Bandwidth Analysis of a Simulated Computer Network Executing OTB

J. Vargas, R. DeMara, A. Gonzalez, M. Georgiopoulos

University of Central Florida
Orlando, Florida

jvargas@ucf.edu, demara@mail.ucf.edu

ABSTRACT

This paper describes the development of an Embedded Training computer network simulation using the OMNeT public-domain modeling tool. The purpose of this simulation is to determine the bandwidth requirements of military mission rehearsal activities while enroute to deployment. A predefined vignette running under OneSAF Testbed Baseline (OTB) provides the base data for the simulator. The simulation includes 24 computers onboard 8 military airplanes, 3 computers per plane, representing 3 combat vehicles connected via a 100Mbps Ethernet LAN. A router interconnects the resources aboard each plane to those of other planes and also to a remote ground station via 64Kbps and 256Kbps wireless communications channels. The simulation is run and the results are collected, plotted, and several conclusions are drawn regarding bandwidth and latency implications of the Embedded Training exercises.

The simulation model of this system in OMNeT is also presented in detail. OMNeT is a general discrete event simulation tool that contains features aimed to facilitate computer network modeling. OMNeT runs under different platforms including UNIX systems, Linux, and Windows. It contains a graphical development environment that can be used to instantiate and connect varying icons representing the objects of the simulation: computers, communication channels, routers, airplanes, etc. The behavior of each object is then defined using C++ code. The final product is an executable standalone program that models system behavior generated using a variety of libraries. It runs as an animation or as an express simulation. As shown, a wide range of statistics is produced and immediately available for analysis.

Bandwidth Analysis of a Simulated Computer Network Executing OTB

Juan J. Vargas, Ronald F. DeMara
University of Central Florida
Orlando, Florida
jvargas@ucf.edu, demara@mail.ucf.edu

INTRODUCTION

In 2002, the U.S. Army Simulation, training and Instrumentation Command (STRICOM) and the Computer Engineering Department of the School of Electrical Engineering and Computer Science at the University of Central Florida (UCF) began a project to assess "*Bandwidth and Latency Implications of Integrated Training and Tactical Communication Networks*." The main goal of the project is to determine the bandwidth requirements for running Objective Force Embedded Training (OFET) methods.

OFET methods benefits include the ability to perform tasks such as mission planning and rehearsal while enroute to deployment. Benefits include the ability to perform "in-situ" exercises on actual equipment, more direct provision of support for the variety of equipment in the field, and a greater opportunity to develop new training exercises using much shorter lead times than were previously possible with stand-alone training systems.

A fully operational OFET platform also presents several challenging, yet surmountable technology challenges. In particular, management of Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance (C4ISR) resources is required for successful integration of simulation within the actual environment. The purpose of this study is to assess communication requirements to support Enroute Mission Planning and Rehearsal (EMPR) in a Future Combat System (FCS) environment.

The first step towards the development of the network simulator was the design of the vignette illustrating a mission rehearsal operation enroute to deployment. This vignette included a database terrain, friendly and enemy forces, strategies, and other details representative of FCS operations. The vignette was run using the software tool called OneSAF Testbed Baseline (OTB), which acts as a simulator for executing the vignette.

Each entity in OTB communicates with the other entities by a message-passing paradigm. Messages in OTB are called *Protocol Data Units* (PDUs). PDU formats are defined in the IEEE Standards (IEEE Std 1278.1, 1995), (IEEE Std 1278.2, 1995), (IEEE Std 1278.3, 1996) and (IEEE Std 1278.1a, 1998). After running the OTB simulator, the PDUs generated in that session are collected and stored in an output file for further processing. In this project, the three most relevant features of the PDUs are the sender ID, the byte length and the timestamp of creation because the network modeling tool uses these parameters as a basis to model communication traffic. The destination address is not part of PDUs because OTB broadcasts all the messages to all of the participating entities.

The developing team selected a discrete simulation tool that was very flexible from a programming point of view, Windows-based for availability and readiness, provided an extensive Graphical User Interface (GUI) capable of showing simulation with animation for demonstrations, easy to learn for C++ users, and of low cost. After a short survey, the team decided to select OMNeT++ (OMNeT, 2003), a software tool developed by András Varga at the Budapest University of Technology and Economics. It is a general discrete event simulation tool that contains features to facilitate computer network simulations. OMNeT runs under different platforms including UNIX systems, Linux and Windows. It contains a graphical development environment that can be used to instantiate and connect varying icons representing the objects of the simulation: computers, communication channels, routers, airplanes, etc. as described below.

USING OMNET

OMNeT is a discrete event simulation environment based on C++. It provides means for describing the topology of the network graphically and through the use of the NED language (Varga, 2003). In particular, OMNeT operates on two types of files.

