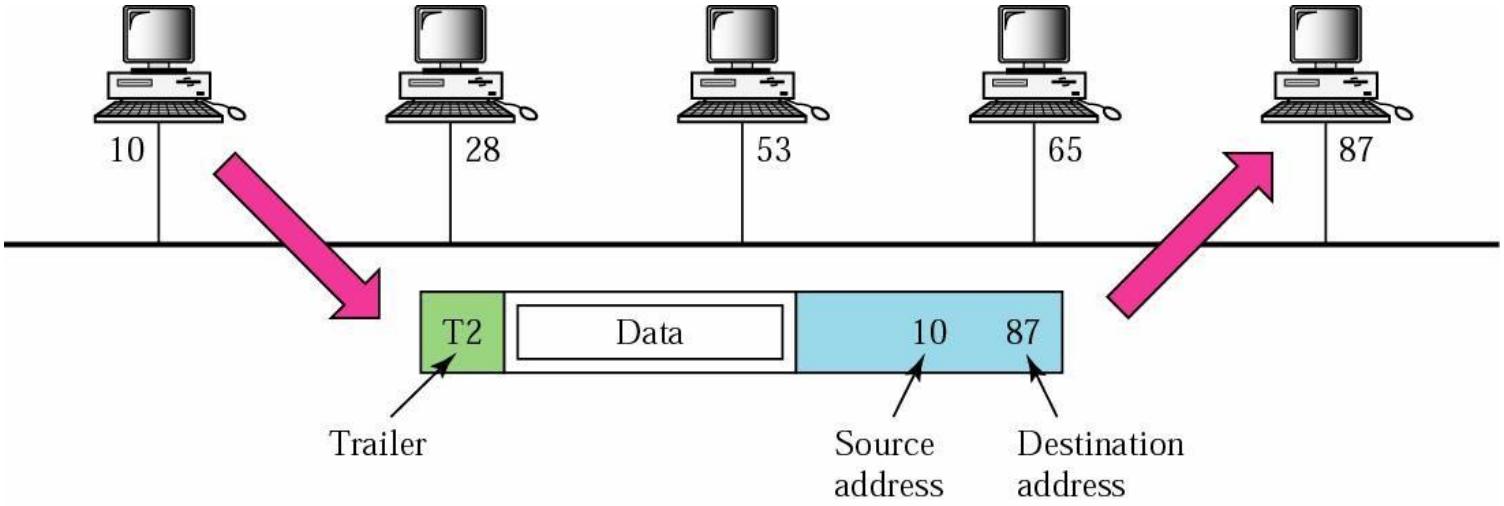
- Also called link address, the address of a node as defined by its LAN or WAN
  - Ethernet: 6 byte
  - LocalTalk: 1 byte
- Physical address can be either
  - Unicast address
  - Multicast address
  - Broadcast address

Example 1

Figure 2.18 shows an example of physical addresses.

F

Physical Addresses



Example 2

Most local area networks use a 48-bit (6 bytes) physical address written as 12 hexadecimal digits, with every 2 digits separated by a hyphen as shown below:

07-01-02-01-2C-4B

A 6-byte (12 hexadecimal digits) physical address

Internet Address

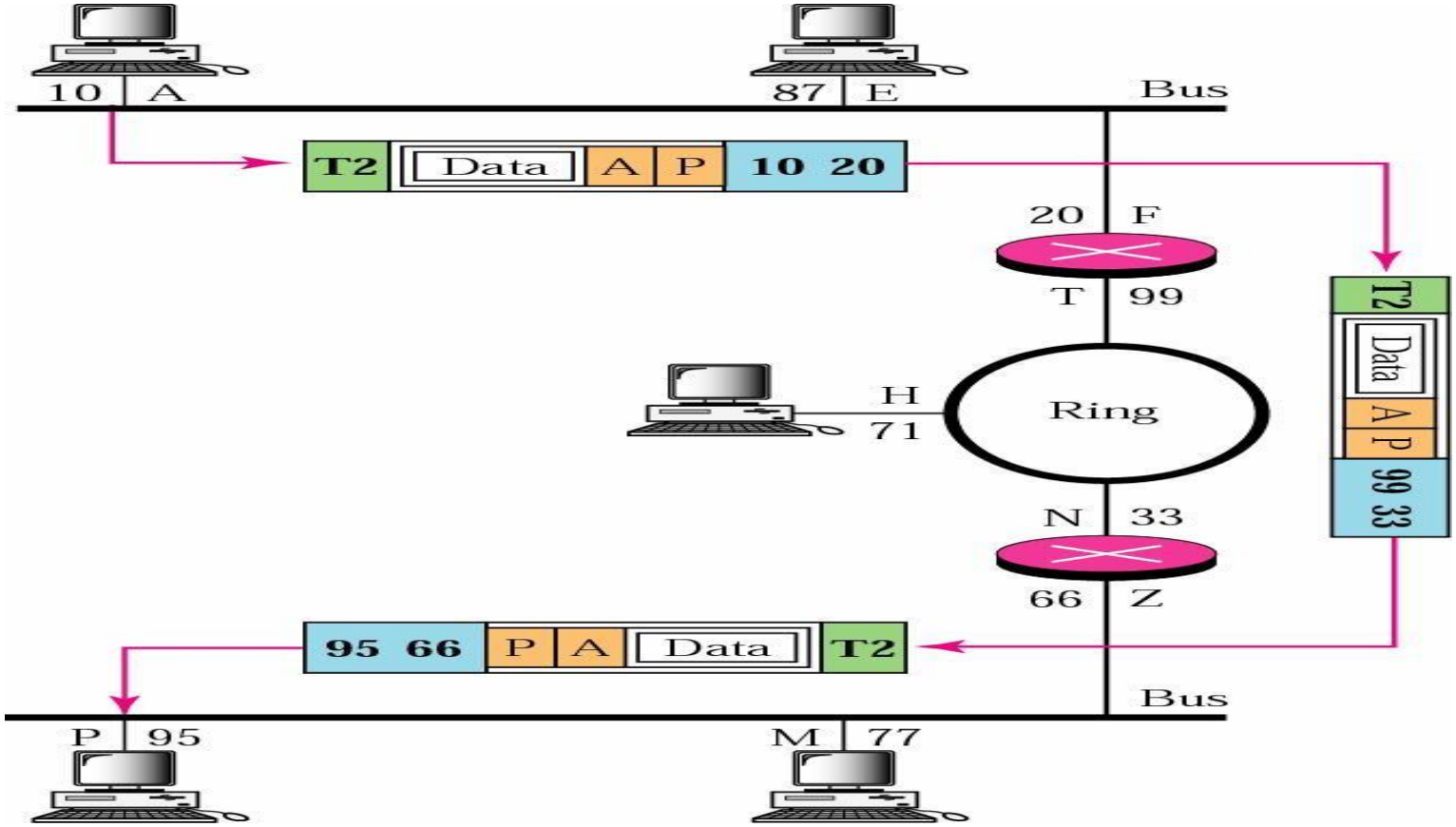
- Provide universal communication services that are independent of underlying physical network
  - Different networks can have different address format
  - A universal addressing system is thus needed
- Internet address can also be either in unicast, multicast and broadcast
- 4 bytes long in IPv4
- 16 bytes long in IPv6

Example 3

Next slide shows an example of Internet addresses.

- Network address A and physical address 10 => send data to => Network address P and physical address 95
- IP address does not change along the trip
- However, physical address changes from network to network

IP Addresses



Example 4

Internet address (in IPv4) is 32 bits in length, normally written as four decimal numbers, with each number representing 1 byte. The numbers are separated by a dot. Below is an example of such an address.

132.24.75.9

Example 5

Internet address (in IPv6) is 128 bits in length The preferred IPv6 address representation is: **x:x:x:x:x:x:x:x** , where each x is the hexadecimal values of the eight 16-bit pieces of the address. IPv6 addresses range from

0000:0000:0000:0000:0000:0000:0000:0000

to

ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff

Port Address

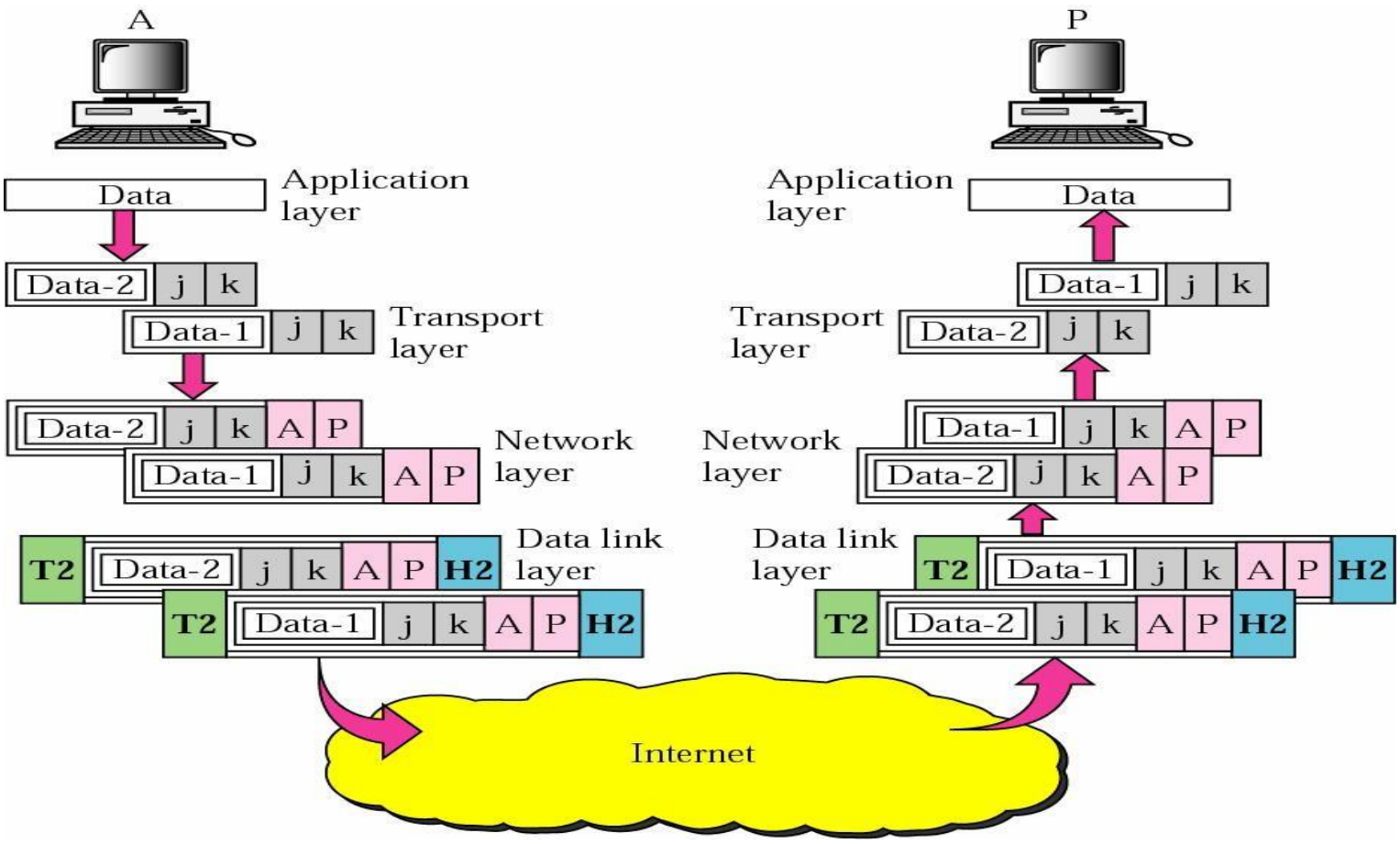
- A label assigns to a process
- 16 bits long

Example 6

Next slide shows an example of transport layer communication.

- Process with port address *j* send data to another process with port address *k*

Port Addresses



Example 6

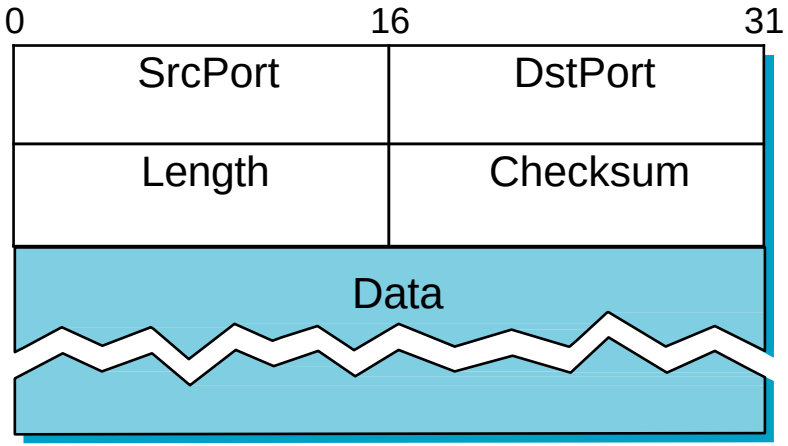
A port address is a 16-bit address represented by one decimal number as shown below.

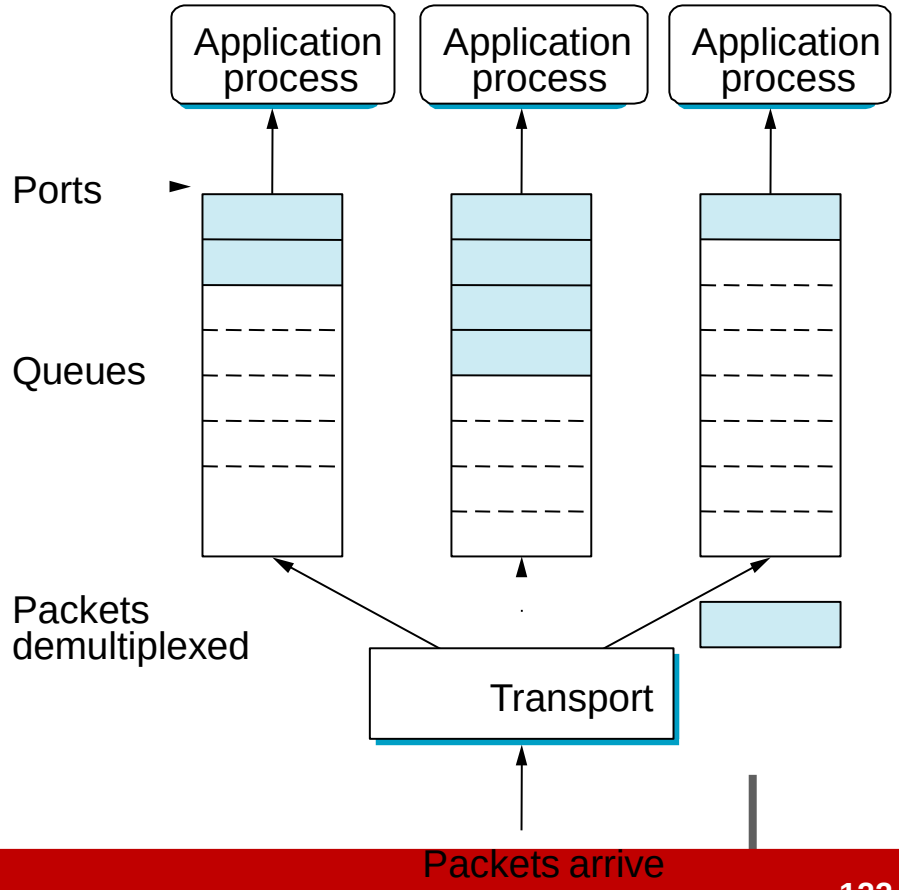
753 **A 16-bit port address**

Socket Programming

Demultiplexing

- Convert host-to-host packet delivery service into a process-to-process communication channel





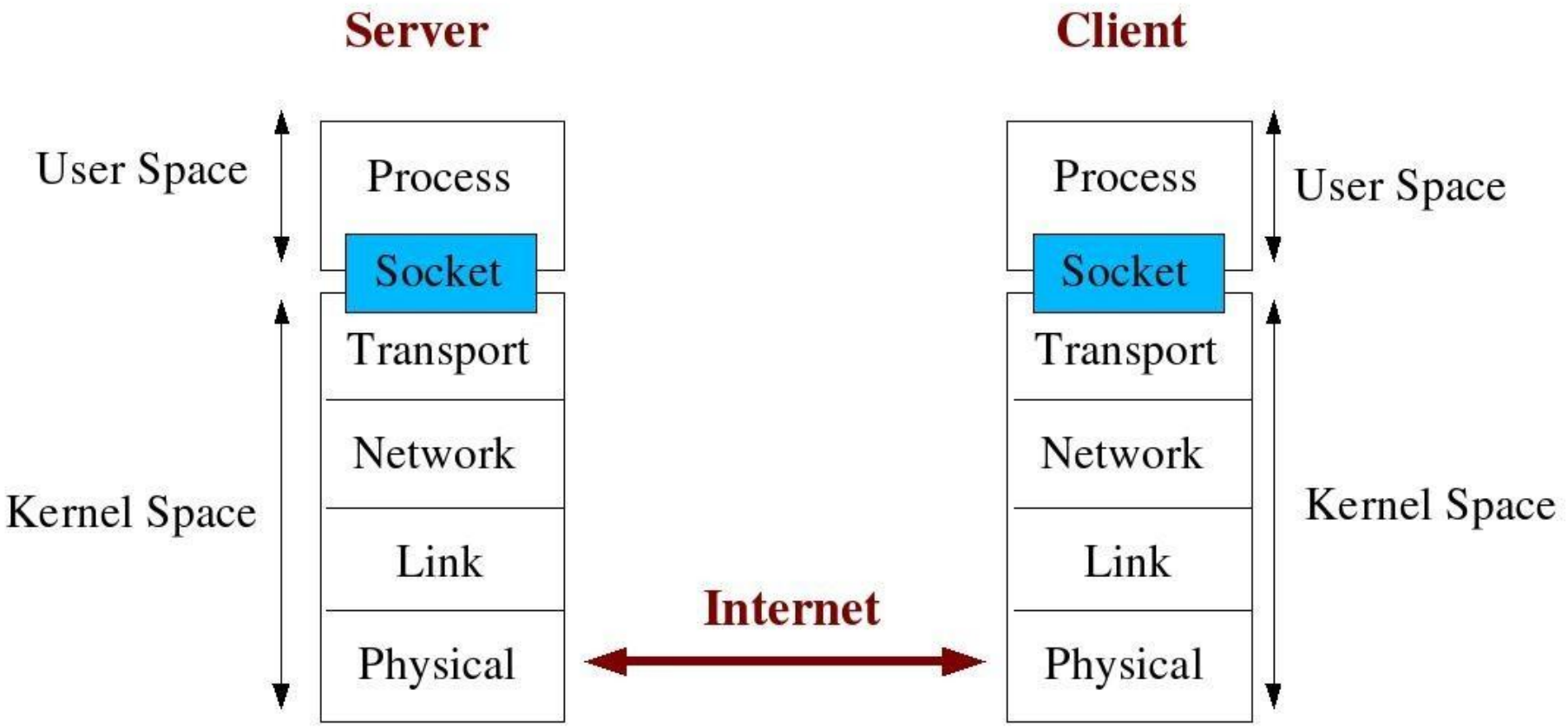
Byte Ordering

- Two types of “Byte ordering”
  - Network Byte Order: High-order byte of the number is stored in memory at the lowest address
  - Host Byte Order: Low-order byte of the number is stored in memory at the lowest address
  - Network stack (TCP/IP) expects Network Byte Order
- Conversions:
  - htons() - Host to Network Short
  - htonl() - Host to Network Long
  - ntohs() - Network to Host Short
  - ntohl() - Network to Host Long

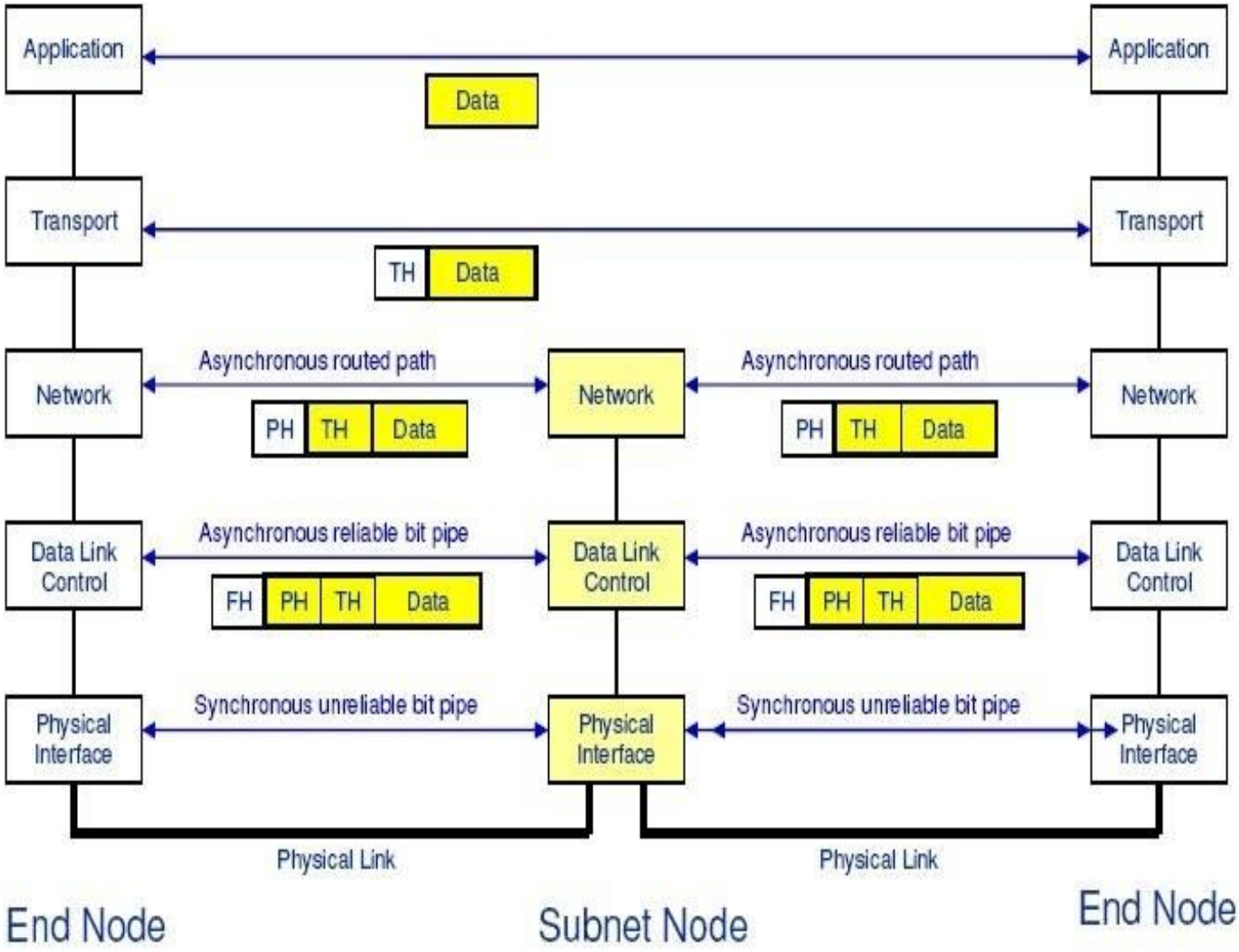
What is a socket?

- Socket: An interface between an application process and transport layer
  - The application process can send/receive messages to/from another application process (local or remote)via a socket
- In Unix jargon, a socket is a file descriptor – an integer associated with an open file
- Types of Sockets: **Internet Sockets**, unix sockets, X.25 sockets etc
  - Internet sockets characterized by IP Address (4 bytes), port number (2 bytes)

Socket in TCP/IP Description



Encapsulation



Types of Internet Sockets

- Stream Sockets (SOCK_STREAM)
  - Connection oriented
  - Rely on TCP to provide reliable two-way connected communication
- Datagram Sockets (SOCK_DGRAM)
  - Rely on UDP
  - Connection is unreliable

socket() -- Get the file descriptor

- int socket(int domain, int type, int protocol);
  - domain should be set to PF_INET
  - type can be SOCK_STREAM or SOCK_DGRAM
  - set protocol to 0 to have socket choose the correct protocol based on type
  - socket() returns a socket descriptor for use in later system calls or -1 on error

```
int sockfd;
sockfd = socket (PF_INET, SOCK_STREAM, 0);
```

Socket Structures

- struct sockaddr: Holds socket address information for many types of sockets

```
struct sockaddr {
        unsigned short   sa_family;     //address family AF_xxx
        unsigned short   sa_data[14]; //14 bytes of protocol addr
}
```

- struct sockaddr_in: A parallel structure that makes it easy to reference elements of the socket address

```
struct sockaddr_in {
        short int                sin_family;     // set to AF_INET
        unsigned short int       sin_port;       // Port number
        struct in_addr           sin_addr;       // Internet address
        unsigned char            sin_zero[8];   //set to all zeros
}
```

- sin_port and sin_addr must be in **Network Byte Order**

Dealing with IP Addresses

- ```
  struct in_addr {
      unsigned long s_addr; // that's a 32bit long, or 4 bytes
  };
  ```
- int inet_aton(const char *cp, struct in_addr *inp);

```
struct sockaddr_in  my_addr;
my_addr.sin_family = AF_INET;
my_addr.sin_port = htons(MYPORT);
inet_aton("10.0.0.5",&(my_addr.sin_addr));
memset(&(my_addr.sin_zero),'\0',8);
```

- inet_aton() gives non-zero on success; zero on failure

- To convert binary IP to string: inet_noa()

```
printf("%s",inet_ntoa(my_addr.sin_addr));
```
```

bind() - what port am I on?

- Used to associate a socket with a port on the local machine
  - The port number is used by the kernel to match an incoming packet to a process
- int bind(int sockfd, struct sockaddr *my_addr, int addrlen)
  - sockfd is the socket descriptor returned by socket()
  - my_addr is pointer to struct sockaddr that contains information about your IP address and port
  - addrlen is set to sizeof(struct sockaddr)
  - returns -1 on error
  - my_addr.sin_port = 0; //choose an unused port at random
  - my_addr.sin_addr.s_addr = INADDR_ANY; //use my IP adr

Example

```
int sockfd;
struct sockaddr_in my_addr;
sockfd = socket(PF_INET, SOCK_STREAM, 0);
my_addr.sin_family = AF_INET;          // host byte order
my_addr.sin_port = htons(MYPORT);      // short, network byte
   order
my_addr.sin_addr.s_addr = inet_addr("172.28.44.57");
memset(&(my_addr.sin_zero), '\0', 8); // zero the rest of the struct
bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct
   sockaddr));
/******  Code needs error checking. Don't forget to do that ****** /
```

connect() - Hello!

- Connects to a remote host
- int connect(int sockfd, struct sockaddr *serv_addr, int addrlen)
  - sockfd is the socket descriptor returned by socket()
  - serv_addr is pointer to struct sockaddr that contains information on destination IP address and port
  - addrlen is set to sizeof(struct sockaddr)
  - returns -1 on error
- No need to bind(), kernel will choose a port

Example

```
#define DEST_IP    "172.28.44.57"
#define DEST_PORT 5000
main(){
   int sockfd;
   struct sockaddr_in dest_addr;   // will hold the destination addr
   sockfd = socket(PF_INET, SOCK_STREAM, 0);
    dest_addr.sin_family = AF_INET;          // host byte order
    dest_addr.sin_port = htons(DEST_PORT);    // network byte
   order
    dest_addr.sin_addr.s_addr = inet_addr(DEST_IP);
    memset(&(dest_addr.sin_zero), '\0', 8);   // zero the rest of the
   struct     connect(sockfd, (struct sockaddr *)&dest_addr,
   sizeof(struct sockaddr));
  /******* Don't forget error checking ********/
```

listen() - Call me please!

- Waits for incoming connections
- int listen(int sockfd, int backlog);
  - sockfd is the socket file descriptor returned by socket()
  - backlog is the number of connections allowed on the incoming queue
  - listen() returns -1 on error
  - Need to call bind() before you can listen()
    - socket()
    - bind()
    - listen()
    - accept()

accept() - Thank you for calling !

- accept() gets the pending connection on the port you are listen()ing on
- int accept(int sockfd, void *addr, int *addrlen);
  - sockfd is the listening socket descriptor
  - information about incoming connection is stored in addr which is a pointer to a local struct sockaddr_in
  - addrlen is set to sizeof(struct sockaddr_in)
  - accept returns *a new socket file descriptor* to use for this accepted connection and -1 on error

Example

```
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#define MYPORT 3490     // the port users will be connecting to
#define BACKLOG 10      // pending connections queue will hold
main(){
    int sockfd, new_fd;  // listen on sock_fd, new connection on
    new_fd
    struct sockaddr_in my_addr;     // my address information
    struct sockaddr_in their_addr; // connector's address information
    int sin_size;
   sockfd = socket(PF_INET, SOCK_STREAM, 0);
```

Cont...

