Topic 3: (Part F)

I/O and Peripheral Devices Management

Wireless communication in the Nintendo DS

Systèmes Embarqués Microprogrammés

Prof. David Atienza Alonso, SEL-STI

Content of Session

- Basics of networking communication
  - General network topology structure
  - Internet protocol concepts
- Understanding wireless communication with the NDS
  - Wireless communication devices in the NDS: Wi-Fi and Bluetooth
  - Description of DSWifi library methods for Wi-Fi
  - Use of high-level Mini-Wifi library methods for NDS
- Use of wireless communication in the NDS
  - Basic Wi-Fi functionality: Send/receive data between two NDS
  - Robust communication protocols - Protocols using acknowledgments
    - Stop-and-Wait ARQ protocol
    - Selective Repeat ARQ protocol

©ESL/EPFL

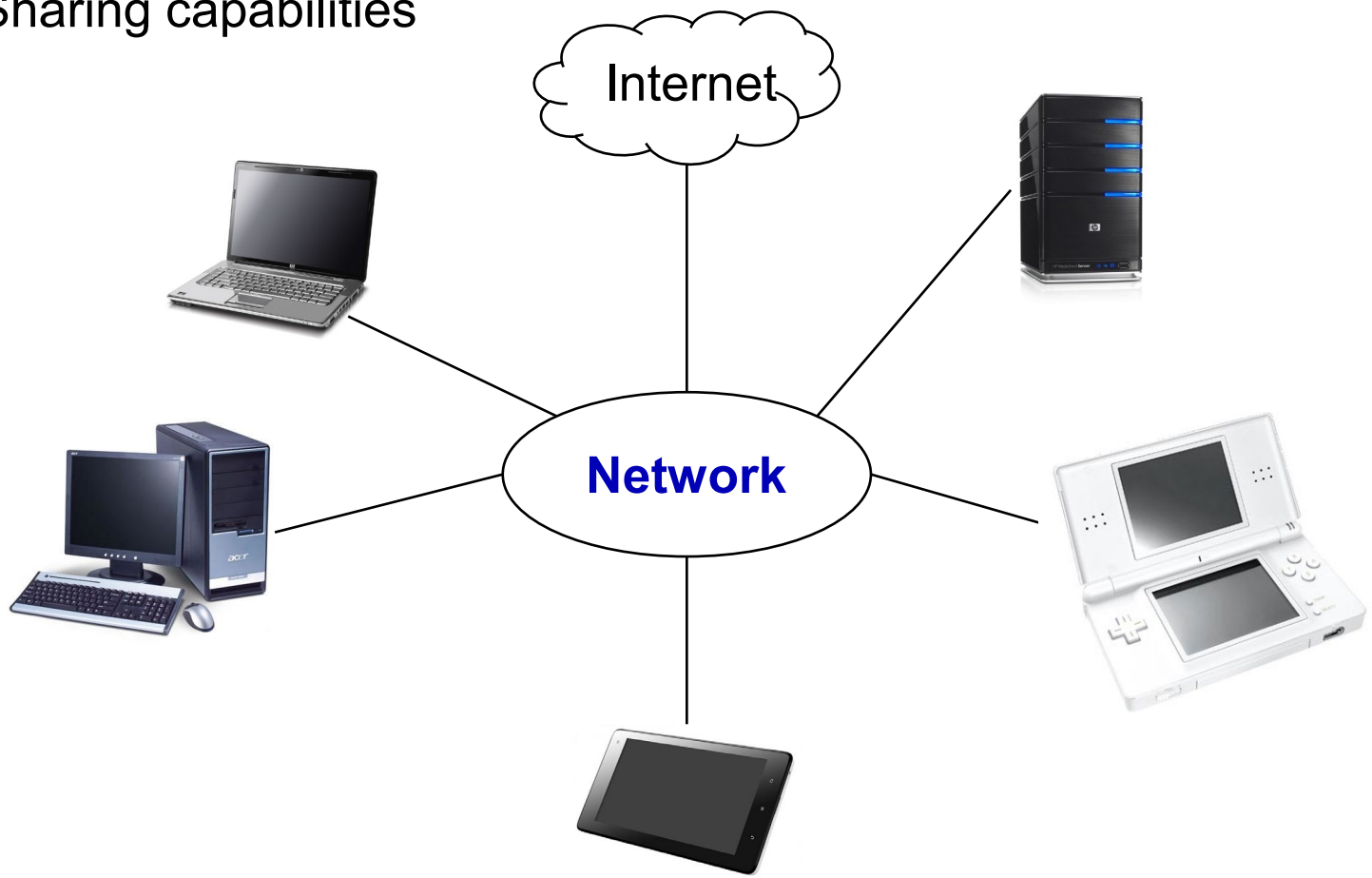
Connectivity in Embedded Systems

- Network capabilities are becoming common in almost any kind of consumer electronic devices
  - Smartphones, TVs, Cars, Refrigerators,…
- Wired and Wireless interfaces most integrated
  - Ethernet adapters, WiFi, Bluetooth, RFID, Zigbee
- Different types of networks depending on their characteristics and/or size
  - Near field communication (NFC), Body Area Network (BAN), Personal Area Network (PAN)
  - Local Area Network (LAN), Enterprise Area Network (EAN)
  - Metropolitan Area Network (MAN), Wide Area Network (WAN)

