INES: Network Simulations on Virtual Environments

Ricardo Lent
Intelligent Systems and Networks
Imperial College, London, UK
r.lent@imperial.ac.uk

ABSTRACT

The paper discusses the goals, design, models and implementation state of a new discrete-event simulator of computer networks. The Integrated Network-Environment Simulator (INES) aims at creating a suitable simulation tool for the evaluation of network algorithms and protocols operating on a virtual environment. This feature makes the simulator particularly useful for testing MANETs, VANETs, WSNs, situated communications, and other algorithms or technologies with realistic situations where the environment could significantly affect their operation. INES uses mesh geometries to model the objects of the desired environment, which may restrict node roaming and affect wireless communications during a simulation execution. Complementing INES development is the implementation of the *Packet Animator*, which can visualise INES traces through an OpenGL-based animation engine. Finally, the paper discusses examples that make use of the mobility and communication models currently supported by the simulator.

General Terms

I.6.7 Simulation Support Systems

Keywords

Network simulation, virtual environments

1. INTRODUCTION

While research on mobile networks, wireless sensor networks and situated communications gain momentum, it turns out indispensable to have proper tools to evaluate algorithms and protocols with consideration of their environment or geographical context. The most natural representation of a scenario for these simulations is with the use of tri-dimensional models recreating a virtual world where nodes may exist and interact. This model of the environment would provide the proper representation of the physical structures that may affect node mobility and wireless communications. Moreover, this model would allow to create a continuous data-flux of environmental information for the algorithms requiring it.

Network simulation is a very active field that currently offers several successful examples of simulators, which differ in their architecture and scope. Representative examples are NS-2 [9], OMNeT++ [5], GTNeTS [25], SSF [11], Glomosim [8] and QualNet [7], YANS [15], CNET [23] and OPNET [6]. However, the problem of situating network simulations within a specific context has receive little attention, with very little or no support provided by these simulators.

Simulations without a proper representation of the operating environment could lead to erroneous conclusions. For example, most studies on MANETs rely on simple mobility models that unrealistically assume no obstructions creating great uncertainty about their expected behaviour once deployed on a real environment. Moreover, while mobiles indeed tend to move on horizontal planes with few (or temporal) exceptions, communications are usually not restricted to the plane. For instance, wireless communications can be established (or create interference) between floors of a building and intelligent sensors can be purposely placed at specific places, likely at different levels, to optimize sensing. Most previous studies on mobility with obstacles in the literature are restricted to the plane [14, 26, 22] limiting their applicability and realism.

INES (Integrated Network-Environment Simulator) is a software project aimed at creating a new discrete-event network simulator that will offer the possibility of conducting realistic evaluations of network algorithms and protocols in virtual environments. The general design goals of INES are:

1. Create a general network simulator simple to use and with an efficient object model to simplify the development of extensions.

2. Offer the possibility of conducting network simulations within a virtual environment, such as a urban setting with buildings and cars, or indoor settings with walls and furniture.

3. Support emulation, both by accepting external input and by producing input to external processes. The typical emulation process would include the possibility of accepting real packets, processing them in the simulated network and potentially returning them to the real network. In addition, it is planned to include support for the generation of virtual topologies for emulated mobile networks superseding MTM [16].

SIMUTools 2008, March 3-7, 2008, Marseille, France

4. Finally, enable distributed execution for scalability and improved simulation speed.

The rest of the paper will discuss the models currently supported by the simulator. Later on, a few examples will be presented.

2. SOFTWARE ARCHITECTURE

The core of INES is a discrete-event scheduler, which is configurable to advance the simulated time either to match the time of the next event in the execution list or the wall clock (in emulation mode). As with other simulators, the simulations can be defined using scripting. Simulation execution is separated from visualisation. INES is being developed in ANSI/ISO C++ and relies on the standard template library for most internal structures. In addition, INES makes extensive use of various Boost libraries [1]. Scripting is supported via an embedded Python interpreter, which is provided by Boost Python.

2.1 Events

Events are handled in a similar way to other network simulators. An event (`class Event`) is the most basic data structure in the simulator and allows to represent an action to be executed. An event scheduler (`class Scheduler`) handles the execution timing of events within the simulator and permits inserting new events in logarithmic time and retrieving the next event in constant time. When the scheduler selects an event for execution, it invokes a function of the associated handler for the event as indicated in the handler itself (derived from `class Handler`). Moreover, events can be cancelled either individually or by group (events within a time period) in linear time. The implementation relies on `std::multiset` with an specialized `std::less` functor to handle the ordering of event objects on insertion. Timers on the other hand (`class Timer` and `Timer_handle`), simplify the creation of events by allowing developers indicate a callback function to the scheduler engine.

