Università degli Studi di Milano
Dipartimento di Informatica
Course “Simulation” - Exam Project, A.Y. 2023-2024
Student: Alessandro Di Gioacchino
Project Title: Sensor Network Simulator

Introduction

The current project involves the modeling of a sensor network system with a central edge server, emphasizing data communication and security aspects. The system consists of a collection of sensors, categorized as *roots* and others, with roots directly linked to the edge server, while the remaining sensors communicate wirelessly. Each sensor is tasked with collecting and transmitting data to the central edge server.
The primary goal of the project is to simulate the behavior of the system. This includes the generation of simulated data, its transmission across the sensor network, and the consideration of potential security threats, particularly the interception of information through selective eavesdropping on communication channels.
Critical elements within the system encompass sensors functioning as data collection points, communication channels, and the edge server responsible for aggregating and processing received data. The system faces challenges such as ensuring efficient data transmission, managing dependencies in communication channels, and addressing security concerns associated with potential eavesdropping.
Possible objectives for the simulation model involve optimizing communication protocols, assessing the impact of diverse network topologies on data integrity, and evaluating the system's resilience against potential security threats. Through the simulation of various scenarios and adjustments of parameters, the model aims to offer insights into system performance. This, in turn, facilitates the refinement of communication strategies and strengthens the overall robustness of the sensor network.

Model

The chosen simulation paradigm for this model is discrete events, an approach that governs the temporal dynamics of the system. In this paradigm, the system's behavior is directed by distinct occurrences referred to as *events*, each marking a significant point in time where state changes or transitions occur within the system.
The key characteristics of the discrete events paradigm are as follows.

- Event-driven nature:
  The discrete events paradigm operates on the principle of discrete occurrences that cause changes in the system's state. These events are crucial moments, representing actions or phenomena that impact the behavior of entities within the model.
- Simulation clock and time advancement:
  In this paradigm, a simulation clock directs the progression of time. Unlike continuous simulation models, time moves forward in discrete steps dictated by the occurrence of events. Each event is tied to a specific time point, allowing for precise control over the simulation's temporal evolution.

- Event list management:
  The simulation maintains an event list, an organized structure that stores upcoming events in chronological order. This dynamic list facilitates the scheduling and execution of events, ensuring that the simulation progresses in a well-defined and orderly manner.
- Entities and state transitions:
  Entities within the system respond to events by undergoing state transitions. The discrete nature of events allows for clear and distinct changes in the state of entities, providing a precise representation of the system's evolution over time.

The paradigm integrated into our simulation model offers a methodical and thorough approach to depict the dynamic nature of the sensor network system. It facilitates a detailed exploration of the interactions and behaviors of entities within the model, contributing to a realistic and insightful simulation experience.

Within the scope of our simulation, our focus centers on several key events that define the system's dynamics. These events involve the actions of sensors and an adversarial entity.

- Making measurements:
  Sensors initiate the process by generating measurements, which we simplify by using integers, produced according to an exponential distribution; the rate of measurement production varies based on the sensor type.
  This sets an initial boundary for the system.
- Sending measurements:
  The transmission of measurements adds another layer to our simulation. We simplify the complexities of network packet transmission, leveraging measurements that traverse communication channels at a speed determined by a Gaussian distribution. The simulation disregards packet headers, treating each measurement as slightly larger than a single integer, establishing a lower boundary in our modeling.
- Battery charging:
  To simulate the energy aspect of the system, sensors start charging with a 0.5 probability when needed. This assumption is based on the average of 12 hours of light per day over a

year, allowing for solar-powered charging. The decision to start charging is essentially determined by a coin toss, reflecting the probabilistic nature of energy availability.

- Intercepting measurements:
  The adversarial component begins with the event of successfully intercepting measurements, approximated by a Bernoulli random variable. When the attacker detects a packet in the chosen communication channel, they attempt to fetch it, adding an element of risk and security assessment to our simulation.

Implementation

The behavior of root sensors in the simulation model unfolds through a series of distinct stages, each characterized by specific actions and conditions.

