DiNS: Nature Disaster in Network Simulations

Nisal Hemadasa*, Wanli Yu†, Yanqiu Huang‡, Leonardo Sarmiento†, Amila Wickramasinghe§ and Alberto Garcia-Ortiz†
* Hamburg University of Technology, Germany, nisal.hemadasa@tuhh.de
† University of Bremen, Germany, {wyu, sarmient, agarcia}@uni-bremen.de
‡University of Twente, Netherlands, yanqiu.huang@utwente.nl
§ Victoria University, Australia, p.wickramasinghe@live.vu.edu.au

2022 18th International Conference on Mobility, Sensing and Networking (MSN) | 978-1-6654-6457-4/22/$31.00 ©2022 IEEE | DOI: 10.1109/MSN57253.2022.00098

***Abstract*—Wireless sensor networks (WSNs) is a promising solution for disaster management because of its scalability and low cost operations. However, testing the effectiveness of WSNs in real-world disasters is time consuming, costly and in some cases even infeasible. In this paper, we propose a Disaster in Network Simulations (DiNS) framework based on OMNeT++ to replicate a WSN deployed in a disaster in a simulated environment. DiNS allows researchers to observe how the disaster influences the sensor network and how the network can respond in return. A generic coupling interface is developed to support different disaster types. Moreover, we develop a verification tool for functional debugging and verification during new functions developing of sensor nodes, and an optimization tool to support the mathematical optimization of the network operations in response to the disaster. The functionality of DiNS is demonstrated with a case study using wildfire disaster. It provides an easy way for validating and optimizing the disaster management with WSNs.**

***Index Terms*—OMNeT++, sensor networks, disaster management, wildfire simulation**

I. Introduction

Unpredictability is one dominant characteristic inherent in natural disasters that intensifies the caused damage. Disaster management has been gaining great attention in the past decades. Managing a disaster includes mainly four stages, namely mitigation, preparation, response and recovery [1] for preventing the potential threat to lives, property and the environment. The same stages can be perceived with respect to the timeline of the disaster occurrence, e.g., overseeing events in the pre-disaster (such as considering preventive measures and monitoring), the post-disaster (such as the analysis after the incident), and the disaster window (such as handling and responding for the situation during the period of active occurrence of the disaster) [2]. Over the years, technology has emerged to aid disaster management, especially in early detection and acting within the window of the disaster. The basic principle followed by these technologies is primarily to monitor the environmental condition and communicate them for further processing and decision-making.

Wireless Sensor Network (WSN) is a promising technology because of the low-power and low-cost operations, scalability, and being able to formulate intelligent solutions. It has been applied widely in many studies for disaster management, e.g., in [3]–[5]. However, testing and verification of these proposed works under real-world disaster scenarios are rarely done mainly due to the potential danger, cost and unavailability of resources and expertise. An alternative solution is to couple WSN simulators with disaster simulators using empirical, semi-empirical or physical models to evaluate the efficiency and effectiveness of the monitoring network.

Towards this direction, on the one hand, a proportion of work emphasizes developing accurate disaster models while simplifying the functionality of sensor networks. For example, Terzis et.al in [6] simulate hill movements during landslides using the sophisticated Finite Element Model, HOPDYNE [7], but the sensor network model is assumed to be free of errors and node failures. On the other hand, a number of studies focus on the accurate functionalities of the networks while using simplified disaster models. For instance, Wenning et.al. [8] proposes a routing scheme for monitoring virtual wildfire spreading. The network simulator OPNET [9] has been used to study the functionality of the sensor network, under the assumption that the fire spreads in an elliptical form at a constant speed of 1 m/s on the minor axis and 2 m/s on the major axis to preserve the simplicity of the disaster model. Similarly, AL-Dhief et. al [10] assume that the fire front is an enlarging circle whose radius increases at a constant speed as time elapses to evaluate their proposed routing protocols. As argued by FDS [11], Flammap [12], the shape of the fire spread and its rate of propagation depend on numerous factors and do not remain as constants. Simplifying either the network models or the disaster models reduces the evaluation complexity, but brings bias to the effectiveness in reality.

Although sophisticated coupling between realistic network and disaster models has been carried out recently, the existing coupling interface between the network simulator and the disaster simulator is fixed, which allows no flexibility when the disaster simulator changes. For instance, Aslan et. al in [13] present a custom simulator for monitoring wildfires using WSNs. The simulator is developed using the C# programming language. The wildfire simulation library FireLib [14] is interfaced into the simulator to model the wildfire and to feed its output to the simulator. However, its compatibility of interfacing with simulators other than FireLib is not clarified. Similarly, Garcia et. al in [15] propose EIDOS (Equipment Destined for Orientation and Safety), where TOSSIM [16] is adopted to simulate the wireless sensor network with an interface fixed to FARSITE [17] wildfire simulator. Besides the flexibility limitation, the above-mentioned studies provide no open access source codes and are difficult to be re-used.

To address the above-mentioned limitations, this work proposes a simulation framework, "Nature Disaster in Network Simulations (DiNS)", to couple the realistic disaster and network models with a generic interface for researchers to efficiently and accurately test, validate and analyze the WSN-based disaster management approaches with open-source code [1] [2]. The main contributions of this work are as follows:

[1]https://gitlab.com/nisalhem/DiNS-nature-disaster-in-network-simulations
[2]https://gitlab.com/nisalhem/DiNS-verification-optimization-tools

978-1-6654-6457-4/22/$31.00 ©2022 IEEE
DOI 10.1109/MSN57253.2022.00098

Authorized licensed use limited to: UNIVERSITY OF TWENTE.. Downloaded on August 15,2023 at 07:27:59 UTC from IEEE Xplore. Restrictions apply.

