SAPIENZA
UNIVERSITÀ DI ROMA

"Sapienza" University of Rome
Faculty of Information Engineering,
Informatics and Statistics
Department of Computer Science

Autonomous Networking

Lecture notes integrated with the book "Reinforcement Learning: An Introduction", R. S. Sutton, A. G. Barto

Author
Simone Bianco

June 29, 2025

Contents

Information and Contacts

Personal notes and summaries collected as part of the *Autonomous Networking* course offered by the degree in Computer Science of the University of Rome "La Sapienza".

Further information and notes can be found at the following link:
https://github.com/Exyss/university-notes. Anyone can feel free to report inaccuracies, improvements or requests through the Issue system provided by GitHub itself or by contacting the author privately:

- Email: **bianco.simone@outlook.it**
- LinkedIn: **Simone Bianco**

The notes are constantly being updated, so please check if the changes have already been made in the most recent version.

Suggested prerequisites:

Sufficient knowledge of computer networks, algorithms and probability

Licence:

1

Autonomous wireless systems

1.1 Autonomous networks

With the exponential growth in the use of electronic devices, the emplyment of autonomous networks naturally increased. These type of networks revolve around four main characteristics:

- **Self-governance**: they don't require human interventions, making independent decision based on predefined rules and real-time data
- **Self-sufficiency**: they exist and function as an independent organism capable of managing its own resources and processes
- **Embedded intelligence**: they use advanced algorithms and artificial intelligence to interpret envitonmental inputs, learn from past experiences and adapt to changing conditions
- **Practive response**: they continuously monitor the environment, anticipate changes and react autonomously to maintain optimal performance and reliability

To achieve autonomous networks, *wireless technologies* are preferred due to their natural independence between each other: if there is no cable connecting two devices then the two devices can freely interact with the environment.

The main examples of such wireless technologies involve *Radio Frequency Identification (RFID)* devices, *sensor networks* and *smart devices*.

While designing autonomous networks, we address *autonomy* mainly at the networking level (communication and routing), but also in mobility. In modern days, the best answer for such task it **reinforcement learning**, which can be summarized as all the set of techniques that can be used to determine how an intelligent agent can learn to make a good sequence of decisions. In this chapter we'll discuss the main wireless technologies that can be used in autonomous networks. In the following one, we'll discuss how reinforcement learning can be applied to such technologies in order to solve a networking problem.

1.2 Radio Frequency Identification (RFID)

Radio Frequency Identification (RFID) is a technology that uses radio frequencies in order to uniquely identify and track objects, people and animals. RFID communications are based on three components:

- **RFID tag** (or *transponder*): devices that contain a chip with an embedded unique identifier (they may also be provided with some memory chips). The chip may be *passive* (no battery required) or *active* (battery required)
- **RFID reader** (or *interrogator*): devices that send radio signals in order to activate RFID tags and read their informations
- **Managing system**: a server that handles the data received by the readers

Tags are typically implemented through very small chips, while readers consist of more powerful devices provided with antennas. RFID object identification has many applicaations, from inventory and logistics to domots and assisted living.

There are two main types of RFID communication: reader-to-tags and tags-to-reader. In these communications, passive tags reflect the high-power constant signal generated by the reader, embedding their ID and info inside the reflected signal (**backscattering**), typically containing 96 bits of information (max 256 bits).

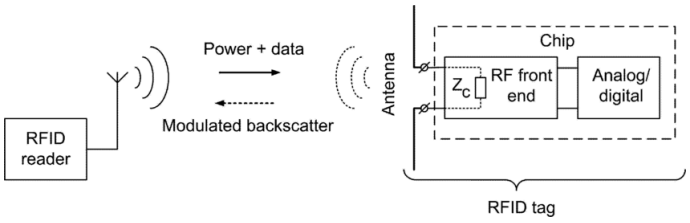


Figure 1.1: Description of RFID communication.

The RFID communication channel is, by definition, shared among all devices. In fact, the system is very susceptible to **collisions**: if multiple tags are reached by the signal of the same reader, they'll simultaneously reply. Collisions aren't the only issue in RFID communications. In fact, by construction of the system itself it is clear that tags outnumber readers with a large margin. This raises the problem of identification among tags with the least amount of data needed. Moreover, tags cannot ear each other since they have no *carrirer sense* and no *collision detection*, implying that the reader is in charge of managing the **channel access** all by itself.

