Visual-Netsim: Development of an Interactive Browser based Network Simulator

Omar Shaikh
Dhahran High School
Al-Khobar 31952, Saudi Arabia
Email: shaik.o.418@isg.edu.sa

Farrukh Shahzad
Polestar Global
Houston , TX 77082, USA
Email: farrukh.shahzad@polestarglobal.com

***Abstract*—Browser based simulators for wireless networks remain an unheard concept. Although wireless networks are utilized in various applications and are providing the backbone for the Internet of Things (IoT), it remains fairly difficult to quickly prototype a visualization. The need for an easily accessible simulator is a necessity, sparked by the alarming growth of sensor networks and the IoT. The development of a reliable and robust large-scale wireless system requires that design concepts are visualized in some digital form, before implementation. Simulations provide a cost effective and feasible method of examining the correctness and scalability of the system before deployment.**

This work presents a lightweight and interactive topology generator as part of a novel web based network simulator. The simulation framework, to the best of our knowledge, is unlike any other, as it renders and generates graphs in the browser. A simple GUI allows the user to generate isotropic and anisotropic topologies of different shapes, and provides tools that help manipulate complex graphs on client browsers and mobile devices. This tool has its application in several domains including computer network configuration and management, sensor and ad-hoc networks, medical infrastructure, biological and social media visualizations. This application can also facilitate research and education in the related areas.

***Keywords*—Internet of Things; Network Topology; Visualization; JavaScript; Web based Application**

I. Introduction

Wireless Sensor Networks (WSNs) are widespread and used in important applications as diverse as intrusion detection, object tracking, telecommunication networks, industrial/home automation, smart structure, and medical tracking devices. WSNs usually require visualizations before they are implemented [1]. Simulators often carry unnecessary tooling, or require complicated setup procedures. The simulation environment for WSNs include either greenfield or brownfield developments. Brownfield developments extend simulation environments that existed before the idea of WSNs emerged. These simulation environments are improved to support wireless functionality and adapted for use with WSNs. Greenfield developments, however, consider WSN specific characteristics from their onset [2].

Simulators can be divided into three major categories based on their level of complexity: algorithm level, packet level, and instruction level. Notable algorithm level simulators include NetTopo [3], Shawn [4] and AlgoSensim [5].

The primary contribution of this work is the design and implementation of a generic topology generator, as part of an interactive browser-based network visualizer and simulator called *Visual-Netsim*. It is a lightweight utility that allows for quick and powerful web-based graph visualizations.

The rest of the paper is organized as follows: In section II, we provide some background related to existing simulation tools. We present our work in section III, and explain the graph generation algorithm in section IV. We describe the user interface of the presented application in section V. In section VI, we provide several examples of topology generation and manipulation. We conclude in section VII.

II. Background and Related Work

Simulation has always been useful for wireless network-related research. A large number of simulators have been proposed for wireless ad hoc networks. These simulators have different design goals and largely vary in the level of complexity. They support several hardware and communication layer assumptions, focus on many distributed network implementations and environments, and come with a diverse set of tools for modeling, analysis, and visualization. Classical simulation tools include NS-2/3, OMNeT++, J-Sim, OPNET, TOSSIM, and others [2], [6], [7].

OMNeT++ is a discrete event simulation environment. Its primary application area is the simulation of communication networks, but because of its generic and flexible architecture, OMNet++ is successfully used in other areas like the simulation of complex IT systems, and the queuing of networks or hardware architectures [8], [9].

Pymote is an example of a high level Python library for the event based simulation of distributed algorithms in wireless networks [10]. The library allows the user to implement sensor network algorithims using Python - a popular, easy to learn, full featured, object oriented programming language. The library focuses particularly on quick prototyping and approaches networks at algorithm level. Pymote avoids specification overhead using predefined distributed computing environments. Pymote has been extended, by one of the authors, to include a generic topology generator module [11].