- It develops a simulation framework, DiNS, based on OMNeT++ discrete event simulator integrated with a generic interface to couple any disaster simulators with existing accurate models or historical data of disasters that occurred in the real world. It allows the disasters to directly impose their influences on the network while also allowing the network to adapt its functionalities in response to the disasters. With the generic interface, DiNS is independent of the disaster simulators/types (different from existing dedicated solutions).
- It further develops two extensions integrated in DiNS namely, the "verification tool" and the "optimization tool", using Python programming language. The verification tool performs as a generic solution to address the functional verification and debugging issues during developing new functions in sensor nodes. The optimization tool enables users to optimize the network operations by choosing the Python mathematical functions off-the-shelf, without having to program them from the scratch in OMNeT++, e.g., calling the existing learning function to optimize the routing protocol.
- It demonstrates the functionality of DiNS using a case study of a wildfire disaster scenario to show the 2-ways interaction, the use of verification and optimization tools.

The remaining sections of the paper are organized as follows. A brief introduction of OMNeT++, which is the base of DiNS, is provided in section II. In section III, it presents the details of the developed DiNS framework including the integration of the verification and optimization tools. In section IV, it demonstrates the usability of DiNS via a case study using a wildfire disaster based on Fire Dynamic Simulator (FDS). Finally, section V concludes this work.

II. Background: OMNeT++ Simulator

As OMNeT++ is the core of the proposed DiNS simulation framework, this section briefly overviews its main functionalities and the associated INET framework.

OMNeT++ [18] is a discrete event network simulator, which has frequently been used for network simulations among the scientific and academic communities. OMNeT++ provides the base for network simulation models and frameworks in research areas such as wireless communication networks, communication protocols at different levels of the network stack, etc. It could be directly used or integrated as plug and play components into novel models or frameworks. Moreover, it also features application-specific simulation frameworks, e.g., vehicular networks [19]. The network models associated with OMNeT++ are structured in modular architectures. These modules are programmed mainly using C++ programming language. In addition, NEtwork Description (NED) is used as the topology description language to define the abstract level specifications of a module such as its interfaces, parameters passed, interconnections between other modules and sub-modules, etc. The communication between modules is carried out using message passing. `omnetpp.ini` file is used to provide network and model parameters, and configurations to the simulation externally. OMNeT++ editor offers a graphical runtime environment, which is based on the Eclipse Integrated Development Environment (IDE) [20]. It also provides the users with the same debugging features that Eclipse platform provides and users can refer to the log files created after the simulation. Although OMNeT++ is a discrete event simulator, it replicates a network whose nodes run in parallel and independent of each other. Moreover, together with the factors of the wide range of simulation models and frameworks, comprehensive documentation, cost-free availability, etc., OMNeT++ has become the foundation for a broad set of open-source projects and is constantly growing.

INET [21] is the standard framework for developing network models in OMNeT++. It is an open-source model library consisting of a variety of protocols and support modules corresponding to different layers of the network stack. These could be used as building blocks directly as plug and play modules to put together simulation networks. The users could also develop novel modules and integrate them into the existing frameworks or deploy in new frameworks, in addition to native modules that INET already offers. INET is mainly community-driven and hence new model contributions are welcome. As a result, it is also possible to utilize INET as the foundation to create new frameworks or to extend the existing ones.

The groundwork of DiNS has been laid on top of the INET 4.0.0 and OMNeT++ 5.4.1, which was the latest versions when this work is initiated. Despite of the merits mentioned above, OMNeT++ lacks the visual debugging interface when adding new functional blocks and the mathematical library to enable the sensor network optimally reacting to the disaster environment. These limitations are addressed in DiNS framework.

III. The Proposed Design of DiNS

This section introduces the DiNS framework that couples the OMNeT++ network simulator with the virtual disaster scenario using a generic interface to replicate a WSN deployed in a disaster in a simulated environment. The whole design of DiNS is demonstrated as a four-layered architecture framework as shown in Fig. 1. Layers 1-3 involve the tightly coupling of the disaster and network environments while Layer

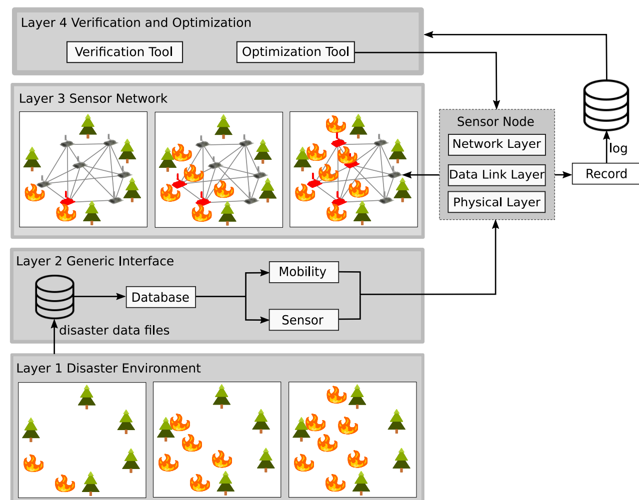

Fig. 1. The proposed four-layered architecture of DiNS: Layer 1 creates the disaster environment; Layer 2 provides the generic interface between the disaster and network environments; Layer 3 mainly includes the network simulation; and Layer 4 contains the verification and optimization tools for visually debugging and network optimization.

Authorized licensed use limited to: UNIVERSITY OF TWENTE.. Downloaded on August 15,2023 at 07:27:59 UTC from IEEE Xplore. Restrictions apply.

4 mainly focuses on the functionality verification and network optimization. Specifically, Layer 1 is the disaster environment, e.g., historical data, disaster models or simulators; Layer 2 is the generic coupling interface between the disaster and network (i.e., OMNeT++) environments; Layer 3 represents the network architecture and simulations in OMNeT++; The last one, Layer 4, includes the new verification and optimization tools for visually debugging and network optimization.

A. Layer 1: Disaster Environment

Layer 1 creates the virtual disaster environment using either a simulator software or the historical data of a real-world disaster for reconstructing the complete incident. In the case of simulator software, the simulations are executed and a bunch of output data is collected and forwarded to the network simulator (Layer 3) through the generic interface (Layer 2). Therefore, the representation and the accuracy of the timely output data of the disaster incident are the main focuses for creating the simulated disaster environment. For example, in a wildfire simulation, this could be temperature and humidity measurements and their respective time stamps of the occurrences. It also includes information on the geographical area in which the disaster is taking place. In this sense, DiNS can support any type of disasters and is independent of disaster simulators and models because of the generic interface.

