Wireless Sensor Network Simulators A Survey and Comparisons

Harsh Sundani
EECS Department,
University of Toledo,
2801 W. Bancroft, MS 301, Toledo, Ohio 43606

Haoyue Li
EECS Department,
University of Toledo,
2801 W. Bancroft, MS 301, Toledo, Ohio 43606

Vijay Devabhaktuni vijay.devabhktuni@utoledo.edu
EECS Department,
University of Toledo,
2801 W. Bancroft, MS 301, Toledo, Ohio 43606

Mansoor Alam
EECS Department,
University of Toledo,
2801 W. Bancroft, MS 301, Toledo, Ohio 43606

Prabir Bhattacharya
Department of Computer Science,
University of Cincinnati,
814 Rhodes Hall, Cincinnati, OH 45221

Abstract

Simulation tools for wireless sensor networks are increasingly being used to study sensor webs and to test new applications and protocols in this evolving research field. There is always an overriding concern when using simulation that the results may not reflect accurate behavior. It is therefore essential to know the strengths and weaknesses of these simulators. This paper provides a comprehensive survey and comparisons of various popular sensor network simulators with a view to help researchers choose the best simulator available for a particular application environment. It also provides a detailed comparison describing the pros and cons of each simulator.

Keywords: Wireless sensor networks, Simulator, Emulator, Comparison, Performance evaluation.

1. INTRODUCTION

Sensor networks are composed of large numbers of tiny sensing and computing devices. Each of these devices, called motes, has very limited communication, computational and energy resources. Often embedded in uncontrolled physical environments, these networks require distributed algorithms for efficient data processing, while individual motes require highly concurrent and reactive behavior for efficient operation. Sensor networks face many problems that do not arise in other types of networks [1]. Power constraints, limited hardware, decreased reliability, and a typically higher density and number of

nodes than those found in conventional networks are few of the problems that have to be considered when developing protocols for use in sensor networks.

Fig.1 shows a typical simple wireless sensor network. As can be seen, a complete wireless sensor network usually consists of one or more base stations (or gateway), a number of sensor nodes, and the end user. Sensor nodes are used to measure physical quantities such as temperature, position, humidity, pressure etc. The output of those sensor nodes are wirelessly transmitted to the base station (or gateway) for data collection, analysis, and logging. End users may also be able to receive and manage the data from the sensor via a website from long-distance or applications in console terminal [2]. However due to the associated cost, time and complexity involved in implementation of such networks, developers prefer to have first-hand information on feasibility and reflectivity crucial to the implementation of the system prior to the hardware implementation.

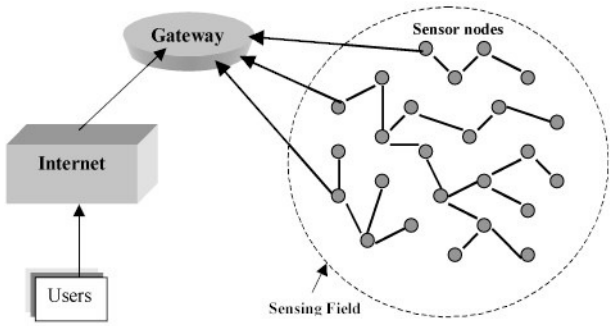

FIGURE 1: A simple wireless sensor network

This is especially true in sensor networks, where hardware may have to be purchased in large quantities and at high cost. Even with readily available sensor nodes, testing the network in the desired environment can be a time consuming and difficult task. Simulation-based testing can help to indicate whether or not the time and monetary investments are worthwhile. Simulation is, therefore, the most common approach to developing and testing new protocol for sensor networks. There are a number of advantages to this approach including lower cost, ease of implementation, and practicality of testing large-scale networks.

In order to effectively develop any protocol based on simulations, it is important to know the different tools available and their benefits and drawbacks. Given the facts that simulation is not perfect and that there are a number of popular sensor network simulators available, thus making different simulators accurate and most effective for different situations/applications. It is crucial for a developer to choose a simulator that best fits the application [3]. However, without a working knowledge of the available simulators, this is can be a challenging task. Additionally, knowing the weaknesses of available simulators could help developers to identify drawbacks of their own models, when compared with these simulators, thus providing an opportunity for improvement. It is thus imperative to have a detailed description of a number of the more prominent simulators available. In this paper, we have compared various sensor network simulators with emphasis on their ease of use, key features, limitations, availability, and environments best supported [4].

2. DESIGN OF SENSOR NETWORK SIMULATOR

The design of a Wireless Sensor Network (WSN) is a very application-specific task, especially because of the peculiarity of the considered deployment environment. Generic reliable predictive models for data correlation or radio propagation are seldom available. A thorough preliminary test phase is thus necessary, either by means of specifically crafted test beds, or via reliable simulations. WSN applications must be tested on a large scale, and under complex and varying conditions in order to capture a sufficiently wide range of interactions, both among nodes, and with the environment. A WSN simulator

consists of various modules namely events, medium, environment, node, transceiver, protocols, and applications. Each category is represented by an interface that defines its methods and events generated and consumed.

1. Event
Event is an abstract base class that provides basic functionality for all events. It contains the time at which an event should work, and provides methods to: compare events based on their fire times, determine whether events are equal, print themselves to a string, and an abstract method to fire the event.

2. Medium
Medium models the wireless medium. It allows nodes to broadcast signals, and is responsible for informing nodes of signals that affect it. In order to do this, Medium must be informed of the presence of every node, and any changes in position or radio properties such as transmitter power or receiver sensitivity. Medium has the properties of bandwidth and wavelength of the medium modeled and a reference to a propagation model that is given to it at the time of construction. The propagation model provides the strength at a particular receiver from a signal transmitted by a given transmitter.

3. Environment
The Environment module is similar to Medium module. The difference is that the implementation of Environment has properties that relate to the physical phenomenon modeled. Environment also has a propagation model that models the propagation of the physical phenomena modeled. Physical phenomena of interest in sensor networks include: temperature, light, humidity, magnetic field, sound, optical, chemical presence.

4. Node
It represents a single node in a wireless sensor network. As such, it serves as a container for all of the components, both hardware and software, in a node. These components should be included: processor, transceiver, sensors, actuators, energy source (such as a battery), network protocols, and applications. In addition each node has the properties of location and identification.