III. Development of a new Simulator

Existing simulators run on the client's machine as an application and require an installation process. *Visual-Netsim* begins by eliminating this overhead, as the development of the simulator involves the use of JavaScript, a web-based

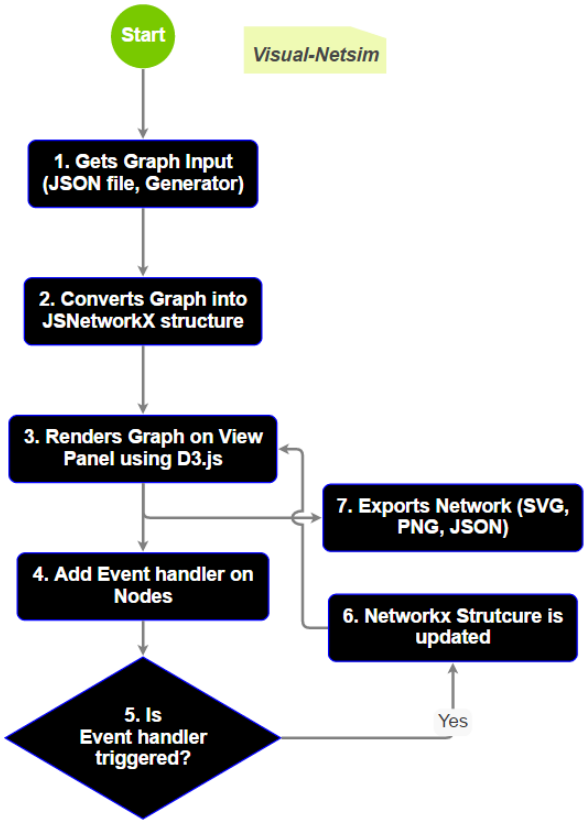


Fig. 1: The *Visual-Netsim* Application Flow

programming language. JavaScript also helps introduce the familiarity of a typical website and allows the user to begin a quick prototyping process. *Visual-Netsim* avoids the use of closed-source flash based animation libraries and instead uses Scalable Vector Graphics (SVG), an XML based format for displaying and storing 2D vectors. We utilize the following web technologies, third-party libraries, and frameworks to design and implement *Visual-Netsim*.

- **HTML5** is a larger set of technologies (including CSS3) that powers diverse Web sites and applications.
- **jQuery** is a fast, small, and feature-rich JavaScript library which allows users to write less, and do more [12].
- **D3.js** is a JavaScript library for manipulating documents and SVG elements, based on data [13].
- **JSNetworkX** is a port of the popular python NetworkX library. JSNetworkX brings various graphing algorithms and structures from NetworkX [14].
- **Bootstrap** is a popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web [15].

Visual-Netsim is developed to accompany the following 3 methods for importing and loading network data.

- Interactive Generation based on various parameters. This input is processed by *Visual-Netsim* on the client itself. Custom parameters for graph generation include communication range, graph dimensions, number of nodes, and obstacles. This will be described in the next section.
- Load pre-generated topologies from server (JSON format). The server is run on Node.JS, a simple server-side framework. Its role is to expose an API to the front-end. The API consists of two endpoints: a $/data$ endpoint that returns all pre-generated files, and a $/data/:filename$ endpoint, where the server returns a specified JSON file defined by the $:filename$ parameter.
- Load user-defined custom topologies from the client (JSON format). This option allows the user to import a custom JSON file describing the graph structure. Importing a custom file allows the user to predefine links, nodes, and general graph settings. In order to be accepted by *Visual-Netsim*, the file should be composed of various objects, including arrays of nodes and links object.

The nodes are stored in an array, each element containing a node object that has node specific attributes. This includes x and y coordinates, color, shape, and size, which is rendered by D3.js on runtime. Links are stored in another array, and are formatted using an object with a source and target key. Each individual links source refers to the index of a node in the node array. The same logic applies to the target key. General graph data is an object in the same file. This object includes information about the graph, including overall size (width and length in pixels), node dragging ability, communication radius, and graph name. *Visual-Netsim* reads this data immediately after importing the file, setting up the D3 canvas for the nodes to be displayed.

After Netsim sets up the initial canvas for D3 and the nodes, it transitions all data to the NetworkX graph data structure. Edges and nodes are copied first to the structure and the previous JSON file is no longer referred to. General graph data, however, is stored in a local object for future reference. Each link in the NetworkX graph is fitted with various attributes. Names are no longer tied to numerical ids based on array indexes, but are taken from the name property from each node object. Edge weight is also set based on the distance from the nodes.

An event listener, bound to the mouse-drag event, is then attached to every single node. Dragging nodes will cause a translation in the node position, depending on the direction of the dragging action. New node data automatically updates the NetworkX object, refreshing the main data structure. Not only do the node positions change, but links are recalculated and updated in the NetworkX object. The drag fires an event and uses the node as a reference point for the recalculations. If a value for the communication range is set in general network settings, then links are created and removed, based on each node's distance from the reference point.