- Initial disconnected state:
  Root sensors begin the simulation in a disconnected state, awaiting the establishment of a connection to an available communication channel linked to the edge server. This state represents the starting point for each root sensor.

- Measurement collection:
  After establishing a connection, root sensors start the collection of measurements from their respective environments. This phase continues until the battery percentage of a sensor falls below a predetermined threshold, indicating the need for power conservation measures.
- Transition to power-saving state:
  Upon reaching the low battery threshold, root sensors transition into a power-saving state. In this mode, the rate of data collection is halved compared to the original rate, in order to save energy and prolong sensor operation. This adaptive behavior ensures efficient resource utilization in the face of dwindling battery levels.
- Battery management and charging:
  Within the power-saving state, root sensors make decisions regarding battery management and charging. At each step, there is a 50% probability of initiating the charging process. Charging may occur until the battery level surpasses a predefined threshold or until the charging process is interrupted (because of the setting sun).
- Daylight-based charging probability:
  Leveraging the assumption outlined in the "Model" section, most root sensors are likely to initiate charging when six hours of daylight remain. This timing influences the probability of sensors returning to the power-saving state, resulting in a greater likelihood of transitioning from the power-saving to the charging phase than vice versa.
- Communication with the edge server:
  Communication between root sensors and the edge server is facilitated through a queue with fixed capacity and a delay mechanism. This configuration emulates real-world scenarios, simulating the avoidance of collisions on a wired channel. Each packet is transmitted only after the successful delivery of the previous one to the edge server.

Radio sensors in the model work quite similarly to root sensors, but they use up more battery when sending measurements. These radio sensors are connected to a root sensor, which acts like a middleman for sending data, and to other radio sensors that share the same root. To make this connection happen, there are two steps: first, a link is created between the root and the radio sensor, and then the radio sensor learns about its siblings (other radio sensors connected to the same root). Only after these steps can the radio sensors start collecting data.
When radio sensors send measurements, they randomly choose whether to send them directly to the root or pass them to a sibling radio sensor. There is no specific reason to choose one over the other, although sending the packet directly to the root is slightly preferred. Even if the connected root is temporarily unavailable, and the radio sensor might need to wait, it is still the path each packet has to take to reach the main server. So, the unavailability of the root is not a good reason to pick a destination. This leads us to the `realTimeRequirement` parameter, a number between zero and one that tells us how long a measurement can wait before being processed. A high value means it should be processed almost immediately. If the connected root is not available and this parameter is large,

the radio sensor does not waste its battery producing a measurement. Instead, it waits until the root is back on.
When there are too many waiting measurements, the radio sensor manages data from its siblings in a separate state. This depends on the `realTimeRequirement` parameter: smaller values allow for more sibling packets to wait.

Communication channels in the simulation go through a couple of steps: first, they wait to be connected, and once that happens, they get to work serving the root sensor. They stay busy until the edge server gets the packet.
For what concerns the speed at which data packets move along these channels, we use a normal random variable whose unit of measure is Kbytes per second. This random variable has an average speed (expected value), and there is a little wiggle room around that speed (standard deviation), although the distribution is very concentrated around that expected value as wired links usually do not have many issues with reliability.
(in the project: add a similar thing for radio sensors)

The router serves a pivotal role in overseeing the interconnections between root sensors and communication channels, as well as the connections between root and radio sensors.
Initially, the router awaits connection requests from root sensors, facilitating the linkage of each root sensor to an available communication channel. Then it processes connection requests from root sensors, employing a uniform random selection method to designate a suitable root sensor for each request.
In the context of informing radio sensors about their siblings, the router also maintains a mapping between root and radio sensors. After processing all sensors, the relevant section of this mapping is conveyed to the radio sensors. This transmission enables radio sensors to establish connections with their siblings, thereby completing the connection phase.

Key System Parameters (KSP) and Key Performance Indicators (KPI)

For both root and radio sensors, the most important parameter is the sampling rate, as it directly influences a key system metric, i.e. the volume of packets received by the edge server. Specifically, there is a clear correlation between this KPI and the sampling rate; however, an increase in the sampling rate is also likely to lead to a faster discharge of the sensors. Finding a balance between maximizing the number of measurements and minimizing downtime becomes an interesting consideration.

