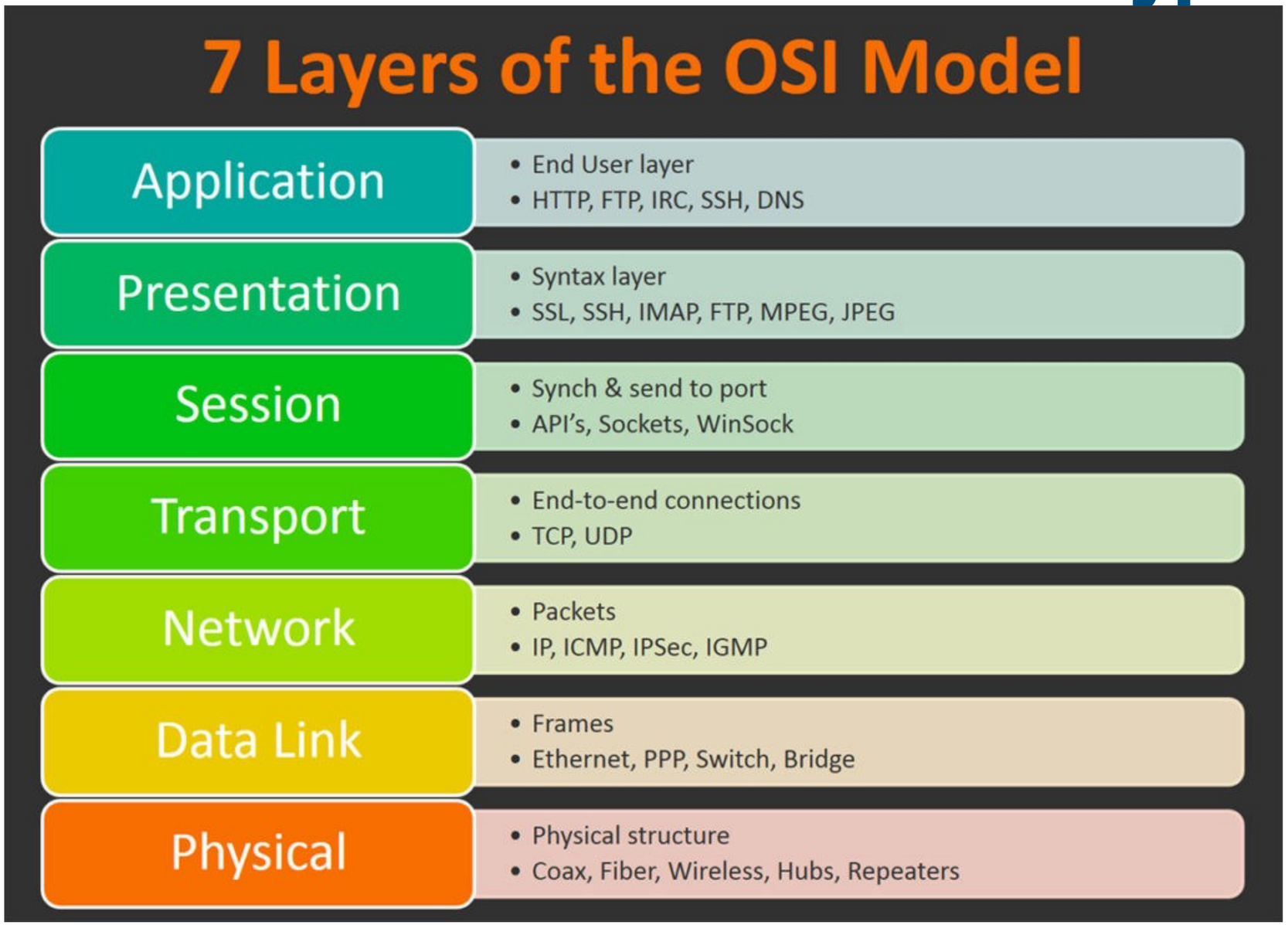
Internet of Things #2

Taco Walstra, June 2025

Outline

- Typical IoT communication standards
- BLE communication (lab #2)
- Asyncio in Python (lab #2)
- Publish Subscribe communication
  - ☑ MQTT
  - ☑ ZeroMQ (lab #2)
- IoT Manet Routing
  - ☑ DSR, AODV, Ant Colony routing
- Week 2 IoT

Internet and WiFi communication types

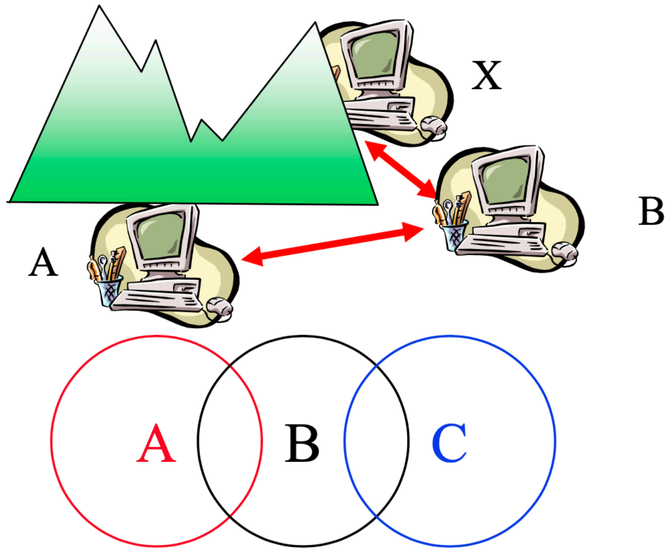


IEEE 802.11 - WiFi

- IEEE 802.11 is a standard but with MANY options
  - ☑ Equipment based on 802.11 can be incompatible…..
  - ☑ IEEE 802.3 => ethernet
    IEEE 802.11 => WiFi
    IEEE 802.15 => ZigBee, 6LowPAN
    IEEE 802.16 => WiMAX
- WiFi = Wireless Fidelity, trademark.
  - ☑ Test compatibility between different manufacturers of wireless equipment
  - ☑ Equipment with WiFi logo should be interoperable.

IEEE 802.11

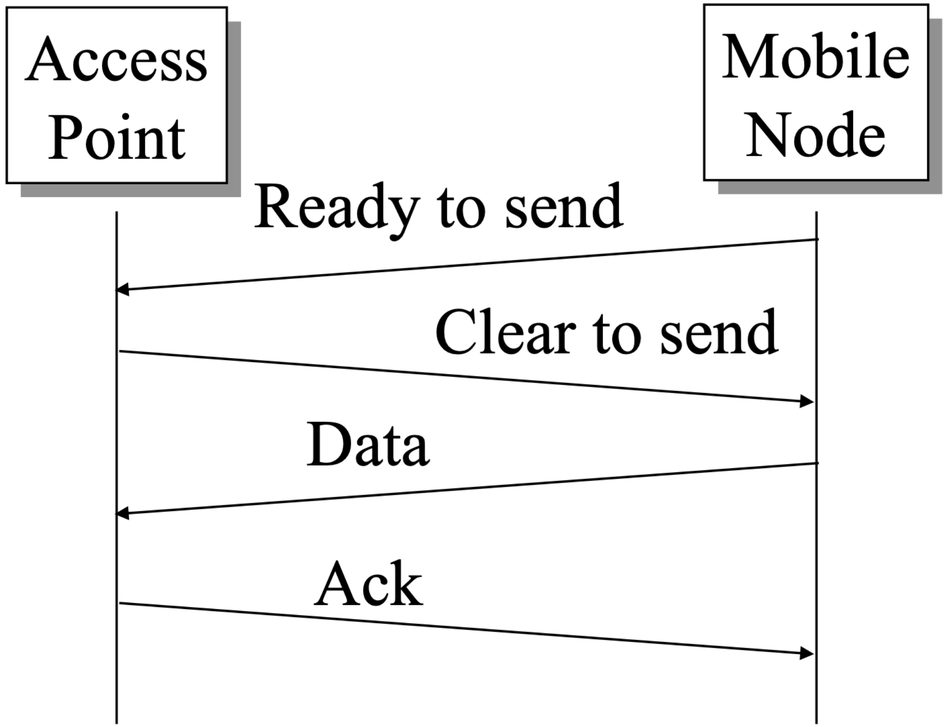
- Original version 1997 was 1 and 2 Mbps
  - ☑ Newer versions: 11 Mbps, 54 Mbps, 108 Mbps, 200 Mbps etc.
- Collisions are possible (**hidden node problem**):
  - ☑ A is able to comm. with B (receiver/access point)
  - ☑ B is able to comm. with C
  - ☑ C cannot comm with A
  - ☑ A and C can start transmission simultaneously, the signal collide in B resulting in data loss



  - ☑ If a device cannot sense another's transmission it might wrongly assume the medium is free, leading to collisions

Hidden node problem

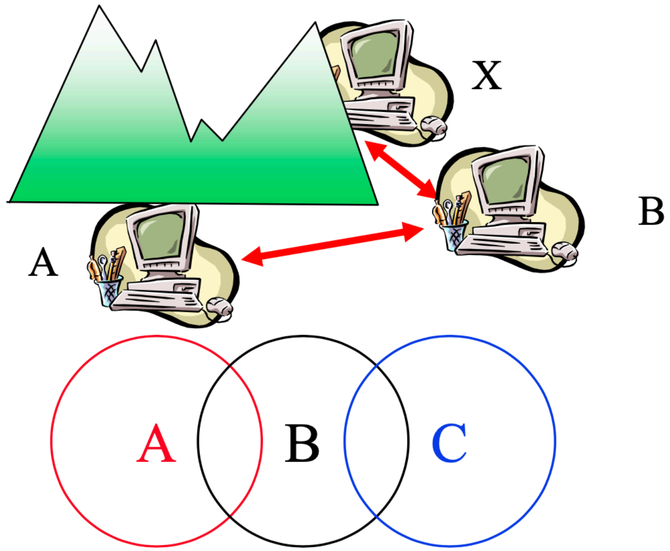
- Solution: use a **handshake** protocol:
  - Send RTS
    - Everybody should shut up for some specific period of time
  - Destination sends CTS
    Other will wait for duration of message
  - Each packet is acknowledged (ACK)
  - Retransmission if ACK not received.



- Introduces latency and there is overhead
  - Overhead can be problematic when sending short messages.

Hidden node problem

- Solution2: reduce the transmission power or adjust the placement of antenna’s
  - ☑ Improves spatial separation



Suitable IoT transmission datalinks

- There are many systems: IEEE 802.11, Bluetooth low energy, 802.15: ZigBee, (6LowPAN), LoRaWAN, WiMAX ……
  - ☑ All different because we have many different requirements
    - Small distance transmission - Large distance transmission
    - Low/Medium power - ultra low power (hours/days vs years on 1 battery)
    - Low datarate - Large datarates
    - Light weight protocols vs high security / encryption

Power usage

Type	Bit rate	TX power	mJ/MB
802.11b (wifi)	11 Mbps	50 mW	36.4
802.11g (wifi)	54 Mbps	50 mW	7.4
802.11a (wifi)	54 Mbps	200 mW	29.6
802.15.1 (bluetooth)	1 Mbps	1 mW	8.0
802.15.3 (bluetooth)	55 Mbps	200 uW	0.03

BLE (Bluetooth Low Energy) range

- Range can be limited when used indoor (wall and objects give attenuation + multi-path effects due to interference from signals coming at different angles)

- 1-2Mbps PHY: Bluetooth Classic V2.1
  2 Mbps PHY: Bluetooth LE V5.0, more throughput but range is limited
  coded PHY: long range, but only 500 kbps or 125 kbps

	Bluetooth Classic (v2.1 and v3.0)	**Bluetooth LE (v4.2)**	**Bluetooth 5 LE 2Mbps**	**Bluetooth 5 LE Long Range**
Range	Up to 100m	Up to 100m	Up to 50m	Up to 400m
Range (Free Space)	Up to 100m	Up to 100m	Up to 50m	Up to 400m

Source: https://argenox.com/library/bluetooth-low-energy/maximizing-bluetooth-low-energy-ble-range

Power and networking problems

- Power issues
  - ☑ Wifi nice for large bandwidth, but power hungry.
  - ☑ Standard Bluetooth looks promising for power usage, but maintains connection even when no data is transmitted.
    - ▶ Not suitable for ultra low power applications (battery lifetime 1 or more years)
  - ☑ Possible solutions: BLE (Bluetooth low energy), or 6LoWPAN: low power wireless personal lan: Compression of IPv6
- Network issues
  - ☑ IPv4 is too small (not enough ip addresses. IPv6 too much overhead
  - ☑ 48 bit device address in Bluetooth not sufficient for large number of devices

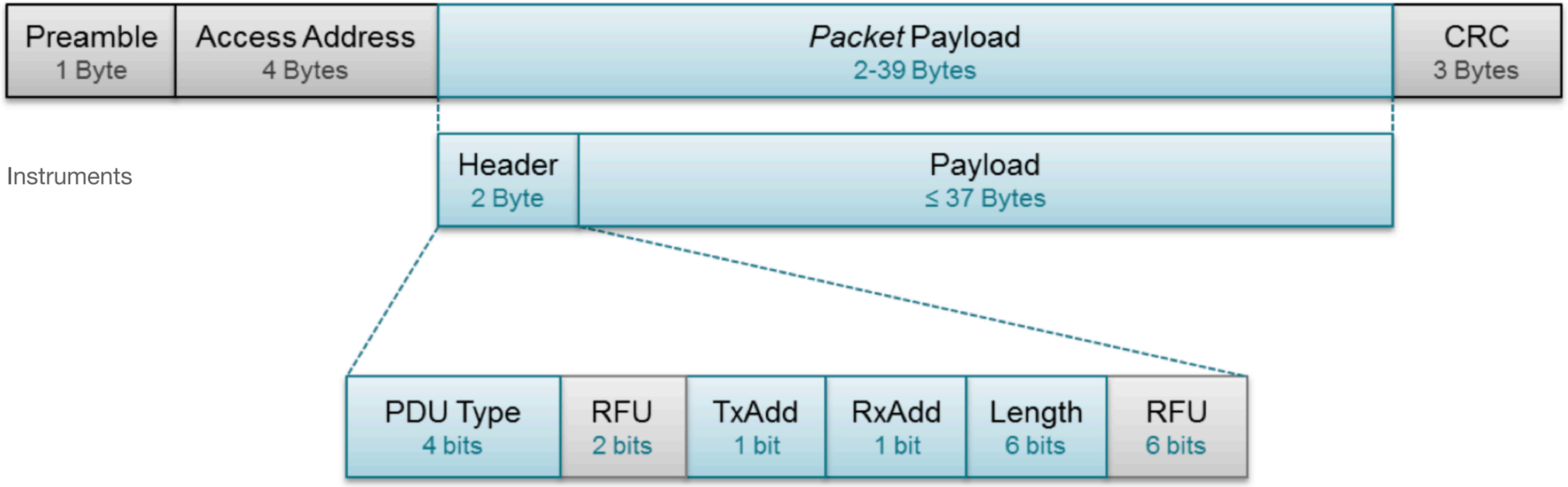
Outline

- Typical IoT communication standards
- BLE communication
- Asyncio in Python
- Publish Subscribe communication
  - ☑ MQTT
  - ☑ ZeroMQ
- IoT Manet Routing
  - ☑ DSR, AODV, Ant Colony routing

Bluetooth

- Started as Ericssons project in 1994 for radio communication between cell phones over short distances
- 1998: Intel, IBM, Nokia, Toshiba, Ericssons form Bluetooth agreement/ standardization.
- Key features
  - ☑ 10 mA standby, 50 mA transmitting
  - ☑ Low cost
  - ☑ Small chips (9 mm^2 or less)
  - ☑ 1 Mbps initial datarate
- Later versions: 24 Mbps, low energy (Bluetooth 4.0 2010), larger packets (dec. 2014)