Visual-Netsim also allows for graph exporting after a user has edited the network. Formats such as SVG, PNG, and an updated JSON file are possible export options. *Visual-Netsim* uses a simple $href$ tag and $download$ property to save the

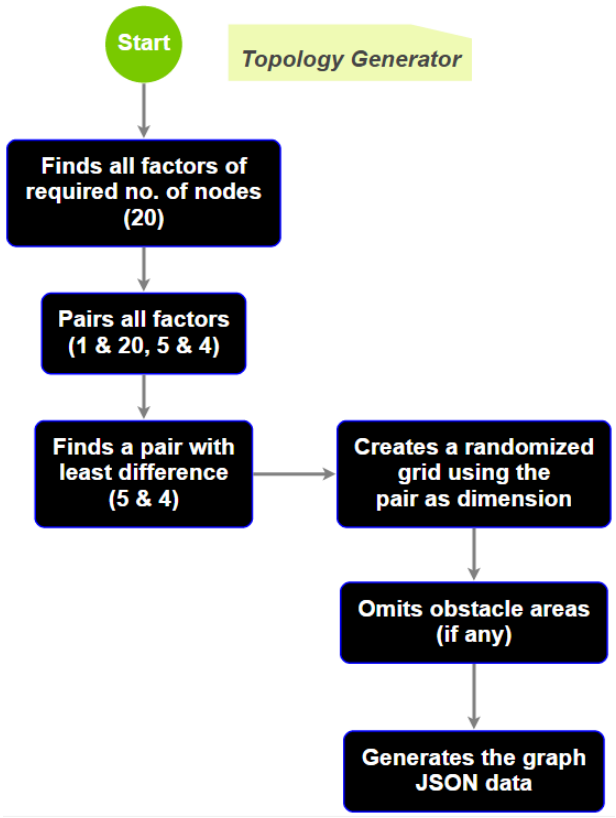


Fig. 2: Topology Generator Module

file directly to the client's machine or device, without the intervention of a server. The flowchart in Fig. 1 summarizes the application flow.

IV. Graph Generation

Visual-Netsim is capable of generating a graph on the client itself using a configurable algorithm. Since *Visual-Netsim* is a client-based application, it executes this algorithm on the user's device, not relying on a remote server for the process.

The generator uses several parameters, including communication range value, total number of nodes, the length and width of the network deployment area, and a random offset. The communication range value specifies the maximum distance between nodes in the graph for a link to form. The length and width parameters are used to evenly place nodes in the SVG element. The random factor serves as a magnitude for the offset applied (in both the x and y direction) to every node created and added to the matrix

The algorithm performs the following 4 step process when generating graphs:

- Finds multiples with smallest absolute difference that multiplies to the number specified nodes,
- Generates a matrix (or grid) with the width being the larger multiple and the height being the smaller,
- Applies a random offset (in both the x and y direction) to every node created and added to the matrix,
- Creates a Graph Object with user parameters and generated nodes.

Visual-Netsim first finds all possible factors of the number using modular arithmetic. Upon finding a new factor, it goes through the list of detected factors and finds a pair of factors that multiplies to the number of desired nodes. After storing this pair, the algorithm continues finding the remaining factors, and pairs them in a similar fashion.

The algorithm then uses the pair with the smallest absolute difference between its values. This pair is used to make a graph nearest to the shape of a rectangle. Next, *Visual-Netsim* generates the coordinates to place the nodes on an SVG canvas. It also applies the given random factor to every node placed on the coordinate, by setting an offset to both the x and y attributes of the node.

The generator also excludes locations where nodes cannot be placed. The user can create a blockade object in the graph, preventing nodes from forming in the specified area. Blockade presets are also available, allowing for the generation of S, O, and C shaped graphs.

After generating the coordinates and omitting blocked points, corresponding node objects are created. These objects are fitted with attributes provided by the user, such as color, size, and shape. A node array is then created with all the node objects. The generator then creates an object similar to a typical *Visual-Netsim* JSON file, and renders it. Fig. 2 summarizes the topology generation process. The numbers in the parenthesis show an example for 20 nodes.