Several **Media Access Control (MAC)** protocols have been proposed to identify tags in a RFID system. Sequential MAC protocols for RFID systems aim at distinguishing tag transmissions through a specific serial number (**singulation**). There are two main types of sequential protocols: *tree based protocols* and *aloha based protocols*.

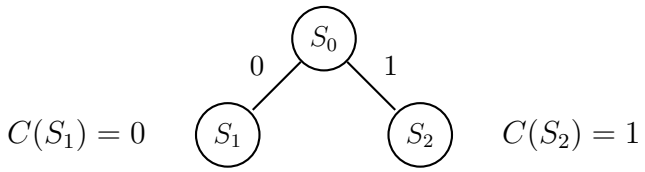
1.2.1 Tree based MAC protocols

The first tree based MAC protocol for RFID communications that we'll discuss is the **Binary Splitting (BS)** protocol. The BS protocol recursively splits answering tags into two subgroups until it obtains single-tag groups. Tags answer to reader's queries according to the generation of a binary random number.

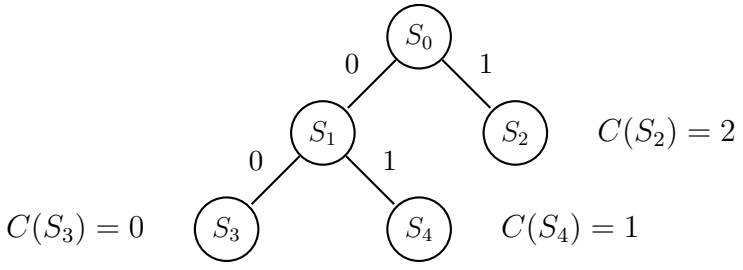
Suppose that we have a set of tags to identify. Each tag has a counter initially set to zero. Each time the counter of a tag is set to 0, it will reply to the reader's query.

1. The reader sends a query
2. If more than one tag replies (collision), each replying tag generates a random bit and sums it to the counter, while each non-replying tag increases its counter by 1.
3. If at most one tag replies (no collision), all tags decrement their counter by 1 (the minimum value for the counter is 0).
4. The process is repeated from step 1 until all tags are identified

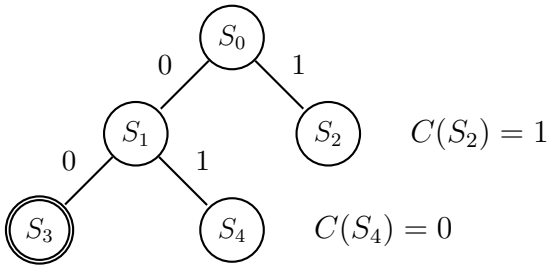
Consider the following example. Suppose that we have a set S_0 of 8 tags to identify. Initially, the counter of each tag is set to 0, hence they all reply. After the collision, some tags will have the counter set to 0 (subgroup S_1), while the others will have the counter set to 1 (subgroup S_2).



On the second iteration, tags in group S_2 will reply the next reader's query, forming another split into two sets S_3 (counter set to 0) and S_4 (counter set to 1), while the counter of the tags in the set S_2 gets set to 2.



Suppose now that in the third iteration S_3 contains only one tag. Since the will be no collision, this unique tag will be correctly identified by the reader. Then, the counters of S_4 and S_2 get decreased by 1 since a tag has been identified.



The fourth iteration then proceeds on S_4 and will eventually proceed also on S_2. In general, the binary splitting protocol can be viewed as a depth-first search that prioritizes tags whose counter is currently set to 0. Binary splitting works good on average. However, the use of random counter increments may yield cases with a non optimal split (i.e. most of the tags may fall in the same set), requiring multiple splits.

A more simple protocol based on this idea is the **Query Tree (QT)** protocol. Here, tags are queried according the the binary structure of their IDs (recall that each tag typically has an ID of 96 bits):

1. The reader queries all tags by sending a binary string. Only the tags whose ID have a prefix matching the string answer the query
2. The string is initially set to the empty string ε.
3. If more than one tag replies (collision), the reader recursively makes two queries. The first query extends the string with a 0, while the second extends the string with a 1.
4. If at most one tag replies (no collision), the corresponding branch of the protocol gets killed.