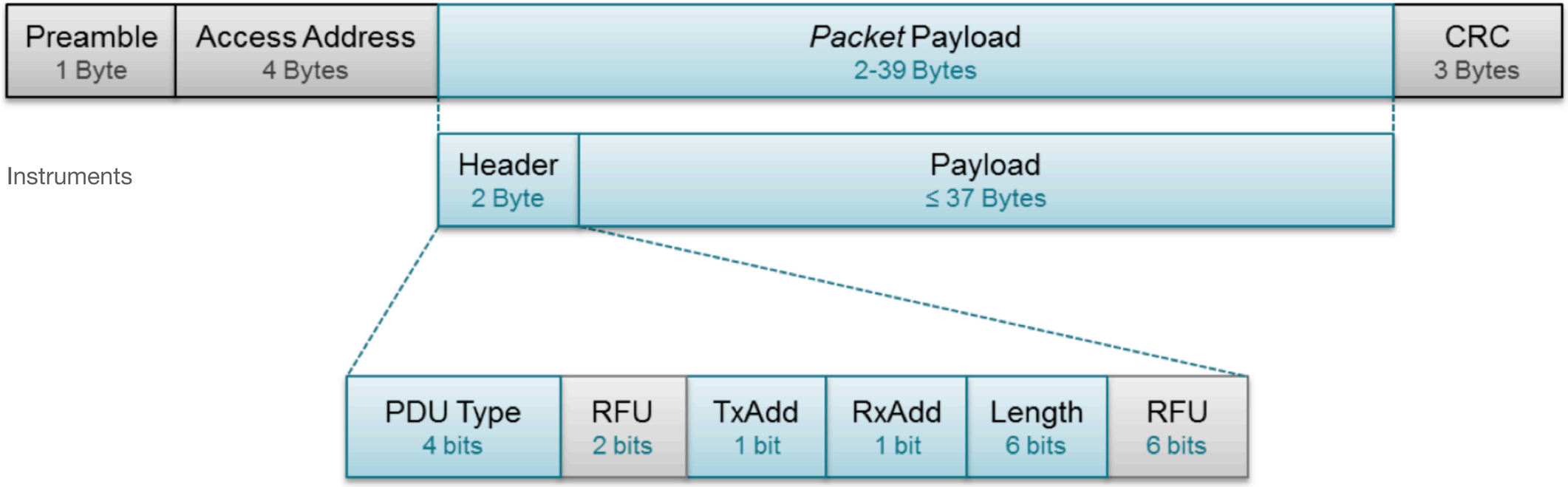
Bluetooth packets



Source: Texas Instruments

- Preamble 1 byte: 0xAA for broadcasting packets, or 0xAD, or 0x55. or 2 byte value for LE 2M PHY devices)
- Access Address 4 bytes: fixed to 0x8E89BED6 for broadcast packets. Random value to “lock” receiver to this device.

Bluetooth packets



Source: Texas Instruments

- PDU determines packet type: advertisement, scan request, connection request
- RFU: reserved
- TxAdd: public or random

Bluetooth packets

- Bluetooth devices have an unique device address of 48 bits:
  - ☑ Manufacturer ID: 3 bytes
  - ☑ Device ID: 3 bytes
- Payload is the data transmitted/received: 2 - 37 bytes for BLE “advertising” packets. 257 bytes for “data packets”
- CRC is an error detection check
  - ☑ 3 bytes calculated from payload data by using some fixed polynomial
- More information: https://www.bluetooth.com/blog/bluetooth-low-energy-it-starts-with-advertising/

Bluetooth vs BLE

- BLE (Bluetooth Low Energy) uses the same protocol as Bluetooth
- They differ in power consumption, range, throughput, connection speed, and number of connections
- BLE devices are able to work 5-10 years on a small coin cell battery
  In general they use approx. 1/10 of the energy of normal Bluetooth devices.
- The range of BLE devices is up to 25 m. Under optimal conditions up to 100 or 400 m, depending on interference and obstacles. Accuracy for positioning can reach centimetres

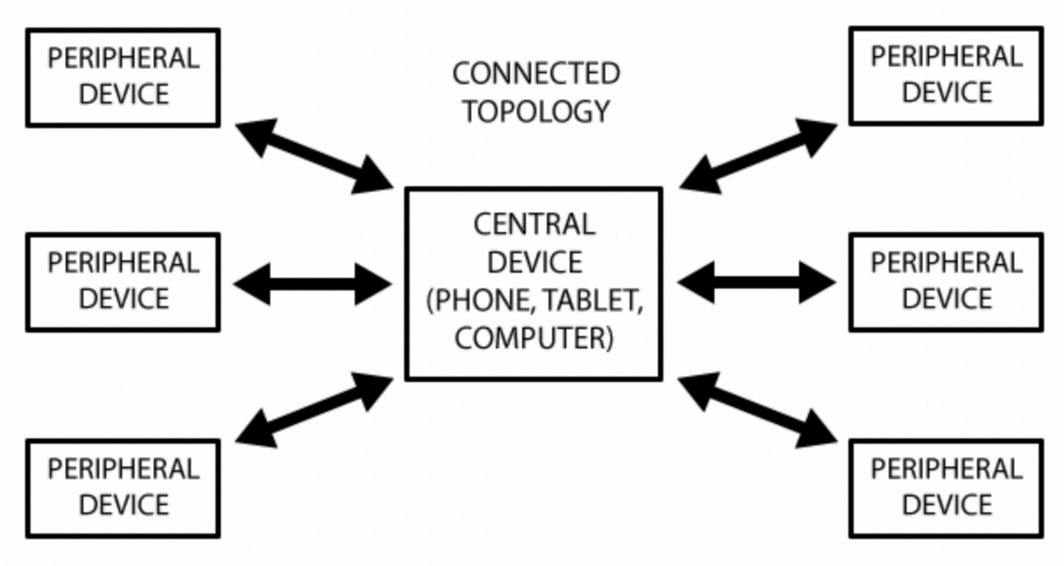
BLE device characteristics

- BLE device are connected to an environment where each peripheral is connected to only one central device like a laptop or smartphone.

- Two important protocols are being used:

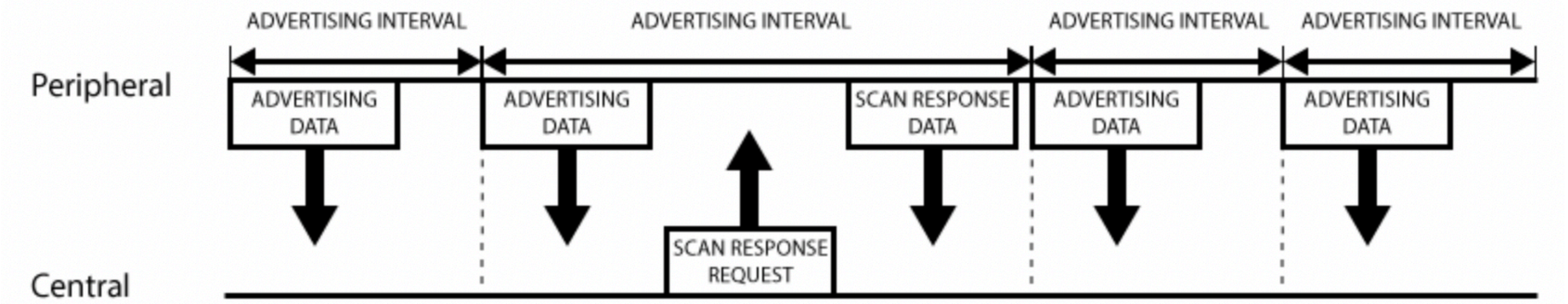
  - ☑ GAP

  - ☑ GATT



GAP (Generic Access Profile)

- This controls how a device is **advertising** itself
  - ☑ it determines how the device is visible to the outside “bluetooth” world and how 2 devices can interact with each other
- The **GAP** can transmit an **Advertising Data** payload or it can send a Scan Response payload.
  - ☑ both payloads are identical and contain 31 bytes
  - ☑ the Advertising payload is mandatory (and continuous)
  - ☑ The Scan response payload is send when a central device asks for it



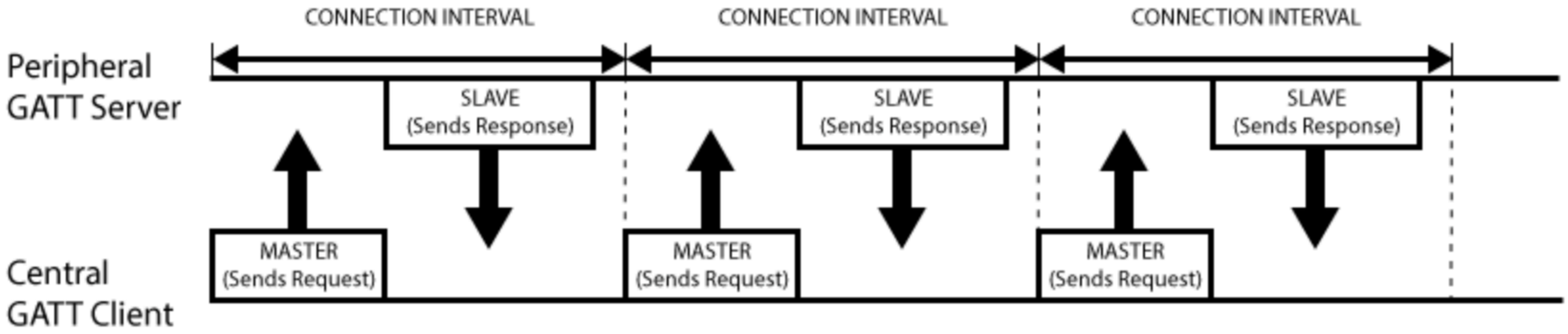
Connection with a central peripheral

- When a BLE device has established a connection with a central peripheral it will **stop** advertising itself.
- We use the **GATT services** for communication between device and central peripheral
- A BLE device can only be connected to **one central peripheral**

GATT (Generic ATTribute) Profile

- Determines how 2 BLE devices or BLE communicate with each other
- It stores **Services** and **Characteristics**
- The communication follows a client-server model
  - ☑ the server holds the ATT lookup data and characteristics definitions
  - ☑ the client sends requests to the server and the server will send a response.
- The BLE peripheral will in general ask for a “connection interval”: the central device will try to reconnect to the BLE device to see if data is available.
  - ☑ see next page. Note: the BLE device is here server and sends responses to a slave client (your laptop for example)

GATT (Generic ATTribute) Profile



- Gatt transactions are nested objects called **Profiles, Services** and **Characteristics**

- **Profiles** are listed in a list compiles by the standard. See https://www.bluetooth.com/specifications/specs/

- **Services** are logical entities consisting of chunks of data and have an unique ID (UUID). the are 16 bit or 128 bits.

GATT (Generic ATTribute) Profile

- **Services** use the list which is available at https://www.bluetooth.com/specifications/assigned-numbers/

- For example a **heart rate service** use a 16 bit UUID which is 0x180D and the data which refer to the characteristics.

- The service can have multiple **characteristics** of which one is mandatory

- For example the heart rate **service** has 3 characteristics: heart rate measurement, body sensor location, heart rate control point

- Each **characteristic** contains again a UUID which is 16 bit or 128 bit.

  - ☑ the heart rate characteristic UUID is 0x2A37 and actual data

  - ☑ The characteristics are the central point where you interact with the data of a sensor

Lab #2 BLE communication

- Read https://docs.arduino.cc/libraries/arduinoble/
- Execute first a few examples and try to understand the code.

Outline

- Typical IoT communication standards
- BLE communication
- Asyncio in Python
- Publish Subscribe communication
  - ☑ MQTT
  - ☑ ZeroMQ
- IoT Manet Routing
  - ☑ DSR, AODV, Ant Colony routing

Asyncio library in Python

- What is the difference between parallelism, concurrency and threading?

Asyncio library in Python

- What is the difference between parallelism, concurrency and threading?
- parallelism / multiprocessing will spread tasks over multiple processor cores
  - ☑ especially suitable in for loops where identical tasks are executed without mutual dependencies
- Concurrency means that the tasks execute in overlapping manner
  - ☑ concurrency does NOT imply parallelism
- Threading is a concurrent model where multiple threads execute the tasks
  - ☑ one process can have multiple threads.
  - ☑ Python has some difficulties here “thanks” to the GIL (Python Global Interpreter)

Intermezzo: GIL (Python Global Interpreter)

- Is basically a mutex
  - ☑ locks the python interpreter for one thread
  - ☑ only one thread executes which creates a performance bottleneck in multi-threaded code
  - ☑ A multicore processor (nowadays almost every processor, except some small processors for embedded purposes) will still execute only one Python thread!
  - ☑ Purpose is to protect memory race conditions where 2 or more threads increase or decrease a value simultaneously
    - memory blocks that are never released or
    - release memory blocks that are still in use
    - Reference counter will lock all data structures shared among threads
- **demo of single_threaded, multi_threaded and multiprocessing in Python**

Asyncio in Python

- I/O tasks involve a lot of waiting for something to happen…
  - ☑ wait until serial buffer empty, wait until buffer has been sent etc.
- Python supports:
  - ☑ threading: threading library
  - ☑ multiprocessing: multiprocess library
  - ☑ concurrency: concurrent library (especially concurrent.futures package)
- Asyncio is single process, single threaded. It uses “cooperative multitasking”:
  - ☑ gives you the feel of concurrency while still single thread

Asyncio in Python

- Asyncio in Python is related to the threading library but contains an asynchronous flavour:
  - ☑ async routines are able to “wait” for something to happen and allow other routines to continue to be executed.
- a far-fetched but still interesting example
  - ☑ 4 slow arduinos, 1 laptop. Laptop takes 3 milliseconds to execute a task, the arduinos take 40 milliseconds to finish a task. A complete task takes 30 operations of arduino+laptop.
  - ☑ synchronous version: laptop takes for each operation: (3+40 ms) * 30=1290 ms. = 1.29 seconds. with 4 arduinos: 5.16 seconds.
  - ☑ asynchronous version: after laptop executes first operation it executes the next. so: one operation on all 4 arduinos takes 4*3 = 12 milliseconds. For 30 operations: 12*30 = 360 ms = 0.36 seconds!

Asyncio in Python

- **demo1: countasync.py and countsync.py**

- The await functions will give control back to the event loop.

- if you use **await** to get results from a function to return values the caller must be defined as **async**

  ☑ def myfunc( ):
  do something…
  return (val1, val2)

  async def callerfunc( ) :
  (val1, val2) = **await** myfunc( )

Asyncio and Bleak in Python

- The **Bleak library** is used for Bluetooth communication and you will need to make use of the async constructions

- UUID is a globally unique identifier of 128 bits used to identify services

  - ☑ BLE supports a shortened 16 bits UUID, but we will use some predefined ones in the python templates. Note that this is not “just” a hexadecimal number; it’s located in BLE libraries, not in Arduino libraries!