V. User Interface

The *Visual-Netsim* web application has two panels (apart from the top header panel) as shown in Fig. 3. Panels can scroll horizontally or vertically (if needed), allowing a user to view hidden content. The left panel displays the network topology and the right panel shows user controls and inputs. The user control panel has four tabs as shown in Fig. 4.

The Generate Tab allows for a simple way to create any desired network topology (Fig. 4a) with the required number of nodes, dimensions, and randomness in node coordinates. Options also exist to create a network with obstacles (rectangular-shaped) of a desired size. This helps simulate practical node deployment scenarios where buildings or other structures represent obstacles or holes in the network.

In the Network Option Tab, a user can click on options to show node labels, display grid lines across the networks, or enable a dark background (Fig. 4b). Users can also load JSON files from a server or the client's device and can update existing network settings on the fly. The user also has buttons to save the current network in JSON, SVG or PNG image format, which can directly be imported into Latex and other publishing applications. The JSON file represents the current state of the topology (node position and other attributes) letting the user load it again later to continue working on the topology.

The Node Tab allows a user to edit attributes of a selected node (Fig. 4c). Nodes are selected either by clicking on the node in the network view panel or by picking it from the Info tab. The Info Tab shows some of the network parameters

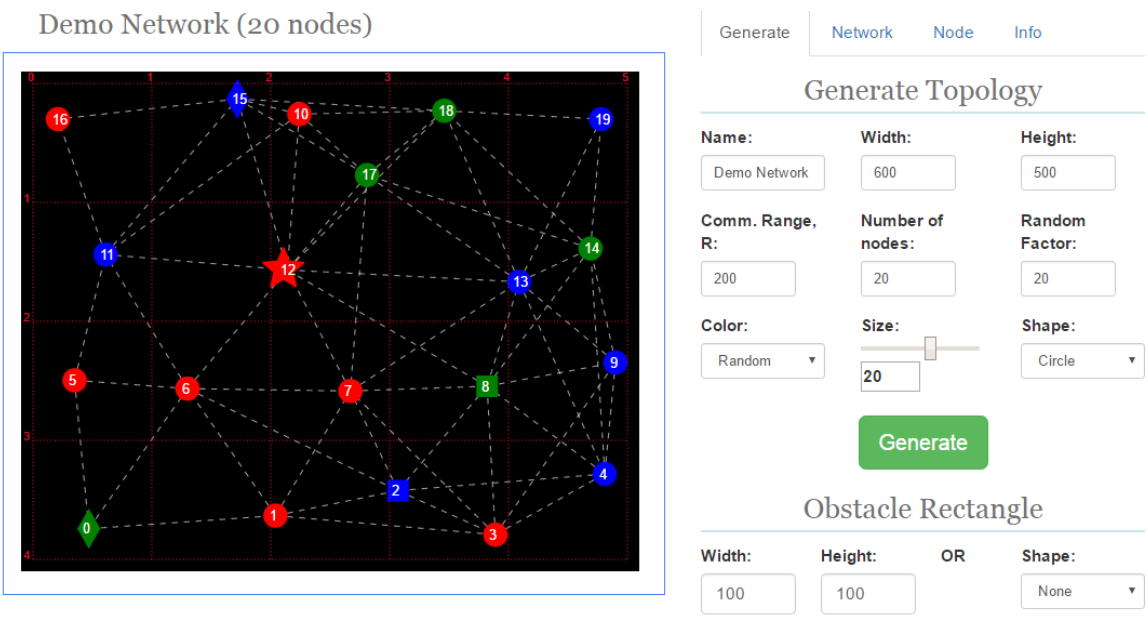
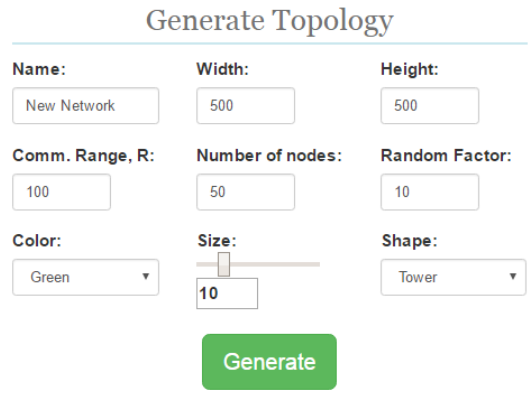
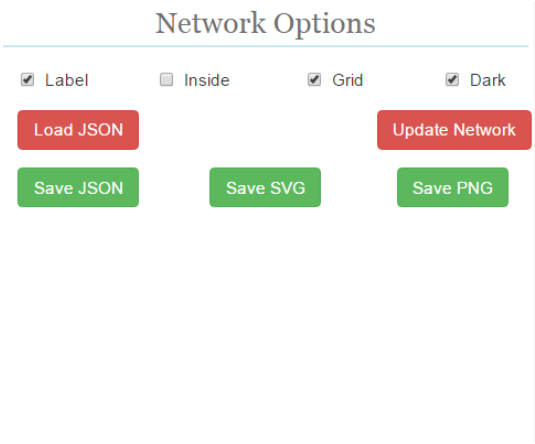