```
my_addr.sin_family = AF_INET;          // host byte order
my_addr.sin_port = htons(MYPORT);      // short, network byte order
my_addr.sin_addr.s_addr = INADDR_ANY; // auto-fill with my IP
memset(&(my_addr.sin_zero), '\0', 8); // zero the rest of the struct

// don't forget your error checking for these calls:
bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr));
listen(sockfd, BACKLOG);
sin_size = sizeof(struct sockaddr_in);
new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size);
```

send() and recv() - Let's talk!

- The two functions are for communicating over stream sockets or connected datagram sockets.
- int send(int sockfd, const void *msg, int len, int flags);
  - sockfd is the socket descriptor you want to send data to (returned by socket() or got from accept())
  - msg is a pointer to the data you want to send
  - len is the length of that data in bytes
  - set flags to 0 for now
  - sent() returns the number of bytes actually sent (may be less than the number you told it to send) or -1 on error

send() and recv() - Let's talk!

- int recv(int sockfd, void *buf, int len, int flags);
  - sockfd is the socket descriptor to read from
  - buf is the buffer to read the information into
  - len is the maximum length of the buffer
  - set flags to 0 for now
  - recv() returns the number of bytes actually read into the buffer or -1 on error
  - If recv() returns 0, the remote side has closed connection on you

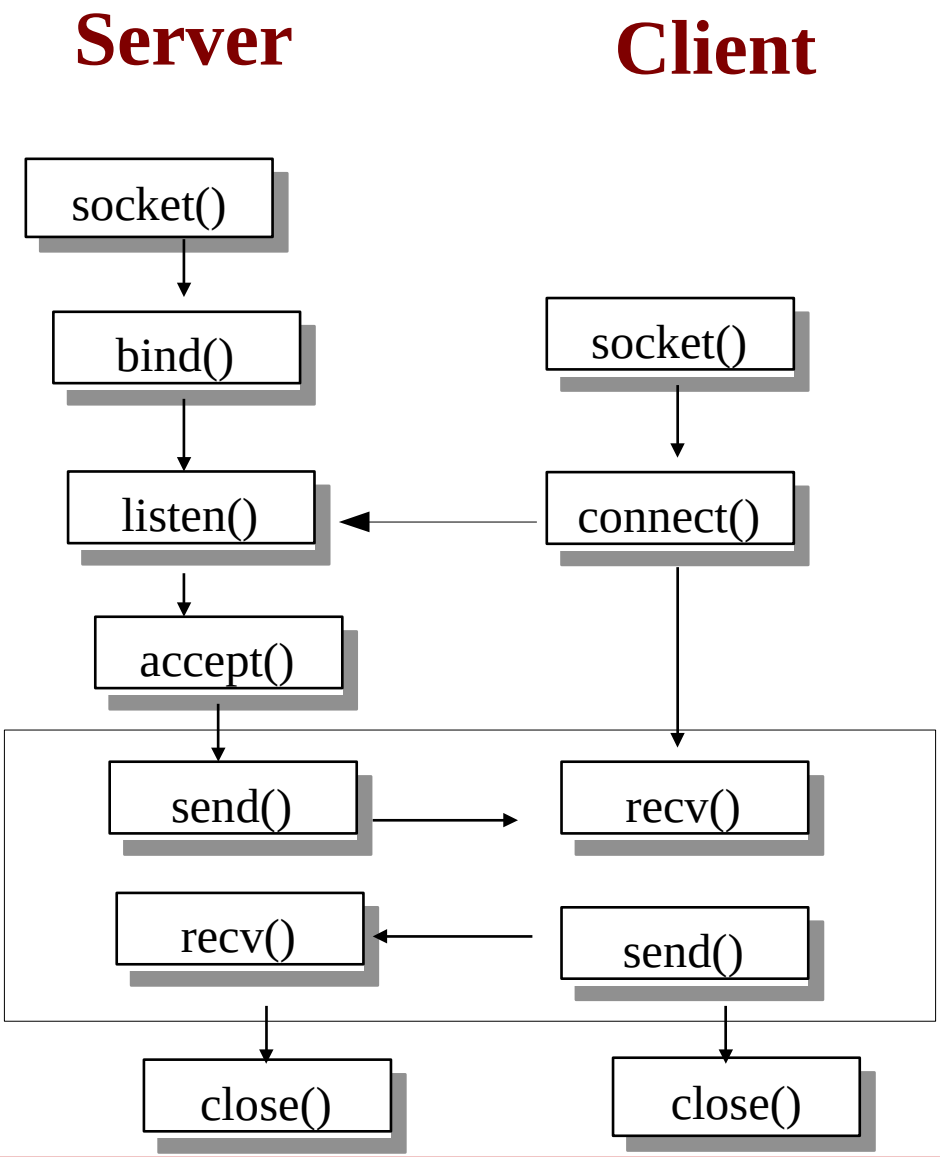
sendto() and recvfrom() - DGRAM style

- int sendto(int sockfd, const void *msg, int len, int flags, const struct sockaddr *to, int tolen);
  - *to* is a pointer to a struct sockaddr which contains the destination IP and port
  - *tolen* is sizeof(struct sockaddr)
- int recvfrom(int sockfd, void *buf, int len, int flags, struct sockaddr *from, int *fromlen);
  - *from* is a pointer to a local struct sockaddr that will be filled with IP address and port of the originating machine
  - *fromlen* will contain length of address stored in *from*

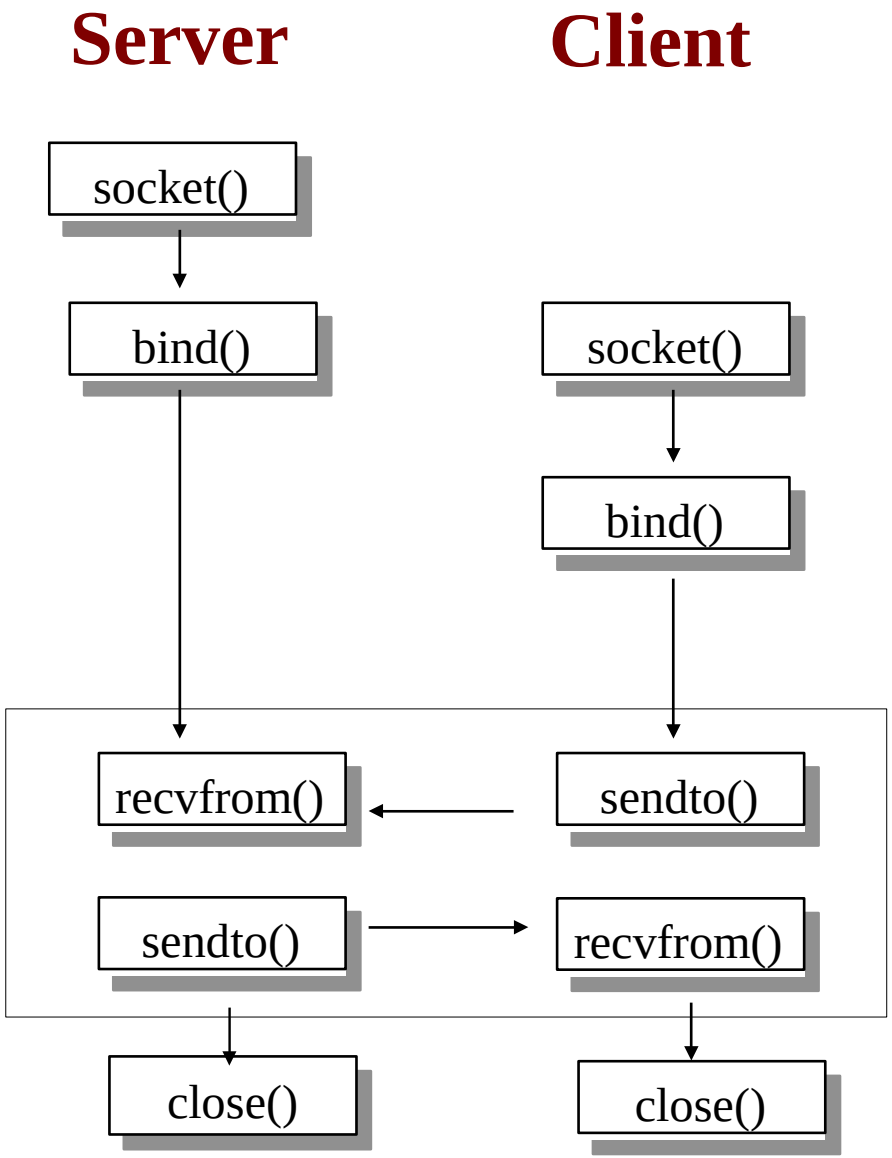
close() - Bye Bye!

- int close(int sockfd);
  - Closes connection corresponding to the socket descriptor and frees the socket descriptor
  - Will prevent any more sends and recvs

Connection Oriented Protocol



Connectionless Protocol



Miscellaneous Routines

- int getpeername(int sockfd, struct sockaddr *addr, int *addrlen);
  - Will tell who is at the other end of a connected stream socket and store that info in *addr*
- int gethostname(char *hostname, size_t size);
  - Will get the name of the computer your program is running on and store that info in hostname

Miscellaneous Routines

- struct hostent *gethostbyname(const char *name);

```
struct hostent {
    char    *h_name;        //official name of host
    char    **h_aliases;    //alternate names for the host
    int     h_addrtype;     //usually AF_NET
    int     h_length;       //length of the address in bytes
    char    **h_addr_list; //array of network addresses for the host
}
#define h_addr h_addr_list[0]
```

- Example Usage:

```
struct hostent *h;
h = gethostbyname("www.cdacacts.in");
printf("Host name : %s \n", h>h_name);
printf("IP Address: %s\n",inet_ntoa(*((struct in_addr *)h>h_addr)));
```

Summary

- Sockets help application process to communicate with each other using standard Unix file descriptors
- Two types of Internet sockets: SOCK_STREAM and SOCK_DGRAM
- Many routines exist to help ease the process of communication

Syllabus

1. IoT Trends, IoT Architecture, IoT Applications
2. IoT Standards and Protocols
3. Wireless LAN: IEEE 802.11
4. Wireless PAN: IEEE 802.15.1 & 802.15.4, Zigbee
5. Bluetooth, BTLE, LPWAN (LoRa, NBIoT), 6LowPAN
6. REST, CoAP, MQTT
7. Basics of Cryptography, Overview of IoT and Embedded security
8. Overview of 5G technologies

IoT Trends, IoT Architecture, IoT Applications

Internet of Things (IoT)

- The term IoT can most likely be attributed to Kevin Ashton in 1997 with his work at Proctor and Gamble using RFID tags to manage supply chains.
- Then, in 2009: “Conventional diagrams of the Internet include servers and routers and so on, but they leave out the most numerous and important routers of all: people. The problem is, **people have limited time, attention and accuracy** […] If we had **computers that knew everything** there was to know **about things** […] we would be able to […] greatly **reduce waste, loss and cost**. We would know when things needed replacing, repairing or recalling, and whether they were fresh or past their best

Internet of Things - Scenarios

- Since 1999 the concept evolved far beyond RFID
- **Everything can now be connected to the Internet**

- The term IoT now refers to different scenarios:
  - Wireless Sensor Networks (WSN)
  - Near Field Communications (NFC)
  - Biotechnology and Body Area Networks (BAN)
  - Machine-to-Machine communications (M2M)
  - Personal Area Networks (PAN)
  - …

IoT versus machine to machine

Machine2Machine:

- It is a general concept involving an autonomous device communicating
- directly to another autonomous device.
- It may very well be the case that an M2M device uses no internet services or topologies for communication.
- This leaves out typical internet appliances used regularly for cloud services and storage.
- An M2M system may communicate over non-IP based channels as well, such as a serial port or custom protocol.

Internet of Things:

- IoT systems may incorporate some M2M nodes (such as a Bluetooth mesh using non-IP communication), but aggregates data at an edge router or gateway.
- An edge appliance like a gateway or router serves as the entry point onto the internet. Regardless of where the internet *on-ramp* exists, the fact that it has a method of tying into the internet fabric is what defines IoT.

IoT - How Big?

- Possibly every single device and object will be connected to the Internet
- About 50-100 billions devices in 2020 (data from SAP/Intel and Ericsson)
- IBM Smarter Planet vision:
  - instrumented
  - interconnected
  - intelligent

- Several real world examples: industrial control systems, health monitoring, smart metering, home automation, …

Internet of Things - Use Cases

Smart Wearables

Smart Home

Smart City

Smart Agriculture

Connected Car

Health Care

Industry Automation

Smart Energy

IoT Applications – IIoT, Consumer

Industrial IoT (**IIoT**) is one of the fastest and largest segments in the overall IoT space by the number of connected things and the value those services bring to manufacturing and factory automation.

Industrial and manufacturing IoT use cases and impact

Following are the industrial and manufacturing IoT use cases and their impact:

- Preventative maintenance on new and pre-existing factory machinery
- Throughput increase through real-time demand
- Energy savings
- Safety systems such as thermal sensing, pressure sensing, and gas leaks
- Factory floor expert systems

Consumer IoT use cases

The following are some of the consumer IoT use cases:

- **Smart home gadgetry**: Smart irrigation, smart garage doors, smart locks, smart lights, smart thermostats, and smart security.
- **Wearables**: Health and movement trackers, smart clothing/wearables.
- **Pets:** Pet location systems, smart dog doors.

IoT Applications – Retail

Retail IoT use cases

Some of the retail IoT use cases are as follows:

- Targeted advertising, such as locating known or potential customers by proximity and providing sales information.
- Beaconing, such as proximity sensing customers, traffic patterns, and inter-arrival times as marketing analytics.
- Asset tracking, such as inventory control, loss control, and supply chain optimizations.
- Cold storage monitoring, such as analyze cold storage of perishable inventory.
- Apply predictive analytics to food supply.
- Insurance tracking of assets.
- Insurance risk measurement of drivers.
- Digital signage within retail, hospitality, or citywide.
- Beaconing systems within entertainment venues, conferences, concerts, amusement parks, and museums.

IoT Applications – Healthcare, Logistics

Healthcare IoT use cases

Some of the healthcare IoT use cases are as follows:

- In-home patient care
- Learning models of predictive and preventative healthcare
- Dementia and elderly care and tracking
- Hospital equipment and supply asset tracking
- Pharmaceutical tracking and security
- Remote field medicine
- Drug research
- Patient fall indicators

Transportation and logistics IoT use cases

Following are some of the transportation and logistics IoT use cases:

- Fleet tracking and location awareness
- Railcar identification and tracking
- Asset and package tracking within fleets
- Preventative maintenance of vehicles on the road

IoT Applications – Agricultural

Agricultural and environmental IoT use cases

Some of the agricultural and environmental IoT use cases are as follows:

- Smart irrigation and fertilization techniques to improve yield
- Smart lighting in nesting or poultry farming to improve yield
- Livestock health and asset tracking
- Preventative maintenance on remote farming equipment via manufacturer
- Drones-based land surveys
- Farm-to-market supply chain efficiencies with asset tracking
- Robotic farming
- Volcanic and fault line monitoring for predictive disasters

IoT Applications – Smart City , military

Smart city IoT use cases

Some of the smart city IoT use cases are as follows:

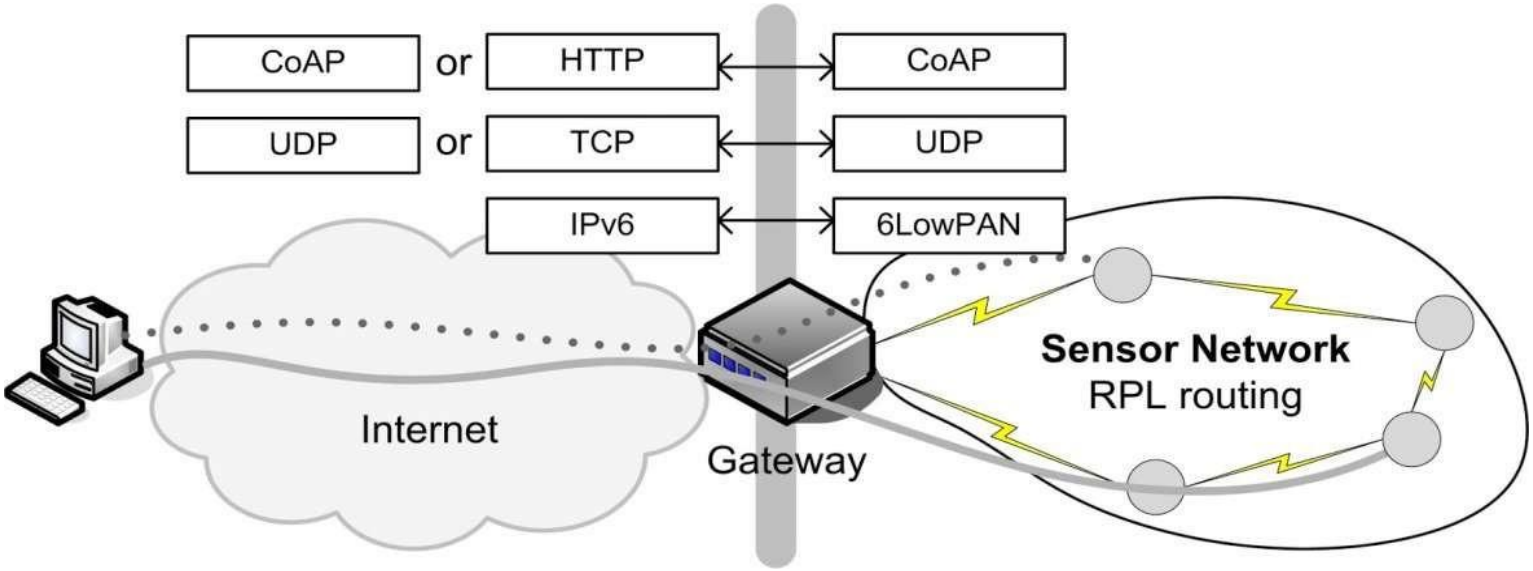
- Pollution control and regulatory analysis through environmental sensing
- Microclimate weather predictions using citywide sensor networks
- Energy efficiency of city lighting on demand
- Smart irrigation of parks and public spaces, depending on weather
- Smart cameras to watch for crime and real-time automated AMBER Alerts
- Smart parking lots

Government and military IoT use cases

Following are some of the government and military IoT use cases:

- Terror threat analysis through IoT device pattern analysis and beacons
- Swarm sensors through drones
- Sensor bombs deployed on the battlefield to form sensor networks to monitor threats
- Government asset tracking systems
- Real-time military personal tracking and location services
- Synthetic sensors to monitor hostile environments
- Water level monitoring to measure dam and flood containment

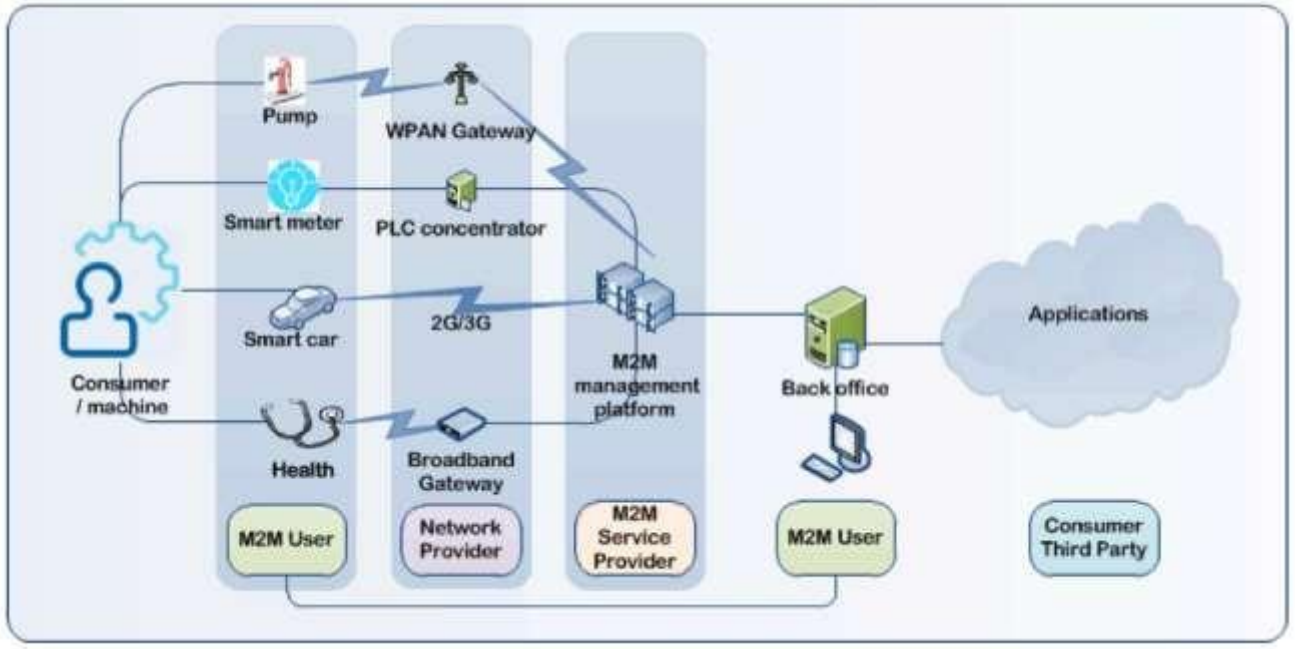
IoT - the General Idea



- The general idea behind the (commonly accepted) vision of IoT consists in the **extension of Internet protocols to Wireless Sensors Networks (WSNs)**, composed of sensors as well as actuators

IoT for Manufacturing - Machine to Machine

- The term “Machine to Machine” (**M2M**) describes devices **connected to each other**, by using a variety of fixed/wireless networks and through the Internet to the wider world. They are **active** communication devices



- Note the **wide variety of communication** infrastructures and services and **massive number of nodes**

IoT - Main Tasks

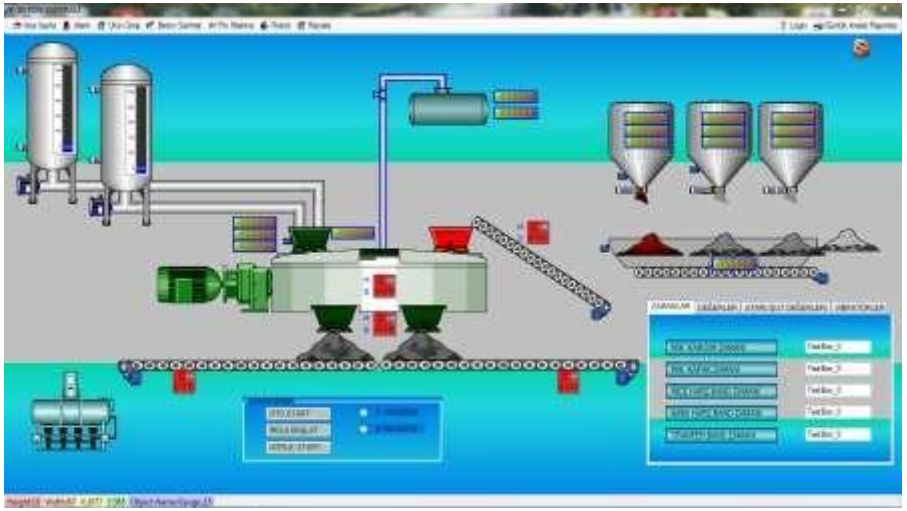
- **Gather information from things** and send commands to things
  - monitoring: state information
  - control: command enforcement
- **Send information back and forth** remote locations (private/public cloud)
- **Store and aggregate** information
- **Analyze** information to improve system knowledge
- **Take decisions**, in a human-assisted or autonomous manner

Outline

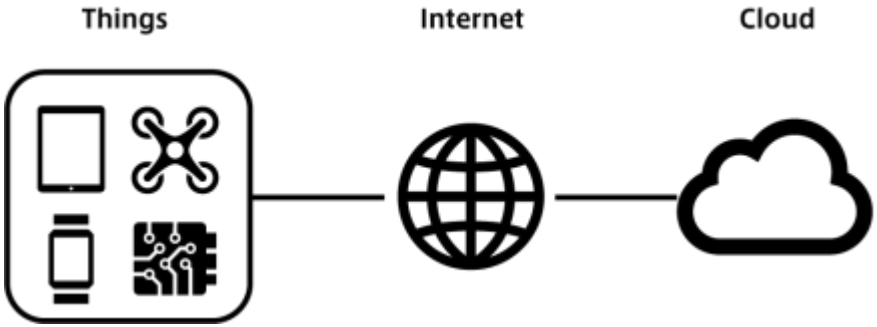
1. Introduction to IoT
   - **definition, enabling technologies and concepts to better understand the general framework of IoT solutions**
   - layering architecture, cloud computing vs. fog computing
2. Most relevant components of IoT solutions
   - devices, wireless communication protocols, data exchange protocols, IoT platforms, and data analysis

SCADA - Supervisory Control And Data Acquisition

- Before IoT, **remote monitoring and control already available for years**
- Central control system with sensors/actuators, controllers, communication equipment, and software
- Issues with SCADA : **Expensive, lack of standard, no interoperability**



IoT - Main Goals, in a nutshell…



- IoT can be seen as a novel business approach
  - based on a set of (already available) technology tools
  - exploited in novel (industrial) scenarios
- The Internet (and its standards) exploited to transfer data about things in an ***efficient***, ***interoperable,*** and ***secure*** manner

- Things can be
  - **physical**: Cyber Physical System (CPS) solutions to create a bridge between the physical and digital worlds
  - **digital**: information are already in binary format, but typically not in a standard format

- Gathered data analyzed to acquire a more **in-depth knowledge of physical/digital systems**, typically composed of **geographically distributed** and **heterogeneous** things

IoT Definition – Small Environment Scenario

- An IoT
  - is a network that
  - connects uniquely identifiable “Things”
  - to the Internet
- The Things
  - have sensing/actuation and
  - potential programmability capabilities
- Through the exploitation of unique identification and sensing
  - information about the Thing can be collected
  - and the state of the Thing can be changed
  - from anywhere, anytime, by anything

IoT Enabling Technologies

- Reduced **hardware** cost and size
  - from special-purpose to Commercial Off-The-Shelf (COTS)
- Pervasive and cheap **wireless communication**
  - from cables to large-bandwidth and/or wide-coverage wireless communication

actual game changers, IMHO

- Consolidated and emerging **Web-based communication**
  - from close protocols to open standards, also applied in constrained devices
- **Standards**, to achieve interoperability
  - e.g., communication standards and data representation
- General purpose **horizontal solutions**
  - from SCADA to IoT platforms
- Automatic tools to **infer knowledge**
  - wide application of AI (Artificial Intelligence) techniques

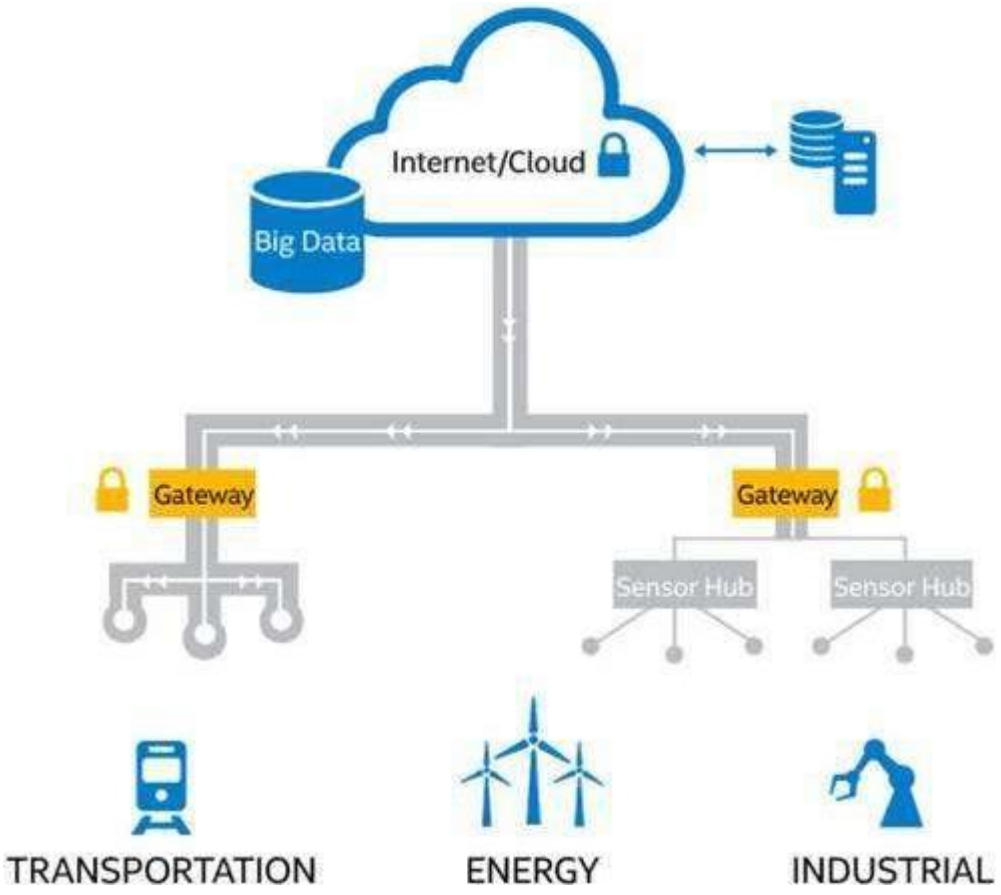
Outline

1. Introduction to IoT
   - definition, enabling technologies and concepts to better understand the general framework of IoT solutions
   - **layering architecture, cloud computing vs. fog computing**

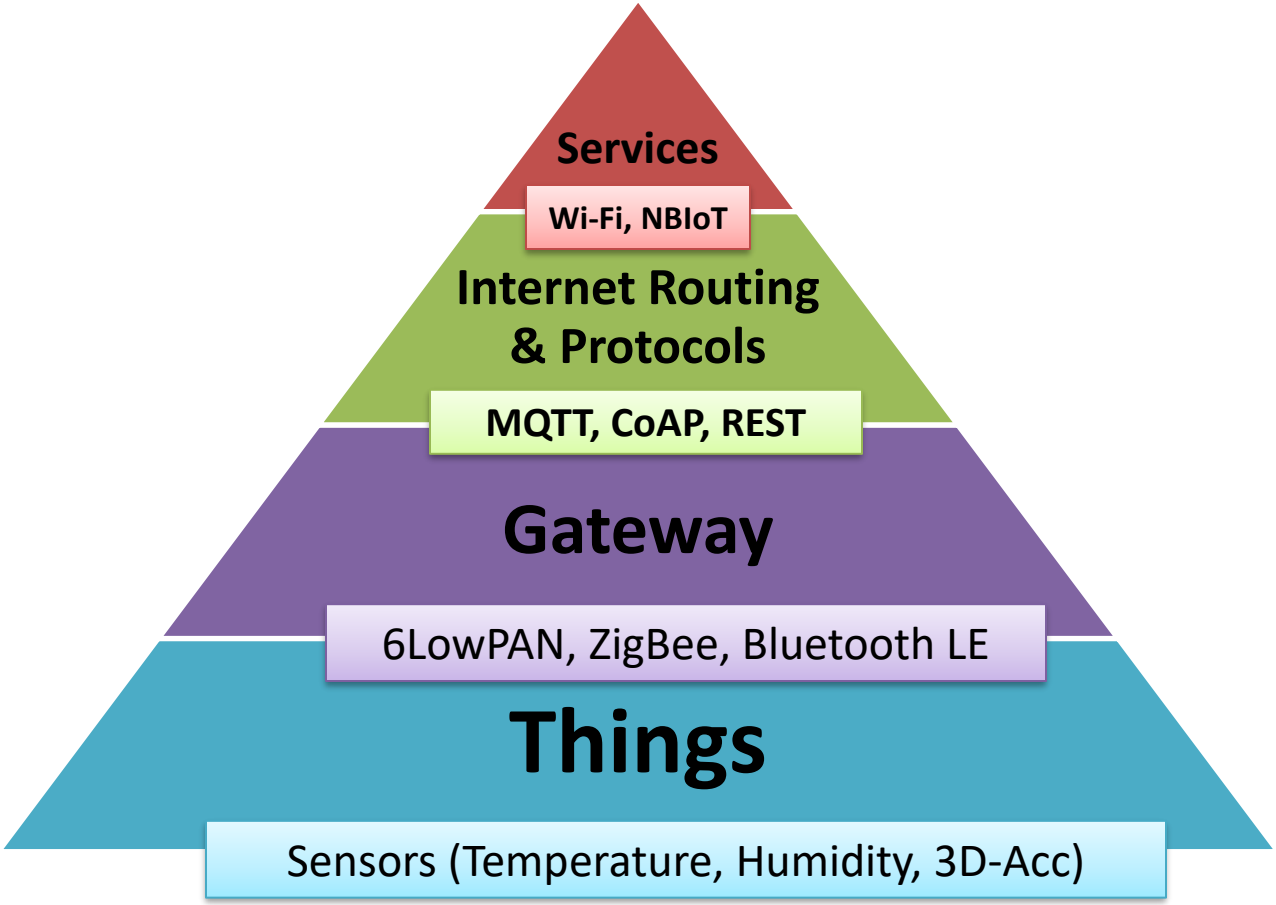
2. Most relevant components of IoT solutions
   - devices, wireless communication protocols, data exchange protocols

Typical Cloud-based IoT Architecture

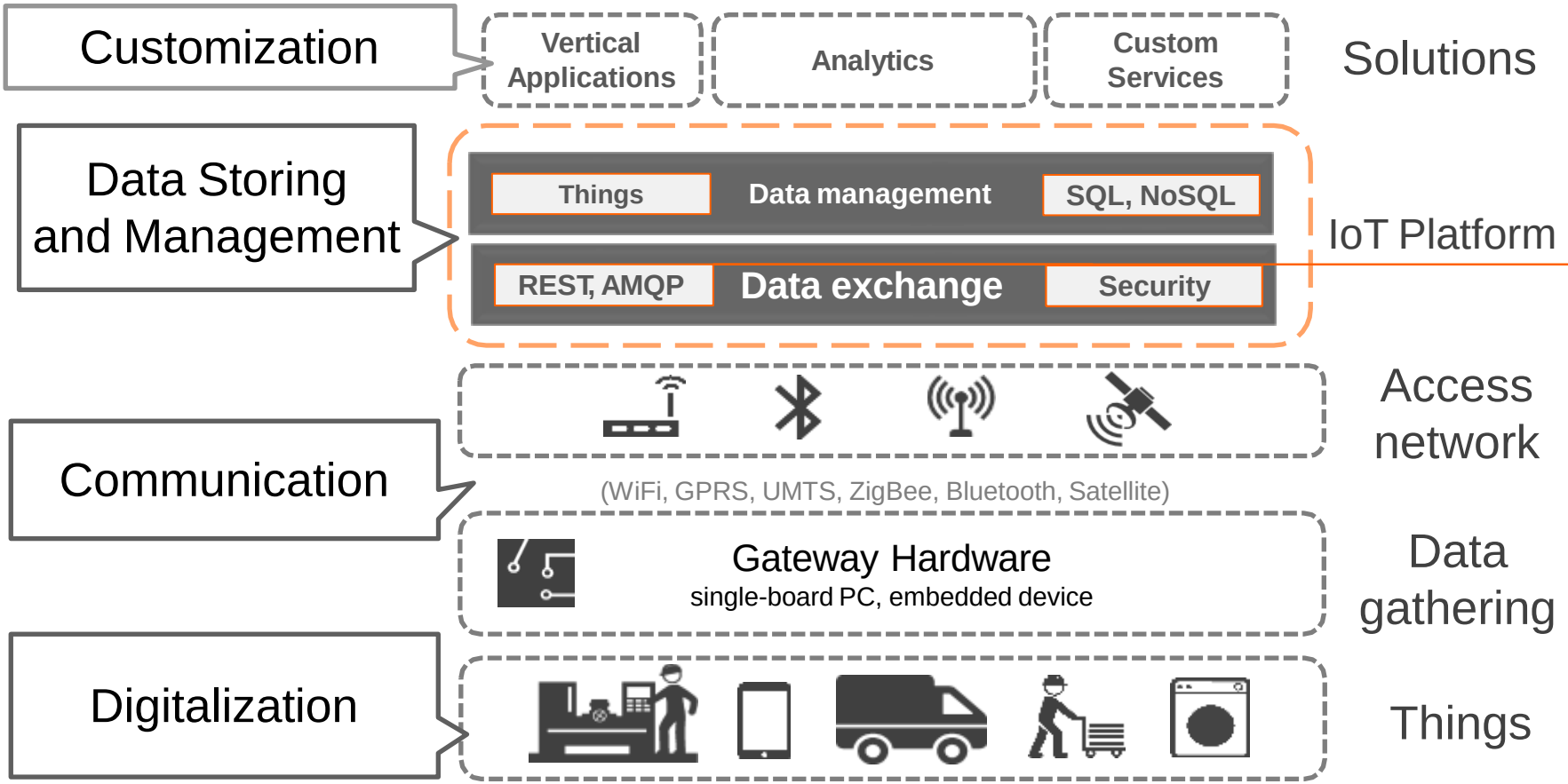
- Several heterogeneous **things**, e.g., sensors and actuators
- Multiple **gateways** geographically close to sensors/actuators
  - directly interact with things
  - dispatch data to/from the Internet
- Server-side remote **applications** stored in the Cloud and managing data



