Journal of Ubiquitous Systems & Pervasive Networks
Volume 18, No. 2 (2023) pp. 69-76

Enhancement of the TSCH-Sim Simulator via Web Service Interface to Support Co-simulation Optimization

Tarana Ara [a], Aida Vatankhah [a], Ramiro Liscano [*a]

[a] *Ontario Tech University, Oshawa, Canada, L1G 0C5*

Abstract

Co-simulation is an important concept in the optimization of computer networks because a typical optimization scenario integrates an optimization algorithm with a network simulator. In many cases optimization algorithms are implemented in the MATLAB environment while network simulators are implemented as stand alone applications. In this paper we present enhancements to the TSCH-Sim network simulator in order to facilitate its integration with an optimization algorithm. The core enhancement is the definitions and implementation of a set of REST APIs for TSCH-Sim that allows a remote optimization algorithm to set the network configuration, routes, and 802.15.4e TSCH schedule of a sensor network. The significance of the REST API is demonstrated through the integration of a Differential Evolution based TSCH scheduling optimizer executing in MATLAB leveraging the TSCH-Sim simulator through the REST APIs in order to find a TSCH schedule that maximizes throughput.

Keywords: *802.15.4e TSCH, TSCH-Sim, REST-API, Co-simulation, DE Optimization, MATLAB Simulation*

1. Introduction

Nowadays, one of the most trustworthy MAC protocols for industrial IoT applications is the amended IEEE 802.15.4e Time Slotted Channel Hopping (TSCH) standard [1] and the IETF 6TSCH [2] initiative that supports IPv6 over 802.15.4e. The TSCH protocol is targeted toward applications that require high reliability and predictability by combining the time synchronization and channel hopping approaches to ensure high throughput, low latency, and energy efficiency. The multi-channel technique of TSCH supports several simultaneous transmissions, which results in an overall increase in network capacity and reduced delay. Furthermore, the channel hopping mechanism assists in mitigating the multi-path fading by constantly switching between multiple frequency channels [3]. Consequently, it improves the network's reliability and lowers the energy consumption that might require for data re-transmission.

The Key functionality of the TSCH protocol is the TDMA transmission, reception, and idle schedule and the standard does not specify the use of any particular scheduler allowing manufacturers to implement and develop the best schedule for their application. On the other hand, the 6TiSCH standard has adopted the 6TiSCH minimal schedule. Recently, several scheduling algorithms have been proposed for TSCH [3–8] with the goal of creating an optimized and reliable scheduling algorithm. Within our research lab, we have also developed an optimization algorithm for the TSCH schedule based on Differential Evolution (DE) optimization using the MATLAB platform.

In order to optimize a TSCH schedule one has to integrate an optimization algorithm with either a real network or a simulated world. In terms of simplicity and cost-effectiveness the use of simulation environments is commonly used for designing and validating complex systems systems such as the simulation of car pooling agents [9], traffic data collection [10] and security in wireless networks [11], because of the flexibility in creating scenarios for these systems and performing analysis on these different scenarios.

Figure 1 is a sequence diagram that reflects a typical integration between an optimization algorithm and a TSCH simulator that captures the scenario of the optimization of a TSCH schedule. In our work the optimization algorithm is developed in the MATLAB

[*]Corresponding author. Tel.: +19057218668
Fax: +11111111111; E-mail: rliscano@ieee.org

DOI: 10.5383/JUSPN.18.02.003

simulation environment while the TSCH simulator is TSCH-Sim [12].

In order to achieve this co-simulation environment several modifications were done to the TSCH-Sim simulator that include the static configuration of routes and TSCH schedules as well as the definition of a set of REST-APIs for TSCH-Sim.[1] Details of these modifications are presented in this paper along with an example of a co-simulation scenario where a TSCH schedule is determined that maximizes throughput for a heterogeneous sensor network leveraging a DE optimization algorithm.

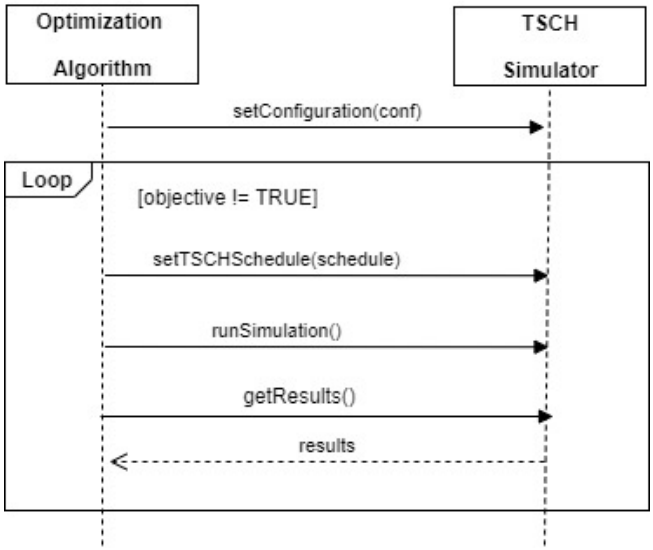


Fig. 1: Co-simulation between TSCH-SIM and optimizer sequence diagram

The remainder of this paper is organized as follows: Section 2. is a discussion of recent works in TSCH scheduling and their respective validation tools. The section 3. presents a background study on TSCH scheduling with a simple example. Section 4. describes the additional features added to TSCH-Sim to support manual schedules and static routes and the REST API used to integrate with other simulators like MATLAB. Section 5. presents the TSCH schedule co-simulation optimization strategy that integrates the DE optimization algorithm executing in MATLAB with TSCH-Sim. In section 6. a dense network example is shown that leverages the co-simulation architecture to find a TSCH schedule that maximizes the throughput. Finally, Section 7. concludes the paper.

