Interactive Web Visualizer for IEEE 802.11ah ns-3 Module

Amina Šljivo, Dwight Kerkhove, Ingrid Moerman, Eli De Poorter, Jeroen Hoebeke
Ghent University - imec, IDLab, Department of Information Technology
Ghent, Belgium
{amina.sljivo, dwight.kerkhove, ingrid.moerman, eli.depoorter, jeroen.hoebeke}@ugent.be

ABSTRACT

The main purpose of running ns-3 simulations is to generate relevant data sets for further study. There are two strategies to generate output from ns-3, either using generic predefined bulk output mechanisms or using the ns-3's Tracing system. Both require parsing the raw output data to extract and process the data of interest to obtain meaningful information. However, parsing such output is in most cases time consuming and prone to mistakes. Post-processing is even harder when a large number of simulations needs to be analyzed and even the tracing system cannot simplify this task. Moreover, results obtained this way are only available once the simulation is finished.Therefore, we developed a user-friendly interactive visualization and post-processing tool for IEEE 802.11ah called ahVisualizer. Beside the topology and MAC configuration, ahVisualizer also plots our traces for each node over time during the simulation, as well as averages and standard deviations for each traced parameter. It can compare all the measured values across different simulations. Users can easily download figures and data in various formats. Moreover, it includes a post-processing tool which plots desired series, with desired fixed parameters, from a large set of simulations. This paper presents the ahVisualizer, its services and its architecture and shows how this tool enables much faster and easier data analysis and monitoring of ns-3 simulations with 802.11ah.

CCS CONCEPTS

• **Networks** → **Network performance evaluation**; • **Human-centered computing** → *Visualization*;

KEYWORDS

ns-3, visualization, analysis, post-processing, distributed simulations, IEEE 802.11ah, Wi-Fi HaLow

ACM Reference Format:
Amina Šljivo, Dwight Kerkhove, Ingrid Moerman, Eli De Poorter, Jeroen Hoebeke. 2018. Interactive Web Visualizer for IEEE 802.11ah ns-3 Module. In *Proceedings of the 2018 Workshop on ns-3 (WNS3 2018), June 2018, Surathkal, India.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3199902.3199904

WNS3 2018, June 2018, Surathkal, India

ACM ISBN 978-1-4503-6413-3/18/06…$15.00
https://doi.org/10.1145/3199902.3199904

1 INTRODUCTION

In order to meet the rising demands for Internet of Things technologies, the Wi-Fi community has developed Wi-Fi HaLow (IEEE 802.11ah) [1]. This standard extends Wi-Fi to support dense deployments of autonomous devices with various power capabilities and traffic patterns over a wide area. Wi-Fi HaLow can support up to 8192 devices over 1 km. To achieve such performance, it introduced a number of novel physical and data link layer mechanisms such as Restricted Access Window (RAW), which reduces contention between large number of stations, and Traffic Indication Map (TIM) segmentation, which organizes stations in a hierarchy, thus enabling association of up to 8192 stations with a single Access Point (AP).

The ns-3 simulator opted to be an appropriate starting point for research on Wi-Fi Halow for two reasons, namely because (1) ns-3 closely reflects actual protocol behavior and can easily be set up to evaluate a broad range of network and traffic conditions and (2) there is no hardware supporting Wi-Fi HaLow available at the market yet. ns-3 is a discrete-event network simulator targeted primarily for research and educational use [4]. ns-3 is free software and is publicly available for research, development and use. The simulation core and models are implemented in C++ and the main program defines the network topology and configuration, and starts the simulator.

The whole point of running ns-3 simulations is to obtain information for further study. Information can be obtained by (1) generating output from ns-3 and (2) monitoring with existing visualizers. The two methods, however, do not provide coinciding information as the existing visualizers are limited in functionality and mainly serve monitoring purposes, whereas an ns-3 user can generate any desired output directly from the simulator. The latter can be done in several ways: by simple printout, using NS_LOG or PCAP and using the Tracing System. Unlike the Tracing system, the other output methods typically provide both necessary and unnecessary information available once the simulation is finished. The bulks of raw data then need to be parsed in order to extract useful information. The Tracing system provides mechanisms for more selective output and it can immediately be formated into a form acceptable by gnuplot, but only for a single simulation. Analysis of a large set of simulations still requires extensive parsing after the simulations. Parsing of such output requires writing of additional scripts, which is time consuming, error-prone and a repetitive task.

So far, two visualizers for ns-3 have been developed: PyViz [2] and NetAnim [6]. PyViz provides insight in the simulation data during the simulation, unlike NetAnim that can only animate packets once the simulation is finished. PyViz is useful for debugging and provides a built-in interactive Python console to debug the state of running objects. It also includes the FlowMonitor [3] - a network monitoring framework which detects all flows passing through the

network and stores some metrics which can either be integrated with PyViz or exported to standard output.