The first type corresponds to NED files (xxx.ned) that describe the different elements or nodes involved in the simulation, their input and output gates and the connections between them. In other words, the NED files describe the topology of the network. A NED file uses a special language called the NED language to describe modules. There are two types of modules. *Simple modules* contain no other modules inside them. They are used to describe the most basic elements of the simulator. In this project, message generators, message sinks, communication channels (wireless and Ethernet buses) and routers correspond to simple modules. *Compound modules* contain other modules inside. For example, a computer onboard an airplane is a compound module because it contains a message generator and a sink. A plane is also a compound module that contains a computer, a router and an Ethernet bus. The largest compound module corresponds to the total network that contains airplanes, and a wireless bus to link the planes.

The second type of files corresponds to CPP files (xxx.cpp) that use C++ to describe the functionality of the simple modules. Each simple module requires a C++ source code that indicates how to process each packet that arrives to an input gate, as well as how to send a packet to an output gate. Using C++ code we can program the packet contents, its destination, its length, and the distribution used (exponential, normal, uniform, etc.) to schedule packets to be sent. The CPP files corresponding to the buses (Ethernet or wireless) handle the transmission of packets including delivery and collision detection.

Once the source files are ready, the simulator is compiled. In this project, Microsoft Visual C++ was used to accomplish this task. The NED files are precompiled by using a special compiler that is supplied with OMNeT. The result of compiling a NED file, say xxx.ned, is a source CPP file called xxx_n.cpp. Next, all the cpp files are compiled together with a graphic library called Tcl/Tk and the executable file is created. This graphic library is software in the public domain (Tcl Developer Xchange, 2003).

For C++ programmers, a sufficient understanding of the way OMNeT works along with the capability of developing basic models can be achieved in a very short time because the main concepts come from general knowledge about C++, as compared to other simulators like in which there are several long manuals to read and the concepts learned are specific to that simulator.

MODEL DESIGN

The model to be simulated is composed of eight airplanes carrying three computers and a router each, plus a satellite and a ground station. The generation rate and inter-message distribution for packets from each component can be driven by a random message generator with a specific distribution. Alternatively, as in our study, each component's C++ code actually reads the messages to be broadcasted from an input text file that contains the timestamp, length and type of each message as noted by OTB's PDU logger as a result of running the EMPR vignette.

Module descriptions

The topology of the network model is described by NED files. Additionally, compound modules can be displayed as a graph. Due to length constraints, Figures 1, 2 and 3 show the graph of the compound modules without the associated NED codes. Figure 1 shows the sink and the generator of each computer.

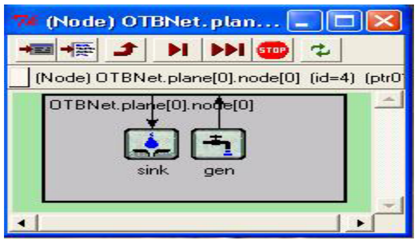

Figure 1. OMNeT View of a Embedded Training Node and Components.

The legend `OTBNet.plane[0].node[0]` indicates that this compound module belongs to node # 0 located inside plane # 0 which is part of the general network called OTBNet. The eight planes are identified by using consecutive integers 0, 1, 2, … , 7. Within each plane, the three computers are identified as node 0, node 1, and node 2. The brackets indicate that the set of planes and the computers onboard each plane are represented as arrays of simple modules. The arrow entering the sink and the one leaving the generator represent the input and output gates.

Figure 2 shows the three computers, the router and the Ethernet bus onboard each airplane.

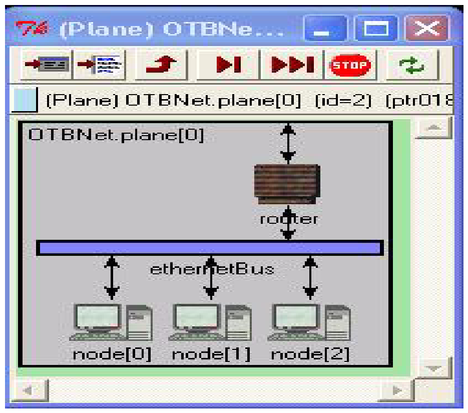

Figure 2. Airplane Components - Embedded Training Stations, Bus, and Router.

Figure 3 shows the complete network composed of 8 airplanes, a ground station, a satellite and 3 wireless channels.