5. Transceiver
Transceiver models the hardware transceiver on each sensor node. It models the transceiver states (i.e. sleep, standby, receive, and transmit), and their associated behavior and power consumption. Transceiver consumes events informing it of the beginning and ending of every signal it receives. It sums active signals to maintain the interference. Transceiver generates events for the beginning and ending of every signal it transmits. These events are all exchanged with an instance of the Medium module.

6. Physical Protocol
The Physical protocol is the lowest layer in a network stack. It is often implemented in the transceiver hardware. The Physical layer provides services for: changing the state of the transceiver, carrier sensing, sending and receiving packets, received energy detection on received packets, changing channels on physical layers that support multiple channels.

7. MAC Protocol
The MAC protocol is the next layer in a network stack. It is usually implemented in software running on the node's processor. The MAC layer provides services for: changing the state of the MAC layer (i.e. low power mode), setting and getting protocol parameters, sending and receiving packets, etc. A WSN simulator usually offers implementations for several sensor network MAC protocols.

8. Routing Protocol
The Routing protocol resides above the MAC protocol and provides services for routing messages over multiple hops between nodes that cannot communicate directly.

9. Application Layer

The Application layer resides at the top of the network stack. It interfaces with the lower layers in the network stack as well as the sensors and actuators to implement a wireless sensor network application.

Most of the WSN simulators are based on the design described above. In addition to including the different modules, a WSN simulator should also have the following capabilities:

i. Reusability and availability
Simulation is used to test novel techniques in realistic and controlled scenarios. Researchers are usually interested in comparing the performance of a new technique against existing proposals [5].

ii. Performance and scalability
Performance and scalability is a major concern when facing WSN simulation. The former is usually bounded to the programming language effectiveness. The latter is constrained to the memory, processor and logs storage size requirements [6].

iii. Support for rich-semantics scripting languages to define experiments and process results
The vast amount of variables involved in the definition of a WSN experiment requires the use of specific input scripting languages, with high-level semantics. Additionally, it is likely that large quantities of output data will also be generated through many replicas of the experiments [7]. Therefore, a suitable output scripting language, which helps to obtain the results from the experiments quickly and precisely is desirable.

iv. Graphical, debug and trace support.
Graphical support for simulations is interesting in three aspects:
(a) As a debugging aid. The primary and more practical way to quickly detect a bad behavior is to “watch” and follow the execution of a simulation. The key features that a graphical interface should support are: Capability of inspection of modules, variables and event queues at real time, together with “step-by-step” and “run-until” execution possibilities. These features make graphical interfaces a very powerful debugging tool. Note that the key is the ability to interact with the simulation.
(b) As a visual modeling and composition tool. This feature usually facilitates and speeds the design of small experiments or the composition of basic modules. However, for large scale simulations, it is not very practical.
(c) Finally, it allows quick visualization of results without a post-processing application [8].

However, there are various challenges associated with the available WSN simulators. For instance, some simulator lack of available protocol models, which causes the increase of developing time, some simulators limit the scalability, etc. Additionally, modeling problems arise when considering the new environment and the energy components. They also compromise scalability and accuracy [9]. A deep study of these issues is mandatory for a better understanding and characterization of sensor networks and their corresponding simulators.

3. SIMULATOR COMPARISON

There are many different possible platforms for simulation and testing of routing protocols for WSNs. Below is a list of some of the most popularly used simulators for WSNs. Different aspects like energy efficiency, resources, decentralized collaboration, fault tolerance, simulation scenarios, global behavior etc. have been compared.

3.1 NS-2

1. Summary: NS-2 is a discrete event simulator targeted at networking research. NS-2 began as a variant of the REAL network simulator in 1989 and has evolved substantially over the past few years. NS-2 has a modular approach and hence is effectively extensible [10].
2. Environment: It provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks. Support for wireless networks was added later to simulate wireless LAN protocols, mobile ad-hoc networks and wireless sensor networks. The simulator focuses on following the ISO/OSI model.

3. Simulation language: Simulations are based on a combination of C++ and OTcl. [11].

4. Key features: NS-2's extensibility is perhaps what has made it so popular for sensor networks. It has an object-oriented design which allows for straightforward creation and use of new protocols. The key features for WSNs include sensor channels, battery models, lightweight protocol stacks, hybrid simulation support, and scenario generation tools. It provides a visualization tool called NAM (Network AniMator). Due to its popularity and ease of protocol development, there are a high number of different protocols that are publicly available. All the simulations are run at the packet level, allowing for detailed results.

5. Limitations: NS-2 has a long learning curve and requires advanced skills to perform meaningful and repeatable simulations. Another drawback to NS-2 is the lack of customization available. Packet formats, energy models, MAC protocols, and the sensing hardware models all differ from those found in most sensors. NS-2 also lacks an application model. In many network environments this is not a problem, but sensor networks often contain interactions between the application level and the network protocol level.

3.2 TOSSIM

6. Summary: TOSSIM is a discrete event simulator for TinyOS wireless sensor networks, which were developed at UC Berkeley. TOSSIM captures the behavior and interactions of networks not on the packet level but at network bit granularity. TOSSIM is designed specifically for TinyOS applications to be run on MICA Motes. TOSSIM simulates entire TinyOS applications [12]. It works by replacing components with simulation implementations. TOSSIM can replace a packet-level communication component for packet-level simulation, or replace a low-level radio chip component for a more precise simulation of the code execution [13] [14].

7. Environment: TinyOS is a sensor network operating system that runs on custom 'mote' hardware. TOSSIM provides mechanisms for TinyOS developers to choose the accuracy and complexity of the radio model necessary for their simulations [15].

8. Simulation language: It has a component-based programming model, provided by the nesC language, a dialect of C [16].

9. Key features: TOSSIM simulates the TinyOS network stack at the bit level, allowing experimentation with low-level protocols in addition to top-level application systems [17]. The simulation provides several mechanisms for interacting with the network, packet traffic can be monitored and packets can be statically or dynamically injected into the network. The transmission is simulated at the bit level.

10. Limitations: TOSSIM's run-instantly execution model does not capture CPU time. Since interrupts are discrete events, TOSSIM does not model preemption and the resulting possible TinyOS data races. Compilation steps lose the fine-grained timing and interrupt properties of the code, which can be important when the application runs on the hardware and interacts with other nodes [18]. TOSSIM is based on the assumption that each node in the network must run the exact same code, thus making it less flexible. TOSSIM does not model energy consumption, though there is an add-on PowerTOSSIMz that corrects this problem.