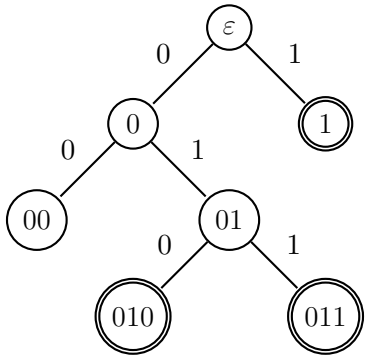


Figure 1.2: Tree generated by the QT protocol for the IDs $0100, 0111$ and 1010

In case of uniform distribution, the tree induced by BT and QT are almost the same due to each split having the same expected size. To measure performance of RFID protocols, we want to determine how fast a protocol is to collect all tags ID. If we have n tags, the protocoll will end when all n tags have responded singularly. The **system's efficiency** is measured as the ration between the number of tags $n$and the number of queries q.

$$\mathrm{SE} = \frac{n}{q}$$

When $n = q$, the protocol is clearly optimal. However, in practice the SE value is far below 1. For instance, consider the BS protocol. To evaluate $\mathrm{SE}_{\mathrm{BS}}$, we need to estimate the total number of queries for n tags, denoted with $\mathrm{BS}_{\mathrm{tot}}(n)$. We observe that on each query the tag set is split into two sets, one with k elements and one with $n-k$ elements, giving the following recursive equation.

$$\mathrm{BS}_{\mathrm{tot}}(n) = \begin{cases} 1 + \sum_{k=0}^{n} \binom{n}{k} \left(\frac{1}{2}\right)^k \left(1 - \frac{1}{2}\right)^{n-k} \left(\mathrm{BS}_{\mathrm{tot}}(k) + \mathrm{BS}_{\mathrm{tot}}(n-k)\right) & \text{if } n > 1 \\ 1 & \text{otherwise} \end{cases}$$

For large values of n we get that:

$$\lim_{n \to +\infty} \mathrm{SE}_{\mathrm{BS}} = \lim_{n \to +\infty} \frac{n}{\mathrm{BS}_{\mathrm{tot}}(n)} \approx 0.38$$

1.2.2 Aloha based MAC protocols

In Aloha based protocols, time is *slotted*. On each slot, at most one tag can communicate (slot duration is equal to the tag's ID transmission time). Slots are grouped into frames, where each tag randomly picks a slot.

In **Framed Slotted Aloha (FSA)**, when the reader isses a start of a frame, it includes the number of N slots in such frame. The n tags randomly pick a slot of the frame. If a collision occurs, i.e. two tags try to communicate in the same slot, the process repeats is reapplied on all the tags that have collided, until all tags are identified.

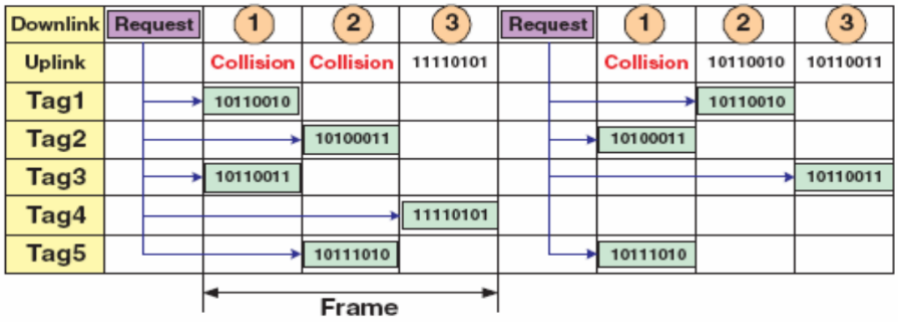


Figure 1.3: The FSA protocol in action (6 total slots with 3 collisions and 3 identifications).

In general, the FSA protocol reaches the best performance when the number of slots in a frame is equal to the number of tags to be identified, i.e. when $N = n$, achieving a 37% of identification slots and 63% of collision slots.

In particular, the probability that r out of n tags answer in the same slot among N slots is given by the binomial distribution:

$$\Pr[r \text{ tags in same slot}] = \binom{n}{r} \left(\frac{1}{N}\right)^r \left(1 - \frac{1}{N}\right)^{n-r}$$

implying that the number $s(r)$ of slots with exactly r tags is expected to be:

$$s(r) = N\binom{n}{r}\left(\frac{1}{N}\right)^r\left(1-\frac{1}{N}\right)^{n-r}$$

Let $R_{\text{idle}}, R_{\text{id}}, R_{\text{col}}$ be the number of identification, collision and idle rounds during the tag identification process. It's easy to see that:

$$R_{\text{idle}} = N\left(1-\frac{1}{N}\right)^n \quad R_{\text{id}} = n\left(1-\frac{1}{N}\right)^{n-1} \quad R_{\text{col}} = N - R_{\text{idle}} - R_{\text{id}}$$

The system's efficiency in this case is given by:

$$\text{SE}_{\text{FSA}} = \frac{R_{\text{id}}}{R_{\text{idle}} + R_{\text{id}} + R_{\text{col}}}$$

In case of rounds of the same exact duration, this value tends to 37%.