Radio sensors have the ability to promptly transmit their measurements to either the root or a sibling, so they do not need a buffer (although it could be implemented for low values of the real-time parameter). On the contrary, root sensors require a method to temporarily store packets while waiting for the communication channel to become available. From the perspective of measurements received by the edge server, a larger buffer is advantageous as it allows more packets to wait, but it also increases the cost for each sensor. Conversely, the smallest buffer size is more cost-effective but results in a higher number of discarded measurements. To ascertain the most favorable buffer size, both the aforementioned aspects and the duration each packet spends in the queue should be considered, particularly for systems with a robust real-time requirement.

Wired communication channels possess a crucial KSP in the form of the average speed, which predictably influences the volume of packets delivered to the edge server. Although this KSP would be significant if larger entities depended on the channel for travel, in a modern scenario, it is more likely for systems to operate atop existing infrastructures rather than constructing new ones from the ground up.

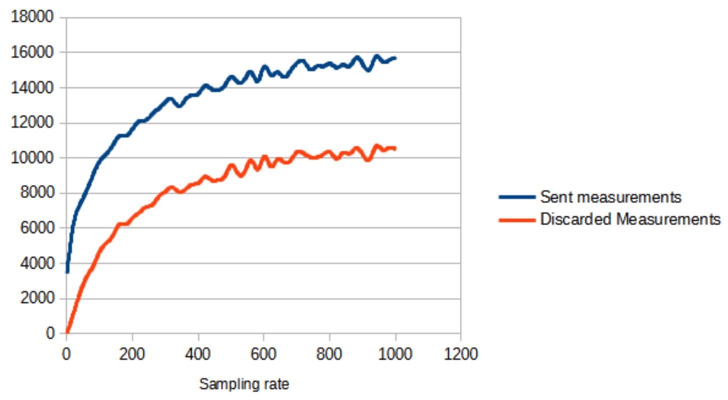
All parameters of the malicious agent can impact the KSI represented by the quantity of intercepted packages:

- Larger values of `wiretapSuccess` increase the probability of the attacker successfully connecting to a communication channel.
- Larger values of `eavesdropSuccess` increase the probability of the attacker successfully intercepting a measurement.
- Larger values of `ability` decrease the amount of time the attacker has to wait before attempting to connect to another communication channel.

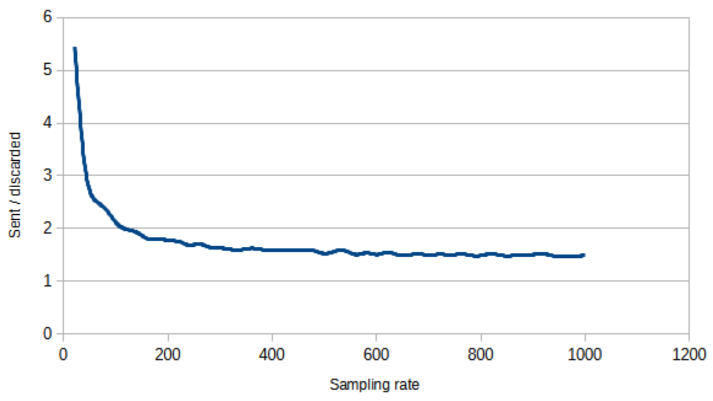
While setting `eavesdropSuccess` to one might be an option, it's reasonable to assume that not all packets traveling on the selected communication channel will be intercepted. This is primarily due to the speed of the communication channel; there's a possibility that the packet travels too quickly, and what the attacker intercepts is an empty object. Therefore, the speed of the communication channel should also be a system parameter influencing the system indicator related to the malicious agent.