3.3 GLoMoSim

11. Summary: Global Mobile Information System Simulator (GloMoSim) is a scalable simulation environment for large wireless and wired communication networks. The node aggregation technique is introduced into GloMoSim to give significant benefits to the simulation performance. In GloMoSim, each node represents a geographical area of the simulation. Hence the network nodes which a particular entity represents are determined by the physical position of the nodes.

12. Environment: GloMoSim has several choices for radio propagation, CSMA MAC protocols (including 802.11), mobile wireless routing protocols, and implementations of UDP and TCP. GloMoSim is good at simulating mobile IP networks [19].

13. Simulation language: GloMoSim is a library for the C-based parallel discrete-event simulation language PARSEC (Parallel Simulation Environment for Complex Systems)

14. Key features: The ability to use GloMoSim in a parallel environment distinguishes it from most other sensor network simulators. Like Ns-2, GloMoSim is also designed to be extensible, with all protocols implemented as modules in the GloMoSim library [20]. GloMoSim can be executed using a variety of synchronization protocols and was successfully implemented on both shared memory and distributed memory computers.

15. Limitations: GloMoSim currently supports protocols for a purely wireless network. In the future, the developers anticipate adding functionality to simulate a wired as well as a hybrid network with both wired and wireless capabilities. GloMoSim is effectively limited to IP networks because of low level design assumptions. Therefore, it suffers the same problems as Ns-2, the packet formats, energy models, and MAC protocols are not representative of those used in wireless sensor networks.

3.4 UWSim

16. Summary: UWSim is a simulator used for Underwater Sensor Networks (UWSN). Most of the currently available simulators focus greatly on ground-based sensor and ad hoc networks they do not consider the factors that affect the underwater communication [21]. UWSim, on the other hand, has its focus on handling scenarios specific to UWSN environments, for example low bandwidth, low frequency, high transmission power, and limited memory. It is based on a network component-based approach, rather than a layer/protocol-based approach.

17. Environment: Tailor-designed for simulations of under-water sensor networks.

18. Simulation language: The software was developed on Windows XP using Microsoft. Net Framework 2.0 and is made to support all versions of Windows including 64-bit machines. The actual software development was carried out in a purely object-oriented fashion using C# capabilities [22].

19. Key features: Most simulators assume the characteristics of ground-based wireless ad hoc and sensor networks. These simulators use radio frequency transmission whereas UWSN needs a simulator which simulates the acoustic network. UWSim simulates the acoustic network [23]. It is based on a novel routing protocol proposed by the developers, unlike traditional simulators which are based on either proactive or reactive routing protocols such as AODV and DSR. The various characteristics of underwater networks such as low bandwidth, need for high frequency and the effect of salinity and temperature with depth are considered while simulating sensor networks with UWSim.

20. Limitations: Currently, UWSim has support for limited number of functionalities and it calls for further extensions to support a wide range of UWSN simulation exercises. Also another obvious disadvantage is that it cannot be used for simulating any other sensor network except UWSN.

3.5 Avrora

21. Summary: Avrora, a research project of the UCLA Compilers Group, is an open-source cycle-accurate simulator for embedded sensing programs. Unlike other simulators, that are able to simulate only specific platforms, Avrora has language and operating system independence. It provides a framework for program analysis, allowing static checking of embedded software and an infrastructure for future program analysis research. Avrora simulates a network of motes, runs the actual microcontroller programs (rather than models of the software), and runs accurate simulations of the devices and the radio communication [24].

22. Environment: It can emulate two typical platforms, Mica2 and MicaZ, and run AVR elf-binary or assembly codes for both platforms.

23. Simulation/Programming language: It is implemented in Java, which helps flexibility and portability.

24. Key features: One of the key features of Avrora is that it is an accurate and scalable simulator for the actual hardware platform on which sensor programs run. Avrora has a nearly complete implementation of the mica2 hardware platform, including a nearly complete ATMega128L implementation, and an implementation of the CC1000 AM radio [25]. Avrora is also capable of running a complete sensor network simulation with full timing accuracy, allowing programs to communicate via the radio using the software stack provided in TinyOS. It also has an extension point that allows users to create a new simulation type and choose the type of simulation to perform, depending on the number and orientation of the nodes. Developers claim that Avorora scales to networks of up to 10,000 nodes and performs as much as 20 times faster than most other simulators with equivalent accuracy [26].

25. Limitations: One major limitation of Avrora is that it does not model clock drift, a phenomenon where nodes may run at slightly different clock frequencies over time due to manufacturing tolerances, temperature, and battery performance. In recent literature, the developers have modeled distance-attenuation for multi-hop scenarios, but have not yet been able to model mobility. It is 50% slower than TOSSIM.

3.6 SENS: A Sensor Environment and Network Simulator

26. Summary: SENS is a wireless sensor network simulator with modular, layered architecture with customizable components which model an application, network communication, and the physical environment. It enables realistic simulations, by using values from real sensors to represent the behavior of component implementations. Such behavior includes sound and radio signal strength characteristics and power usage.

27. Environment: SENS is a customizable sensor network simulator for WSN applications. Multiple component implementations in SENS offer varying degrees of realism. Users can assemble application-specific environments, such environments are modeled in SENS by their different signal propagation characteristics. The same source code that is executed on simulated sensor nodes in SENS may also be deployed on actual sensor nodes, this enables application portability

28. Simulation/Programming language: It is written in C++. There exists a thin compatibility layer to enable direct portability between SENS and real sensor nodes.

29. Key features: SENS is platform-independent: as new WSN platforms are introduced, their parameter profiles can be added to the simulator. The ability to develop portable applications is an important feature, considering that WSN platforms constantly evolve as new sensor node implementations emerge. Another salient feature of SENS is its novel mechanism for modeling physical environments. WSN applications feature tight integration of computation, communication and interaction with the physical environment. To provide users with the flexibility of modeling the environment and its interaction with applications at different levels of detail, SENS defines an environment as a grid of interchangeable tiles.

30. Limitations: SENS is less customizable than many other simulators, providing no opportunity to change the MAC protocol, along with other low level network protocols. SENS does appear to use one of the most sophisticated environmental models and implements the use of sensors well. However, the only phenomenon detectable is sound. It can be argued as to whether or not it is better for a simulator to support several phenomena well, or to support one phenomenon extremely well.