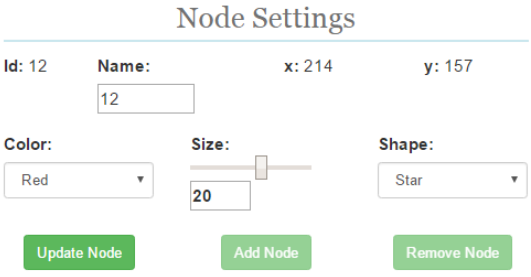
Fig. 3: *Visual-Netsim* User Interface



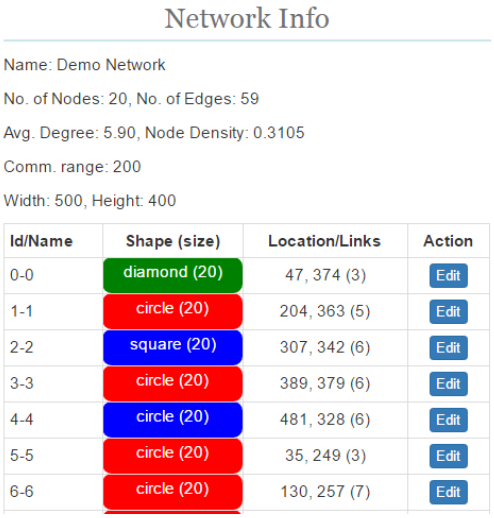
(a) Generate Topology Tab



(b) Network Option Tab



(c) Node Edit Tab



(d) Network Info Tab

Fig. 4: The four Tabs in User Control Panel

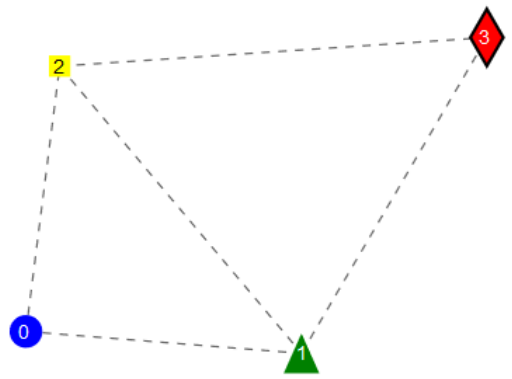


Fig. 5: As simple 4 node Network.

like number of nodes, edges, average degree and node density (Fig. 4d). A list of all nodes and their attributes are also displayed.

VI. Examples

In this section, we provide examples to highlight the user friendly and powerful features of *Visual-Netsim*. A user can design a network from scratch by selecting number of nodes and communication range of a node. User can then drag nodes in the deployment region as desired and can also add obstacles within the region. Finally, the user can modify node's attributes like label, color, shape and size as needed. We believe no comparable web-based tool exists for network visualization, where a desired topology can be created and manipulated with such a ease. The demo version is available at http://beyondccie.com/netsim/. The tool is under active development.