B. Layer 2: Interface Between Disaster and Sensor Network

Layer 2 in the DiNS framework provides a generic interface to couple the disaster and the network simulations, which is independent of the disaster types different from existing dedicated solutions in [13]–[15]. It consists of a buffer to store the generated disaster data in an OMNeT++ readable file, a database to convert the data format and feed the data to the network simulator, a mobility module to synchronize the data in space (map the disaster position with the sensor node location) and a sensor module for data synchronization in the time domain (map the data generation frequency of the disaster simulator with the sampling rate of sensor nodes). More specifically, the disaster data file (e.g. a CSV file) is generated to buffer the timely output data of the disaster simulator, including the disaster parameters (e.g., humidity, temperature, etc.) and the corresponding locations at each time stamp. A "Database" module as shown in Fig. 1, which is programmed inside of OMNeT++ network simulator, reads the data from the disaster data file, converts the data format and feeds the disaster data into the simulated network environment in Layer 3. The (x, y, z) coordinates of the disaster data are read into network simulation by the "Mobility" module. For a static network, this would be constant values. DiNS however could be extended to support mobility networks, simply either by reading coordinates or changes in coordinates with respect to time. This enables importing mobility traces into the network simulation. Static network coordinates are further interpolated/sub-sampled to produce sensor readings at the specified locations of sensor nodes in Layer 3. The "Sensor" module estimates the readings of a node at any given moment by interpolating/sub-sampling the data using its past samples. This mitigates the mismatch of data generation frequency at Layer 1 and the sampling rate of sensor nodes at Layer 3. Besides, as the disaster may destroy the sensor node (e.g. due to high temperature), all the functions in Layer 3, including receiving, transmission, channel checking, etc, should be thereby disabled when the node is dead. The "Sensor" module also estimates the the node's death time by comparing the interpolated readings with a threshold, and sends a flag to Layer 3 to disable all related network functions.

The generic interface of DiNS has an extensible architecture. In addition to CSV, extending the framework to read disaster data files in many other machine-readable formats such as JSON, XML, etc., is therefore convenient.

C. Layer 3: Network Architecture and Simulation

The Spatio-temporal output data of the simulated disaster buffered in Layer 2 will be read and fed into the network simulations of OMNeT++, i.e., Layer 3, to help create the simulated network. By sharing the timeline, the integration of the disaster and network simulations is able to mimic the scenario of a WSN deployed and operated in a live disaster environment. The disaster will directly interact and influence the network, because the sensor nodes read the data from the output data of the disaster environment. The measured disaster data of the WSN will be stored by the "Record" module and fed into Layer 4 of DiNS for further operations. Based on the user-defined objectives, the output of Layer 4 can directly guide the network to respond to the disaster environment.

The main task in Layer 3 is to create the simulated WSN network using OMNeT++, which consists of multiple sensor nodes. Thus the key point is the modular architecture of the sensor node, which defines the network stack of the sensor nodes and how the disaster data is merged into the network simulation. Typically, the sensor node architecture involves the designs of the network layer, data link layer and physical layer.

- Network layer: It is responsible for the data packet transmission including routing operations within the network or through different networks. The INET framework in OMNeT++ features a range of routing protocols. Besides the standard IPv4 and IPv6 network layer protocols, a wide range of alternative network layer protocols can be configured, e.g., Flooding, WiseRoute, AdaptiveProbabilitsticBroadcast, AODV, OSPFV, etc.
- Data Link layer: It provides the services such as framing and link access, reliable delivery, flow control, error detection and correction, half-duplex and full-duplex. In addition to the available data link layer protocols integrated into OMNeT++, many open-source protocols can also be used according to different objectives.
- Physical layer: A radio model component is responsible for modelling the physical layer and relies on the antenna model, transmitter model, receiver model, error model and energy consumption model. The INET framework in OMNeT++ provides several physical layer standards such as IEEE 802.11 and IEEE 802.15.4.

Note that, the users can either select their preferred modules from the INET libraries or develop their own modules for the specific applications. In addition to the sensor node architecture modules, a "Record" module is also included in Layer 3 of DiNS as shown in Fig. 1. It is used to generate a log file to store the network output and feed that into Layer 4 for further operations. The output data includes the information of the data packets received and transmitted by each node

Authorized licensed use limited to: UNIVERSITY OF TWENTE.. Downloaded on August 15,2023 at 07:27:59 UTC from IEEE Xplore. Restrictions apply.

throughout the simulation, the physical measurements (e.g., temperature, humidity, etc.) and the death time of the sensor nodes, etc. These data can be analyzed and used according to the user's preferences for different applications.

D. Layer 4: Verification Tool and Optimization Tool

When developing new modules in OMNeT++, developers often encounter functionality problems and bugs. Among the currently available debugging methods and practiced in OMNeT++, a key part that is missing is the unavailability of a visual interface, to distinctly cross-refer activities of nodes [18], [22]. Moreover, in response to the disaster impacts, the network protocols and/or node parameters need to be optimized. Iterative data generation, processing and training are required to find the optimal solution for a certain disaster scenario. However, OMNeT++ lacks the corresponding mathematical library and developing the mathematical functions from scratch is inconvenient and time consuming.

To address these problems, we build two reusable tools based on Python in Layer 4 of DiNS: a) the verification tool for debugging and verifying the functionality when developing new modules in OMNeT++: b) the optimization tool for automating the simulation process and optimizing the network operations under certain disaster environments using the optimization functions/algorithms in the Python library. The details of these tools will be presented in this section.