Experimental analysis

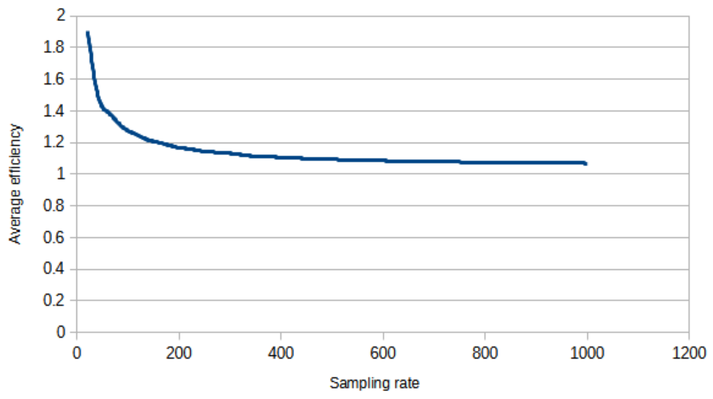
Regarding root sensors, sent and discarded measurements exhibit a consistent behavior across different sampling rates, as both follow an exponential distribution. The distinction lies in their magnitude: we would like discarded measurements to be even fewer than what the following plot shows, an option could be to introduce a parameter for the power-saving threshold and see how root sensors behave.



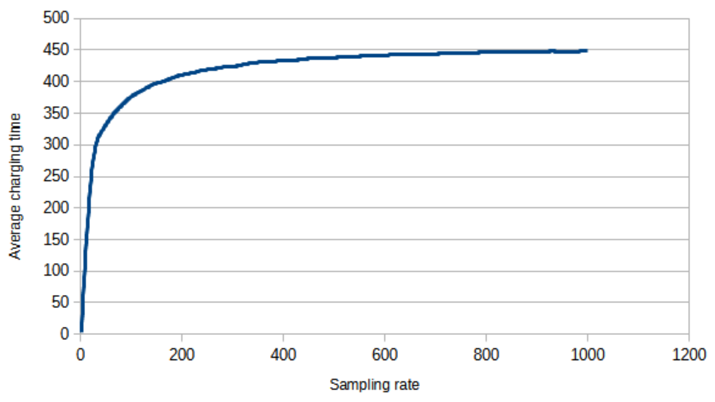
Lets check another measure, that is the ratio of sent packets over discarded ones.



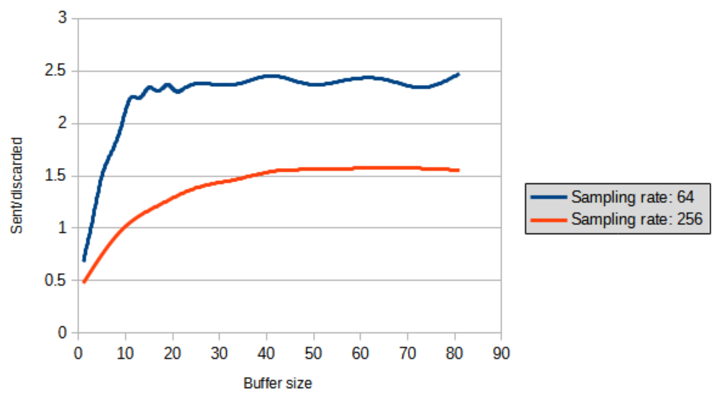
This plot is very similar to that of the efficiency measure, defined as the running time of the experiment over the average time root sensors are charging.



Finally, we can consider the average charging time per sensor.