©ESL/EPFL

Basic networking: Abstract Network

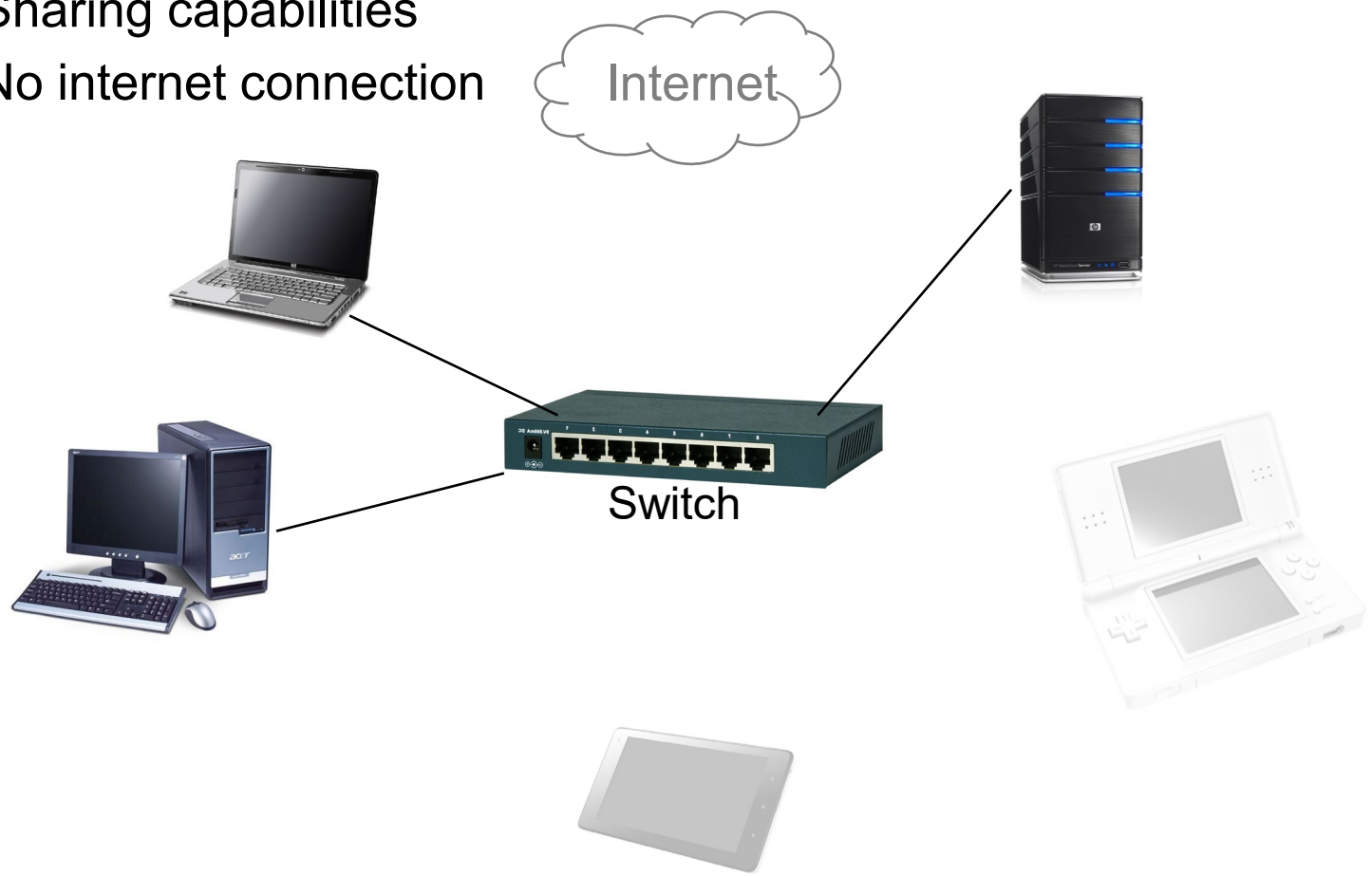
- Devices connect to a ‘common’ network
  - Internet access
  - Sharing capabilities



©ESL/EPFL

Basic networking: Wired LAN

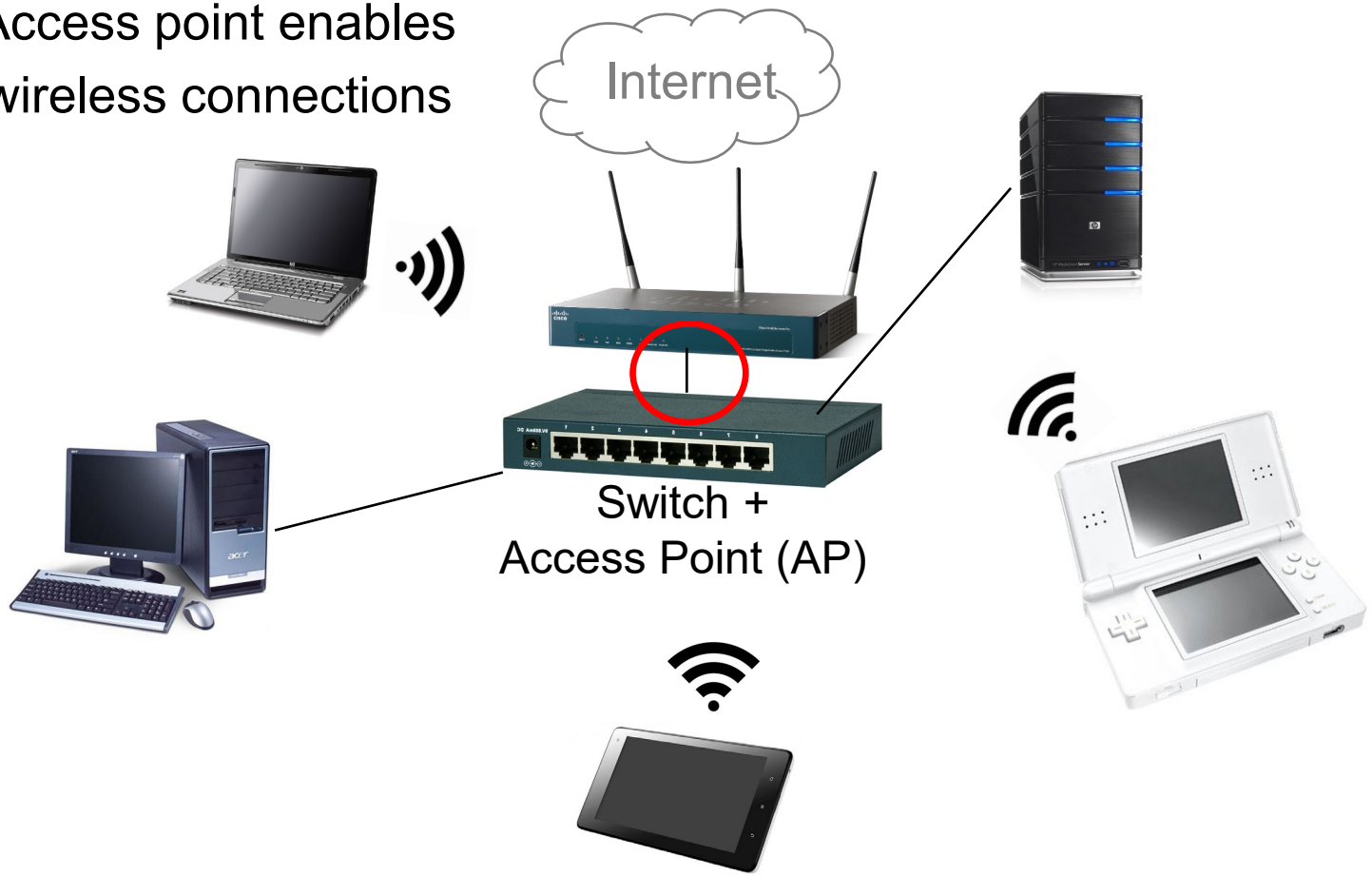
- Ethernet (IEEE 802.3)
  - Switch (or hub) interconnects the devices
  - Sharing capabilities
  - No internet connection