2.2 Nodes and Protocols

Nodes support an arbitrary number of protocols and interfaces connecting to wired or wireless media. A protocol component (class `Protcomp`) is the base class from which all network protocols are derived. A `Protcomp` is an event handler that can send messages to other protocol components. Protcomps implement the concept of layering and can have associated a certain overhead for the handle of messages (packet header and trailer).

Specific protocol implementations may implement buffers and related functionality. Lower layer protocols offer in addition callback functionality that is triggered by packet processing errors, for example, when dropping a packet because of a wireless collision or after excessive retransmission attempts.

The current version of INES supports both static routing and dynamic routing via Cognitive Packet Networks (CPN) [13, 12]. The static routes are relevant only to the INET protocol and they are computed at the beginning of the simulation with Dijkstra's algorithm.

2.3 Links

Links are also handlers, which depending on their nature may connect to the MAC protocol of two or more nodes. Each link has associated a set of physical characteristics: transmission rate, preamble time, packet error rate and propagation delay for wired links. For wireless links, they also indicate the propagation speed and radio characteristics (maximum transmission power, modulation type, antenna gain, etc.) Links use the decorator design pattern to add the extra information and functionally required by the nodes for the specific operation of the link.

2.4 Tracing and Support for Computer Clusters

INES offers support for two types of traces, which can be enabled or disabled as desired. The first is suitable for calculating statistical results of simulations when running the program either in a single machine or in a computer cluster. The second type generates a trace file suitable for the visualiser, which can also be enabled if so desired.

The current support for computing cluster execution is limited to replicating complete simulation jobs on many computers. For this, a custom scheduler distributes jobs and keeps processors busy until the end of the batch. The scheduler was written in C++ with the message passing interface (LAM/MPI) [4].

2.5 User Interface

The main user interface to INES is the embedded Boost Python interpreter wrapping (and extending) C++ objects. The interface allows specifying simulation parameters, creating and connecting nodes, and establishing traffic flows and events. A direct mapping between Python and C++ objects allows the latter to offer decorator functionality by introducing additional support, such as combining primitive calls to form higher order functions. For example, mesh, bus and ring networks can be easily instantiated with a single command. Likewise, networks with random graph, small world or scale free properties can be quickly instantiated by making use of the Boost Graph library. Topologies can be imported from ascii files and exported in .dot format (for Graphviz [3]) and postscript.

3. ENVIRONMENT MODELLING

INES represents the environment (scenario) with 3D geometry consisting of objects portrayed as a collection of faces (convex polygons). Each face has associated a material type that defines its physical characteristics and a normal vector that is used for collision detection. A material defines whether an object face may block a node's motion and the attenuation that radio signals should suffer when propagating through that object face. Obstacles can be defined in one or more standard Wavefront object files (.obj), which can refer to multiple .mtl files for material information. The standard .mtl file format was extended to allow the inclusion of mobility and radio propagation information that is required by the simulator. Specifically, three definitions have been introduced to .mtl files. `#Ines_move_block` and `#Ines_prop_block` take values of either 0 or 1 to enable or disable motion and radio propagation collisions with the material. `#Ines_prop_loss` defines signal attenuation for materials that affect radio propagation. The '#' symbol indicate a comment in the standard .mtl format and its use prevent misinterpretations of the added definitions outside the INES domain.

Object files and material files are convenient to use because they are in ASCII and therefore easy to manipulate. Moreover, most 3D modeler programs can import and export the .obj mesh file format, which is also convenient. There is a large number of products able to convert other mesh file formats (e.g. 3ds, dxf, lwo, vrml) to object files. Satellite imagery and 3D buildings from Google Earth [2] can be imported into INES via object files, which can make simulations particularly accurate in terms of locations, geometry and object dimensions. A scenario that was constructed from various buildings borrowed from Google Earth Warehouse is shown in Figure 5. The scenario is used in one of the examples described later on with a IEEE 802.11 network. In the future, the simulator will support additional terrain features, such as an uneven ground.

Material files often include references to the visual appearance of the objects, such as illumination parameters and textures. Visual properties are only relevant to the visualisation (Packet Animator) to represent objects on the screen as intended.

Optionally, objects can be associated to an alternative mesh. Usually, a low complexity model is used for simulation executions to keep running times short, while high-poly count models can be used to achieve a better visualisation and taking advantage of an OpenGL hardware acceleration whenever available.

3.1 Mobility

In the simulator, all nodes have associated a location. Mobile nodes have at least one wireless interface and their location in the 3D space is configurable (their location can be either defined by the user or controlled by a mobility model or determined by a mobility trace). This property can be overridden as needed to better model the certain nodes, for example, a workstation equipped with a wireless card should not be allowed to move under normal circumstances. Nodes without a wireless interface are stationary by default. The location of these nodes does not have a direct meaning in the simulation execution. However, the Packet Animator may assign an arbitrary positions to stationary nodes for visualisation purposes. Moreover, the animator can be enabled to manipulate these nodes and establish a suitable network layout with the Fruchterman-Reingold forced directed algorithm (from the Boost Graph library).

