# Introduction to IoT: Home exam 2019

TOMMASO CARRARO (015147592), University of Helsinki

**ACM Reference Format:**
Tommaso Carraro (015147592). 2019. Introduction to IoT: Home exam 2019. 1, 1 (November 2019), 15 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 TASK 1

### 1.1 Modulation

The modulation is the process of mixing a signal with another one to create a new signal that is better suited for transmission, so it is a technique that is used to improve transmitted signals. The signal used for mixing is called the carrier signal, while the final mixed signal is called modulated signal. In IoT applications the modulation is mainly used for transmitting information over long distances without interferences but it could be used also to separate the signals from different transmitters or reduce the size of the antenna needed for the communications. To perform the modulation the frequency, magnitude or phase angle of the signal can be manipulated. There are different types of modulations and the types we have seen during the course are:

- **Frequency-shift keying modulation**: these are techniques where the frequency of the modulated signal varies according to change in the data. This is widely used in short range communication technologies;
- **Multiple access protocols**: these are protocols used to allow different transmitters to use the same channel simultaneously. These are needed because wireless channels are usually not dedicated to specific transmitter/receiver pairs. There are different ways to implement these protocols:
  - **Frequency division multiple access**, where the bandwidth is divided into sub-bands so that each transmitter can be allocated its own band;
  - **Time division multiple access**, where the bandwidth is divided into time slots and transmitters use their allocated slots;
  - **Code division multiple access**, where all data is carried simultaneously but using different codes, one code per transmitter.

Some use cases of the modulation in IoT that we have seen during the course are the modulations used in the different PAN technologies we have studied, for example ZigBee uses the phase-shift keying modulation, specifically the offset quadrature phase-shift keying modulation.

### 1.2 Preemption

The preemption is a scheduling technique used in the operating systems. The scheduler is one of the key components of an embedded IoT operating system because it affects the energy-efficiency, the

Author's address: Tommaso Carraro (015147592), tommaso.carraro@helsinki.fi, University of Helsinki.


XXXX-XXXX/2019/11-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

reactivity of the system and the programming model. There are two types of scheduling algorithms we have seen during lectures:

- **preemptive**, where a task can be interrupted and there is a limited CPU time allocated per task. Since every task has a fixed CPU time there is the need of a timer to manage the tasks. It is more flexible than the non-preemptive scheduling because the tasks can't be interrupted but it has overhead related to the management of these tasks. Finally, the main disadvantage of this technique is that it can starve processes with low priority if processes with high priority continue to arrive;
- **non-preemptive**, where the tasks must complete or yield for others to have a chance to be executed. In this technique there isn't a priority for the tasks and that means that every task has the same priority, so the tasks are executed in the order they arrive, like in a FIFO row. The main disadvantage of this technique is that if a task that requires a long CPU time arrives, the following tasks have to wait the entire execution time before they can be executed by the processor, even if they are small tasks compared to the first one.

A use case of these scheduling techniques could be the TinyOS embedded operating system we have seen during lectures. It uses preemption for events and commands and non-preemption for the tasks, so it uses an hybrid scheduling technique.

### 1.3 Duty cycle

The duty cycle is an energy management technique where the sampling of the sensor is reduced by alternating between idle and active states of the sensor. This technique is highly used in IoT and specifically in low-end devices to reduce the energy drain, in fact in this class of devices the energy is a key criteria for operation because since they are usually really small, they have a limited battery. The duty cycle can be defined as the percentage of time where a sensor is active and it is defined by the following formula: $D = \frac{T}{P} * 100\%$, where $T$ is the active time of the sensor and $P$ is the period of the sensor. Duty cycle schemes have to be carefully designed in IoT applications and many aspects have to be taken into account:

- possible dependencies between sensors, networking and control units that could affect the power draw;
- activation costs related to the transition between a lower sampling state to an higher sampling state;
- delays related to the transition between an high sampling state to a lower sampling state;
- cross-device interactions: there could be correlated energy profiles or cross-device sensitivities.

Finally, determining the optimal duty cycle requires information about the power draw (power profiles) of the different software and hardware components of the specific IoT device. To provide an example in an IoT context I copy one of the questions from this exam: "A car is driving at 120km/h along a snowy road. The car has an object detection algorithm that has a range of 200m. During these conditions, the braking distance from 120km/h to 0km/h is 189 meters. What should the duty cycle of the Lidar be so that the car has sufficient time to stop in case there is an obstacle?". This example shows a simple but detailed case where the computation of the optimal duty cycle is critical for IoT, not only to save battery but also to make the system reliable.