2. Related work

To the best of our knowledge, we found only three open-source simulators that support TSCH network protocol during the investigation. Those are Cooja [14], 6TiSCH simulator [15], and TSCH-Sim [12]. However, none of these can be utilised in their current implementation as they do not support manual configuration of the routing and scheduling information and do not have an API interface that allows for the integration with an optimization algorithm.

As mentioned in Section 1. there have been several works that investigated different TSCH scheduling algorithms [3–8]. In this paper, we are more interested in which particular simulators were used to validate the schedule. Table 1 is a summary of the papers and the approach used by the researchers to validate their algorithm.

Table 1. TSCH scheduling papers and their respective validation environments

Reference	Validation Environment
[16]	Hardware implementation
[5]	TSCH Simulator
[6]	Hardware implementation
[7]	Cooja
[8]	Hardware Implementation
[17]	Hardware Implementation
[18]	Custom Python simulator (defunct)
[19]	NS3 (defunct)
[20]	Custom Python simulator (defunct)
[21]	6TiSCH simulator

From this review, one can see that many papers have TSCH implementations in actual sensor hardware. This code is custom code and is difficult to work from although it is a better validation of the scheduler than using a simulator. Such implementations are subject to a limited number of sensors and a lack of several re-configurations.

Even though there is some work that utilized the popular NS-3 network simulator the official site of NS-3 does not have any support for TSCH. This is also the situation for OMNeT++, which is another popular network simulator. It does not include support for the TSCH protocol.

OpenSim is the simulator associated with the OpenWSN [22] library, which is the defacto standard TSCH and 6TiSCH implementation for embedded devices. OpenSim though is more of an emulator than a simulator allowing the running of the OpenWSN firmware on the emulator prior to deploying it on the actual hardware. It is not really a simulator that can be easily used to create and test a TSCH schedule.

3. Background on TSCH

TSCH utilizes both time synchronization and frequency hopping techniques in the MAC layer to support more reliable and high-performance transmission. In TSCH, time is divided into a fixed-duration slot called a time slot. A slot frame is made up of a defined number of those time slots. To synchronize all the nodes in a network, the slot-frame repeats periodically. In general, each time slot's duration in a frame ranges from 10ms to 20ms to deliver a data packet and receive an acknowledgment between a pair of nodes.

Another mechanism in the TSCH network is known as channel hopping. This approach's goal is to change the transmission channel according to a known pattern constantly. With the

[1] This paper enhances the work published in [13], by introducing REST-APIs as a web interface to enable to TSCH-Sim simulator to coordinate with other simulator.

advantage of multiple channels, it allows more nodes to exchange their packet simultaneously but using different channel offsets. According to IEEE 802.15.4e, there are 16 channels available for communication [23]. A channel offset represents each channel. In TSCH, a schedule is a matrix of channel offset and time slots that represents the cell. A cell located at the intersection between a defined row and column (i.e., channel offset and time slot offset) of that matrix describes a link between neighbor nodes at the data link layer. A cell can be shared between multiple nodes or dedicated. A shared cell allows two or more nodes to transmit simultaneously, whereas a dedicated cell is bound to only one transmission, yielding a contention-free message.

Each time slot in a network is identified by an Absolute Slot Number (ASN). At the beginning of the network, ASN initialized to 0 and gradually increased by one at each time slot. The physical channel of a shared or dedicated cell can be calculated as

$$f = F[(ASN + Channel\_Offset)\%N_{channels}] \qquad (1)$$

where F is a lookup table that contains a sequence of available physical channels and $N_{channels}$ denotes the total number of available channels.

From figure 2a, we can see a simple tree topology with five nodes and a slot frame of length four with four-time slots and channel offsets. Thereby, there are 16 cells in this slot frame. Each cell in the TSCH slot frame is considered half-duplex. This cell specification implies that a node cannot transmit and receive or only receive from multiple nodes at the same timeslot, even though it sticks with a different channel offset. This term is known as collision and is depicted in figure 2b. In this figure, at slot 4, node two has transmitted and received packets from two different nodes. Again at slot 1, both nodes send a packet to the same destination. Both of these scenarios result in a packet collision.

4. Modifications to TSCH-Sim

Even though several specific examples of TSCH scheduling are available in TSCH-Sim [12] there was no mechanism implemented to specify a static TSCH schedule with the author choosing to "hard" code a scheduling algorithm into the corresponding scheduler module. This was also the case for routing, where several common routing algorithms were implemented, such as `RPL` but no method for setting a static route.