©ESL/EPFL

Basic networking: Wired + Wireless LAN

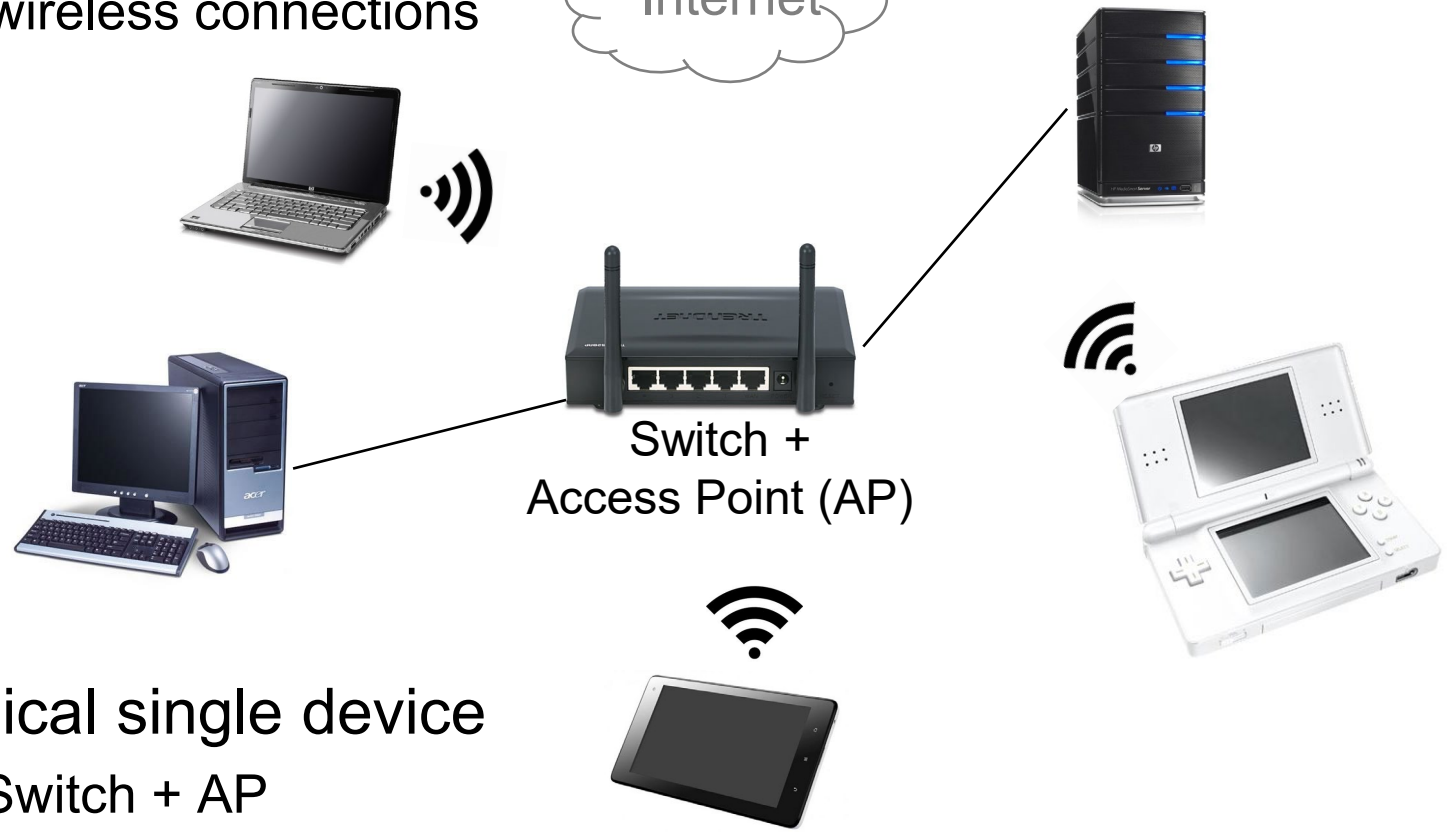
- Ethernet (IEEE 802.3)
- Wi-Fi (IEEE 802.11)
  - Access point enables wireless connections



©ESL/EPFL

Basic networking: Wired + Wireless LAN

- Ethernet (IEEE 802.3)
- Wi-Fi (IEEE 802.11)
  - Access point enables wireless connections