IoT Architecture



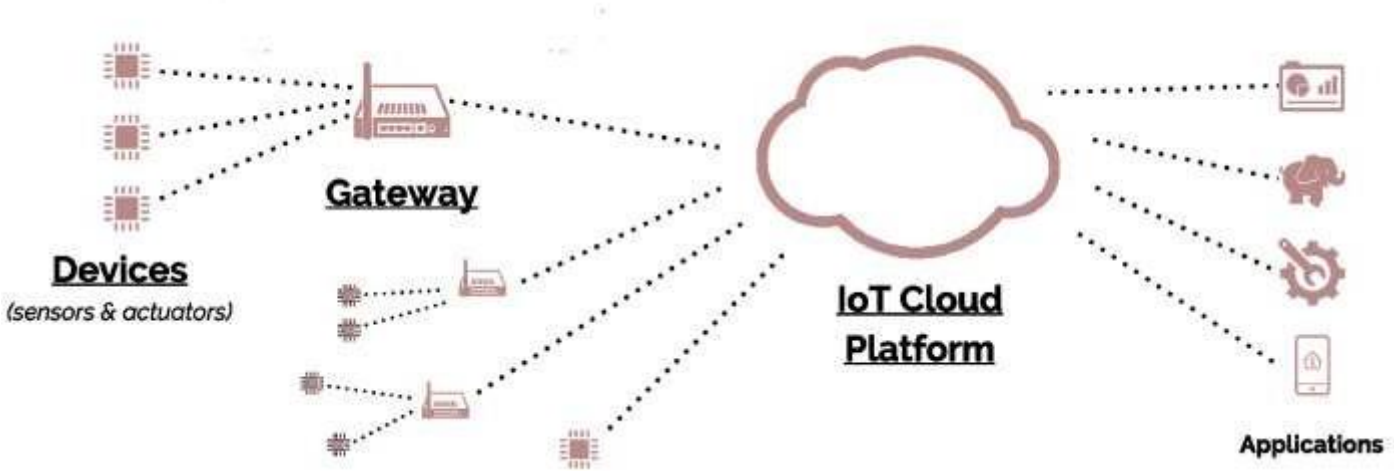
IoT: One Solution, Many Layers



Layering to Simplify Complexity

- **Things**: any physical or digital objects that should be monitored or controlled
  - physical objects must be digitalized
  - virtual objects must be standardized
- **Gateway**: close to one or multiple things to interact with them and send data and command back and forth the Internet
- **Communication protocol**: wired/wireless technology to actually send bytes
- **Data exchange protocol**: software protocol to standardize how information are transported (and eventually also represented)
- **IoT Platform**: data storing and management, application of (simple) aggregation/processing functions on data
- **Analytics**: complex analysis on data to infer new knowledge

Gateway



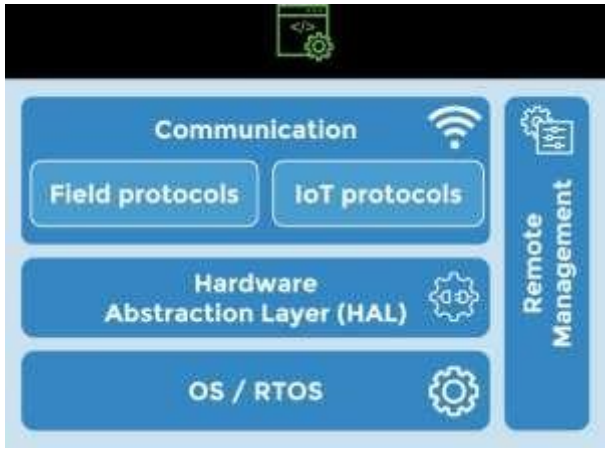
- **Protocol translation** between peripheral trunks of the IoT, eventually provided with lower parts of the communication stacks
- Gateways can also provide: pre-processing, **security**, scalability, service discovery, **geo-localization**, **billing**, etc.

- Pre-processing:
  - data **buffering**: temporarily store data to wait for connectivity or to increase efficiency
  - data **efficiency**: temperature read every 1s, but only per-minute average sent
  - data **aggregation**: water level from different silos, but only the sum is sent
  - data **filtering**: send temperature values only if grater than 25°C

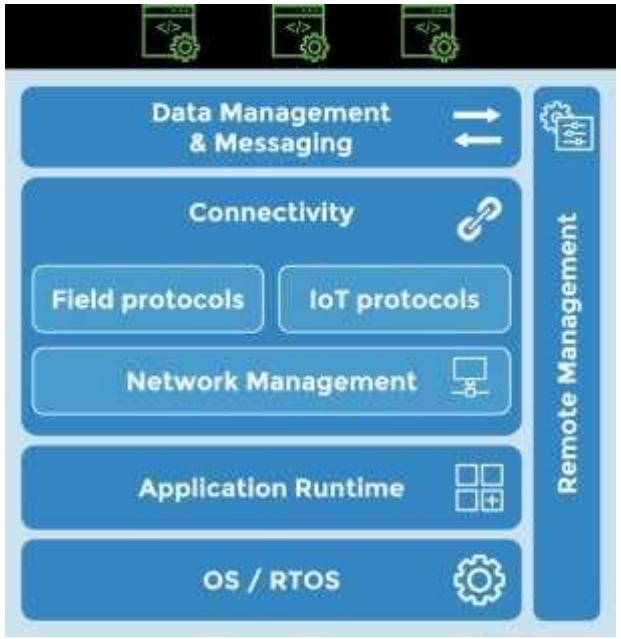
With/Without Gateway

- If there is no gateway, things have to send/receive data on their own
- In case of constrained devices, reduced set of capabilities, e.g.,
  - no security since cryptography is CPU-intensive
  - no data buffering, filtering aggregation
  - no programmability
  - …

Stack for constrained devices



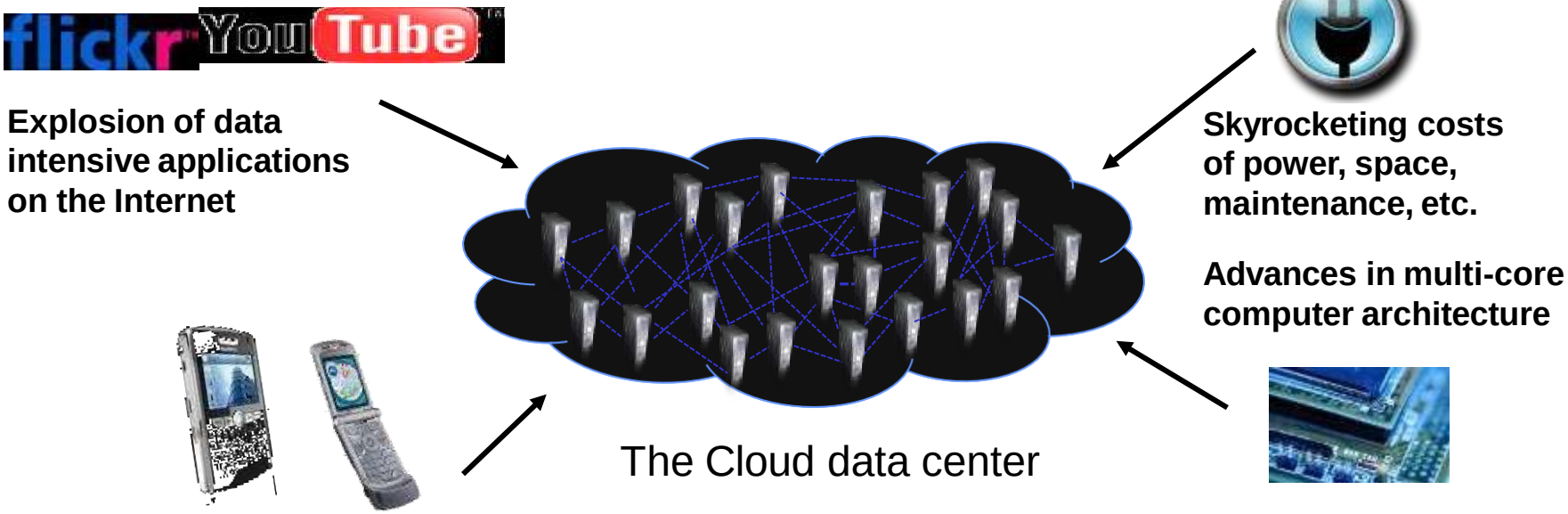
Stack for gateways



Cloud computing: problem space

*"It starts with the premise that the **data services and architecture** should be on **servers**. We call it **cloud computing** – they should be in a 'cloud' somewhere. And that if you have the right kind of **browser** or the right kind of access, it doesn't matter whether you have a PC or a Mac or a mobile phone or a BlackBerry or what have you – or new devices still to be developed – you can get access to the cloud…"*

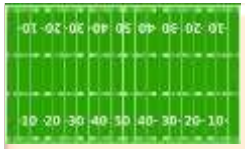
Dr. Eric Schmidt, Google CEO, August 2006



Clouds are Cheaper… and Winning…

- Range in size from “edge” facilities to **megascale**
- **Scale economies**
- Approximate **costs for a small size center** (1K servers) and a **larger**, 50K server center

Technology	Cost in small-sized Data Center	Cost in Large Data Center	Cloud Advantage
Network	$95 per Mbps/ month	$13 per Mbps/ month	7.1
Storage	$2.20 per GB/ month	$0.40 per GB/ month	5.7
Administration	~140 servers/ Administrator	>1000 Servers/ Administrator	7.1



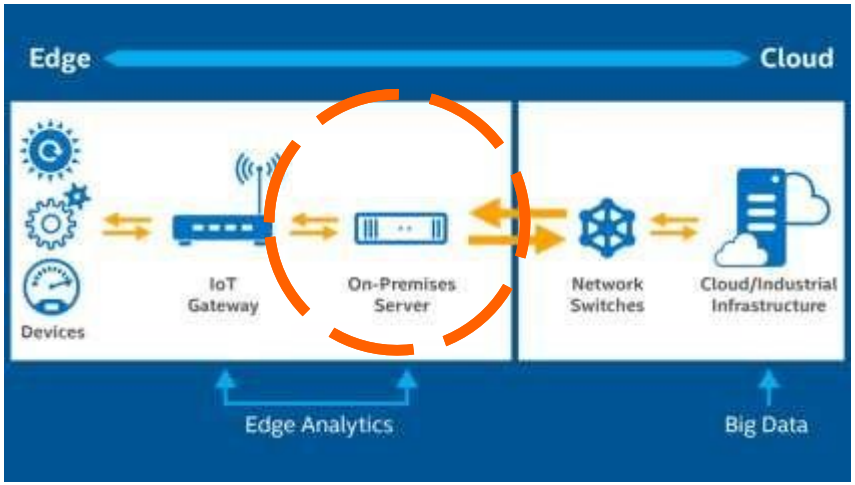
Each data center is
11.5 times
the size of a football field

Cloud Computing: a brief introduction

- Primary concepts
  - IT **on demand** pricing
  - best benefits in a **reliable** context
  - pool of **virtualized** computer resources
  - rapid live **providing** while demanding
  - systems on **scaling** architecture

- What is a Cloud
  - one Cloud is capable of providing **IT resources "as a service"**
  - one Cloud is an IT service delivered to users that have:
    - **reduced incremental management costs** when additional IT resources are added
    - services oriented management architecture
    - massive scalability

Beyond the cloud:
From Cloud Computing to Fog/Edge Computing



- First evolution wave: **IoT Cloud Computing** architecture
  - most of the computation on the Cloud
  - **only gateways are deployed close to things**
  - gateways perform **few and simple tasks**
- Second evolution wave: IoT **Fog/Edge** Computing architecture
  - additional **relatively powerful devices**
  - **close to things**, but between gateways and the Cloud
  - **complex analytical tasks** on the client-side, before sending data to the Cloud

Fog/Edge Computing for IoT

- Cloud models are not designed for the volume, variety, and velocity of data that the IoT generates
- Fog/Edge Computing allows to
  - minimize **latency**
  - conserve network **bandwidth**
  - address **security** concerns in transit and at rest
  - **move data** to the best place for processing

- When to consider Fog/Edge Computing
  - data is collected at the **extreme edge**: vehicles, ships, factory floors, roadways, railways, etc.
  - **thousands or millions of things** across a large geographic area are generating data
  - it is necessary to analyze and **act on data promptly**, in less than a second

Fog/Edge Computing for IoT use case: Rails

- **Improve passenger safety**
  - analyze and correlate data from cameras on the trains and at stations
  - monitor sensors on wheels and brakes to determine when parts need service before failure causes an accident
- **Prevent cybersecurity attacks**
  - take automated actions such as suspending operations or transferring control to a failover system
- **Alert drivers** to treacherous conditions ahead
  - Fog nodes gather sensor data on tracks and trains to detect unsafe conditions

Fog/Edge Computing for IoT use case: Manufacturing

- **Increase agility**
  - quickly change production lines and introduce new products
- **Reduce downtime**
  - predictive maintenance to avoid costly equipment downtime
  - Fog/Edge nodes collect machine data and report early signs of problems
- Continually **confirm that safety systems are intact**
  - analyze machine data in real-time
  - promptly shut down compromised equipment automatically, without waiting for a human to respond to an alert

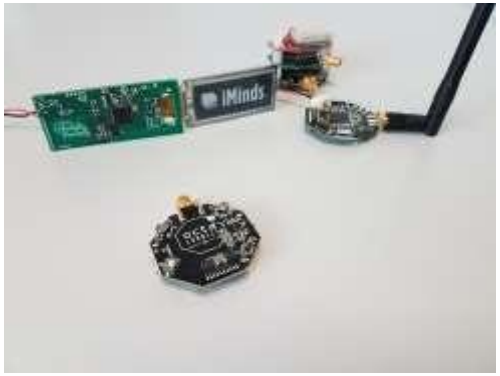
IoT Standards and Protocols

Outline

1. Introduction to IoT
   - definition, enabling technologies and concepts to better understand the general framework of IoT solutions
   - layering architecture, cloud computing vs. fog/edge computing

2. Most relevant components of IoT solutions
   - **devices**, wireless communication protocols, data exchange protocols

Constrained Devices



- "A node where some of the characteristics that are otherwise pretty much taken for granted for Internet nodes **at the time of writing** are not attainable, […] due to **cost, size, and energy constraints**"

- Significant constraints on:
  - maximum code complexity (ROM/Flash)
  - size of state and buffers (RAM)
  - available computational power
  - connectivity

Constrained Networks

- “A network where some of the characteristics pretty much taken for granted with link layers in common use in the Internet **at the time of writing** are not attainable”

- Significant constraints on:
  - low achievable **throughput**
  - high **packet loss**
  - highly **asymmetric links**
  - severe penalties for using **larger packets**
  - limits on **reachability** over time

Classes of Constrained Nodes

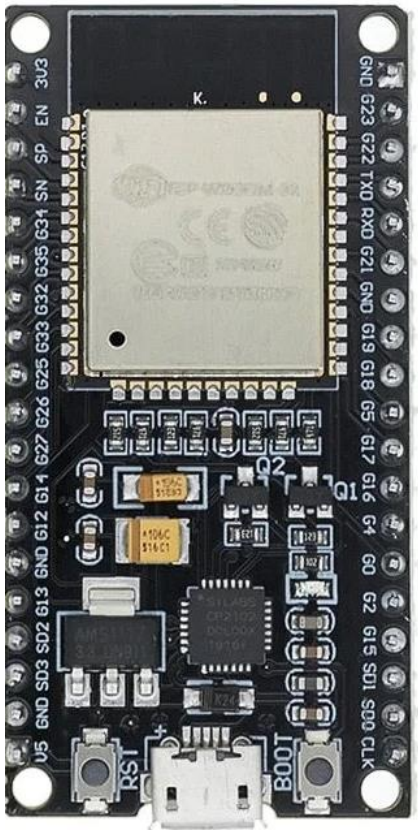
Name	Data size (RAM)	Code size (Flash)
Class 0	<< 10 KiB	<< 100 KiB
Class 1	~ 10 KiB	~ 100 KiB
Class 2	~ 50 KiB	~ 250 KiB

Values in 2014

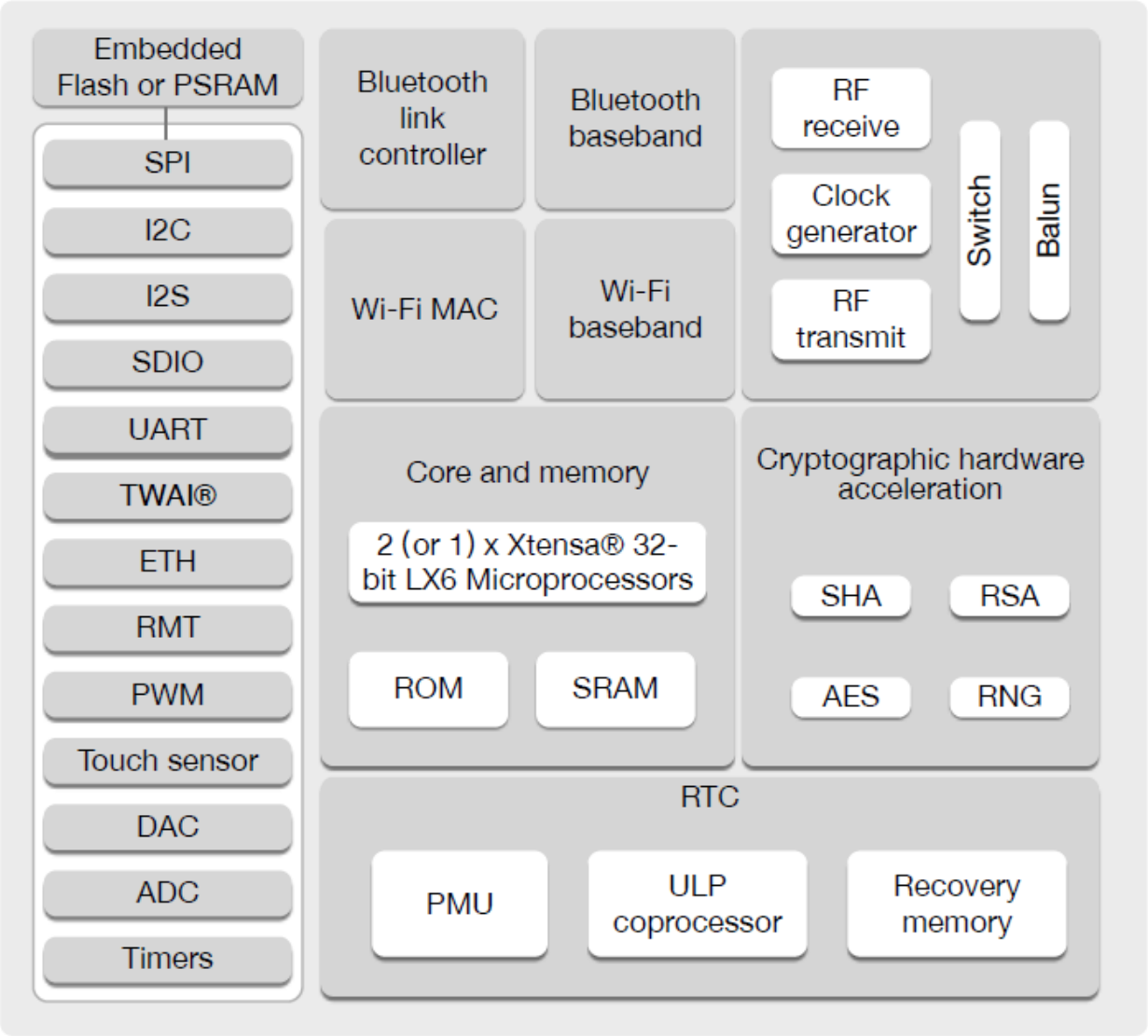
- C0 Devices
  - no direct secure Internet connection
  - use larger devices as gateways/proxies
  - preconfigured and rarely reconfigured
- C1 Devices
  - can use environment specific protocols, e.g., CoAP
  - no access to standard Internet protocols, e.g., HTTP, TLS
  - can be integrated into an IP network
- C2 Devices
  - can use most of protocols

IoT Device Example: ESP32

- Ultra Low Power Solution
- Wi-Fi Key Features
  - 802.11 b/g/n
  - 802.11 n (2.4 GHz), up to 150 Mbps
- Bluetooth Key Features
  - Compliant with Bluetooth v4.2 BR/EDR and Bluetooth LE specifications
  - Class-1, class-2 and class-3 transmitter without external power amplifier
  - Bluetooth Piconet and Scatternet
  - Multi-connections in Classic Bluetooth and Bluetooth LE
  - Simultaneous advertising and scanning
- CPU and Memory
  - Xtensa® single-/dual-core 32-bit LX6 microprocessor(s)
  - 4MB of Flash



ESP32 Block Diagram



Beyond Class 2: Single Board Computers

- In the last 5/10 years
  - tremendous improvement in CPU/memory capabilities
  - dramatically reduced costs

- Modern Single-Board Computers (SBCs) are very cheap (~100$) and powerful
  - can host complete operating systems
  - extensions via daughterboards
  - mostly designed to be mains-powered

Outline

1. Introduction to IoT
   - definition, enabling technologies and concepts to better understand the general framework of IoT solutions
   - layering architecture, cloud computing vs. fog computing

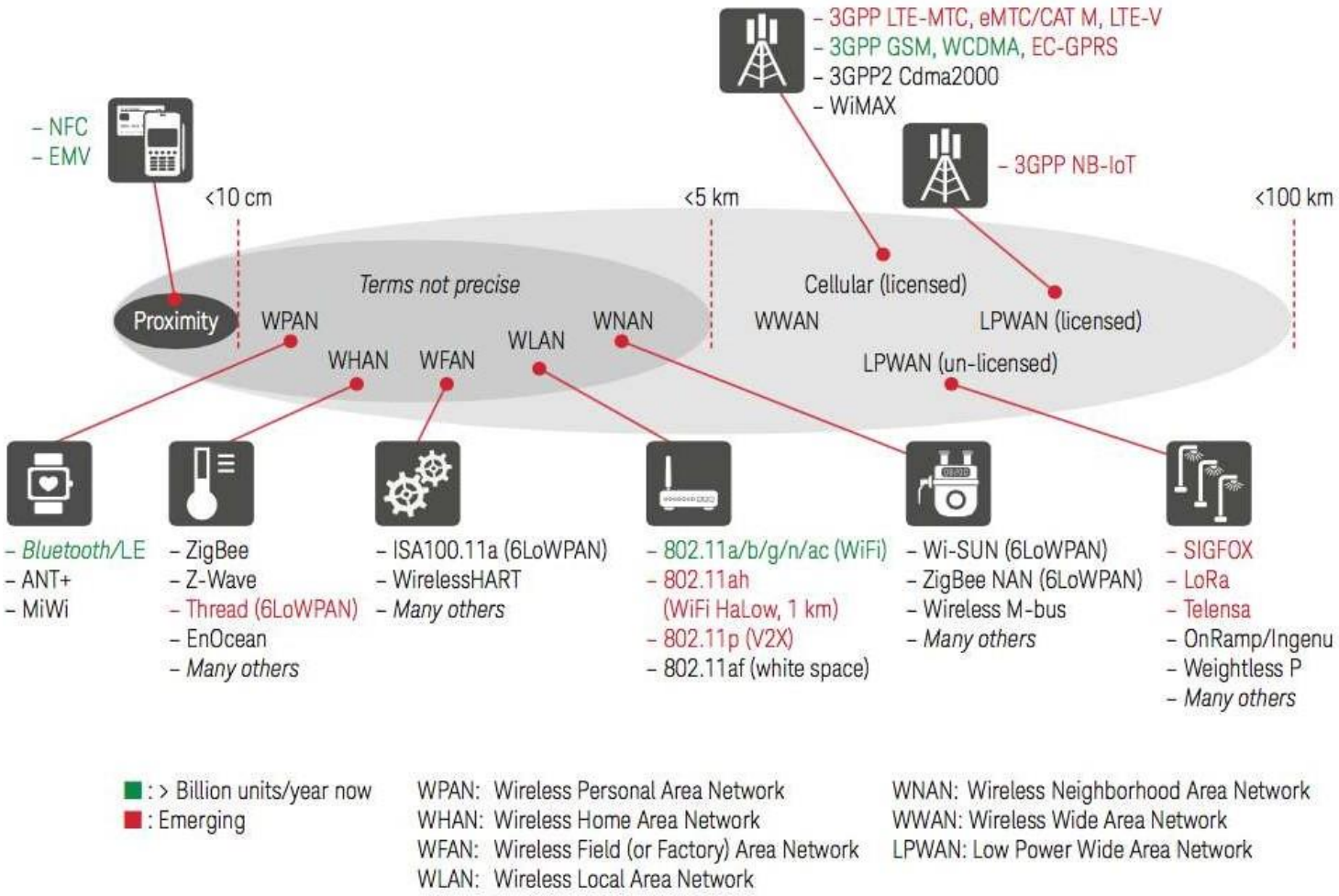
2. Most relevant components of IoT solutions
   - devices, **wireless communication protocols**, data exchange protocols

Wireless Communication Protocols for the IoT

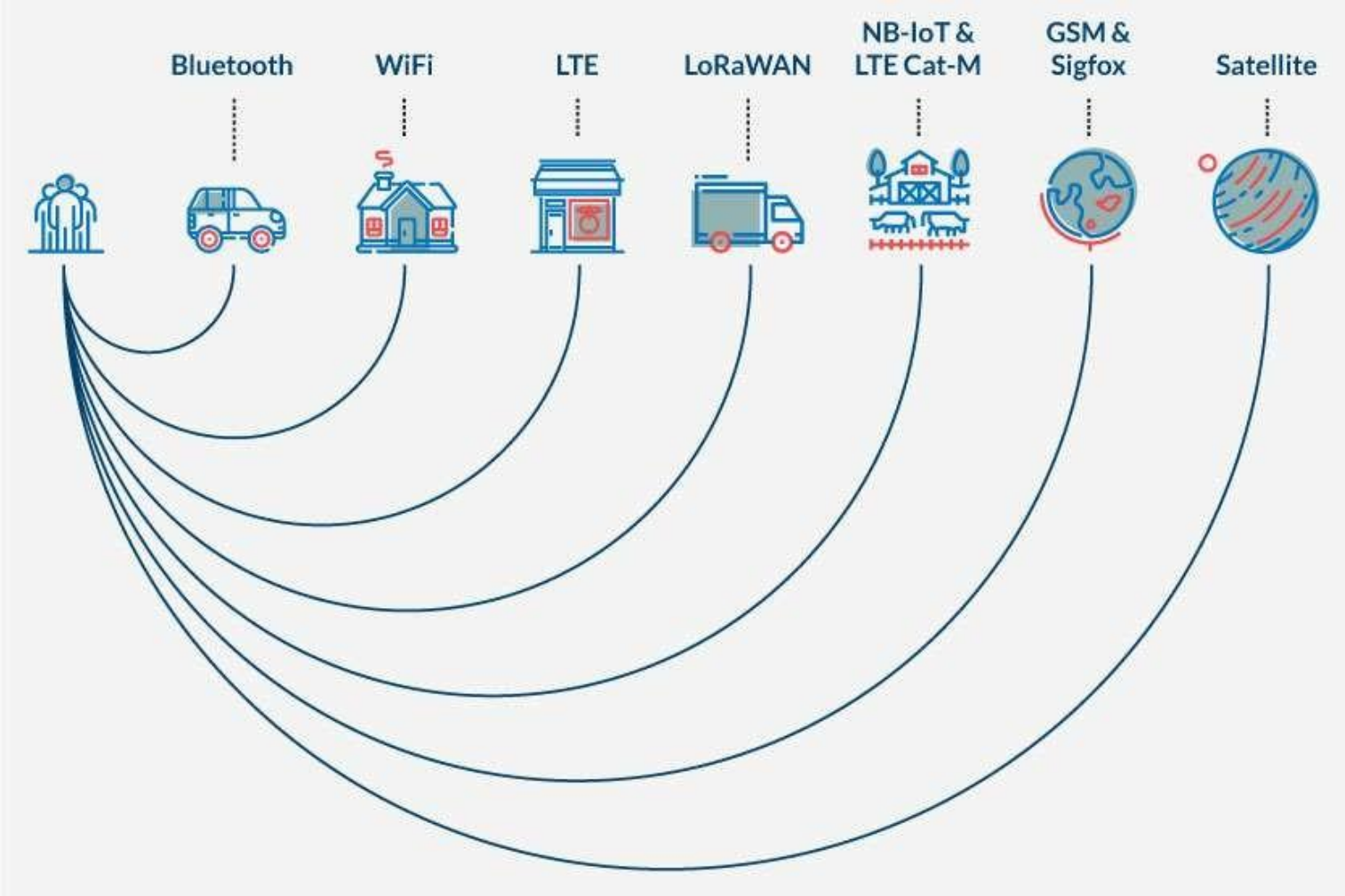
- The capability of connecting to things in a **seamless**, **ubiquitous**, and **cheap** manner have pushed the spread of IoT solutions

- IoT wireless communication protocols primarily differs in relation to
  - **coverage range**: from few cm to several km
  - **power consumption**: from few mW to several W
  - **bandwidth**: from few bytes per day to hundreds of MB/s
  - **security**: from plain data to strong encryption
  - **cost**: greatly varying, both for equipment and data transmission

Several Wireless Communication Protocols



Wireless Protocols per Coverage Range



Bluetooth

- Bluetooth and BLE are radio protocols for **Personal Area Networks (PAN)**
- Mostly these are on a person's body or in close proximity to them

- Typical range: very short, 20m (or less)
- Max output power: very limited, 0.003 W
- Bandwidth: limited, 0.7–2.1 Mbit/s
- Security: pairing task to exchange encryption keys
- Cost: cheap equipment, no transmission costs

- Good for: devices that stay **in close proximity of each other**, like between a smartphone and a headset, heart rate monitor, bicycle speedometer

Wireless LAN: IEEE 802.11

WiFi - IEEE 802.11

- WiFi is meant for **broadband network** connections in a confined area
- Normally less than 100 square meter per access point

- Typical range: short, 50m
- Max output power: medium/high, 0.1 W
- Bandwidth: large, up to hundreds of MB/s
- Security: suffered recent exploits, work in progress
- Cost: cheap equipment, no transmission costs

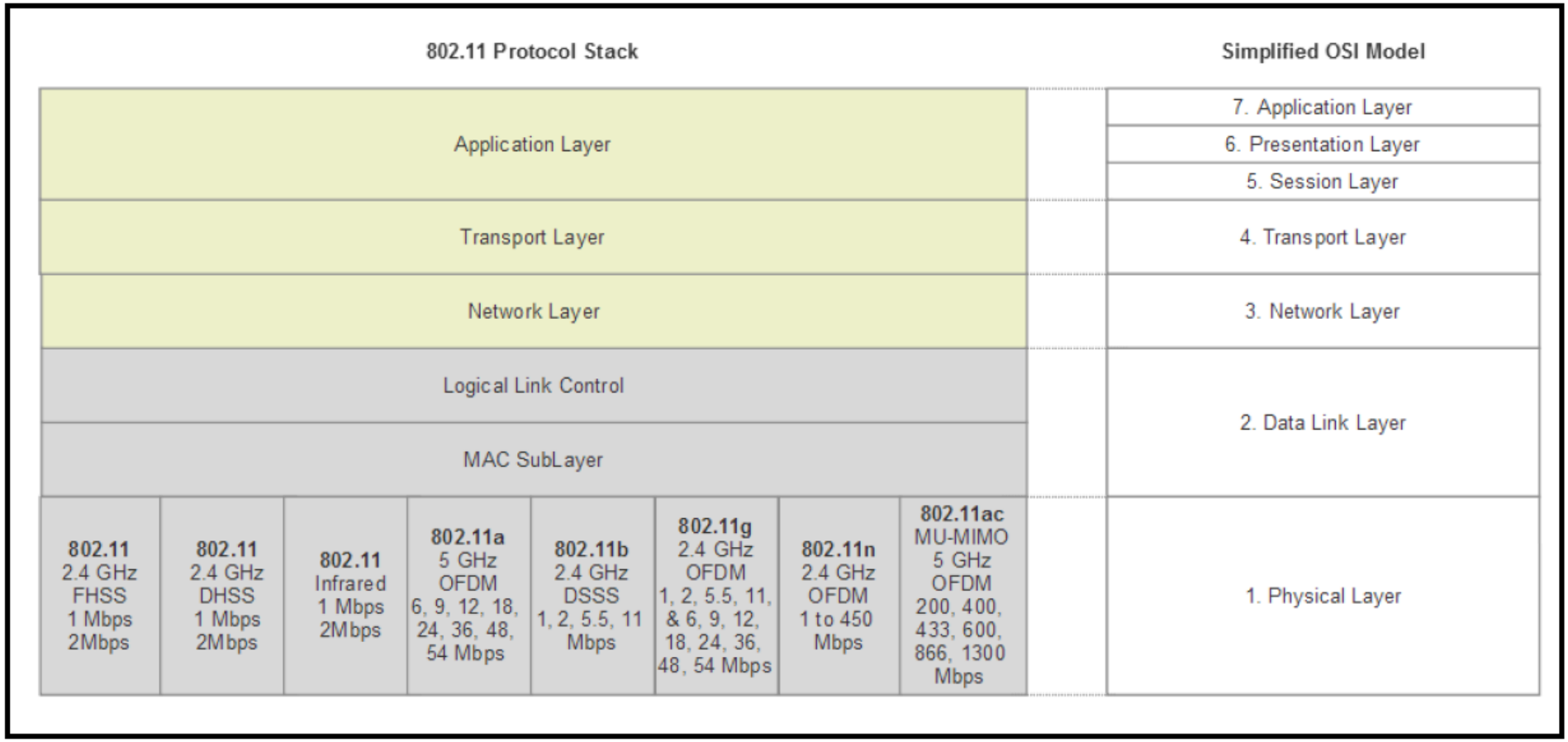
- Good for: security cameras, power meters or anything that is installed in a **fixed location, has power, and needs bandwidth**

IEEE 802.11suit of Protocol and Comparison

IEEE 802.11 Protocol	Use Case	Release Date	Frequency (GHz)	Bandwidth (MHz)	Streaming Data Rate per Channel min-max (Mbps)
802.11	First 802.11 design	Jun-97	2.4	22	1 to 2
a	Release simulataneously with 802.11b Less prone to interefernce than 802.11b	Sep-99	5 3.7	20	6 to 54
b	Release simulataneously with 802.11a Significant speed increase over 802.11a at improved range	Sep-99	2.4	22	1 to 11
g	Speed increase over 802.11b	Jun-03	2.4	20	6 to 54
n	Multiple antenna technology for improved speed, and range.	Oct-09	2.4 / 5	20 40	7.2 to 72.2 15 to 150
ac	Better performacne and coverage over 802.11n. Wider channel and improved modulation. Allows multiple users using MU-MIMO. Introduced beamforming.	Dec-13	5	20 40 80 160	7.2 to 96.3 15 to 200 32.5 to 433.3 65 to 866.7
ah	"WiFi HaLow" Designed for IoT and sensor networks. Very low power and wider range.	Dec-16	2.4 / 5	1 to 16	347