The core architecture of TSCH-Sim comprises a number of loosely structured components conceptually grouped into the following three layers: interface, network, and device/link. It is implemented in modular JavaScript using classes and code separation in files representing the key software components. A class diagram of the original code and the additional classes we developed is shown in figure 3. The lowest layer is the data-link layer that performs the routing and scheduling protocols, maintaining the IEEE 802.15.4e amendment. We have modified that layer to support Manual scheduler and Static routing. Some changes are also done in the configuration file to reduce the path dependency required to read the route and schedule file. The intermediate is the network layer responsible for link connectivity, radio propagation model, transmission range, and packet generation. Finally, the highest layer is the interface that implements the global configuration, building connections between all required processes, time synchronization, and logging file to visualize the outputs.

```
{"SOURCE":2,"DESTINATION":1,"TS":1,"CO":1};
{"SOURCE":3,"DESTINATION":1,"TS":2,"CO":2};
{"SOURCE":4,"DESTINATION":2,"TS":3,"CO":1};
{"SOURCE":5,"DESTINATION":2,"TS":4,"CO":2};
```

Listing 1: TSCH schedule JSON configuration example

4.1. The TSCH Manual Scheduler Module

A new manual scheduler module, `"scheduler_manual.mjs"`, was implemented that supports the specification of a schedule in a JSON configuration file. The new module was a copy of the `Leaf-Forwarder` module with the addition of a function to read the schedule from a configuration file and add the schedule to the corresponding nodes.

Defining a schedule manually requires declaring a time slot and channel offset that a packet can be transmitted. A schedule was specified in the `"schedule.json"` file and a sample of the `"schedule.json"` file is depicted in Listing 1 for the 5 node topology shown in figure 2a. In this figure, the `SOURCE` and `DESTINATION` specify the `id` of the current node and the `id` of the next-hop node. `TS` and `CO` correspond to the time slot and channel offset that the packet is to be transmitted and received on for the respective transmitting and receiving nodes. To be able to use this manually specified schedule we defined the new scheduling approach as `"ManualScheduler"` and modified the code in the `"Node"` module to refer to the manual scheduler module if the TSCH-Sim main configuration file contains the `"SCHEDULING_ALGORITHM": "ManualScheduler"` line.

```
{"NODE_ID":2,"DESTINATION_ID":1,"NEXTHOP_ID":1};
{"NODE_ID":3,"DESTINATION_ID":1,"NEXTHOP_ID":1};
{"NODE_ID":4,"DESTINATION_ID":1,"NEXTHOP_ID":2};
{"NODE_ID":5,"DESTINATION_ID":2,"NEXTHOP_ID":2};
{"NODE_ID":5,"DESTINATION_ID":1,"NEXTHOP_ID":2};
```

Listing 2: TSCH ROUTE JSON configuration example

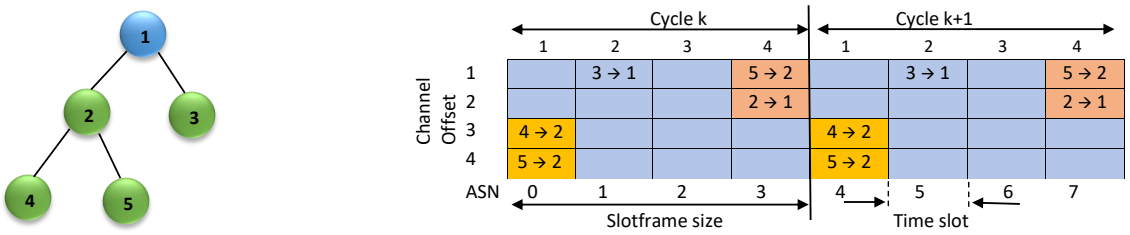
4.2. The Static Routing Module

For our experiments, we want to implement static routes so as to properly evaluate the performance of our scheduler. A new module, namely `"routing_manual.mjs"` was implemented and integrated into the simulator. This new module was based on the `"routing_null.mjs"` which contains empty functions that can be populated for particular routing algorithms. For static routes, this requires that a routing configuration file `"route.json"` be read that specifies the static routes and adds these to each of the nodes routing tables. An example of `"route.json"` file for a tree network of five nodes is depicted in the Listing 2. This example identifies the next-hop node `ID` for a packet coming from a source node `ID` to a particular destination `ID`. To enable static routing the TSCH main configuration file must contain the statement `"ROUTING_ALGORITHM": "ManualRouting"` in it.

4.3. TSCH-SIM REST APIs

4.3.1. REST-API Framework

An API (Application Programming Interface) is a set of definitions and protocols that acts as a mediator to enable two different systems or software to interact. The API architecture



(a) (b)

Fig. 2: (a) A tree network with 5 nodes; (b) Scheduling process in IEEE 802.15.4e TSCH.