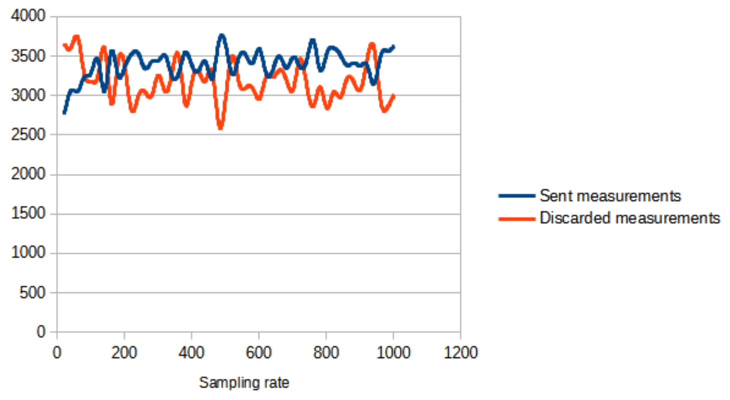


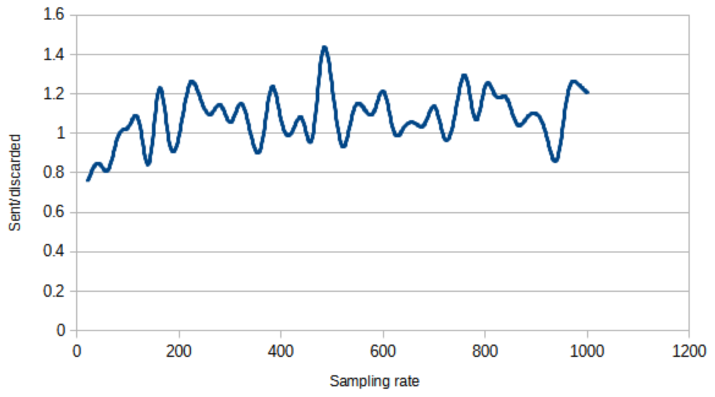
What these plots say when considered together is that there is a good value of sampling rate in range [1, 100] allowing us to balance between the amount of produced measurements and the overall root sensor efficiency.
Lets consider what happens for varying buffer sizes.



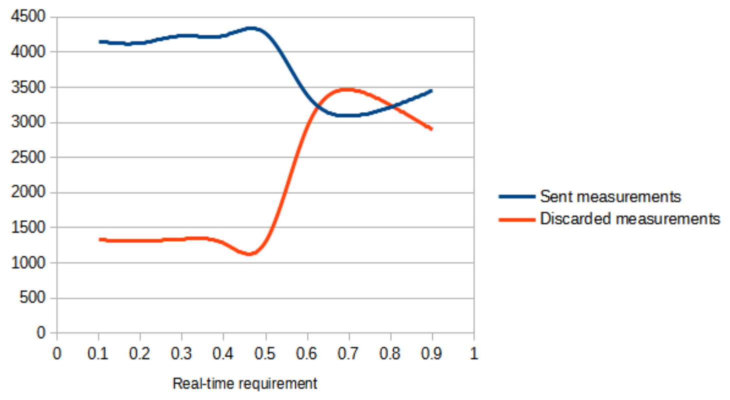
The chart tells us that, for a relatively small sampling rate, we can fix the buffer size at 20 and retain the same ratio of sent over discarded measurements. Unsurprisingly, for larger sampling rate the distribution reaches a plateau a bit later, around a buffer size of 40.

The distributions of sent and discarded measurements are considerably different for radio sensors: this happens because not only one every two measurements is discarded in power-saving mode (just like root sensors), but also when the associated root sensor is not running and the real-time requirement is large.