A. A simple 4 node Network

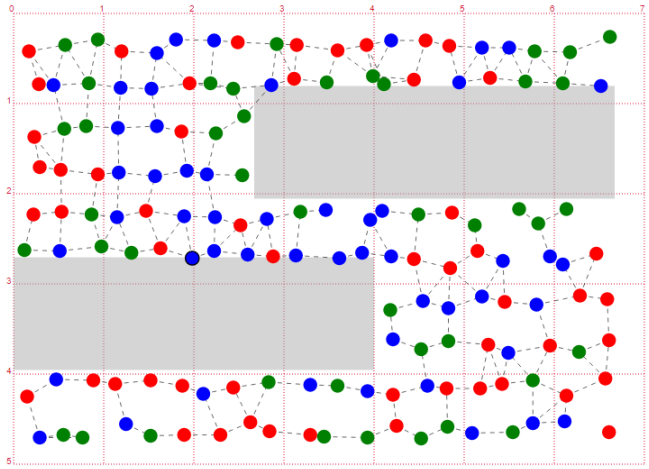
A simple network with 4 nodes is generated with the communication range set to 300 (Fig. 4a). The topology is shown in Fig. 5, with node id's labeling each node. Node 1 and 2 are within the communication range of the other three nodes; therefore, *Visual-Netsim* creates a link between them. The corresponding JSON file for the topology is shown in Listing 1. The $graph$ object contains the general parameters, the $nodes$ is an array of node objects (4 elements) and finally $links$ is an array containing 5 elements representing source and target node's id for each link.

Listing 1: JSON format for a 4-node topology

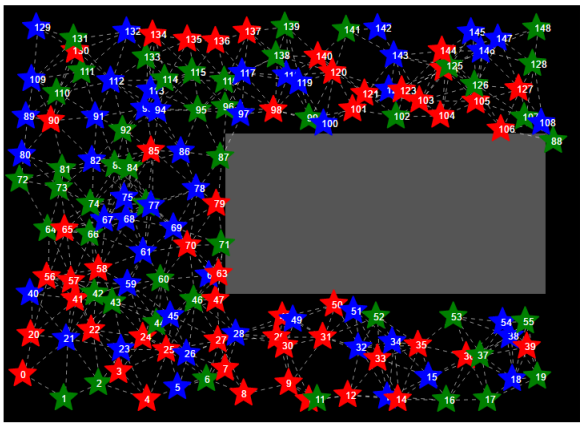
```
{
    "graph": {
        "r": 300,
        "name": "A simple 4-node Network",
        "width": 700,
        "height": 500,
        "drag": true
    },
    "nodes": [
        {
            "x": 177.99999618530273,
            "y": 240.5,
            "color": "#0000FF",
            "id": 0,
            "shape": "circle",
            "name": "0",
            "size": "19",
            "weight": 2,
        },
        {
            "x": 341.5,
            "y": 253.5,
            "color": "#008000",
            "id": 1,
            "shape": "triangle-up",
            "name": "1",
            "size": "14",
            "weight": 3,
        },
        {
            "x": 197.99999618530273,
            "y": 83,
            "color": "#FFFF00",
            "id": 2,
            "shape": "square",
            "name": "2",
            "size": "14",
            "weight": 3,
        },
        {
            "x": 451.5,
            "y": 66,
            "color": "#FF0000",
            "id": 3,
            "shape": "diamond",
            "name": "3",
            "size": "20",
            "weight": 2,
        }
    ],
    "links": [
        {
            "source": 0,
            "target": 2
        },
        {
            "source": 1,
            "target": 3
        },
        {
            "source": 2,
            "target": 3
        },
        {
            "source": 2,
            "target": 1
        },
        {
            "source": 0,
            "target": 1
        }
    ]
}
```

B. Anisotropic Network

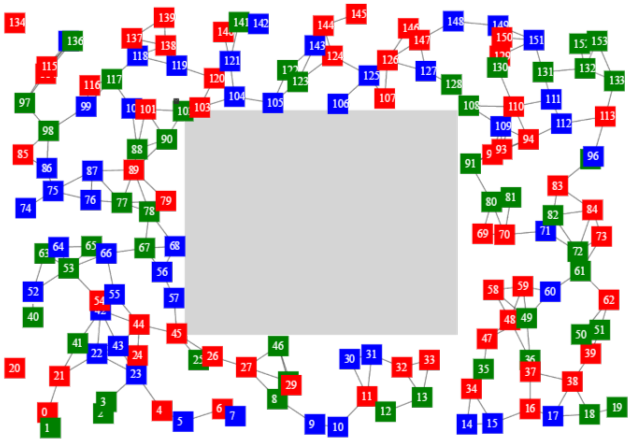
In this example, obstacle options are used to create some anisotropic topologies. These topologies visualize some practical network deployment scenarios where buildings and other structures are modeled as holes or obstacles. We use different node colors, shapes and sizes to show possible network customizations. In Fig. 6a, an S-shaped topology is generated with random colored circular nodes (total 150 nodes) with the grid overlay enabled. The two rectangular holes are shown in