### 1.4 Vertical and horizontal scaling

Since big data arrive very fast they increase very fast in volume, so sometimes they need to be processed in real-time to avoid loss of opportunities or important analysis. The computational power of the IoT devices is usually limited if compared to the server clouds used to deal with the

big data. The vertical and horizontal scaling are techniques used in the big data field to increase the resources of IoT devices in such a way that they are capable to deal with the velocity and the volume of the big data. The vertical scaling is used when we want to increase the power of the device changing its internal architecture, for example swapping microcontrollers or using a more powerful surrogate. The horizontal scaling is used when we want to increase the power of the system by using more than one single device and connecting them in a distributed way in such a way to use the paralelization to deal with the big data. We have seen that the horizontal scaling is a better approach in case of big data because the improvements on the computational power are slower than the velocity of the big data. An example of horizontal scaling is the distributed model used in the data management for sensor networks. When there isn't the possibility to use a central gateway to store the data of the different sensors in the network we have to use the devices capabilities to be able to store these data. The sensor network becomes a distributed database of sensor measurements generated by the different sensors in the network. In this case we can say that this is horizontal scaling because we have to use different devices connected in a distributed way to be able to store and managed all the data, in fact only one device hasn't the memory to store the data that arrive from all the different sensors in the network. An example of vertical scaling could be the adding of RAM to a specific IoT device to make it capable to implement more powerful functionalities.

### 1.5 Distributed file system

A distributed file system is a file system where data are distributed along a series of nodes in the network. These nodes are usually contained in a rack and different racks are connected together by the network. The main design goals of a distributed file system are:

- **scalability**: since big data grow very fast we need to scale gracefully to be able to store these data;
- **failure tolerance**: since the failure of storage components is common there is the need of data replication between different nodes that are located in different parts of the distributed network. In this way, if a node fails the process can be started another time in another node that has the same data of the failed node.

Finally, a distributed file system is built on the following assumptions:

- there is a low number of files that have a large size;
- the random writes are rare because the files are typically read and written sequentially and most of the writes append data to the end of the file.

The initial goal of the distributed file systems was the management of web data but they fit well the IoT environments because in these environments there are usually a lot of devices that produce a huge amount of measurements or data in general, so there is the need to store large size files containing those measurements with the possibility to append new measurements to the end of the file. This is the main reason of the use of distributed file systems in IoT environments. Hadoop distributed file system is an example of distributed file system that we have seen during lecture. This file system is highly used by the Hadoop ecosystem when we need to perform a map reduce algorithm to compute a specific job. These file systems are the basement of the big data frameworks which are responsible for handling the execution of programs submitted by the clients. The client usually submits a job that is divided in tasks that are distributed by the framework on different nodes of the distributed file system to perform the computation. In IoT applications a distributed file system is used every time a job is sent to the server cloud for a batch processing.

### 1.6 Inversion attack

The model inversion attack is a sensing pipeline attack where the attacker tries to recreate aggregate information of the classes used to train a model. Having access to a machine learning model and a label it is possible to create synthetic data to fed it to the model and then compute a loss between the output of the model on the provided data and the known label. The idea is to find the synthetic data that minimize the loss function between the output of the model and the known output. This attack could be performed in every IoT environment that uses sensing pipelines to perform its functionalities and a specific example could be the smart home and specifically the facial recognition system used to access it. Having access to the facial recognition algorithm and a person's name the attacker could try to recreate the face of the person using an inversion attack, accessing the identity of the person.

### 1.7 Cyber-foraging

The cyber-foraging is one of the first model invented for the computational offloading. The computational offloading is the process of transferring the computation of a heavy task from one device to another that has more computational power than the first one. The device that sends the task is called client, while the device that takes the computation of the task is called surrogate. Offloading paradigms differ on two aspects:

- where the surrogate is physically located;
- how offloading decisions are made.

In the cyber-foraging the surrogate is located in the Personal Area Network where the client is located, in fact the idea is to "forage" the resources available in the infrastructure around the user and to send their heavy computations that can't be performed on the client. The following example explains a use case of cyber-foraging in an IoT application. We assume we have an application that performs image recognition based on deep learning. This application is installed on a smartphone that hasn't enough computational power to perform the inference of the model. With cyber-foraging we can send the heavy computation of the inference of the model to a laptop in our Personal Area Network. In this way the only task that the smartphone has to do is to send the image to the laptop and then display the result to the user after the laptop has returned it. Note that in this example only the inference of the model is performed on the laptop, while the training of the model has to be performed on a server cloud because on the laptop there isn't enough computational power.