p	"Wireless Access in Vehicular Environments" "Intelligent Transport Systems" Dedicated Short Range Communication Trnasport uses cases: toll collection, safety and collision emergencies, vehicular networking.	Jun-09	5.9	10	27
af	"white WiFi" or "Super WiFi" Deploy unused spectrum in TV bands to provide last mile connectivity in India, Kenya, Singapoere, US and UK	Nov-13	0.470 to 0.710	6 to 8	568
ad	WiGig Alliance 60 GHz Wireless for HD video and projectors Audio and video transport and cable replacement	Dec-12	60	2160	4260
ax	"High Efficiency Wireless (HEW)" Next gen 802.11 4x increase in capacity over 802.11ac Average increase of 4x speed per user over 802.11ac Backwards compatible to 802.11a/b/g/n/ac Dense deployment scenarios	2019	2.4 / 5	20 40 80 160	450 to 10000

802.11 Architecture

From a stack perspective, the 802.11 protocols reside in the link layer (one and two) of the OSI model, as shown in the following figure:



802.11 Topologies

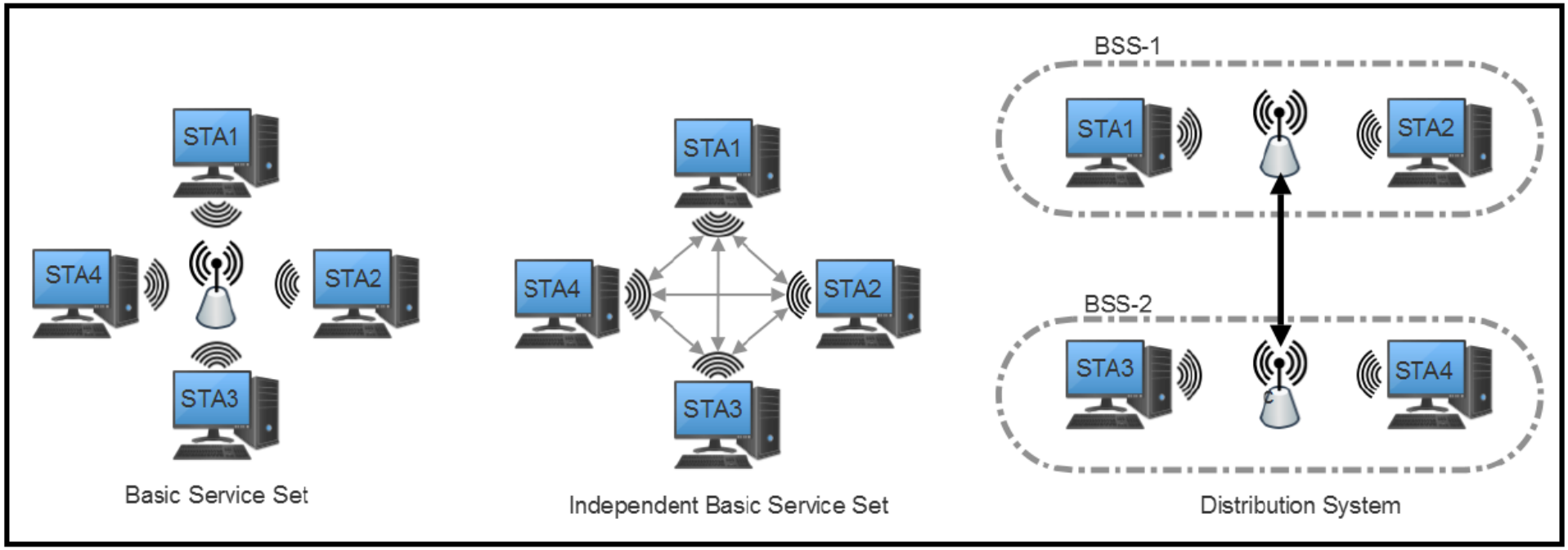
802.11 systems support three basic topologies:

- **Infrastructure:** In this form, a **Station** (**STA**) refers to an 802.11 endpoint device(like a Smartphone) that communicates with a central **access point** (**AP**). An AP can be a gateway to other networks (WAN), a router, or a true access point in a larger network. This is also known as **Infrastructure Basic Set Service** (**BSS**). This topology is a star topology.

- **Ad hoc:** 802.11 nodes can form what is called an **Independent Basic Set Service** (**IBSS**) where each station communicates and manages the interface to other stations. No access point or a star topology is used in this configuration. This is a peer-to-peer type of topology.

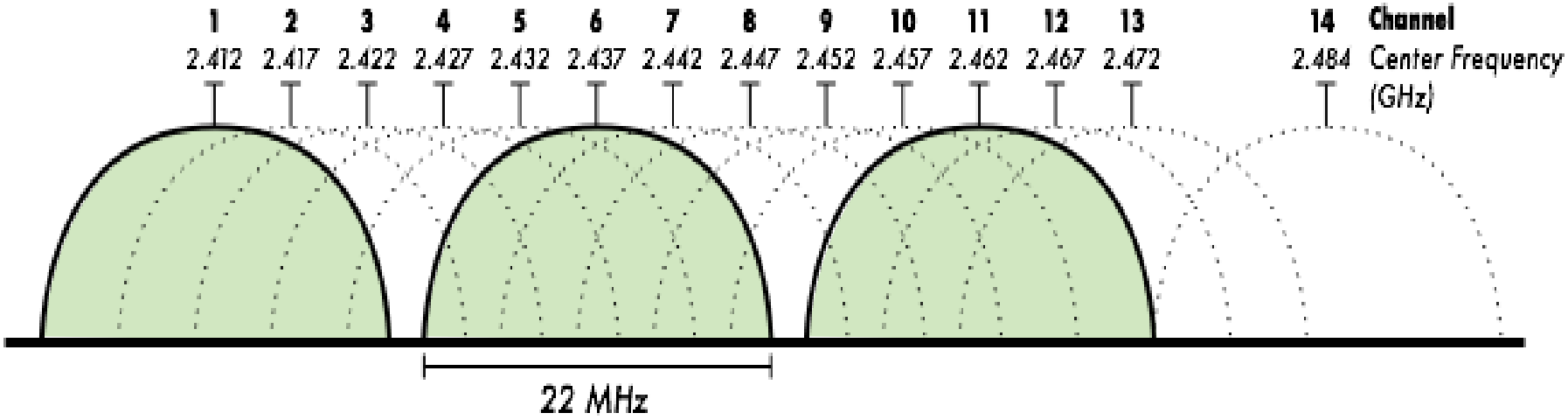
- **Distribution system (DS):** The DS combines two or more independent BSS networks through access point interconnects.

802.11 Topologies Diagram



IEEE 802.11 spectrum allocation

- The first 802.11 protocol used a spectrum in the 2 GHz and 5 GHz ISM region and evenly spaced channels roughly 20 MHz apart from each other.
- The channel bandwidth was 20MHz, but later amendments from IEEE allowed 5 MHz and 10 MHz to operate as well.
- Three of the channels are non-overlapping (1,6,11)



IEEE 802.11 modulation and encoding techniques

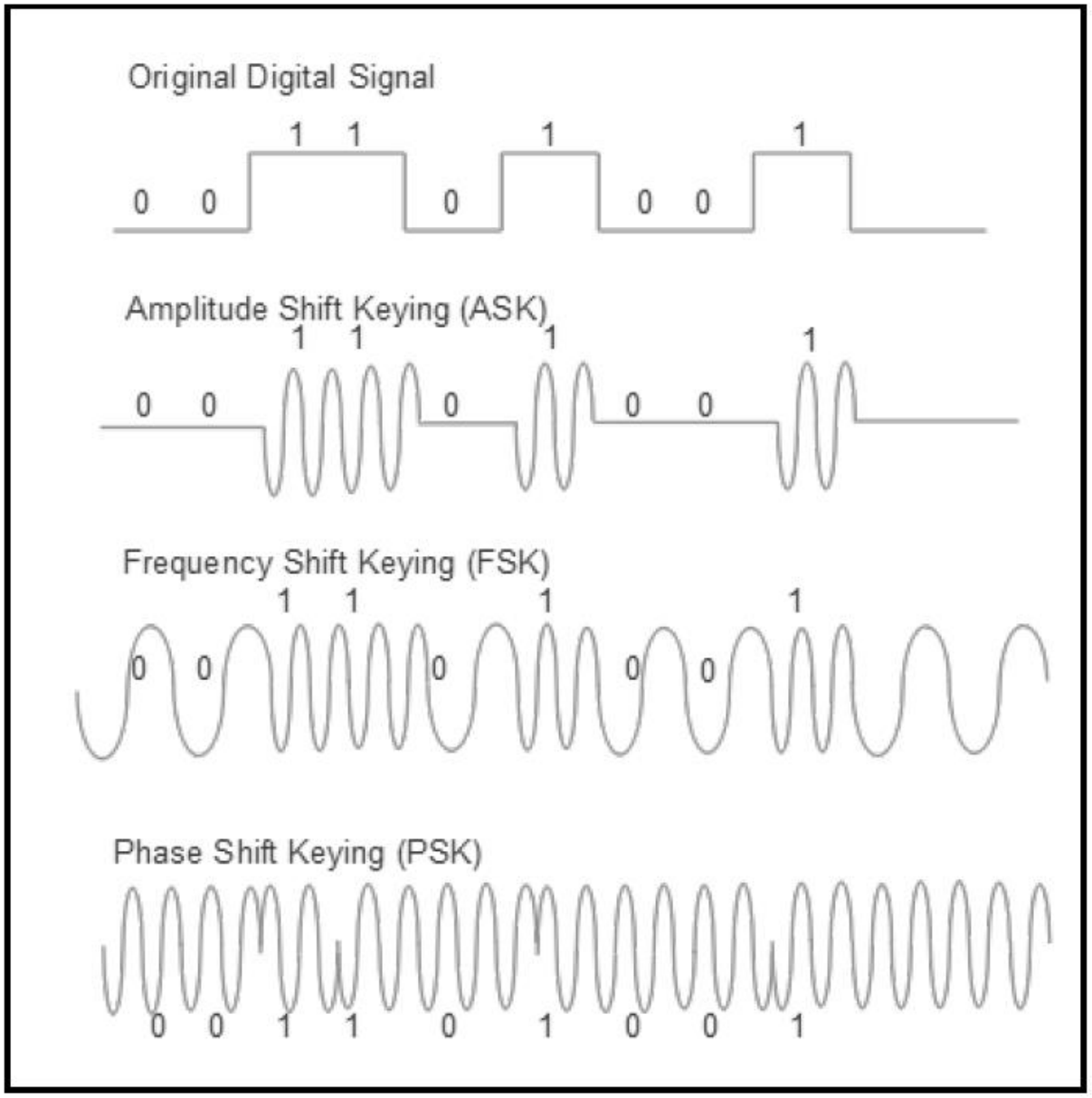
- **Amplitude Shift Keying (ASK)**: This is a form of amplitude modulation. Binary 0 is represented by one form of modulation amplitude and 1 a different amplitude.

- **Frequency Shift Keying (FSK)**: This modulation technique modulates a carrier frequency to represent 0 or 1. The simplest form shown in the following figure is **Binary Frequency Shift Keying** (**BPSK**), which is the form used in 802.11 and other protocols.

- Bluetooth and Z-Wave, those protocols use a form of FSK called **Gaussian Frequency Shift Keying** (**GFSK**) that filters the data through a Gaussian filter, which smooths the digital pulse (-1 or +1) and shapes it to limit spectral width.

- **Phase Shift Keying (PSK)**: Modulates the phase of a reference signal (carrier signal). Used primarily in 802.11b, Bluetooth, and RFID tags. PSK uses a finite number of symbols represented as different phase changes.

IEEE 802.11 modulation



802.11 standards interference mitigation techniques

- **Frequency Hopping Spread Spectrum (FHSS)**: Spreads signal over 79 nonoverlapping channels that are 1 MHz wide in the 2.4 GHz ISM band. Uses a pseudo-random number generator to start the hopping process.

- **Direct sequence spread spectrum**: First used in 802.11b protocols and has 22 MHz-wide channels. Each bit is represented by multiple bits in the signal transmitted. The data being transmitted is multiplied by a noise generator. This will effectively spread the signal over the entire spectrum evenly using a pseudorandom number sequence (called the *Pseudo-Noise* PN code).

- **OFDM**: Used in IEEE 802.11a and the newer protocols. This technique divides a single 20 MHz channel into 52 sub-channels (48 for data and four for synchronization and monitoring) to encode data using QAM and PSM.

IEEE 802.11 MIMO

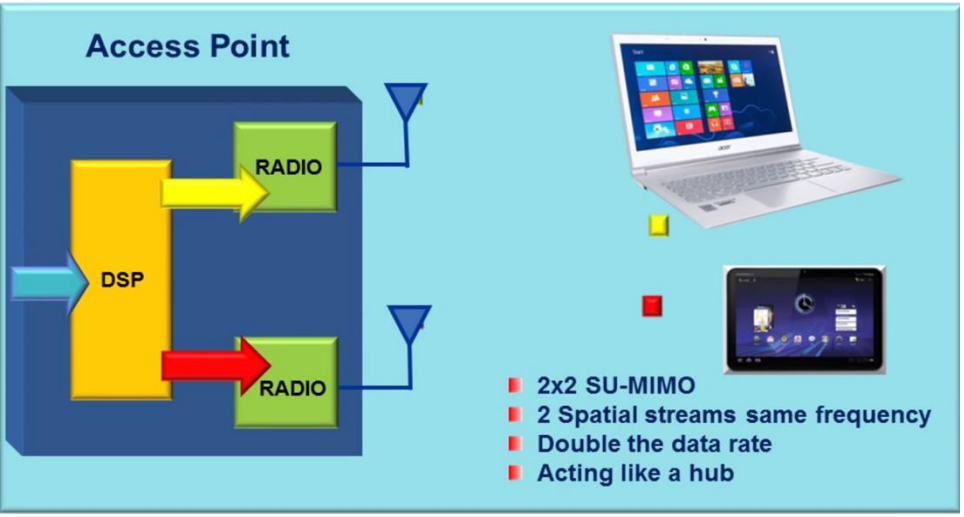
- MIMO is the acronym that refers to multiple input multiple output.
- MIMO exploits a RF phenomenon called multipath. Multipath transmission implies that signals will reflect off walls, doors, windows, and other obstructions. A receiver will see many signals each arriving at different times via different paths. Multipath tends to distort signals and cause interference, which eventually degrades signal quality (this effect is called multipath fading).
- With the addition of multiple antennas, a MIMO system can linearly increase the capacity of a given channel by simply adding more antennas. There are two forms of MIMO:

- **Spatial diversity**: This refers to transmit-and-receive diversity. A single stream of data is transmitted on multiple antennas simultaneously using space-time coding. These provide improvements in the signal to noise ratio.

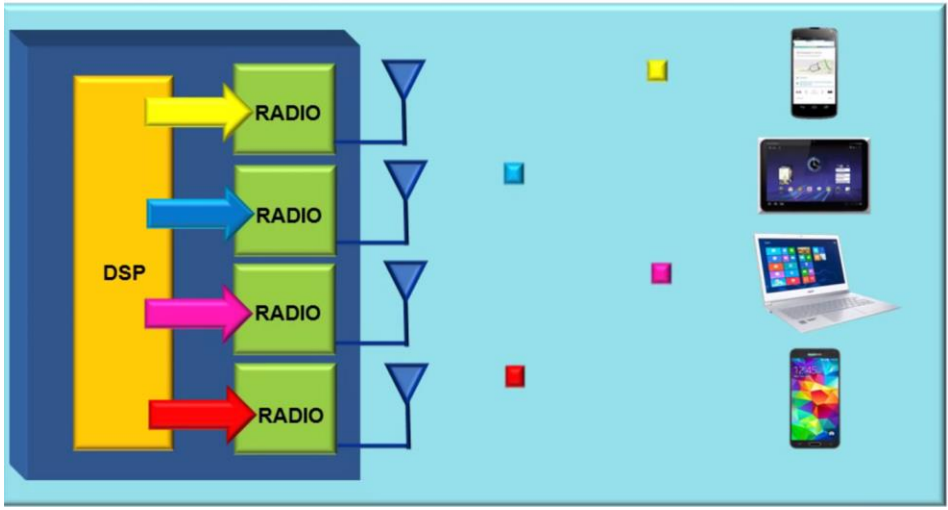
- **Spatial multiplexing**: Used to provide additional data capacity by utilizing the multiple paths to carry additional traffic; that is, increasing the data throughput capability.

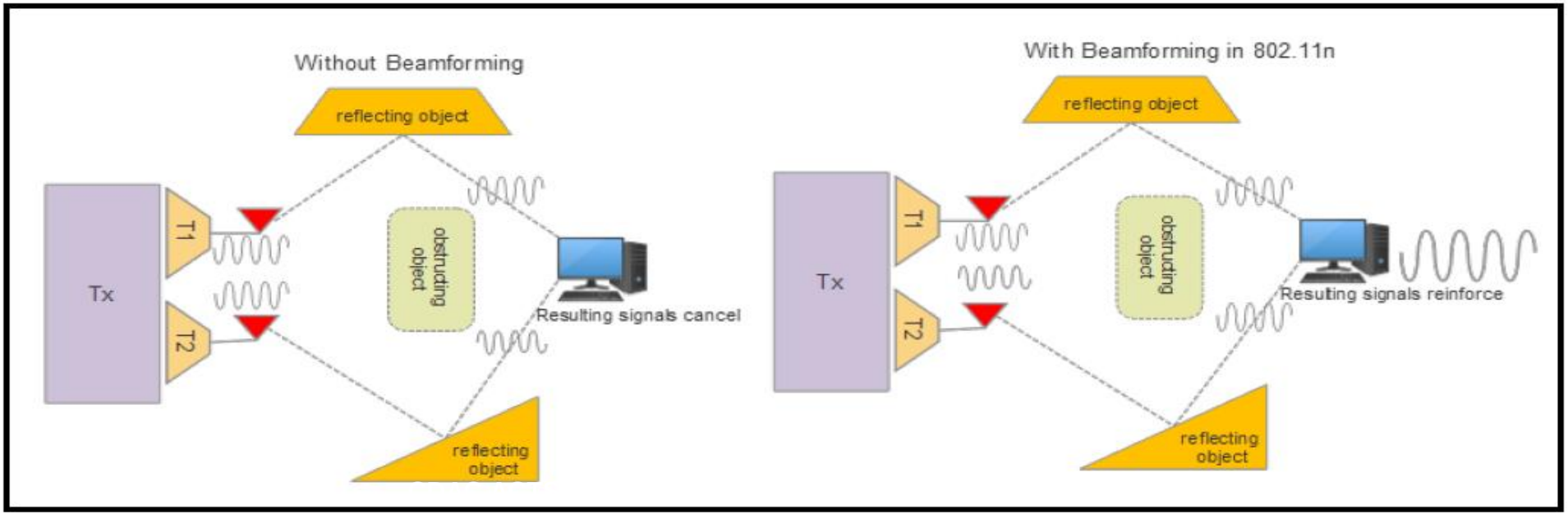
SU-MIMO vs MU-MIMO

Single User-Multiple In Multiple Out (SU-MIMO)

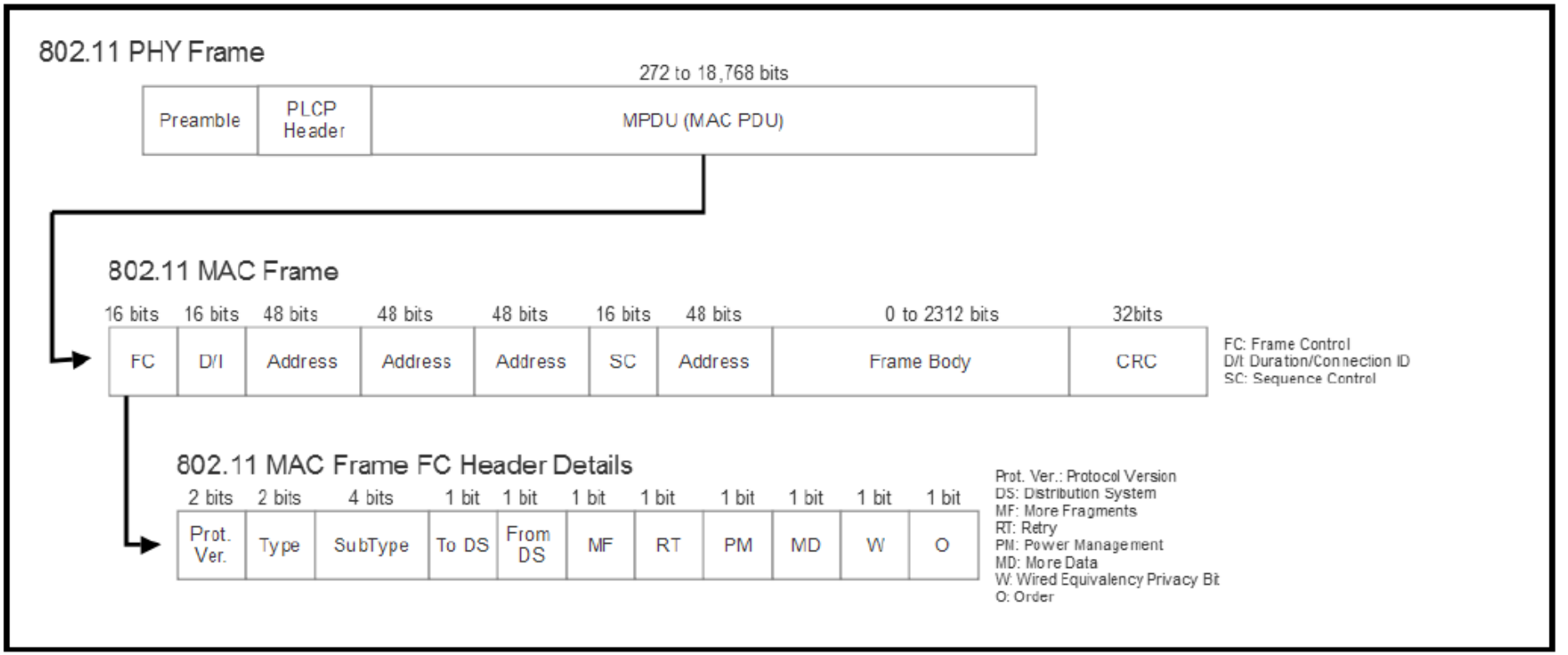


Multi User-Multiple In Multiple Out (MU-MIMO)





IEEE 802.11 packet structure



Address fields: 802.11 can manage four MAC addresses in the following order:
Address 1: Receiver
Address 2: Transmitter
Address 3/4: Used for filtering

IEEE 802.11 operation

- An STA is considered a device equipped with a wireless network interface controller. An STA will always be listening for active communication in a specific channel. The first phase of connecting to Wi-Fi is the scanning phase.
- After a channel is selected, the device performing the scan will receive beacons and probe requests from nearby STAs. An access point may transmit a beacon, and if the STA receives the transmission, it may progress to join the network.

The beacon packet contains information needed by the STA:

- **SSID**: Service Set ID. 1 to 32-character network name (this field can optionally be hidden by setting the SSID length to zero. Even if it is hidden the other portions of the beacon frame are transmitted as usual.

- **BSSID:** Basic Service Set ID. Unique 48-bit following layer-2 MAC address conventions. Formed by the combination of the 24-bit Organization Unique Identifier and the manufacturer's assigned 24-bit identifier for the radio chipset.

IEEE 802.11 security

Types of authentication used on Wi-Fi WLANs and various strengths and weaknesses:
WEP: Wired equivalent privacy. This mode sends a key in plain text from the client. The key is then encrypted and sent back to the client. WEP uses different size keys but they are typically 128 bit or 256 bit. WEP uses a shared key, which means that the same key is available to all clients. It can be easily compromised by simply listening and sniffing for all the authentication frames coming back to clients joining a network to determine the key used for everyone.

WPA: Wi-Fi protected access (or WPA-Enterprise) was developed as the IEEE 802.11i security standard to replace WEP .One significant difference is WPA uses a **Temporal Key Integrity Protocol** (**TKIP**), which performs per-packet key mixing and rekeying. This means that each packet will use a different key to encrypt itself, unlike in the case of WEP. Data can now be encrypted.

WPA-PSK: WPA pre-shared key or WPA-Personal. This mode exists where there is no 802.11 authentication infrastructure. Here, one uses a passphrase as a pre shared key. Each STA can have their own pre-shared key associated with its MAC address.

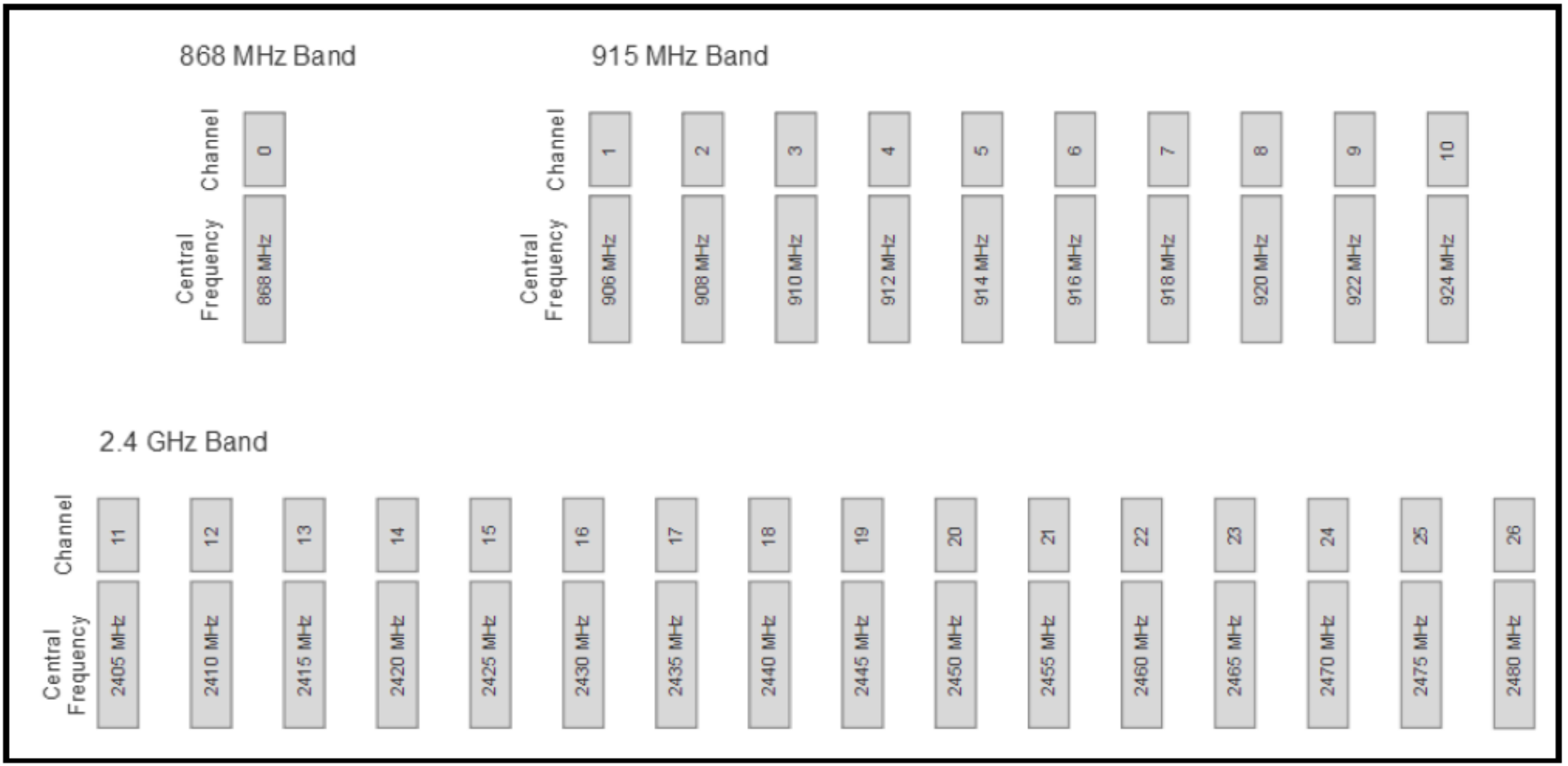
Wireless PAN: IEEE 802.15.1 & 802.15.4, Zigbee

802.15 standards

- The 802.15 group was initially formed to focus on wearable devices (coining the phrase personal area network).
- Their work has expanded significantly and now focuses on higher data rate protocols, meter to kilometer ranges, and specialty communications.
- Over one million devices are shipped each day using some form of 802.15.x protocol.
- **802.15**: Wireless personal area network definitions
- **802.15.1**: Original foundation of the Bluetooth PAN
- **802.15.4:** Low data rate, simple, simple design, multi-year battery life specifications (Zigbee)

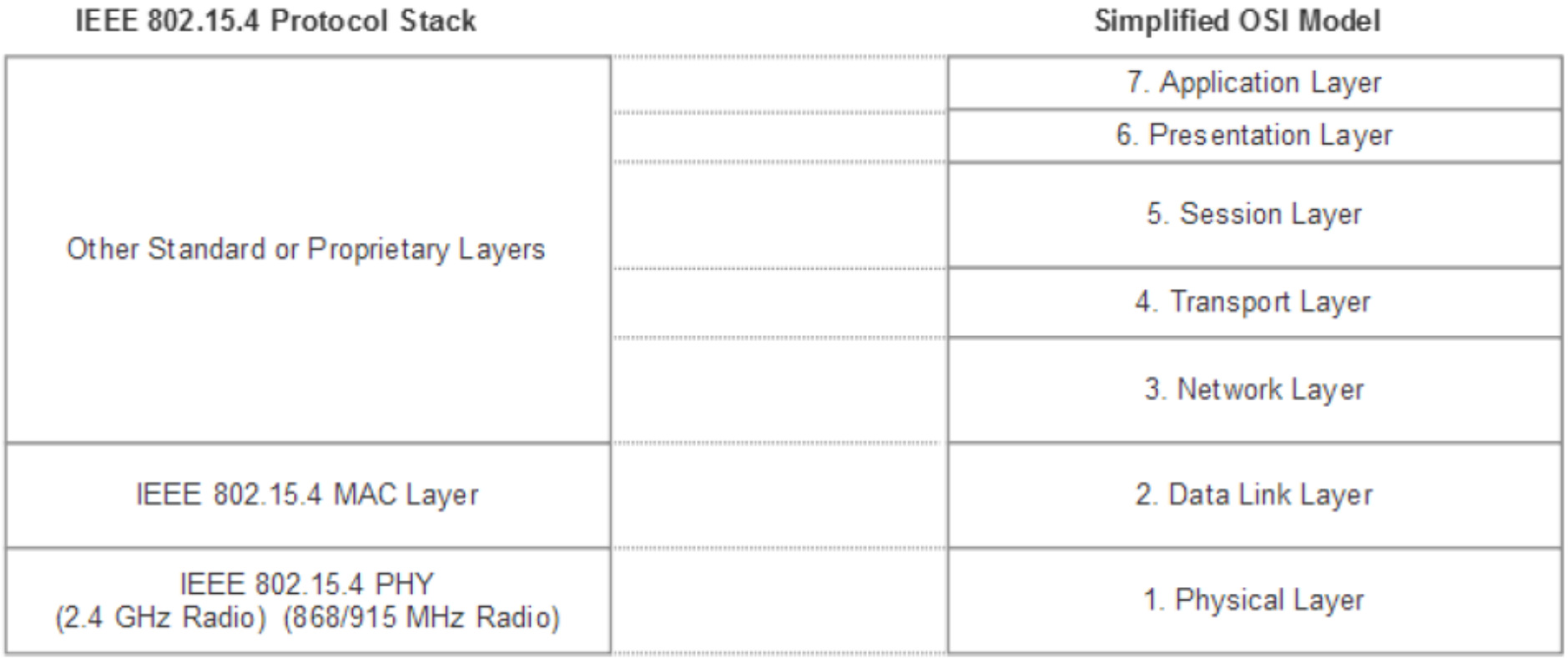
IEEE 802.15.4 architecture - Frequency

The IEEE 802.15.4 protocol operates in the unlicensed spectrum in three different radio frequency bands: 868 MHz, 915 MHz, and 2400 MHz.



IEEE 802.15.4 architecture – Principle & Protocol Stack

To manage a shared frequency space, 802.15.4 and most other wireless protocols use some form of **Carrier Sense Multiple Access Collision Avoidance** (**CSMA/CA**).
Since it is impossible to listen to a channel while transmitting on the same channel, collision detection schemes don't work; therefore, we use collision avoidance.



Communications in IEEE 802.15.4

There are two types of communication in IEEE 802.15.4:

1. **Beacon** communication.
2. **Beaconless** communication.

- For a ***beacon-based network***, the MAC layer can generate beacons that allow a device to enter a PAN as well as provide timing events for a device to enter a channel to communicate.
- The beacon is also used for battery-based devices that are normally sleeping.
- The device wakes on a periodic timer and listens for a beacon from its neighbors.

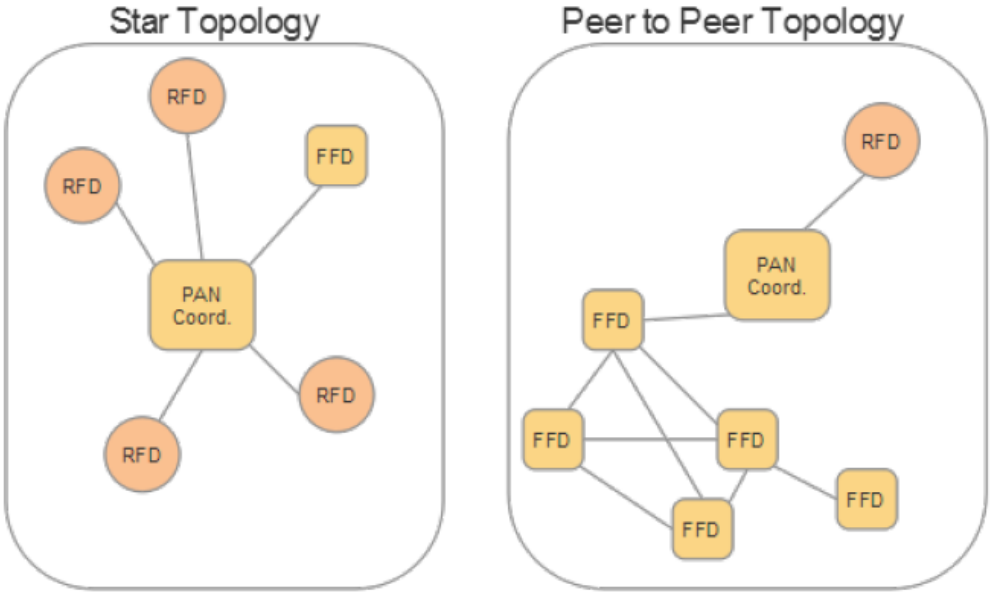
- IEEE 802.15.4 allows for ***beacon-less networking***. This is a much simpler scheme where no beacon frame is transmitted by the PAN coordinator. It implies, however, that all nodes are in a receiving mode all the time. This provides full-time contention access using unslotted CSMA/CA.
- This mode will consume much more power than beacon-based communication.

IEEE 802.15.4 topology

There are two fundamental device types in IEEE 802.15.4:

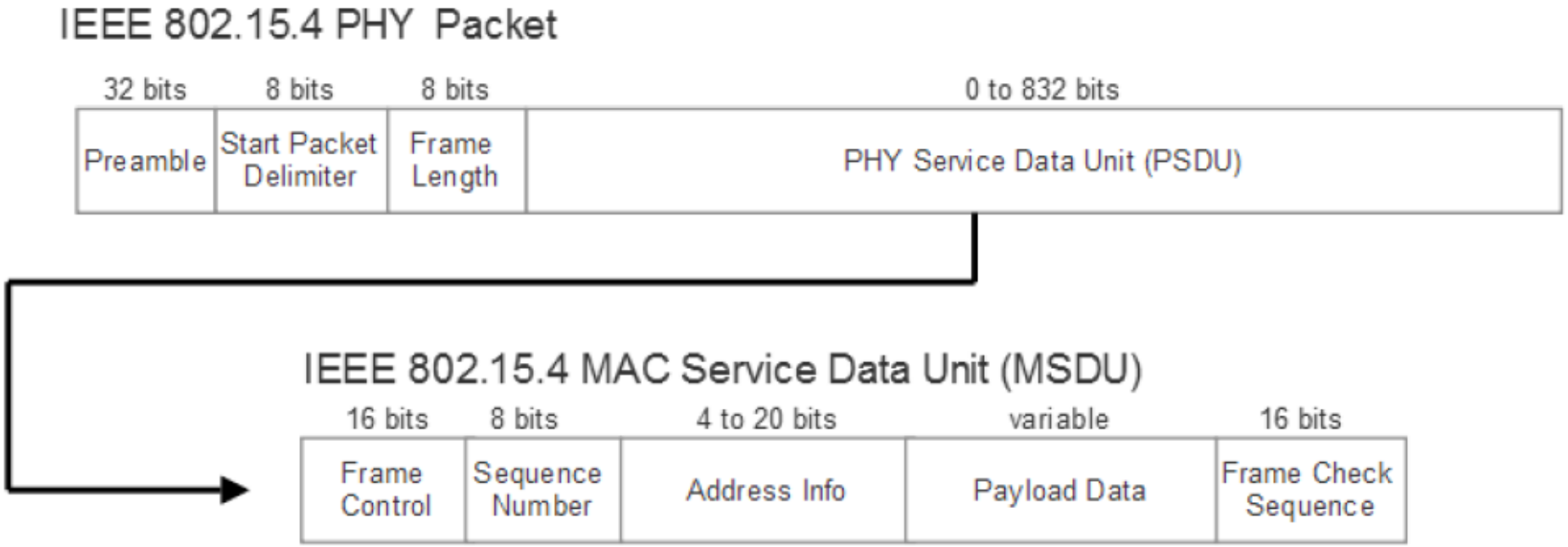
- **Full function device (FFD)**: Supports any network topology, can be a network (PAN) coordinator and can communicate to any device PAN coordinator
- **Reduced function device (RFD)**: Limited to only a star topology, cannot perform as a network coordinator, can only communicate with a network coordinator

The ***star topology*** is the simplest but requires all messages between peer nodes to travel through the PAN coordinator for routing.
A ***peer-to-peer topology*** is a typical mesh and can communicate directly with neighbor nodes.



IEEE 802.15.4 address modes and packet structure

- The standard dictates that all addresses are based on unique **64-bit values** (IEEE address or MAC address).
- However, **to conserve bandwidth and reduce the energy of transmitting** such large addresses, 802.15.4 allows a device joining a network to "trade-in" their unique 64-bit address for a short **16-bit local address**, allowing for more efficient transmission and lower energy.



IEEE 802.15.4 start-up sequence

IEEE 802.15.4 maintains a process for startup, network configuration, and joining of existing
networks. The process is as follows:

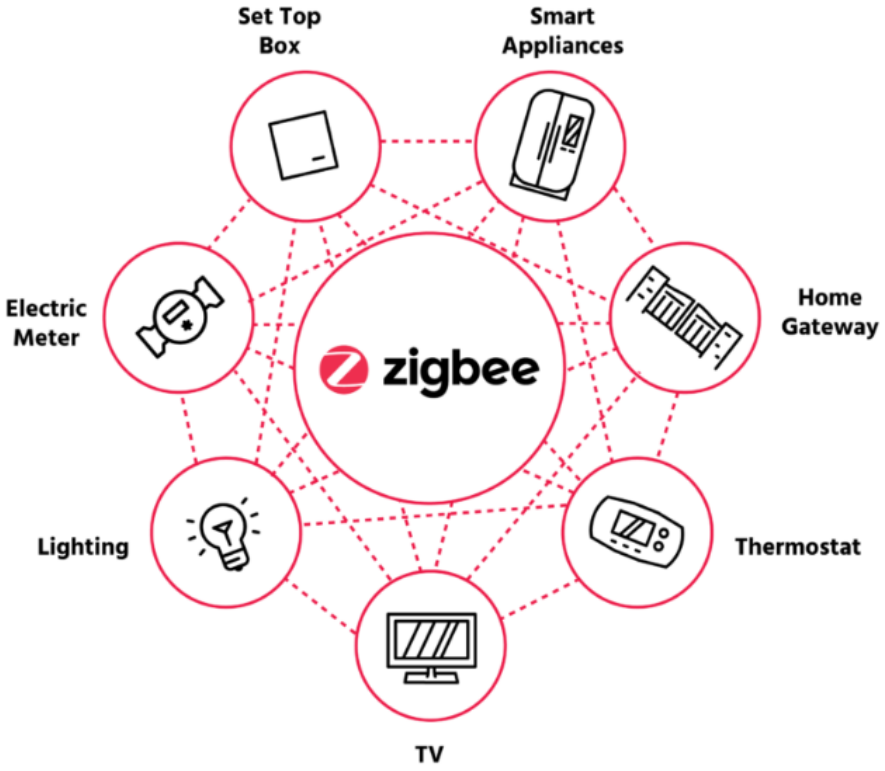
1. Device **initializes** its **stack** (PHY and MAC layers).
2. PAN **coordinator** is **created**. Each network has only one PAN coordinator.
3. The PAN coordinator will **listen** to **other networks** it has access to and **derives** a **PAN ID** that is unique to the PAN it will administer.
