Network requirements evaluation of a multi-user virtual environment

Juan L. Font (*), Daniel Cascado, Jose L. Sevillano, Gema Lopez, Salvador Romero, Gabriel Jimenez.
ETS Ingenieria Informatica
Universidad de Sevilla
Avda. Reina Mercedes s/n. 41012, Sevilla (Spain)
(*) juanlu@atc.us.es

***Abstract*—In this paper, the network traffic requirements of a virtual world based on Wonderland are studied. Object synchronization and voice traffic are studied and several experimental measures are made in order to obtain an accurate model. A simulator based on ns-3 is developed that allows us to obtain some useful preliminary results about the networking resources needed to execute Wonderland. The simulation model used in this preliminary study can be used in future experiments to obtain useful results on scalability and needed network resources when the Wonderland Virtual World is used by a number of distributed users placed in different locations.**

Keywords-Virtual Worlds, Wonderland, Network Traffic, ns-3 simulations.

I. INTRODUCTION

Recently, a number of so-called "persuasive systems" have proliferated as a way to motivate users to acquire healthy lifestyle habits, perform physical exercises and so on [1]. These systems try to overcome the lack of motivation and isolation usually suffered by elderly people and/or people with chronic diseases, which is a very important concern given the demonstrated benefits of exercise and healthy habits. Our research group has developed a persuasive system based on the use of virtual worlds to motivate the user to perform physical exercises [2,3]. In this system, named "Virtual Valley", commercial devices are used to detect the user movements while performing the exercise (eg. Wiimote, Wii Balance Board) together with sensors for monitoring medical parameters during exercise (eg, oximeter). The user movements are incorporated into the virtual world allowing him/her to play games that encourage them to complete/continue the exercise program. Additionally, the social component of virtual worlds allows users to perform the exercise with their family or friends remotely connected to the virtual world. Doctors and/or carers can monitor and supervise the patient´s exercises and rehabilitation. Figure 1 shows a screenshot of Virtual Valley.

Virtual Valley is based on Wonderland, a Java open source software for creating collaborative 3-D virtual worlds (also known as Collaborative, Networked or Distributed Virtual Environments). It was originally conceived as a tool for collaborative working by Sun employees [4], and as such it has some characteristics that make it very interesting for our application: focuses on social interaction and communications; open platform that allows new developments; can be installed and used by organizations within their own infrastructure, without the cost of renting a virtual space on a third party server and also allowing control of private medical data; etc. [5].

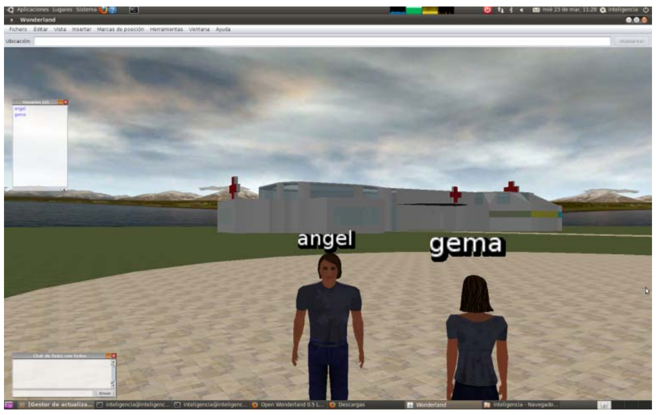

Figure 1. Screenshot of the Wonderland-based virtual world.

The latest version of this client-server architecture is Project Wonderland 0.5, which relies on the following main projects (see Figure 2) [6]:

- Darkstar: The server software architecture
- jVoiceBridge: Real-time immersive audio (via VoIP) with distance attenuation and a selectable range of qualities.
- jMonkeyEngine: Scene graph with features like render-to-texture, programmable shader, etc.

One of the most important features in Wonderland is the possibility of sharing applications among different users. Some of these applications are already integrated in Wonderland, like the multi-user PDF Viewer and the SVG White board. But users can also share additional external applications installed in the server (like Firefox or OpenOffice) using the Shared Applications Server-SAS.