The **Electronic Product Code Generation 2 (EPC Gen 2)** standard is a protocol based on FSA. EPC adapts frame length according to the number of collision and empty slots. In particular, EPC Gen 2 specifies the **transmission time model**, which allows us to estimate a temporal evaluation of the protocol's performance.

The key aspect behind this model stands in observing that *idle responses*, i.e. slots where no tag answers, can last less than identification or colliding responses because the tags don't have to reply to the reader.

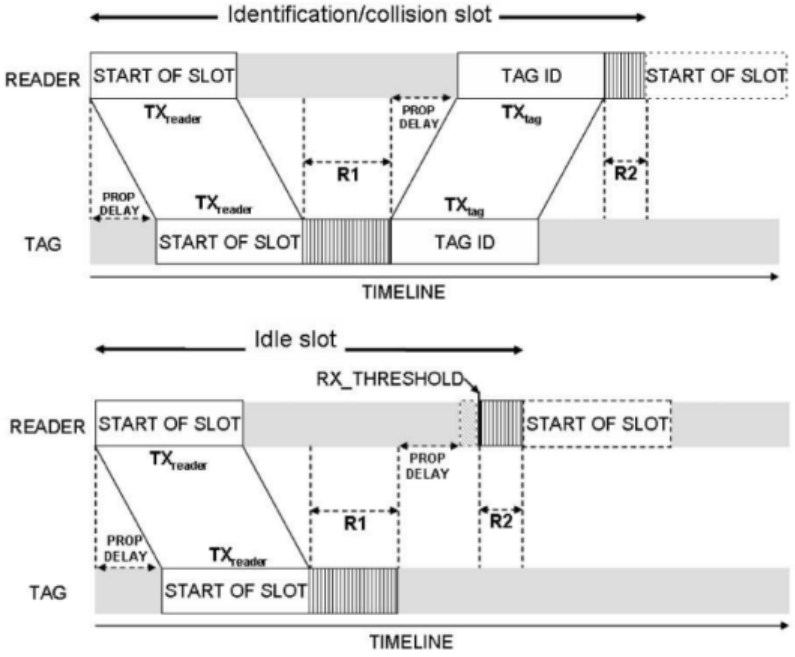


Figure 1.4: The two types of slots used by the EPC Gen 2 standard.

Hence, if idle rounds last a β faction of the identification and collision rounds, we get that:

$$\text{TimeSE}_{\text{FSA}} = \beta \frac{R_{\text{id}}}{R_{\text{idle}} + R_{\text{id}} + R_{\text{col}}} = \frac{n\left(1 - \frac{1}{N}\right)^{n-1}}{(\beta - 1)N\left(1 - \frac{1}{N}\right)^{n} + N}$$

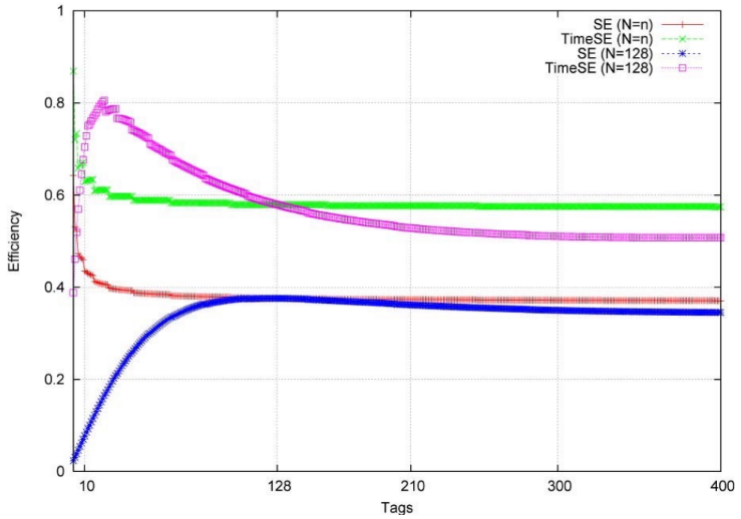


Figure 1.5: Performance comparison with and without the transmission time model.