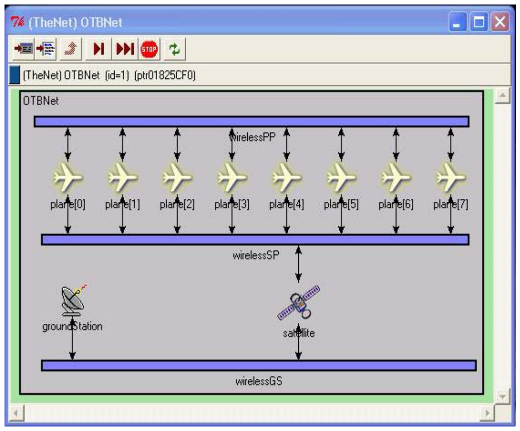

Figure 3. General View of the Network Showing 8 Planes, Satellite, Ground Station and the 3 Wireless Channels**.**

As depicted in Figure 3, the architecture is composed of eight airplanes carrying three computers and a router each, plus a satellite and a ground station. Each computer onboard an airplane contains a message generator and a message sink. The three computers are connected via an Ethernet bus. The router is also connected to this bus as well as to a wireless link used to interconnect all the planes. Another wireless link connects the routers in all planes to a communications satellite that in turn is connected via a third wireless link to a ground station. In this way, each router is connected to three different links and the satellite is connected to two. The NED code of some simple modules is listed in Figure 4.

Figure 4 contains the NED code of a generator, a sink and the satellite. For simple modules, the NED code indicates the module name, the input parameters and the input and output gates that connect this module to others.

```
simple Generator
parameters:
startTime: numeric,
fromAddr: numeric,
totalNodes: numeric;
gates:
out: out;
endsimple

simple Sink
gates:
in: in;
endsimple

simple Satellite
parameters:
startTime: numeric,
satelliteID : numeric,
satServiceTime : numeric,
totalNodes   : numeric,
WGSposition : numeric,
WSPposition : numeric;
gates:
in:  inBus1;
out: outBus1;
in:  inBus2;
out: outBus2;
endsimple
```

Figure 4. NED Code of Some Simple Modules.

The NED code of a compound module includes more features like the values of parameters of internal modules and connections between module gates. The C++ code of simple modules is written in a separate file. Each simple module becomes a C++ class. Compound modules do not need user-written C++ source code because their behavior is completely defined by their simple modules. All NED files are converted to C++ by a NED compiler that comes with OMNeT++.

The input parameters to the modules are used to control the conditions under which each simulation runs. These parameters can be directly given in the NED files, or they can be read from a configuration file at run time. Each OMNeT simulation starts by reading the configuration file called `omnetpp.ini` that contains initialization values and input parameters.

Figure 5 shows the configuration file `omnetpp.ini` used in one of the simulations of this project.

```
[General]
network = OTBNet

ini-warnings = no
random-seed = 1
warnings = yes
snapshot-file = planes.sna
output-vector-file = planes.vec
sim-time-limit = 2550s  # simul. sec
cpu-time-limit = 20h    # 20 hours
total-stack-kb = 4096
[Cmdenv]
module-messages = yes
verbose-simulation = yes
display-update = 0.5s
[Tkenv]
default-run=1
use-mainwindow = yes
print-banners = yes
slowexec-delay = 300ms
update-freq-fast = 10
update-freq-express = 100
breakpoints-enabled = yes
[DisplayStrings]
[Parameters]
[Run 1]
OTBNet.startTime     = 1034s
OTBNet.nodesPerPlane = 3
OTBNet.numPlanes     = 8
OTBNet.LANgapTime   = 50us
OTBNet.LANbandwidth = 100E6
OTBNet.LANdelay     = 4.761904762ns
OTBNet.WPPgapTime   = 50us
OTBNet.WPPbandwidth = 64000
OTBNet.WPPdelay     = 3.333333333ns
OTBNet.WSPgapTime   = 50us
OTBNet.WSPbandwidth = 64000
OTBNet.WSPdelay     = 3.333333333ns
OTBNet.WGSgapTime   = 50us
OTBNet.WGSbandwidth = 64000
OTBNet.WGSdelay     = 3.333333333ns
OTBNet.satServiceTime = 5us
OTBNet.routerServiceTime = 5us
```

Figure 5. Omnetpp.ini File Used in Model Execution.

INPUT DATA

The input data to the model comes from the OTB logger. After setting up a particular vignette, OTB runs it and the logger records all the PDUs into an output file that is later converted to text. The PDUs are the messages to be exchanged by the entities during the simulation. OTB generates PO_PDUs (Persistent Object PDUs), which are a specialization of the general category of PDUs.