In Wonderland, like in any other Virtual World, the user is represented by a 3D object known as an avatar. There can be many other objects in the virtual world, which can be 3-D objects like pieces of furniture, buildings, etc.; or 2-D objects like screens with applications (web browsers, word processors, and so on). The usual way to model the spatial relationships between objects is using a scene graph. Each object is a node (or *Cell* in Wonderland terminology) in this graph. The Cells (representing any volume of space of the virtual world) are organized in a graph with a tree hierarchy [7]. An example of such a tree is shown in Figure 3.

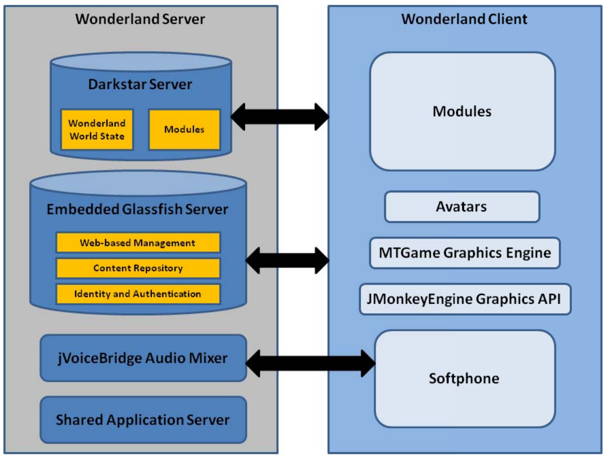

Figure 2. Wonderland Client-Server Architecture.

As we will discuss in the next section, the network traffic derived from Wonderland is mainly due to three sources. First, object synchronization which allows all users to have a coherent view of the virtual world (including moving objects like avatars). Second, messages intended to support communications among users, including voice traffic (the main source of traffic) but also text messages (chat). And, finally, traffic due to the execution of applications shared among different users. The latter is very difficult to model, as it depends on the particular application. Therefore, in this paper we will focus on the first two sources: object synchronization and voice traffic. Our aim is twofold: on the one hand, we try to perform a simulation and experimental study of a simple configuration that allows us to obtain some useful preliminary results about the networking resources needed to execute Wonderland. On the other hand, the simulation model used in this preliminary study will be used in future experiments to obtain some results on scalability and needed network resources when the Wonderland Virtual World is used by a number of distributed users placed in different locations.

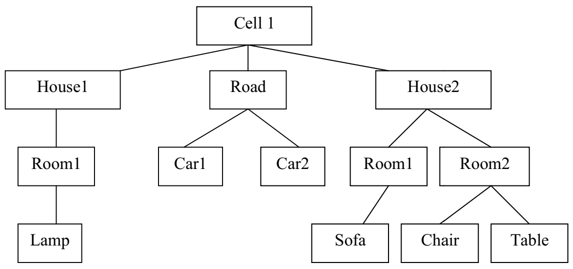

Figure 3. An example of a tree representing a Wonderland cell.

There are not many works in the literature dealing with the networking resources needed to support the execution of Networked Virtual Environments (NVE). A review of architectures and approaches to provide Quality of Service (QoS) for NVEs can be found in [8]. Some QoS experimental results are provided in [9]. Most of the available studies focus on multi-player online games. In [10] the effect of network latency with a high number of on-line players is studied. In [11] the network bandwidth requirements of some popular multi-player online games are experimentally measured by monitoring the network traffic generated by different game tournaments in a LAN Party. To the best of our knowledge, there are no published studies about the networking resources needed to support the execution of Virtual Worlds based on Wonderland. The results obtained in this paper will be useful when designing and implementing these systems, mainly in applications, like e-health systems, with specific characteristics like a minimum level of dependability, timeliness of some critical messages (e.g. alarms in case of falls, altered medical parameters, etc.), limited resources (bandwidth, etc.), and so on.