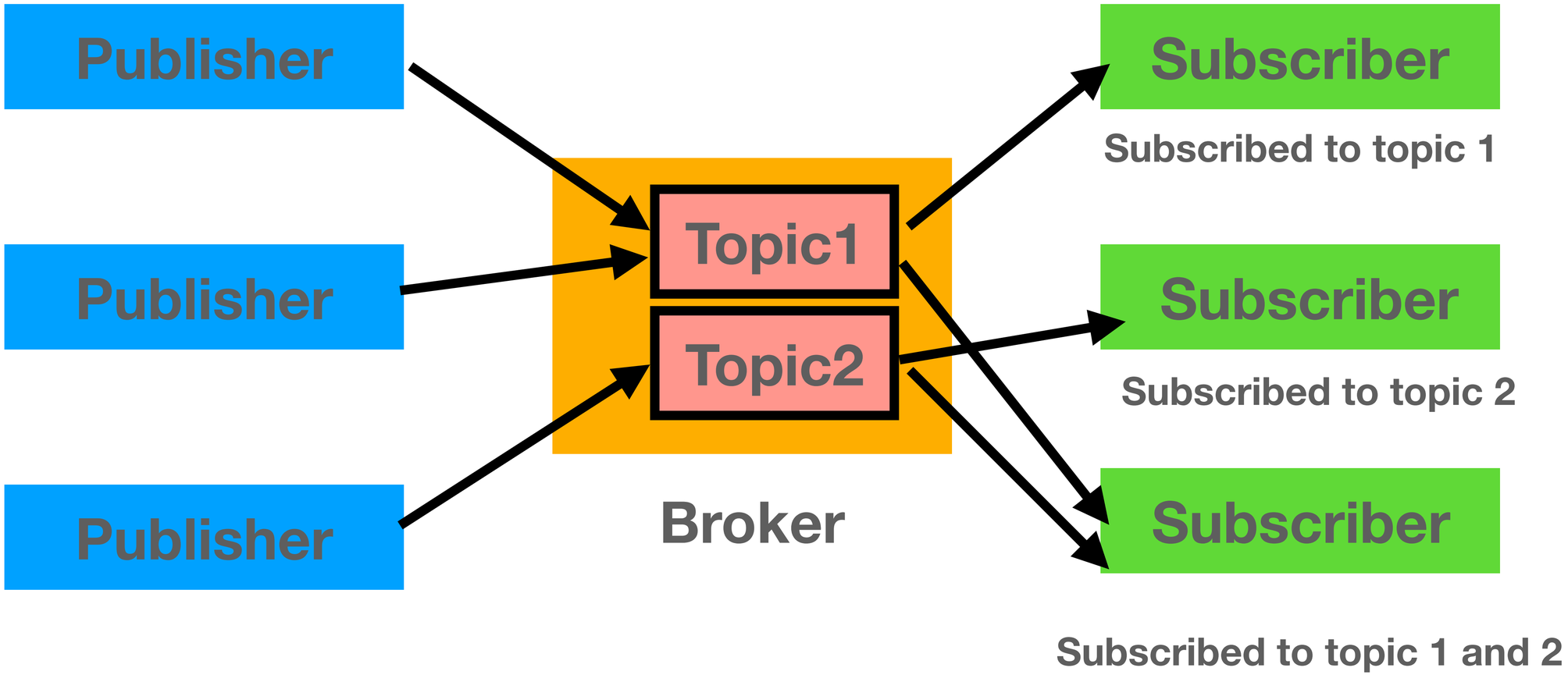
- demo ble_scan.py

- (demo get_device_info.py)

Outline

- Typical IoT communication standards
- BLE communication
- Asyncio in Python
- Publish Subscribe communication
  - ☑ MQTT
  - ☑ ZeroMQ
- IoT Manet Routing
  - ☑ DSR, AODV, Ant Colony routing

Publish subscribe model



Publish subscribe model

- Translated to your Arduino environment:
  Sensor data read by Arduino and transmitted by BLE or WiFi to your laptop.

- Laptop reads the “topic” data (sensor values) and is a “broker” / server for clients connected to the server.

- Clients only receive data on subscribed topics

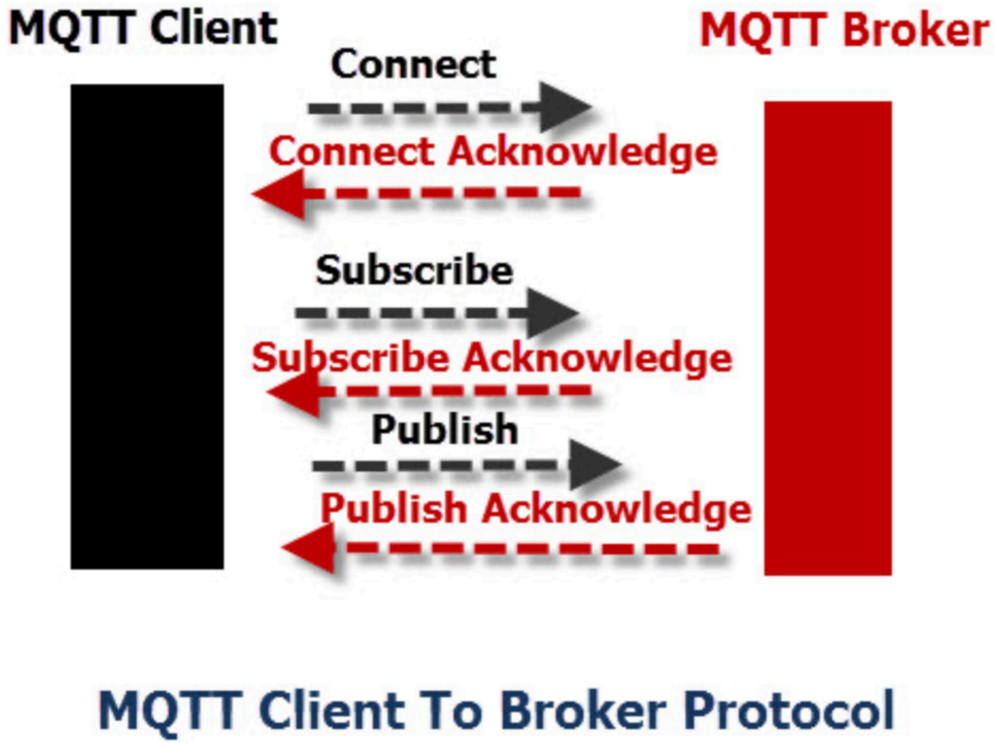
- Topics: acceleration data, gyroscope data, battery voltage, BLE signal strength, temperatures …..

MQTT

- Message Queue Telemetry Transport
  - ☑ “Lightweight” messaging protocol **on top of TCP/IP** (but can be done on Bluetooth as well…)
  - ☑ Publish - Subscribe model
  - ☑ A “message broker” is required
  - ☑ Standard:ISO/IEC PRF 20922
  - ☑ Small code footprint, royalty free since 2010
  - ☑ Binary protocol

MQTT - publish subscribe model

- The MQTT broker accepts publishers and subscribers as client
- A publisher sends data like measurement data to the broker for a specific “topic”
- A subscriber subscribes to a “topic” and receives all data which is related to this topic.



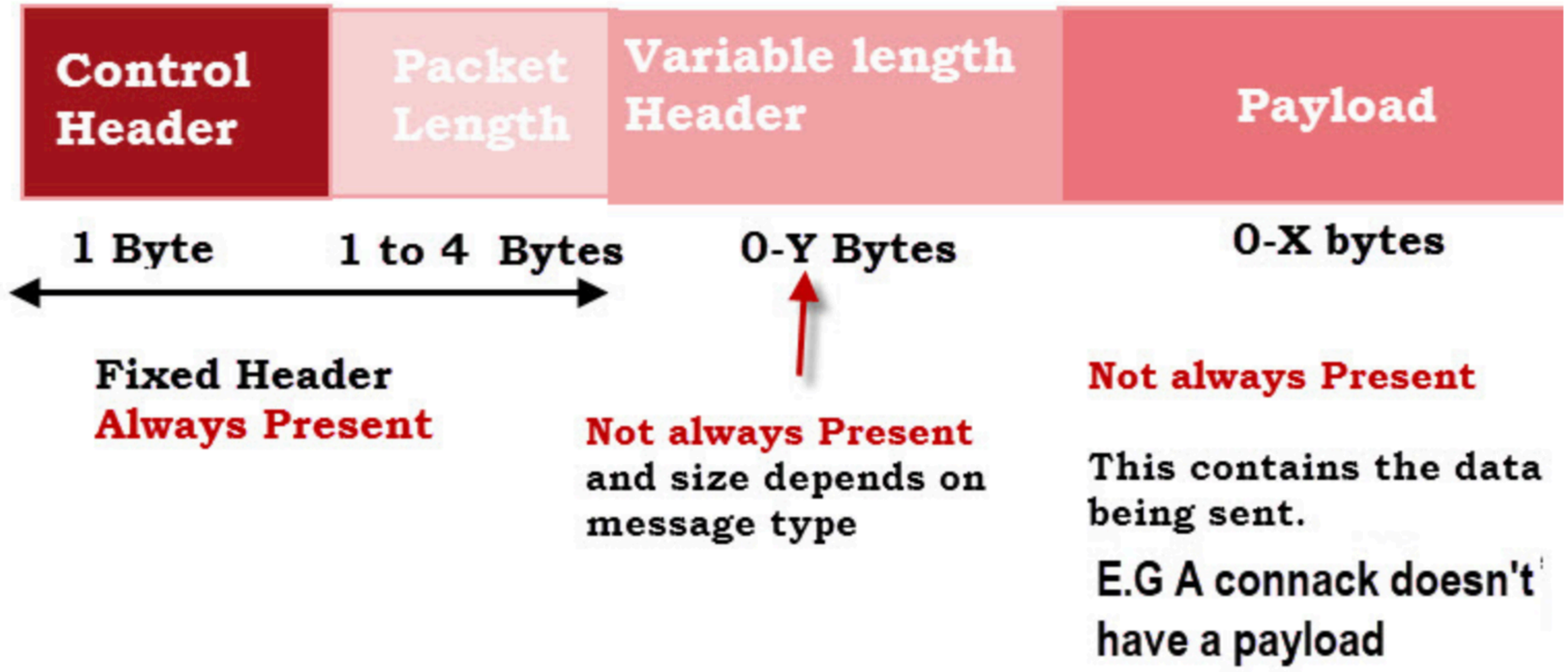
MQTT Client To Broker Protocol

MQTT - publishing data

- After a sensor has connected to the broker it can publish its data

MQTT packets

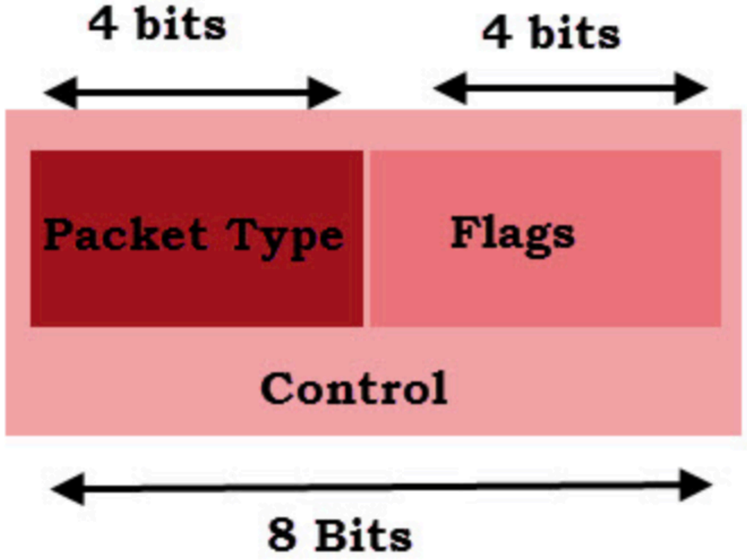
- Each MQTT command is acknowledged.
- Control elements are binary bytes, not text strings.. Control header consists of a control field and a packet length field. Minimum size: 2 bytes.



MQTT Standard Packet Structure

MQTT packets

- Control packet



Packet Type Examples:

Connect =0001 =1
Connack=0010 =2

Disconnect=1110=14

Name	Value	Dir	Desc
CONNECT	1	Client to server	Client request to server. Control byte= 0x10
CONNACK	2	Server to client	Connect acknowledge. Control byte = 0x20
PUBLISH	3	Client to Server	Publish message
PUBACK	4	Server to Client	Publish acknowledge

MQTT

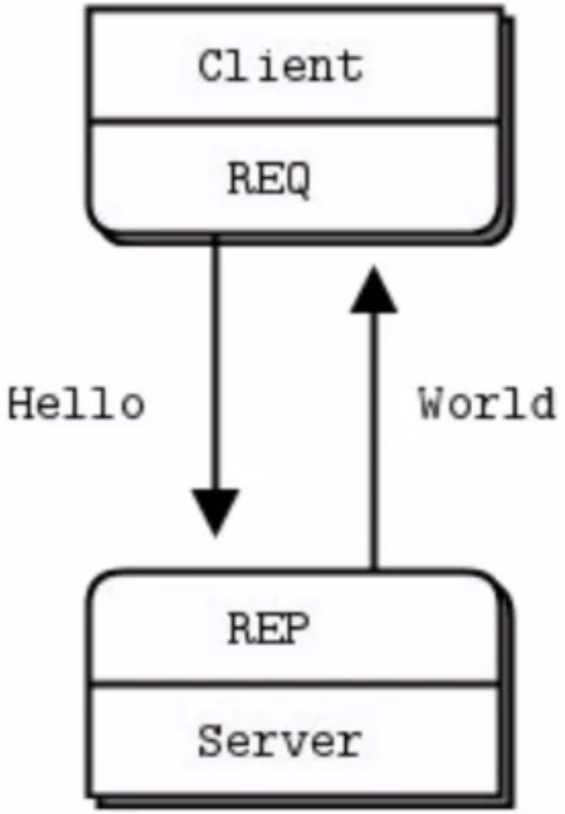
- Other possible control bytes:
  - ☑ Subscribe, Unsubscribe, ping request, disconnect, publish received, publish complete
- See for more info: http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/MQTT_V3.1_Protocol_Specific.pdf
-

Zero Message Queue (0MQ)

- Library available for Python, C, C#, java, Haskell, Erlang, Go, Ruby, Rust…. (40 programming languages).
- Send messages to heterogeneous systems: different processor types, Operating systems, multiple nodes
- Do I/O asynchronously, Concurrent ZeroMQ applications do not need locks or semaphores
- Connections come and go, ZeroMQ reconnects automatically (no “socket already in use” errrors)
- Queue messages until they can be delivered
- Does not impose any format on the messages

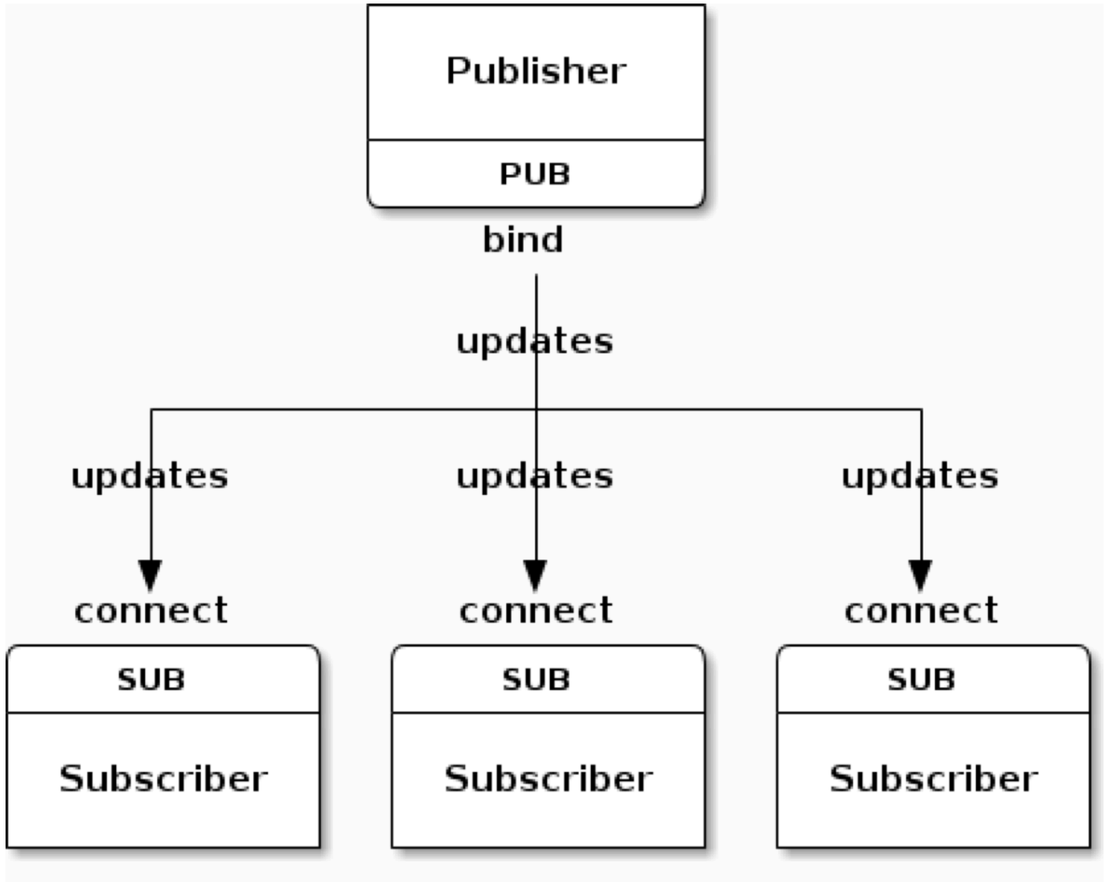
ZeroMQ - basic send - receive

- Synchronous send - receive of messages
- REQ: send then receive
- REP: receive then send
- Note that when sending commands to a system it is possible that a command will not arrive. It’s necessary to check arrival. A REQ - REP combination will take care of this