Figure 6 is an example of a PO_PDU. This is a typical PDU. From among all the fields, the most important to the OMNeT simulator are those containing the *site* identification, the *length* in bytes and the *timestamp*. The example shows `do_header.object_id. Simulator. site = 1082`, `po_header. length = 147` bytes, and `dis_header.timestamp = :01: 33.432` (1 minute, 33 seconds and 432 milliseconds). The timestamp represents the time the entity generated this PDU and put it on the output queue.

```
<dis204 po_variable PDU>:
dis_header.version=4
dis_header.exercise=1
dis_header.kind=250
dis_header.family=140
dis_header.timestamp=:01:33.432 (rel)
dis_header.sizeof=196
po_header.po_version=28
po_header.po_kind=2
po_header.exercise_id=1
po_header.database_id=1
po_header.length=147
po_header.pdu_count=7905
do_header.database_sequence_number=0
do_header.object_id.simulator.site=1082
do_header.object_id.simulator.host=23825
do_header.object_id.object=685
do_header.world_state_id.simulator.site=0
do_header.world_state_id.simulator.host=0
do_header.world_state_id.object=0
do_header.owner.site=1082
do_header.owner.host=23825
do_header.sequence_number=1
do_header.class=11
do_header.missing_from_world_state=0
reserved9=0
variable.total_length=132
variable.expanded_length=7812
variable.offset=0
variable.length=132
variable.obj_class=8
variable.data="Mine Pallet US M75
```

Figure 6. Example of a PO_PDU Produced by OTB

SIMULATION RESULTS

The simulation referred to in this paper is called "Experiment # 3" and corresponds to the third experiment in a series of four. The experiment generates a set of PDUs involving six senders. The rest of the computers are listeners. The sites sending information were assigned to computers in separate airplanes and the ground station. Two types of analyses were performed.

The first analysis can be called *independent* or *static* analysis because it does not take into account interaction among senders or resource availability. This analysis is subdivided into 2 categories: analysis and assignment of PDUs, and minimum bandwidth requirements.

Analysis and Assignment of PDUs

Figure 7 shows a frequency distribution of all the types of PDUs involved in the experiment, as well as the assignment of sites to actual computer nodes. The assignment was based on the number of PDUs generated by each site.

```
<dis204 fire PDU>:                  23
<dis204 po_objects_present PDU>:   682
<dis204 po_minefield PDU>:          14
<dis204 minefield PDU>:            117
<dis204 iff PDU>:                  851
<dis204 acknowledge PDU>:           36
<dis204 stop_freeze PDU>:            3
<dis204 po_line PDU>:              912
<dis204 po_task_authorization PDU>: 6
<dis204 po_task_frame PDU>:        382
<dis204 po_message PDU>:           119
<dis204 aggregate_state PDU>:      256
<dis204 po_delete_objects PDU>:    110
<dis204 po_parametric_input PDU>:1196
<dis204 laser PDU>:                  3
<dis204 detonation PDU>:            25
<dis204 po_fire_parameters PDU>:   713
<dis204 po_simulator_present PDU>:370
<dis204 entity_state PDU>:       28569
<dis204 mines PDU>:                386
<dis204 po_point PDU>:             659
<dis204 po_task_state PDU>:      11960
<dis204 signal PDU>:               237
<dis204 po_task PDU>:             2274
<dis204 po_unit PDU>:             1793
<dis204 transmitter PDU>:         8642
<dis204 start_resume PDU>:           3

Total PDUs = 60341

Site Assignment: (p=plane, n=node)

Site 1519 (50230 PDUs): (p=0, n=0)
Site 1526 (1056 PDUs): (p=1, n=0)
Site 1529 (483 PDUs): (p=2, n=0)
Site 1532 (7382 PDUs):(ground station)
Site 1533 (553 PDUs): (p=3, n=0)
Site 1538 (637 PDUs): (p=4, n=0)
```

Figure 7. Frequency Distribution of PDUs.

Minimum Bandwidth Requirements

The PDUs of all the sites were merged into one single stream of data and sorted according to their timestamps before any bandwidth computations. This procedure is justified by the fact that all the PDUs are broadcasted and any listening site will have to receive the PDUs from all the generating sites as one single stream of data. Using the sorted PDUs, a separate program calculates the required bandwidth at specific time intervals without using any simulation. Due to traffic changes in time, different minimum instant bandwidths are necessary. This calculation is computed as the ratio of volume of data transmitted to the time interval allotted among consecutive PDUs. Figure 8 shows the graph of minimum bandwidth required at each time. The experiment starts at second 1035 and ends at second 2550. That interval corresponds to 25 minutes and 15 seconds of simulation time.

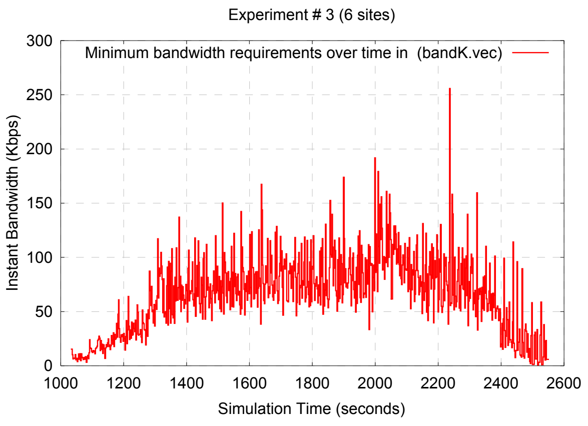

Figure 8. Minimum Bandwidth Requirements.