From the above graph, we observe that the best performance is reached with the transmission time model. In particular, we observe that when the number of tags is known, the amount of collisions can be drastically reduced (initially) by setting $N = n$. On average, this new version of FSA wastes 40% of the time slots in idle and collision slots.

This number can be further improved through a mode complex protocol, that being the **Tree Slotted Aloha (TSA)** protocol. In this variant, slots are executed following a tree: a new *child frame* is issued for each collision slot, where only the tags in the same slot participate in the new child frame.

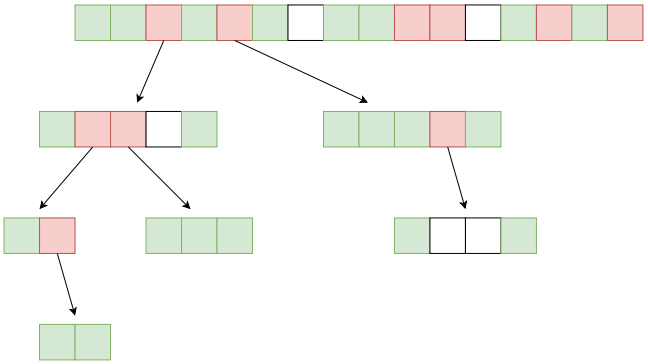


Figure 1.6: The TSA protocol in action. Green squares represent identifications, red squares represent collisions and white squares represent idle rounds.

To estimate the performance of the TSA protocol, we count the number of nodes in a fashion similar to the BS protocol:

$$\mathrm{TSA}_{\mathrm{tot}}(n) = \begin{cases} 1 + \sum_{k=0}^{n} \binom{n}{k} \left(\frac{1}{2}\right)^k \left(1 - \frac{1}{2}\right)^{n-k} \mathrm{TSA}_{\mathrm{tot}}(k) & \text{if } n > 1 \\ 1 & \text{otherwise} \end{cases}$$

For large values of n we get that:

$$\lim_{n \to +\infty} \mathrm{SE}_{\mathrm{TSA}} = \lim_{n \to +\infty} \frac{n}{\mathrm{TSA}_{\mathrm{tot}}(n)} \approx 0.42$$

1.2.3 Estimating tag population

In order for a RFID communication protocol to be effective, it must work with a potentially infinite number of tags. This makes the initial number of tags unknown to the RFID reader. Initial frame size is usually set to a predefined value (typically 128 slots), while the size of the subsequent frames is computed through the output of the following formula:

$$\text{tags per collision slot} = \frac{\text{estimated tot. num. of tags} \; - \; \text{identified tags}}{\textit{collision slots}}$$

The number of identified tags and number of collision slots is known, but we dont know the total number of tags. This last value is estimated according to the outcome of the previous frame using Chebyshev's inequality. Let:

- N be the size of the previous frame
- (c_0, c_1, c_k) be a triple of observed values
- (a_0, a_1, a_k) be a triple of estimated values

The **estimator function** ε is defined as:

$$\varepsilon(N, c_0, c_1, c_k) = \min_{n \in [c_1 + 2c_k, 2(c_1 + 2c_k)]} \left| \begin{bmatrix} a_0^{N,n} \\ a_1^{N,n} \end{bmatrix} a_k^{N,n} - \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} c_k \right|$$

where $a_r^{N,n}$ is the expected number of slots with r tags:

$$a_r^{N,n} = N \binom{n}{r} \left(\frac{1}{N}\right)^r \left(1 - \frac{1}{N}\right)^{n-r}$$

We observe that the estimator function doesn't capture the possibility of high variance in the number of tags. We also observe that the upper bound $2(c_1 + 2c_k)$ is not adequate for networks composed of thousands of nodes. For instance, if we have 5000 tags with $N = 128$, it is highly likely that for $c_1 = 0$ the upper bound is estimated as $2(c_1 + 2c_k) = 512$, which is definitely to small. However, an *unbounded estimator* can still be not accurate.

The key observation here lies in the fact that starting with a propert frame size leads to better estimations for intermediate frames (improved convergence). Hence, the problem can be compressed into estimating the initial tag population to properly set the size of the first frame. The two common solutions for this task are the *Dy_ TSA protocol* and the **Binary Splitting Tree Slotted Aloha (BSTSA)**. The latter protocol uses BS to randomly split tags into groups whose size can be easily estimated and then use TSA in these groups.