To support mobility, each node has the following properties: location, velocity vector and time to stop. The simulator gives facilities for scheduling, from a simulation script, destinations for the nodes and for indicating the use of mobility models. From the simulation script a node object has a `moveto` member function that accepts as arguments the desired destination in the 3D space and the moving speed. To move from one place to another in the scenario, nodes use path planning. Currently, there is support for two types of path planning in INES:

1. *Straight line*, which makes nodes basically follow the direct path connecting their current position and the destination. The simulator allows to enable or disable obstacle collision checking. If enabled, a node-environment collision stops the node at the obstacle.

2. *Obstacle avoiding*, which allows nodes to evade obstructions if they block their direct path to the destination. Obstacles can be avoided by providing nodes with a set of waypoints to follow, which are calculated by the obstacle avoiding module. In spite of nodes may move in any direction in the 3D environment, the current support is limited to horizontal planes, with their level (Z coordinate) defined by the current location of nodes. On the other hand, there is support for having nodes located on different levels (e.g. on multiple floors of a building).

   To identify suitable waypoints, INES constructs 2D maps of the scenario (*floor plans*) by intersecting the 3D meshes modelling the environment with one or more horizontal planes. For each floor, those intersections produce a set of polygons that define the pathways, or more precisely, a graph of waypoints with vertices corresponding to waypoints and edges to paths. Also, new edges are added whenever there exist direct visibility between any two vertices. Polygons are grown by a small factor (e.g. 5%) to create a reasonable distance from obstacles for the nodes to move. The edges of the graph represent both the possibility of going from one waypoint to another and the cost (Euclidean distance) associated. As a result, nodes tend to round obstacles while trying to avoid them. The situation results similar to people walking on the curbs of a street. Figure 1 depicts a few obstacles and the calculated waypoints and pathways.

   Floor plans and the shortest paths between waypoints are calculated at the beginning of the simulation with the A* algorithm (implemented by Dijkstra's algorithm). At any time later, paths can be easily calculated from the pre-calculated table of waypoints, in addition to the distances from the origin and destination to these waypoints, so that paths always follow the minimum distance. This approach produces a rich set of waypoints for nodes, which are not limited to moving on pathways.

   A previous work addressed the used of Voronoi graphs to define pathways for nodes to avoid obstacles. This approach produces paths that tend to go along the middle distance between pair of obstacles [14]. The middle distance seems appropriate to most campus-based scenarios, but maybe not suitable for modelling people on urban environments where there is a tendency for walking next to buildings. The model described in the paper is more suitable for the second situation.

In addition to path planning, INES supports instantiations of *movers*, which are objects that accept node subscriptions and decide their new destinations on the space every time a node stops. Movers simplify the process of defining mobility models for simulations, making it possible, for example to define sets of nodes under different mobility models. The existing support in INES is for the random-way point and Gauss-Markov mobility models [10].

4. WIRELESS LAN SIMULATIONS

4.1 Radio Propagation Model

In a simulated wireless scenario, the propagation model aims at predicting the path loss of a link, so that packet

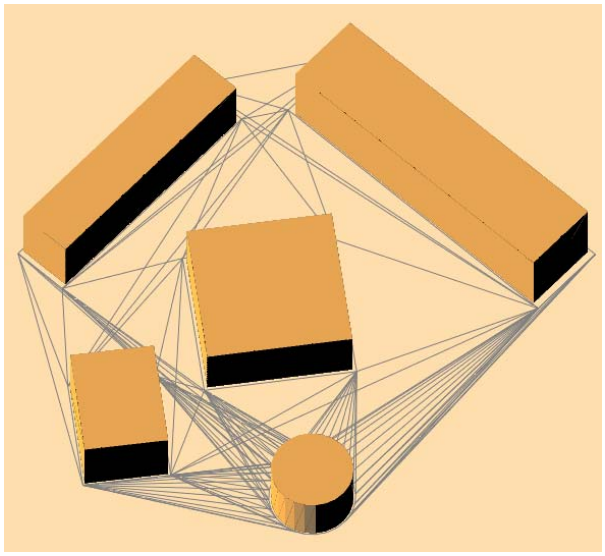

Figure 1: Obstacles and pathways. The lines represent the pathways that have been calculated by the simulator to allow nodes avoid the obstacles.

transmissions can be delivered to the likely receivers. Wireless network protocols typically use microwave signals in the ultra high frequency (UHF) (0.3–3 GHz) or in the extremely high frequency (EHF) (30–300 GHz) radio range. For example, the IEEE 802.11 (WiFi), IEEE 802.15.4 (sensor motes) and Bluetooth use the 2.4 GHz band (also known as the Industrial, Scientific and Medical band). WiMAX (IEEE 802.16) operate in 2.5, 3.5 and 5.8 GHz.