1) Verification tool: Debugging and verification of software products is essential in building error-free, perfectly functioning software applications. OMNeT++ IDE has the same debugging features that the Eclipse platform provides, which requires ample time and effort to observe, monitor, and cross-check the activities of the nodes over time. This is inconvenient and time-consuming when developing new modules from scratch. Motivated by such inefficiencies, we develop a reusable module to monitor the network functionality. This is inspired by digital design verification tools with graphical user interfaces, such as ISim, ModelSim, QuestaSim, etc. Hence, a resemblance of the graphical user interface of the presented verification tool, to the above-mentioned digital design verification tools could be seen. The "verification tool" in OMNeT++ discloses and displays the operations of the radio and the changes in states of the data link layer. The data from the network simulation is fed to the verification tool via a log file created by the record module in Layer 3 of DiNS. Fig. 2 depicts how the data is being transferred from the OMNeT++ environment to the verification tool. As mentioned in the "Record" module in Layer 3 of DiNS, the starting and ending time stamps of the log file writing about the network output data will be recorded. By giving the MAC address of a node and the verification starting and ending time, the verification tool is able to extract the node activities within the exact time duration of concern without having to wait until the entire network simulation is complete. During the simulation, the log file is updated with the nodes' activities in the data link layer (sleep, CCA, send, acknowledgement). Once the log file has been completely updated, the verification tool reads it and displays the activities graphically by interpreting the data as stepped line graphs for each node. The x-axis represents the time while the y-axis represents the nodes' activities with

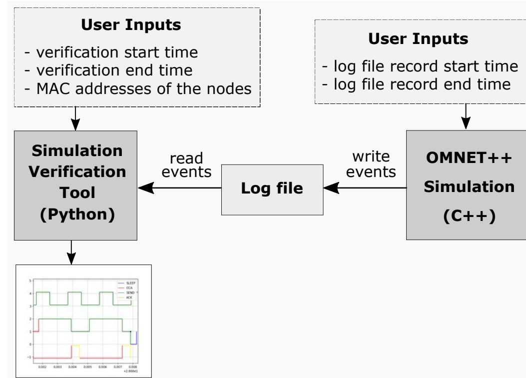

Fig. 2. Verification tool flowchart: A separate log file of the network output data is created by Layer 3 of DiNS, which is then read by the verification tool. The starting and ending time stamps of the log file writing are used to prevent the storage overhead problem caused by logging redundant data. The verification tool is executed independently of the sensor network simulation in OMNeT++ to provide a graphical representation of the network activities.

different colours. The detailed demonstration is illustrated in section IV-B.

2) Optimization tool: As OMNeT++ by default does not inherit supporting libraries for mathematical optimization, users have to program functions from scratch when they want to execute network optimizations, e.g., network-related operations, node parameter fine-tuning, etc. However, modelling complex mathematical theories into algorithms from scratch takes time and effort, especially C++ programming language is the main option. Moreover, such functions have to be integrated to the core network design, without disrupting the internal modular structure or the network architecture.

To tackle the above issues, this work develops an optimization tool for users to do the network optimizations based on Python, since Python provides extensive open-source libraries and functions for mathematical optimization and demands less implementation overhead and intensive programming skills. The developed optimization tool will not interfere with the core network operations of OMNeT++ and release from any undesired alterations of the design that the internal network framework has to undergo. As depicted in Fig. 3, it reads the network output through the log file generated by the "Record" module in Layer 3 of DiNS, inputs the data into the data processing interface and executes the optimization algorithms. After that, the optimized network operations are forwarded to the network to response the corresponding disaster scenarios.

The typical implementation of an optimization algorithm can be briefly summarized as follows. The first step is to initialize the variables, which will be then fed into the objective function. The objective value will be generated and the variables are accordingly adjusted and fed into the objective again. This process iterates until the optimal objective value (minimum/maximum) is reached. For the optimization tool of DiNS, it initiates the variables, i.e., the parameters of the network output, then executes the selected optimization algorithms from the Python library. The obtained network parameters will be updated in `omnetpp.ini` and the OMNeT++ application will be executed externally via the command-line interface, i.e., Cmdenv, which is given in the OMNeT++

Authorized licensed use limited to: UNIVERSITY OF TWENTE.. Downloaded on August 15,2023 at 07:27:59 UTC from IEEE Xplore. Restrictions apply.

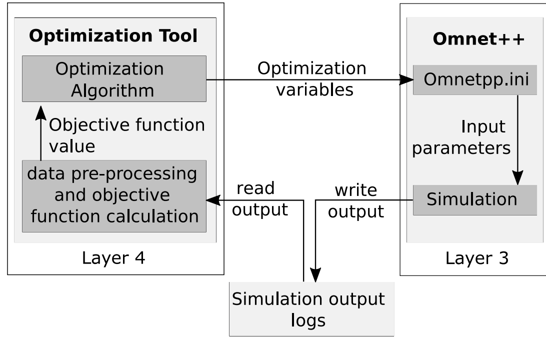

Fig. 3. Optimization tool flowchart: The Python application initiates input parameters of the selected optimization algorithm or function. The network parameters in the `omnetpp.ini` are modified using such selected input parameters. After modification of parameters, the OMNeT++ application is executed externally by the Python application. At the end of the simulation, the results are read, processed and is fed back into the optimization function.

simulation manual [22]. The optimization tool waits for the new input variables from the "Record" module after the network responds to the disaster for further optimizations. The optimization tool uses the buffer update time stamps to detect the newly written data in the buffer and converts the data for the Python development environment. The data hence could be processed and eventually be transformed into a numerical value using a defined objective function. A simple use case of the optimization tool could be optimizing the routing path of sensor nodes to maximize the reception of the most important data during a disaster. For more details, please refer to Sec. IV-C.

The optimization tool could be not only used to optimize network parameters but also to automate the execution of OMNeT++ multiple times without any human intervention because its overall architecture resembles a master (optimization tool) - slave (OMNeT++) model as shown in Fig. 3. This could be strongly useful for emerging applications such as machine learning model training, dynamic network optimization, etc. Besides, the optimization tool further provides support in diverse data analysis functionalities since the output data is read into the Python development environment.

IV. Testing the Framework: A Case Study

In this section, we demonstrate how to use the proposed DiNS framework to simulate a WSN under a wildfire disaster and the performance of the verification tool through an example of detecting and identifying the anomalies in the communication among the sensor nodes. Moreover, multivariate optimization using the optimization tool is illustrated to optimize the network operations in response to the disaster.