We avoided the difficulty of post-processing and the limited visualization options by developing a tool that covers both, the ahVisualizer[1]. We extensively use ns-3 for our research on the new IEEE 802.11ah technology [7]. Thus, we needed an easy way to analyze the data obtained from thousands of simulations and to be able to show the evolution of interesting parameters over time while simulating very dense networks with thousands of nodes, which can take hours to complete. The ahVisualizer is based on the Tracing system and currently only supports IEEE 802.11ah module [8]. However, its design does not prevent the visualizer to be applied in a broader context than IEEE 802.11ah. In the future, we therefore plan to extend it for general use with ns-3.

This paper presents the new ahVisualizer and compares it against the existing post-processing methods and visualizers. Section 2 gives a short overview of both direct-output mechanisms available in ns-3 (Subsection 2.1) and the existing visualization tools (Subsection 2.2). The ahVisualizer is presented in Section 3, namely its functionality in Subsection 3.1 and its implementation in Subsection 3.2. Finally, conclusions and future work are summed up in Section 4.

2 OUTPUT ANALYSIS AND VISUALIZATION

This section gives an overview of the existing methods for obtaining useful information from ns-3 simulations, namely pros and cons of existing data exporting methods on the one hand, and existing visualization tools on the other hand.

2.1 Direct Output

To extract the interesting information from ns-3, or any program for that matter, the obvious solution is printing out the values of interest to the standard output. Of course, this will turn out to be very unsatisfactory in the long term. As the number of print statements increases in a program, the task of dealing with the large number of outputs will become more and more complicated. Moreover, scripting for parsing purposes is not only time-consuming and boring, but also error-prone. Commercial software typically has 20 to 30 bugs for every thousand lines of code, according to Carnegie Mellon University's CyLab Sustainable Computing Consortium. In the academic community, a single unnoticed bug in parsing and plotting scripts can result in publishing incorrect results.

To optimize the output, ns-3 introduced the Tracing system. The Tracing system enables selective data extracting from the core system without having to change and recompile the core system. The tracing system can selectively extract data when an item of interest changes or an event of interest happens.

2.1.1 Tracing System. The ns-3 Tracing system is a stable facility using stable APIs that avoid most of the problems inherent in bulk output mechanisms. First, the amount of output data can largely be reduced by only tracing the events of interest. This selectivity avoids I/O bottlenecks otherwise present in large simulations when everything is being dumped to the disc for post-processing. Second, tracing enables the control over the format of the output, which can highly reduce the amount of post-processing time and effort. Also, each user can add trace-hooks in the core which can then be accessed by other users, but these hooks will produce no information unless explicitly asked to do so.

The ns-3 Tracing system is based on the concepts of independent trace-sources and trace-sinks, along with a uniform mechanism for connecting them. Trace-sources are entities that can signal events that happen in a simulation and provide access to the underlying data. For instance, a trace-source can indicate when a packet is received by a net device and provide access to its contents for all interested sinks. A trace-source can also indicate when an interesting value/state change happens in a simulation, providing access to the old and the new value/state. Trace-sinks can be connected to trace-sources and they consume the information provided by trace-sources. Trace-sinks process and/or export the obtained information. Each trace-sink can be connected to zero or more trace-sources. That way, many users can use the same information provided by the same trace-source for their own different purposes. Moreover, the tracing system introduces a very small execution overhead, and adding new trace-sources does not require the compilation of the entire core.

The tracing system is based on the Callback system in ns-3. This callback system makes use of pointers-to-functions and represents a standard mechanism which enables one piece of code to call a function or method without any inter-module dependency. In essence, a trace sink is a callback. When a trace sink connects to a trace source to receive trace events, it adds itself as a callback to a list of callbacks internally held by the trace source. When a traced event occurs, the trace source uses the Callback system to provide zero or more arguments to all trace sinks in its list of callbacks. Tracing system provides a straightforward way of accessing internal variables or states from a simulation scenario in only a few lines of code [5].

2.2 Visualization Tools

Two visualizers for ns-3 exist up to date, PyViz [2] and NetAnim [6]. However, none is preferred over the other and both have certain scope limitations.

2.2.1 PyViz. ns-3 PyViz is a live simulation visualizer (Figure 1), meaning that it uses no trace-files. It can be useful for debugging purposes, i.e. to figure out if mobility models are what you expect, where packets are being dropped, etc. There is also a built-in interactive Python console that can be used to debug the state of the running objects. Although it is mostly written in Python, it works both with Python and pure C++ simulations. PyViz has been integrated into mainline ns-3, starting with version 3.10.

PyViz includes the FlowMonitor module that makes it easier to collect and save a common set of network performance metrics. The module automatically detects all flows passing through the network and stores a number of statistical data as shown in the class diagram in Figure 2.