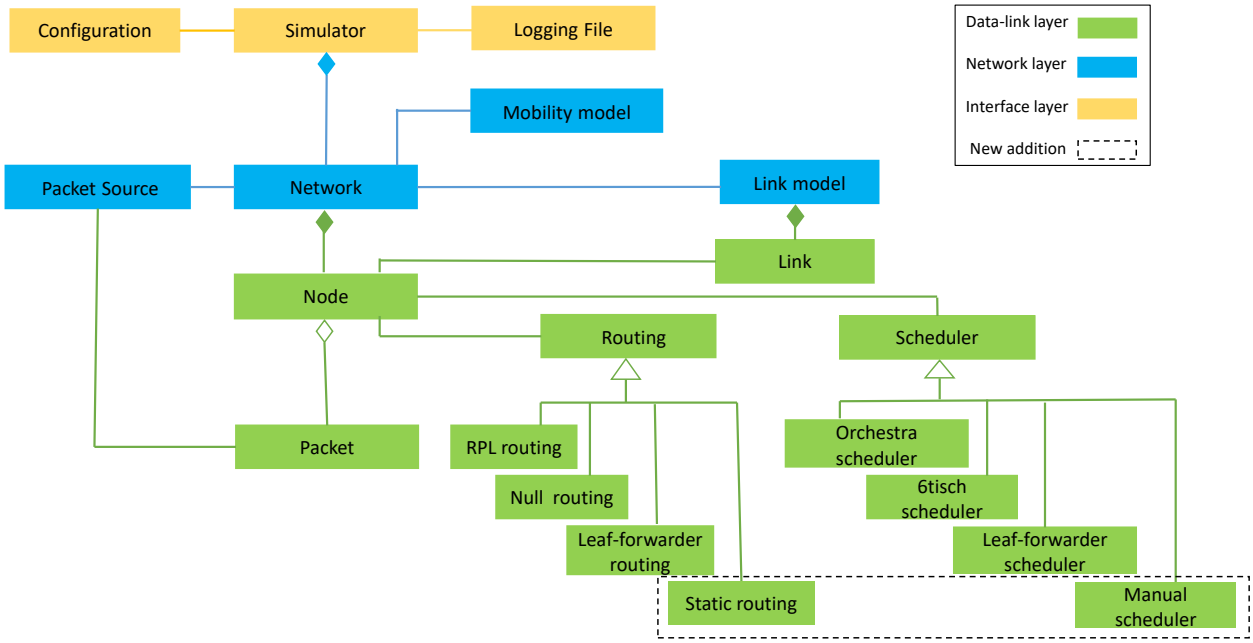


Fig. 3: TSCH-Sim class diagram showing original and additional new classes

is generally explained in terms of a client and server interaction model. Any software program invokes a request to the API to get some information within the application is known as a client. On the other hand, a server is another program that the application uses, receives client requests, and responds back to the desired resources. Resources can be any piece of information provided by the server. The core advantage of the RESTful API is it keeps both machines conveniently independent by simplifying communication between client and server. This way, a client system can grow smoothly with no dependency on the server, and the server application can be modified without affecting the client. One of the most popular types of API is REST or RESTful API. It stands for Representational State Transfer, which means that the server transfers back the resource in a standardized representation whenever a client requests a resource using REST API [24]. This representation is delivered through HTTP in several formats: XLT, JSON, HTML, or plain text. Among these, JSON is the most popular file format as it is readable by humans and machines. Figure 4 illustrates a generic interface using REST-API between a client and server machine. For our experiment, a lightweight API communication has been proposed and implemented between the server and client devices, slightly reducing resource-sharing overheads. Calling a RESTful service over the HTTP protocol

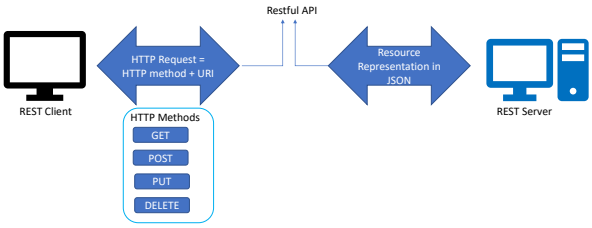


Fig. 4: Interaction between two distinct machine using REST API

can be implemented in many programming languages. For this experiment, we used NodeJS, a web server technology based on JavaScript. It used V8 JavaScript Engine with integrated module add-ons. REST APIs are designed using resources, which specifies any kind of data or object or services the client device can access. Every resource has an identifier called Universal Resource Identifier (URI) to uniquely identify the resource. It contains the name and address of this particular resource. Generally, an HTTP client manipulates a resource by connecting to the server that hosts

Table 2. TSCH-Sim REST APIs

REST API	Description
POST, *http://api/config* Requested Body:{ "SIMULATION_SEC": x "ROUTE": [ ] "SCHEDULE": [ ] ……}	Writes a configuration file to the TCH-Sim directory
GET, *http://api/config*	Retrieves the TSCH-Sim configuration file.