UHF and SHF radio signals propagate mainly through the direct path, but the presence of obstacles in the environment may cause additional propagation mechanisms (multipath). Depending on the materials present in the obstacle composition, reflections may occur with large objects compared to the wavelength of the signal. Diffraction may occur with sharp edges of objects blocking or near the direct path. If small objects are present, such as traffic lights, the signal may suffer in addition of scattering. These propagation mechanisms determine the path loss and signal distortions and fading that may vary significantly the signal strength with small movements of the transmitter and receiver.

A basic model to predict the direct path loss is given by the Friis formula (in dB) for free space [24]:

$$P_L = 32.4 + 20log(F) + 20log(D) \quad (1)$$

where F is the carrier frequency in MHz and D the distance in kilometers, and assuming unit-gain antennas. However, path losses are usually worse than predicted by the free-space model. A popular model that was constructed from indoor measurements predicts:

$$P_L = 40 + 35log(d)$$

where d is the separating distance in meters.

The availability of the environment geometry allows far more accurate predictions, which can be achieved by ray tracing and taking into account reflections, diffractions and scattering with the objects in the scene. Unfortunately, the complexity of ray tracing is very high [21, 16].

To balance computational complexity and accuracy, the propagation model implemented in INES takes into account only the objects lying on the direct path (shadowing). The path loss is then calculated by:

$$P_L = P_u dB + 10nlog(d) + \sum_{i-1}^{K} W_i$$

where P_u is the loss for the first meter and n the power-delay index that depends on the environment. For example, $P_u = 40$ and $n = 3.5$ are adequate for indoors. The model assumes a separation distance between transmitter and receiver of d meters, and K walls along the direct path, each producing an attenuation of W_i dB.

4.2 Packet reception

The radio propagation model described in Section 4.1 allows predicting the receiving signal strength for each bit of a packet transmission. In most simulation cases, more than one transmitter may be active at any given time, creating the possibility of interference. The situation occurs for example, when multiple wireless nodes are in relative close proximity or when both a MANET and a WSN operate at the same time on a given area.

The receiver correctly decodes a packet whenever the received signal strength from the transmitter is sufficiently strong to overcome channel thermal noise and interference, in other words, when there is a sufficient signal-to-interference-noise ratio. INES assumes that mobility does not affect a single packet transmission; therefore it is possible to partition any packet reception time (or b bits) into a set of time segments, each with a constant SINR, $T = \cup_i T_i$. The time of one bit is therefore, $T_b = T/b$.

Formally, if P_i is the power received at any point in time from the desired source i and P_j is the power received from other transmitters $i \neq j$. The SINR (γ) at the receiver is:

$$\gamma_i = \frac{G_{ii} P_i}{\sum_{j \neq i} \theta_{ij} G_{ij} P_j + \eta}$$

where G_{ij} is the gain from source j to sink i, θ_{ij} the fraction of the transmitted signal that is projected onto the signal space of i and η the noise power of both thermal noise (No) and receiver noise figure. For different channels in spread spectrum systems with matched filters, it is common to assume $\theta = r/W$, where r is the data rate and W the spread bandwidth.

The SINR of a segment and the modulation system employed determine the probability of decoding the corresponding bits. To decode successfully a packet, it would be necessary to successfully decode all its segments. SINR can be related to the bit error rate (BER or P_e) with knowledge of the modulation scheme. For example, IEEE 802.11b uses DBPSK (1 Mbps), DQPSK (2 Mbps) and CCK (5.5 and 11 Mbps). SINR can be related to BER from analytical and simulation models, or measurements. To avoid the extra computation effort in calculating BER from the simulated SINR of a segment, INES implements look-up tables of known models with linear interpolation and extrapolation.

The probability E_i that segment T_i would contain at least one bit error is:

$$E_i = 1 - (1 - P_e)_i^n$$

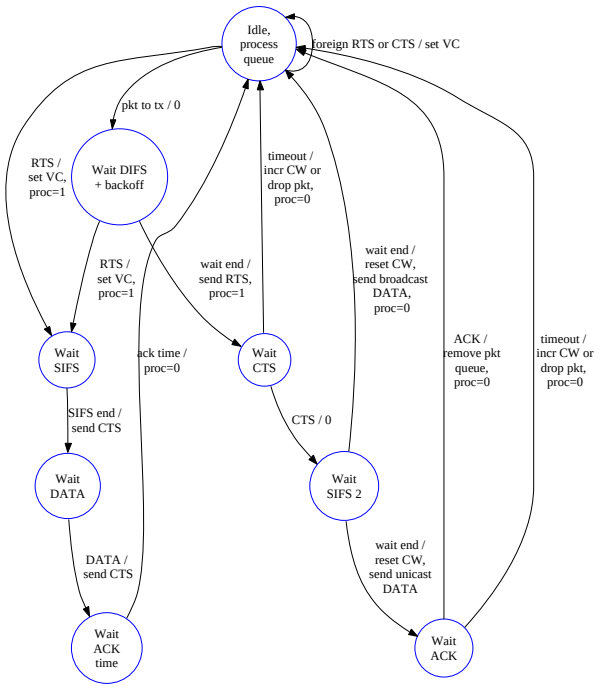

Figure 2: FSM of IEEE 802.11 with RTS/CTS support.