This tool enables measuring the performance of network protocols but not overall network performance.

2.2.2 NetAnim. NetAnim is an offline animator which animates packets over wired and wireless links using an XML trace collected

[1] https://github.com/imec-idlab/ahVisualizer

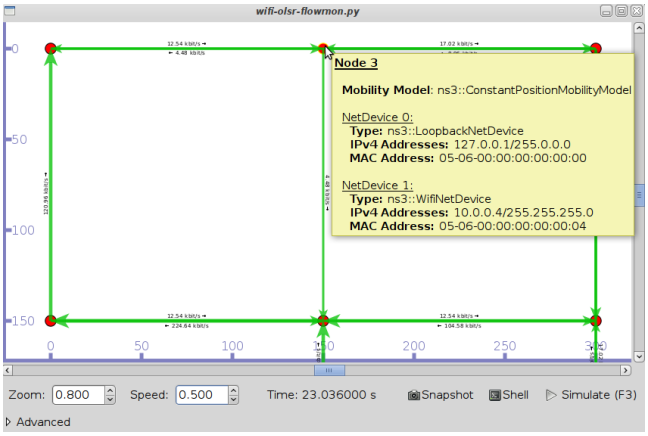

Figure 1: PyViz shows packet flows and basic NetDevice information [2]

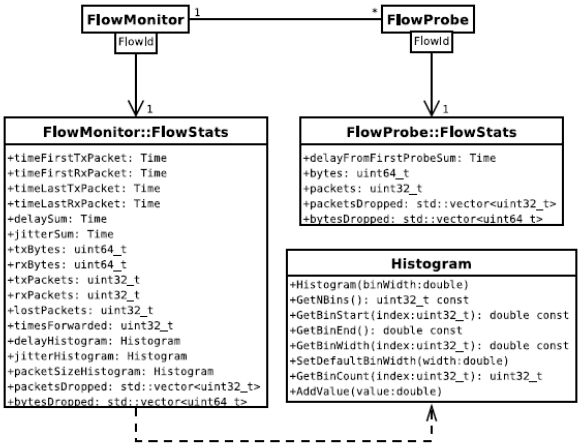

Figure 2: FlowMonitor class diagram and available statistics [3]

during the ns-3 simulation. It offers a number of features such as plotting of node-position statistics with node-trajectory (path of a mobile node), printing brief packet meta-data, parsing flow-monitor XML files and displaying statistics for each flow, showing IP and MAC information, displaying double or uint32 valued counters versus time for multiple nodes in a chart or table, printing routing tables at nodes at various points in time, stepping through one event at a time in a simulation and pause the simulation at a desired moment.

NetAnim is a handy tool indeed, but it is still limited to a single finished simulation. Complex simulations in ns-3 can take hours to complete and, as NetAnim cannot support visualization during the simulation, it does not provide the insight in trends and values of interest before the simulation is done. Another large disadvantage of existing visualizers is that they do not allow comparison of different simulations, e.g. with different PHY setup.

3 AHVISUALIZER

As our research on IEEE 802.11ah requires extensive simulations of very dense networks (up to 8192 nodes) with different PHY and MAC configurations, we needed a quick and effective way to compare several simulations against each other, analyze a large number of simulations and occasionally monitor them. Given that neither NetAnim nor PyViz enable neither overall insight in the simulated network nor post-processing for a set of simulations, we developed a new tool to cover our needs - the ahVisualizer. This tool uses the Tracing system of ns-3 to collect data, forwards the traced data each second via TCP to a NodeJS server that serves WebSocket clients with the data. It visualizes ns-3 simulations, offers offline comparison of a set of simulations and provides an interface for quick and easy analysis of a large number of simulations. The three aforementioned features of the ahVisualizer are described in this section, along with the overall program structure.

3.1 Features

The ahVisualizer currently supports ns-3 version 3.25 with 802.11ah Wi-Fi. It is custom designed to animate the TIM Segmentation and RAW mechanisms on the MAC layer of 802.11ah. Besides, it also shows the network topology and plots the time changes of many metrics for each node, but also the mean values and standard deviations of all the metrics for the whole network.