3.7 COOJA (COntiki Os JAva)

31. Summary: COOJA is a simulator for the Contiki sensor node operating system. MSPSim can be integrated into COOJA, forming COOJA/MSPSim. It allows simultaneous cross-level simulation at application, operating system and machine code instruction set level [27]. COOJA combines low-level simulation of sensor node hardware and simulation of high-level behavior in a single simulation.

32. Environment: COOJA is flexible and extensible in that all levels of the system can be changed or replaced: sensor node platforms, operating system software, radio transceivers, and radio propagation models. As network communication is central to a WSN, the COOJA Simulator supports adding and using different radio mediums [28].

33. Simulation/Programming language: COOJA is a Java application, all interaction with compiled Contiki code is done through Java Native Interface (JNI).

34. Key features: COOJA is primarily a code level simulator for networks consisting of nodes running Contiki OS. Nodes with different simulated hardware and different on-board software may co-exist in the same simulation. Code level simulation is achieved by compiling Contiki core, user processes and special simulation glue drivers into object code native to the simulator platform, and then executing this object code from COOJA [29]. It is able to execute the Contiki programs in two different ways: either by compiling the program code directly on the host CPU, or compiling it for the MSP430 hardware. It can simulate sensor networks simultaneously at different levels, including the operating system level and the network (application) level.

35. Limitations: However, due to its extendibility, the simulator has relatively low efficiency. Simulating many nodes with several interfaces each requires a lot of calculations, especially when plugins are started and registered as observers to those interfaces. Supports a limited number of simultaneous node types, the simulator has to be restarted once and a while if the number of nodes exceed allowable limit. A test interface GUI is absent, thus making extensive and time-dependent simulations difficult.

3.8 Castalia

36. Summary: Castalia is an application-level simulator for Wireless Sensor Network based on OMNeT++. It can be used to evaluate different platform characteristics for specific applications, since it is highly parametric, and can simulate a wide range of platforms. In Castalia, sensor nodes are implemented as compound modules, consisting of sub-modules that represent, for instance, network stack layers, application, and sensor. Node modules are connected to wireless channel and physical process modules [30].

37. Environment: It is a generic simulator with realistic wireless channel and radio model based on measured data. Since it is based on the OMNeT++ platform, it can be used by researchers and developers who want to test their distributed algorithms and/or protocols in realistic wireless channel and radio models, with a realistic node behavior especially relating to access of the radio.

38. Simulation/Programming language: It is developed in C++ at the National ICT Australia [31].

39. Key features: Features of Castalia include: physical process modeling, sensing device bias and noise, node clock drift, and several MAC and routing protocols implemented. Castalia has a highly tunable Medium Access Control (MAC) protocol and a flexible parametric physical process model. Distinct physical process modules in Castalia represent different sensing devices (e.g. temperature, pressure, light and acceleration). Castalia can consider sensing device noise, bias and node clock drift [32].
40. Limitations: It should be noted that Castalia is not sensor-platform specific. Castalia is meant to provide a generic reliable and realistic framework for the first order validation of an algorithm before moving to implementation on a specific sensor platform. It is not useful if one would like to test code compiled for a specific sensor node platform.

3.9 Shawn

41. Summary: Shawn is a customizable sensor network simulator, it is open source and designed to support large-scale network simulation. Instead of simulating a phenomenon, Shawn is designed to simulate the effect of the phenomenon. It is claimed to provide the highest abstract level and support larger network comparing to other simulators such as ns-2, SENSE, OmNeT++, GloMoSim, and TOSSIM. However, details on simulating wireless sensor networks cannot be found [33].

42. Environment: The simulation environment is the home of the virtual world in which the simulation objects reside. The simulated nodes reside in a single World instance. The nodes themselves serve as a container for so-called processors.

43. Simulation/Programming language: It is written in Java.

44. Key features: Shawn features persistence and decoupling of the simulation environment by introducing the concept of Tags. They attach both persistent and volatile data to individual nodes and the world. They decouple state variables from member variables, thus allowing for an easy implementation of persistence [34]. Another benefit is that parts of a potentially complicated protocol can be replaced without modifying.

45. Limitations: Detailed simulations of issues such as radio propagation properties or low-layer issues are not well considered.

3.10 EmStar

46. Summary: EmStar provides a flexible environment for transitioning between simulation and deployment for iPAQ-class sensor nodes running Linux [35]. Users have three options: running many virtual nodes on a single host with a simulated network, running many virtual nodes on a single host with each virtual node bridged to a real-world one for networking, and running a single real node on a host with a network interface.

47. Environment: EmStar offers both a range of runtime environments, from pure simulation to actual deployment. EmStar was designed to be compatible with two different types of nodes. Like the other emulators, it can be used to develop software for Mica2 motes [36]. It also offers support for developing software for iPAQ based microservers.

48. Simulation/Programming language: Emstar is a Linux-based framework [37].

49. Key features: EmStar provides an option to interface with actual hardware while running simulation. The simulation model is component-based and uses a discrete event model. .It has a half simulation/ half-emulation approach similar to SensorSim's, where software is running on a host machine and interfacing with the actual sensor. This allows for use of the actual communication channel and sensors. EmStar code may be run on a diverse set of execution platforms, each run the same code and use the same configuration files, making it easy for developers to seamlessly iterate among all the modes. It brings together a number of the stronger features of other simulators and emulators. EmStar's use of the component-based model allows for fair scalability.

50. Limitations: EmStar uses a very simple environmental model and network medium. As the purpose is to migrate the code to a real sensor environment, simple environment and network medium abstractions sufficed for the developers. Secondly, the simulator will only run code for the types of nodes that it is designed to work with. Also, if a user is attempting to model a system by progressing through the development cycle, he must either ensure that the hardware configuration being used matches the configuration file, or he must bear in mind that there will be differences and compensate appropriately, if necessary [38]. It does not support parallel simulations and lacks algorithms that are reactive to real dynamics, seen in physically situated sensors.