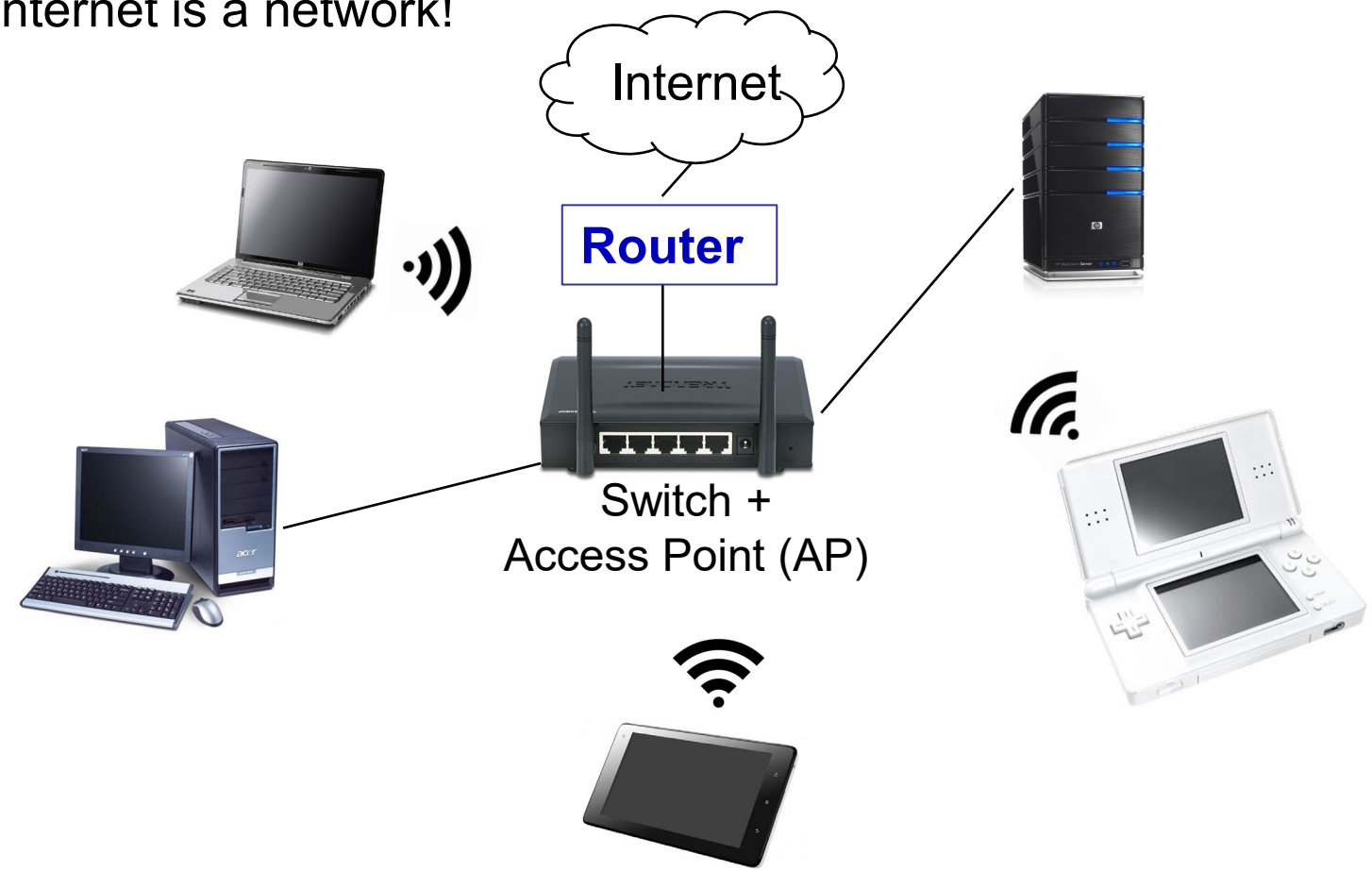


- Typical single device
  - Switch + AP

©ESL/EPFL

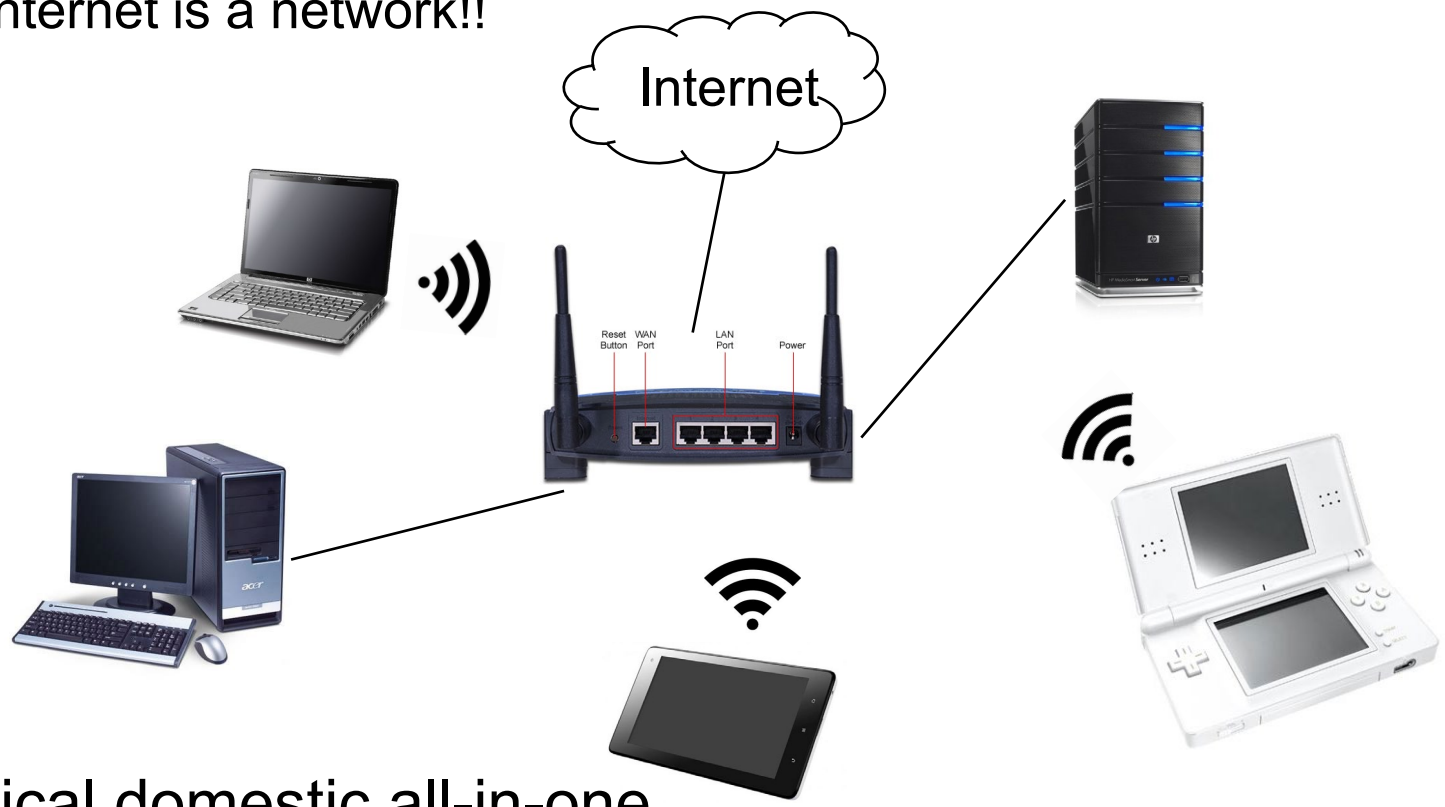
Basic networking: Network interconnection

- A Router connects two or more networks (e.g. Home LAN with provider WAN)
  - Internet is a network!



Basic networking: Network interconnection

- A Router connects two or more networks (e.g. Home LAN with provider WAN)
  - Internet is a network!!



- Typical domestic all-in-one
  - Router + switch + AP

©ESL/EPFL

Network identification on a LAN
Internet Protocol Basic concepts

- Devices connected to a network need an identification
- Internet Protocol (IP)
  - Protocol to exchange information between host in network(s)
  - Two versions IPv4 and IPv6.
    - IPv4 is widely used and will be used during the course
- IP Address
  - Unique for every device within a network
  - 32 bits address (divided in 4 bytes dot-separated for displaying)
    - 11000000 10101000 00000001 00001111 → 192.168.1.15
- Subnet mask or network mask
  - Divides the IP address into network prefix and rest (host id)
  - 32 bits: chain of 1s follow by a non-empty chain of 0s
    - 11111111 11111111 11111111 00000000 → 255.255.255.0
  - Bit-AND of IP address and mask gives network prefix

©ESL/EPFL

Network identification on a LAN
IP Basics

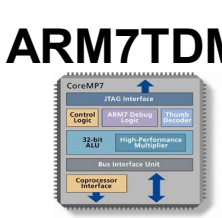
- Broadcast Address
  - Traffic sent to this address is supposed to be heard by all devices belonging to the same address
  - Address usually composed by the network prefix and a chain of 1s (maximum host id in the network)