3.1.1 Visualization during Simulation. Live monitoring of ns-3 simulations can be especially handy when an ns-3 user accidentally mis-configures his simulation scenario for a long lasting simulation. Live charting enables users to immediately notice unexpected network behavior that suggests wrong setup. In very long simulations that can take hours, noticing wrong results in the beginning of the simulation can save vast amount of time. Live visualization with the ahVisualizer is shown in Figure 3. The live simulation website consists of seven parts:

(1) *Topology map* shows the positions of the nodes in the network. Metrics in the Node statistics table (3) refer to the node that is selected in the Topology map. If no node is selected, then the Node statistics table shows mean values and standard deviations of measurements for all nodes. The color code of the nodes is related to the selected metric in the Node statistics table. The color of the nodes corresponds to the performance scale of each metric, namely red-yellow-green refers to bad-medium-good scale (i.e. if "Packet loss" is selected in the Node statistics table, nodes with little packet loss are considered good and thus shown in green, whereas if "Throughput" is selected, nodes with low throughput are considered bad, thus shown in red). If no performance scale is defined for the selected metric, the color of nodes is black. Concentric circles in the map indicate the distances in the network. The difference between two neighboring radii is $100m$.
(2) *Configuration table* shows static configuration parameters for the simulation, current time, total channel traffic and total packet loss in the network during the simulation. Clicking

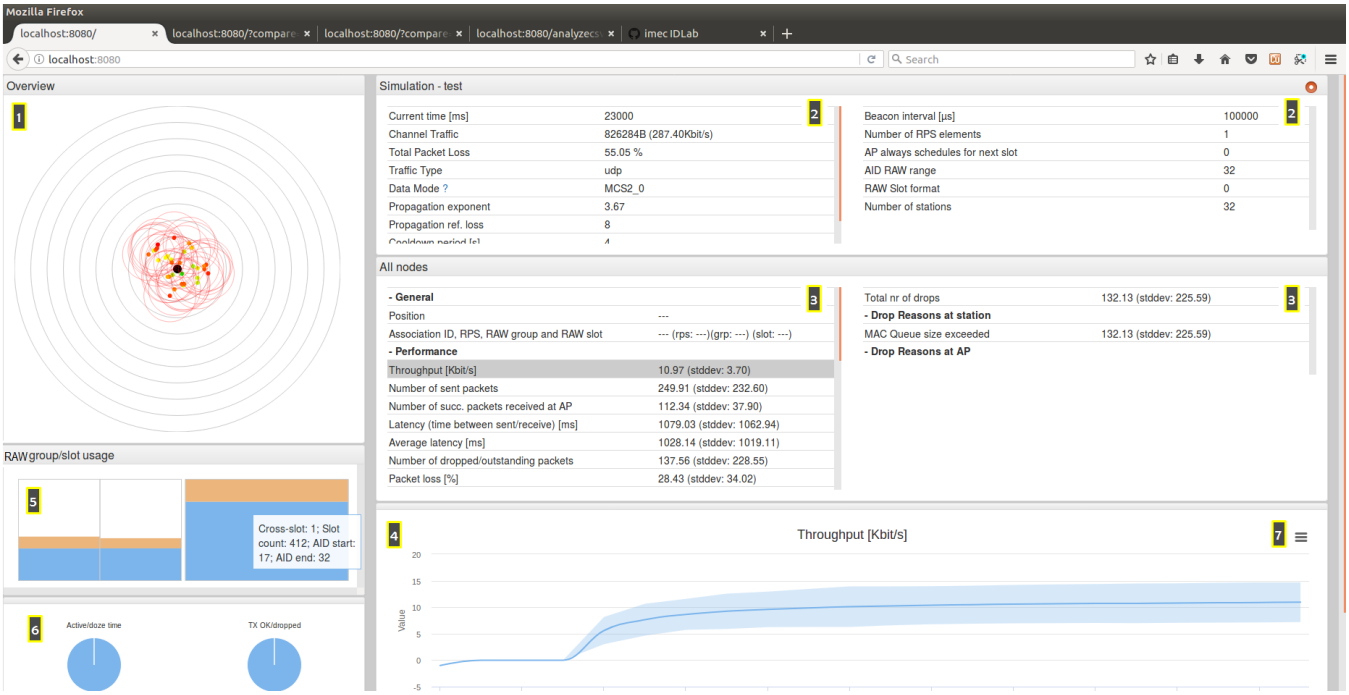

Figure 3: Live visualization of ns-3 simulation with ahVisualizer. (1) Topology map; (2) Configuration table; (3) Node statistics table; (4) Chart of the selected metric in the Node statistics table for the selected node in the Topology map. If no node is selected, mean and standard deviation for all nodes is plotted; (5) RAW group/slot usage; (6) Pie charts; (7) Menu

on "Channel Traffic" or "Total Packet Loss" field plots the diagram of total channel traffic or total packet loss over time during the simulation. Parameters shown include: Transmission Interval (TI), TI deviation, protocol, Modulation and Coding Scheme (MCS), propagation loss exponent, maximum time of packets in queues, beacon interval, total number of stations and number of stations using RAW.