where $n_i = T_i/T_b$ is the number of bits in segment T_i. The model calculates iteratively E_i immediately testing for an error occurred in the segment against a uniformly distributed random variable. If case of error, the calculation is aborted and the packet is declared corrupted.

4.3 IEEE 802.11

INES implements the DCF of IEEE 802.11 as a `Protcomp`, allowing the inclusion of any number of interfaces (with different channels) per node. Multiple transmitters on the same or different channels may cause interference as described in the previous section.

Each interface implements its own buffer, which can be cleared entirely or selectively by destination id on request by another `Protcomp` (e.g. DSR). The implementation follows the standard closely, including the transmission of preambles and allowing the use of RTS/CTS packets (Figure 2) and virtual carrier sensing whenever a packet transmission is for a packet length above the minimum threshold. The implementation uses two timers to track the time progress. One is used for DIFS and SIFS waiting times and the other for tracking timeouts that trigger MAC level retransmissions.

5. PACKET ANIMATOR

Packet Animator is a program being developed concurrently with INES that is able to read and graphically represent the execution of a simulation from INES traces. The program also has partial support for reading NAM traces from NS-2.

The software architecture of Packet Animator uses the observer design pattern, with a model (subject) that reads the events from INES traces to reproduce them over the simulated time. An observer then displays the current state of the model. The model learns from INES key events in the simulation but needs to generate the states between them to produce a smooth animation of the simulation. For example, the model generates the estimated position of nodes and the wireless coverage area from a single INES event indicating the starting time of the action.

Packet Animator is OpenGL based and takes advantage of hardware acceleration whenever available to speed up model drawing. As with INES, the Packet Animator is able to read tri-dimensional geometries in object format (Section 3). In addition to the models defined by INES traces, it is possible to include other geometries in the visualisation (e.g. alternative representation of nodes). The animations support node mobility, the transmission of packets on wired and wireless links and monitors. INES traces may indicate colour changes for nodes and packets at any time during the simulation to facilitate visualisation.

To visually track measurements collected during the simulation, for example, to observe the end-to-end delay of a sequence of packets at some point of the network, the animator allows the creation of monitors (or *meters*), which can plot these values according to the simulation time. It is also feasible to generate special marks (in the form of lines drawn by the animator) from INES to show arbitrary geometries during the visualisation of a simulation. This feature is useful to highlight the resulting pathways of a path planning algorithm. When dealing with wireless links, the animation allows to observe the potential interfering area of transmitters over time.

User interaction is through the OpenGL API, which allows starting, stopping and changing the speed of the animation. The viewing camera is flexible and can be positioned to get suitable views of the animation from any angle and zoom level. Finally, the animator allows to record displayed frames for the creation of movie clips of the simulations.

Next sections (Figures 1, 3 and 5) will show direct screen captures from the Packet Animator. A few video demonstrations are also available online [19].

6. EXAMPLES

This section describes a couple of simple examples that illustrate the simplicity and potential of INES for network simulations. The scripts describing the simulations are very short, so they are included to complement the description. Further applications can be found in the literature [17, 18, 20].

6.1 Small-World Network Topology

A simple example can illustrate the basics of INES of both its user interface and operation. Consider the simulation of a small-world topology of 100 nodes to calculate the average throughput between two nodes (0 and 50) of a CBR traffic flow of packets of 128 bytes. The script describing this simulation is:

```
set_param('link_random', 1);
set_param('link_pdelay', 0.001);
set_param('link_txrate', 1000000);
set_param('buffer_size', 10);

create_net_sw(100, 4, 0.2)
compute_routes()
```