### 1.8 Multi-factor authentication

The multi-factor authentication is a method to perform a secure authentication to a web service that consists on the use of multiple authentication factors for verifying the identity of the user that is trying to log in the web service. One example of multi-factor authentication is two-step authentication that combines a knowledge factor with an ownership factor. In this method when the user tries to access to the web service with his credentials the server sends to his device a one-step passcode that has to be inserted by the user to complete the login. The multi-factor authentication is a secure technique that is highly used in IoT applications to verify the admin of the system. Another example of multi-factor authentication that we have seen is the sound-proof. In this method when the user tries to log in the web service the server asks to the browser and to the user's device to record an audio of the environment (this is called environmental fingerprint). After the audios have been recorded the browser sends the audio to the server and the server sends it to the user's device. Finally, the user's device compares the received audio with the recorded audio and if they are similar the authentication can be completed. This is a technique that is subject to sound-danger attack, a form of context attack. An important use case of this method could be healthcare. For

example if it is necessary to access to a pacemaker management page the login must be secure because an attacker could access it and use the device's sensors to kill the patient (this is written in one article I've chosen for the course exercises).

## 2 TASK 2

### 2.1 Question 1

The different peripherals used in this IoT application are:

- on the robot:

(1) **XBee shield receiver**: this is an antenna that receives the impulses that arrive from the XBee shield transmitter placed on the glove. These impulses define the movements that the robot has to perform (forward-left, forward-right, forward, backward). It is a sensor because it receives signals;
(2) **Buzzer**: this is an actuator that emits 8-bit sound when the user is moving the hand but he isn't pressing the custom button placed on the glove. This is a way to alert the user that the robot isn't moving because the custom button isn't pressed;
(3) **Gearmotors**: these are the motors that allow the robot to move, so they are actuators;
(4) **Acceleromenter**: the accelerometer on the robot isn't used in this application but it is a sensor to detect the speed of the robot;
(5) **Mechanical bumper**: it is used to detect if the robot is touching a wall so it is a sensor but it is not used in this application;
(6) **Line follower**: it is a sensor used to allow the robot to detect and follow lines in the flatground but it is not used in this application.

- on the glove:

(1) **XBee shield transmitter**: this is a transmitter that sends signals to the robot when motion is detected by the accelerometer in the glove. Using these signals it is possible to move the robot. Since the signals are sent we can say that this transmitter is an actuator because it is sending something and not receiving;
(2) **Custom button**: this custom button is made of snappable pins, conductive thread, and wire. This is used to detect the contact between the thumb and the middle finger of the user's hand, in fact if the contact is detected it is possible to move the robot with the hand motions. Since this is a button it is classified like a sensor;
(3) **Accelerometer**: it is used to detect the motion of the hand and to understand in which direction the motion is performed, so it is a sensor. The motion of the robot is controlled by the orientation of the hand;
(4) **RGB LED**: the led is used to allow the user to understand if he has performed a motion of the hand in the right way, in fact it will light up based on the mode and the orientation of the hand. The LED is an actuator.

### 2.2 Question 2

I would use a microcontroller for this IoT application because:

(1) **the computing requirements are limited**: since this smart object is only made of sensors the computing requirements are limited, in fact the only thing that has to be done in this application is to detect the hand motion and send signals to the robot;
(2) **the power requirements are really small**: the devices are powered with small power sources (alkaline batteries). It isn't necessary that the battery last for a long period of time because it is possible to charge it in every moment but the control unit doesn't have to waste to much battery because the amount of battery is limited (we want to avoid changing or

recharging the battery many times). The microcontroller is really useful in this case because it is power efficient, in fact it has been designed for low end devices like the devices in this application;

(3) **a lot of peripherals are needed**: there are a lot of sensors that have to be used to control the robot, so a board with a lot of pins is required. In this application Arduino has been used. This microcontroller offers an easy way to integrate external components, such as sensors, so a microcontroller is enough for this IoT application.

### 2.3 Question 3

The requirements that the objects would impose to the operating system are:

- **Memory**: since this application uses Arduino microcontroller an operating system with a microkernel has to be used in this case. A microkernel operating system is an OS that includes only the essential operations in such a way that it can be lightweight and it can be installed on a microcontroller that has a small amount of memory (Arduino has only 32kb in total for programs and OS);
- **Energy management**: sleep modes can be used to save energy in this IoT application. For example when the custom button on the glove isn't pressed this means that the robot doesn't have to move, in fact the only thing it has to do in this case is to alert the user with a 8-bit sound emission. In this particular case we can use a light sleep on the robot and re-active it when the button is pressed. In light sleep mode the robot uses only the buzzer to emits sounds and the network interface to listen for new signals from the glove, while all the other sensors can be turned off;
- **Programming model**: since the application is only made of sensors an event-based programming model could be used. The implementation could be the following one:
  - when the user touches the custom button on the glove an event is triggered and an event handler will active the robot (the robot was in light sleep mode);
  - when the user moves his hand the accelerometer detects a motion and creates an event related to that motion. An event handler will move the robot in the direction defined by the hand movement;
  - if the user releases the button an event is triggered and an event handler will put the robot in light sleep mode.