(3) *Node statistics table* shows live measurements for the selected node or all nodes during the simulation. By default, the list of these measurements is shrunk, showing only relevant measurements for the simulated scenario. Expanding the list shows all measurements supported by the visualizer, however not all of them are measured in every simulation. The table has several drop-down headers: General, Performance, Transmission, Reception, TCP, AP Packet scheduling, Application, Drop Reasons at station and Drop Reasons at AP. Each of those headers hides non-relevant parameters for the run simulation, i.e. TCP statistics are irrelevant for simulations with UDP traffic, therefore the TCP statistics are hidden in that case.
Beside the position, Association Identifier (AID) and RAW configuration, supported metrics are: throughput, latency, packet loss, number of sent/received/echoed/dropped or outstanding packets, round trip time, jitter, number of missed beacons, total transmit (TX) time, number of TXs successful/dropped, EDCA queue length, number of MAC TX RTS fails/missed ACKs/collisions, inter-packet delay, number of TXs during RAW slot, number of TXs cancelled due to crossing RAW slot boundary, total receive (RX)/doze/active time, number of RXs successful/dropped/dropped by destination, TCP connection status, congestion window, Retransmission Timeout (RTO), number of TCP RTOs/TCP RTOs from AP, slow start threshold, estimated bandwidth, number of packets scheduled to the next slot by the AP/sent immediately by the AP, average remaining slot time when sending packet immediately, average sending/receiving rate and number of drops by reason at station or AP.

(4) *Chart* shows the time diagram of the selected metric from the Node statistics table for the selected node at the Topology map. If no node is selected on the Topology map, then Chart shows mean value and standard deviation of the selected metric for all nodes.

(5) *RAW group/slot usage* animates live traffic in RAWs. RAWs are depicted as slotted rectangles separated by white space. All RAW groups placed horizontally next to each other belong to the same RAW Parameter Set (RPS) element, i.e. they are located in the same beacon interval. Ergo, each RPS element is illustrated as a set of RAW groups horizontally placed one next to each other. Two RAWs belonging to the same RPS are depicted in the Figure 3.
The blue bar in a RAW slot represents the percentage of the slot duration used for uplink traffic, whereas the orange bar on top of the blue bar represents the percentage of the slot duration used for downlink traffic. White space on top of the

orange bar in the slot represents the percentage of unused time.
When hovering over a RAW group, a part of the RAW group configuration is shown, namely the values of cross-slot boundary, slot count, AID start and AID end fields.

(6) *Pie charts* show percentages of active versus doze time and successful versus unsuccessful transmissions.
(7) *Menu* offers additional options related to the plotted data, namely print chart, download PDF, SVG, PNG or JPEG image, Download CSV or XLS and view data table on the website.

3.1.2 Offline Comparison of Simulation Results. The ahVisualizer supports comparison of metrics of interest across different simulations. During an ns-3 simulation, the ahVisualizer stores the simulation data in a plain text file which can then be loaded to the ahVisualizer to compare the metrics of interest between a set of different simulations.

Metrics in the Node statistics table refer to the selected simulation from the set of loaded simulations. Next to each value in the table there is an up- or down-pointing arrow indicating whether the value increased or decreased in comparison to the average of the corresponding metrics from all the other loaded simulations. The color of the circle next to the arrow represents the z-score, namely the signed number of standard deviation by which the value of a measurement is above the mean value of measurements from the entire set of simulations simulations. When hovering across both the circle and the arrow, the exact values of the z-score and the average for the corresponding metric are shown respectively.

3.1.3 Data Analysis. Analysing a large set of ns-3 simulation results (order of magnitude of thousands) is complex and time-consuming. When assessing the impact of various MAC configurations by means of multiple ns-3 simulations, a researcher might want to compare e.g. the average throughput in the network achieved for these different MAC configurations. Additionally, he might want to see the influence of each individual MAC parameter he varied in across different MAC configurations on the average throughput. This requires writing a parsing script to extract the average throughput for each individual simulation, extracting the MAC parameters of interest for all simulations and once the data is obtained in a meaningful way - plotting. Ergo, for a single plot that includes data from a set of simulations a researcher would need to invest quite some time and effort. In order to avoid that, we constructed a framework for easy analysis of a large set of ns-3 simulation results.

Figure 4 shows the website for data analysis. After loading a CSV file, it is possible to select any of the varied variables or measurements as an X or Y axis, or as a series. The rest of the variables and measurements can then be fixed if desired, or left general. After choosing what to plot and which variables to fix, a single click on the "Generate" button results in the desired series plot as illustrated in Figure 4.

We generate a CSV file that contains the results of a set of ns-3 simulations in a uniform way using the aforementioned files in which individual simulation data is previously stored. We developed a Perl script to process each individual file from the desired batch of files and compress the data from each file to a single line in CSV. Three Perl command line arguments are needed: (1) a folder containing a set of chosen files for analysis, (2) names of the desired configuration variables and (3) names of the desired measurement variables. The Perl script will then extract the desired configuration parameters, along with maximum, minimum, median and average values of the desired measurements from each processed file and store it to the CSV file. This CSV file can then be used for analysis with the ahVisualizer.