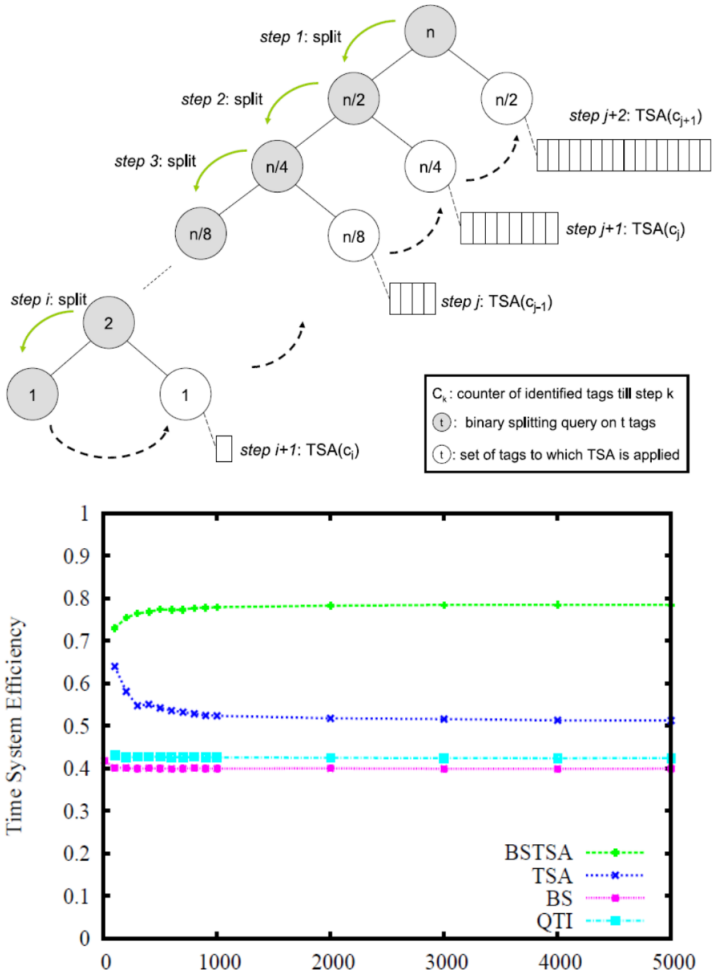


Figure 1.7: The BSTSA protocol in action and its performance compared to other protocols.

1.3 Wireless Sensor Networks (WSN)

In wireless networks, sensors are devices provided with battery that continuously sense the environment, listening on the channel (*carrier sense*) and spontaneously transmitting information (*no backscattering*) when something is sensed. The device receiving the information of multiple sensors is often referred to as *sink*. In particular, we observe that, very differently from RFID tags, sensors can achieve complex computations. Moreover, multi-hop communication is also possible: sensors can exchange information either through the sink or by communicating directly with each other.

Formally, a **Wireless Sensor Network (WSN)**is a sensor networks composed of distributed devices that monitor and record environmental conditions, sending the discovered data to a central node for processing and analysis. The central node may then send a signal to a control device that takes action regarding the gathered information.

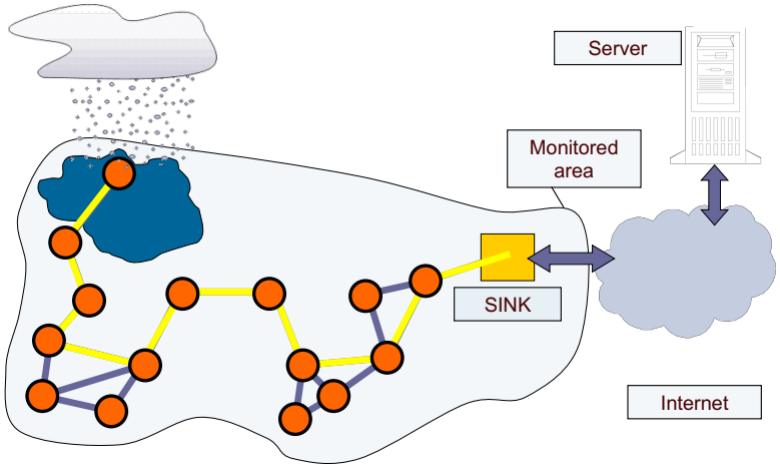


Figure 1.8: Example of a WSN with sensors that measure humidity in order to detect weather changes.