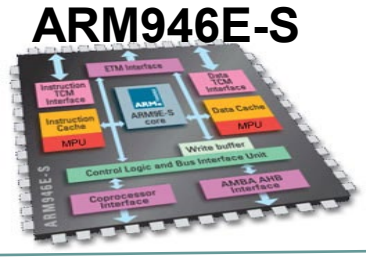
- Example
  - IP addr.: 192.168.1.15 → 11000000 10101000 00000001 00001111
  - Mask: 255.255.255.0 → 11111111 11111111 11111111 00000000
  - Network Prefix (Bit-AND):
    192.168.1.0 ← 11000000 10101010 00000001 00000000
  - Broadcast Address (network prefix + **max host id**):
    192.168.1.255 ← 11000000 10101000 00000001 **11111111**

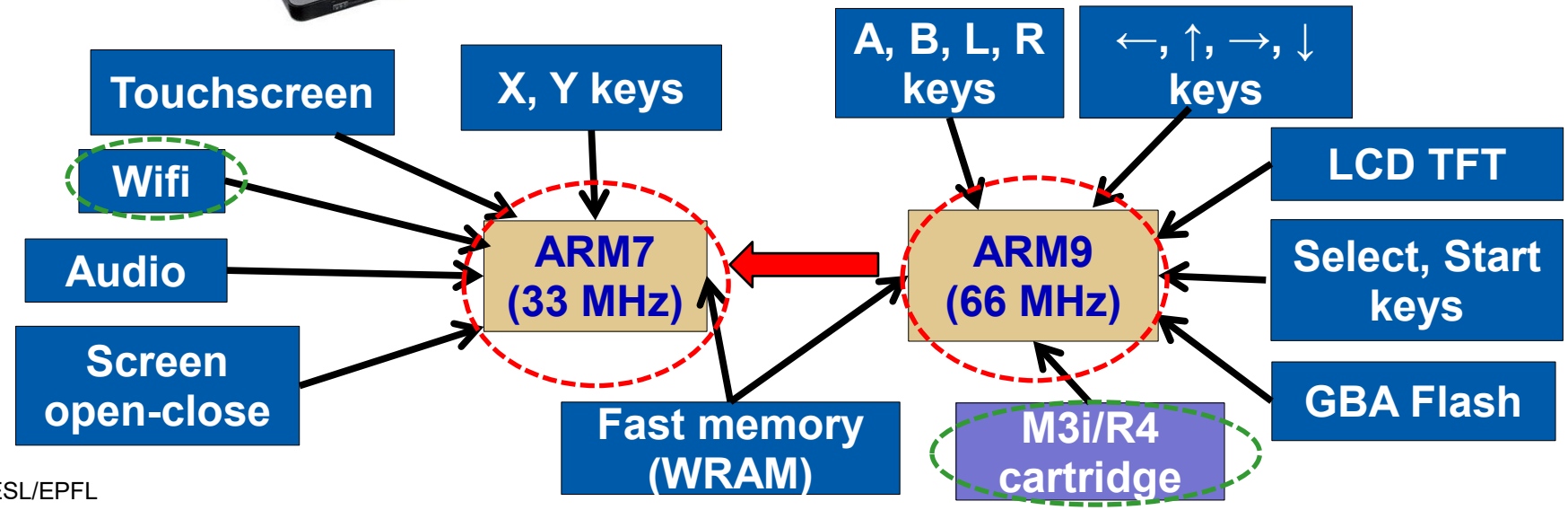
©ESL/EPFL

I/O subsystem management on the NDS: Wireless Communication

- Two interfaces available in the NDS for wireless communication
  - Wi-Fi or IEEE 802.11b standard: only the ARM7 processor has access to this device, so inter-processor communication (IPC) needed to use it by the ARM9.
  - Bluetooth: uses the SPI interface instead of the M3i/R4 cartridge, after boot-up







Differences between Wi-Fi and Bluetooth

- NDS provides Wi-Fi connectivity (IEEE 802.11 b/g)
  - Nintendo proprietary protocol (Ni-Fi) for gaming
- External modules can add more connectivity
  - Bluetooth, Infrared, zig-bee and more

	Wi-Fi (802.11b)	**Bluetooth (v2.0)**
Power	Higher power consumption	Lower power consumption
Range	Long-range (>30m)	Short-range (<10m)
Speed	up to 11 Mbit/s	Up to 3 Mbit/s
Target	Wireless Local Area Network (WLAN) Intended for wireless network infrastructure	Wireless Personal Area Network (WPAN) Intended for wireless ad hoc connection

©ESL/EPFL

NDS networking possibilities

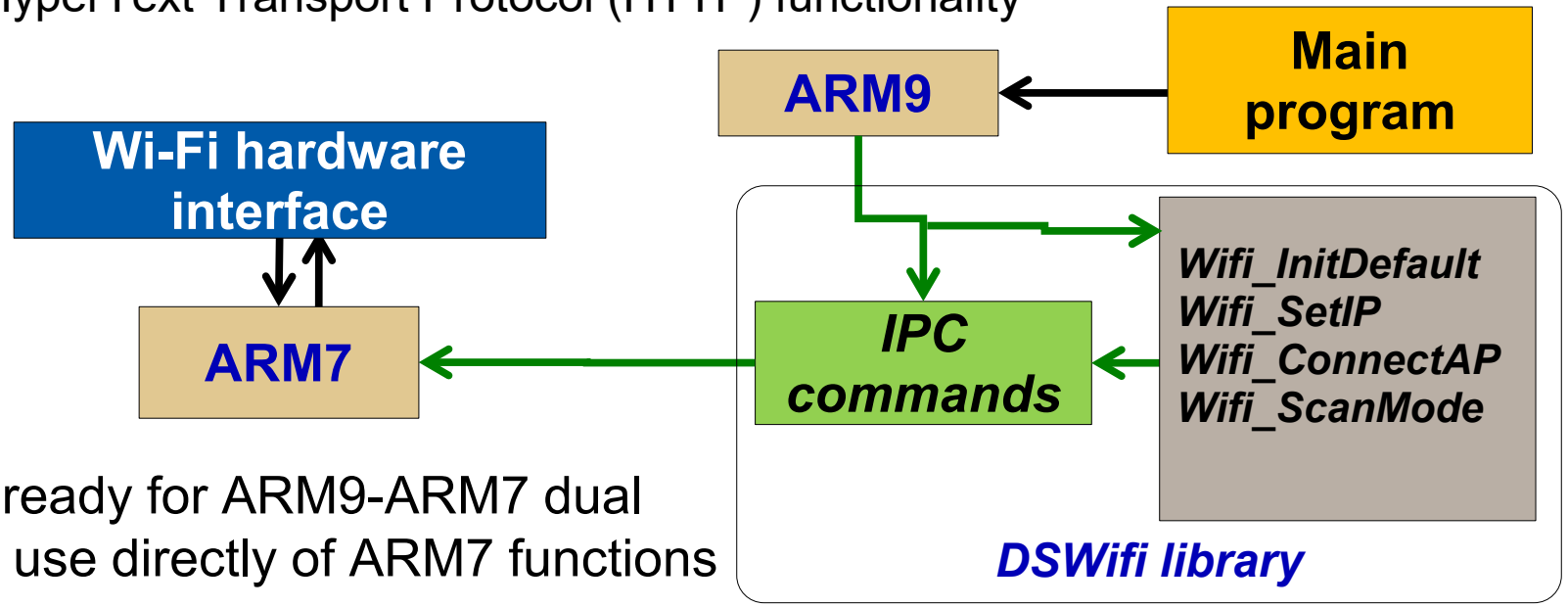
- Bluetooth peer-to-peer connection
  - Need of additional Bluetooth cartridge