ZeroMQ Publish Subscribe model

- Publisher can accepts multiple clients (subscribers) which subscribe to “topics”
- The publisher (“broker”) can also receive information from clients with the data
- Important: the subscriber can subscrib to multiple topics
- Subscribers can also subscribe to multiple publishers



ZeroMQ Publish Subscribe model

- demo

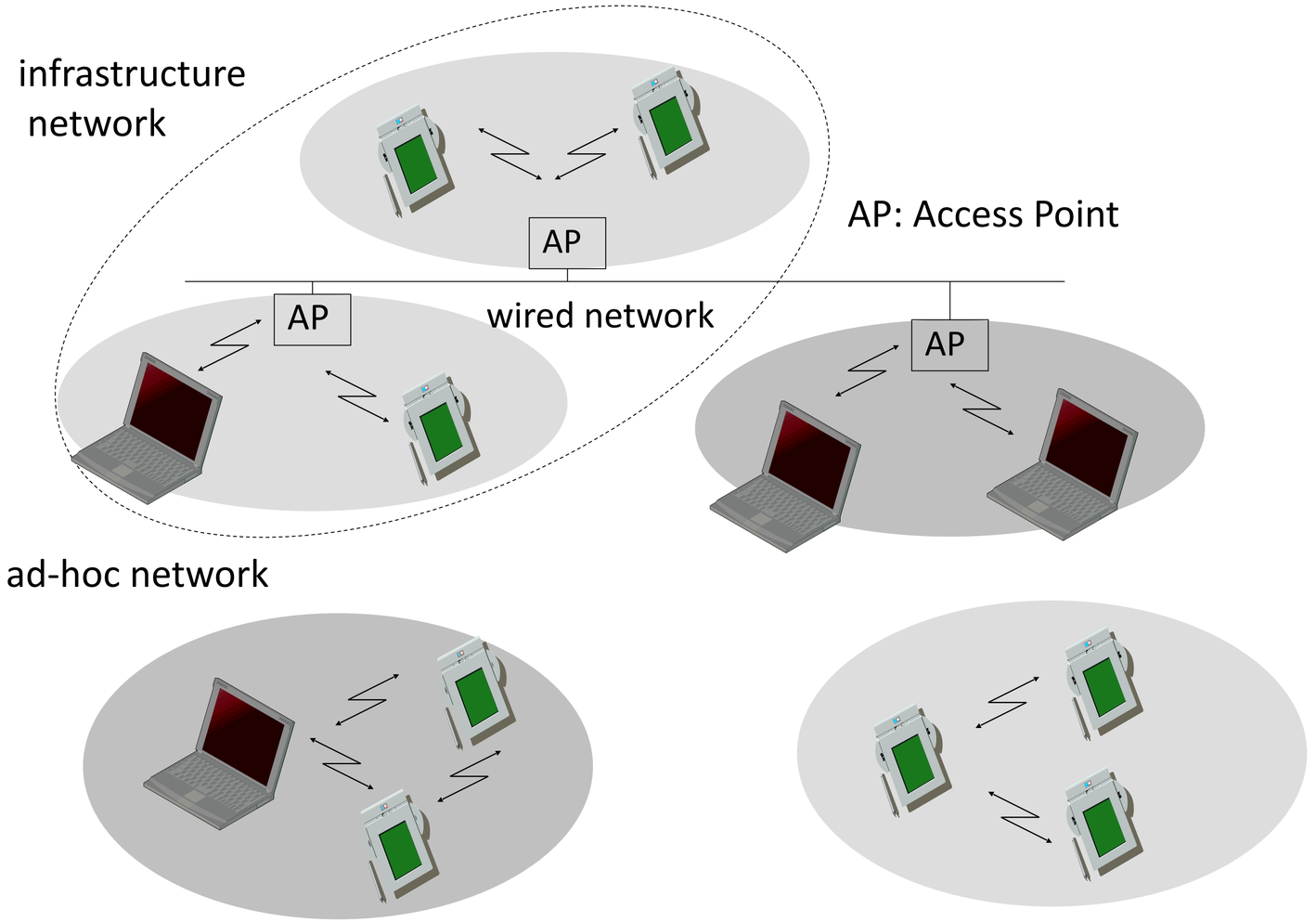
```
topics_pub.py tcp://localhost:5555
topics_sub.py tcp://localhost:5555 stocks
```

-

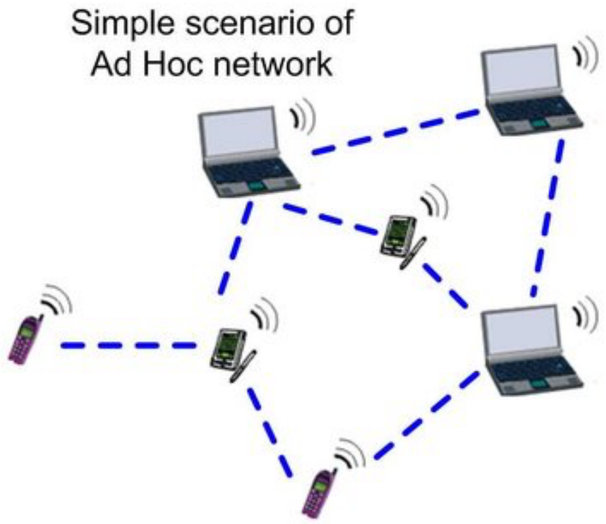
Outline

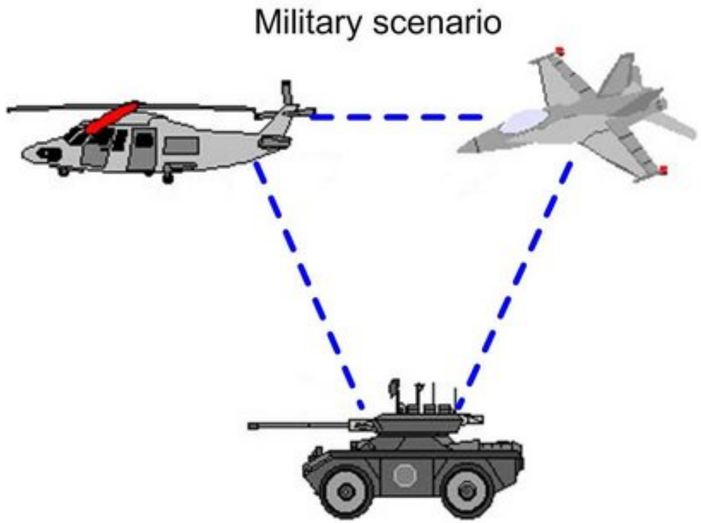
- Typical IoT communication standards
- BLE communication
- Asyncio in Python
- Publish Subscribe communication
  - ☑ MQTT
  - ☑ ZeroMQ
- IoT Manet Routing
  - ☑ DSR, AODV, Ant Colony routing

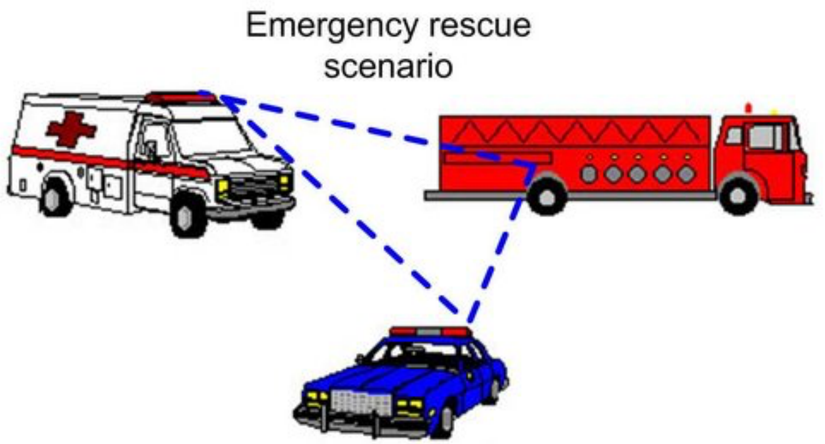
Ad hoc networks



MANETS: Mobile Ad Hoc Networks

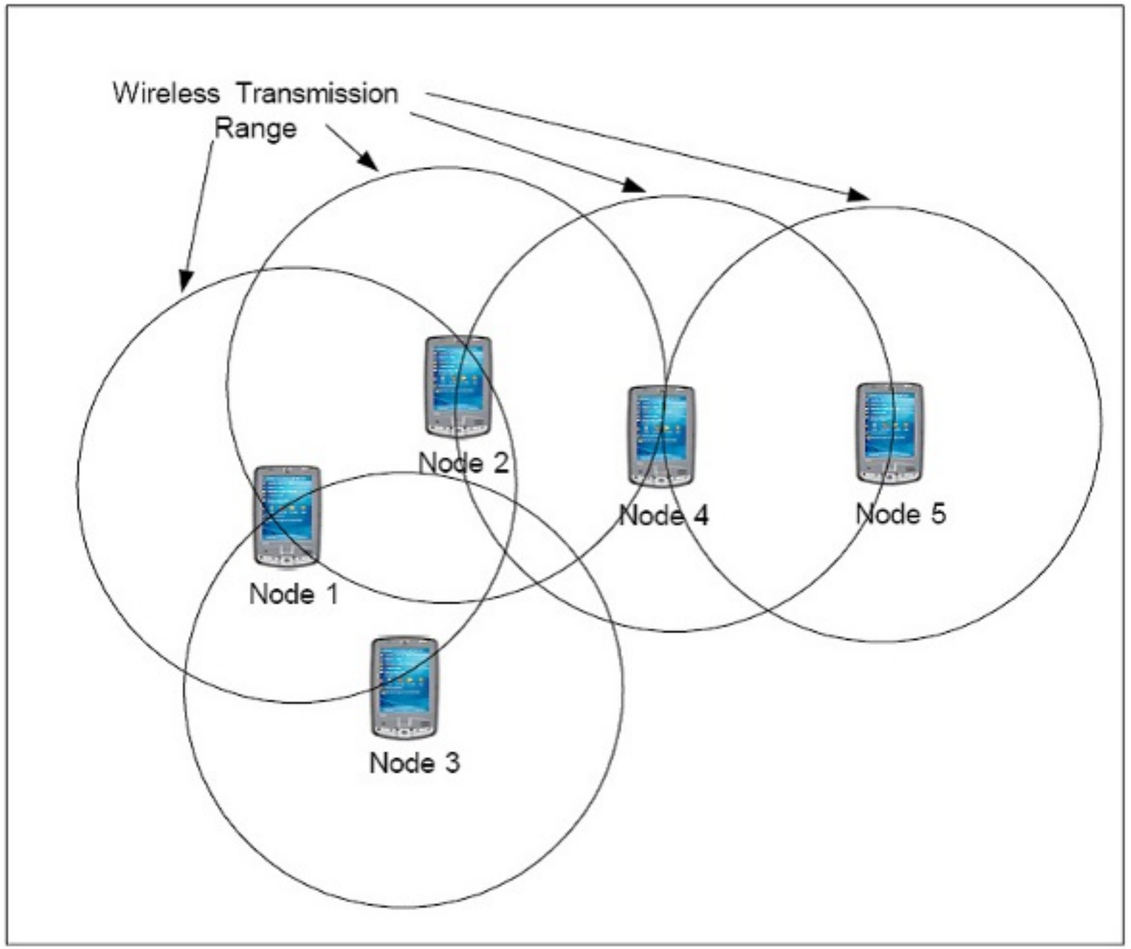






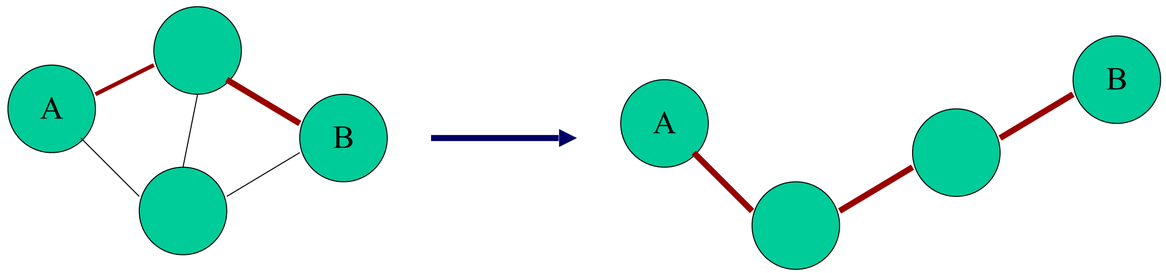
Ad Hoc networks - MANETS

- Ad hoc => “for this purpose” / can be changed In nearby future
- No need for routers / infrastructure
  - ☑ Only interconnected devices
- MANET: Mobile Adhoc NETwork
  - ☑ Routing can be done using “multihops” to reach remote devices which cannot be reached directly



Ad Hoc networks

- Dynamic topology (mobile network)
- Multi-hopping: we need to reach remote receivers to circumvent obstacles and to save energy
- Self organization: neighbour discovery, routing , clustering, localization
- Energy optimization
- Datarates: data congestion is common
- Security: often limited. Research topic
- Should still work when one node is not active due to error / power



Typical Ad hoc networks

- Emergency / disaster control (Rescue, Fire discovery)
- Military/Surveillance/Scientific UAV ad hoc networks
- Medical ad hoc networks
- Robot ad hoc networks
- “Corona app”

Definitions

- WANET: Wireless Adhoc NETwork: does not rely on pre-existing infrastructure (routers, access points). Data is forwarded to nodes based on neighbourhood discovery

- MANET: Mobile Adhoc NETwork: the same as WANET, but nodes **move around**

- Other Adhoc networks:

  - ☑ VANET: intelligent Vehicular NETwork. Adhoc communication between vehicles in order to prevent collisions and other accidents

  - ☑ FANET: Flying Adhoc NETwork: UAV flying objects communication in order to prevent collisions, detection of obstacles, intelligent behaviour

Manet

- Need dynamic routing
  - ☑ frequent topology changes possible
  - ☑ Very different from dynamic routing in internet
- Routing overhead must be kept minimal
  - ☑ wireless -> low bandwidth
  - ☑ mobile -> low power
  - ☑ Minimize numbers of routing control messages
  - ☑ Minimize routing state at each node

Outline

- Typical IoT communication standards
- BLE communication
- Asyncio in Python
- Publish Subscribe communication
  - ☑ MQTT
  - ☑ ZeroMQ
- IoT Manet Routing
  - ☑ DSR, AODV, Ant Colony routing

disclaimer: The NINA-W102 Wifi module

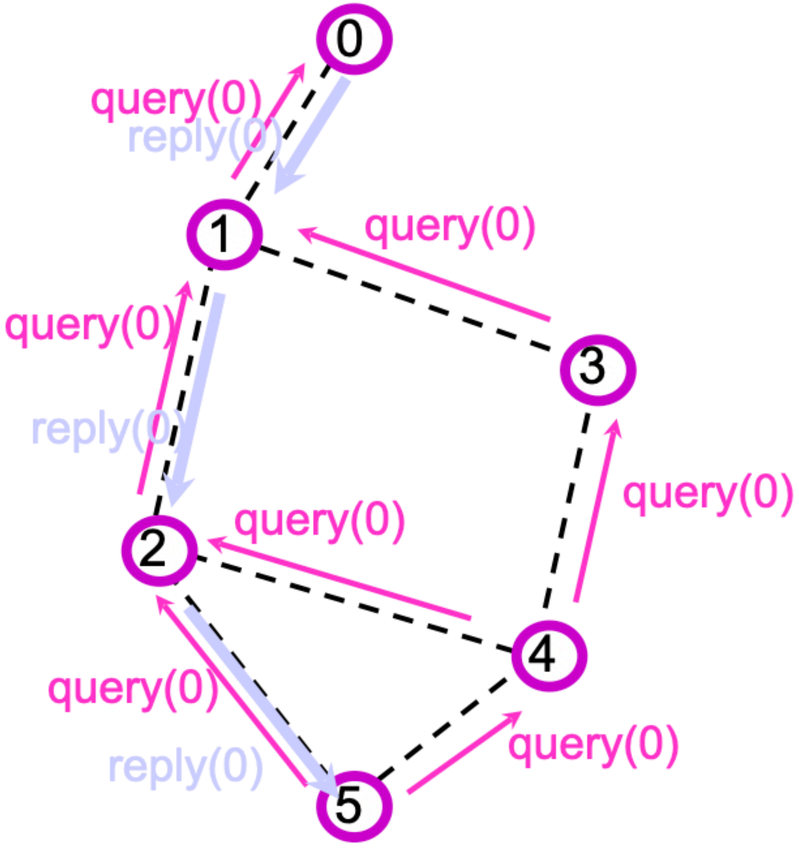
- Unfortunately (and quite stupid..) the Arduino IoT 33 board contains a Nina - W102 Wifi chip which does not support WiFi ad hoc or mesh mode (only station (STA) and access point (AP) mode)

- this makes it very difficult to setup an Ad-hoc system with this board.