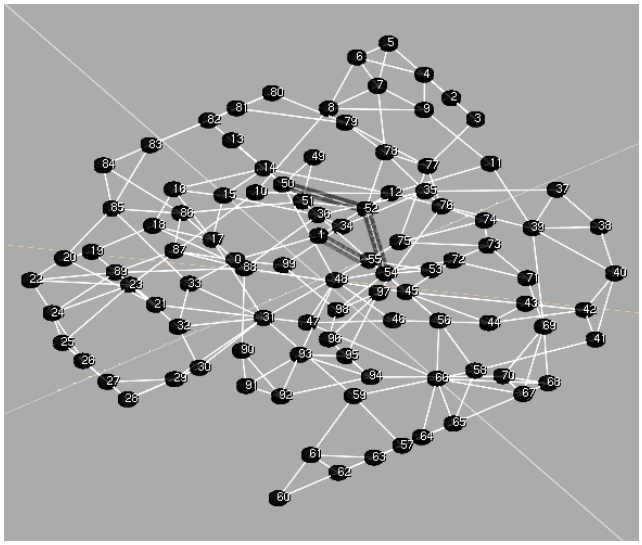

Figure 3: Example 1: Small-world network topology and CBR traffic flow.

```
cmd = "cbr(0, 50, %s, 128, 1000)" % sys.argv[1]
at(0.1, cmd)
```

Function *set_param()* allows the definition of various default values in the simulation. In this script, it enables to randomise the default bandwidth and propagation delay of links at creation time. It also defines the default buffer size for all network interfaces. Function *create_net_sw()* creates the network topology with 100 nodes having a nominal degree of 4 with a probability of re-wiring of 0.2. Function *compute_routes()* creates static routing tables in the nodes which use IP and UDP by default. Finally, a traffic flow of 1000 packets is scheduled to start at time 0.1 with a bit rate supplied from the command line.

Given the random nature of the topology, the simulation was executed several times to obtain average values. Figure 3 depicts one of the resulting topologies in the Packet Animator and the results are shown in Figure 4.

6.2 Mobile Wireless Network

As a second example, consider a typical urban scenario (Figure 5) with 30 nodes moving at random speeds between 2 and 5 m/s and according to the random-waypoint model with no pause times. Random destinations, outside the buildings, are selected every time a node stops and nodes move avoiding obstacles as described in Section 3.1. The script describing the simulation is:

```
s0 = scene("buildings.obj")

w0 = link("wlan", "dsr")
w0.param('link_txrate', 11e6);
w0.param('buffer_size', 50);
w0.param('rtscts_threshold', 0);

mov = mover("rwp")
mov.param("initpos", 1)
mov.param("field", 350.0, 250.0)
mov.param("minspeed", 2.0)
```

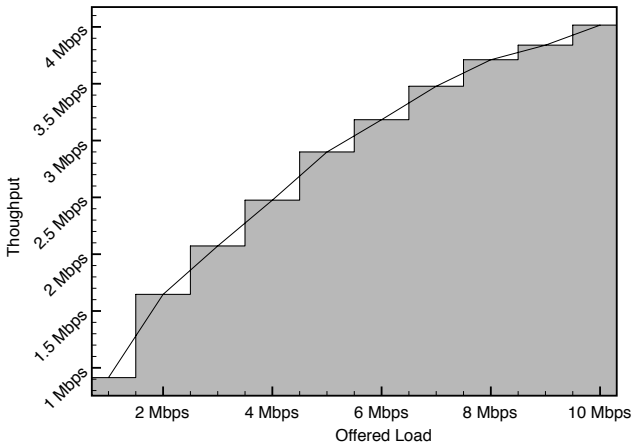

Figure 4: Measured throughput between nodes 0–50.

```
mov.param("maxspeed", 5.0)
mov.param("minpause", 0.0)
mov.param("maxpause", 0.0)

N = 30
n = {}
for i in range(0,N):
        n[i] = node()
        w0.connect(n[i])
        mov.add(n[i])

at(0.0001, "mov.start()")
at(12000, "mov.stop()")
at(1.0, "cbr(n[0].id(), n[1].id(), 16000, 256, 85000)")

start()
```

Scenarios can be loaded with the `scene()` function, which accepts standard .obj files. The `link()` function creates in this case, a wireless link (wlan). The function is overloaded to accept an association with a MANET routing protocol (DSR). The `mover()` function returns an instance of the mover indicated in the argument call (random way point in this case). A mover object can then be manipulated to change its default values to the desired field size, speed and pause times. Nodes can be subscribed to the mover with `Mover::add()`. By default, simulations check object collisions and use path planning for mobiles with object avoidance, so there is no need to define them in the script.

One CBR packet flow of UDP/IP packets of 256 bytes at 16 Kbps is started from node 0 to node 1. The goal of this simulation is to find out the expected path length of the flow and compare it to the results of a similar scenario without obstacles. The simulation script for the latter case is identical to the one just shown, but without the line with the `scene()` function.