3.11 JSim

- Summary: J-Sim is a general purpose simulator modeled after Ns-2, developed at the University of Washington by the National Simulation Resource. Unlike Ns-2, however, J-Sim uses the concept of components, replacing the notion that each node should be represented as an object. J-Sim uses three top level components: the target node (which produces stimuli), the sensor node (that reacts to the stimuli), and the sink node (the ultimate destination for stimuli reporting). Each component is broken into different parts and modeled differently within the simulator. The breakdown of each component makes it easy to use different protocols in different simulation runs.
- Environment: J-Sim provides support for sensors and physical phenomena. Energy modeling, with the exception of radio energy consumption, is also appropriately provided for sensor networks [39].
- Simulation/Programming language: J-Sim is similar to ns in that is written in two languages, however, in J-Sim's case they are Java and Jacl, a Java version of Tcl.
- Key features: J-Sim features several improvements on Ns-2 and other simulators. Most importantly, its component based architecture scales better than the object oriented model used by Ns-2 and other simulators. Furthermore, J-Sim features an improved energy model and the ability to simulate the use of sensors for phenomena detection. Like SensorSim, applications may be simulated, and there is support for the connection of real hardware sensors to the simulator [40].
- Limitations: J-Sim is however, relatively complicated to use. While no more complicated than Ns-2, the latter simulator is more popular and, thus, more people are willing to spend the time to learn how to use it. J-Sim, while more scalable than many other simulators, also faces its share of inefficiencies. Java, in general, is arguably less efficient than many other languages. There is also unnecessary overhead in the intercommunication model. The only MAC protocol that can be used is 802.11, a problem that seems to occur in most sensor simulators that are built on top of all-purpose simulators.

3.12 SENSE

- Summary: SENSE was a sensor network simulator developed in 2004.
- Environment: SENSE supports an energy model that is sufficient for wireless sensor networks [41].
- Simulation/Programming language: It is similar to J-Sim in that it is component based, but is written in C++ in order to avoid the perceived inefficiency of Java. SENSE runs on top of COST, a component based discrete event simulator that is written in CompC++, a component extension to C++.
- Key features: The most significant feature of SENSE is its balanced consideration of modeling methodology and simulation efficiency. SENSE is a user- friendly simulator that is also very fast. Unlike object-oriented network simulators, SENSE is based on a novel component-oriented simulation methodology that promotes extensibility and reusability to the maximum degree. At the same time, the simulation efficiency and the issue of scalability are not overlooked.
- Limitations: SENSE is still in its active development phase. Although the core of the simulator has been gradually stabilized, it still lacks a comprehensive set of models and a wide variety of configuration templates for wireless sensor networks. Besides, a visualization tool is desirable which can quickly track down what goes wrong during the simulation. Without such a tool, the output of the simulation is hard to interpret. Visualization can also facilitate the configuration phase by allowing networks to be constructed graphically [42].

3.13 VisualSense

- Summary: Modeling of wireless networks requires sophisticated representation and analysis of communication channels, sensors, ad-hoc networking protocols, localization strategies, media access control protocols, energy consumption in sensor nodes, etc. VisualSense is designed to support a component-based construction of such models.
- Environment: VisualSense provides an accurate and extensible radio model. The radio model is based on a general energy propagation model that can be reused for physical phenomena. VisualSense provides a sound model based on this propagation model that is accurate enough to use for localization.

- Simulation/Programming language: VisualSense is a modeling and simulation framework for wireless sensor networks that build on and leverages Ptolemy II. The extension to Ptolemy consists of a few new Java classes and some XML files [43]. The classes are designed to be sub-classed by model builders for customization, although non-trivial models can also be constructed without writing any Java code.
- Key features: It supports actor-oriented definition of network nodes, wireless communication channels, physical media such as acoustic channels, and wired subsystems. The software architecture consists of a set of base classes for defining channels and sensor nodes, a library of subclasses that provide certain specific channel models and node models, and an extensible visualization framework. Customized channels can be defined by subclassing the WirelessChannel base class and by attaching functionality defined in Ptolemy II models [44]. It is intended to enable the research community to share models of disjoint aspects of the sensor nets problem and to build models that include sophisticated elements from several aspects.
- Limitations: VisualSense, however, it does not provide any protocols above the wireless medium, or any sensor or physical phenomena other than sound.

3.14 (J)Prowler

- Summary: Prowler and (J)Prowler are probabilistic wireless sensor network simulators. Prowler is an event-driven simulator that can be set to operate in either deterministic mode (to produce replicable results while testing the application) or in probabilistic mode (to simulate the nondeterministic nature of the communication channel and the low-level communication protocol of the motes). It can incorporate arbitrary number of motes, on arbitrary (possibly dynamic) topology, and it was designed so that it can easily be embedded into optimization algorithms [45].
- Environment: (J)Prowler is targeted to the Berkeley MICA Mote hardware platform running application built on TinyOS, though it could be modified to simulate more general systems.
- Simulation/Programming language: Prowler is written in Matlab, while JProwler is written in Java.
- Key features: The simulator runs under MATLAB, thus it provides a fast and easy way to prototype applications, and has nice visualization capabilities. The network simulator models the important aspects of all levels of the communication channel and the application. The nondeterministic nature of the radio propagation is characterized by a probabilistic radio channel model. A simplified, but accurate model is used to describe the operation of the Medium Access Control (MAC) layer [46].
- Limitations: (J)Prowler provides an accurate radio model. However, it provides only one MAC protocol, the default MAC protocol of TinyOS

4. ANALYSIS

Programming sensor network systems is and will likely remain a challenging task. A primary goal of simulation platforms is to help alleviate these burdens. For wireless sensor network systems, two features of simulators are extremely valuable: reproducible experimentation and dynamic environment modeling. In addition to providing a fairly comprehensive list of simulators available, this paper also addresses most of the issues facing the developers of sensor networks. At the topmost level, developers must decide whether they want a simulator or an emulator. Each has advantages and disadvantages, and each is appropriate in different situations. The key features and limitations of each of these simulators/emulators are highlighted in Table I below.