3.2 Architecture Overview

The ahVisualizer program consists of three components, namely the ns-3 component, the NodeJS web server and the website. The three components, their connections and the overview of the new classes in each component are illustrated in Figure 5. An illustrative example of the *SimulationEventManager* usage in the *main* function in ns-3 is presented in Algorithm 1.

In the ns-3 component, we implemented several new classes in order to handle the traced data in a structured manner, to serialize it and to send it to the ahVisualizer. Figure 6 illustrates ahVisalizer's ns-3 component. All trace-sinks are implemented in *NodeEntry* class to collect traced data. It is necessary to assign a *NodeEntry* instance to each simulated node. *NodeEntry* class stores basic information of the node relevant for the ahVisualizer, such as position, AID, association status, queue length etc. *NodeEntry* also updates *NodeStatistics* class which stores all the relevant information measured by the tracing system for a node. The *Statistics* class contains *NodeStatistics* of each node along with other general information about the network, such as number of nodes and time when association is finished. Each second, a snapshot of *Statistics* is sent to the server. *SimulationEventManager* manages the simulation events, serializes them, and through the *SimpleTCPClient* establishes a TCP connection to the NodeJS server and forwards the data to the server.

Algorithm 1 High-level overview of `main ()`

```
initialize Configuration, Statistics, SimulationEventManager;
SimulationEventManager.send (Configuration);
for node ∈ NodeContainer do
    n ← NodeEntry
    nodeEntryContainter ← nodeEntryContainter ∪ n
    connect trace sinks for n;
    n.position ← node.position
    SimulationEventManager.send (n.position)
end for
SimulationEventManager.send (AP.position)
loop
    for node ∈ NodeContainer do
        if node.isAssociated then
            SimulationEventManager.send (n.RawData)
        end if
    end for
    SimulationEventManager.send(Statistics)
end loop
```

The NodeJS web server acts as a host for the simulation data received from ns-3. The server forwards the data to the web-browser clients via WebSocket in case of live data. It also writes simulation data to a file which can later be retrieved by the clients to compare

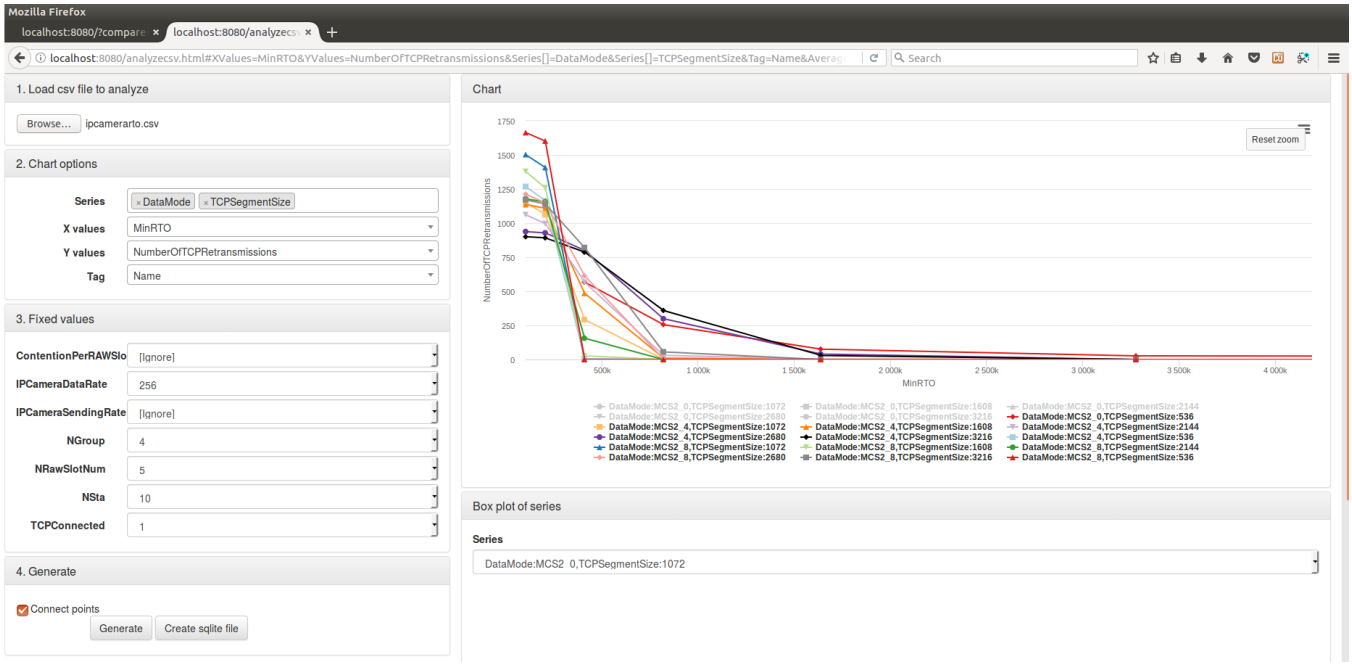

Figure 4: Web interface for quick and easy analysis of large ns-3 simulation sets