Nodes use a transmission power of 16.6 dBm and a transmission rate of 11 Mbps. Each object face in the scenario attenuate the incoming signal by 3 dB. Under the free-space propagation model, this transmission power allows for a communications range of about 150m. Figure 5 shown a screen capture of the Packet Animator illustrating the same

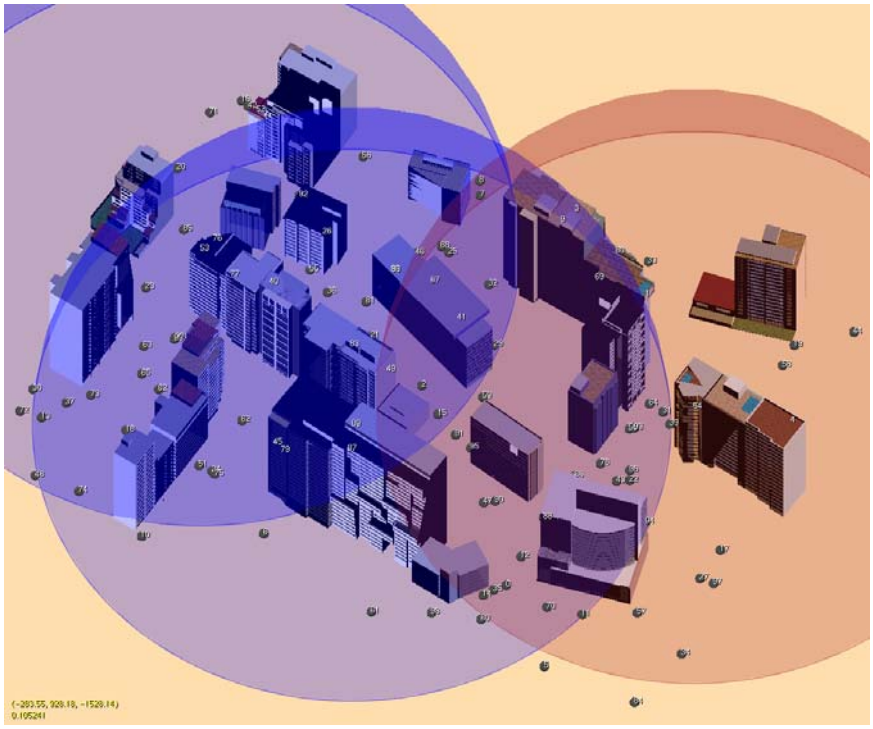

Figure 5: Urban scenario and IEEE 802.11 / DSR mobile network.

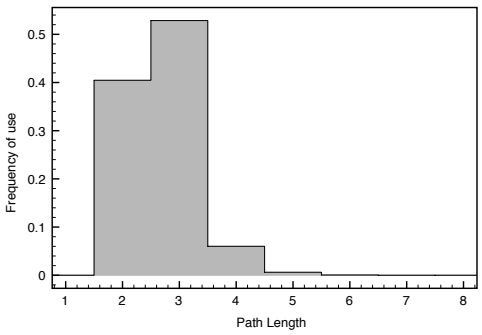

Figure 6: Distribution of path lengths of packet transmissions on an unobstructed scenario.

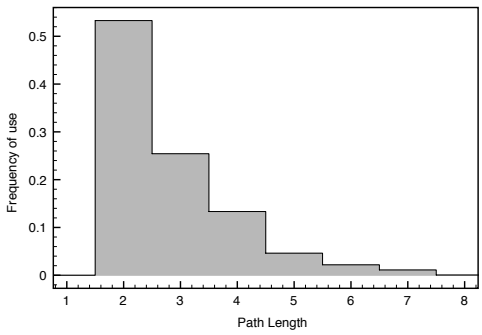

Figure 7: Distribution of path lengths of packet transmissions on the scenario shown in Figure 5.

scenario but with 100 nodes. During the animation, translucent semi-spheres are drawn around radio transmitters to represent the expected temporal radio coverage over time. However, the actual coverage may be less because of radio interference and obstacles (Sections 4.1 and 4.2), which are determined packet by packet. The average observations after running the script 50 times are shown in Figures 6 and 7. They show the normalized histogram of observed path lengths (frequency of use) and indicate that paths tend to be longer when obstacles are present as expected. Longer paths are the result of obstacles that restrict the physical location of nodes to certain areas on the field and that reduce the effective coverage of wireless communications.

7. CONCLUSION

The paper has discussed the goals, design, models and current development state of INES. INES is a software project aimed at creating a network simulator for the realistic evaluation of computer networks on a virtual environment. The simulator defines an environment with a set of tri-dimensional objects, which may affect wireless communications by introducing extra attenuation to signals when they need to traverse the scenario structures. The extra attenuation produces a higher packet loss probability and reduces effective wireless ranges when compared to unobstructed transmissions. Objects may also restrict node roaming.

Consideration of the environment effects on wireless communications and node roaming could be crucial for the cor-

rect design and evaluation of wireless algorithms and related technologies. Simulations that unrealistically assume unobstructed environments could lead to significant errors. To illustrate the effects of the environment, the paper has evaluated a MANET on a urban setting and quantified the difference in DSR hop counts compared to an scenario without obstructions. The longer paths obtained with the obstructed environment would cause higher energy consumption than expected, which comes from the need of more packet transmissions to cover the longer paths or from the use of higher power transmissions to maintain the hop counts.

