AIMM

Newsletter

July 2022

System Simulations on AI for Network Operation & Management

Executive Summary

AIMM is a two-year CELTIC-NEXT European collaborative research and development project targeting performance improvements and efficiency dividends for 5G and beyond Radio Access Network (RAN), through advanced antenna array (Massive MIMO) and Reconfigurable Intelligent Surface (RIS) technologies, powered through and managed by the latest advancements in Artificial Intelligence (AI).

This newsletter provides an outline of the simulation principles and frameworks which are being considered within the AIMM Work Package 5 on AI for Network Operation and Management.

Introduction

The objectives of the AIMM project are to enhance the performance of beyond 5G Radio Access Network (RAN), via developing novel algorithms based on Artificial Intelligence (AI) and Machine Learning (ML) techniques. The AIMM project addresses two aspects of AI in the RAN. The first, “bottom-up” approach, is to use AI to optimise the air-interface performance and enable the practical implementation of antenna structures and network architectures. The second, “top-down” approach, is to incorporate big-data management features coupled with AI functionalities to facilitate RAN intelligence and automation at the system level.

Work on the top-down approach has leveraged on simulation techniques to enable the life cycle of AI model development. These are obviously the only reasonable approach, given the complexities of running experimental algorithms on live networks at initial stages of the development cycle. A decision was made early in the project to develop code for a completely new simulation platform.

System-level 5G network simulation

Recent developments in the field of AI/ML provide new capabilities of generating automated solutions for network management functions. Specifically, Reinforcement Learning (RL) is an approach for dynamically controlling and solving Markov Decision Processes. An RL intelligent agent learns to make sequential decisions by interacting with the environment. Other options include neural network and deep learning methods. To gather information and train any of these intelligent agents, it is necessary to have the existence of an accurate simulation framework of radio network management functionalities, activities, processes and use cases.

AIMM Sim – general design considerations

AIMM Sim is a system-level simulator which emulates a full cellular radio system following 5G concepts and channel models. The intention is to have an easy-to-use and fast system simulator written in pure Python with minimal dependencies. It is especially designed to be suitable for interfacing to AI engines such as ‘TensorFlow’ or ‘pytorch’, and it is not a principal aim for it to be extremely accurate at the level of the radio channel. For the latter task, pre-computed look-up tables (based on simulated channel models) are used to obtain fast run-times. If a more precise link-level model is required, a simulator such as ns-3 can be used.

The code has a structure as shown in Figure 1.