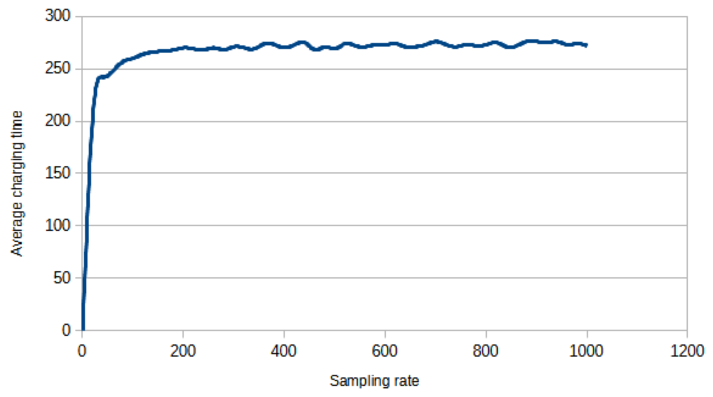


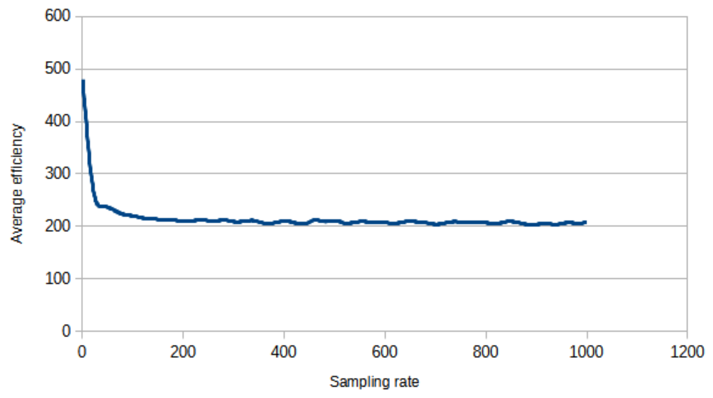
Lets see what happens for fixed sampling rate and varying real-time requirement.



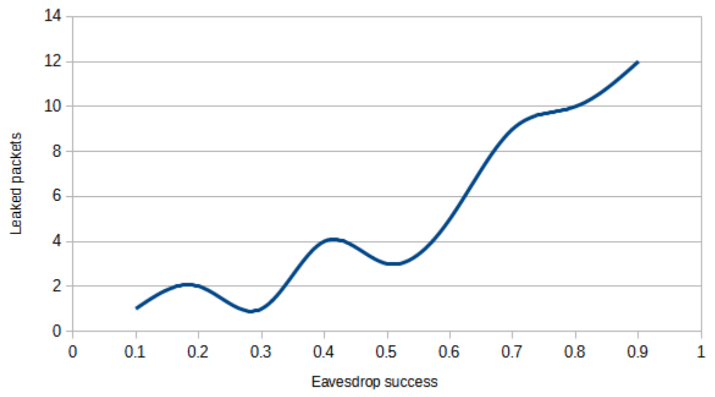
Obviously, more measurements are discarded for large real-time requirement. 0.6 is the threshold that was used to determine if a real-time requirement is large or small.

The average charging time and efficiency of a radio sensor are not so different from root sensors'.





Finally, we briefly consider the malicious agent: with increasing values of `eavesdropSuccess`, the attacker is able to fetch a higher number of packets, although there are some oscillations.



Conclusions

We discussed the creation of a simulation model in AnyLogic for a sensor network system with root and radio sensors connected to an edge server. The system involves data measurement, transmission, and potential interception by an adversarial agent. Key parameters such as sampling rate, buffer size, and communication channel speed were considered, along with balancing trade-offs between maximizing measurements and minimizing downtime. The simulation is designed with a discrete events paradigm, capturing the dynamic nature of the system's behavior and providing insights into communication strategies, network topologies, and security resilience.

Problems and improvements

An issue related to root sensors is that measurements coming from their children bypass the `measurementQ` component and are sent directly to the communication channel. This happens because, with the current implementation, measurements native to root sensors are actually generated in `measurementSink`, meaning that if we relied on the same structure we would lose the actual content of all measurements coming from the children. A possible solution to this is creating a variable that contains the measurement to send next: the variable is populated in the `collectingData` and `managingChildrenData` states, and its content is sent to the channel in `measurementSink`.

Given the way it was implemented, the `realTimeRequirement` parameter simply acts as a Boolean value: if it is larger than or equal to 0.6 (and the associated root sensor is not running), the radio sensor discards the measurement; otherwise it sends it. An alternative to this approach, that actually takes advantage of the whole domain for the parameter, is to use it as argument of a Bernoulli random variable: we discard the measurement when the root sensor is offline and the random variable returns zero.
Another issue related to radio sensors is that the first one in the population never manages to complete the two-phase connection.
Measurements produced by radio sensors are immediately sent to the association root sensor, or to a sibling. Instead, they should go through a wireless communication channel, characterized by a lower speed and a lower reliability than wired communication channels. This means radio packets can also be lost while traversing a wireless channel.

Finally, whenever they start transmitting a packet to the edge server, wired channels also send a message to the attacker: this obviously should not happen, but implementations that leverage events or query directly the communication channel for its current state did not work.