Generally, a simulator is more useful when looking at things from a high level. The effect of routing protocols, topology, and data aggregation can be seen best at a top level and would be more appropriate for simulation. Emulation is more useful for fine-tuning and looking at low-level results. Emulators are effective for timing interactions between nodes and for fine tuning network level and sensor algorithms. If the developers decide to build a simulator, another design level decision that must be made is whether to build their simulator on top of an existing general simulator or to create their own model. If development

No.	Simulator	Programming language/ Platform	Key features	Limitations
1	Ns-2	C++	Easy to add new protocols. A large number of protocols available publicly. Availability of a visualization tool.	Supports only two wireless MAC protocols, 802.11, and a single-hop TDMA protocol.
2	TOSSIM	nesC	High degree of accuracy or running the application source code unchanged. Availability of a visualization tool.	Compilation steps lose the fine-grained timing and interrupt properties of the code,
3	GloMoSim	Parsec	Parallel simulation capability. It is tailored specifically for wireless networks. Availability of a visualization tool.	Effectively limited to IP networks because of low level design assumptions. Unavailability of new protocols.
4	UWSim	C++	Publicly available and designed solely for UWSN.	Supports only a limited number of functionalities and calls for extension.
5	Avrora	Java	Can handle networks having up to 10,000 nodes. Enables validation of time-dependent properties of large-scale networks	Fails to model clock drift. 50% slower than TOSSIM. Cannot model mobility.
6	SENS	C++	Platform-independent Users can assemble application-specific environments Defines an environment as a grid of interchangeable tiles.	Not accurately simulate a MAC protocol. Provides support for sensors, actuators, and physical phenomena only for sound.
7	COOJA	Java (Simulations in C)	Concerning both simulated hardware and software. Larger-scale behavior protocols and algorithms can be observed.	Not extremely efficient. Supports a limited number of simultaneous node types. Making extensive and time-dependent simulations difficult..
8	Castalia	C++	Physical process modeling, sensing device bias and noise, node clock drift, and several MAC and routing protocols implemented. Highly tunable MAC protocol and a flexible parametric physical process model.	Not a sensor specific platform. Not useful if one would like to test code compiled for a specific sensor node platform.
9	Shawn	Java	Not limited to the implementation of distributed protocols Can simulate vast networks	Detailed simulations of issues such as radio propagation properties or low-layer issues are not well considered.
10	EmStar	Linux	May be run on a diverse set of execution platforms. Combination of simulator and emulator. EmStar's use of the component-based model allows for fair scalability	Only run code for the types of nodes Does not support parallel simulations. Not as efficient and fast as other frameworks
11	J-Sim	Java	Provides support for energy modeling, with the exception of radio energy consumption Support mobile wireless networks and sensor networks. Component-oriented architecture.	Low efficiency of simulation. The only MAC protocol provided for wireless networks is 802.11. Unnecessary run-time overhead
12	SENSE	C++	Balanced consideration of modeling methodology and simulation efficiency. Memory-efficient, fast, extensible, and reusable.	Not accurate evaluation of WSN research. Lacks a comprehensive set of models Absence of a visualization tool
13	VisualSense	Ptolemy II	Provides an accurate and extensible radio model as well as a sound model that is accurate enough to use for localization.	Does not provide any protocols above the wireless medium, or any sensor or physical phenomena other than sound.
14	(J)Prowler	Matlab/Java	Probabilistic wireless sensor network simulators. (J)Prowler provides an accurate radio model.	Provides only one MAC protocol, the default MAC protocol of TinyOS.

TABLE 1: Key features and limitations of some popular WSN simulators

time is limited or if there is one very specific feature that the developers would like to use that is not available, then it may be best to build on top of an existing simulator. However, if there is available

development time and the developers feel that they have a design that would be more effective in terms of scalability, execution speed, features, or another idea, then building a simulator from the base to the top would be most effective. In building a simulator from the bottom up, many choices need to be made. Developers must consider the pros and cons of different programming languages, the means in which simulation is driven (event vs. time based), component-based or object-oriented architecture, the level of complexity of the simulator, features to include and not include, use of parallel execution, ability to interact with real nodes, and other design choices. While design language choices are outside of the scope of this paper, there are some guidelines that stand out upon looking at a number of already existing simulators. Most simulators use a discrete event engine for efficiency. Component-based architectures scale significantly better than object-oriented architectures, but may be more difficult to implement in a modularized way. Defining each sensor as its own object ensures independence amongst the nodes. The ease of swapping in new algorithms for different protocols also appears to be easier in object-oriented designs. However, with careful programming, component based architectures perform better and are more effective. Generally, the level of complexity built into the simulator has a lot to do with the goals of the developers and the time constraints imposed. Using a simple MAC protocol may suffice in most instances, and only providing one saves significant amounts of time. Other design choices are dependent on intended situation, programmer ability, and available design time. The use of a standard configuration file may be a limiting factor in many situations.

Figure 2 presents a comparative graph of the most prominent simulation frameworks according to the criteria of scalability and abstraction level. In this figure, it does not express the maximal feasible network sizes, but reflects the typical application domain.

Regarding availability of models, OMNET++, GloMoSim lack of available protocol models compared to other simulators (specially, NS-2), which increases development time. Attending to the ability to compose models from basic pieces, the component or actor based packages J-Sim offer the maximum flexibility. Tools like Shawn or Avrora allow any, Linux or Java respectively, application to be used in a simulation. This feature greatly increases their possibilities. Specific tools such as TOSSIM are able to simulate real sensor code.

Focusing on the performance, one can expect better performance from C/C++ engines than from their Java counterparts. Obviously, parallel simulations should perform and scale better than sequential ones. For instance, parallel simulators as GloMoSim (whose goal is performance rather than scalability) can simulate up to around 10,000 wireless nodes.

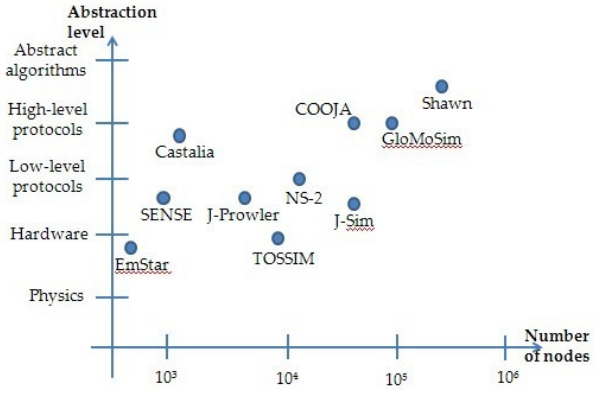

FIGURE 2: Comparison of simulators based on scalability and level of abstraction

Almost all the packages provide graphical support, which makes user can really easily to understand the whole simulation situation. OMNET++ and J-Sim provide powerful GUI libraries for animation, tracing and debugging. Current support in NS-2 is the unelaborated and simple trace reproduction Nam tool. Specific tools also provide surprisingly rich GUIs. TinyViz is the TOSSIM visualization tool, an extensible Java application that provides useful debug information. Besides, it can control and drive the simulation elements. Users can develop their own plugins, which listen for TOSSIM events published by TinyViz and perform some action. However, the latest TOSSIM version 2.x currently does not support TinyViz because the exact radio metadata interfaces have not been determined yet and the code hasn't been written, so if the graphical support is needed, we can use the TOSSIM 1.x instead.