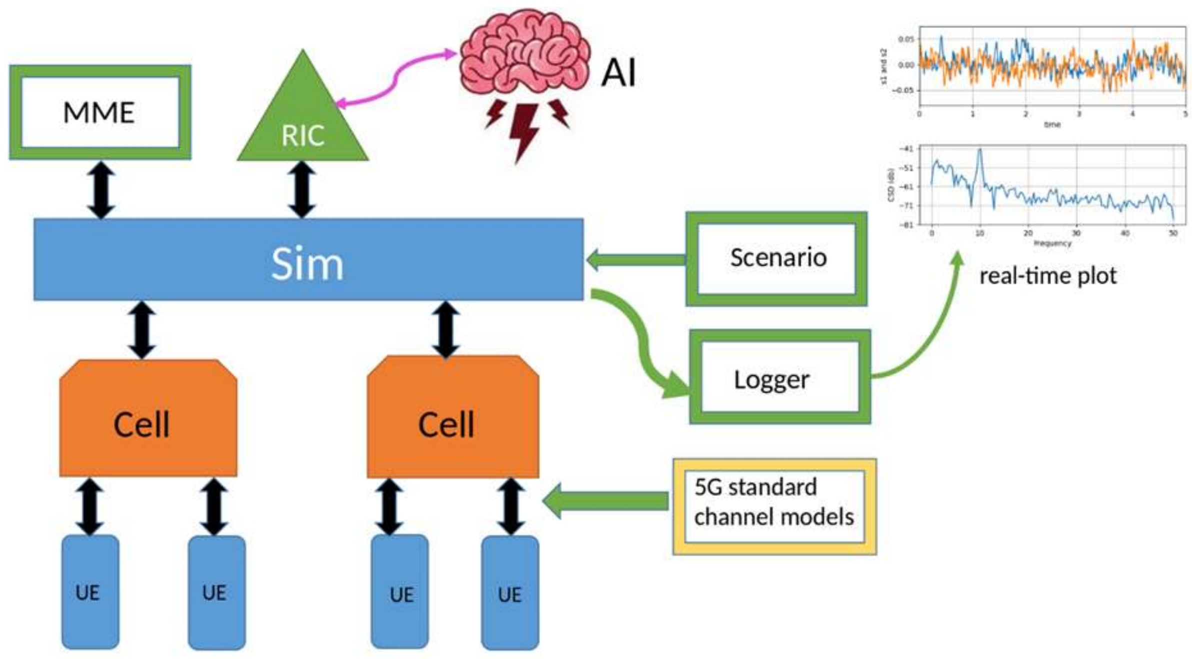


Figure 1 AIMMSim Block Structure.

AIMM Sim – detailed design

The following factors have influenced the overall software architecture:

1) The software architecture should closely mimic the real system, with a class for each type of network component.

2) The components should exchange traffic in a similar way to the real system. However, “traffic” here is an abstraction; there is no concept, for example, of IP packets, or of resource blocks at the physical layer. These constraints are imposed to get sufficient speed from the simulator, to get as many ML training episodes in a given time as possible.

3) There should be a RAN Intelligent Controller (RIC) module, at the top level of management. The AI or ML methods will operate solely in the RIC, effectively as xApps and rApps.

4) The simulation technique should be the discrete-event method. In the core of the simulator, a queue of pending events is maintained. Most events will be periodic (such as UE reporting), and an easy-to use framework is provided for this. The discrete-event method has negligible overheads and allows easy mapping from simulated time to real time.

5) Sub-banding (division of the channel into sub-channels which may be dynamically reallocated between cells) is implemented on all Cell objects, but the number of sub-bands may be set to 1, effectively switching off this feature.

6) All simulations take place in three spatial dimensions, for example, to allow modelling of high office buildings. Some simple capabilities for accounting for wall losses in indoor scenarios are provided.

7) Dynamic features of a specific simulation are handled by a Scenario class. This can, for example, move users according to some mobility model.

8) UE handovers between cells will be handled internally by a heuristic based on received signal reference power (RSRP), as in a real system. This is

implemented in the MME class. class. However, for research into smart or AI-based handover strategies, this default heuristic can be overridden.

9) In fact, all modules can be overridden or have their default behaviour modified if desired, using the usual subclassing technique.

Software design considerations

The following factors influenced the software design:

1) The core simulator should be monolithic (meaning that only one import will be needed by applications), but will not implement plotting or post-simulation analysis. These can be done better by existing tools.

2) The output of a simulation run will be a logfile in a standard format (by default, tab-separated columns). The lines in the logfile are constructed and formatted by an instance of the Logger class.

3) For testing and debugging purposes, a real time plotter is provided as a separate program. This reads and plots the logfile as it is generated, through a shell pipeline.

4) Python was chosen for portability, ease of development, and ease of interfacing to existing AI software.

5) Extensive use of Numerical Python (numpy) means that most of the code is running at the level of compiled C code. Sufficient speed is thus attained.

6) External dependencies are kept to a minimum; essentially the only one is simpy to handle the event queue, but little of its capabilities are in fact used, and simpy could easily be replaced by a small local module.

7) Sensible defaults are provided for all system parameters, such as operating frequency, channel bandwidth, etc.

8) Implementations are provided for several 3GPP standard channel models.

9) Extensive online documentation, with a full set of tutorial examples, is provided at https://aimm.celticnext.eu/simulator/.