- Wi-Fi Local Area Network
  - Need of an access point

©ESL/EPFL

Synchronization of Wi-Fi interface: use of DSWifi library in libNDS

- Control ARM9-ARM7 for Wi-Fi interaction with DSWifi library:
  1. IPC synchroniz.: ARM9 acts as master and ARM7 as slave in Wi-Fi commands
  2. Wi-Fi buffering: commands stored in internal NDS fast memories (WRAMx)
     - Synchronization with access points
     - HyperText Transport Protocol (HTTP) functionality



- DSWifi ready for ARM9-ARM7 dual project: use directly of ARM7 functions
- All the documentation available in the header files
  - /opt/devkitPro/libnds/includes/dswifi7.h
  - /opt/devkitPro/libnds/includes/dswifi9.h

©ESL/EPFL

DSWifi
Application Programming Interface (API)

- Limited networking capabilities are implemented to cover the four main aspects of Wi-Fi communication
  - Search and connection to Access Points (AP)
    - Open networks or with WEP encryption (WPA not supported yet)
  - IP assignment: DHCP client support if server is present
  - DNS resolution support: IP address can be obtained from a given Uniform Resource Locator (URL)
    - External Domain Name Service (DNS) server must be reachable
  - Send/receive commands: basic Wi-Fi socket support available
    - Open socket (setup bidirectional communication channel)
    - Send/receive data (UDP only)
    - Close socket

- Extra documentation can be consulted in the header files
  - **/opt/devkitPro/libnds/includes/netdb.h**
  - **/opt/devkitPro/libnds/includes/sys/socket.h**
  - **/opt/devkitPro/libnds/includes/netinet/in.h**

©ESL/EPFL

DSWifi
Application Programming Interface (API)

- Initialization
  - bool Wifi_InitDefault (bool useFirmwareSettings);
    - Initializes the Wi-Fi library. If the input parameter is true, the WiFi chip will try to connect using WiFi Connection **(WFC)**. This technology allows to save on-chip the parameters of up to 4 known access points.

- Network Configuration
  - void Wifi_SetIP (unsigned long IP, unsigned long gateway, unsigned long submask, unsigned long dns1, unsigned long dns2);
    - Sets the network configuration (Internet Protocol Address, Default gateway, subnet mask, DNS servers). All the parameters represent a valid IP address represented in a 32-bit hexadecimal number (i.e., 192.168.1.1 = 0xC0A80101)
    - If IP is set to 0, the rest of the parameters are ignored and all are obtained via DHCP

©ESL/EPFL

DSWifi
Application Programming Interface (API)

- Access Point Management and connection
  - int Wifi_GetNumAP();
    - Returns the number of reachable access points (AP).
  - bool Wifi_GetAPData (int index, Wifi_AccessPoint* apdata);
    - Returns true if success. The structure pointed by the second parameter will be filled the AP information specified in the position of the internal hidden (hardware managed) list.
  - int Wifi_ConnectAP (Wifi_AccessPoint* apdata, int wepmode, int wepkeyid, int wepkey);
    - Tries to connect to the access point specified in the structure of the first input argument. Returns 0 if successful or -1 in case of failure.
  - int Wifi_DisconnectAP();
    - Tries to disconnect the device from the AP (if any)

©ESL/EPFL

DSWifi
Application Programming Interface (API)

- Miscellaneous
  - unsigned long Wifi_GetIP();
    - Returns the IP of the device.
  - struct hostent* gethostbyname(const char * url);
    - Returns the data from the host specified in the input URL if it succeeds
      - Host name
      - Aliases
      - Address list

```
struct hostent {
    char * h_name;
    char ** h_aliases;
    int h_addrtype;
    int h_length;
    char ** h_addr_list;
};
```

©ESL/EPFL

DSWifi
Application Programming Interface (API)

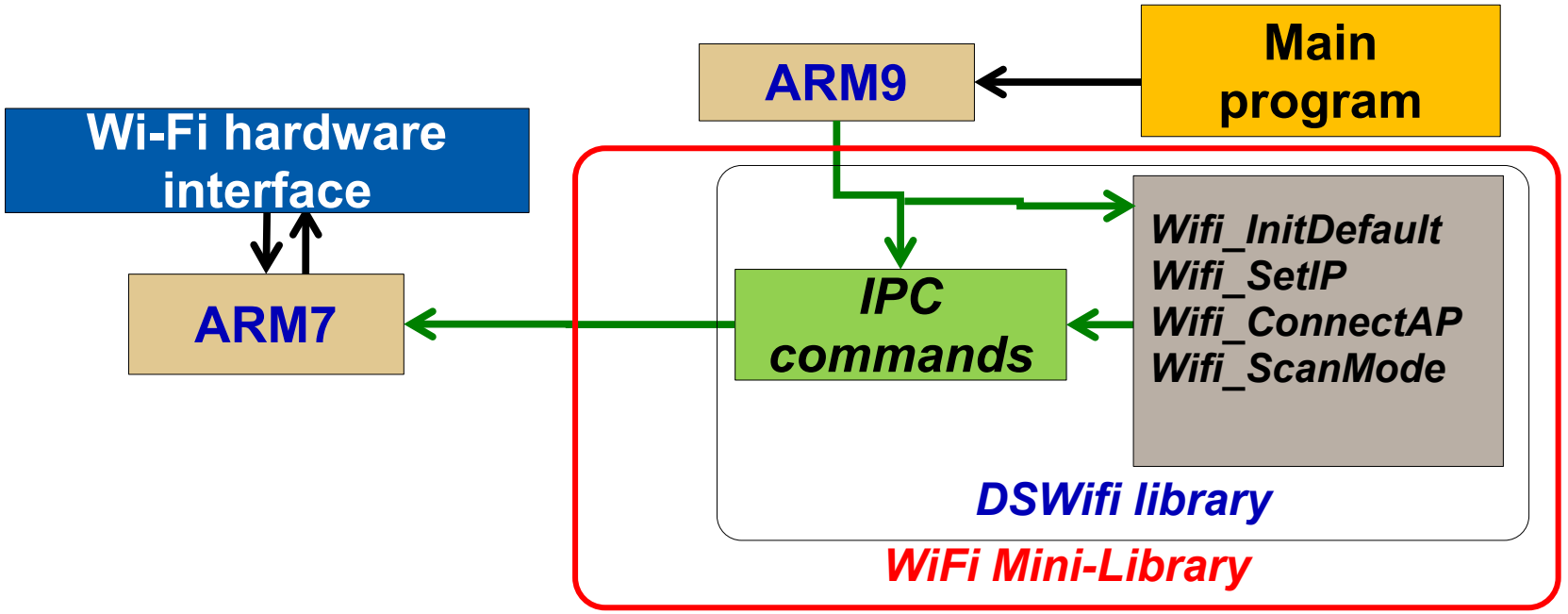
- Socket Management
  - int socket(int domain, int type, int protocol);
    - Opens an anonymous socket with a given type (UDP recommended) in the given domain (only IPv4 available for NDS) using the proper protocol (0 always in the NDS case). It returns the socket id.
  - int bind(int socket, struct sockaddr * addr, int addr_len);
    - Binds an opened anonymous socket with a given configuration
  - int shutdown(int socket, int shutdown_type);
    - Shut down read or writ operations (or both) of a given socket
  - int closesocket(int socket);
    - Closes the socket with the given id

```
struct sockaddr_in {
    unsigned short      sin_family;
    unsigned short      sin_port;
    struct in_addr      sin_addr;
    unsigned char       sin_zero[8];
};
```