Sensor networks have multiple benefits compared to other architectures:

- **Large-scale coverage**: they can collect data across vast geographic areas, even in remote or inaccessible locations.
- **Autonomous operation**: each sensor is independent from the others, minimizing the need of human intervention
- **Real-time data**: their simplicity allows immediate access to critical data, enabling rapid response to environmental changes

For these reasons, WSNs are emplyed everywhere there is a need for monitoring a physical space or using sensors for controlling a procedure (e.g. industrial control, marine monitoring, health care, smart homes, structural health, ...).

We categorize devices taking part in a WSN in three main types: **sources** of data (*sensors*), **sinks** of data (*central nodes*) and **actors** (*control devices*). To deploy sensors, there are three main methods:

- **Random deployment**: sensors are dropped from aircraft using an uniform at random distribution over the area
- **Regular deployment**: sensors are disposed in a well planned schema, not necessarily resembling a geometric structure (often assumed to be for cost efficiency, even when false).
- **Mobile sensors**: sensors can autonomously move according to a distributed algorithm, compensating uncovered areas. They can be passively moved around by some external force (wind, water, ...) or actively seek our areas of interest

In particular, mobile sensors deployment rose in recent years due to the introduction of network paradigms based on continuously changing and adapting topologies, instead of fixed ones. However, all types of WSNs come with some issues that must be mitigated:

- **Information content**: data must be processed in-network and it must provide answers to queries or events triggered by sensors. This implies that there is an asymmetric flow of information inside the network (sensors to sink).
- **Quality of service**: traditional network QoS metrics do not apply in sensor networks.
- **Fault tolerance**: the network must be robust against node failure, which may happen more commonly due to sensors being fragile.
- **Energy consumption**: sensors networks must achieve their task for as long as possible, making energy one of the main issues due to the low resources available and the necessity for continuous data gathering. This is the main critical issue in WSNs.

Sensor nodes are composed of few components: a CPU, a memory unit, a power source (usually a battery), an antenna, a radio frequency transceiver and a sensor unit. Alls these components are managed through a standard operating system called TinyOS, initially developed by the University of California. Sensor units capture a signal corresponding to a physical phenonmenon, which then gets prepared for further use through *signal conditioning* (e.g. amplification, attenuation, filtering, ...) and converted from analog to digital.

To mitigate energy consumption, sensors are constructed in an energy efficiency principle: sensor networks are typically deployed in an ad hoc fashion, with individual nodes remaining largely inactive for long periods of time, but then becoming suddenly active when something is detected (**trigger**). Nonetheless, some energy is wasted by nature of the sensors themselves:

- Collisions may force the sensors to retrasmit information
- A node may receive packets that are meant for other nodes (**overhearing**)
- A minimal number of control packets must be used for data transmissions
- Nodes must be constantly listening to an idle channel in order to get triggered

- A node may transmit packets to a node that is currently busy (**overemitting**)

Along with energy consumption, WSNs share the same wireless communication criticall issues as any other wireless network:

- **Attenuation**: the electromagnetic signals decreases rapidly as the distance from the transmitter increases (the signal is dispersed in all directions).
- **Multi-path propagation**: when a radio-wave hits an obstacle, part of the wave is reflected with a loss of power. This implies that a source signal may arrive at the same destination multiple times through multiple reflections.
- **Interference**: a recipient can receive multiple signals from the desired sender due to multi-path propagation or from multiple trasmitters that are using the same frequency to communicate with other recipients (e.g. Wi-Fi, Bluetooth and Microwaves all share the same frequency range).

These issues are measured through **Signal-to-Noise Ratio (SNR)**, the ratio between good (signal) and bad (noise) signals received by a device. When the SNR is high, the signal is stronger than the noise. When low, the amount of noise is greater.

Furthermore, there are some inheritent issue that arise in wireless communication due to its very own nature such as the **Hidden Terminal Problem (HTP)**: two nodes A and B may be communicating with the same device B without knowing the existence of each other, forming collisions in the network.

From the above discussion, it should be clear that the key to mitigating both energy consumption and wireless issues lies in the MAC protocols used by the sensors. In particular, WSN MAC protocols must control *when* a packete must be sent and *when* a packet must be listened. All protocols for sensor networks are based on two communication patters that alternate between each other:

- **Broadcast** (or *Interest Dissemination*): the sink transmits information to all sensors in the network (One-to-All), typically used to send queries, update sensors and control packets
- **Convergecast** (or *Data Gathering*): a subset of sensors nodes send data to the sink (Many-to-One), typically used for collecting sensed data

1.3.1 Contention-based MAC protocols

WSN MAC protocols are based on one of two techniques: **contention** and **scheduling**. In the former, on-demand allocation is used for nodes that have information to transmit, managing collisions when they occur. In the latter, the sink specifies when and for how long each node may transmit over the shared channel. Contention based protocols are more scalable but less energy efficient, while scheduling based ones are more energy efficient but require a synchronization and a central authority in the network.