5. CASE STUDY

The aim of this study is to demonstrate a performance comparison for one specific application. It should be however noted that each simulator has a varied range of performance depending on the application and thus the choice of simulator is greatly based on the application. Here, we present a case study to compare the performance of NS2, TOSSIM and Shawn because they are the most popular and very widely used in the WSN area. In this case, a simple application is simulated to broadcast a message every 250ms, the communication range of the sensor nodes is set to 50 length units and each simulation runs for 60 simulated time units. The size of the simulated area is 500x500 length units. A number of simulations with increasing node count were performed. Therefore, the network's density increases steadily as more nodes are added to the scenario. This application has been implemented on TOSIM, NS-2, and Shawn.

Since the exchange of wireless messages is the key ingredient in wireless sensor network, we have presented a measurement to show the required CPU time and memory for each simulator. Figure 3 and Figure 4 shows the time and memory consumption in the three simulators.

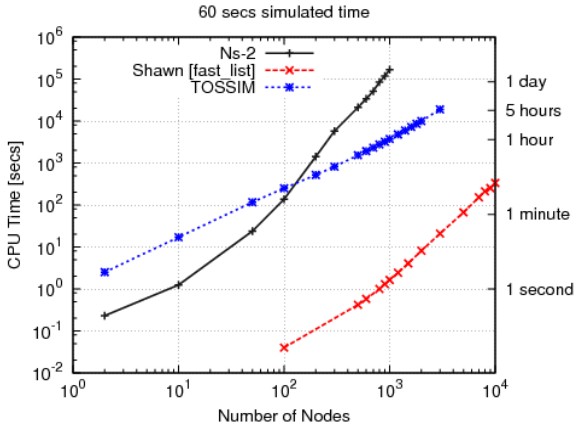

FIGURE 3: Computational time for simulations in NS-2, TOSSIM and Shawn

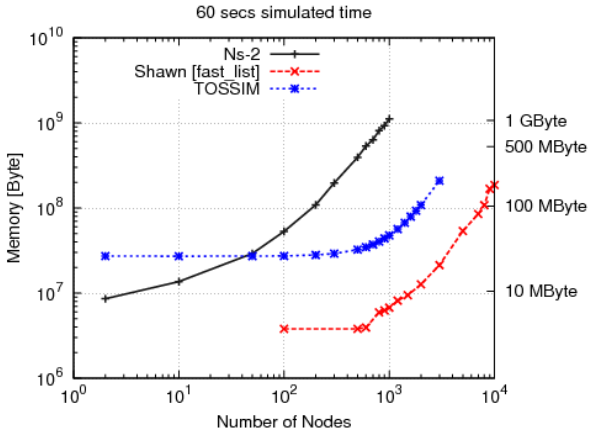

FIGURE 4: Memory consumption for NS-2, TOSSIM and Shawn

Figure 3 shows a comparison of the computation time for different number of nodes in each simulator. From the figure, it can be seen that Shawn can simulate a network with about 10,000 nodes in less than an hour. The performance of NS-2 is good for 100 nodes, which decreases significantly as the number of nodes increase. TOSSIM follows a linear curve. Figure 4 shows the memory consumption for the three simulators. Shawn has the least memory consumption of the three simulators. This clearly shows that Shawn excels in its specialty: the simulation of large-scale sensor networks with a focus on abstract, algorithmic considerations and high-level protocol development.

6. CONCLUSION

The goals of this paper have been to provide comprehensive survey and background on a number of different sensor network simulators and present the pros and cons of each simulator. Knowledge of the strengths and weaknesses of a number of different simulators is valuable because it allows users to select the one most appropriate for their testing. Simulators are compared based on different criteria, and comparative results are presented in tabular form. In addition, short descriptions of simulators are also provided. Since no single simulator under survey is universally applicable to all situations, appropriate guidelines for choosing the best simulator for a particular application environment are also provided.

7. REFERENCES

1. M. Ilyas and I. Mahgoub, Handbook of sensor networks: compact wireless and wired sensing systems, BocaRaton, FL., CRC Press, 2004.

2. J. Liu, et. al., “Simulation modeling of large-scale ad-hoc sensor networks,” European Simulation Interoperability Workshop 2001, London, England, June 2001,

3. I.F. Akyildiz and W. Su and Y. Sankarasubramaniam and E. Cayirci, “A Survey on Sensor Networks,” IEEE Communication Magazine, vol. 40, no. 8, pp. 102-116, Aug. 2002.

4. David Curren, “A survey of simulation in sensor networks,” University of Binghamton, NY, 2005.

5. John Heidemann, Kevin Mills, Sri Kumar, Expanding Confidence in Network Simulations.

6. E. Egea-López, J. Vales-Alonso, A. S. Martínez-Sala, P. Pavón-Mariño, J. García-Haro, Simulation Tools for Wireless Sensor Networks