- If you want to do some ad-hoc routing project use multiple different boards (ESP32, RPi etc). A limited number of ESP32 boards will be available for a project.

reactive (on demand) routing

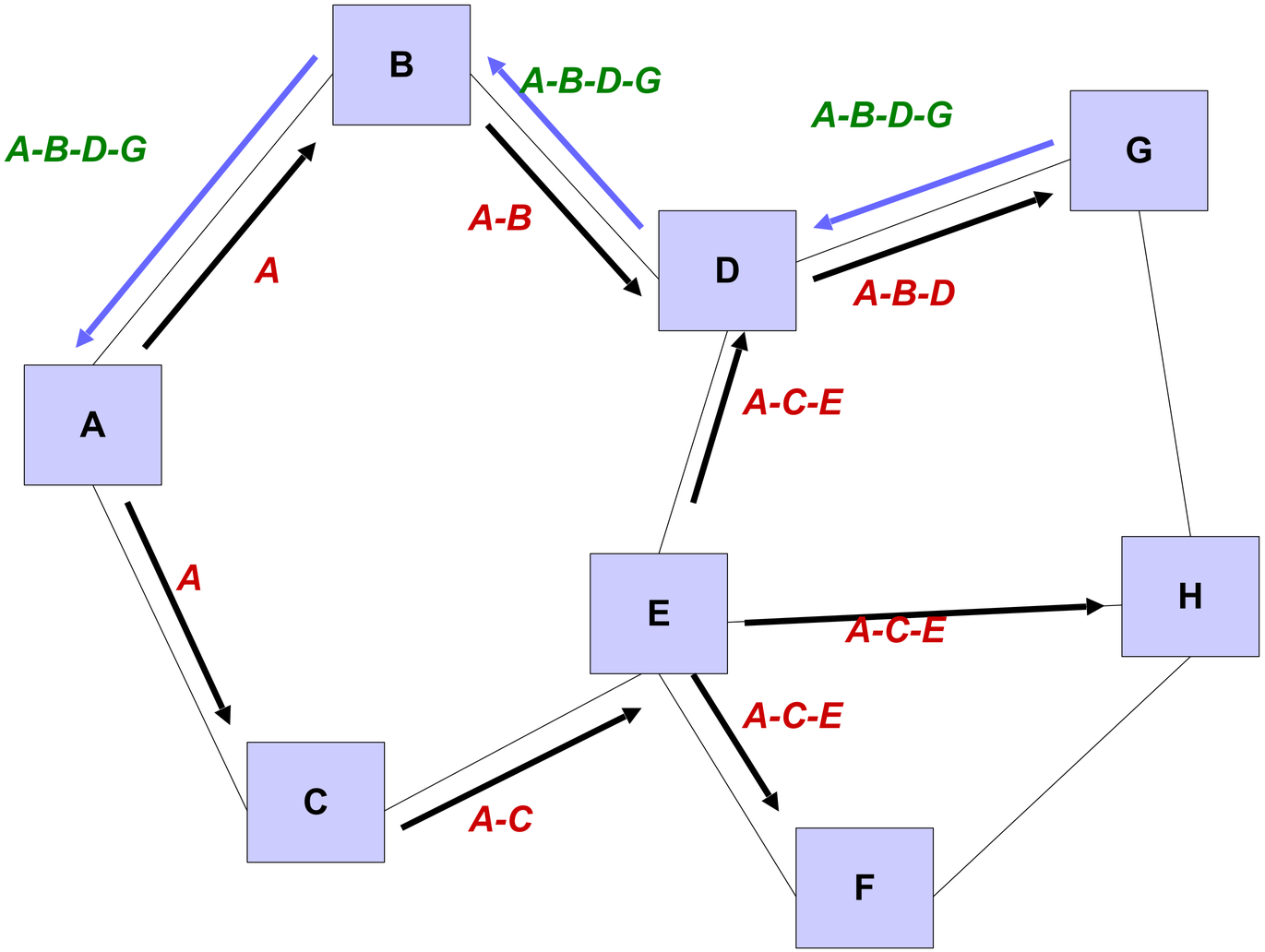
- Source floods the network with a **route request** when a route has to be found to a destination
  - ☑ flood is propagated outwards from source
  - ☑ Every node transmits the RREQ only once
- Destination replies with a RREP to request
  - ☑ reply uses the reverse path of the RREQ
  - ☑ the reply sets up the forward path
- 2 protocols use this method: DSR and AODV



Dynamic Source Routing (DSR)

- Only for relative small IoT networks of approx. 5-10 hops
- Detectable packet error
- Route reply packet contains the information for the forward route
  - ☑ or can be send by intermediate nodes to originator that have already a route to the destination
    - ▶ reverse the order of the route in the cache and send it as reply
- Each node maintains a route cache/route table with routes it has learned and overheard over time
- Source determines complete route to destination

DSR Route discovery



RREQ FORMAT

Initiator ID
Initiator seq#
Target ID
Partial route

A-B-C

Route Request (RREQ)

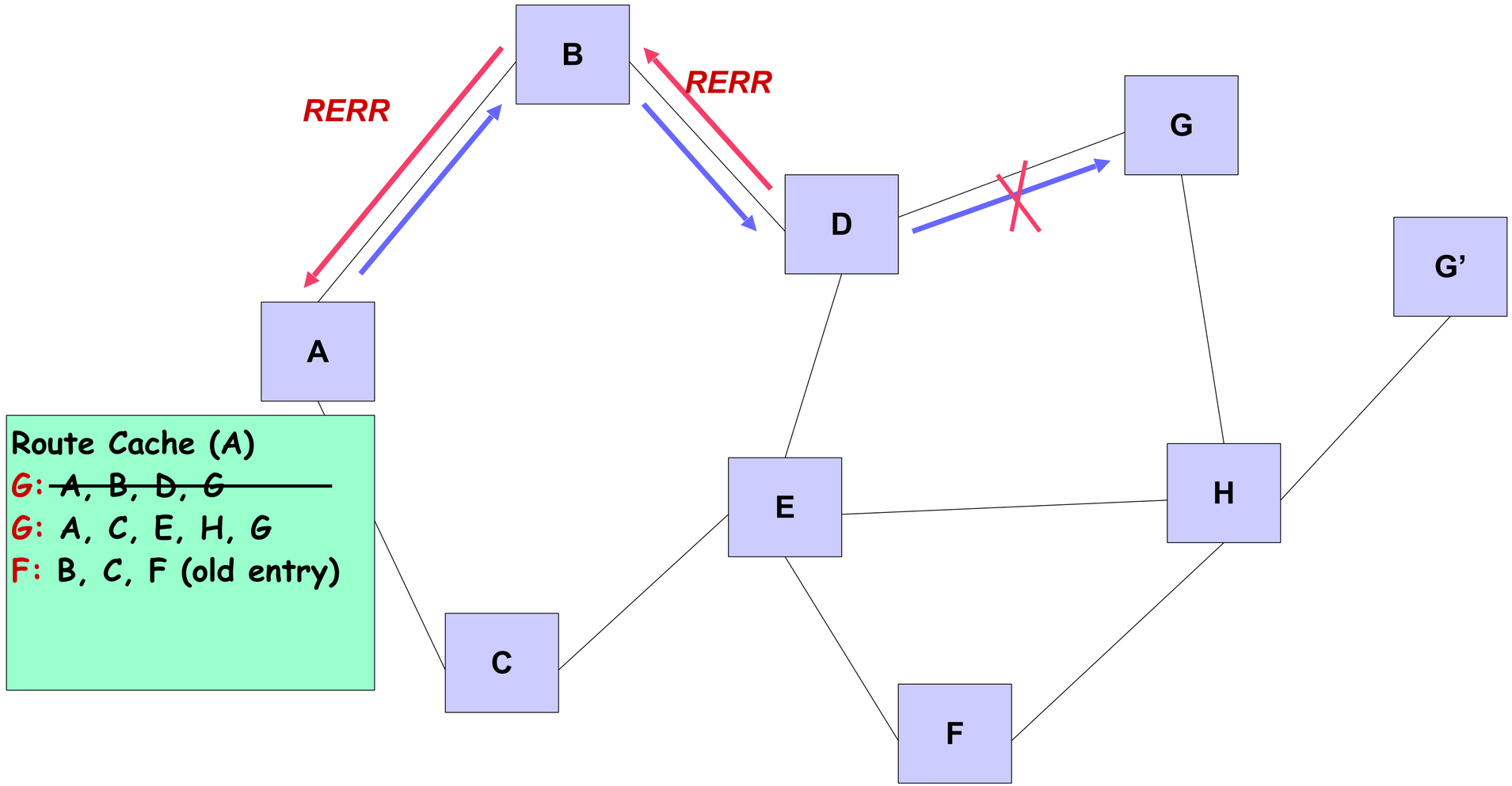
A-B-C

Route Reply (RREP)

DSR Route maintenance

- Route maintenance performed only while route is in use
- Error detection
  - ☑ monitors the validity of existing routes by passively listening to data packets transmitted at neighbouring nodes
- when problem detected send Route Error packet RERR
  - ☑ originating node should perform a new route discovery
  - ☑ host detects the RERR and sends back the RERR to sender of the packet - original src

DSR Route maintenance



DSR summary

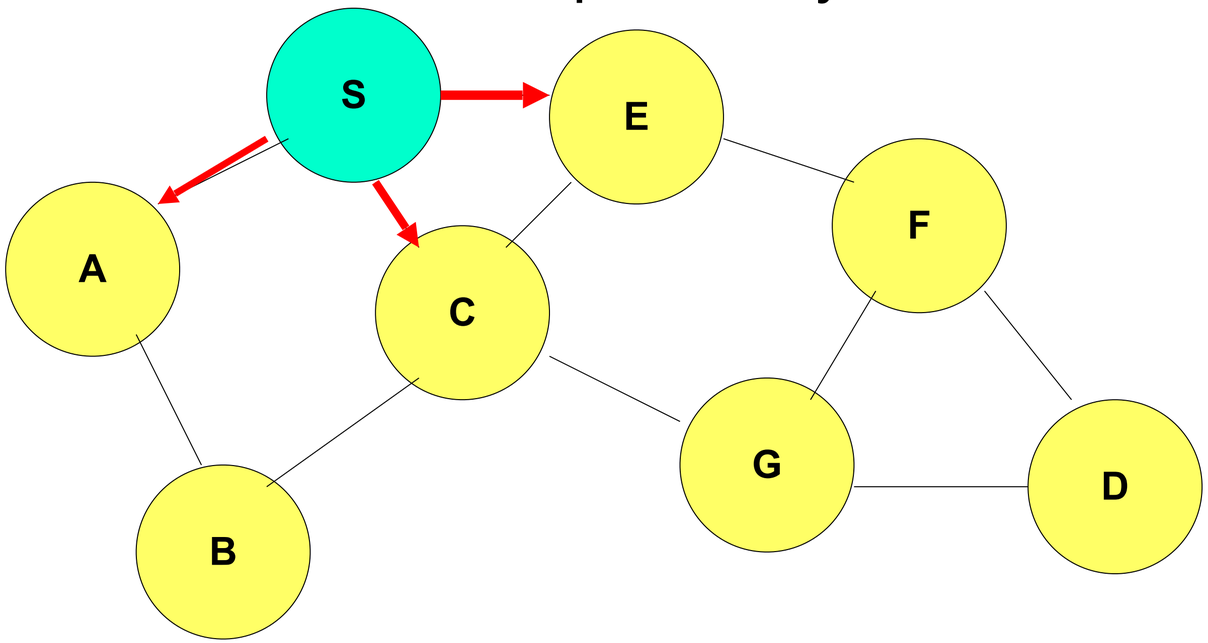
- Advantages
  - ☑ entirely on demand routing
- Disadvantages
  - ☑ high packet delays
  - ☑ space overhead in packets and routing tables: cache contains complete route to destination, not just to neighbour nodes.
    Data is sent with a header which contains complete route to destination
  - ☑ Could lead to routing loops

Outline

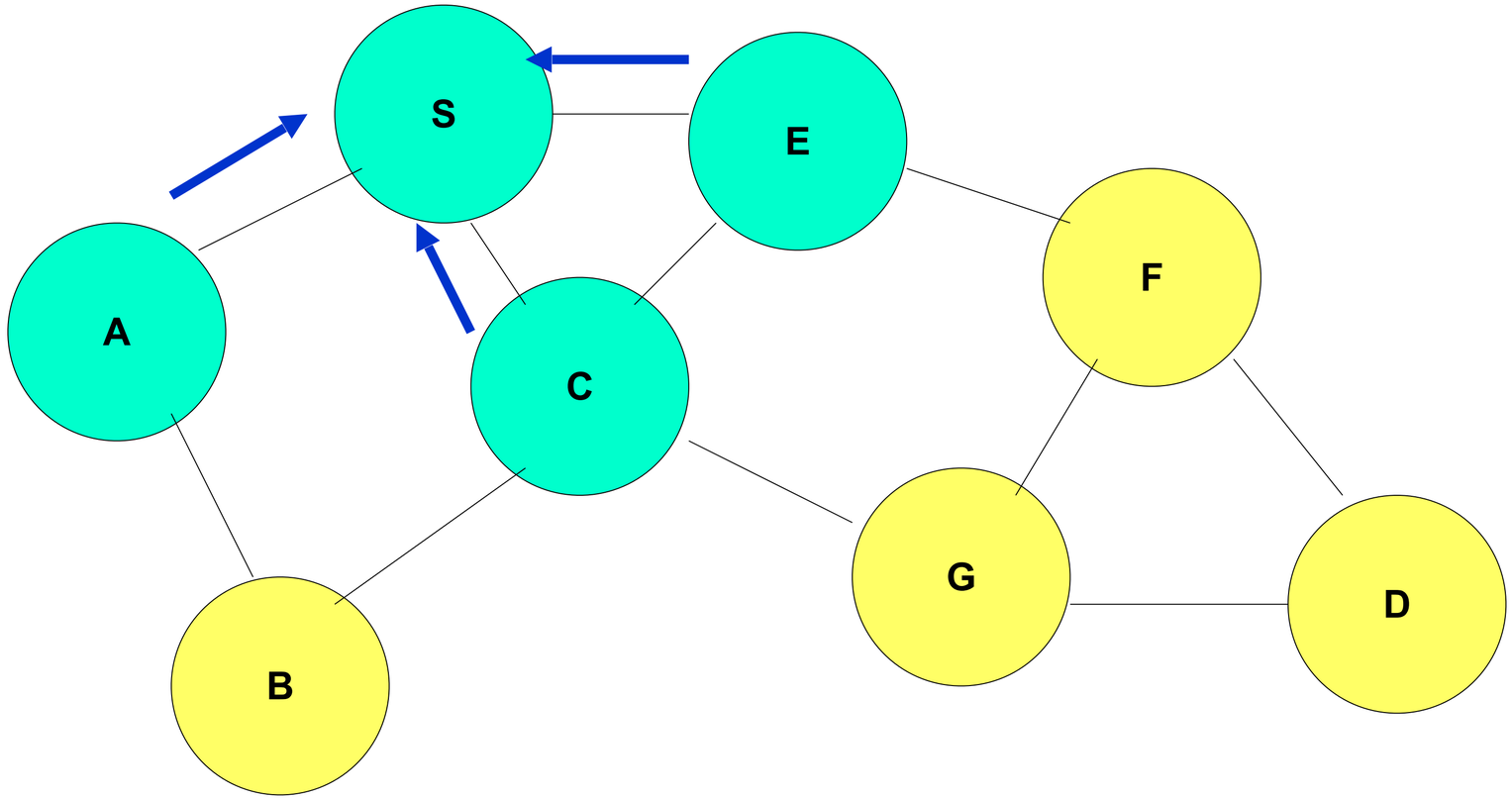
- Typical IoT communication standards
- BLE communication
- Asyncio in Python
- Publish Subscribe communication
  - ☑ MQTT
  - ☑ ZeroMQ
- IoT Manet Routing
  - ☑ DSR, AODV, Ant Colony routing