©ESL/EPFL

Wi-Fi Mini-Library for NDS

- Simple library available in the Moodle Site for this course
- It illustrates the use of a basic Wi-Fi connectivity functionality
  - Wi-Fi initialization / de-activation
    - Based on WFC (need to be configured in advance by the TAs)
  - Socket management and data transfer
    - Based in UDP datagrams (packets) broadcasting



©ESL/EPFL

WiFi Mini-Library
Application Programming Interface (API)

- Wi-Fi Management
  - int initWiFi();
    - Initializes the Wi-Fi library and tries to connect to the stored access point using the WFC. Returns 1 on success and 0 otherwise.
    - The SSID of our access point is: "MES-NDS" (defined in "*WiFi_minilib.h*“)
  - void disconnectFromWiFi();
    - Closes the Wi-Fi connection by disassociating from the access point.

- Socket Management
  - int openSocket();
    - Opens a bidirectional UDP channel in the local port 8888 (defined in the library). Returns 1 in case of success and 0 otherwise.
  - Void closeSocket();
    - Closes the UDP bidirectional channel on port 8888 if it is opened.

©ESL/EPFL

WiFi Mini-Library
Application Programming Interface (API)

- Data transmission
  - int receiveData(char* data_buff, int n);
    - Tries to read n bytes from the socket (if it is correctly opened) and writes them into the array *data_buff*. It returns the number of read bytes (dumped to the array *data_buff*) or -1 if no read was possible.
      - **Non-blocking function**
  - void sendData(char* data_buff, int bytes);disconnectFromWiFi();
    - Tries to send n bytes of data from the array *data_buff* through the bidirectional UDP channel. Data is broadcasted and no confirmation is delivered by the sender.

©ESL/EPFL

Wi-Fi Mini-Library example: Bidirectional communication of pressed key

1. Wi-Fi Initialization
   - Connects to the access point configured in the WFC
2. Socket opening
3. SendMessage Function
   - Broadcasts 1 byte indicating pressed key (if any)
4. ReceiveMessage Function
   - Listens the port to check if any other user has sent a message

```
75 int main(void) {
76
77     consoleDemoInit();
78
79     //Initialize WiFi
80     if(initWiFi())
81         printf("WiFi OK!\n");
82     else
83         printf("Error WiFi\n");
84
85     //Open Socket
86     if(openSocket())
87         printf("Socket OK!\n");
88     else
89         printf("Error Socket\n");
90
91
92     while(1)
93     {
94         //Send a message if key is pressed
95         sendMessage();
96         //Receive message of someone else that pressed a key
97         receiveMessage();
98
99         swiWaitForVBlank();
100    }
101 }
```

©ESL/EPFL

Wi-Fi Mini-Library example: sendMessage() function

1. Read keypad by polling
2. Send message (1 Byte) with pressed key (A,B,X or Y)
   - Messages can be more complex and several bytes long.
   - The byte sent in the message is a token defined by the following enumerate:

```
typedef enum Messages_e {
    A,  //0x00
    B,  //0x01
    X,  //0x02
    Y   //0x03
}Message;
```

```
void sendMessage()
{
    char msg[1];

    //Poll the keypad
    scanKeys();
    unsigned short keys = keysDown();

    //Print and send a message if key pressed
    switch(keys)
    {
    case KEY_A:
        printf("You pressed A\n");
        msg[0] = (char)A;
        sendData(msg, 1);
        break;
    case KEY_B:
        printf("You pressed B\n");
        msg[0] = (char)B;
        sendData(msg, 1);
        break;
    case KEY_X:
        printf("You pressed X\n");
        msg[0] = (char)X;
        sendData(msg, 1);
        break;
    case KEY_Y:
        printf("You pressed Y\n");
        msg[0] = (char)Y;
        sendData(msg, 1);
        break;
    }
}
```

©ESL/EPFL

Wi-Fi Mini-Library example: receiveMessage() function

1. Listen to the UDP channel to check if someone sent a message
   - If receiveData returns a positive number, a message has been received