DiNS could be used not only with wildfires, but also for other disaster incidents in which applications of WSN needs to be simulated such as volcano monitoring [23], earthquakes [24], landslides [25], mining catastrophes [26], etc.

A. Applying DiNS in Wildfire Scenario

This work presents a case study of DiNS using a wildfire disaster scenario. The monitoring area is a $1024m \times 1024m$ woodland with geographical variations and various vegetation. This work exploits the Fire Dynamic Simulator (FDS) [11] in

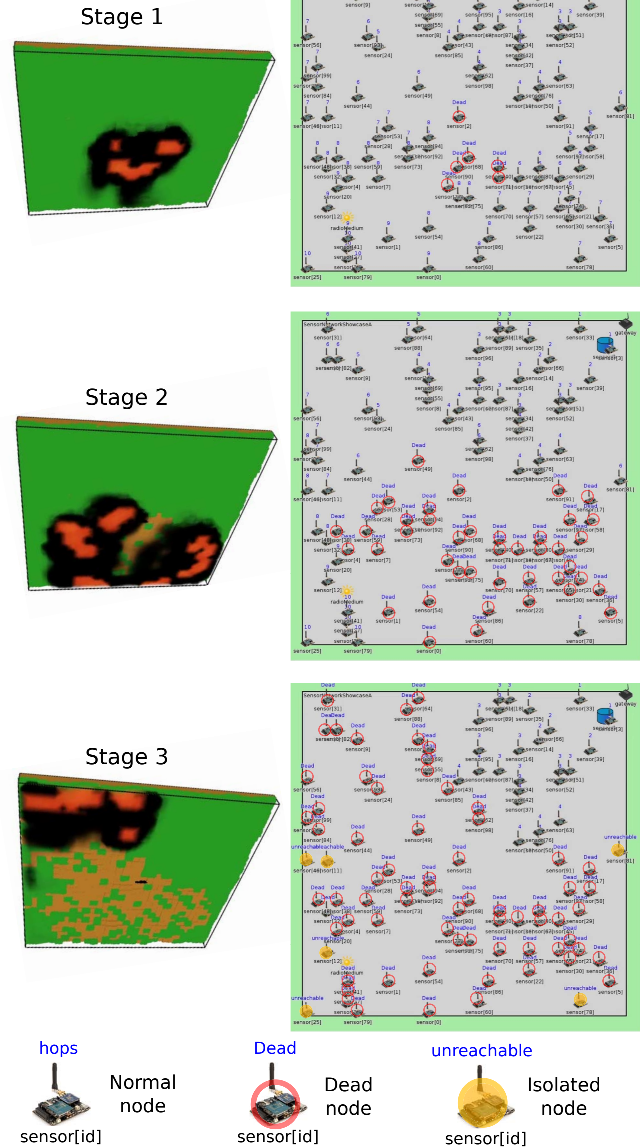

Fig. 4. Step-wise propagation of a fire in a virtual forest in Fire Dynamic Simulator (FDS) and the effect on the WSN network in OMNeT++.

Layer 1 of DiNS framework to model and simulate the wildfire disaster. A fire starts at a random place in this woodland, and the spreading profile of the fire is affected by various factors, e.g., the topography of the terrain, vegetation, ignition/flash point of fire, wind, etc. This complete virtual woodland is made up of aggregating individual cells to form a simulation computational mesh for the fire. Fig. 4 demonstrates the stepwise propagation of a wildfire scenario. The duration of fire disaster lasts for 1400 seconds. In this work, the resolution of the computational cells of the woodland is set to $32m \times 32m$ due to the limited computing resources. However, this resolution could be made even more granular to meet the level of precision and detail expected from the simulation.

Authorized licensed use limited to: UNIVERSITY OF TWENTE.. Downloaded on August 15,2023 at 07:27:59 UTC from IEEE Xplore. Restrictions apply.

TABLE I

ONE EXAMPLE OF 63 DUPLICATED DATA CHUNKS IN THE DATA RECORD AT THE GATEWAY FROM 28.5554 TO 28.9858 SECONDS (THE 1ST ROW IS THE RECEIVING TIME AT THE GATEWAY; THE 2ND ROW STANDS FOR THE TIME OF THE DATA CHUNK GENERATED BY THE SOURCE NODE; THE 3RD ROW REPRESENTS THE SOURCE NODE ID; THE 4TH ROW IS THE TYPE OF DATA, E.G., 0 REPRESENTS THE TEMPERATURE AND 1 REPRESENTS THE HUMIDITY; THE LAST ROW GIVES THE CORRESPONDING VALUES OF THE MEASURED DATA, E.G., 21.511°C.).

...	28.4716	28.5554	28.5554	28.5554	28.5554	28.5718	28.5718	...	28.9858	28.9858	28.9858	28.9858	28.9910	...
...	15.2828	20.9023	20.9023	15.9023	15.9023	20.9023	20.9023	...	20.9023	20.9023	15.9023	15.9023	20.618	...
...	50	27	27	27	27	27	27	...	27	27	27	27	86	...
...	1	0	1	0	1	0	1	...	0	1	0	1	0	...
...	39.0254	21.5110	39.8852	18.6616	39.1556	21.5510	39.8852	...	21.5110	39.8852	18.6616	39.1556	18.6626	...

After the fire simulation, three files (disaster data files) in ".csv" (comma separated value) format are created for Layer 2 of DiNS including: the coordinates of the sensing points (cells) of the woodland; the temperature readings at each second of all sensing points; and the humidity readings. These disaster data files are read by the "Database" module as the input of the WSN network simulated in OMNeT++. The WSN network consists of 100 sensor nodes randomly deployed in the woodland. The "mobility" module matches the locations of the sensor nodes and the cell coordinates of the woodland, and the "sensor" module samples the corresponding temperature and humidity at a given rate of 0.2 Hz, i.e., the sensor node measures the data every 5 seconds. Those measurements will be the input of the simulated sensor network and can be configured in OMNeT++ via the `omnetpp.ini` file.