AODV routing

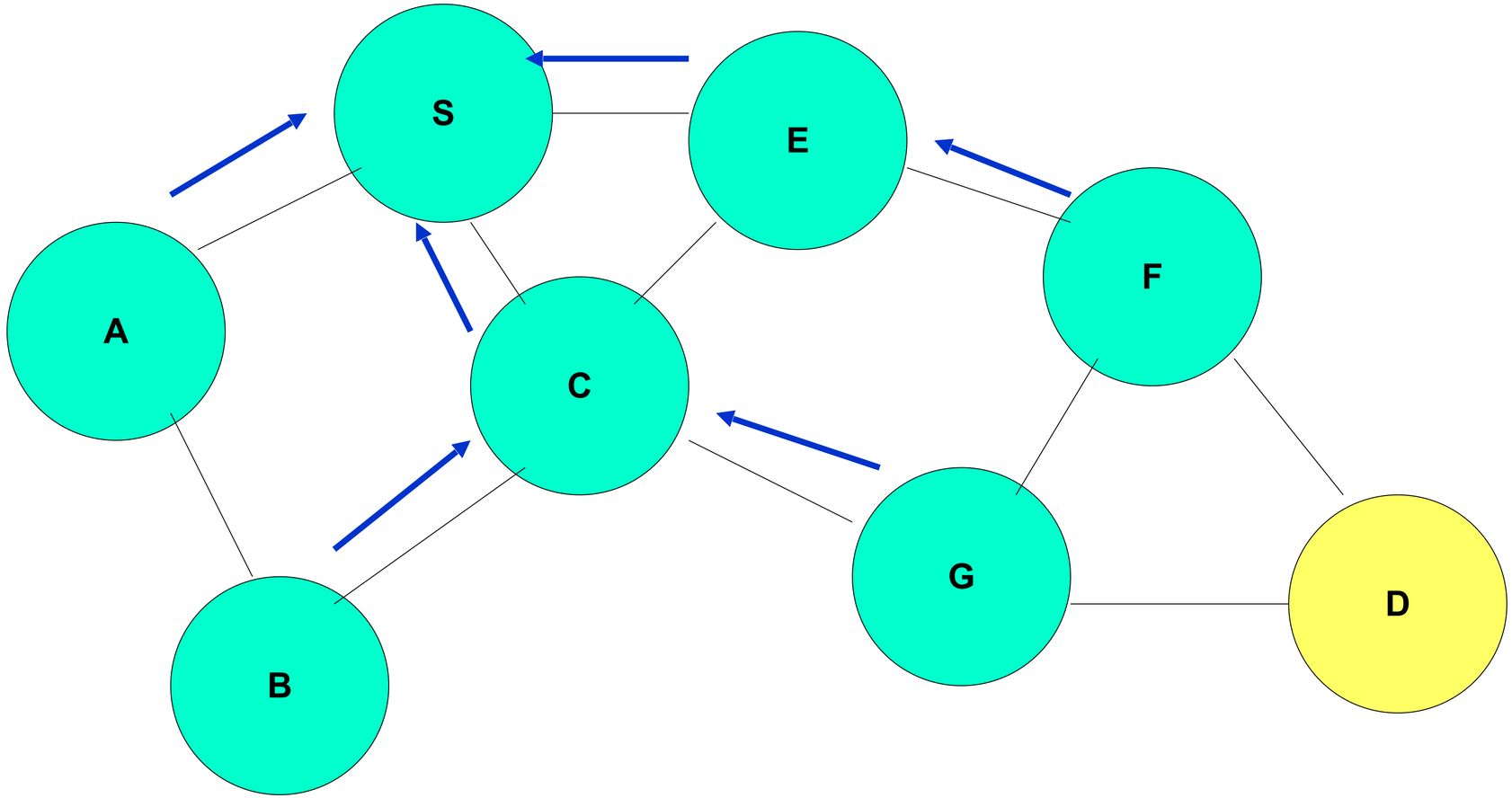
- Source transmits RREQ into the network to **neighbour** nodes with final destination in packet header
- Reverse paths are formed when a node hears a RREP
- Each node forwards the request only once



AODV route discovery

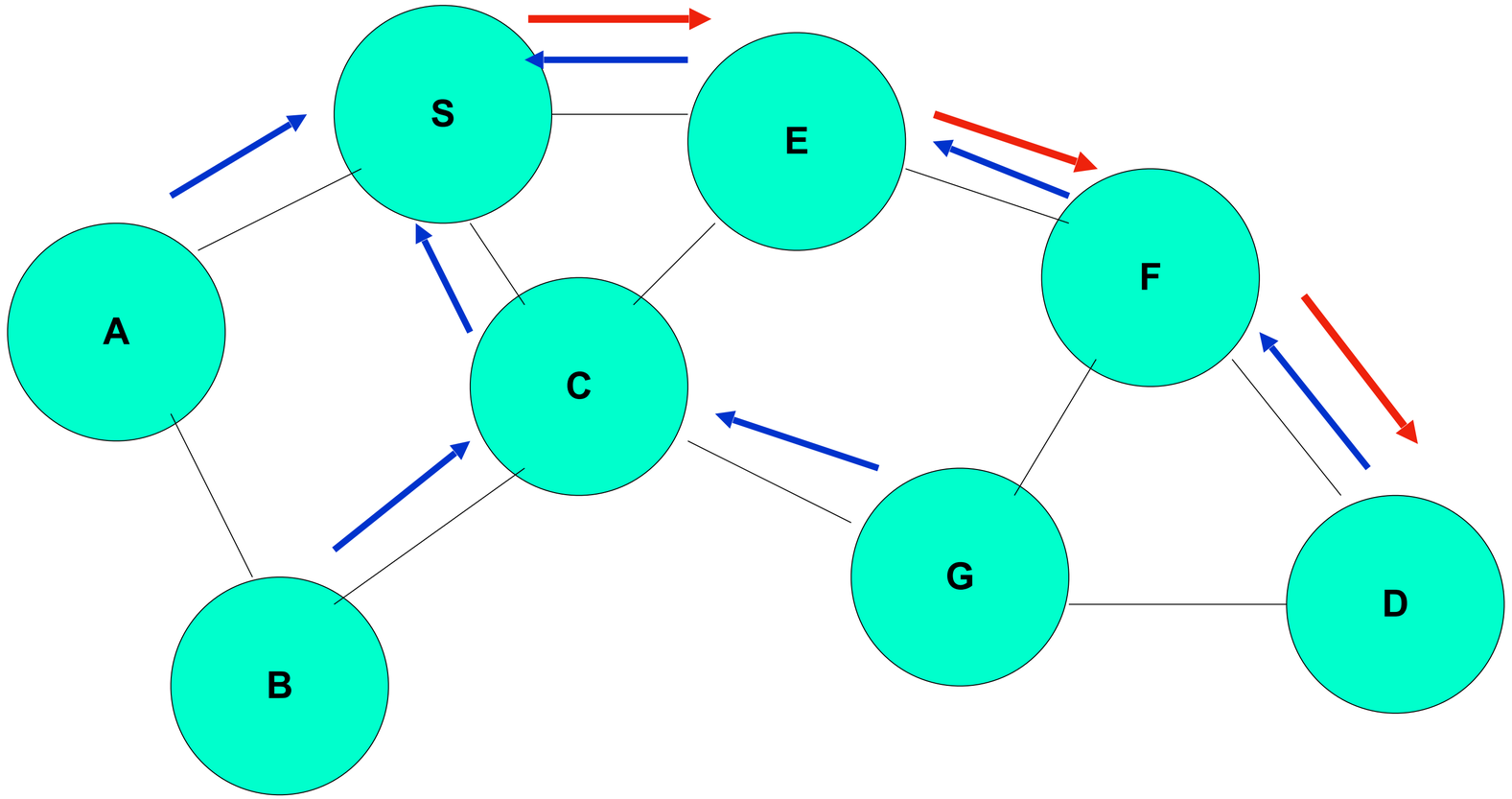


AODV route discovery



AODV route discovery

- RREQ has reached destination. RREP is used as final path to destination
  Intermediate nodes take notice of the RREP and update their routing tables

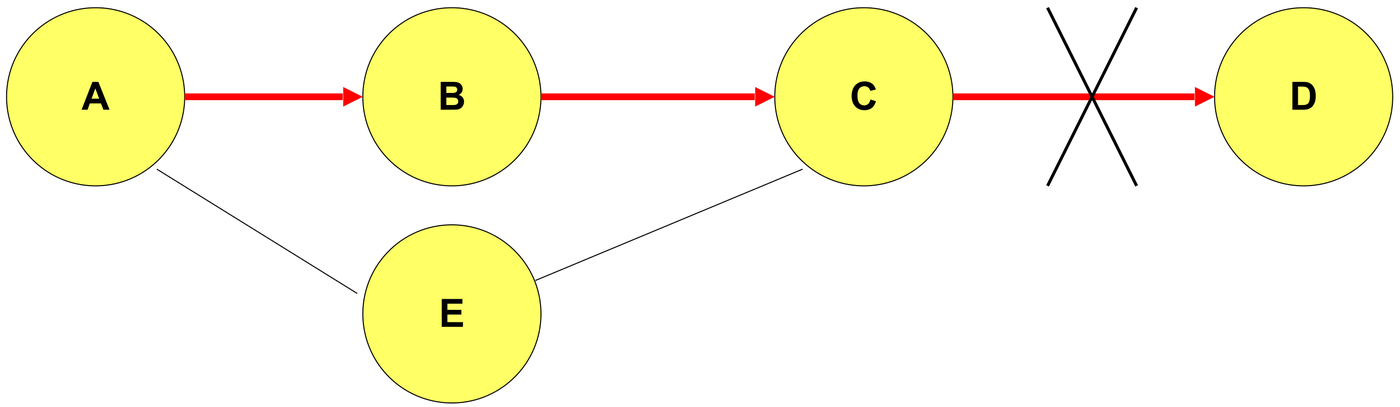


AODV optimization

- Big advantage in this model is that each node can reply to RREQ when it has an entry for the destination
  - ☑ faster operation
  - ☑ stops Route Request floods
- However this optimization can cause loop in case of link failures

AODV routing loops

- C-D fails and no RERR is received by A
- C performs a route discovery for D because the packet could not delivered to D with the old routing table
- A receives route request by route C-E-A
- A replies to C because it knows a route to D: A-B-C-D
- results in a loop: C-E-A-B-C

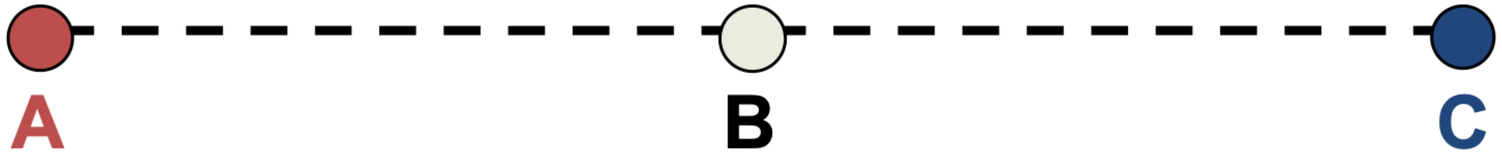


AODV -distance determination

- How to determine the distance between nodes?
  - ☑ GPS module in each node. Transmit the location to other nodes.
  - ☑ Update routing table according to received location of other nodes.
  - ☑ Transmission to other nodes using Bluetooth device addresses, BLE signal strength, Wifi (ip address based)
- if you do not have a GPS available you can also think of a grid based location of each node. Store the current location in the grid and update location when your node moves to a new location.

AODV with routing table example

- We can set up a routing table for every node. The nodes **broadcast** periodically their routing table to update their information on the current network