  The work of the scheduler is obviously more complex but this is just an example to motivate my programming model choice;
- **Scheduling**: since in this application there aren't priorities for the tasks and there isn't the need of allocating a limited CPU time for each task, the non-preemptive scheduling can be used. Other motivations could be that the system has to be power-efficient and that it hasn't to be real time and that means that each task can take its time to be executed by the CPU (even if there are delays on the movements of the robot it isn't critical for this application);
- **Memory management**: static memory allocation is used on the stack, while dynamic memory allocation is used on the heap of the Arduino;
- **Communication support**: since the microcontroller has a small amount of memory and the device has a small amount of battery, a low power networking stack has to be used (subset of TCP/IP stack);
- **Programming support**: since Arduino has to be programmed in C++, the operating system have to support the C++ programming language.

I would use Contiki OS for this application because it fits perfectly the requirements I have found for the application:

(1) small size operating system;
(2) event-based programming model;
(3) non-preemptive events;
(4) dynamic allocation support (for the heap);
(5) subset of TCP/IP stack.

### 2.4 Question 4

The smart objects in this IoT application need to use batteries as power sources, in fact these aren't batteryless portable devices and they can't be connected to the main power supply. Since batteries can be replaced we can focus in the maximization of the lifetime of these small batteries (I mean they last a lot before they need to be changed). The battery that fits these requirements is the Nickel-Metal Hybride battery that has an high energy density but low longevity. Since we can replace it, the poor longevity isn't a problem, in fact our goal is to maximize the energy density.

### 2.5 Question 5

In this application the energy harvesting could be used as supporting power source because the devices already have batteries. The best energy harvesting technique we have seen that can be used in this application is the kinetic energy harvesting, where piezoelectric harvesters convert kinetic vibrations into energy. In this application the source of the kinetic energy is the hand movement performed by the user to move the robot. The kinetic energy produced by the hand could be used for supplementing the glove battery.

## 3 TASK 3

### 3.1 Question 1

The classification of the sensors of the Google's self-driving car is provided in Table 1:

| Sensor | Proprioception vs exteroception | Active vs passive | Contact vs non-contact | Visual vs non-visual |
|---|---|---|---|---|
| LIDAR | Exteroception | Active (it uses the laser) | Non-contact | Visual |
| Windshield-mounted camera | Exteroception | Passive | Non-contact | Visual |
| Radar | Exteroception | Active (it uses radio waves) | Non-contact | Non-visual |
| GPS receiver | Proprioception | Passive | Non-contact | Non-visual |
| Ultrasonic sensor | Exteroception | Active (it uses ultrasounds) | Non-contact | Non-visual |
| Altimeter | Proprioception | Passive (it takes the air pressure) | Non-contact | Non-visual |
| Gyroscope | Proprioception | Passive | Non-contact | Non-visual |
| Tachymeter | Proprioception | Passive | Contact | Non-visual to take measurements and visual to display these |

Table 1. Google self-driving car sensors classification

### 3.2 Question 2

(1) The system corresponds to a:
- **Reactive intelligence**: the camera is able to observe the environment detecting objects and traffic signs on it but the car is able to act only when these immediate observations arrive, in fact the camera isn't able to be proactive. This means that it can't predict traffic signs or object movements, but it can only compute their movements in real-time and only if it is able to detect and see their;
- **Narrow AI**: the performance of the detection algorithm are comparable to human performances but the system performs only one task (real-time object detection). The system isn't able to learn, perceive, understand and function like a human being because it isn't able to predict object movements and to make proactive decisions on their;
- **Black-box intelligence**: the camera uses CNN (Convolutional Neural Network) for the object detection task. The CNNs are Deep Neural Networks that have become the standard for image classification and object detection tasks.

(2) As I said before the camera uses deep learning algorithms to perform the object detection task. In particular, it runs the inference of a CNN model, while the training has been performed on a Google's server cloud because it requires heavy processing.