Once the simulation is initiated, the complete execution replicates a well-functioning WSN deployed in a woodland under the potential threat of wildfire. From this point onward, the users have complete control over the network and the corresponding design, e.g., select or build their own preferred network stack, set their own objectives for the WSN, program their own routing algorithms or even add their own novel modules and test them in the wildfire conditions. In this case study, towards the objective of low-power and efficient computation and communication, we exploit specific protocols for the sensor node architecture in Layer 3 of DiNS as follows.

- Physical Layer: The IEEE 802.15.4 Standard is applied in the physical layer for low-cost ubiquitous communication between wireless sensor nodes.
- Data Link Layer: This case study selects the BoX-MAC-2 protocol for the data link layer, because of the ultra-low power utilization and comparatively high throughput
- Network Layer: RPL (Routing Protocol for Low-Power and Lossy Networks) routing protocol is used in the network layer, since it is a proactive protocol optimized for wireless networks with low power consumption and generally susceptible to packet loss.

When the simulation starts, the "Record" module in Layer 3 of DiNS generates three network output files in ".csv" format, i.e., `nodeParameters<file_number>.csv`, `sensorData<file_number>.csv` and `liveState<file_number>.csv`, to record the events that occurred in each node and its neighbors, the measured temperature and humidity data chunks and the corresponding time stamps of data producing and receiving, and the live and death time, respectively. Those results are the input to feed the verification and optimization tools in Layer 4 of DiNS for further operations.

When the network starts to work, each sensor node senses the temperature and humidity every 5 seconds corresponding to its own location, compiles the measurements into packets and forwards the data along with the network to the gateway. To create the desynchronization among the nodes, each node is made to start operating at a random point of time between 0th and 1st second at the very beginning of the simulation. The sensor node stops this periodical operation at the end of the simulation (i.e., after 1400 second) or when it dies due to the harsh disaster environment. In this work, we define a sensor node to be dead if the temperature it measured is larger than or equal to 70°C and the relative humidity is below 40%. Fig.5 illustrates the sensor readings of the temperature and relative humidity values of node 1 and node 5. They died at different time. "Sensor" module estimates the exact death time and deactivates all functions of these nodes in Layer 3. The effect of the wildfire on the network is illustrated in Fig. 4. In the initial stage of the fire, 6 sensor nodes die because of the high temperature. With the propagation of the fire, more area of the woodland is affected and 42 sensor nodes are dead. When the fire is finally ended, in total 68 sensor nodes died because of the harsh environment. Although there are 38 sensor nodes still alive, 7 out of them lost the connections to the gateway and become isolated.

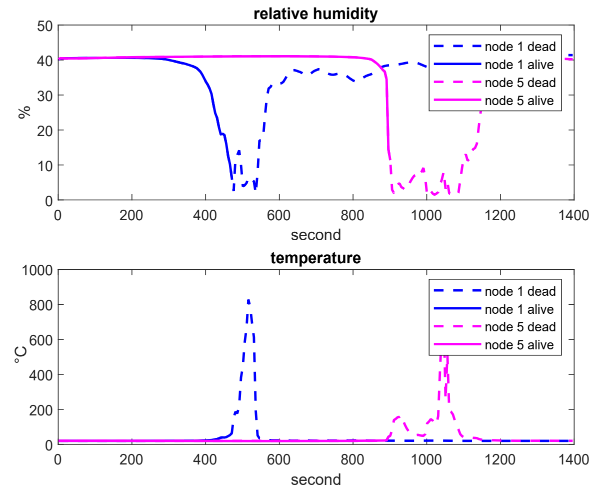

Fig. 5. The sensor readings of the temperature and relative humidity values of node 1 and node 5 as fire propagates. After the node died (temperature>70°C and relative humidity<40%), the functionality in Layer 3 is deactivated.

B. Abnormality analyzing using the verification tool

According to the data records in the gateway, we observe that more than 80% of the data is repeatedly received in the gateway. It is hard or even impossible to investigate why it happens. Tab. I illustrates one example of the duplicated data

Authorized licensed use limited to: UNIVERSITY OF TWENTE.. Downloaded on August 15,2023 at 07:27:59 UTC from IEEE Xplore. Restrictions apply.

in the data record: the 1st row is the receiving time at the gateway; the 2nd row stands for the time of the data chunk generated by the source node; the 3rd row represents the source node ID; the 4th row is the data type (0 represents the temperature and 1 represents the humidity); the last row gives the corresponding values of the measured data. In the table, there are 63 repetitions of the data generated by the node 27 when the simulation runs from 28.56 to 28.99 seconds.

In order to figure out the reason, the developed verification tool is applied to visualize the behaviour of this abnormality. Fig. 6 depicts one example of the abnormal scenario of duplicated receiving data chunks. Specifically, node 2 turns on the radio in transmission (TX) mode, and transmits data to the neighbouring node. After a while, node 1 starts to transmit data to the gateway, and the gateway turns on the radio in receiving (RX) mode, to receive the data. After the successful reception, the gateway sends the acknowledge (ACK) back to node 1, which is interfered by the transmission of node 2. As a result, node 1 repeats the transmission because of the absence of ACK while the gateway receives all the re-transmitted data, which is clearly shown in Fig. 7 (a magnified view of Fig. 6). When node 2 stops its transmission and turns off the radio, node 1 successfully receives the ACK and ends its own transmission as shown in Fig. 8. It becomes much easier to find the reason for the duplicated data using the verification tool, which is caused by the hidden terminal problem: node 1 is located within the radio range of both node 2 and the gateway while the gateway cannot directly reach node 2. Further strategies can be developed in the MAC layer to cope with this problem.