Outline of usage principles

The basic steps required to build and run a simulation are:

1) Create a Sim instance, as it represents the complete simulation.

2) Create one or more cells with make_cell(). Cells are automatically given a unique index, starting from 0.

3) Create one or more UEs with make_UE(). UEs are automatically given a unique index, starting from 0.

4) Attach UEs with the method attach_to_best_cell().

5) Create a Scenario, which typically moves the UEs according to some mobility model, but in general can include any events which affect the network.

6) Create one or more instances of the Logger class.

7) Optionally create a RIC, possibly linking to an AI engine.

8) If necessary, create a custom Logger class by subclassing.

9) Start the simulation with sim.run().

10) Plot or analyse the results in the logfiles.

A complete simulation code demonstrating these principles is in Figure 2.

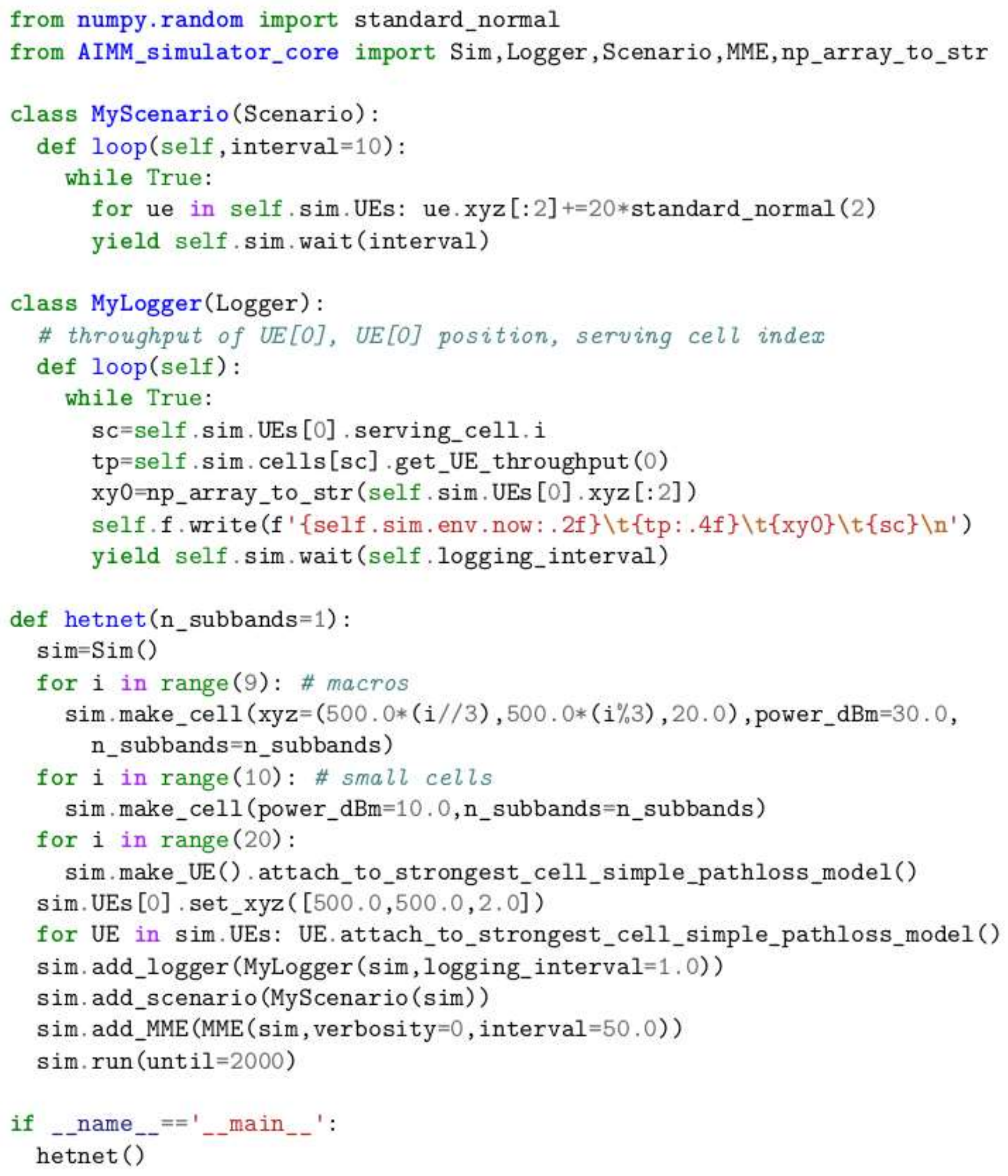
An indoor use-case example

As a meaningful demonstration of the AIMM simulator in application to an indoor use-case, we consider the open-plan office with partitions shown in Figure 3.

The aim in this example is to have ML agent learn to control the transmit power of two or more indoor small cells, in such a way as to react to changes in UE location. The hope is that by suitably setting the powers, the impact of interference on throughput will be minimized. A very important question concerns the choice of an objective function, or (in ML applications), the reward function. We have used a technique to compute the distribution of throughput across all UEs, and then use the performance of the lower 25% quantile users, as reference for decision making.

The user mobility model was the one we called “wave”. Here the users start with a uniform distribution over the building, but gradually all move to one end of the building, and then back to uniform. This cycle repeats indefinitely. The intention is to provide the ML agent with experience on extremes of user distributions, from completely uniform to highly non-uniform.