For our interest, we'll discuss contention based MAC protocols. In particular, we'll focus on the **CSMA/CA** protocol defined by the IEEE 802.11 standard. Here, CSMA/CA stands for **Carrier Sense Multiple Access with Collision Avoidance**, implying that

the protocol cannot detect collision, but it can only try to avoid them as much as possible (a more advanced protocol, the CSMA/CD, is capable of doing so).

When a node wants to transmit a frame, it first *senses* the channel for a small time slot called **interframe space (IFS)**. If the channel is *idle*, i.e. no device transmits during the sensing period, the device immediately trasmits when the IFS ends.

Otherwise, if the channel is busy, the station continues to monitor the channel until the transmission ends. Once the transmission is over, the station delays another IFS. Then, the node enters a *contention window* with the other nodes, where each of them waits for a randomly chosen amount of time (**back-off phase**).

The "luckiest" node whose wait time expires first will be the first one to return to the sensing phase, making it the first one to trasmit. The back-off clock of each node is randomly chosen from the range $[0, \mathrm{CW} - 1]$. In the *binary exponential back-off*, the value of CW is doubled each time the node enters the back-off phase, getting reset to the initial value when the node successfully transmits.

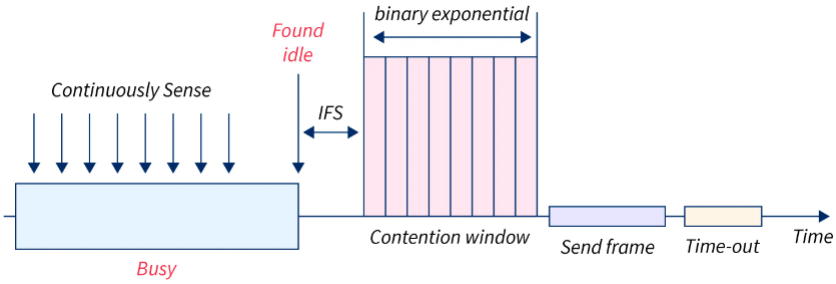


Figure 1.9: The CSMA/CA protocol in action.

Advanced versions of CSMA/CA use different type IFS in order to dictate priorities among nodes. In particular, **Short IFS (SIFS)** are used for packets of the highest priorities while **Distributed coordination function IFS (DIFS)** are are used for packets of the lowerst priorities and asynchronous data services. DIFS are usually as long as a SIFS plus two time slots.

In *Distributed coordination function CSMA/CA with Acknowledgement* (DFS CSMA/CA ACK), each station has to wait a DIFS before sending data (instead of a generic IFS). When the destination receives the data, it waits for a SIFS and then immediately sends an ACK message (without sensing the channel) if the packet was correctly received. If the ACK is lost, the trasmission is repeated. Otherwise, the sending node waits a DIFS and then enters the contention window.

The CSMA/CA protocol suffers from not only the Hidden Terminal Problem, but also from the *Exposed Terminal Problem*, a similar problem that forms when a terminal's communication is postponed by the carrier sense even though no collision would happen.

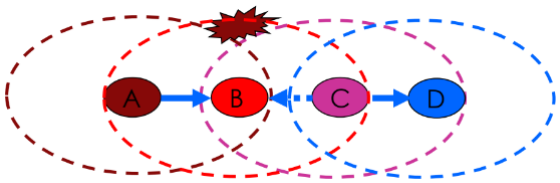


Figure 1.10: Since the nodes A and C cannot hear each other, when A transmits to B, C cannot sense the transmission. Hence, C may transmits to D during this time slot, forming a collision for B.

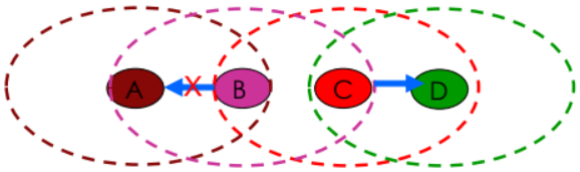


Figure 1.11: Since B and D cannot hear each other, B may transmit to A while C may transmit to D without creating a collision. However, when B tries to transmit to A, it may detect the channel as occupied by C, postponing the transmission.