4. The PAN coordinator will **choose** a **specific radio frequency** to use for the network.
5. The network will be started by configuring the PAN coordinator and then **starting** the device in **coordinator mode**. At this point, the PAN coordinator can **accept requests**.
6. Nodes can join the network by finding the PAN coordinator using an active channel scan where it **broadcasts** a **beacon request** across all its frequency channels. In a beacon-based network, the PAN coordinator will routinely send out a beacon and the device can perform a passive channel scan and listen for the beacon. The device will then send an **association request**.
7. The PAN coordinator will determine if the **device should** or can join the network. If accepted, the PAN coordinator will **assign a 16-bit short address** to the device.

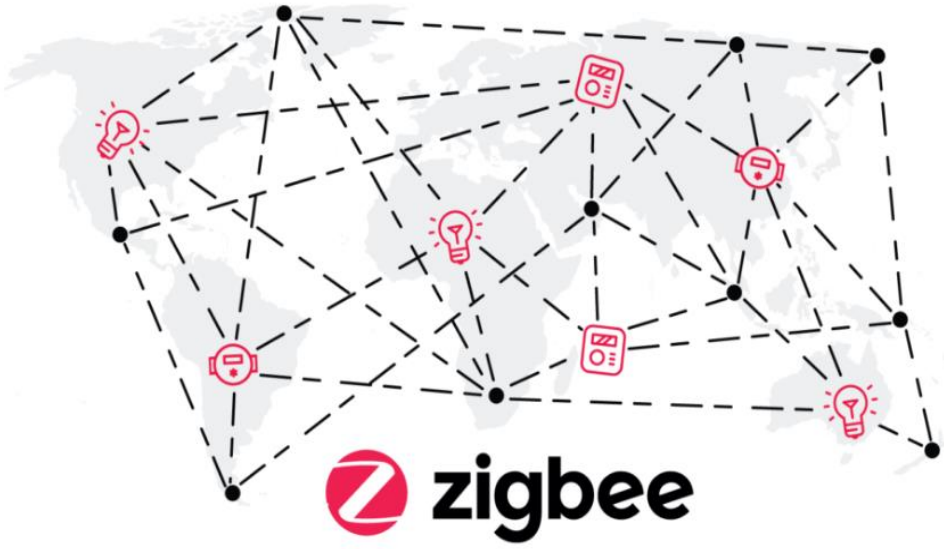
IEEE 802.15.4 security

- The IEEE 802.15.4 standard includes security provisions in the form of **encryption** and **authentication**. The architect has flexibility in the security of the network based on cost, performance, security, and power.
- AES-based encryption which uses a block cipher with a counter mode can be used.

Zigbee

- Zigbee is a WPAN protocol based on the IEEE 802.15.4 foundation targeted for commercial and residential IoT networking that is constrained by cost, power, and space.
- Zigbee got its name from the concept of a bee flying. As a bee flies back and forth between flowers gathering pollen, it resembles a packet flowing through a mesh network - device to device.





Zigbee Overview

- Zigbee is based on 802.15.4 but layers on network services similar to TCP/IP. It can form networks, discover devices, provide security, and manage the network.
- It does not provide data transport services or an application execution environment.
- It is essentially a mesh network, its self-healing and ad hoc in form.

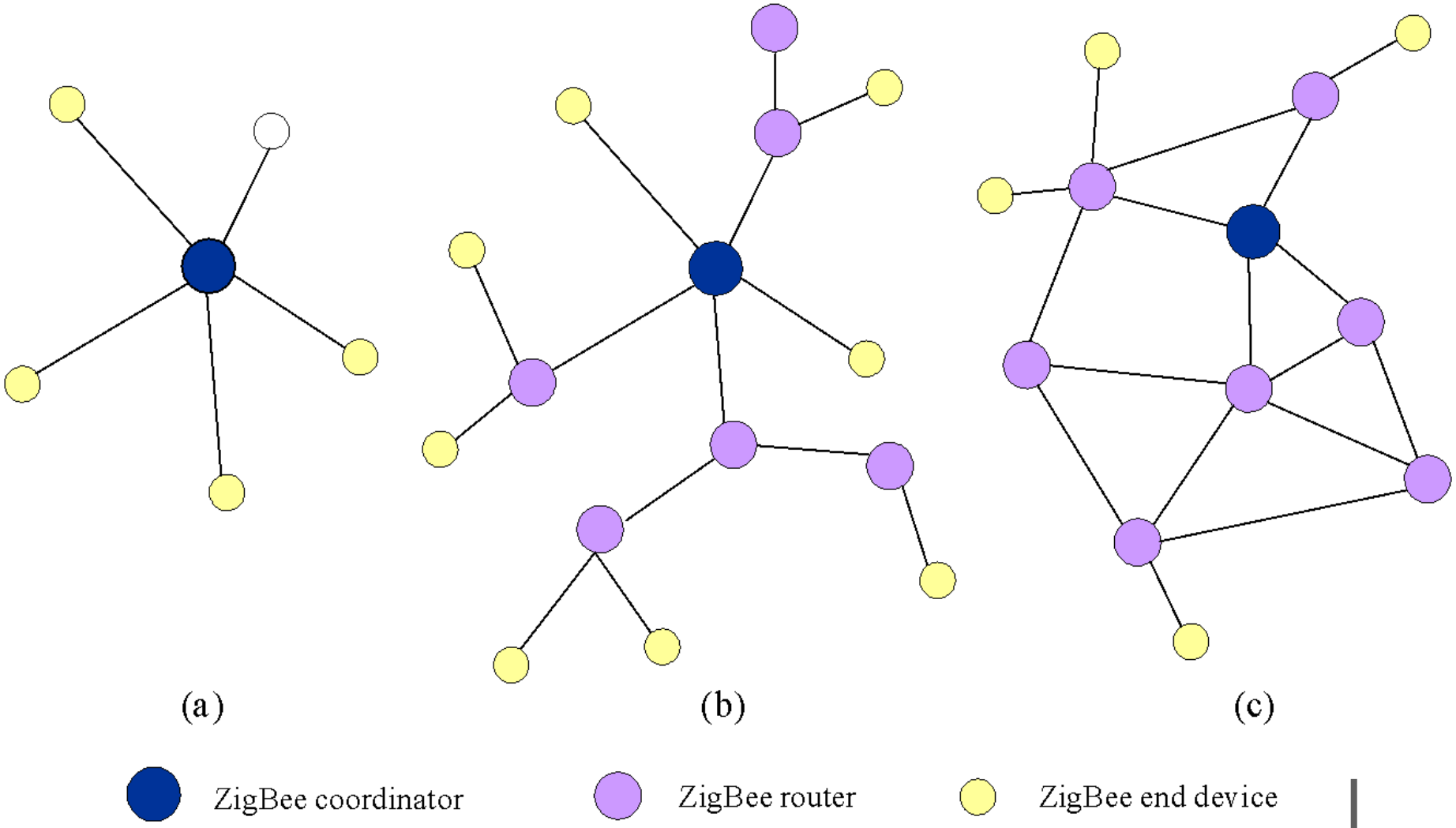
There are three principal components in a Zigbee network.
Zigbee controller (ZC): Highly capable device on a Zigbee network that is used to form and initiate network functions. Each Zigbee network will have a single ZC that fulfills the role of an 802.15.4 2003 PAN coordinator (FFD).

Zigbee router (ZR): This component is optional but handles some of a load of mesh network hopping and routing coordination. It too can fulfill the role of an FFD and has an association with the ZC.

Zigbee end device (ZED): This is usually a simple endpoint device such as a light switch or thermostat. It contains enough functionality to communicate with the coordinator. It has no routing logic; therefore, any messages arriving at a ZED that are not targeted to that end device are simply relayed.

Zigbee Topologies

Three kinds of networks are supported: star, tree, and mesh networks



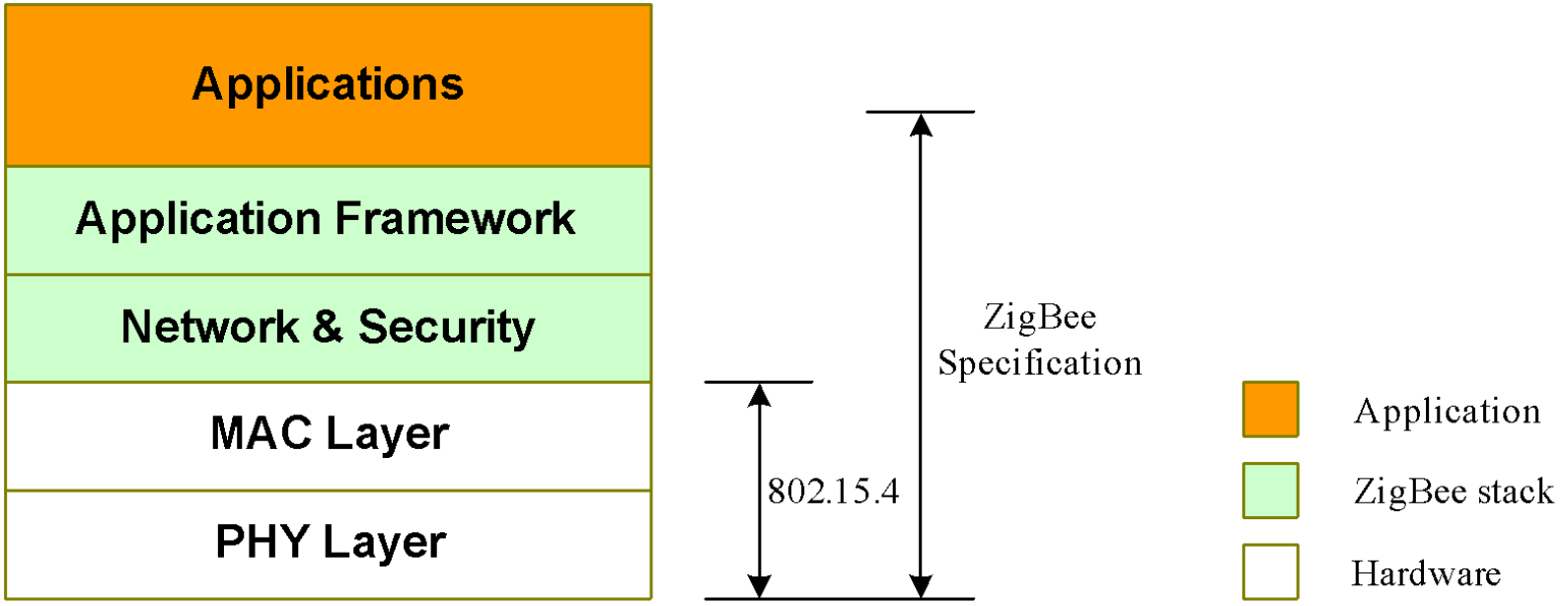
Zigbee Data Traffic

Zigbee targets three different types of data traffic.

- **Periodic data** is delivered or transmitted at a rate defined by the applications (for example, sensors periodically transmitting).
- **Intermittent data** occurs when an application or external stimulus occurs at a random rate. A good example of intermittent data suitable for Zigbee is a light switch.
- R**epetitive low latency data**. Zigbee allocates time slots for transmission and can have very low latency, which is suitable for a computer mouse or keyboard.

Zigbee PHY and MAC

- Zigbee operates primarily in the 2.4 GHz ISM band.
- It also operates at 868 MHz and 915 MHz because of the lower frequency, it has better propensity to penetrate walls and obstacles over traditional 2.4 GHz signals.
- Zigbee does not use all the IEEE802.15.4 PHY and MAC specifications.
- Zigbee does make use of the CSMA/CA collision avoidance scheme.
- It also uses the MAC level mechanism to prevent nodes from talking over each other.



Zigbee addressing

- The Zigbee protocol, as shown, resides on top of the 802.15.4 PHY and MAC layers and reuses its packet structures.
- The network diverges at the network and application layers.

Zigbee uses two unique addresses per node:

Long address (64 bit): Assigned by the manufacturer of the device and is immutable. Uniquely identifies the Zigbee device from all other Zigbee devices. This is the same as the 802.15.4 64-bit address. The top 24 bits refer to the organizational unique identifier (OUI) and the bottom 40 bits are managed by the OEM.
Short address (16 bit): This too is the same as the PAN ID of the 802.15.4 specification and is also optional.

Zigbee Routing Protocol

Zigbee can route packets in multiple manners:

- **Broadcasting**: Transmit packet to all other nodes in the fabric.
- **Mesh routing (table routing)**: If a routing table for the destination exists, the route will follow the table rules accordingly. Very efficient. Zigbee will allow a mesh and table to route up to 30 hops away.
- **Tree routing**: Unicast messaging from one node to another. Tree routing is purely optional and can be disallowed from the entire network. It provides better memory efficiency than mesh routing since a large routing table doesn't exist. Tree routing does not have the same connection redundancy as a mesh, however. Zigbee supports tree routing hops up to 10 nodes away.
- **Source routing**: Used primarily when a data concentrator is present. This is how Z-Wave provides mesh routing.

Zigbee association

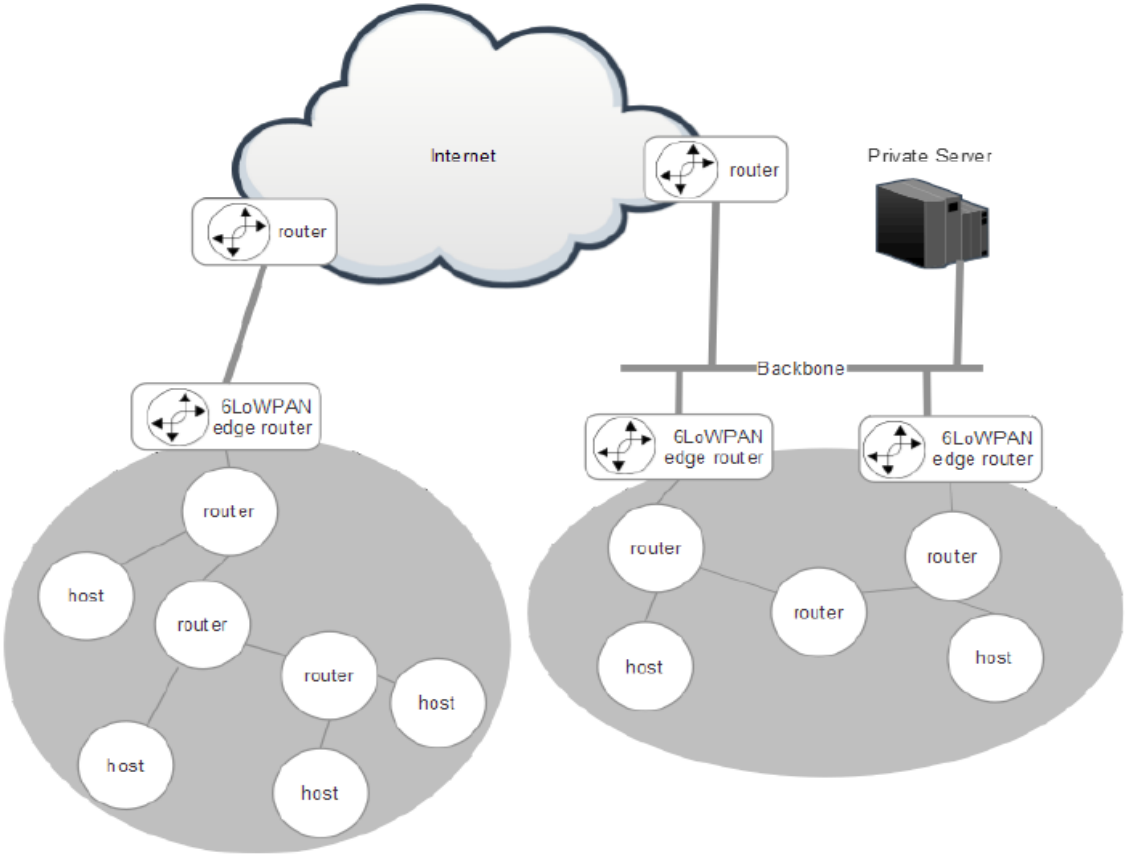
- **Zigbee end devices** (**ZED**) do not participate in routing. End devices communicate with the parent, who is also a router.
- When a **Zigbee coordinator** (**ZC**) allows for a new device to join a network, it enters a process known as an *association*. If a device loses contact with its parent, the device can at any time rejoin through the process known as *orphaning*.
- To formally join a Zigbee network, a *beacon request* is broadcast by a device to ask for subsequent beacons from devices on the mesh that are authorized to allow new nodes to join. At first, only the PAN coordinator is authorized to provide such a request; after the network has grown, other devices may participate.

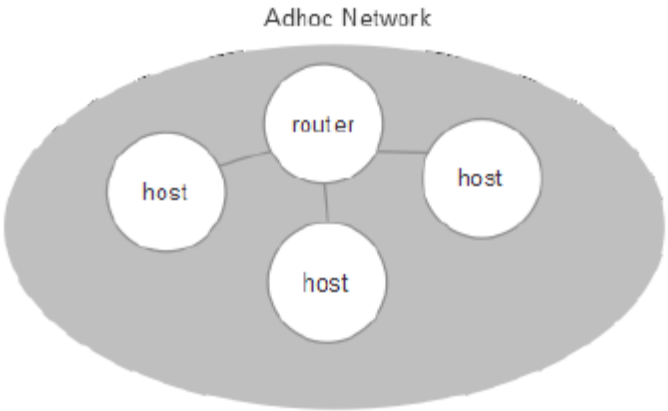
6LowPAN

- To bring IP addressability to the smallest and most resource-constrained devices, the concept of 6LoWPAN was formed in 2005.
- 6LoWPAN is an acronym that stands for IPV6 over low power WPANs. The intent is for IP networking over low-power RF communication systems for devices that are power and space constrained and do not need high bandwidth networking services.
- The protocol can be used with other WPAN communications such as 802.15.4 as well as Bluetooth, sub-1 GHz RF protocols, and **power line controller** (**PLC**).
- The principal advantage of 6LoWPAN is that the simplest of sensors can have IP addressability and act as a network citizen over 3G/4G/LTE/Wi-Fi/Ethernet routers.
- A secondary effect is that IPV6 provides significant theoretical addressability of 2128 or 3.4x1038 unique addresses. Therefore, IPV6 is well-suited for IoT growth.

6LoWPAN topology

- 6LoWPAN networks are mesh networks residing on the periphery of larger networks.
- The topologies are flexible, allowing for ad hoc and disjointed networks without any binding to the internet or other systems, or they can be connected to the backbone or the internet using edge routers.





6LoWPAN- Types of Nodes

There are three types of nodes within the 6LoWPAN mesh:

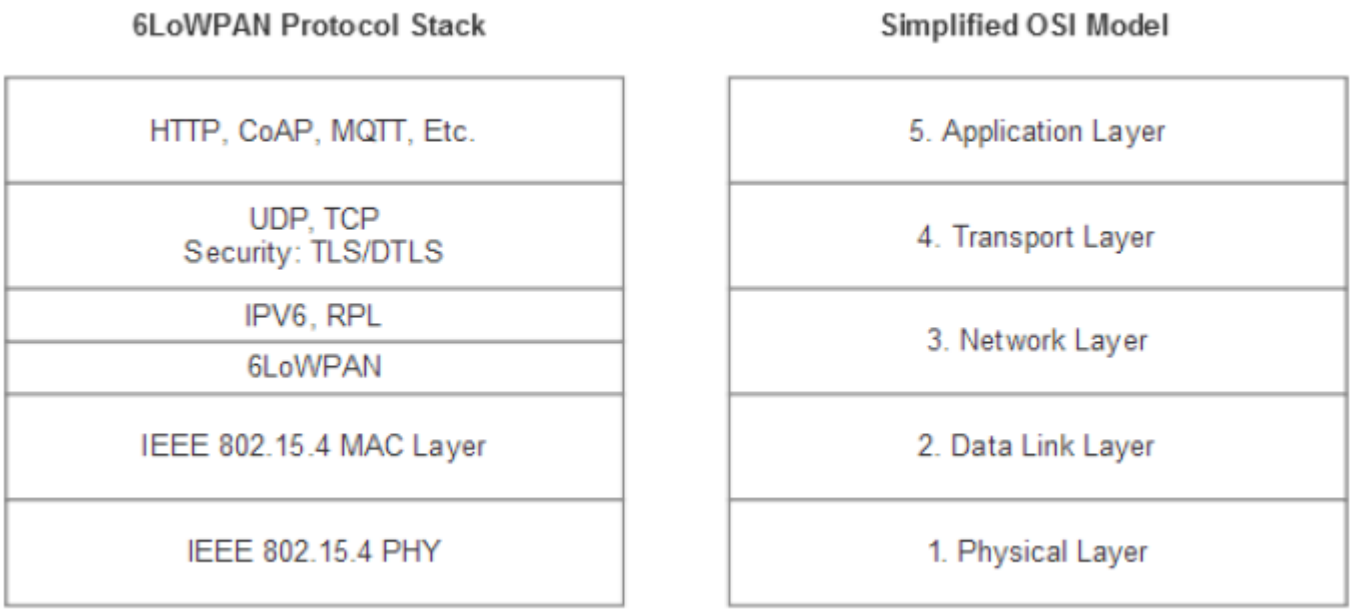
- **Router nodes**: These nodes marshal data from one 6LoWPAN mesh node to another. Routers can also communicate outward to the WAN and internet.
- **Host nodes**: Hosts in the mesh network cannot route data in the mesh and are simply endpoints consuming or producing data. Hosts are allowed to be in sleep states, occasionally waking to produce data or receive data cached by their parent routers.
- **Edge routers**: These are the gateways and mesh controllers usually at a WAN edge. A 6LoWPAN mesh would be administered under the edge router.

Nodes are free to move and reorganize/reassemble in a mesh. For that matter, a node can move and associate with a different edge router in a multi-home scenario or even move between different 6LoWPAN meshes.

In an ad hoc mesh without an edge router, a 6LoWPAN router node could manage a 6LoWPAN mesh. This would be the case when WAN connectivity to the internet is not necessary.

6LoWPAN- Protocol Stack

To enable 6LoWPAN on a form of communication media such as 802.15.4 there is a set of recommended features necessary to support an IP protocol. These features include framing, unicast transmission and addressing.

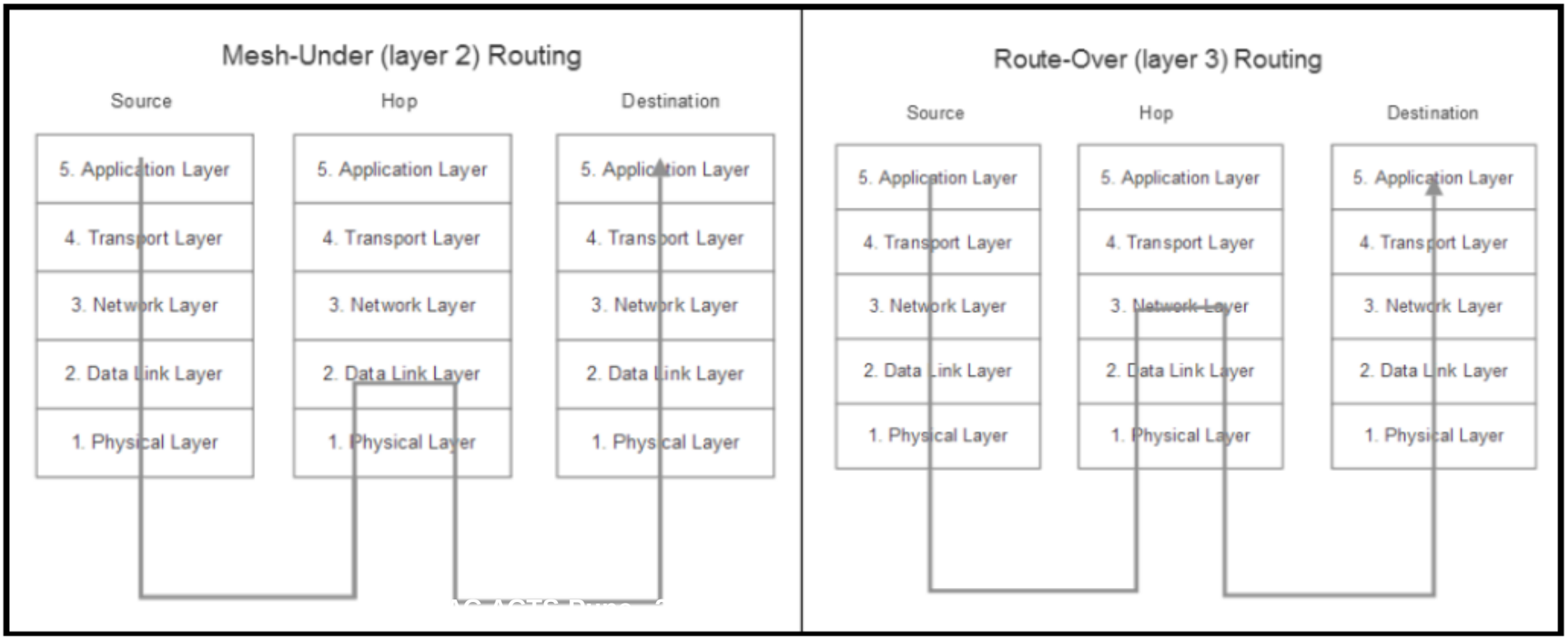


6LoWPAN- Mesh Network routing

6LoWPAN mesh networks utilize two schemes for routing:

- **Mesh-under network**: A message is broadcast in a single domain and is sent to all devices in the mesh. This method generates considerable traffic. Mesh-under routing will move from hop to hop in the mesh but only forward packets up to layer two (data link layer) of the stack. 802.15.4 handles all the routing for each hop in layer two.

- **Route-over network**: In a route-over topology, networks will incur the charge of forwarding packets up to layer three (network layer) of the stack. Route-over schemes manage routes at an IP level. Each hop represents one IP router.



6LoWPAN- Header Compression

IPv6 on an 802.15.4 link poses some challenges that must be overcome to make 6LoWPAN usable.

- **First** is the fact that IPv6 has a **Maximum Transmission Unit**(**MTU**) size of 1280 bytes, while 802.15.4 has a limit of 127 bytes which leads to **fragmentation.**
- **Second issue** is that IPv6 in general adds **significant header overhead** to an already bloated protocol. For example, in IPv6 headers are 40 bytes long.

Header compression is a means to **compress and remove redundancy** in the IPv6 standard header for efficiency reasons.

IPv6 Header

1. Within 6LoWPAN mesh

FF80::00FF:1234:4321:0001 → FF80::00FF:1234:4321:0002

2. Communication from 6LoWPAN device to known address outside mesh

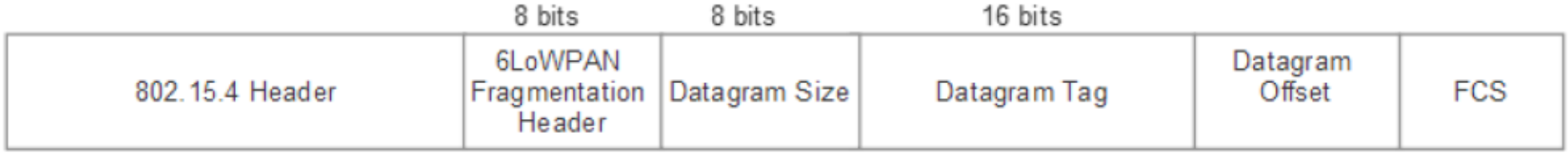
1003::9876:ABCD:0000:0001 → 1003::1234:4321:AAAA:BBBB

6LoWPAN- Fragmentation

Fragmentation is a secondary issue, since the MTU sizes are incompatible between 802.15.4 (127 bytes) and IPv6 at 1280 bytes. The fragmentation system will divide each IPv6 frame into smaller segments. On the receiving side, the fragments will be reassembled.

- **Mesh-under routing fragmentation**: Fragments will be reassembled only at the final destination. All fragments need to be accounted for during re-assembly. If any are missing, the entire packet needs retransmission.
- **Route-over routing fragmentation**: Fragments will be reassembled at every hop in the mesh. Each node along the route carries enough resources and information to rebuild all fragments.

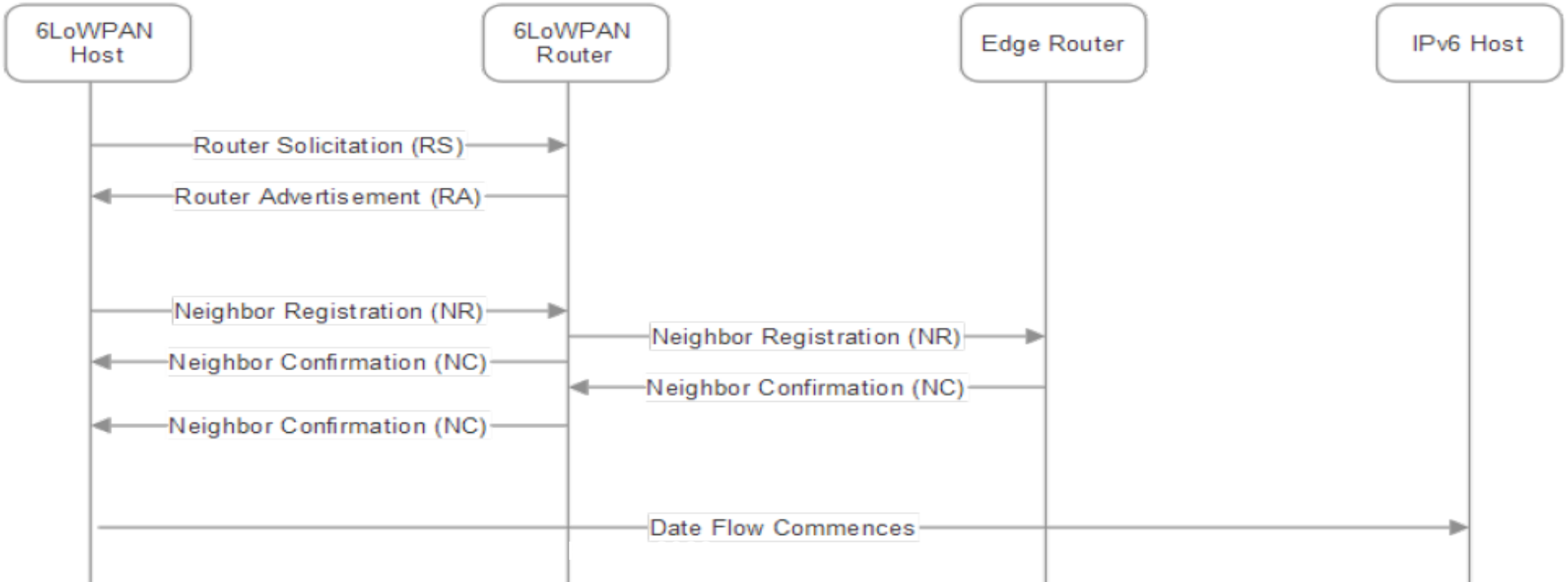
6LoWPAN Fragmentation Header



6LoWPAN- Neighbor Discovery

This is a formal contract between neighbor nodes in the mesh and allows nodes to communicate with each other. ND is the process of discovering new neighbors, as a mesh can grow, shrink and transform, resulting in new and changing neighbor relations. There are two basic processes and four message types in ND:

- **Finding neighbors**: This includes **Neighbor Registration** (**NR**) and **Neighbor Confirmation** (**NC**) phases
- **Finding routers:** This includes **Router Solicitation** (**RS**) and **Router Advertisement** (**RA**) phases



LTE, the 4th generation mobile network system

- LTE supports voice calls, but designed for **scalability and wireless broadband**