PUT, *http://api/MatlabSchedule*
Requested Body:
A TSCH schedule
{
"SCHEDULE": [ ]
}
	Updates a TSCH schedule in TSCH-Sim configuration file.
	Response Body: Updated Configuration file in JSON

GET, *http://api/run_simulation*	Executes the TSCH-Sim simulator.
GET, *http://api/simulation_result*	Returns the simulation results from TSCH-Sim Response Body:{ "Throughput": "Delay": .....}

it and sending an invoking method and path to the resource. In the proposed approach, we used localhost for executing the actions of the host. Three different endpoints are available to interact with the TSCH-Sim simulator for our purposes. A GET request can be made directly through the Uniform Resource Identifier (URL) string, where the POST or PUT request allows the client to provide the requested body as the input value.

In this implementation, the ***"api/config"*** endpoint takes the configuration file as a parameter and responds with a positive message if it successfully updates the configuration file. A client can view or update any input file with the same endpoint.

The ***"api/run_simulation"*** endpoint is used to run the simulator, and if any error occurs during the running phase, it throws the corresponding error to the client.

The ***"api/ simulation_result"*** endpoint returns a complete list of each node's outcome and the performance metrics: delay, throughput, and PDR values. The TSCH-Sim REST APIs those have been implemented are depicted in table 2.

4.3.2. Co-Simulation Process

Every time the server receives the HTTP request from the MATLAB Simulator, it starts processing based on the parameter and embedded data. Our local host server is always listening on port number 3000. The Initialization phase begins when the client simulator (MATLAB) sends a configuration file of a particular network topology by invoking the HTTP POST method with a valid URI (http://api/config). The file format is JSON and contains all the necessary information required to run the Simulator, such as Simulation time, Routing, Scheduling, Links, Nodetypes, Connection, and the Position of each node of the network. After retrieving the information from the endpoint, the API decodes and writes it to the TSCH-Sim simulator directory and responds with a successful message to the client simulator. A client can also show the provided input configuration file by simply calling an HTTP GET method using the same URI. After viewing the configuration file, a client can update or replace the recently provided parameters or any particular portion of this file by invoking the PUT method. Once the client sets all the parameters, an HTTP GET method is called to run the child process TSCH-Sim simulator. As soon as the Simulator finishes its running phase, the API sends the status code "200" with the corresponding output as a successful response. If the server finds any error during the running phase of the simulation, it sends a status code as "404" notifying it of an error. The output sent by the server is a JSON file containing the performance metrics such as delay, throughput, and PDR of the network.

5. TSCH Schedule Co-simulation Optimization Strategy

In this paper, the TSCH-Sim REST API was used to find an optimized TSCH schedule with the objective of maximizing the throughout of the network. We leveraged Differential Evolution (DE) for this implementation, however, any optimizer can be used instead. DE is a population-based method that performs optimization by iteratively trying to improve the candidate solution according to a given objective function.

In every generation of the DE, a new schedule will be generated which is explained thoroughly in [25]. Then, based on an objective function, the fitness value of the new candidate will be evaluated and compared to the current schedule. That is, the population of the next generation will be established by comparing the results of the objective function of the parent and child population where the individual with the best objective function output will pass to the next generation. The process is repeated until a termination criteria is met. In our work, the fitness value calculations are done in TSCH-SIM to obtain realistic values.

The DE optimization is developed in MATLAB and the TSCH-SIM REST API is used to make a connection between MATLAB and TSCH-SIM.

Figure 5 is a flowchart depicting the iteration steps of the DE optimization and interaction with TSCH-Sim via the REST APIs. The steps basically consist of the generation of a new schedule, (SCH_{New}), and populate the JSON file with the recently generated schedule and use the REST API PUT operation to set the schedule. Then using the REST API GET requests acquire the results of the simulation via a JSON file. In this particular case we are interested in the throughput value from the Result.json file. According to this value, the DE optimizer will decide on whether to terminate the optimization process or to continue it. Here, the termination criteria is set to reaching a specific optimization iteration value.

6. Example Implementation Scenario

As an example scenario we created a collection tree network topology consisting of 13 nodes connected to a root node as shown in Figure 6. In this heterogeneous network, the Packet Rates (PR) as well as the location of these sensor nodes are shown in Table 3. Packet Rate of a node shows the number of packets that has been generated in one second. For instance, a node with $PR = 0.5$ generates 0.5 packet in one second. Since nodes cannot generate

Table 3. Location and Packet rate of the 13 node tree network

ID	1	2	3	4	5	6	7	8	9	10	11	12	13
X (m)	20	22.5	17.5	17	26	22.5	17.5	22.5	20	25	14	25	28
Y (m)	25	22	22	27	22	19	19	16	14	14	19	10	13
PR (/s)	0	1	0.2	0.5	0.2	0.5	1	0.2	0.5	0.2	0.2	0.5	0.2