### 3.3 Question 3

In this exercise I assume that when the LIDAR sensor detects an object, the car begins to brake instantly. Since the object detection range of the LIDAR sensor is 200 meters and the braking distance of the car in the provided conditions is 189 meters, it means that we have only 11 meters to perform the LIDAR measurement and then begins to brake if an object has been detected. This

means that we need a duty cycle that is able to sample the sensor at least one time every 11 meters. Since the car is driving at 120 km/h it means that it is driving at 33.3 m/s. If we sample the sensor three times every second it means that we sample it approximately every 11.1 meters, that it isn't enough to brake safely. If we sample the sensor 4 times every second instead, it means that we sample it approximately every 8.325 meters, that it is enough to brake in safe conditions. The worst case occurs when an object is at 193 meters but the previous sensor measurement has been taken 8 meters before, so when the car was at 201 meters from the object (out of the sensor range). In this case the car is able to detect the object only in the next sensor sample (8 meters after the previous one), so the car detects the object when it is at 193 meters from it and it begins to brake. This means that the car will stop approximately 3/4 meters away from the object, that it is enough to save the driver's life. Assuming that the measurement takes one thousandth of second to be performed, sample the sensor one thousandth of second every quarter of second corresponds to the following duty cycle: $T = 0.01$ second, $P = 0.25$ second $\implies D = 4\%$.

### 3.4 Question 4

The car has enough time to stop in time if the sensor is sampled five times every second because it is an higher duty cycle (5%) than the previously computed one (4%), that was enough high to brake in safe conditions.

## 4 TASK 6

### 4.1 Question 1

At individual sensor level the IoT layers more exposed to security attacks are:

- **Network interface layer**: for every sensor it is possible to intercept a message between the cow and the gateway and modify the information on it in such a way that:
  - if the attacker modifies the messages that arrive from the necklace, he can hide health problems of the cow;
  - if the attacker modifies the messages that arrive from the acid monitor, he can hide digestive problems of the cow;
  - if the attacker modifies the messages that arrive from the tail sensor, he can hide the fact that the cow is going to calve causing the death of the cow and of the calf;
  - if the attacker modifies the messages that arrive from the udder sensor, he can change the detected quality of the milk, for example if the milk has a bad quality he can hide this fact causing people to drink bad milk;
  - if the attacker modifies the messages that arrive from the pedometer sensor, he can avoid that the pedometer to alert the farmer if the insemination is arriving, causing a loss of money for the farmer.
- **Sensing layer**: for the necklace the attacker could place a sheet of paper between the neck and the lace to avoid the sensor taking measurements causing problems in the monitoring of the health of the cow.

At group sensor level the IoT layer more exposed to security attacks is the data management layer. When the data recorded from all the sensors are sent to the server cloud for batch processing operations a model extraction attack could be performed altering the predictions of the machine learning model and making the information collected and the results of the model unuseful for the farmer. Some useful information could be the insemination forecast or the calving forecast.

### 4.2 Question 2

In this case a distributed denial of service has the purpose of flooding the cows' network sending a huge amount of traffic on it preventing the access to it. If all the information stored on the cows is transmitted to the gateway at the same time we loose the majority of the information if the DDoS is performed when we send the information. Since the network is congested by the attack, if we send a huge amount of information on it, only a small part of this information will probably arrive to the gateway (moreover, the majority of the information that arrive to the gateway will be probably corrupted), while the majority of the information will be lost. If the information stored on the different cows is transmitted in different times, only the information sent during the DDoS attack will be lost, while the other information will arrive to the gateway (information sent after the end of the attack). This means that we save more information than the previous case. To answer to this question I've assumed that a DDoS attack is temporary.

### 4.3 Question 3

To be sure that sensors' measurements are not corrupted after their transmission to the gateway an integrity check should be performed on the gateway. Every time a cow sends its data to the gateway a SHA-1 cryptographic hash function has to be applied to the data. When the gateway receives the data it applies the same function to the received data and then it compares the results of the two application of the function. If the results are equal this means that the integrity has been preserved, while if the results are different it means that the measurements have been corrupted during the transmission (they may have been intercepted by an attacker or corrupted by connection problems).

### 4.4 Question 4

To create a secure communication channel between two cows it is possible to use the secure association. The general idea of secure association firstly uses the Diffie-Hellman key agreement protocol to exchange the private key between to nodes and then it uses it to encrypt and decrypt all the messages communicated in the channel (asymmetric encryption). A better way to achieve this goal is to use secure channel establishment protocols such as TLS (Transport Layer Security), that are some of the most important cryptographic protocols, enabling the encryption of Internet traffic.