In the static analysis, overheads like retransmissions, packet losses, or collisions are not evaluated. Therefore, the resulting bandwidth estimates can be interpreted as an absolute lower bound for the actual required bandwidth by the C4ISR network. If several PDUs bear the same timestamp, their data volumes are included in the same bandwidth computation. Also, it was taken into account that a time gap must exist between packets, as defined by the IEEE Standard (IEEE Std 802.11, 1997). This gap was set to 50 microseconds for all the buses. If some PDUs are included in the same set and, after considering the timestamp and gaps, it is determined that the next PDU cannot be transmitted in a different set, that PDU is included in the already built set. Otherwise, the bandwidth for the current set is calculated and a new collection of PDUs starts. However, if the current set spans over a time interval of size less than a given parameter, the next PDUs are included within this set regardless of the possibility of a separate transmission. In the results shown, the time interval was set to 2 seconds.

This modeling of the transmission process is performed for various channel capacities. From the graph in Figure 8, the static analysis indicates that the maximum required bandwidth is near 256 Kbps, but the majority of the time the required bandwidth is less than 200 Kbps.

The second analysis performed corresponds to more detailed, realistic interactions obtained by running the OMNeT model to take into account interactions between senders and resources during transmission. This analysis is subdivided into 4 categories: *slack time analysis*, *travel time analysis*, *queue length analysis* and *collision analysis*.

Slack Time Analysis

The *slack time* for each computer node generator is defined as the difference between the timestamp of each PDU and the current simulator time at the moment the PDU is read from the input file. If the difference is positive then the generator is ahead of the planned schedule, otherwise it is behind it. Thus, a negative slack time indicates that the channel bandwidth is not sufficient to transmit the required PDUs without delay.

Figures 9 and 10 show the slack time for all the units (routers and ground station) when the wireless channels are set to 64 Kbps

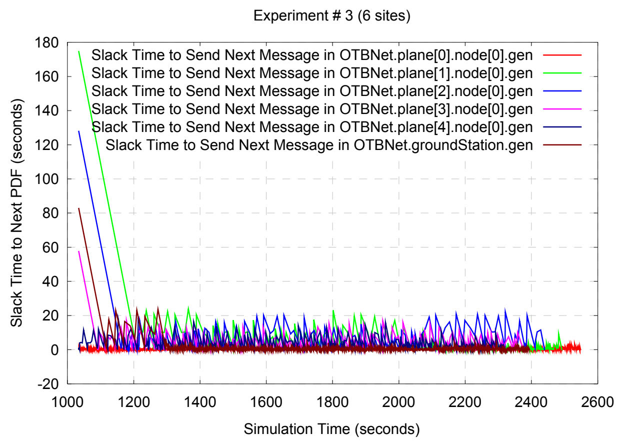

Figure 9. Slack Times at Each Transmitting Node Using 64 Kbps in the Wireless Channels.

Figure 10 shows that the ground station introduces the majority of the negative slack events. This is explained by the fact that the generators onboard the planes are directly connected to high speed Ethernet buses, while the ground station is connected to a lower speed wireless channel.

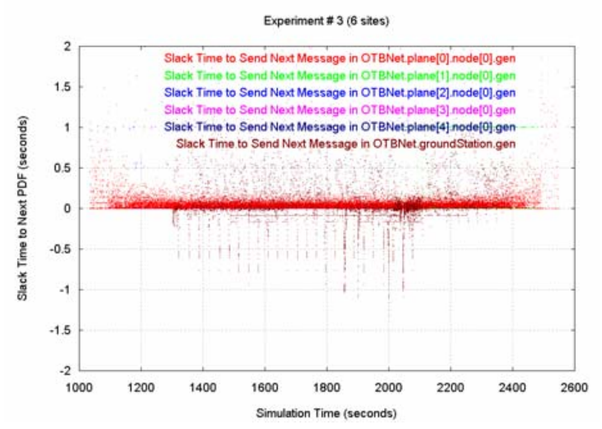

Figure 10. Slack Times at Each Transmitting Node Using 64 Kbps in Wireless Channels.

Figure 11 shows the effect of increasing the wireless bandwidth from 64 to 200, 512 and 1024 Kbps in the ground station channel.