The results shown in Figure 4 demonstrate the transmit power (bottom red curve) being dynamically controlled by the Q-learning agent. The objective (light blue curve in top graph) is being held at a relatively constant level as the UEs move between extremes of high and low density.



```python
from numpy.random import standard_normal
from AIMM_simulator_core import Sim,Logger,Scenario,MME,np_array_to_str

class MyScenario(Scenario):
  def loop(self,interval=10):
    while True:
      for ue in self.sim.UEs: ue.xyz[:2]+=20*standard_normal(2)
      yield self.sim.wait(interval)

class MyLogger(Logger):
  # throughput of UE[0], UE[0] position, serving cell index
  def loop(self):
    while True:
      sc=self.sim.UEs[0].serving_cell.i
      tp=self.sim.cells[sc].get_UE_throughput(0)
      xy0=np_array_to_str(self.sim.UEs[0].xyz[:2])
      self.f.write(f'{self.sim.env.now:.2f}\t{tp:.4f}\t{xy0}\t{sc}\n')
      yield self.sim.wait(self.logging_interval)

def hetnet(n_subbands=1):
  sim=Sim()
  for i in range(9): # macros
    sim.make_cell(xyz=(500.0*(i//3),500.0*(i%3),20.0),power_dBm=30.0,
      n_subbands=n_subbands)
  for i in range(10): # small cells
    sim.make_cell(power_dBm=10.0,n_subbands=n_subbands)
  for i in range(20):
    sim.make_UE().attach_to_strongest_cell_simple_pathloss_model()
  sim.UEs[0].set_xyz([500.0,500.0,2.0])
  for UE in sim.UEs: UE.attach_to_strongest_cell_simple_pathloss_model()
  sim.add_logger(MyLogger(sim,logging_interval=1.0))
  sim.add_scenario(MyScenario(sim))
  sim.add_MME(MME(sim,verbosity=0,interval=50.0))
  sim.run(until=2000)

if __name__=='__main__':
  hetnet()
```

Figure 2 AIMM Sim complete code example.

Conclusions & future work

The AIMM system-level simulator allows easy construction of large-scale 5G network simulations, with a clean interface (through the RIC class) into standard AI software packages. Because the RIC class has privileged access to internal cell data, as well as permission to change settings operating parameters in cells, it is the right place to run any AI or ML components.

Furthermore, current developments such as implementing xApps and rApps with communication via Google 'protobuf' can be accommodated by putting a simple translation layer in the RIC. Thus, the current design is essentially agnostic regarding messaging protocols.