7. Mekni, M. Moulin, A Survey on Sensor Webs Simulation Tools.

8. WeiChung,Hu MingLun, Lee TzungShian, Tsai Hewijin, Christine Jiau, A GUI Simulation Model in Supporting Embedded Software Design
9. David M. Nicol, Scalability of Network Simulators Revisited
10. Ns-2 [Online]. Available: http://www.isi.edu/nsnam/ns/. Retrieved: 02/04/2010.
11. T. Issariyakul and E. Hossain, Introduction to network simulator ns2, Springer, Nov. 2008.
12. P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and scalable simulation of entire tinyos applications," 1st ACM Conference on Embedded Networked Sensor Systems, Los Angeles, CA, Nov. 2003
13. P. Levis and N. Lee, Simulating tinyos networks [Online]. Available: http://www.cs.berkeley.edu/pal/research/tossim.html. Retrieved: 02/04/2010.
14. L. F. Perrone and D. Nicol, "A scalable simulator for TinyOS applications," Proceedings of the Winter Simulation Conference, vol. 1, no. 8, pp. 679-687, Dec. 2002.
15. TinyOS, an open-source operating system for wireless embedded sensor networks [Online]. Available: http://www.tinyos.net/. Retrieved: 02/04/2010.
16. NesC, the sensor network programming language on TinyOS operating system [Online]. Available: http://nescc.sourceforge.net/. Retrieved: 02/04/2010.
17. TOSSIM [Online]. Available: http://docs.tinyos.net/index.php/TOSSIM. Retrieved: 02/04/2010.
18. P. Levis, TOSSIM: Simulating TinyOS Networks [Online]. Available: http://www.eecs.berkeley.edu/~pal/research/tossim.html. Retrieved: 02/04/2010.
19. X. Zeng, R. Bagrodia, and M. Gerla, "GloMoSim: A library for parallel simulation of large-scale wireless networks," SIGSIM Simulation Digest, vol. 28, no. 1, pp. 154-161, 1998.
20. GloMoSim [Online]. Available: http://pcl.cs.ucla.edu/projects/glomosim/. Retrieved: 02/04/2010.
21. S. Dhurandher, S. Misra, M. Obaidat, and S. Khairwal, "UWSim: A simulator for underwater sensor networks," Simulation, vol. 84, no. 7, pp. 327-338, 2008.
22. J. Cui., J. Kong, M. Gerla and S. Zhou, "Challenges: Building scalable mobile underwater wireless sensor networks for aquatic applications," UCONN CSE Technical Report: UbiNET-TR05-02, University of Connecticut, USA, 2005.
23. R. Jurdak, C. V. Lopes and P. Baldi, "Battery lifetime estimation and optimization for underwater sensor networks," Sensor Network Operations, May 2006. pp. 397–420., Wiley IEEE.
24. Ben L. Titzer, Daniel K. Lee, Jens Palsberg, "Avrora: Scalable sensor network simulation with precise timing," 4th Int. Conf. on Information Processing in Sensor Networks, 2005.
25. Avrora [Online]. Available: http://compilers.cs.ucla.edu/avrora. Retrieved: 02/04/2010.

A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - A lightweight and flexible operating system for tiny networked sensors," Proceedings of the 29th Annual IEEE international Conference on Local Computer Networks, Tampa, FL., Nov.2004, pp. 455-462.

26. F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level sensor network simulation with COOJA," 1st IEEE International Workshop on Practical Issues in Building Sensor Network Applications, pp. 8, Tampa, Florida, USA, 2006.
27. F. Osterlind, "A sensor network simulator for the Contiki OS," Swedish Institute of Computer Science (SICS), Tech. Rep. T2006-05, Feb. 2006.
28. P. J. Marrón, et. al., "COOJA/MSPSim: Interoperability testing for wireless sensor networks," 2nd Int. Conf. on Simulation Tools and Techniques, page 7, Rome, Italy, Mar. 2009.
29. S. Park, A. Savvides, and M. Srivastava, "SensorSim: A simulation framework for sensor networks," 3rd ACM Int. Workshop on Modeling, Analysis & Simulation of Wireless and Mobile Systems, Boston, MA, Aug. 2000.
30. S. Park, A. Savvides, and M. B. Srivastava, "Simulating networks of wireless sensors," Winter Simulation Conference, Arlington, Virginia, Dec. 2001.

A. Boulis, "Castalia, a simulator for wireless sensor networks and body area networks," version 2.0, User's manual, May 2009 [Online]. Available: http://castalia.npc.nicta.com.au/. Retrieved: 02/04/2010.

B. Boulis, "Castalia: Revealing pitfalls in designing distributed algorithms in WSN," 5th Int. Conf. on Embedded Networked Sensor Systems, Sydney, Australia, Nov. 2007.

31. Alexander Kr¨oller, Dennis Pfisterer, S´andor P. Fekete, and Stefan Fischer, Algorithms and Simulation Methods for Topology-Aware Sensor Networks.

A. Kr¨oller, D. Pfisterer, C. Buschmann, S. P. Fekete, S. Fischer, Shawn: A new approach to simulating wireless sensor networks.
32. L. Girod, et al., “EmStar: An environment for developing wireless embedded systems software,” USENIX Technical Conference, Boston, MA, June 2004.
33. L. Girod, et. al., “EmStar: An environment for developing wireless embedded systems software,” Technical report, Center for Embedded Networked Sensing, University of California, Los Angeles, CENS Technical Report 009, 2003.
34. L. Girod, et. al., “Emstar: A software environment for developing and deploying heterogeneous sensor-actuator networks,” ACM Transactions on Sensor Networks, vol. 3, no. 3, article 13, Aug. 2007.
35. Familiar Linux [Online]. Available: http://www.handhelds.org. Retrieved: 02/04/2010.
    A. Sobeih and J. C. Hou, “A simulation framework for sensor networks in J-Sim,” Tech. Report UIUCDCS-R-2003- 2386, Dept. of Computer Science, University of Illinois at Urbana-Champaign, November 2003.
36. J-sim [Online]. Available: http://nsr.bioeng.washington.edu/jsim/. Retrieved: 02/04/2010.
37. G. Chen, J. Branch, M. Pflug, L. Zhu, and B. Szymanski, “SENSE: A sensor network simulator,” Dept. of Computer Science, Rensselaer Polytechnic Institute, 2004.
38. SENSE [Online]. Available: http://www.ita.cs.rpi.edu/sense/index.html. Retrieved: 02/04/2010.
39. P. Baldwin, S. Kohli, E. Lee, S. Liu, and Y. Zhao, “VisualSense: Visual modeling for wireless and sensor network systems,” Technical Memorandum UCB/ERL M05/25, University of California, Berkeley, CA, April 2004.
40. VisualSense [Online]. Available: http://ptolemy.eecs.berkeley.edu/visualsense/. Retrieved: 02/04/2010.
41. G. Simon, P. Volgyesi, M. Maroti, and A. Ledeczi “Simulation-based optimization of communication protocols for large-scale wireless sensor networks,” IEEE Aerospace Conference, Big Sky, MT, Mar. 2003. Prowler [Online]. Available: http://www.isis.vanderbilt.edu/projects/nest/prowler/index.html. Retrieved: 02/04/2010.
42. JProwler [Online]. Available: http://www.isis.vanderbilt.edu/projects/nest/jprowler/index.html. Retrieved: 02/04/2010.
43. http://www.mathworks.com/products/matlab/