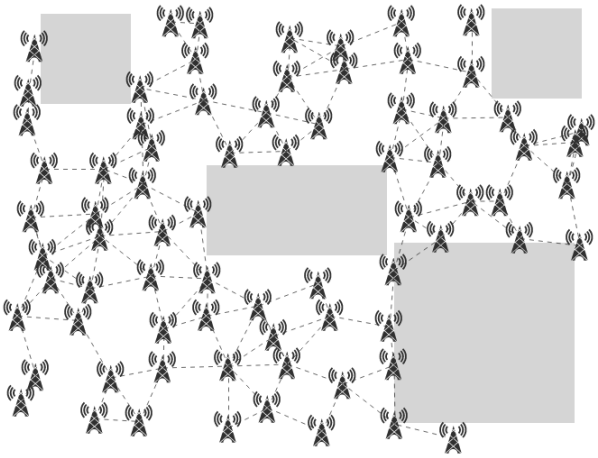
(a) S-shaped Topology



(b) C-shaped Topology

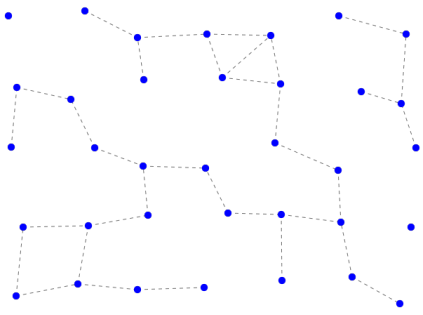


(c) O-shaped Topology

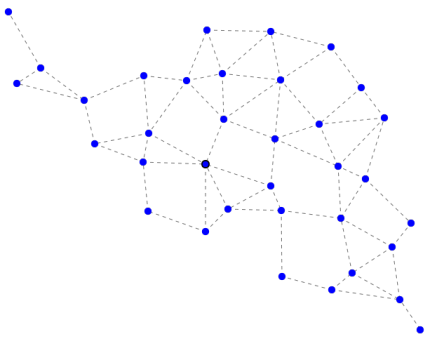


(d) Topology with several obstacles

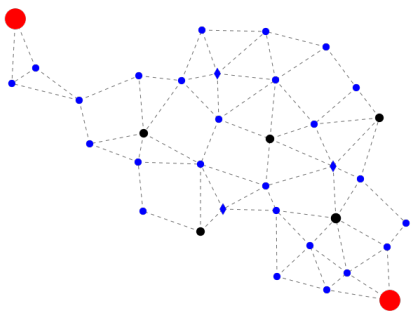
Fig. 6: Some Anisotropic Topologies



(a) A Sample mesh Network



(b) Nodes moved around by dragging them



(c) Some nodes are modified

Fig. 7: Visualization of CodeBlue Network

an opaque gray color. Similarly a C-shaped topology is created with 150 star-shaped nodes with dark background and display node labels enabled, as seen in Fig. 6b. Next, in Fig. 6c, an O-shaped topology is created by inserting a square obstacle in the middle of a 154 nodes network. Nodes are square shaped with labels inside the nodes. A network with multiple obstacles of various dimensions is also shown in Fig. 6d. Black colored tower shaped nodes are used for this network.

C. CodeBlue ad-hoc Network

Visual-Netsim can visualize networks at an algorithmic level and has applications in several other fields including biological and social media visualizations. An example is CodeBlue, an ad-hoc network used primarily for emergency health-care [16]. This case study implemented a powerful sensor network that attends to *emergency wireless infrastructure*, using any-to-any ad-hoc network [16]. CodeBlue lacked a coordination system that could visualize the large multitude of sensors in an easy to use, high level manner [16]. *Visual-Netsim* delivers a potential solution for visualizing the array of sensors, and displaying how they connect in an any to any network system.

A sample mesh network, similar to the one used in CodeBlue, can be created by simply generating a graph with blue nodes, as shown in Fig. 7a. The algorithm readily generates a graph that complies with the parameters. The nodes can then be selected and moved simply by clicking and dragging (Fig. 7b). The event handler for each node is emitted, causing the recalculation of links in the graph. Attributes can be set on any node in the graph. A simple click on the node selects it and allows the user to change its attribute through Node Edit Tab (Fig. 7c). This feature is especially useful when nodes represent a multitude of devices. For CodeBlue, location beacons or patients are smaller (blue) nodes, and fixed terminals or PDAs are bigger (red) nodes.