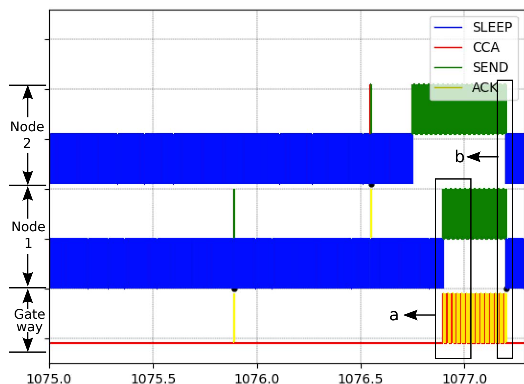

Fig. 6. The visualization of the duplicated data abnormality using the verification tool: The gateway (bottom most) listens (in red), receives the data successfully, and consequently sends an ACK (in yellow) to node 1's (in the middle) transmission (in green); node 2's (top most) transmission interferes with the ACK reception of node 1, which drives node 1 to re-transmit.

C. Multivariate optimization using the optimization tool

Generally, the optimization tool optimizes the network parameters in the OMNeT++ simulator according to the objective functions. Taking the SciPy library in Python [27] for example, a typical optimization has the following format:

```
result = minimize(objective_function, x0,
    method, constraints, options, bounds)
```

where x_0 is the initial estimation of variables to be optimized, and method refers to the mathematical method of optimization e.g., SLSQP, Newton-CG, etc. The $objective_function$

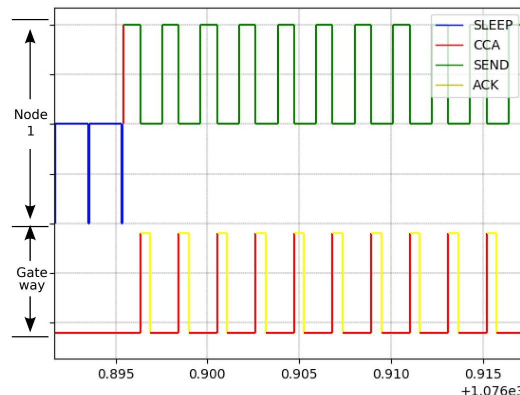

Fig. 7. The magnified view of the box (a) in Fig. 6: The gateway repeatedly sends the ACKs in response to the successful reception of data from node 1.

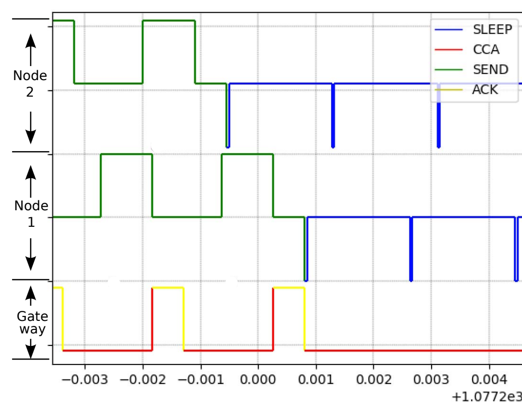

Fig. 8. The magnified view of the box (b) in Fig. 6: Node 1 finally receives the acknowledgement from the gateway, when the interference from node 2 is gone, i.e., node 2 stops its transmission.

is structured in a way such that several additional support functions are led before the mathematical objective function, as shown in the following Python function.

```python
def objective_function(x)
    update_omnetpp_ini(x)
    run_omnet_app()
    raw_data = read_data()
    proc_data = process_data(raw_data)
    return math_objective_fn(proc_data)
```

where each of the supporting functions are:

- $update_omnetpp_ini(x)$: This function takes as a parameter, the input variables (x) of the network. These variables will be written to `omnetpp.ini` file for the OMNeT++ network simulator to read.
- $run_omnet_app()$: This function executes OMNeT++ application via the command-line interface (Cmdenv). Python offers interfaces to the operating system [28] for executing the commands within a Python program. For detailed command guidances please refer to the OMNeT++ simulation manual [22].
- $read_data()$ This function holds the program waiting until the network simulation is complete and the log files are updated. Subsequently, it reads the log files and returns the raw data.

Authorized licensed use limited to: UNIVERSITY OF TWENTE.. Downloaded on August 15,2023 at 07:27:59 UTC from IEEE Xplore. Restrictions apply.

- $process_data(raw_data)$ This function takes in the raw data, pre-processes it, and returns processed data.
- $math_objective_fn(proc_data)$ This function takes in the processed data, applies it in the mathematical models and returns value to the optimization function from SciPy.

We have designed a Reinforcement Learning (RL)-based routing algorithm for the sensor node to decide the appropriate actions in response to the disaster using the developed optimization tool. The algorithm detail is out of this paper's scope. Here we only aim to show how the optimization tool supports the optimization of network operations.

In short, the objective is to deliver the maximum volume of useful data with minimum latency in the dynamic, time-critical disaster environment. This is essential when a disaster occurs to retrieve the current status so that efficient rescue operations can be implemented. The input of RL, i.e., the state of the environment, is the network output from Layer 3 of DiNS including the remaining battery capacity, riskiness class, number of hopes to the gateway, transmission penalties to the neighbouring nodes and remaining memory storage of each sensor node. The actions refer to the activities of the sensor node including sensing, transmitting, receiving, and storing data, while the corresponding reward of each action is related to the transmission of the data with high riskiness. A reward is positive if the transmission succeeds, otherwise, it will be a negative value. We generate various fire disasters in DiNS to train the policy of mapping the network output at a given time instant to the best actions of the sensor nodes, thereby learning the rules to respond to the disasters. The simulation results show that using the RL-based routing algorithm, 19.4% more useful unique data chunks are delivered to the gateway than using the minimum hop routing algorithm. The optimization tool facilitates this protocol development of the sensor networks easily and conveniently.

V. Conclusion

Practical experiments for testing the WSN-based disaster management strategies are often impossible due to the high risk and cost. This work proposes DiNS, a framework to simulate sensor network operations in disasters. DiNS is developed by extending OMNeT++ discrete event simulator with a generic interface to disaster simulators, models or historical data. It allows expandability and applicability to generic sorts of disaster types and simulators. This paves the way that the behaviour of the disaster can directly influence the network. Using the developed verification and optimization tools, the network can be debugged and optimized to respond to the disaster according to the user-defined objectives. A case study has been used to demonstrate the functionality of DiNS by coupling the Fire Dynamic Simulator (FDS) with OMNeT++. The verification tool is applied to graphically visualize and analyse a packet duplication occurrence arising as a consequence of the hidden terminal problem. A generic optimization library function in Python-SciPy is used to show one way of applicability of the optimization tool with OMNeT++.