simulations against each other. When a web-browser client subscribes to multiple streams, i.e. simulations, the server will send each stream in series either by reading out the stored files or forwarding the live data in case of live simulation monitoring. *SocketManager* class keeps track of all active WebSockets.

A web-client, whose IP is set in *SimulationEventManager*, receives the simulation data on port 8080 and visualizes it. In live simulations, the simulation data is updated every second. *EventManager* parses the data received as events and updates the simulation statistics. It features an event queue where all events are stored and processed in the main update loop. If too many events are in the queue, *EventManager* will force processing of the events. *SimulationGUI* class manages the GUI, i.e. draws the nodes, updates the properties of the selected node (or averages when no node is selected) and updates the RAW slot usage. *Charting* class provides methods for the GUI to update the charts.

A web-client for data analysis analyzes the data from a CSV that is previously constructed from a set of NSS files. It reads the loaded CSV file, provides all its attributes in drop-down lists to choose axes and series variables and plots the configured series as illustrated in Figure 4.

The base path "/" will subscribe to the live simulation data. To compare a set of simulations offline, it is necessary to form a query-string with comma-separated names of simulations to be compared, e.g. "/?compare=simName1,simName2,simName3". The "/analyzecsv.html" path will retrieve the page for the analysis of a CSV.

4 CONCLUSIONS

This paper presents the new user-friendly tool for on- or offline visualization of ns-3 simulations with 802.11ah Wi-Fi, as well as quick and easy analysis of large number of simulation results. The ahVisualizer provides web-interfaces for (1) live simulation monitoring, (2) offline comparison of simulation results and (3) analyzing data from large sets of simulations. It forwards the traced data from an ns-3 simulation to a web server, which stores it for post-processing and forwards it to the web client for visualization. With only a few clicks, ahVisualizer can plot any desired chart consisted of data obtained from simulations. This tool avoids the need for parsing simulation outputs altogether, therefore it reduces the probability of parsing errors which could occur each time a new ns-3 experiment is conducted and new different simulation output is obtained and needs to be parsed.

4.1 Future Work

Although the ahVisualizer makes monitoring and analysis of ns-3 simulations substantially quicker and easier, the tool is custom made for the IEEE 802.11ah module, therefore it has very limited usage. We intend to generalize this tool to support any ns-3 simulation. As it is based on the Tracing system, most of the existing functionalities could be generalized. Extending the *Node* class in ns-3 to store desired *NodeStatistics*, systematically organizing desired trace-sinks in the *NodeEntry* class, slightly generalizing the *SimulationEventManager* events and the website design would provide a configurable and scalable tool for live monitoring and analysis of any traced data in ns-3 simulations.