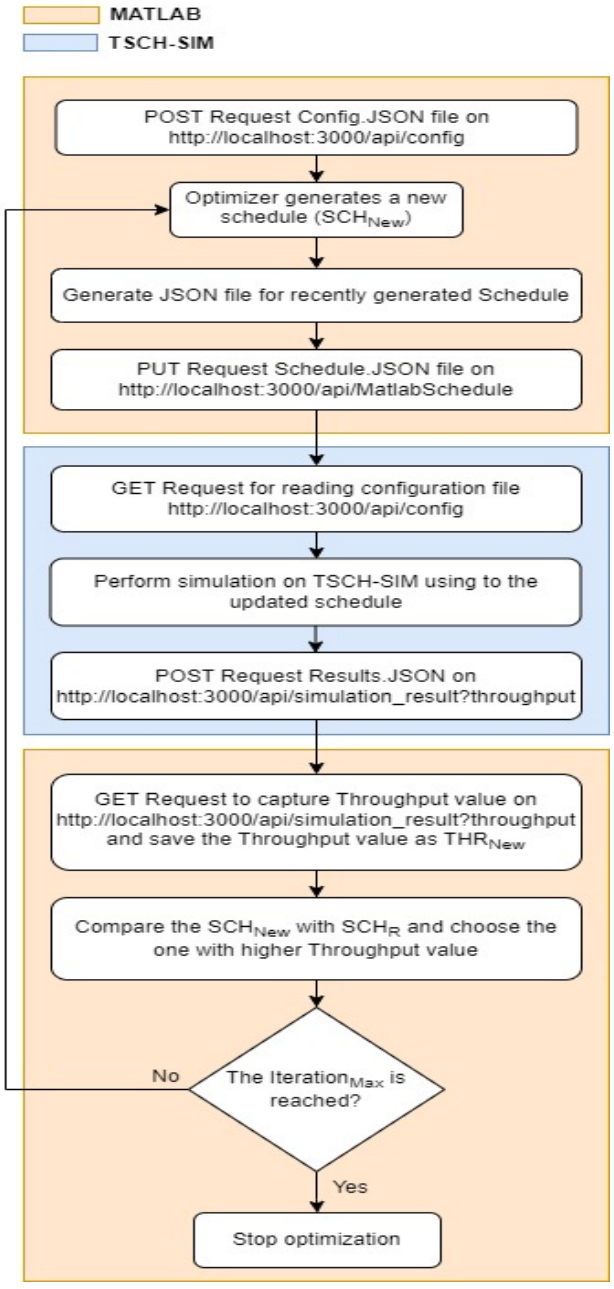


Fig. 5: MATLAB and TSCH-SIM communication diagram

a portion of a packet this would imply that a full packet would be generated every 2 seconds.

For this configuration in TSCH-SIM, we have chosen thirteen distinct node types as `"node1","node2",..., "node13"` with an `APP_PACKET_PERIOD_SEC` of 1, 5, 2, 5, 2, 1, 5, 2, 5, 5, 2 and 5 second, respectively. The position of these nodes is listed in Table 3 and the TSCH-SIM configuration file is depicted in Listing 3 for this scenario.

Table 4. Optimal Schedule with slotframe size of 12

	TS=1	TS=2	TS=3	TS=4	TS=5	TS=6
Ch=1	9,12		3,8	11	9,12	5,8
Ch=2		2,9,12	5			11
Ch=3	10,2,7		8		10,2,7	3
Ch=4		7	12	2,12		12
	TS=7	TS=8	TS=9	TS=10	TS=11	TS=12
Ch=1		3,12	10	13	9,12	12
Ch=2	3,12		2,7			4
Ch=3	8	8,6				8
Ch=4	11,2			2,7	6,3	

In the configuration file `"NODE_TYPES"`, is an object key that contains an array of different types of nodes used in the simulation. Each distinct node type must hold some crucial parameter such as the node's name and the total number of nodes (of this type) captured by the values of `"START_ID"` and of `"COUNT"` representing the starting `ID` of the first node of this type and the number of nodes of this type. The node's packet generation frequency and destination are specified with the `APP_PACKET_PERIOD_SEC` key. The positions of the nodes are specified with the `"POSITION"` key along with the `X` and `Y` positions of the nodes. The `"CONNECTION"` key indicates a connection between two neighboring nodes utilizing `"Logistic Loss"` propagation model. It is important to note that the `"CONNECTION"` specification of a network is not the same as routes and is related the network layer 2 links between nodes and a connection is required between two nodes in order for them to communicate and it must be bi-directional if `ACKs` are used at the MAC layer. The scheduling and routing are specified as manual by respectively specifying the `"ROUTING_ALGORITHM": "ManualRouting"` and `"SCHEDULING_ALGORITHM": "ManualScheduler"` keys.

We assumed a slotframe size of 12 and number of channels to be 4 for analysis. The selected slotframe size is able to include all the required transmissions in the specified time period. Since the network is heterogeneous, the number of packets generated in 12 time slots vary. The goal is to find a schedule that can meet the specified throughput using the Customized DE optimization algorithm. We kept the slotframe size static to observe the throughput optimization according to the extracted result from simulator. Matlab was used for Customized DE optimization, however, to calculate the throughput, the TSCH-SIM simulator was used to obtain more realistic results in the network. Throughput is defined as the total number of successful packet transmissions in the specified time. Our goal is to maximize this value and reach to the optimal TSCH schedule using the interaction between an optimizer and a simulator.