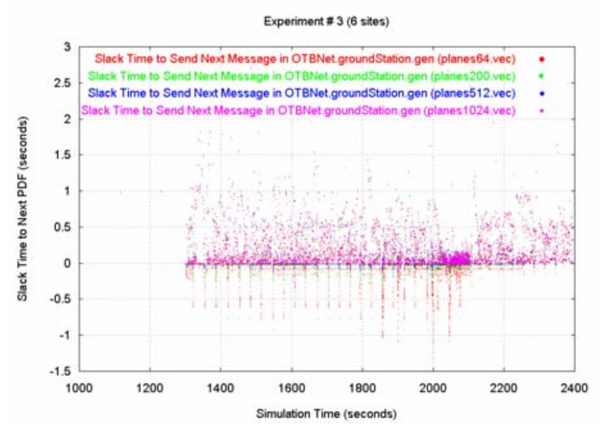

Figure 11. Slack Times at Ground Station Using Wireless Bandwidths of 64, 200, 512 and 1024 Kbps.

Clearly, the more the bandwidth is increased, the less negative slack is encountered. Despite this, some negative slacks will not disappear if the PDUs are not rescheduled.

Travel Time Analysis

The *travel time* is the difference between the arrival time of a PDU to a computer node sink and the sending time from a computer node generator.

Figure 12 show the travel time of PDUs measured by the sink at node 0 using 64 Kbps on the wireless links.

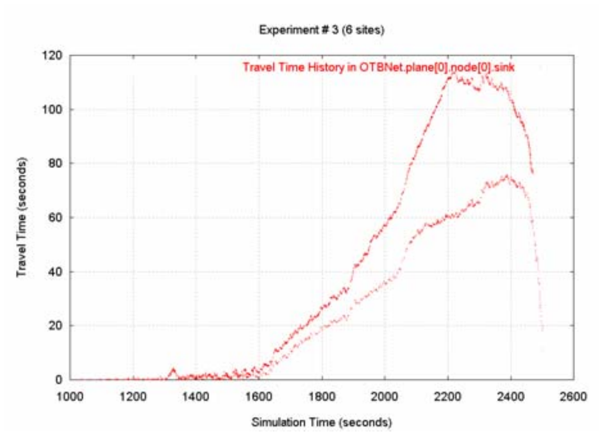

Figure 12. Travel Time Measured at Node 0, Plane 0.

At node 0 the graph clearly shows two types of PDUs. The PDUs that take longer to arrive come from the ground station. These PDUs had to wait on the satellite queue as well as on the router queue. On the other hand, the PDUs coming from computers onboard the other planes had to wait on the router queue only. This produces the two traces as shown. At 64Kbps the travel times of most PDUs are completely unacceptable. Some PDUs took more than 100 seconds since the time they were sent to the time they arrived. The ground station presents a similar behavior due to the relatively long queue and transit times associated with the satellite.

Figure 13 shows travel times measured at the ground station using bandwidths of 64, 200, 512 and 1024 Kbps for wireless links. The graph was zoomed in on the Y axis to show the details in the neighborhood of 0 to 4 seconds. It is worth noting the significant difference between 64 Kbps and 200 Kbps channel capacities. At 200 Kbps, the travel times to the ground station are less than 1 second. Considering that the minimum travel time is about 0.25 seconds due to the satellite distance from Earth, latencies less than 1 second are within a few factors of optimal. At 200 Kbps, the green line shows many discrete spikes that can be diminished by a better scheduling policy, especially if OTB were set to deliver the PDUs in a less bursty mode which is a subject of follow-on research.

Queue Length Analysis

There is a message queue at the satellite and at each router. Every time a PDU arrives to a router or satellite, it counts the number of other PDUs in the queue plus any PDU being serviced. This counter is the *queue length* in terms of the number of messages in the system and is recorded along with the current simulated time.

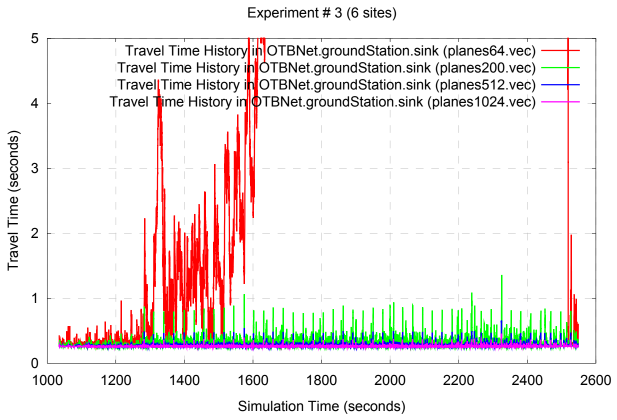

Figure 13. Travel Time Measured at Ground Station Using 64, 200, 512 and 1024 Kbps.

The two most important queues to analyze are the queue at the router onboard plane 0 and the queue at the satellite. Figure 14 shows the messages at the router queue onboard plane 0 and Figure 15 shows the messages at the satellite.