References

[1] A. Baird, *Towards an Explanation and Reduction of Disaster Proneness*, ser. Disaster research unit, occasional paper. Bradford University, Disaster Research Unit, 1975. [Online]. Available: https://books.google.de/books?id=o-AaGQAACAAJ

[2] G. L. Cozannet, M. Kervyn, S. Russo, C. I. Speranza, P. Ferrier, M. Foumelis, T. Lopez, and H. Modaressi, "Space-based earth observations for disaster risk management," *Surveys in Geophysics*, vol. 41, no. 6, pp. 1209–1235, mar 2020.

[3] A. Khalifeh, K. A. Darabkh, A. M. Khasawneh, I. Alqaisieh, M. Salameh, A. AlAbdala, S. Alrubaye, A. Alassaf, S. Al-HajAli, R. Al-Wardat, N. Bartolini, G. Bongiovannim, and K. Rajendiran, "Wireless sensor networks for smart cities: Network design, implementation and performance evaluation," *Electronics*, vol. 10, no. 2, p. 218, jan 2021.

[4] T. S. Durrani, W. Wang, and S. M. Forbes, Eds., *Geological Disaster Monitoring Based on Sensor Networks*. Springer Singapore, 2019.

[5] S. Singh, A. S. Nandan, A. Malik, N. Kumar, and A. Barnawi, "An energy-efficient modified metaheuristic inspired algorithm for disaster management system using WSNs," *IEEE Sensors Journal*, vol. 21, no. 13, pp. 15 398–15 408, jul 2021.

[6] A. Terzis, A. Anandarajah, K. Moore, and I.-J. Wang, "Slip surface localization in wireless sensor networks for landslide prediction," in *Proceedings of the fifth international conference on Information processing in sensor networks - IPSN '06*. ACM Press, 2006.

[7] A. Anandarajah, "Hopdyne: A finite element computer program for the analysis of static, dynamic and earthquake soil and soil-structure systems," *The Johns Hopkins University, Baltimore, Maryland*, 1990.

[8] B.-L. Wenning, D. Pesch, A. Timm-Giel, and C. Görg, "Environmental monitoring aware routing: making environmental sensor networks more robust," *Telecom. Systems*, vol. 43, no. 1-2, pp. 3–11, sep 2009.

[9] OPNET. Opnet modeler. [Online]. Available: $http://opnet.com/solutions/network_rd/modeler.html$

[10] F. T. AL-Dhief, R. C. Muniyandi, and N. Sabri, "Performance evaluation of LAR and OLSR routing protocols in forest fire detection using mobile ad-hoc network," *Indian Journal of Science and Technology*, vol. 9, no. 48, dec 2016.

[11] K. B. McGrattan and G. P. Forney, "Fire dynamics simulator (version 4) :," Tech. Rep., 2004. [Online]. Available: https://pages.nist.gov/fds-smv/

[12] M. Finney, "An overview of flammap fire modeling capabilities," 2006.

[13] Y. E. Aslan, I. Korpeoglu, and Özgür Ulusoy, "A framework for use of wireless sensor networks in forest fire detection and monitoring," *Computers, Environment and Urban Systems*, vol. 36, no. 6, pp. 614 – 625, 2012, special Issue: Advances in Geocomputation. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0198971512000300

[14] C. D. Bevins, *fireLib User Manual and Technical Reference*, 1996.

[15] E. M. García, M. Á. Serna, A. Bermúdez, and R. Casado, "Simulating a WSN-based wildfire fighting support system," in *2008 IEEE International Symposium on Parallel and Distributed Processing with Applications*. IEEE, dec 2008.

[16] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM," in *Proceedings of the first international conference on Embedded networked sensor systems - SenSys '03*. ACM Press, 2003.

[17] M. A. Finney, "FARSITE: Fire area simulator-model development and evaluation," Tech. Rep., 1998.

[18] OMNeT++. [Online]. Available: https://omnetpp.org/

[19] C. Sommer, R. German, and F. Dressler, "Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis," *IEEE Transactions on Mobile Computing (TMC)*, vol. 10, no. 1, pp. 3–15, January 2011.

[20] eclipse. [Online]. Available: https://www.eclipse.org/

[21] INET. [Online]. Available: https://inet.omnetpp.org/

[22] OMNeT++, *OMNeT++ Simulation Manual*, omnet++ version 5.6.1 ed.

[23] R. Lara-Cueva, D. Benitez, A. Caamano, M. Zennaro, and J. L. Rojo-Alvarez, "Performance evaluation of a volcano monitoring system using wireless sensor networks," in *2014 IEEE Latin-America Conference on Communications (LATINCOM)*. IEEE, nov 2014.

[24] S. L. Hung, J. T. Ding, and Y. C. Lu, "Developing an energy-efficient and low-delay wake-up wireless sensor network-based structural health monitoring system using on-site earthquake early warning system and wake-on radio," *Journal of Civil Structural Health Monitoring*, vol. 9, no. 1, pp. 103–115, 2019.

[25] S. Kumar, S. Duttagupta, V. P. Rangan, and M. V. Ramesh, "Reliable network connectivity in wireless sensor networks for remote monitoring of landslides," *Wireless Networks*, vol. 26, no. 3, pp. 2137–2152, 2020.

[26] W. Chen and X. Wang, "Coal mine safety intelligent monitoring based on wireless sensor network," *IEEE Sensors Journal*, vol. 21, no. 22, pp. 25 465–25 471, 2020.

[27] The SciPy community. [Online]. Available: https://docs.python.org/3/library/os.html

[28] Python Software Foundation. [Online]. Available: https://docs.scipy.org/doc/scipy/tutorial/optimize.html

Authorized licensed use limited to: UNIVERSITY OF TWENTE.. Downloaded on August 15,2023 at 07:27:59 UTC from IEEE Xplore. Restrictions apply.