Current enhancements being planned include a tracking of energy consumption in each network component, allowing use in green radio projects. At the completion of the AIMM project in September 2022, it is intended to release the code as open source.

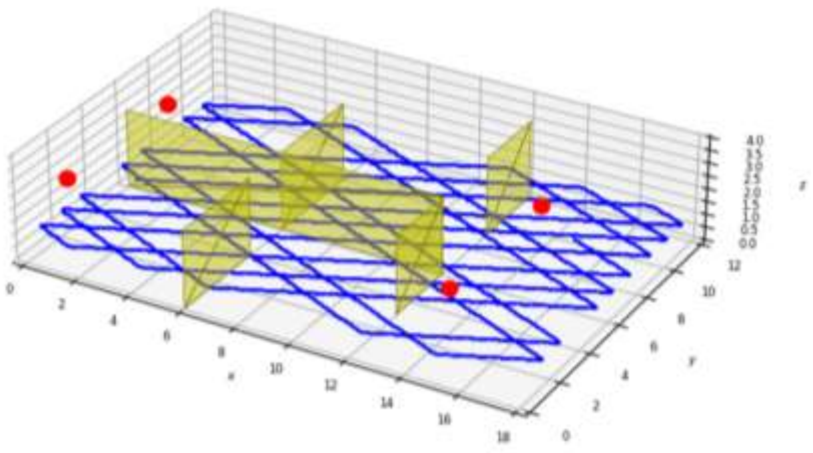


Figure 3 Example open-plan office: six rooms with partitions.

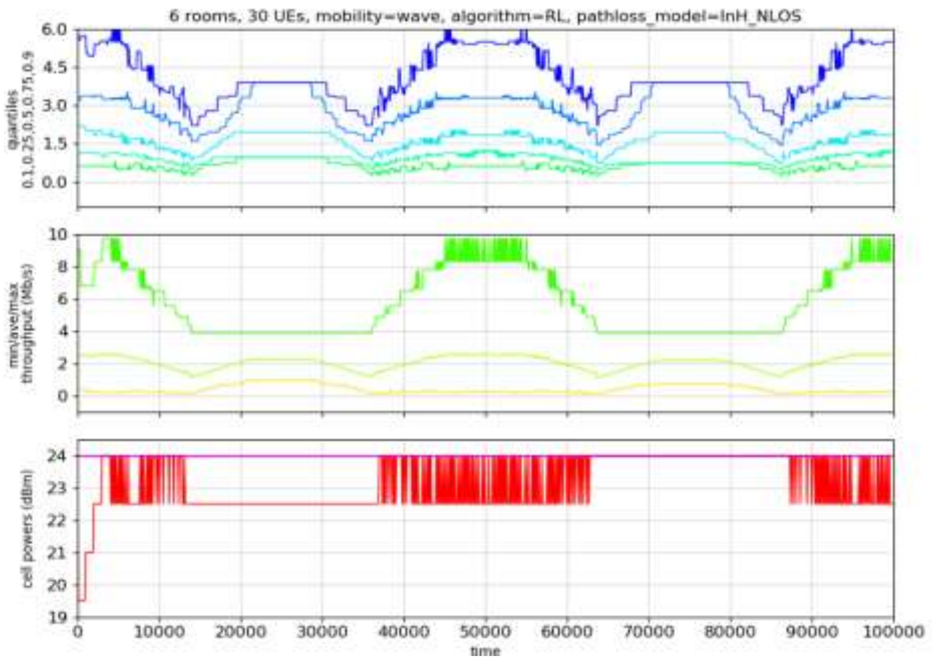


Figure 4 Four-cell indoor scenario with 3GPP InH NLOS propagation model, results from Q-learning.

BT

CEMWorks

IMST

interdigital

Loughborough University

NOKIA

thinkRF

University of BRISTOL

Universität Stuttgart

Vilicom signalling the future

Bundesministerium für Wirtschaft und Energie

UKRI Innovate UK

Canada NRC·CNRC