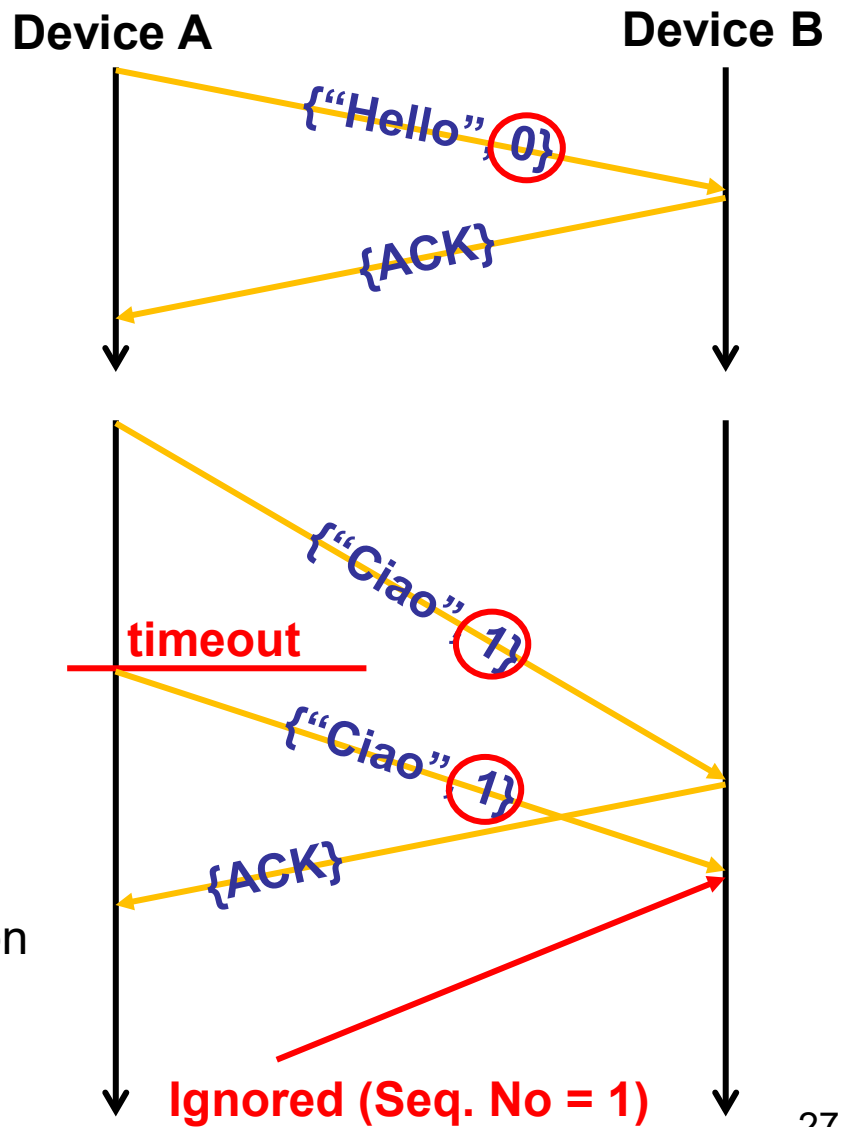
2. Parse message (1 Byte) if something was received and perform the corresponding action

```
49 void receiveMessage()
50 {
51     char msg[1];
52
53     //Listen for messages from others
54     if(receiveData(msg,1)>0 )
55     {
56         //If received, decode the key and print
57         switch(msg[0])
58         {
59         case A:
60             printf("Other pressed A\n");
61             break;
62         case B:
63             printf("Other pressed B\n");
64             break;
65         case X:
66             printf("Other pressed X\n");
67             break;
68         case Y:
69             printf("Other pressed Y\n");
70             break;
71         }
72     }
73 }
```

©ESL/EPFL

Robust communication: The Stop-and-Wait ARQ protocol

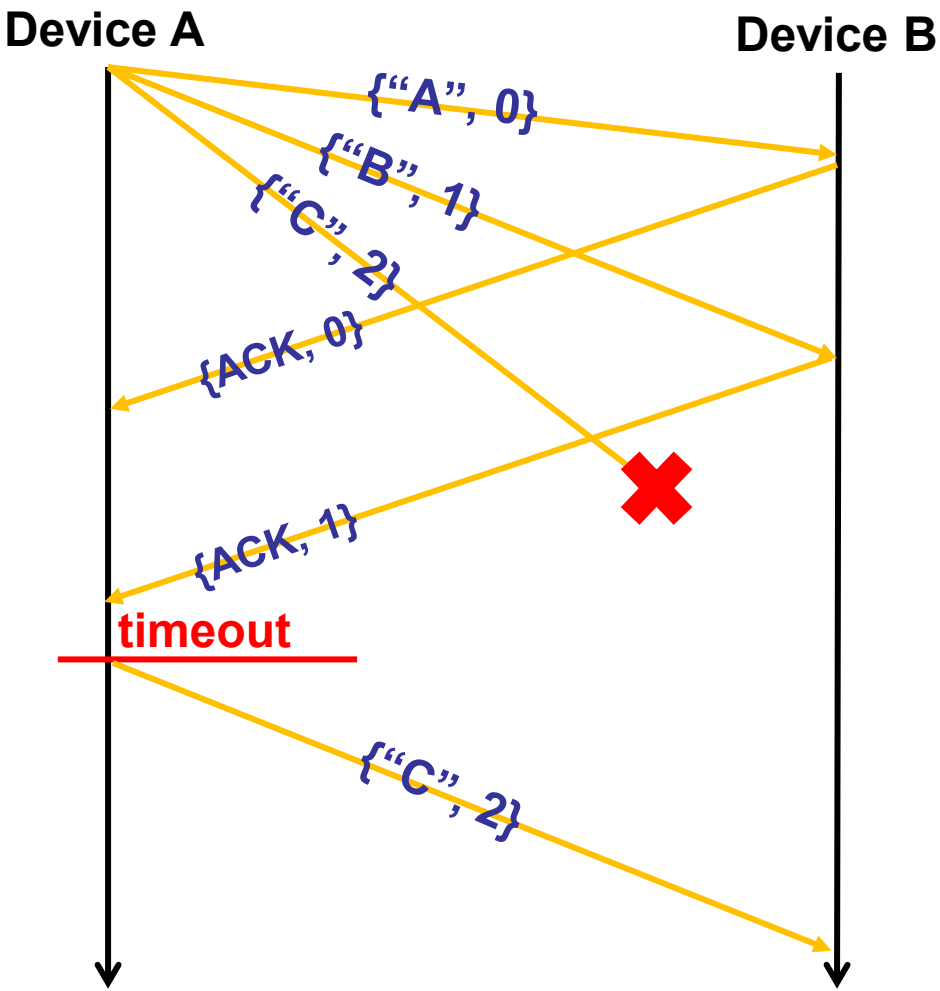
- Stop-and-Wait
  Automatic **R**epeat re**Q**uest (**ARQ**):
  - Sender: Send **one packet at a time**
  - Receiver: Send back **acknowledge (ACK)** message on receival
  - **Timeout**: If no ACK received → sender **retransmits same packet**
  - **Sequence numbers**: Each packet is stamped with a sequential number (**avoid duplicates**)
- Characteristics:
  - **Simple** implementation
  - **Reliable**
    - no packets lost
    - no duplicates
  - Supports **bidirectional** communication
  - **High latency**
    (waiting for ACK / 1 packet on air)



©ESL/EPFL

Robust communication: The Selective Repeat ARQ protocol

- Retains the ARQ principles:
  - Acknowledgment mechanism
  - Timeout
  - Sequence numbers
- Key features:
  - **Multiple packets on air**
    - maximum **X** packets on air
    - e.g., X = 3
  - **Selective retransmission**
    - keep track of lost packets
    - on timeout, retransmit them
- Improvements over Stop&Wait:
  - **Concurrency**
    - avoid idle waiting by sending more packets
  - **Latency**
    - faster ack processing



©ESL/EPFL

Practical Work 12: Wifi use in NDS

Exercises

- Exercise 1 - Bi-directional communication using Wifi
- Exercise 1b – Test the inclusion of acknowledgements to create a more robust communication

©ESL/EPFL

Questions?

Use the wireless connectivity in the NDS!