## 5 BONUS TASK

### 5.1 Question 1

To compare the power consumption between Z1 nodes and SKY nodes I've used the following tabs:

- **average duty cycle**: in Figure 1 it is possible to observe that Z1s and SKYs have similar duty cycles that are in the range of 1.0% and 1.2%. Since all the Z1s are in that range and only one has 1.7% duty cycle, we can assume that this is an outlier of the simulation. Since these sensors have similar duty cycles this means that they are sampled in the same manner. Another pattern that it could be observed is that the radio listen duty cycle is constant, while the radio transmit duty cycle slightly changes (the outlier is out of this observation);

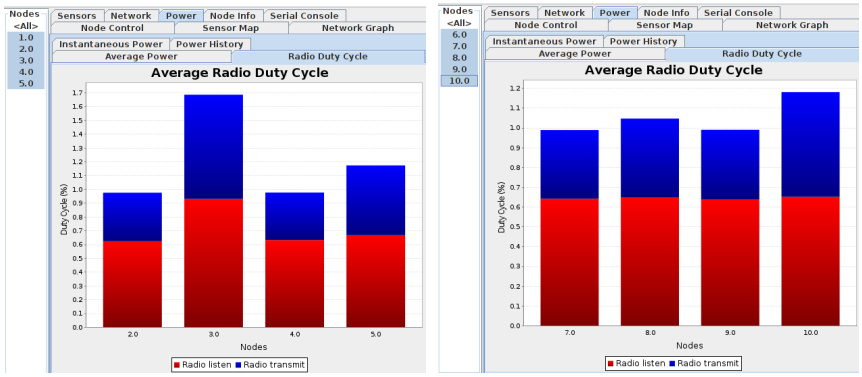


Fig. 1. Z1s (left) and SKYs (right) average duty cycle

- **power history**: in Figure 2 it is possible to observe that the Z1s become power efficient over time (after few minutes) because all the Z1s tend to stabilize around 0.60 mW consumption. The SKYs don't stabilize over time and they stay in the range 0.9-1.1 mW of consumption with different peaks and this means that the SKYs consume more power than the Z1s (going from a lower sampling state to a higher sampling state typically has an additional "activation" cost);

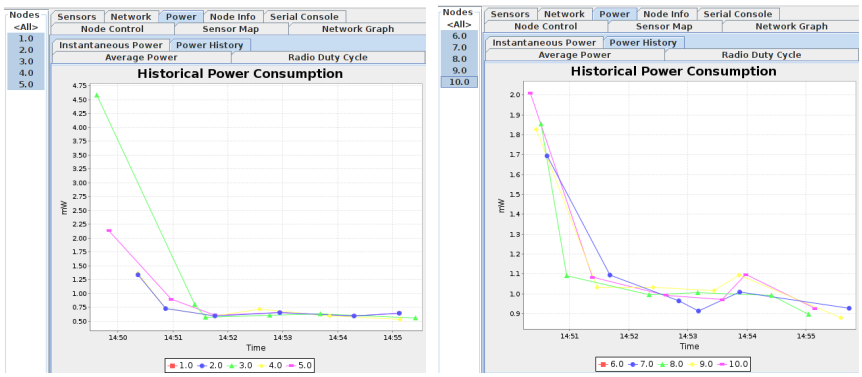


Fig. 2. Z1s (left) and SKYs (right) historical power consumption

- **average power consumption**: we have already said that the third Z1 is an outlier because it has a consumption that is out of the pattern of the consumptions of the Z1s, so I don't use it in my observations. From the average power tab in Figure 3 it is possible to observe that the radio listen, radio transmit and LPM consumptions are very similar between Z1s and SKYs, while SKYs have higher CPU power consumption than the Z1s. This means that the SKYs are less power efficient than the Z1s. It is then possible to observe that due to this CPU consumption difference the SKYs have an higher average power consumption (1.2 mW) than the Z1s (1.0).

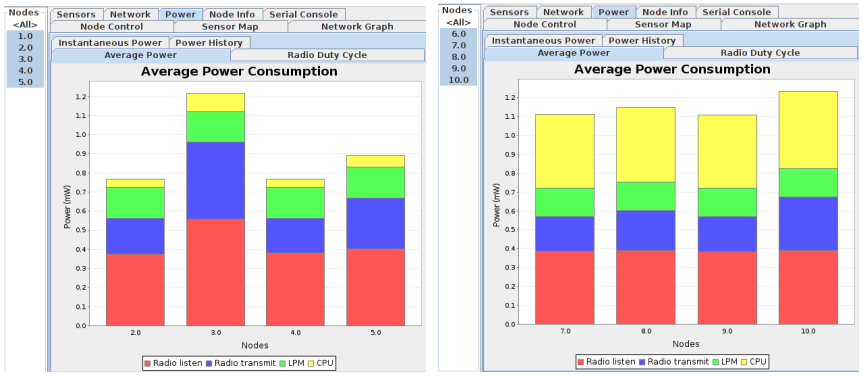


Fig. 3. Z1s (left) and SKYs (right) average power consumption

After these observations it is possible to say that the Z1 nodes perform better respecting power consumption.