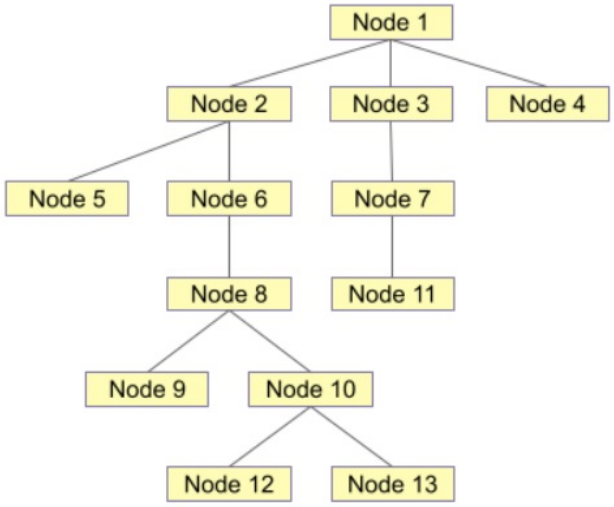


Fig. 6: A tree network with 13 nodes

The optimizer will choose between the recently generated schedule and the already existing schedule according to the fitness value of the objective function. As previously mentioned, the objective of the optimizer in this paper is to maximize the throughput of the network based on the defined data rates of the sensors. As a consequence, providing that the recently schedule has higher throughput in comparison with the existing one, the new schedule will be replaced with the existing schedule.

The optimal schedule is shown in Listing 3. The maximum iteration that was considered in this optimization is 500 and slotframe size is assumed as 12 timeslots.

```
{"SIMULATION_DURATION_SEC": 600,
 "ROUTING_ALGORITHM": "ManualRouting",
 "SCHEDULING_ALGORITHM":"ManualScheduler",
 "NODE_TYPES":[
 {"NAME": "node1",
  "START_ID": 1,
  "COUNT": 1},
 {"NAME": "node2",
  "START_ID": 2,
  "COUNT": 1,
  "APP_PACKETS":{"APP_PACKET_PERIOD_SEC":1.00,
  "TO_ID": 1}    },
 ...
 {"NAME": "node13",
  "START_ID": 13,
  "COUNT": 1,
  "APP_PACKETS":{"APP_PACKET_PERIOD_SEC":5.00,
  "TO_ID":1}}],
 "POSITIONS":[
 {"ID":1,"X":20.00,"Y":25.00},
 {"ID":2,"X":22.50,"Y":22.00},
 ...
 {"ID":13,"X":28.00,"Y":13.00}],
  "CONNECTIONS":[
 {"FROM_ID":2,"TO_ID":1,
              "LINK_MODEL":"LogisticLoss"},
 {"FROM_ID":1,"TO_ID":2,
              "LINK_MODEL":"LogisticLoss"},
 ...
 {"FROM_ID":13, "TO_ID":10,
              "LINK_MODEL":"LogisticLoss"},
 {"FROM_ID":10,"TO_ID":13,
              "LINK_MODEL":"LogisticLoss"}]
}
```

Listing 3: A sample of the configuration file for 13 nodes network

7. Conclusion

This paper presented the integration of two different services utilizing the Rest-API protocol. Our goal is to maximize the throughput of a network by determining an optimal schedule, which is generated by the DE optimization algorithm. The optimization was implemented in Matlab; however, the objective function in the optimization process depends on the value that has been extracted through TSCH-SIM. Matlab is a powerful tool for numerical computation, and TSCH-SIM is a TSCH simulator. To take advantage of both applications, we use REST-API to make a connection between these two simulators. At present, the API's endpoint responds simulation result as a whole JSON file instead of particular metrics (i.e., delay, throughput, PDR, and so on). Again if any error occurs during the simulation running phase, the API returns the 404-error status instead of providing the specific error type. In the future, we plan to add these features and the time synchronization approach to make it a more realistic co-simulation.

References

[1] IEEE Standard Association et al. 802.15.4e-2012—IEEE standard for local and metropolitan area networks—part 15.4: Low-rate wireless personal area networks (LR-WPANs). *IEEE Standard Association: Piscataway, NJ, USA*, 2012.

[2] Xavier Vilajosana, Kris Pister, and Thomas Watteyne. Minimal IPv6 over the TSCH mode of IEEE 802.15.4e (6TiSCH) configuration. *Internet Engineering Task Force RFC series*, (RFC8180), 2017.

[3] Mohamed Mohamadi and Mustapha Reda Senouci. Scheduling algorithms for IEEE 802.15.4 TSCH networks: A survey. In *International Conference on Computer Science and its Applications*, pages 4–13. Springer, 2018.

[4] Sana Rekik, Nouha Baccour, Mohamed Jmaiel, and Khalil Drira. A performance analysis of Orchestra scheduling for time-slotted channel hopping networks. *Internet Technology Letters*, 1(3):e4, 2018.

[5] Xenofon Fafoutis, Atis Elsts, George Oikonomou, Robert Piechocki, and Ian Craddock. Adaptive static scheduling in IEEE 802.15.4 TSCH networks. In *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, pages 263–268. IEEE, 2018.