Dest.	Next	Metric
A	A	0
B	B	1
C	B	2

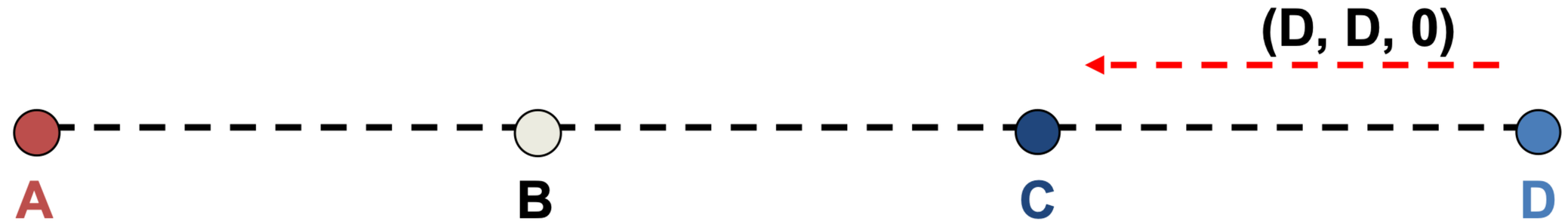
Dest.	Next	Metric
A	A	1
B	B	0
C	C	1

Dest.	Next	Metric
A	B	2
B	B	1
C	C	0

Source: Princeton university

AODV: Adding a new node

- C will update its routing table and broadcast it on the WLAN afterwards

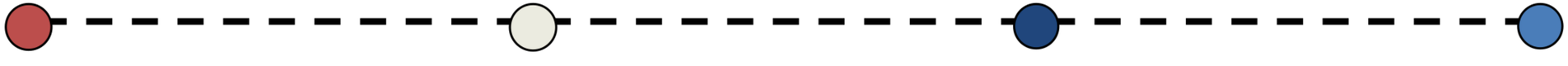


Dest.	Next	Metric
A	A	0
B	B	1
C	B	2

Dest.	Next	Metric
A	A	1
B	B	0
C	C	1

Dest.	Next	Metric
A	B	2
B	B	1
C	C	0
D	D	1

AODV: Updating routing table in B and A

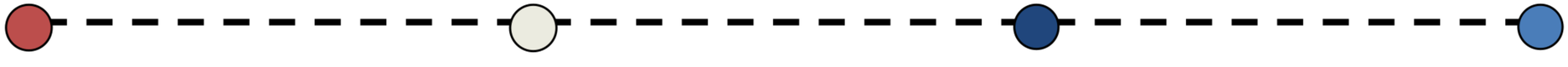


Dest.	Next	Metric
A	A	0
B	B	1
C	B	2

Dest.	Next	Metric
A	A	1
B	B	0
C	C	1
D	C	2

Dest.	Next	Metric
A	B	2
B	B	1
C	C	0
D	D	1

Dest.	Next	Metric
A	C	3
B	C	2
C	C	1
D	D	0



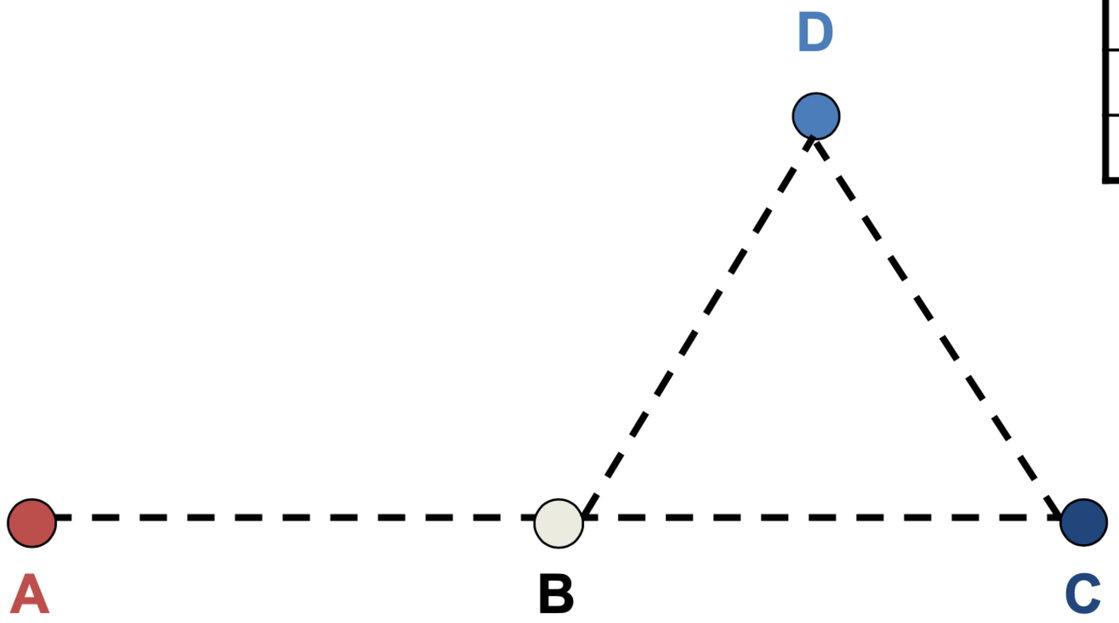
Dest.	Next	Metric
A	A	0
B	B	1
C	B	2
D	B	3

Dest.	Next	Metric
A	A	1
B	B	0
C	C	1
D	C	2

Dest.	Next	Metric
A	B	2
B	B	1
C	C	0
D	D	1

Dest.	Next	Metric
A	C	3
B	C	2
C	C	1
D	D	0

AODV- route changes



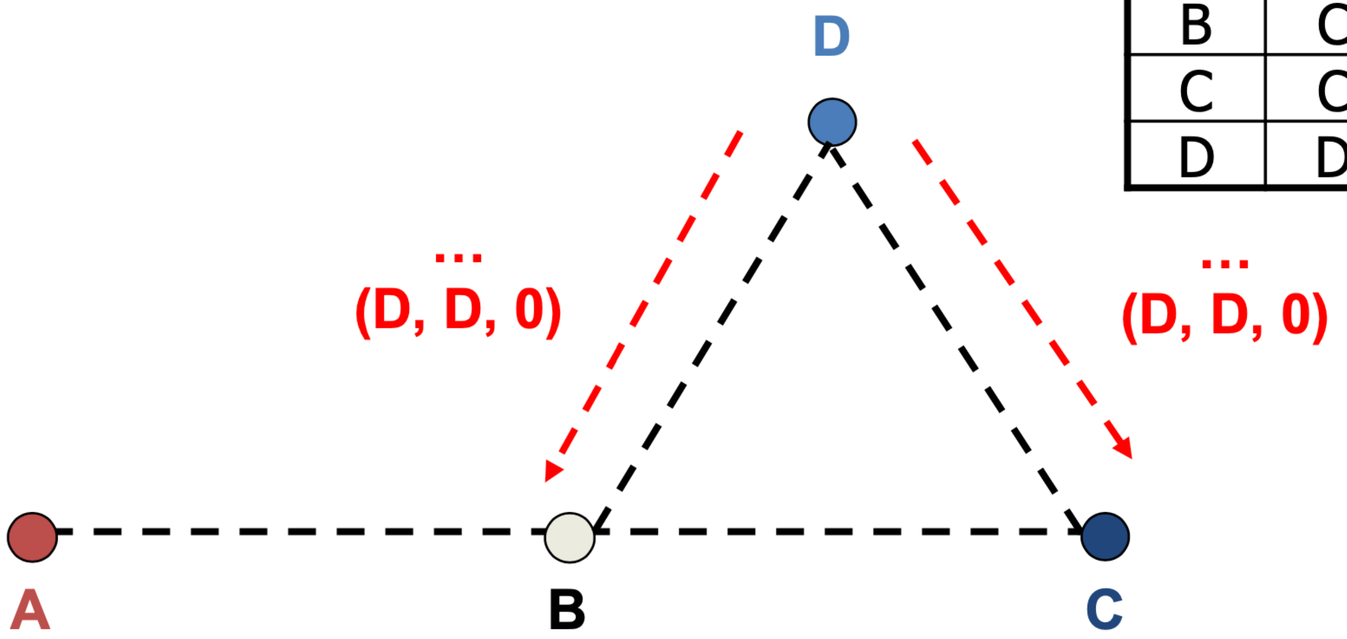
Dest.	Next	Metric
A	C	3
B	C	2
C	C	1
D	D	0

Dest.	Next	Metric
A	A	0
B	B	1
C	B	2
D	B	3

Dest.	Next	Metric
A	A	1
B	B	0
C	C	1
D	C	2

Dest.	Next	Metric
A	B	2
B	B	1
C	C	0
D	D	1

AODV - route changes



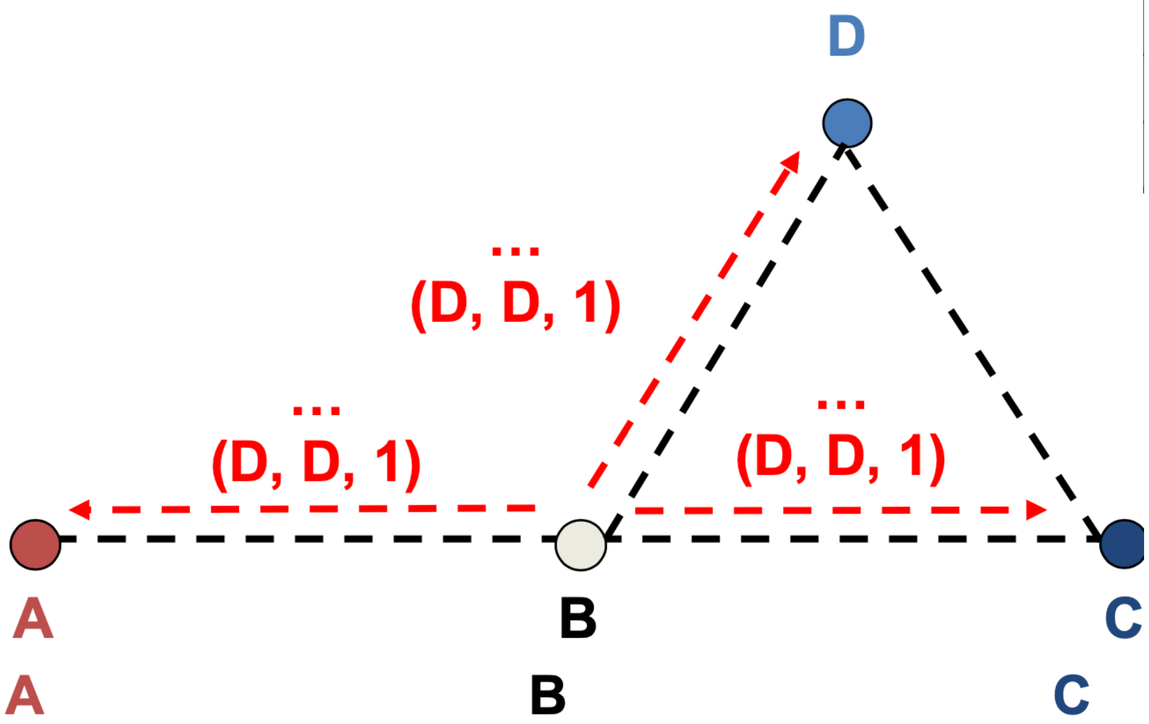
Dest.	Next	Metric
A	C	3
B	C	2
C	C	1
D	D	0

Dest.	Next	Metric
A	A	0
B	B	1
C	B	2
D	B	3

Dest.	Next	Metric
A	A	1
B	B	0
C	C	1
D	D	1

Dest.	Next	Metric
A	B	2
B	B	1
C	C	0
D	D	1

AODV - route changes



Dest.	Next	Metric
A	B	2
B	B	1
C	C	1
D	D	0

A

Dest.	Next	Metric
A	A	0
B	B	1
C	B	2
D	B	2

B

Dest.	Next	Metric
A	A	1
B	B	0
C	C	1
D	D	1

C

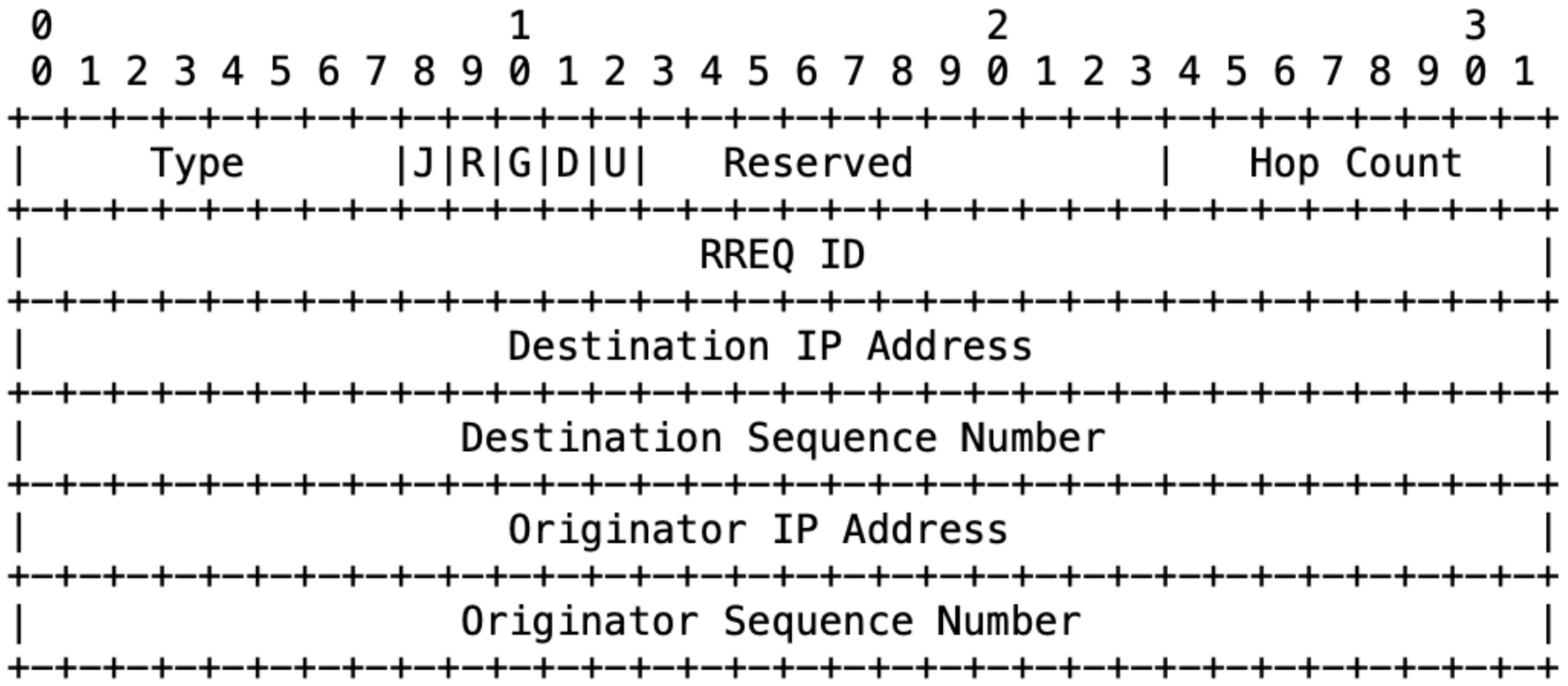
Dest.	Next	Metric
A	B	2
B	B	1
C	C	0
D	D	1

Sequence numbers in AODV

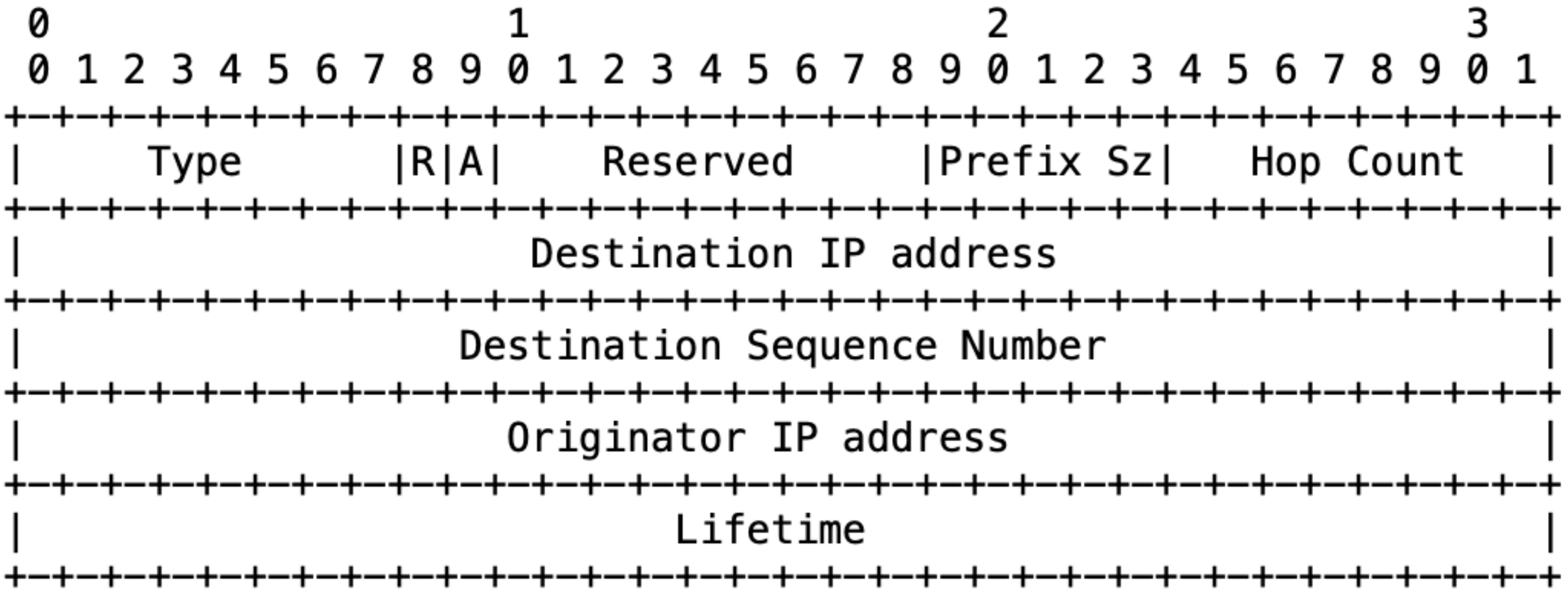
- A sequence number is normally added in AODV in order to have information how often a route has been used.
- Each node periodically advertises routes to other hosts
- The destination of the route adds “2” to the sequence number, others just propagate the routing table information
- Route selection: take the route with largest sequence number
- When a link is broken, node assigns infinity and only increments sequence number with 1 for destination (odd number)

RREQ packets header

5.1. Route Request (RREQ) Message Format



RREP packet header



AODV Sequenced

- Again 4 nodes. Node D is new. Route tables of A, B and C:

DEST	NEXT	HOPS	SEQ
A	A	0	110
B	B	1	104
C	B	2	600

DEST	NEXT	HOPS	SEQ
A	A	1	110
B	B	0	104
C	C	1	600

DEST	NEXT	HOPS	SEQ
A	B	2	110
B	B	1	104
C	C	0	600

AODV Sequenced

- Node D is added to routing table of C

DEST	NEXT	HOPS	SEQ
A	A	0	110
B	B	1	104
C	B	2	600

DEST	NEXT	HOPS	SEQ
A	A	1	110
B	B	0	104
C	C	1	600

DEST	NEXT	HOPS	SEQ
A	B	2	110
B	B	1	104
C	C	0	600
D	D	1	0

AODV Sequenced

- C broadcasts its routing table in order to reflect the changes due to the new node. B will update table. Sequence number of C is incremented with 2

DEST	NEXT	HOPS	SEQ
A	A	0	110
B	B	1	104
C	B	2	600

DEST	NEXT	HOPS	SEQ
A	A	1	110
B	B	0	104
C	C	1	602
D	C	2	0

DEST	NEXT	HOPS	SEQ
A	B	2	110
B	B	1	104
C	C	0	602
D	D	1	0

AODV Sequenced

- B will broadcast its routing table and increments seq. number with 2. A will update its table.