- Range normally less than GSM, but **data rate is orders of magnitude greater**

- Typical range: long, 2Km
- Max output power: medium/high, 0.2 W
- Bandwidth: large, up to hundreds of MB/s
- Security: multiple unique identifiers, static keys, and encryption methods
- Cost: depends on the telco operator

- Good for: **broadband wireless internet connection**, also for streaming security camera videos

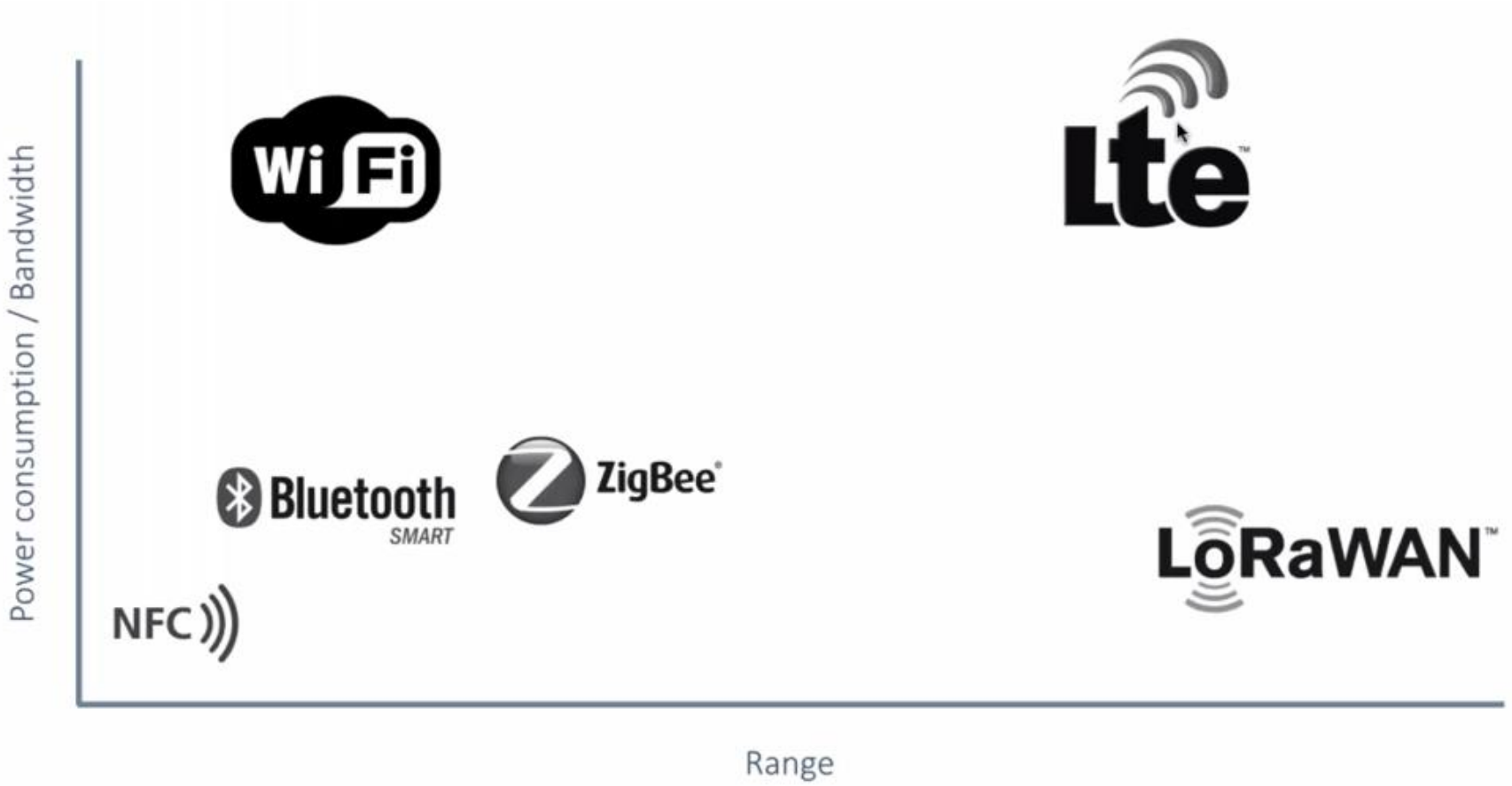
LoRaWAN

- **Low speed, but long range and low power** communication protocol
- Open specification so anyone is free to implement the protocol

- Typical range: long, 5-10 km
- Max output power: low, 0.025 W
- Bandwidth: very low, between 290 bps and 50 kbps
- Security: protocol to exchange to set unique set of AES keys
- Cost: cheap client equipment, needs access point or service provider

- Good for: isolated or private network on a farm or in a city, **ideal for sensors that only seldomly send a value**, like a soil moisture sensor sending its measurements every 10 minutes or a water trough alarming that it is empty

LoRaWAN- Where its placed



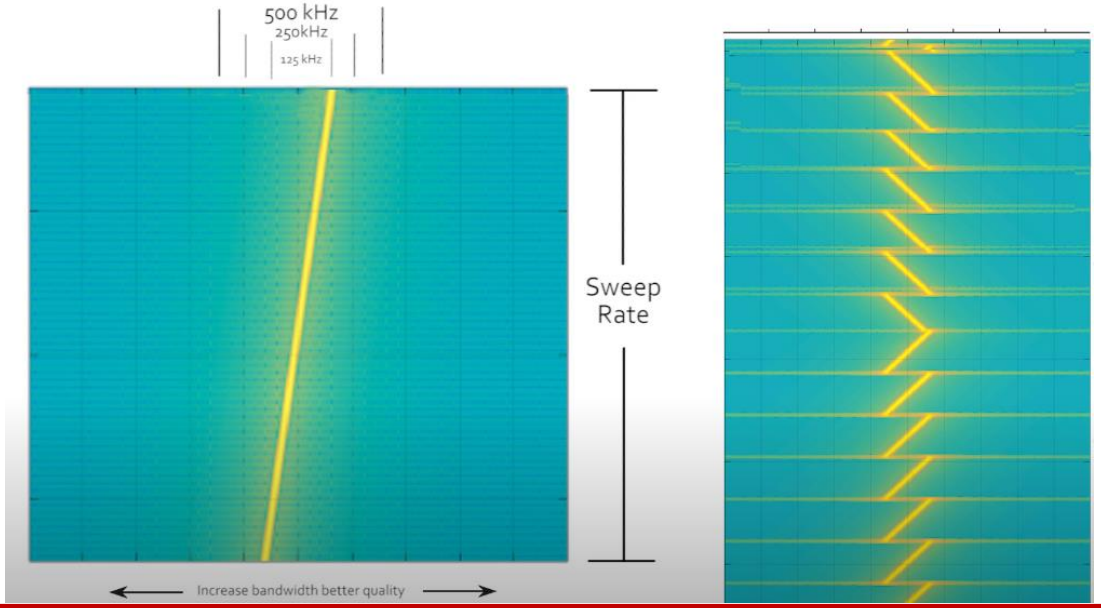
LoRaWAN

- **LoRa** is a **physical layer** for a long-range and low-power IoT protocol while **LoRaWAN** represents the **MAC layer**.
- These proprietary LPWAN technologies and carriers have the advantage of using the **unlicensed spectrum** and that, simply, is reducing data plan cost.
- LoRaWAN will be **5x to 10x lower in their data rates** compared to traditional 3G or LTE connections for large volume deployments (>100,000 units).
- A single LoRaWAN gateway has the **potential to cover a significant amount of area**. Example *Belgium, with a land area of 30,500 km2, is completely covered by seven LoRaWAN gateways.*

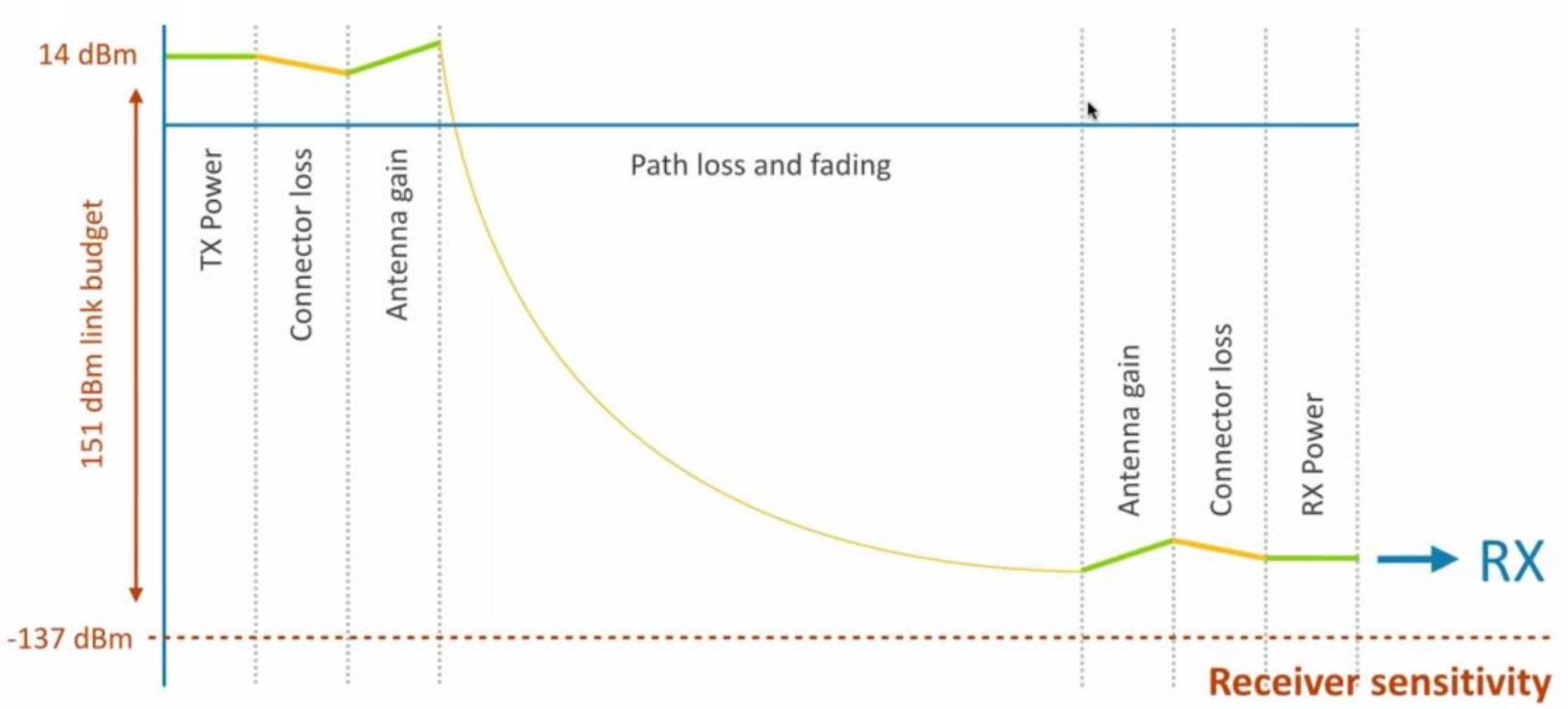
LoRa Physical Layer

- LoRa represents the physical layer of a LoRaWAN network. It manages the **modulation**, **power**, **receiver** and **transmission** radios, and signal conditioning.
- The architecture is based on the following bands in ISM license-free space:
  - **915 MHz:** In the USA with power limits but no duty cycle limit.
  - **868 MHz:** In Europe with a 1% and 10% duty cycle.
  - **433 MHz:** In Asia.

A derivative of **Chirp Spread Spectrum (CSS)** is the modulation technique used in LoRa. CSS balances data rate with sensitivity in a fixed channel bandwidth.



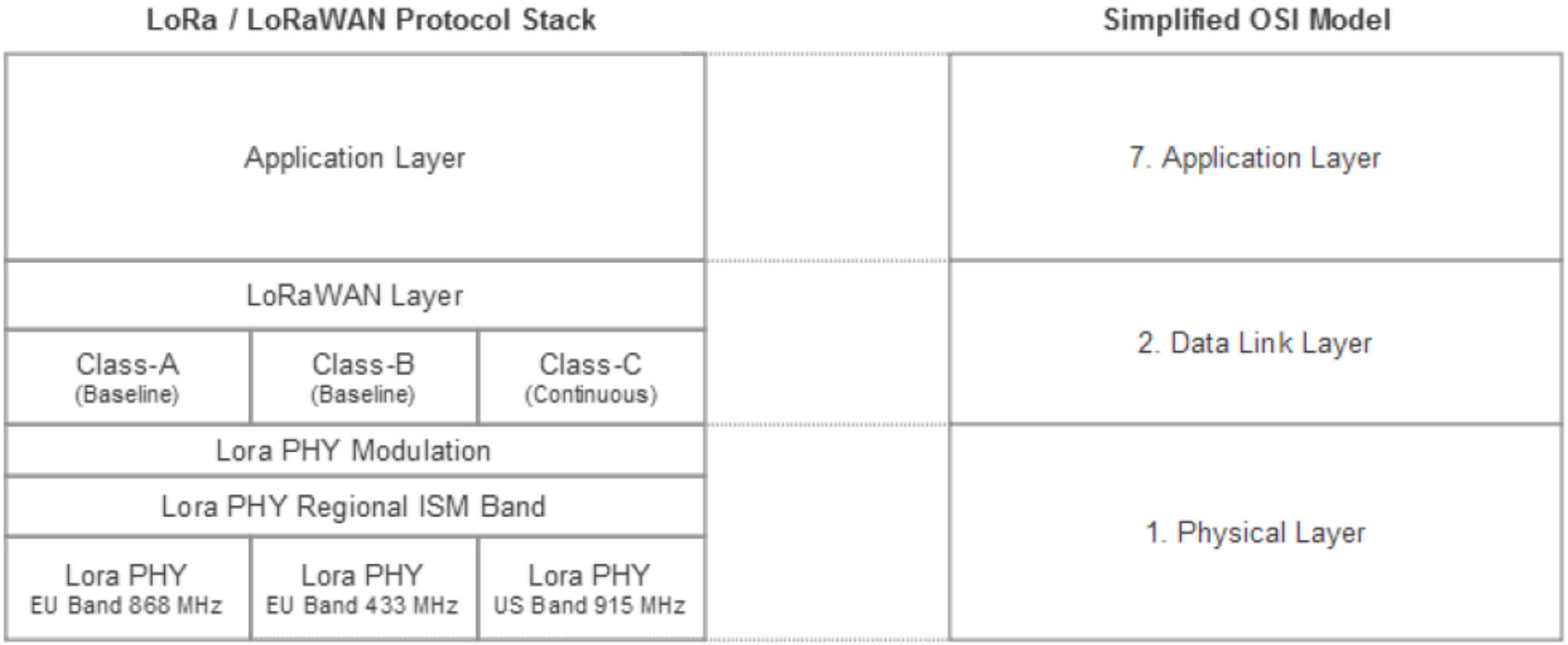
LoRa High Link Budget



LoRaWAN MAC Layer

LoRaWAN represents the MAC that resides on top of a LoRa PHY.
The LoRaWAN **MAC** is an **open protocol** while the **PHY is closed**.
There are three MAC protocols that are part of the data link layer. These three- balance latency with energy usage.

- **Class-A** is the best for energy mitigation while having the highest latency.
- **Class-B** is between Class-A and Class-C.
- **Class-C** has minimum latency but the highest energy usage.



Device Classes

Flexibility in power conservation versus fast network initiated transmission

Class A 😴

Device initiated communication

Devices are typically in deep sleep and send messages on intervals and/or events

Only after uplink transmission, there is a receive window for downlink messages

Best for most sensor applications and battery conservation

Class B ⏰

Time synchronized communication

The network broadcasts beacons for devices to sync time

In so-called ping slots, devices wake up and the network may send downlink messages

Best for most downlink intensive applications

Class C ⚡

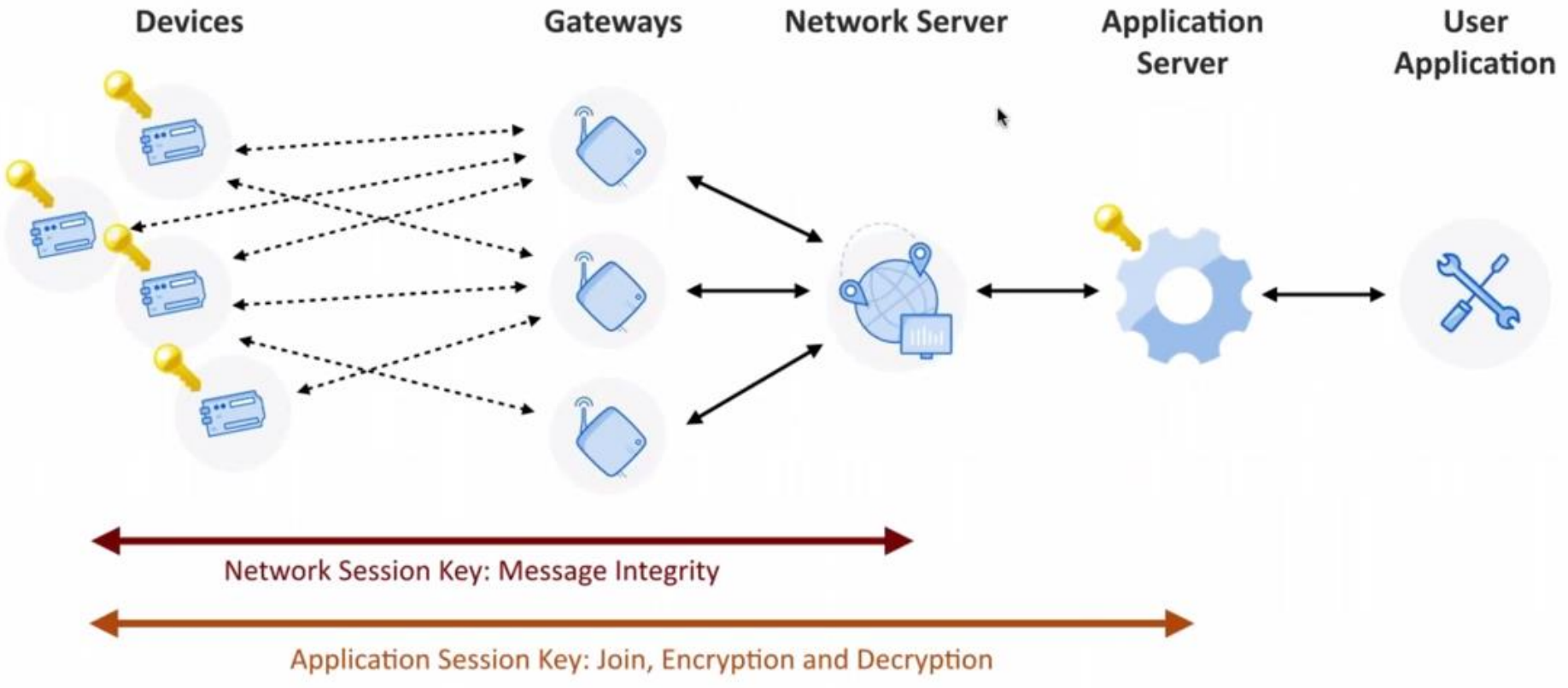
Network initiated communication

The devices are continuously listening, often temporarily or on power supply

The network can send downlink message at any given time

Best for downlink intensive applications that require low latencies

LoRaWAN Topology



SigFox

- **Low speed and low power, but also long range**
- Meant for remote meter reading, also used for any remote data uplink
- Proprietary network and protocol

- Typical range: long, 24 km, global (partial) coverage https://www.sigfox.com/en/coverage
- Max output power: low, 0.025 W
- Bandwidth: very low, up to 140 messages per object per day, 12 bytes payload, 100 bps
- Security: no encryption
- Cost: cheap client equipment, for transmission about 1/12€ per device per year

- Good for: remote electricity or water meter reading, mostly useful if you do not need to send data to the device

NB-IoT Introduction

- **Low Power Wide area network**
- Narrow Band IoT also known as Cat-M2/CAT-NB
- Based on 4G/LTE
- Features Deep Indoor Penetration
- Long Battery Life
- Connections of Thousands of Devices in a Cell
- Cost Efficient Implementation
- Good Network Coverage in difficult environment e.g. basement or remote regions
- Optimal security though end-to-end encryption
- Applications
  - Focused on very low data rates, ideal for simple sensor application
  - Smart Meter
  - Agriculture
  - Home Automation
  - Street lighting

NB-IoT Features

- NB-IoT, also known as Cat-NB
- Cat-NB operates in the licensed spectrum.
- The goal is to reduce power (10-year battery life), extend coverage (+20 dB), and reduce cost ($5 per module).
- Cat-NB is based on the Evolved Packet System (EPS) and optimizations to the Cellular Internet of Things (CIoT).
- Channels are much narrower than even the 1.4 MHz Cat-M1, cost and power can be reduced even further.

Significant differences between Cat-NB and Cat-M1 include:

1. **Very narrow channel bandwidth**: As with Cat-M1, which reduced the channel width to 1.4 MHz, Cat-NB reduces it further to 180 kHz for the same reasons (reducing cost and power).
2. **No VoLTE**: As the channel width is so low, it does not have the capacity for voice or video traffic.
3. **No mobility**: Cat-NB will not support handover and must remain associated with a single cell or remain stationary. This is fine for the majority of secured and fastened IoT sensor instruments.

NB-IoT Deployment Options

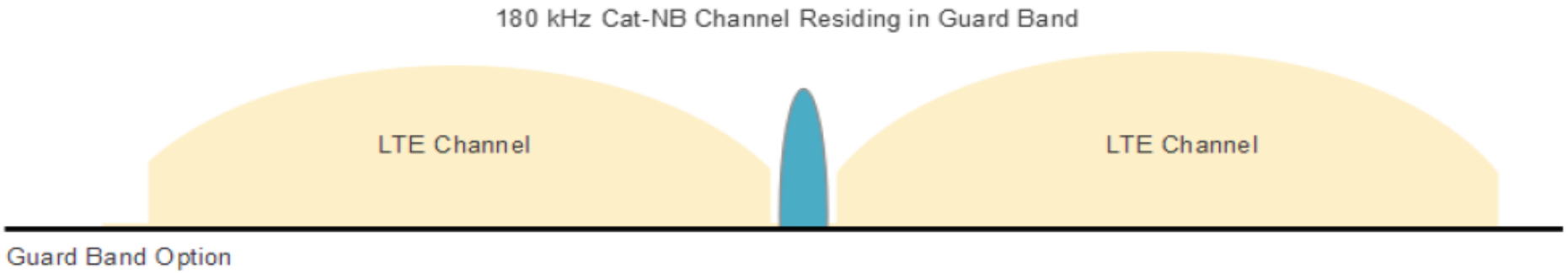
- Since the channel width is so small (180 kHz), it allows for the opportunity to bury the Cat-NB signal
  1. inside a larger LTE channel (**In-band**)
  2. replace a **GSM channel**,
  3. even exist in **the guard channel** of regular LTE signals.

- ***In-band*** provides a massive amount of spectrum to use as the LTE bands are much larger than the 180 kHz band. This allows for deployments of up to 200,000 devices in theory per cell. In this configuration, the base cell station will multiplex LTE data with Cat-NB traffic.

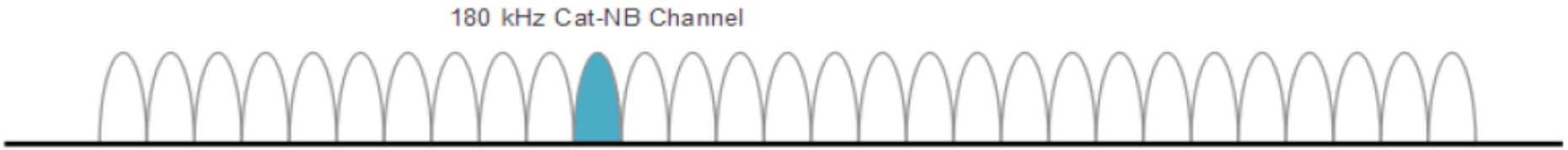
- Using Cat-NB as the LTE ***guard band*** is a unique and novel concept. Since the architecture re-uses the same 15 kHz subcarriers and design, it can be accomplished with existing infrastructure.

- The GSM option is simplest and fastest to market. Some portions of the ***existing GSM traffic*** can be replaced with NB-IoT

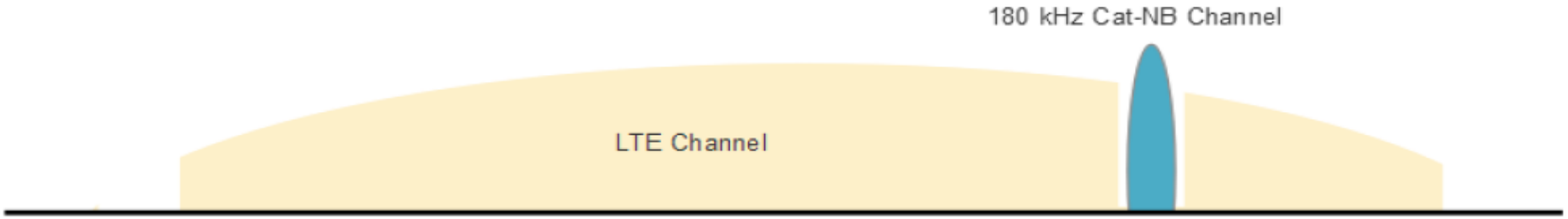
NB-IoT Deployment Options



Guard Band Option



GSM Option



In-Band Option

Other Wireless Protocols

- LTE Cat-M
  - use cellular channels for long range, limited bandwidth communication
  - still in development or with limited coverage
- Satellite
  - huge coverage
  - extremely expensive

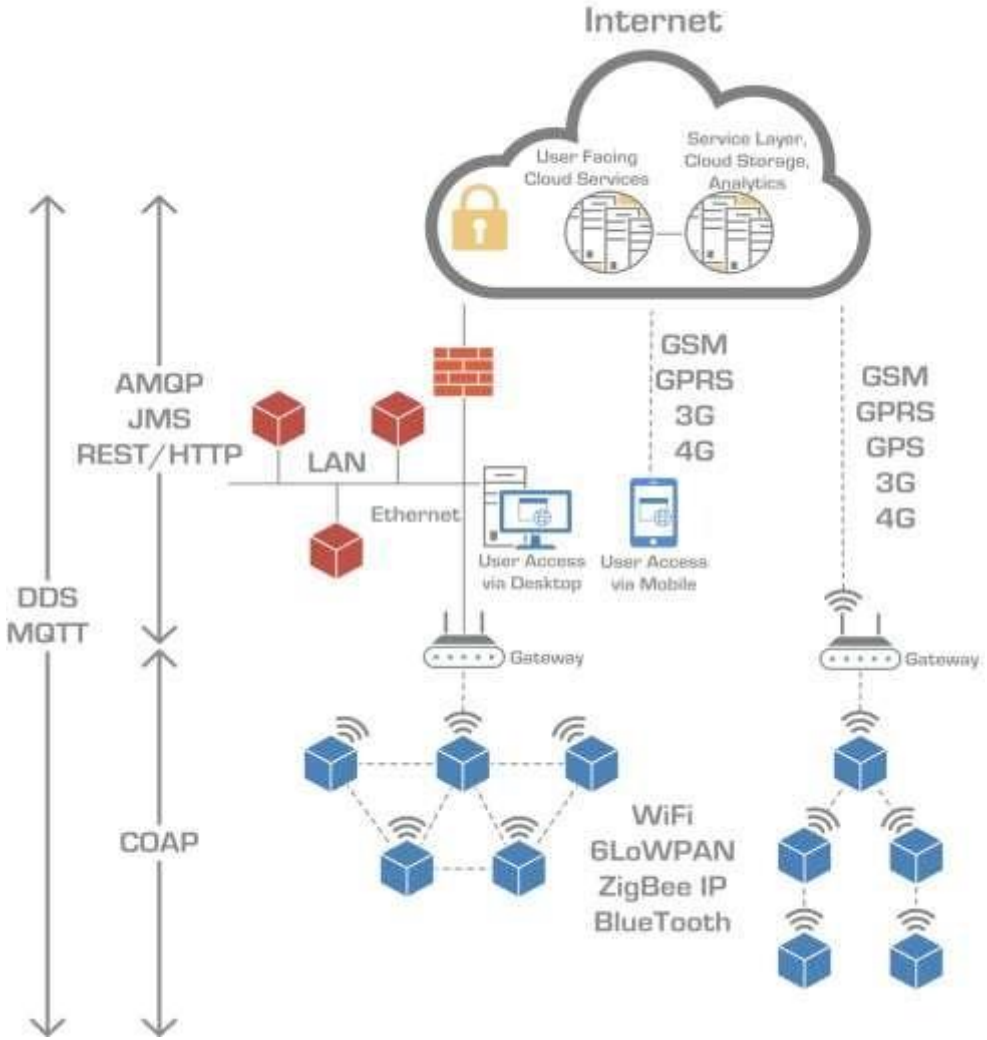
Outline

1. Introduction to IoT
   - definition, enabling technologies and concepts to better understand the general framework of IoT solutions
   - layering architecture, cloud computing vs. fog computing

2. Most relevant components of IoT solutions
   - devices, wireless communication protocols, **data exchange protocols**

IoT Connectivity Problem Space

- Wired/wireless protocols to transport bits
- **Messaging protocols to transfer data**
- **Data**: information and commands described following a given syntax and semantic
- **Messaging protocols to exchange data**: encapsulate data and transmit it via wired/wireless protocols



Data Exchange Protocols

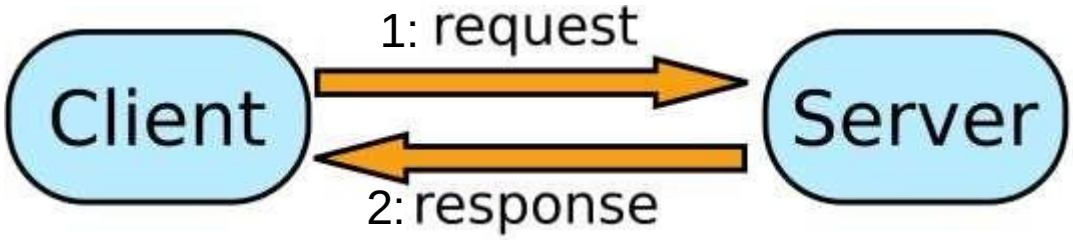
- How to encapsulate information and commands in messages?
- Data from gateways to servers: who should initiate the communication?
  - **push**: gateways autonomously decide to send messages to servers, e.g., when new sensed values are available
  - **pull**: servers ask to gateways to send messages, e.g., only when servers actually require sensed values
- Two primary models
  - **request/response**, can be push or pull
  - **publish/subscribe**, usually only push
- Several protocols:
  - request/response: REST/HTTP, CoAP, …
  - publish/subscribe: MQTT, AMQP, DDS, …

Multiple Protocols supporting Different Features

- Several protocols thriving to become a de facto standard
- It could be difficult to pick up the **best solution**, since it **depends on requirements** of a given application domain

- Some request/response protocols:
  - REST architectural principles (REpresentational State Transfer), 2000
  - CoAP (Constrained Application Protocol), 2014 (also pub/sub)
- Some publish/subscribe protocols:
  - MQTT (Message Queue Telemetry Transport), 1999
  - AMQP (Advanced Message Queuing Protocol), 2003
  - DDS (Data Distribution Service), v1.0 2003; v1.2 2007
- At different level of abstractions, LonWorks (1999), Modbus (1979)…

Request/Response model



- A **client application** requests services (e.g., send data, require data, perform an addition) and a server application responds to the service request (e.g., by providing the data or the addition results)
- **Direct communication** between client and server
- Data exchange only if **the client starts the communication**
  - the server is not able to contact the client autonomously

- Typical interactions in the IoT scenario:
  - **push**: sensors send data to servers
  - **pull**: actuators request to servers new configurations

REST - REpresentational State Transfer

- REST is not an actual protocol, but substantially a **solution architectural style**
- Promote **client/server** and **stateless** interaction, oriented to the usage of **caching** opportunities, also with possibility of code-on-demand to clients
- Usually based on HTTP, the protocol used to surf the Web

- Very simple, but each time the client has to start the communication from the beginning

REST: Identifying and Interacting with Resources

- **URI (Uniform Resource Identifier) to identify** the remote **resource**
  - any resource has a **persistent identifier**
  - do not transfer resources but their representations via HTTP
  - **endpoint** for a service managing user information: www.examples.com/resources/users

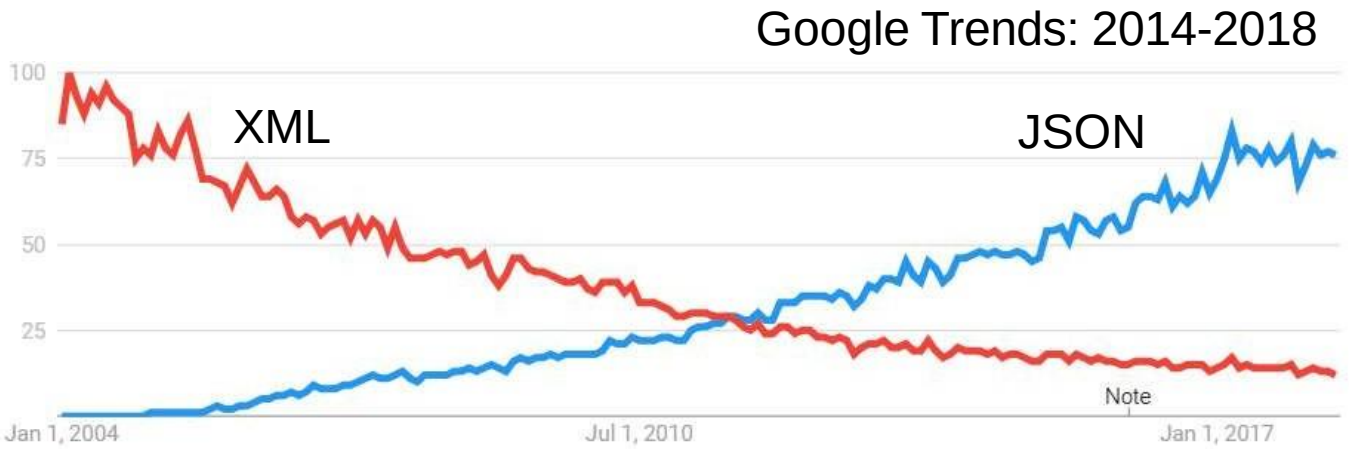
- **HTTP method to interact** with remote resources in a standard manner
  - **GET, retrieve** a specific resource (by id) or a collection of resources
  - **PUT, create** a new resource
  - **POST, update** a specific resource (by id)
  - **DELETE, remove** a specific resource by id

REST: Endpoint Examples

- Remote resource: a software component managing users

HTTP Method	URI	Operation
GET	www.examples.com/resources/**users**	Get list of users
GET	www.examples.com/resources/**users/1**	Get the user with id 1
DELETE	www.examples.com/resources/**users/1**	Delete the user with id 1
POST	www.examples.com/resources/**users/2**	Update the user with id 2
PUT	www.examples.com/resources/**users/1**	Insert the user with id 1

REST: Formatting data



- JavaScript Object Notation (JSON) most spread syntax for **formatting/serializing data**, e.g., objects, arrays, numbers, strings, booleans, and null
- Resource: a group of users, each one with a first name and a last name