[6] Huiung Park, Haeyong Kim, Kyeong Tae Kim, Seon-Tae Kim, and Pyeongsoo Mah. Frame-type-aware static time slotted channel hopping scheduling scheme for large-scale smart metering networks. *IEEE Access*, 7:2200–2209, 2018.

[7] Sana Rekik, Nouha Baccour, Mohamed Jmaiel, Khalil Drira, and Luigi Alfredo Grieco. Autonomous and traffic-aware scheduling for TSCH networks. *Computer Networks*, 135:201–212, 2018.

[8] Seungbeom Jeong, Jeongyeup Paek, Hyung-Sin Kim, and Saewoong Bahk. Tesla: Traffic-aware elastic slotframe adjustment in TSCH networks. *IEEE Access*, 7:130468–130483, 2019.

[9] Stephane Galland, Luk KNAPEN Ansar-Ul-Haque Yasar, Luk Knapen, Nicolas Gaud, Tom Bellemans, and Davy Janssens. Simulation of carpooling agents with the Janus platform. *J. Ubiquitous Syst. Pervasive Networks*, 5(2): 9–15, 2014.

[10] Johan Holmgren, Henrik Fredriksson, and Mattias Dahl. On the use of active mobile and stationary devices for detailed traffic data collection: A simulation-based evaluation. *International Journal of Traffic and Transportation Management*, 2(02):35–42, 2020.

[11] Vishal Krishna Singha, Saurabh Vermaa, and Manish Kumara. Evaluation of privacy preserving in-network aggregation for different routing structures in WSNs. *Journal of Ubiquitous Systems & Pervasive Networks*, 9 (2):15–19, 2017.

[12] Atis Elsts. TSCH-Sim: Scaling up simulations of TSCH and 6TiSCH networks. *Sensors*, 20(19):5663, 2020.

[13] Tarana Ara, Tegveer Singh, Aida Vatankhah, and Ramiro Liscano. Enhancement of the tsch-sim simulator to support manual scheduling and routing. *Procedia Computer Science*, 203:61–68, 2022.

[14] Fredrik Osterlind, Adam Dunkels, Joakim Eriksson, Niclas Finne, and Thiemo Voigt. Cross-level sensor network simulation with Cooja. In *Proceedings. 2006 31st IEEE conference on local computer networks*, pages 641–648. IEEE, 2006.

[15] Municio Esteban, Daneels Glenn, Malisa Vucinic, Steven Latré, Jeroen Famaey, Yasuyuki Tanaka, Keoma Brun-Laguna, Xavier Vilajosana, Kazushi Muraoka, and Thomas Watteyne. Simulating 6TiSCH networks. *Transactions on emerging telecommunications technologies*, 2018.

[16] Huiung Park, Haeyong Kim, Seon-Tae Kim, and Pyeongsoo Mah. Multi-agent reinforcement-learning-based time-slotted channel hopping medium access control scheduling scheme. *IEEE Access*, 8:139727–139736, 2020.

[17] Sana Rekik, Nouha Baccour, and Mohamed Jmaiel. Limitations of static autonomous scheduling for TSCH protocol and advances in adaptive scheduling. In *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 1124–1129. IEEE, 2022.

[18] Andrew Tinka, Thomas Watteyne, and Kris Pister. A decentralized scheduling algorithm for time synchronized channel hopping. In *International Conference on Ad Hoc Networks*, pages 201–216. Springer, 2010.

[19] Pascale Minet, Zied Soua, and Ines Khoufi. An adaptive schedule for TSCH networks in the industry 4.0. In *2018 IFIP/IEEE International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN)*, pages 1–6. IEEE, 2018.

[20] Atis Elsts, Xenofon Fafoutis, James Pope, George Oikonomou, Robert Piechocki, and Ian Craddock. Scheduling high-rate unpredictable traffic in IEEE 802.15.4 TSCH networks. In *2017 13th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 3–10. IEEE, 2017.

[21] Seema Kharb and Anita Singhrova. Slot-frame length optimization using hill climbing for energy efficient TSCH network. *Procedia computer science*, 132:541–550, 2018.

[22] Thomas Watteyne, Xavier Vilajosana, Branko Kerkez, Fabien Chraim, Kevin Weekly, Qin Wang, Steven Glaser, and Kris Pister. OpenWSN: a standards-based low-power wireless development environment. *Transactions on Emerging Telecommunications Technologies*, 23(5): 480–493, 2012.

[23] Rodrigo Teles Hermeto, Antoine Gallais, and Fabrice Theoleyre. Scheduling for IEEE802.15.4-TSCH and slow channel hopping mac in low power industrial wireless networks: A survey. *Computer Communications*, 114: 84–105, 2017.

[24] Amazon Web Services. What is restful api? https://aws.amazon.com/what-is/restful-api, 2023. Accessed: 2023-01-24.

[25] Aida Vatankhah and Ramiro Liscano. Differential evolution optimization of TSCH scheduling for heterogeneous sensor networks. In *2022 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1491–1496, 2022.