DEST	NEXT	HOPS	SEQ
A	A	0	110
B	B	1	106
C	B	2	602
D	B	3	0

DEST	NEXT	HOPS	SEQ
A	A	1	110
B	B	0	106
C	C	1	602
D	C	2	0

DEST	NEXT	HOPS	SEQ
A	B	2	110
B	B	1	106
C	C	0	602
D	D	1	0

AODV sequenced - detection of broken link

- D is not responding. C detects the broken link and updates its routing table Increment seq. number for D with 1 (odd number) and hops to infinity.

DEST	NEXT	HOPS	SEQ
A	A	0	110
B	B	1	106
C	B	2	602
D	B	3	0

DEST	NEXT	HOPS	SEQ
A	A	1	110
B	B	0	106
C	C	1	602
D	C	2	0

DEST	NEXT	HOPS	SEQ
A	B	2	110
B	B	1	104
C	C	0	602
D	D	Inf.	1

AODV sequenced - detection of broken link

- C broadcasts its table. A and B receive the update.
  The seq. number for D is not updated because seq. 1 is larger than 0 and it's clear that D cannot be reached.

DEST	NEXT	HOPS	SEQ
A	A	0	110
B	B	1	108
C	B	2	602
D	B	3	0

DEST	NEXT	HOPS	SEQ
A	A	1	110
B	B	0	108
C	C	1	602
D	C	2	0

DEST	NEXT	HOPS	SEQ
A	B	2	110
B	B	1	108
C	B	0	602
D	D	Inf.	1

AODV sequenced - change in network layout

- D moves to a different location.
  We will broadcast a “route advertisement”
  D sends a change it its routing table

DEST	NEXT	HOPS	SEQ
A	C	3	110
B	C	2	108
C	C	1	602
D	D	0	2

AODV sequenced - change in network layout

- C receives broadcast message and updates tables

DEST	NEXT	HOPS	SEQ
A	C	3	110
B	C	2	108
C	C	1	602
D	D	0	2

DEST	NEXT	HOPS	SEQ
A	A	0	110
B	B	1	108
C	B	2	602
D	B	3	0

DEST	NEXT	HOPS	SEQ
A	A	1	110
B	B	0	108
C	C	1	602
D	D	1	2

DEST	NEXT	HOPS	SEQ
A	B	2	110
B	B	1	108
C	B	0	602
D	D	1	2

AODV sequenced - change in network layout

- B receives broadcast message and updates tables

DEST	NEXT	HOPS	SEQ
A	A	0	110
B	B	1	110
C	B	2	602
D	B	2	2

DEST	NEXT	HOPS	SEQ
A	A	1	110
B	B	0	110
C	C	1	602
D	D	1	2

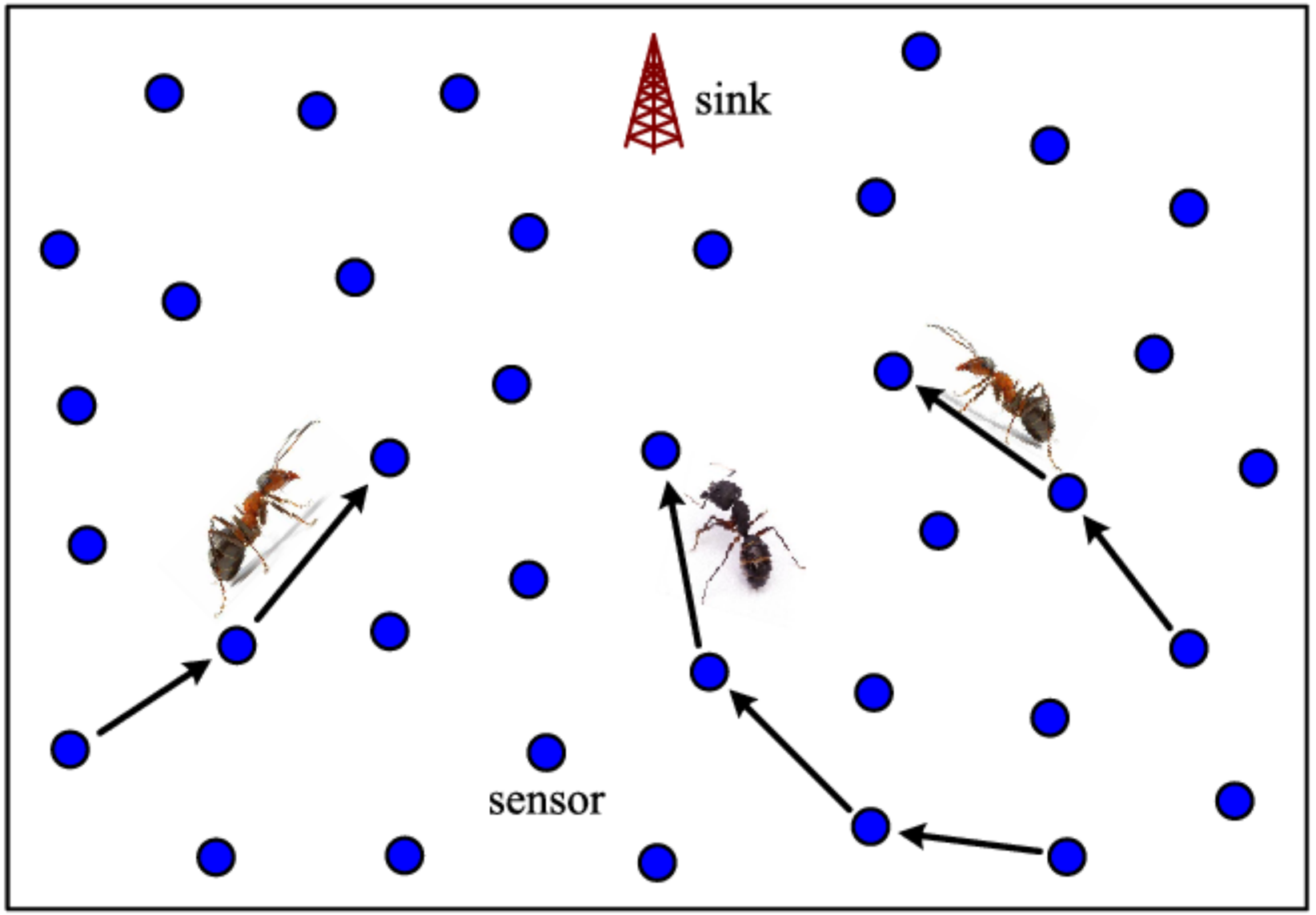
DEST	NEXT	HOPS	SEQ
A	B	2	110
B	B	1	110
C	B	0	602
D	D	1	2

DEST	NEXT	HOPS	SEQ
A	B	2	110
B	B	1	110
C	C	1	602
D	D	0	2

Outline

- Ad Hoc Mobile Networks
- Routing protocols
  - ☑ Dynamic source routing (DSR)
  - ☑ Ad hoc on demand distance vector routing (AODV)
  - ☑ Ant Colony routing
- Publish Subscribe communication
  - ☑ MQTT
  - ☑ ZeroMQ
- IoT Manet Routing
  - ☑ DSR, AODV, Ant Colony routing

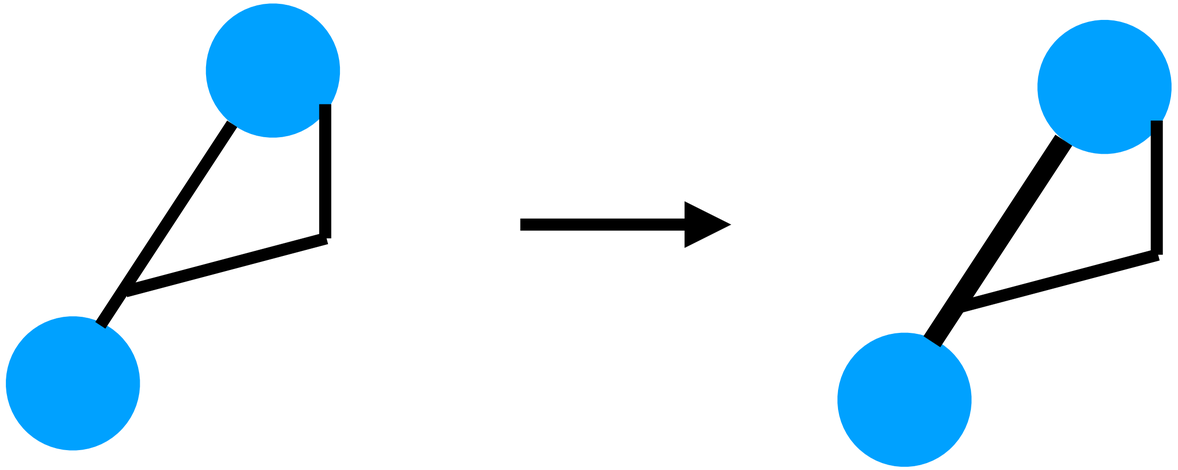
ANT colony routing



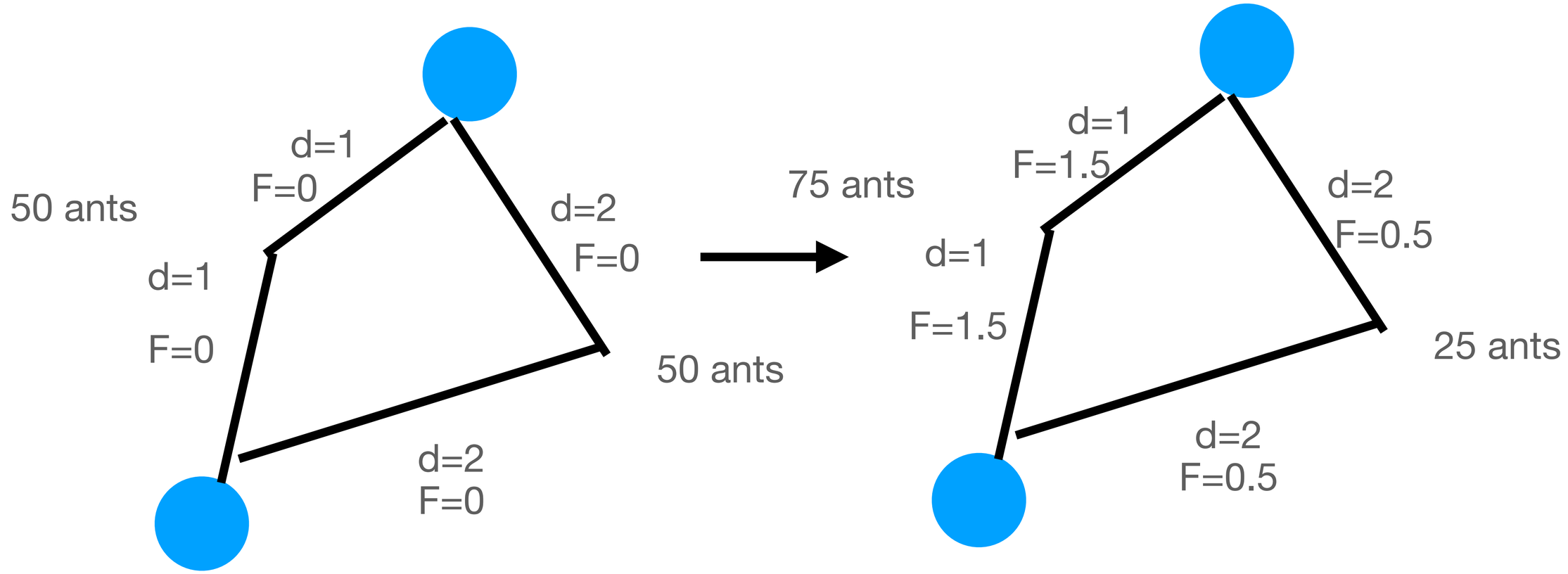
ANT colony routing

- Each ant randomly reaches a destination (the honey pot) leaving a pheromone trail. When returning to the source it travels backwards through the path constructed and updates the pheromone trail

- The trail which is used most often will get more pheromone and become more popular



ANT colony routing



F=amount of pheromone
d=distance

ANT colony routing

- Routing algorithm which is suitable for energy effectiveness in **low power IoT** ad hoc networks

- Also used in traffic optimization, travelling salesman Realtime task scheduling and many biological simulations, the computer science “knapsack” problem

Week 2 IoT “ik ben al klaar met alles”

- “ik ben al klaar met lab2. Kan ik nu naar huis?”
  - ☑ gefeliciteerd!
  - ☑ echter: je bent nooit klaar. Verdiep je in de stof in plaats van zaken zo snel mogelijk af te krijgen. studeren aan de UvA is geen wedstrijd.
  - ☑ Wat zijn die andere functies uit de BLE libraries? Wat doen de andere voorbeelden?
  - ☑ wat kun je allemaal met Zero message queue?
  - ☑ Probeer wat Wifi zaken uit (iedere Arduino zijn aangemeld in IoTRoam van de uva en hebben een password. Passwords staan op Canvas (onder modules)
  - ☑ Probeer de andere sensors uit die op het board zitten (gyro sensor). Welke sensors zijn er en hoe zouden die moeten werken met de Arduino?
- Maak assignment 2 af en zet alle code in Canvas
- Zorg dat beide assignments een cijfer krijgen door langs te gaan (het nakijken gebeurt dus niet automatisch!!!)

Week 2 IoT “ik ben al klaar met alles”

- volgende week meer informatie over de projecten bij het hoorcollege op donderdag!!

- Bedenk met wie je wil samenwerken in een project. Wees flexibel voor verschillende mensen die nog een projectgroep zoeken

- Bedenk met elkaar een onderwerp en maak een beknopte beschrijving

  - ☑ onderwerp/titel, wie, wat is het doel, wat is er voor nodig.

- Oefen eventueel nog wat met git **in een groepje**.

  - ☑ dus ook branching, merging, checkout/pull van vorige locale versie, tagging van een versie etc. etc. Probeer alles te doen met de commandline….

Week 2 IoT “ik ben al klaar met alles”

- Maak niet de blunder om al met allerlei code te beginnen (het zogenaamde “cowboy computing”

- **niet** projectmatig werken (cowboy computing/ hobby manier van aanpak) betekent dat je cijfer omlaag gaat.

- Het IoT project voeren we uit op een manier zoals dat ook in het bedrijfsleven plaatsvindt.

  ☑ dus niet als “hobby” project.