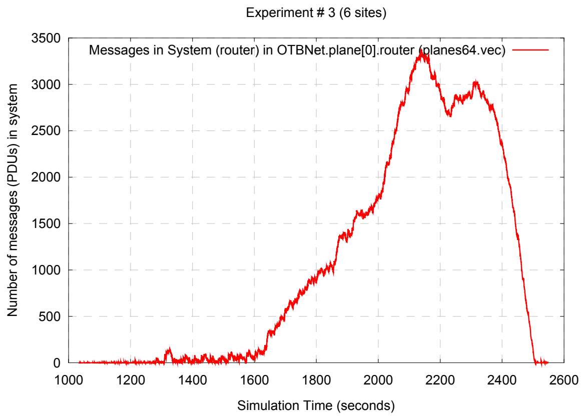

Figure 14. Messages in Router at Plane 0.

The graph represents the number of messages in the router at node 0, using 64 Kbps in the wireless channels. As observed, the queue length becomes really unacceptable, reaching more than 3000 messages during some periods.

On the contrary, routers at other planes have acceptable queues. For instance, plane 3 (graph not shown) has a queue with a maximum of 23 messages.

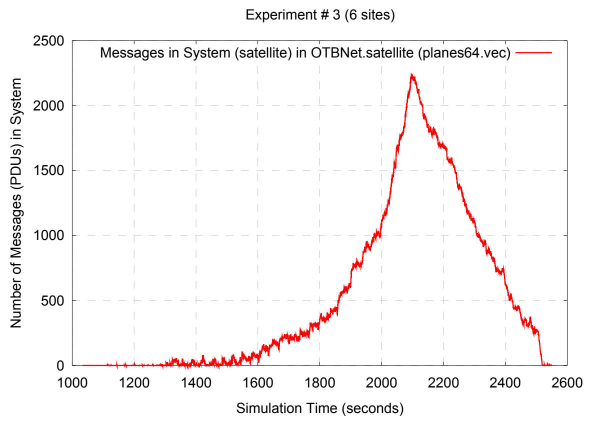

Figure 15. Messages in the Satellite at 64 Kpbs.

The reason is that the corresponding node 9 transmits only 553 PDUs, a number easily handled by the router. As seen in Figure 15, the queue at the satellite (64Kbps) shows unacceptable behavior with a peak of more than 2200 messages.

Simulations run using bandwidths of 64, 200, 512 and 1024 Kbps in the wireless channels showed the effect of a bandwidth increase over the queue length of the router onboard plane 0 as well as over the satellite. The results showed that just by increasing the bandwidth to 200 Kbps the queue length at plane 0 decreases to less than 60 messages in its highest peak. At the satellite less than 40 messages are held in the system during the highest peak. Therefore, this change of transmission speed is remarkable.

Collision Analysis

The satellite and routers keep separate counters for *collisions* of PDUs on each of the buses they are connected to. The satellite and the routers are connected to 2 and 3 buses, respectively, as explained in the Model Design section. Each time a collision is detected, the counter of the collided bus is incremented and its value along with the current simulation time is recorder for future processing. OMNeT collects these results in the form of a time series for plotting.

The results of simulations indicate that at 64 Kbps the highest collision rates occurred in the bus connecting the satellite to the planes and are close to 12 collisions per second, as computed by the router at plane 3. This number is different if computed by different entities. For example, the satellite observed 6 or less collision per second for the same bus.

A reason to explain this difference in collision behavior has to do with how the communication channels were specified to function. The buses were programmed such that a message delivered at some bus gate is not returned back to the sending entity, even if it collided. As a consequence, it is advisable to observe a quiet listener as a sentinel to detect collisions.

Figure 16 shows the collision accumulation sensed at planes 1, 2, and 7. Plane 7 is not an active sender making it a good indicator. The statistics are very similar in the three cases. More than 6000 collisions were detected at 64 Kbps, which represents approximately 10 % of the total number of PDUs.

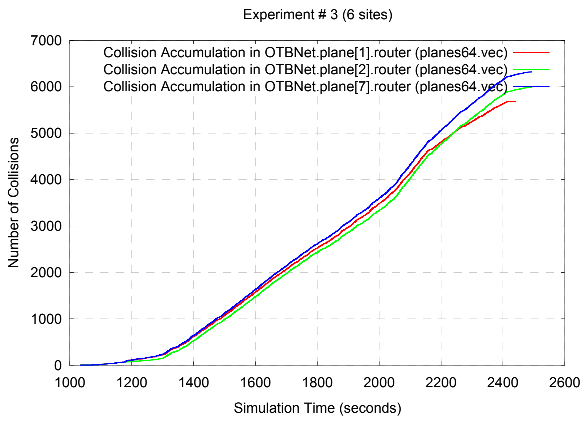

Figure 16. Collision Accumulation at Planes 1, 2 and 7 at 64 Kbps.

At 200 Kbps, results indicate that plane 7 detects a maximum of 4 collisions per second and the total number of collisions is near 4500, which represents about 7% of all the PDUs. Collision accumulation from plane 7 as a function of time for different bandwidth rates is given in Figure 17.