```
{“users":
   [
      { "firstName":"John", "lastName":"Doe" },
      { "firstName":"Anna", "lastName":"Smith" },
      { "firstName":"Peter", "lastName":"Jones" }
   ]
}
```

REST: AWS IoT example

- Adds a thing to a thing group

PUT **/thing-groups/addThingToThingGroup**
HTTP/1.1
Content-type: application/json

```
{
   "thingArn": "string",
   "thingGroupArn": "string",
   "thingGroupName": "string",
   "thingName": "string"
}
```

- Manage “shadows” of things

POST **endpoint/things/thingName/shadow**
HTTP/1.1
Content-type: application/json

```
{
    "state":
    {
      "desired" :
       {
           "color" : { "r" : 10 },
           "engine" : "ON"
       }
    }
}
```

CoAP - Constrained Application Protocol

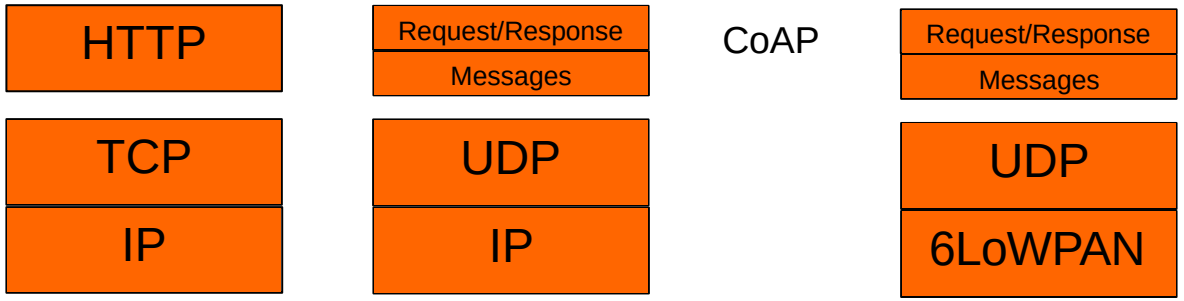
- Designed for M2M/IoT applications such as smart-metering, e-health, building and home automation with:
  - **constrained nodes**, e.g., 8-bit microcontrollers with 16KB RAM and battery-operated
  - **constrained networks**, e.g., Wireless Sensor Networks
- Low-overhead **request/response** protocol that also supports discovery of services/resources
- Based on **UDP** communications (with multicast)
- Inspired by (and compatible with) the **HTTP protocol** and REST architectures: CoAP is a specialized Web transfer protocol

CoAP Features

- **Web RESTful protocol** fulfilling M2M requirements in constrained environments
- **Simple request/response HTTP mapping**, to access CoAP resources via HTTP
- **URI** and Content-type support (a sensor is identified by an URI)
- **Low header overhead** and parsing complexity
- **Security** binding to Datagram Transport Layer Security (DTLS)
- UDP binding with **optional reliability**, supporting unicast and multicast
- **Asynchronous** message exchanges
- Services and resources **discovery**
- Also **publish/subscribe** (and push notifications)
- Simple **caching** (max-age parameter)

- CoAP implementations in many programming language, e.g., C, C++, and Java

CoAP Messaging



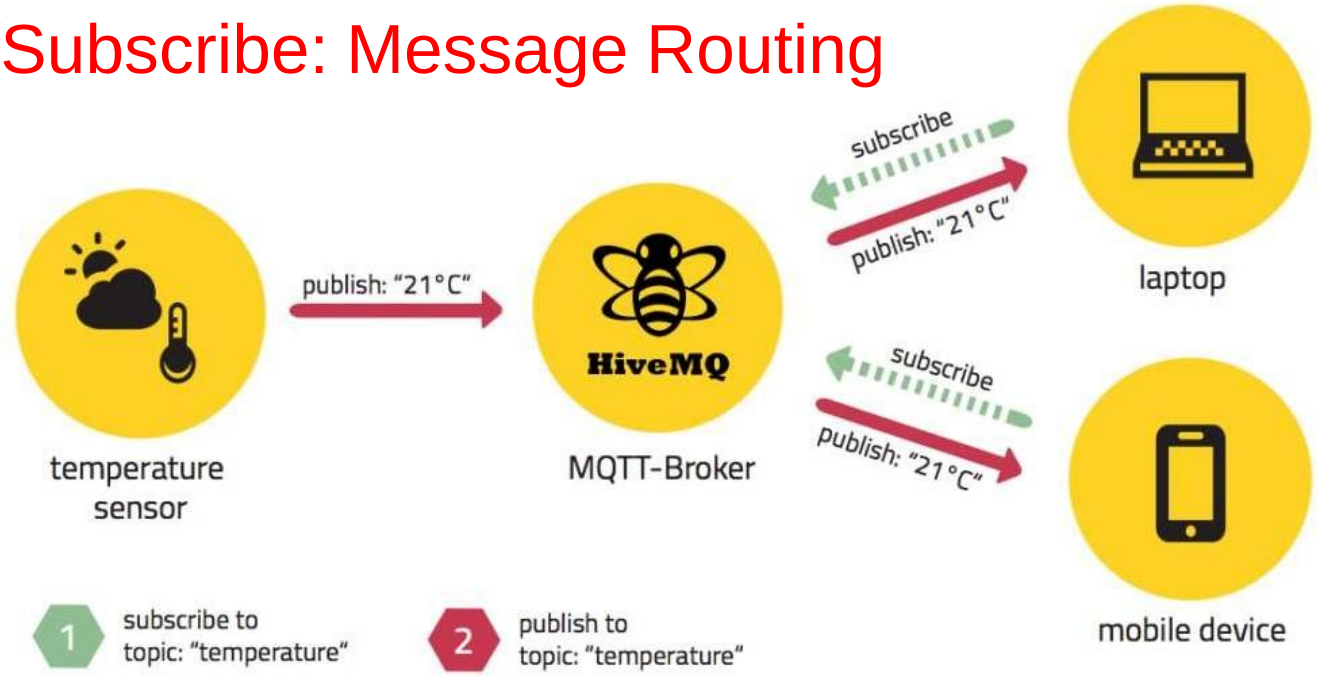
- Duplicate detection and optional reliability
- Possible messages:
  - confirmable message (CON), wait for ACK
  - acknowledgement message (ACK), in response to CON
  - non-confirmable message (NON), no need to wait for ACK
- CoAP on top of UDP allows for multicast requests

Publish/Subscribe model

- Publish/subscribe (pub/sub) pattern alternative to traditional client-server model, where a client communicates directly with an endpoint

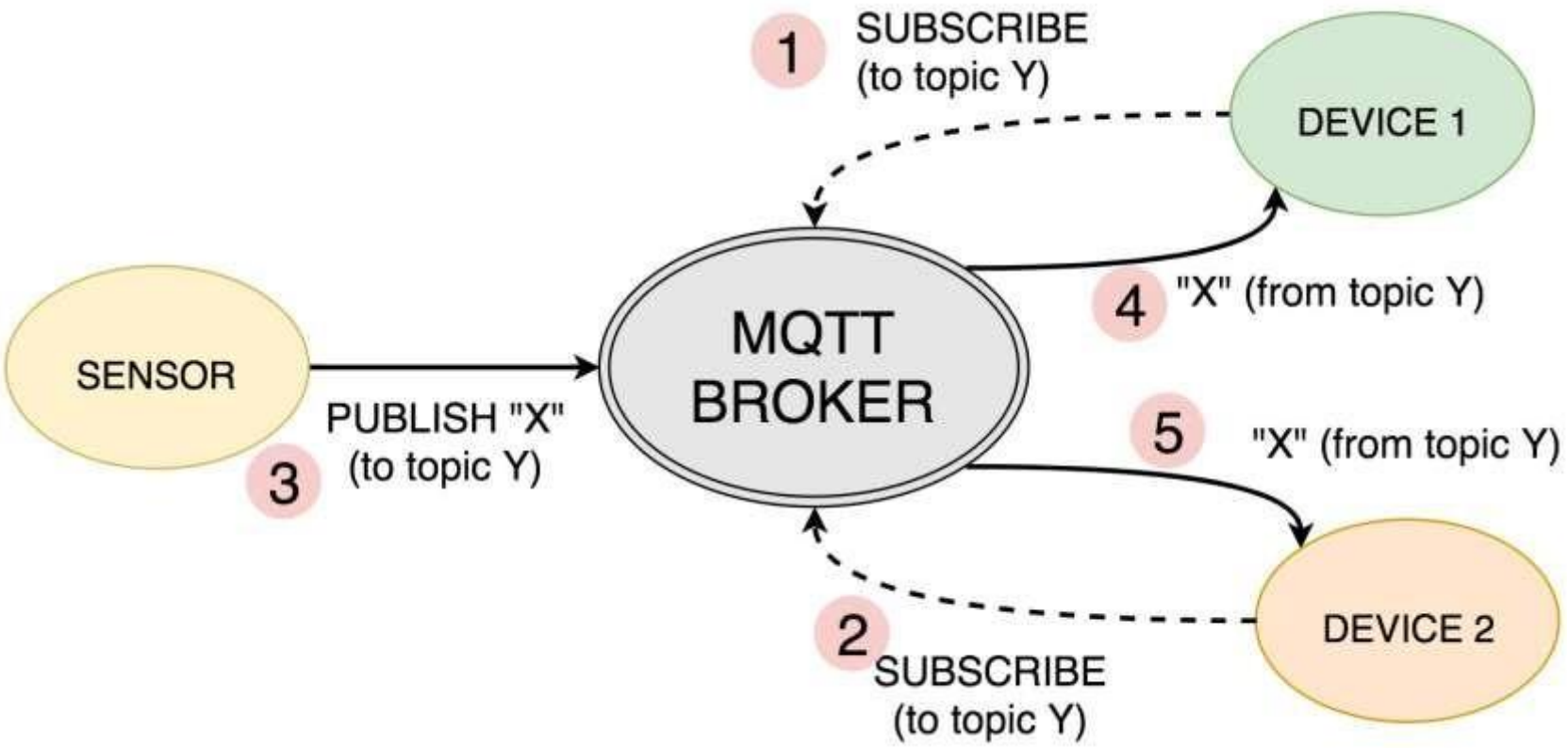
- Roles:
  - **publisher**: a client sending a message
  - **subscriber**: one or more receivers waiting for the message
  - **broker**: a central component receiving and distributing messages to interested receivers

Publish/Subscribe: Message Routing



- The broker **routes messages** (i.e., selects receivers of a message) based on:
  - **Message Topic**: a subject, part of each message. Receiving clients subscribe on the topics they are interested in with the broker and from there on they get all message based on the subscribed topics
  - **Message Type**: depending on the type of the message
  - **Message Header**: depending on a set of fields of the message
  - **Message Content**: possibly depending on the whole message content (expressive but expensive)

Publish/Subscribe: Typical Sequence



Publish/Subscribe: Decoupling

- **Space decoupling**: publisher and subscriber do not need to know each other (by ip address and port for example)
- **Time decoupling**: publisher and subscriber do not need to run at the same time
- **Synchronization decoupling**: operations on both components are not halted during publish or receiving

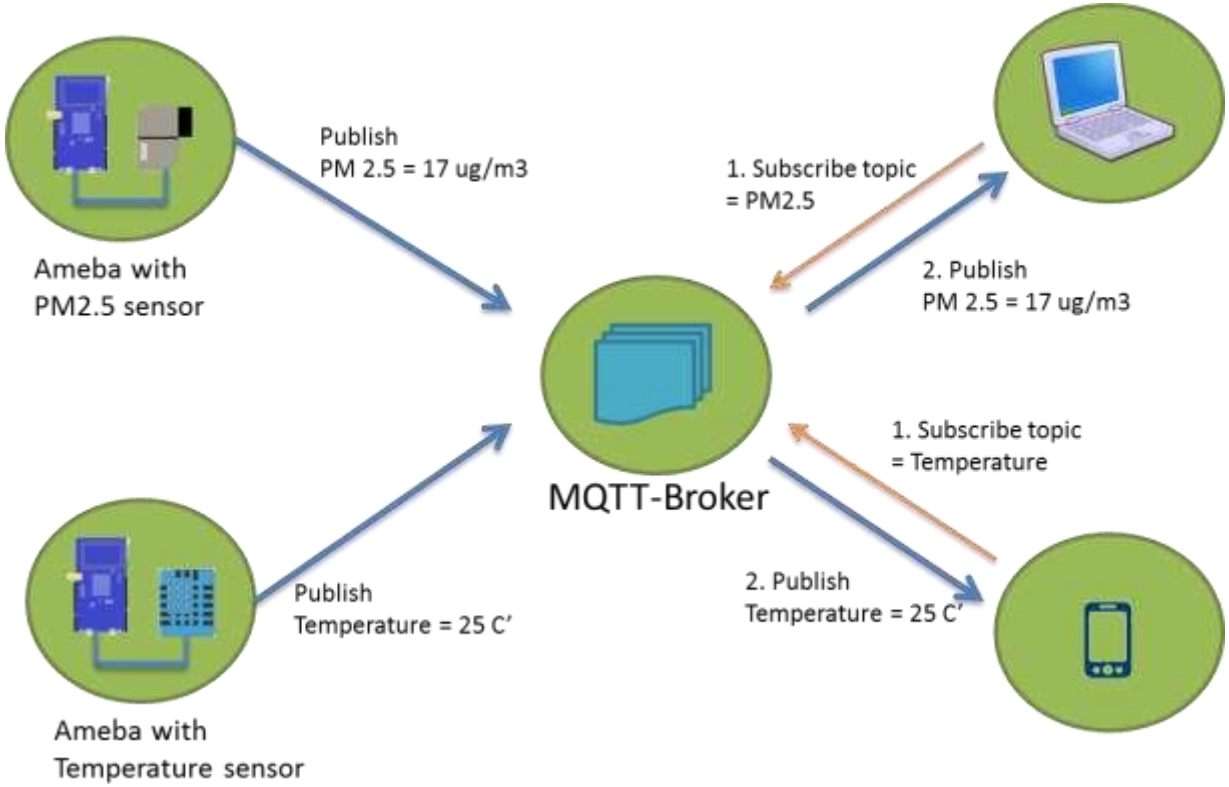
- Event system as **logically centralized** system
  - anonymous communication
  - possibility to use filters (on headers or entire messages)
  - basic primitives: subscribe, unsubscribe, publish

MQTT - Message Queue Telemetry Transport

- IBM developed WebSphere MQ as a message-based backbone mainly for Enterprise Application Integration (EAI), MQTT is its evolution:
  - an open standard but it is strongly supported by IBM
  - designed to permit WebSphere MQ to talk with constrained (smart) devices at the edge of the network

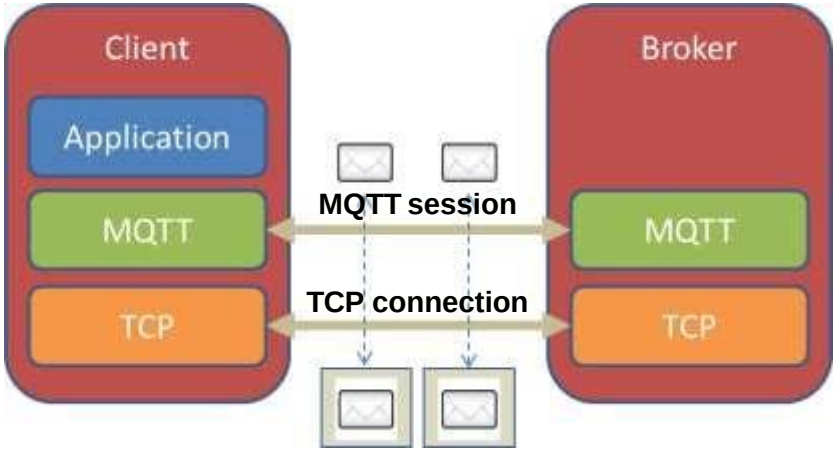
- **MQTT: simple, lightweight, broker-based, publish/subscribe, open messaging protocol**
- Ideal for use in **constrained nodes and networks**
  - on embedded devices with limited processor or memory resources
  - where the network is expensive, has low bandwidth or is unreliable
- MQTT-SN for wireless sensor networks, aimed at embedded devices on non-TCP/IP networks (such as Zigbee).

MQTT: Typical Use Case



1. Subscribe to one ore more topic
2. Publish to a topic
3. Receive messages related to subscribed topics

MQTT: Features (1)



- **Publish/subscribe** message pattern to provide
  - one-to-many message distribution
  - decoupling of applications
- Use of **TCP/IP** to provide basic network connectivity
- **Small transport overhead** (minimal header length just 2 bytes) and protocol exchanges minimised to reduce network traffic
- **Easy to use** with few commands: connect, subscribe, publish, disconnect
- **Retained messages**: an MQTT broker can retain a message that can be sent to newly subscribing clients

MQTT: Features (2)

- **Three message delivery semantics** with increasing reliability and cost:
  - Quality of Service (QoS): at least once, at most once, exactly once
- **Durable connection**: client subscriptions remain in effect even in case of disconnection
  - subsequent messages with high QoS are stored for delivery after connection reestablishment
- **Wills**: a client can setup a will, a message to be published in case of unexpected disconnection, e.g., an alarm

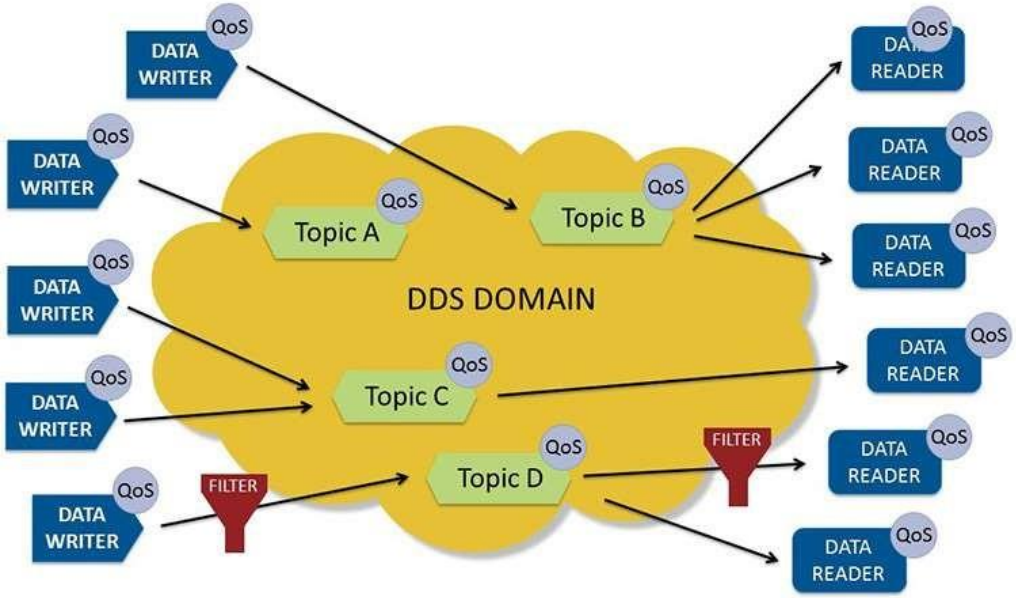
- MQTT is a protocol adopted in several platforms: there are MQTT brokers in WebSphere, Mosquitto, RabbitMQ, etc. and clients in several languages

AMQP – Advanced Message Queuing Protocol

- **Richer semantic** than MQTT, e.g., supports topics and queues, but also **heavier**, e.g., the broker is much more complex
- AMQP implementations, e.g., Apache Qpid, focus on providing **several features**: queuing, message distribution, security, management, clustering, federation, heterogeneous multi-platform support
  - most of them (possibly) not essential in the IoT scenario

- Originally developed at JPMorgan Chase in London, AMQP was designed as a message-oriented protocol for the integration of enterprise IT components (Enterprise Message Bus)

DDS - Data Distribution Service



- Publish/subscribe, but **broker-less** (based on multicast)
  - **scalable**, **real-time**, **dependable**, **high performance** and **interoperable**
- Proposed in several **mission-critical environments** where performance and reliability are essential, e.g., industrial automation, financial applications, air traffic control and management, and even military environments
  - http://twinoakscomputing.com/datasheets/DDS-Brochure.pdf
- Standard developed by the Object Management Group

RabbitMQ

- Not actually an IoT platform, but an interesting **messaging infrastructure**

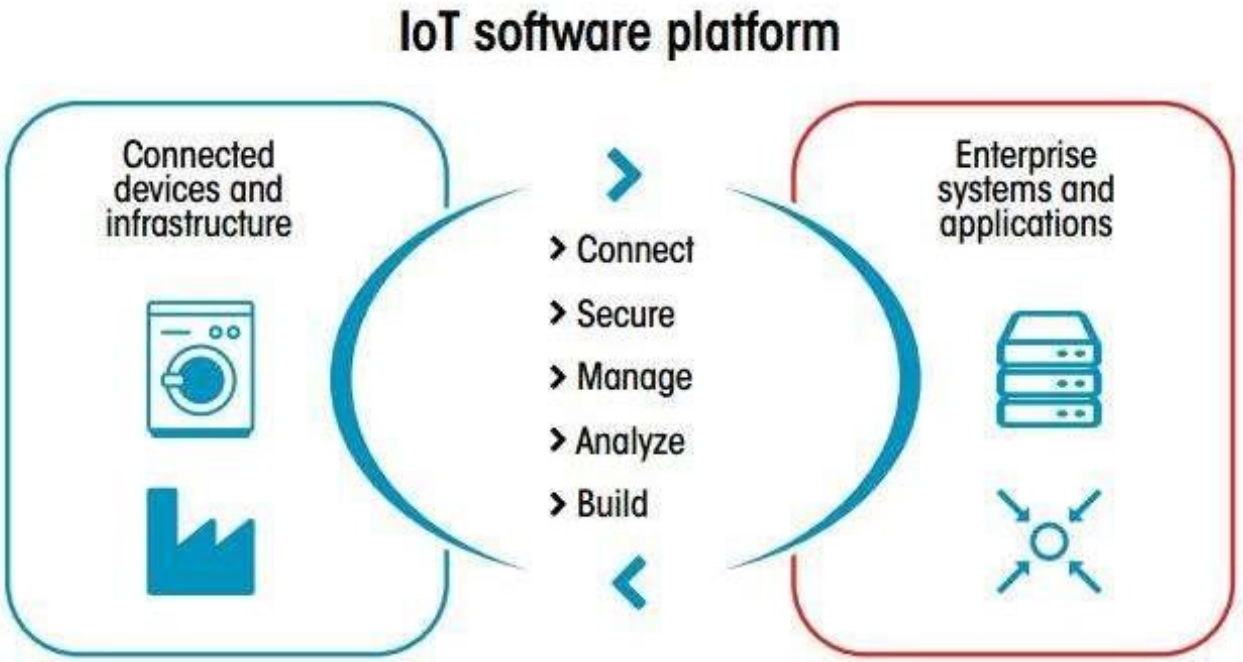
- **Robust** messaging for applications
- **Reliable** message delivery
- Distributed deployment as **clusters** for high availability and throughput
- **Federate** across multiple availability zones and regions.
- **Multi-Protocol**: AMQP, MQTT, HTTP, etc.
- **Managing and monitoring** via HTTP-API, command line tool, and UI

- Runs on all major operating systems
- Supports a huge number of developer platforms
- Open source and commercially supported

RabbitMQ: Durable, Persistent, and Synchronized

- Exchanges (Topics) and Queues can be **durable**
  - capable of surviving to a broker restart (as opposed to transient)
- Messages can be defined as **persistent**
  - only persistent messages survive exchange/queue/brokers/channel failures
- **Synchronization**
  - **automatic** acknowledgement model, i.e., verify that the message is actually delivered to the application waiting for it
  - **explicit** acknowledgement model, i.e., wait for explicit acknowledgment sent back by the application
    - immediately at message reception, after processing, …

IoT Platforms



- IoT Platform between things/devices and business applications
  - **connect** devices to gather information
  - **secure** communication with devices and applications
  - **manage** devices to control their behavhior
  - **analyze** data, e.g., with AI
  - **build** applications, also interacting with CRM/ERP/…

Several IoT Platforms

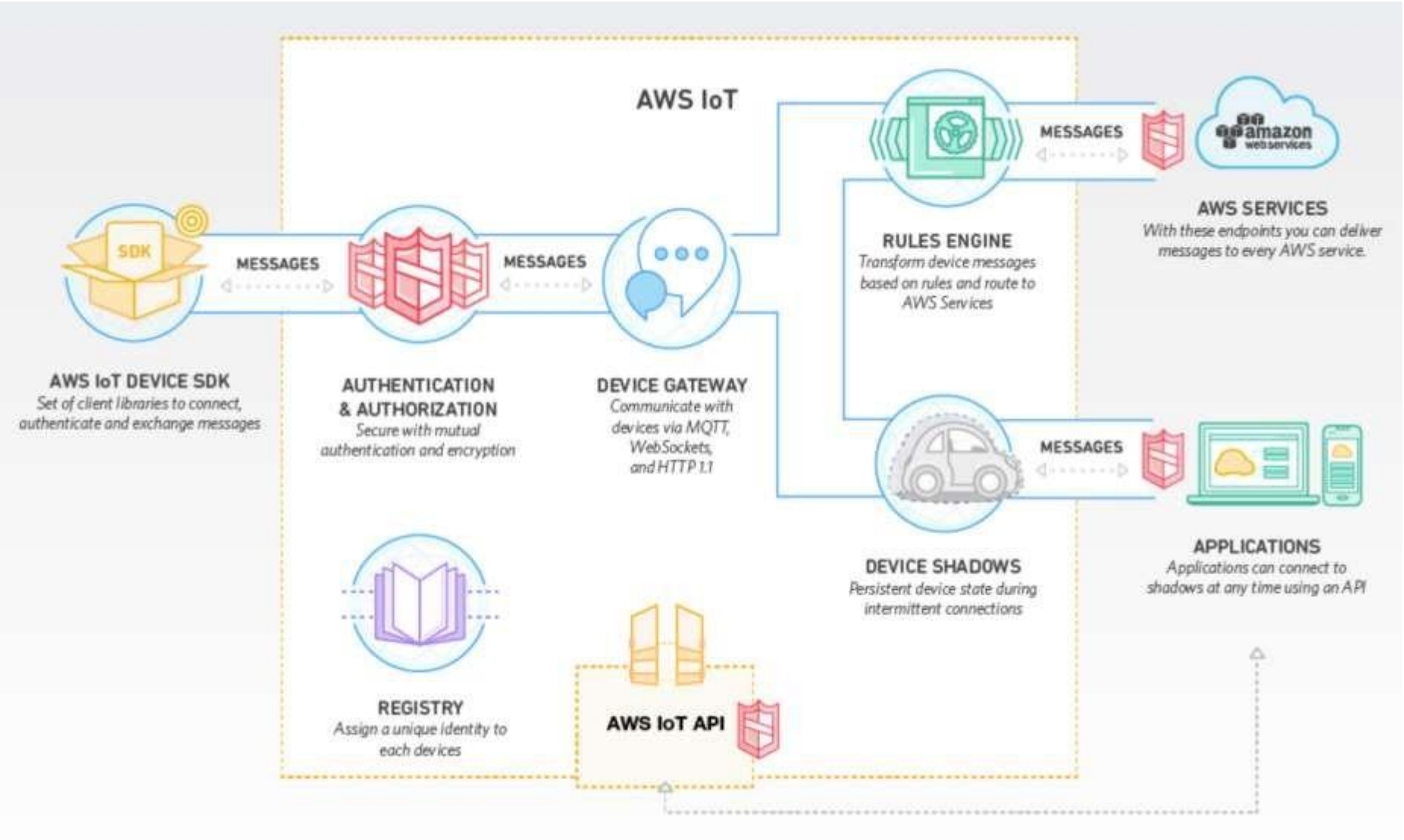
We go a bit deeper into a selection of the many available ones:

- Amazon Web Services (AWS) IoT
- Microsoft Azure IoT Hub
- Mindsphere by Siemens
- EdgeXFoundry

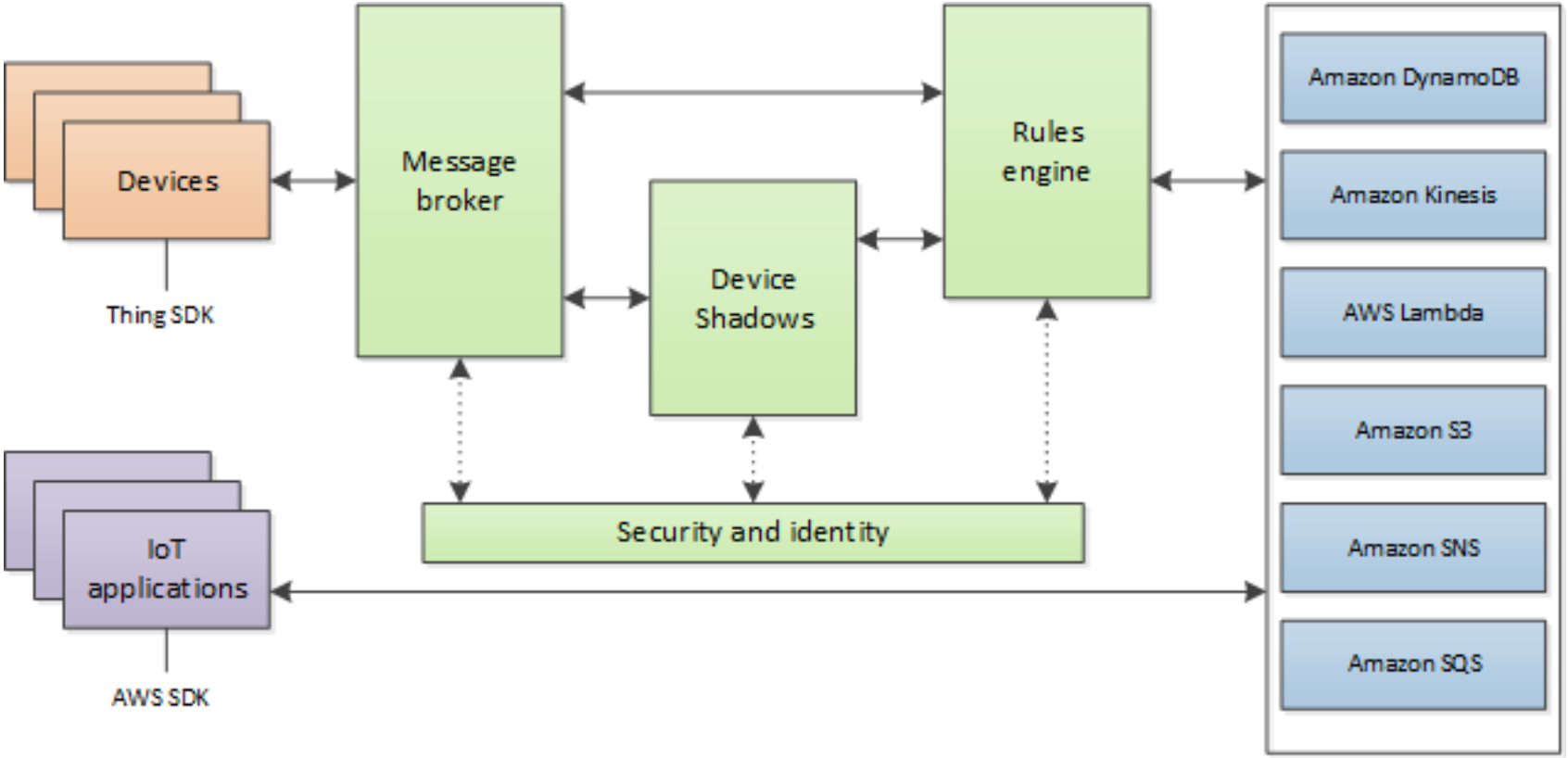
And several others:

- Google Cloud Platform
- ThingWorx IoT Platform
- IBM Watson
- Carriots
- Kaa
- …

Amazon Web Services (AWS) IoT



AWS IoT: Architecture



- https://docs.aws.amazon.com/iot/latest/developerguide/aws-iot-how-it-works.html

AWS IoT: Main Components (1)

- **AWS IoT Device SDK** connects devices to AWS IoT by using the MQTT, HTTP, or WebSockets protocols
  - the Device SDK supports C, Java, JavaScript, Arduino, iOS, Android,…
- **Device Gateway** supports (secure) communication with devices, by using a (1:1 or 1:n) **publish/subscribe** model
  - supports MQTT, WebSockets, and HTTP 1.1 protocols
- **Authentication and Authorization** with mutual authentication and encryption
  - authentication with native AWS (called ‘SigV4’) as well as X.509 certificate based authentication
  - AWS facilitates the whole certificate process management
- **Registry** assigns a **unique identity** to each device
  - also supports **metadata** describing capabilities of devices, e.g., whether a sensor reports temperature and if data in Fahrenheit or Celsius

AWS IoT: Main Components (2)

- **Device Shadows**, i.e., persistent, virtual version of devices
  - applications or other devices exchange messages and **interact through the Shadow Device**
  - Device Shadows **persist** the last reported state and desired future state of each device even when the device is offline
  - exploit it to **retrieve the last reported state** of a device or **set a desired future state** through the API or using the rules engine

- **Rules Engine**, supporting to build IoT applications that gather, process, analyse, and act on data generated by connected devices
  - evaluates inbound messages published into AWS IoT and **transforms and delivers** them to another device or a cloud service, based on business rules you define
  - a rule can apply to data from one or many devices, and it can take one or many actions in parallel