## 5.2 Question 2

To answer to this question I have used the same tabs but related to the network that include both Z1s and SKYs:

- **average duty cycle**: in Figure 4 it is possible to observe that in this case the duty cycle still remains in the range 1.0-1.2% but the radio listen duty cycle is not constant anymore;

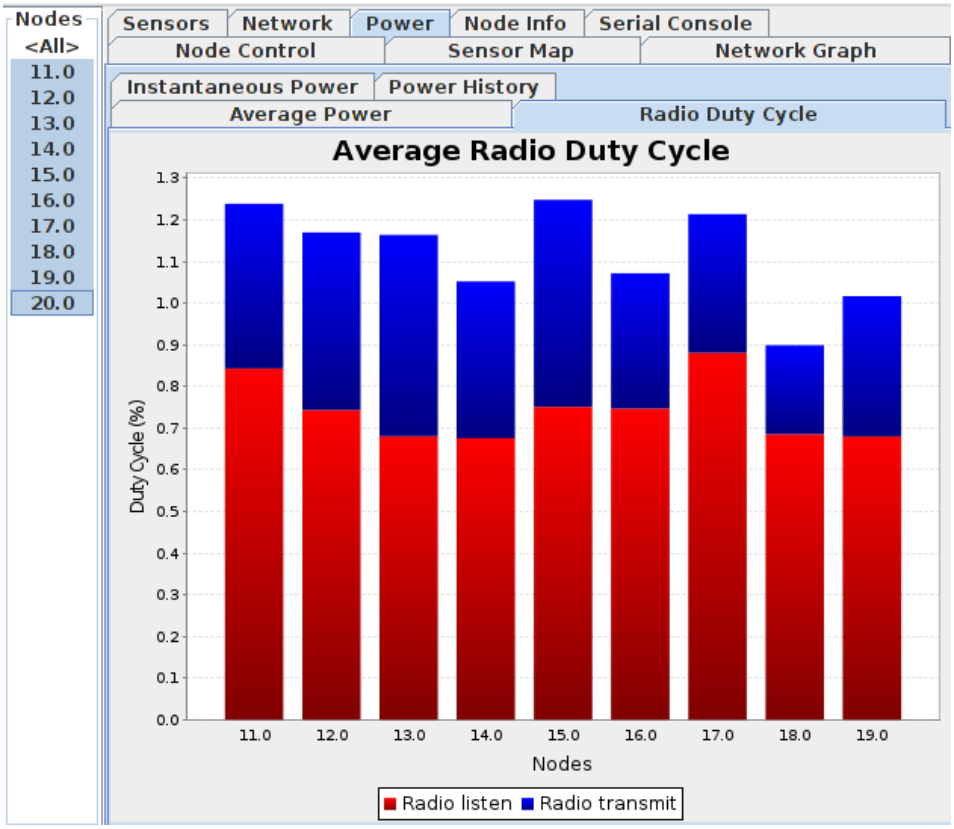


Fig. 4. Z1s and SKYs network average radio duty cycle

- **power history**: in Figure 5 it is possible to observe that the Z1s still tend to stabilize over time and they stabilize around 1.0 mW consumption. The SKYs are in the range of 0.6-0.9 mW power consumption and they are still unstable with a lot of peaks, like in the previous case. In this case even if the Z1s are stable, they have a consumption of 1.0 mW that is higher than the consumption range of the SKYs;