The collisions were calculated as the sum of collisions detected in the three buses that the router is connected to. However, the main component comes from the wireless link between the airplane and the satellite. This link behavior depends on the assignment of transmitting sites to computer nodes in the model. For example, if the PDUs currently assigned to the ground station were assigned to computer node 1 onboard plane 0, the Ethernet link running at 100 Mbps will produce fewer collisions than a wireless link running at

64 Kbps due to the shorter transmission times. Table 1 shows statistics about the total number of collisions, percentage of all the PDUs, and average number of collisions per second.

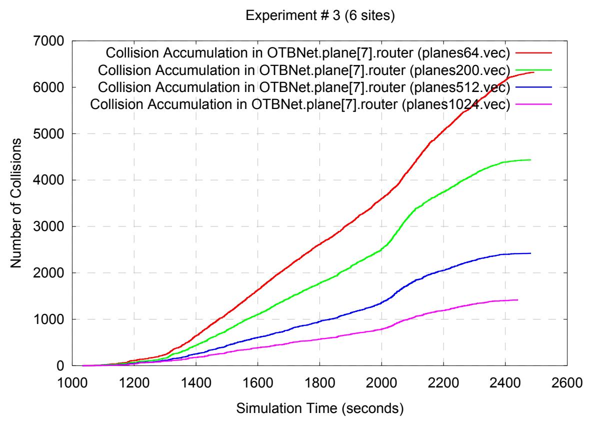

Figure 17. Collision Accumulation at plane 7 at 64, 200, 512 and 1024 Kbps.

Table 1. Collision Accumulation, percentage and average collisions per second at plane 7 for different bandwidths.

64KBps	6300 coll.	10 %	2.5 coll/sec
200KBps	4400 coll.	7.3 %	1.7 coll/sec
512KBps	2300 coll.	3.8 %	0.9 coll/sec
1024KBps	1300 coll.	2.1 %	0.5 coll/sec

CONCLUSIONS

As predicted by the static analysis, a bandwidth of 64 Kbps in the wireless links is insufficient to handle Embedded Training traffic under a DIS-like protocol. Latencies of more than 70 and 100 seconds, respectively, were detected for traffic from simulation stations on other planes and at the ground station where the Opposing Force simulated entities would be controlled. As predicted, a significant improvement was achieved at 200 Kbps, where latencies less than 1 second were almost always the case for messages received at the ground station.

At the router in plane 0 and at the satellite, the queue lengths changed from 3400 and 2200 (max. peak) to less than 60 and less than 40 (max. peak) messages, just by changing the bandwidth from 64 Kbps to 200 Kbps. As seen by the listener router at plane 7, collisions are manageable, and decrease correspondingly as the bandwidth increases.

With regards to the modeling tool used, OMNeT++ is versatile and straightforward to use for modeling computer networks. It offers the possibility of running the model at several speeds with or without animation, including faster-than-real time. The results can be displayed and plotted on the screen as the simulation progresses, and also can be stored into text files for further processing.

REFERENCES

IEEE Std 1278.1-1995. *IEEE Standard for Distributed Interactive Simulation — Application Protocols*. Retrieved June 01, 2003, from http://ieeexplore.ieee.org/xpl/tocresult.jsp?isNumber=10849

IEEE Std 1278.2-1995. *IEEE Standard for Distributed Interactive Simulation — Communication Services and Profiles*. Retrieved June 01, 2003, from http://ieeexplore.ieee.org/xpl/tocresult.jsp?isNumber=10842

IEEE Std 1278.3-1996. *IEEE Recommended Practice for Distributed Interactive Simulation — Exercise Management and Feedback*. Retrieved June 01, 2003, from http://ieeexplore.ieee.org/xpl/tocresult.jsp?isNumber=12808

IEEE Std 1278.1a-1998. *IEEE Standard for Distributed Interactive Simulation — Application Protocols*. Retrieved June 01, 2003, from http://ieeexplore.ieee.org/xpl/tocresult.jsp?isNumber=15722

IEEE Std 802.11-1997 *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications*. Retrieved June 01, 2003, from http://ieeexplore.ieee.org/xpl/tocresult.jsp?isNumber=14251

HyPerformix, (2003). Retrieved June 03, 2003, from http://www.hyperformix.com/products/workbench.htm

OMNeT, (2003). Retrieved June 03, 2003, from http://whale.hit.bme.hu/omnetpp/

Tcl Developer Xchange (2003). Retrieved June 03, 2003, from http://www.tcl.tk/.

Varga, Andras (2003) OMNeT++ *Discrete Event Simulation System, Version 2.3b1, User Manual*. Retrieved June 03, 2003, from http://whale.hit.bme.hu/omnetpp/manual/usman.htm