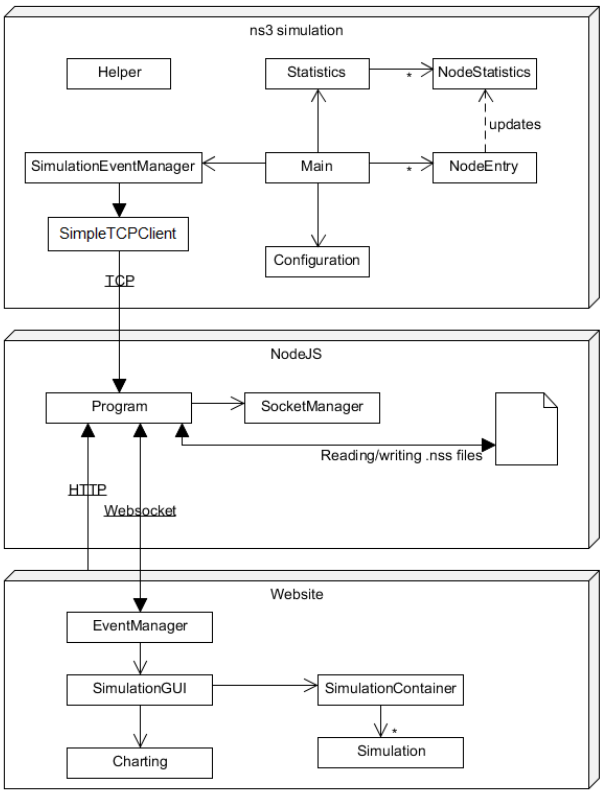

Figure 5: Program structure of ahVisualizer: compoments and classes

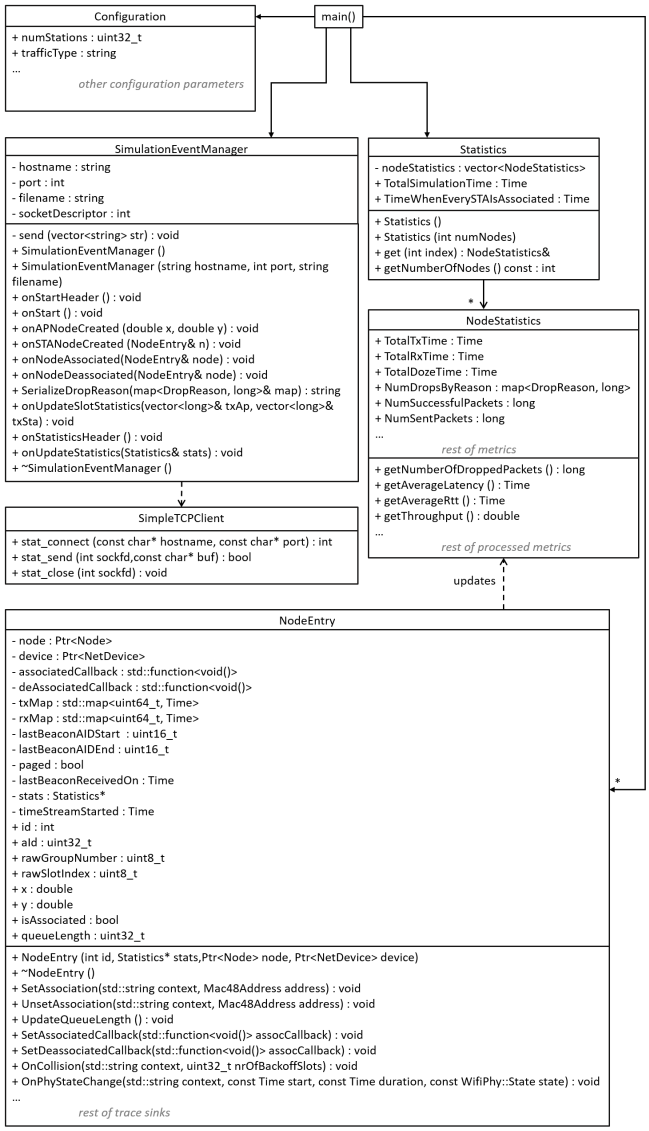

Figure 6: Overview of the ns-3 component

ACKNOWLEDGMENTS

Part of this research was funded by the Flemish FWO SBO S004017N IDEAL-IoT (Intelligent DEnse And Longe range IoT networks) project.

REFERENCES

[1] 2017. IEEE Standard for Information technology–Telecommunications and information exchange between systems - Local and metropolitan area networks–Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 2: Sub 1 GHz License Exempt Operation. *IEEE Std 802.11ah-2016 (Amendment to IEEE Std 802.11-2016, as amended by IEEE Std 802.11ai-2016)* (April 2017), 1–594. https://doi.org/10.1109/IEEESTD.2017.7920364

[2] G. Carneiro. 2015. PyViz. (2015). Retrieved January 11, 2018 from https://www.nsnam.org/wiki/PyViz

[3] G. Carneiro, P. Fortuna, and M. Ricardo. 2009. FlowMonitor – a network monitoring framework for the Network Simulator 3 (ns-3). In *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS '09)*. ACM, Pisa, Italy.

[4] ns-3 Network Simulator 2017. *ns-3 Manual.* ns-3 Network Simulator. https://www.nsnam.org/docs/release/3.27/manual/ns-3-manual.pdf.

[5] ns-3 Network Simulator 2017. *ns-3 Tutorial.* ns-3 Network Simulator. https://www.nsnam.org/docs/release/3.27/tutorial/ns-3-tutorial.pdf.

[6] G. Riley and J. Abraham. 2014. NetAnim 3.104. (2014). Retrieved January 11, 2018 from https://www.nsnam.org/wiki/NetAnim_3.104

[7] A. Sljivo, D. Kerkhove, L. Tian, J. Famaey, A. Munteanu, I. Moerman, J. Hoebeke, and E. De Poorter. 2018. Performance Evaluation of IEEE 802.11ah Networks With High-Throughput Bidirectional Traffic. *Sensors* 18, 325 (2018).

[8] L. Tian, S. Deronne, S. Latré, and J. Famaey. 2016. Implementation and Validation of an IEEE 802.11ah Module for ns-3. In *Proceedings of the Workshop on ns-3 (WNS3)*. Seattle, Washington, US, pp. 49–56. https://doi.org/10.1145/2915371.2915372