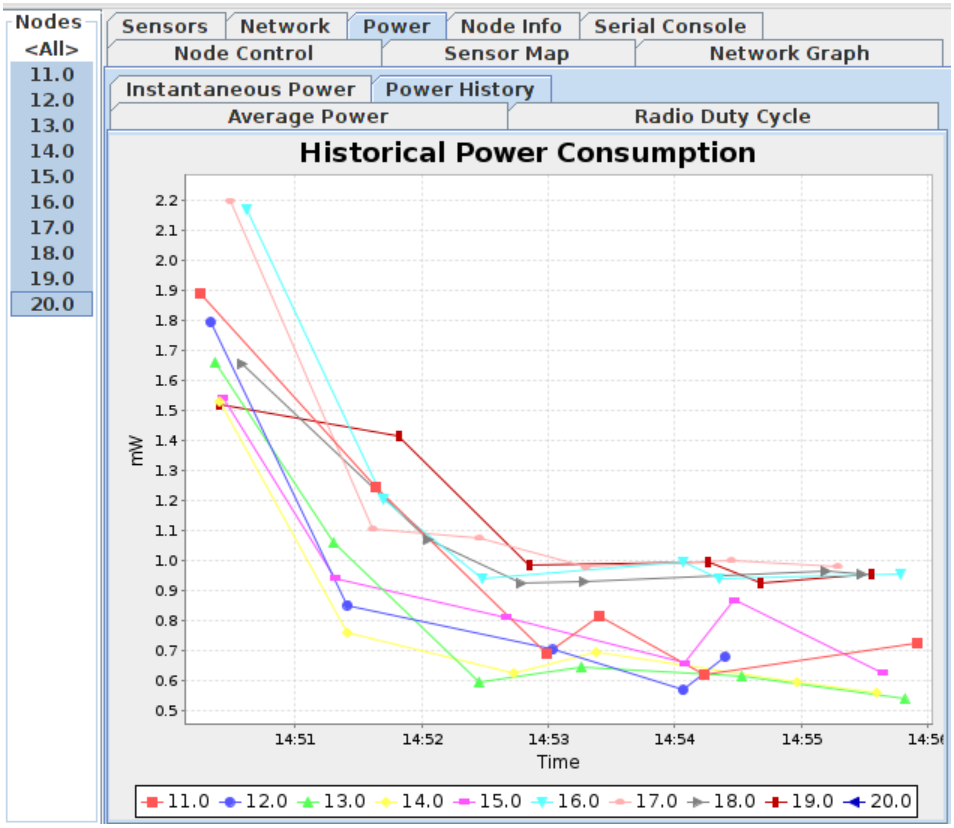


Fig. 5. Z1s and SKYs network historical power consumption

- **average power consumption**: in Figure 6 it is possible to observe that the SKYs have an higher average power consumption than the Z1s, like in the previous case. In fact the radio listen, radio transmit and LPM consumptions are very similar between the Z1s and the SKYs, while the average CPU consumption of the SKYs is higher than the average CPU consumption of the Z1s.

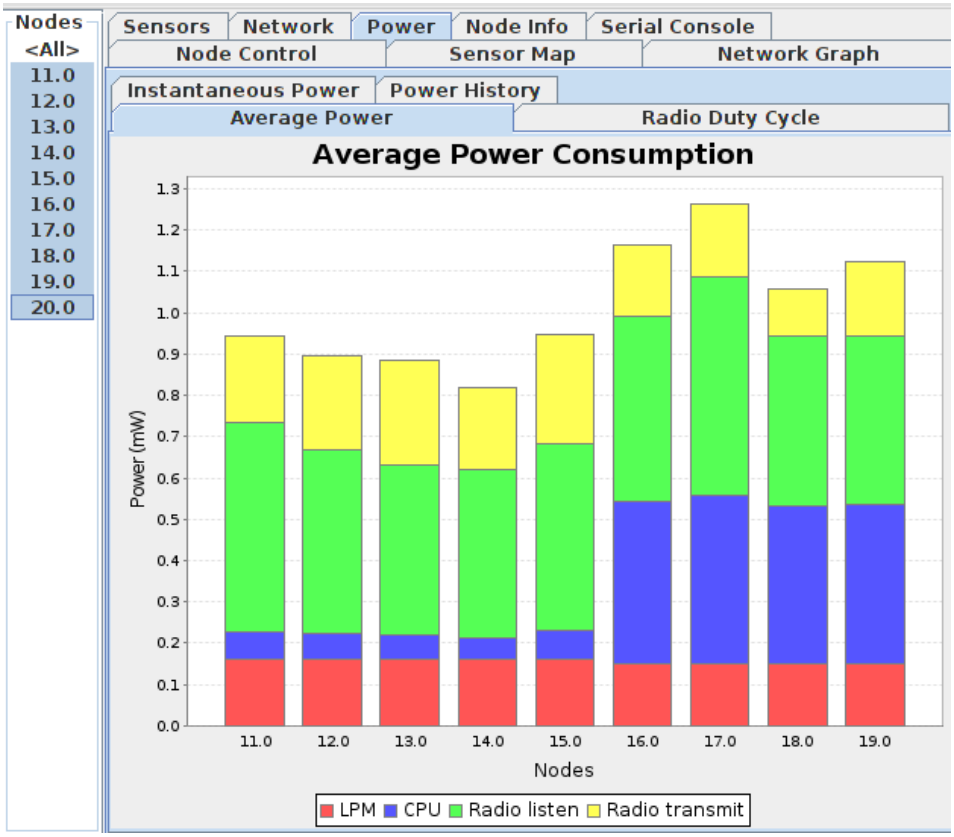


Fig. 6. Z1s and SKYs network average power consumption