VII. Conclusions

The development of a reliable and robust large-scale WSN system requires that the design concepts are checked and optimized before they are implemented and tested for a specific hardware platform. Simulation provides a cost effective and feasible method of examining the correctness and scalability of the system before deployment.

This work introduced a new simulator that prioritized ease of use, and performed network simulation at algorithmic level. The implementation process of the new web-based simulator used D3.js, a powerful library that allowed SVG manipulation. JSNetworkX provided the basis for implementing various graph algorithms, and served as a universal graph data store. In this paper, we focused on topology generator module to generate and visualize several isotropic and anisotropic networks based on desired connectivity, network density and communication range.

As future work, we will continue building on the proposed simulation framework to enhance the topology generator module as well as add algorithm level simulation related to mobility, localization, routing and clustering.

References

[1] A. Abuarqoub, F. Alfayez, M. Hammoudeh, T. Alsboui, and A. Nisbet, "Simulation Issues in Wireless Sensor Networks: A Survey," in *SENSORCOMM 2012 , The Sixth International Conference on Sensor Technologies and Applications*, aug 2012, pp. 222–228.

[2] Q. Ali, A. Abdulmaowjod, and H. Mohammed, "Simulation & performance study of wireless sensor network (WSN) using MATLAB," in *Power and Control (EPC-IQ), 1st International Conference on*, 2010, pp. 307–314.

[3] L. Shu, M. Hauswirth, H.-C. Chao, M. Chen, and Y. Zhang, "NetTopo: A framework of simulation and visualization for wireless sensor networks," *Ad Hoc Networks*, vol. 9, no. 5, pp. 799–820, jul 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570870510001435

[4] A. Kroeller, D. Pfisterer, C. Buschmann, S. P. Fekete, and S. Fischer, "Shawn: A new approach to simulating wireless sensor networks," p. 10, feb 2005. [Online]. Available: http://arxiv.org/abs/cs/0502003

[5] F. J. &. A. Marculescu, "AlgoSenSim - Overview [TCS-Sensor Lab]," 2006. [Online]. Available: http://tcs.unige.ch/doku.php/code/algosensim/overview

[6] A. Sobeih, "J-Sim: A Simulation and emulation environment for wireless sensor networks," *IEEE Wireless Communications*, vol. 13, no. 4, pp. 104–119, aug 2006. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1678171

[7] Y. Tselishchev, A. Boulis, and L. Libman, "Experiences and Lessons from Implementing a Wireless Sensor Network MAC Protocol in the Castalia Simulator," in *2010 IEEE Wireless Communication and Networking Conference*. IEEE, apr 2010, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5506096

[8] A. Varga and R. Hornig, "An overview of the OMNeT++ simulation environment," *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems*, pp. 60:1—-60:10, mar 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1416222.1416290

[9] A. Boulis, "Castalia - Wireless Sensor Network Simulator," 2011. [Online]. Available: https://castalia.forge.nicta.com.au/index.php/en/index.html

[10] D. Arbula and K. Lenac, "Pymote: High Level Python Library for Event-Based Simulation and Evaluation of Distributed Algorithms," *International Journal of Distributed Sensor Networks*, vol. 2013, no. 797354, p. 12, 2013.

[11] F. Shahzad and T. R. Sheltami, "An efficient MAC scheme in wireless sensor network with energy harvesting (EHWSN) for cloud based applications," in *2015 IEEE 40th Local Computer Networks Conference Workshops (LCN Workshops)*. IEEE, oct 2015, pp. 783–788. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7365928

[12] (2016, May) jQuery. [Online]. Available: https://jquery.com/

[13] M. Bostock, V. Ogievetsky, and J. Heer, "D3: Data-Driven Documents." *IEEE transactions on visualization and computer graphics*, vol. 17, no. 12, pp. 2301–9, dec 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=2068462.2068631

[14] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX," in *Proceedings of the 7th Python in Science Conference (SciPy2008)*, Pasadena, CA USA, Aug. 2008, pp. 11–15.

[15] (2016, May) Bootstrap - The world's most popular mobile-first and responsive front-end framework. [Online]. Available: http://getbootstrap.com/

[16] D. Malan, F. Thaddeus, M. Welsh, and S. Moulton, "{CodeBlue\\: An Ad Hoc Sensor Network Infrastructure for Emergency Medical Care}," pp. 12 – 14, 2004.