INES is based on an object model that is flexible and extensible. It offers support for the concurrent use of multiple interfaces to wireless and wired media. Current supported models include wired communications with INET and CPN protocols, wireless LANs using the IEEE 802.11 DCF protocol, radio propagation with intra-channel and extra-channel interference tracking for packet error computation. There is support for path planning with obstacle avoidance and both the random waypoint and Gauss-Markov mobility models. On the other hand, it uses an embedded Python interpreter that make simulations easy to define and execute as shown in the examples.

The Packet Animator complements INES and implements an OpenGL-based animation engine that can visualise INES simulation traces. INES is a work in progress and it is expected that future developments will enhance the model support and node-environment interactions, as well as to include emulation and distributed execution.

8. ACKNOWLEDGMENT

The work presented in this paper was partially supported by the project CASCADAS (IST-027807) funded by the FET Program of the European Commission. The paper represents the work and contribution of an individual party involved in the project.

9. REFERENCES

[1] Boost c++ libraries, online: *http://www.boost.org*.

[2] Google earth, online: *http://earth.google.com*.

[3] Graphviz - graph visualization software, online: *http://www.graphviz.org*.

[4] Lam/mpi, online: *http://www.lam-mpi.org*.

[5] Omnet++ simulator, online: *http://www.omnetpp.org*.

[6] Opnet simulator, online: *http://www.opnet.com*.

[7] Qualnet simulator, online: *http://www.scalable-networks.com*.

[8] L. Bajaj, M. Takai, R. Ahuja, R. Bagrodia, and M. Gerla. Glomosim: A scalable network simulation environment. Technical Report 990027, University of California Los Angeles, 13, 1999.

[9] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu. Advances in network simulation. *IEEE Computer*, pages 59–67, May 2000.

[10] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *Wireless Communication & Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, 2(5):483–502, 2002.

[11] J. Cowie, H. Liu, J. Liu, D. Nicol, and A. Ogielski. Towards realistic million-node internet simulations. In *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, June 1999.

[12] E. Gelenbe and R. Lent. Power-aware ad hoc cognitive packet networks. *Ad Hoc Networks Journal*, (3):205–216, July 2004.

[13] E. Gelenbe, R. Lent, and Z. Xu. Measurement and performance of cognitive packet networks. *J. Computer Networks*, 37:691–701, 2001.

[14] A. Jardosh, E. Belding-Royer, K. Almeroth, and S. Suri. Real-world environment models for mobile network evaluation. *Selected Areas in Communications*, 23:622–632, Mar 2005.

[15] M. Lacage and T. R. Henderson. Yet another network simulator. In *WNS2 '06: Proceeding from the 2006 workshop on ns-2: the IP network simulator*, page 12, New York, NY, USA, 2006. ACM.

[16] R. Lent. Design of a manet testbed management system. *The Computer Journal*, 49(4):171–179, jul 2006.

[17] R. Lent. On the impact of network qos on automated distributed auctions. In *Proceedings of 2nd International Conference on Bio-Inspired Models of Network, Information, and Computing Systems, Budapest, Hungary*, December 2007.

[18] R. Lent. Autonomic migration response to ddos attacks in an auction system. *Security and Communication Networks (to appear)*, 2008.

[19] R. Lent. The integrated network-environment simulator (ines), online: *http://san.ee.ic.ac.uk/ines*, 2008.

[20] R. Lent and E. C.-H. Ngai. Lightweight clustering in wireless sensor-actuator networks on obstructed environments. In *Proceedings of International Symposium on Wireless Pervasive Computing, Santorini, Greece*, May 2008.

[21] G. Liang and H. L. Bertoni. A new approach to 3-d ray tracing for propagation prediction in cities. *IEEE Trans. Antennas and Propagation*, 46:853–863, 1998.

[22] K. Maeda, K. Sato, K. Konishi, A. Yamasaki, A. Uchiyama, H. Yamaguchi, K. Yasumoto, and T. Higashino. Getting urban pedestrian flow from simple observation: Realistic mobility generation in wireless network simulation. In *Proceedings of the 8th ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM2005)*, 2005.

[23] C. McDonald. Network simulation using user-level context switching. In *Proc. of the Australian UNIX Users' Group Conference '93*, pages 1–10, Sep 1993.

[24] T. S. Rappaport. *Wireless Communications*. Prentice Hall PTR, 1996.

[25] G. Riley. Large-scale network simulations with gtnets. In *Proceedings of the 2003 Winter Simulation Conference*, pages 676–684, Dec 2003.

[26] A.-K. H. Souley and S. Cherkaoui. Simulating realistic urban scenarios for ad hoc networks. *International Journal of Business Data Communications and Networking*, 2:18–33, 2005.