The rest of the paper is organized as follows: In the next section, the Wonderland v0.5 communications architecture is described, identifying the main traffic sources. Several experimental measures are made in order to confirm the traffic volume and to identify possible non documented sources. Then, in section III the ns-3 based simulator is described. Section IV shows some preliminary results and finally in section V we present our conclusions and future work.

II. Wonderland v0.5 Communications Architecture

Wonderland is based on a Client-Server architecture [12]. The Wonderland server must have a fixed, public IP address to which clients can connect. Initially, a Client is disconnected from the server until it calls a WonderlandSession.login(). If this succeeds, the session goes into the CONNECTED state. A client may connect multiple sessions to the same server. Once a session is connected to a server, connections may be added to it. Each connection in Wonderland has a unique type for sending different types of data. For example, a client may use one connection for sending cell data, and another for sending voice communications data. Clients may use as many connections as are necessary for their interaction with the server. The only limitation is that a client may only have a single connection of a given type connected to a given session. Clients may also use multiple sessions to get multiple copies of a single connection type.

Once in the CONNECTED state, messages can be passed from a client to the server, from the server to one client (not necessarily in response to a client message or request) and also the same message can be sent by the server to a number of clients. From version 0.5 on, there is no more client-client communication, that is, all messages from clients go directly to the server.

The traffic between clients and server/s can be broadly divided into the following categories:

A. Object Synchronization

Objects in the virtual world have a set of attributes that can have different values. For instance, coordinates in the virtual world, velocity (for moving objects), etc. The state of each object (or cell) is defined as the values of these attributes at a given time. In Wonderland, the server keeps a copy of all the world data, with data about the cells stored in xml files. When a client connects to Wonderland, it obtains from the server the information about the visible objects (cells). These objects can be classified as static (with fixed attributes that do not vary in time, as is the case of a mountain or a building) or dynamic (with attributes that can vary in time, like an avatar moving from one region to another) [13]. Every client should have a consistent view of the virtual world, and therefore a mechanism must exist to ensure synchronization between clients. When an object moves in the virtual word, all clients that want to view the dynamic object must provide the same sequence of state changes. In Wonderland, if anything changes (such as the position of an object), this data is updated on the server and then sent to all the clients, which then update their own local views of the world. For instance, if a client moves its avatar, the client notifies the server and sends the new state of the cell to the server. The server then sends the new state of this object to all other clients, which then update their copies of the cell [7]. A typical sequence is as follows: a client sends a MOVE_REQ message to the server when the position of an object changes. Then the server sends a MOVED message to every other client, one per client. An ACK message is then sent back to the server by every client. The server does not send any other MOVED message until the corresponding ACK has been received. This sequence has been confirmed by several experiments.

Although these MOVE_REQ object synchronization messages are sent every time the object moves, in principle they are limited to five updates per second by the MoveableComponent [14]. However, as we will see shortly, this limitation does not always work.

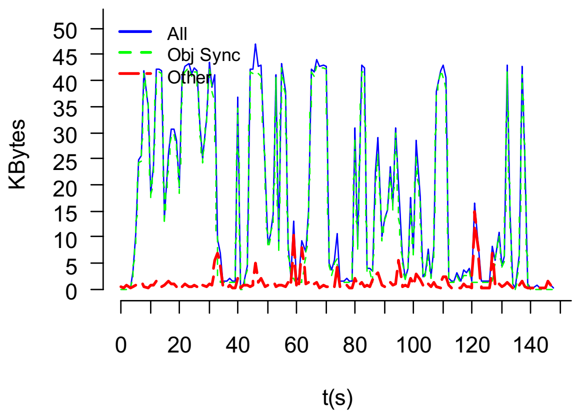

Figure 4. Outgoing and incoming TCP traffic (bytes/s) for one Wonderland client (GNU/Linux). Blue: Total Traffic; Green: Object Synchronization; Other Colours: Protocol Traffic.

Object synchronization messages are sent by default through port TCP 1139, and their length varies between 280-400 bytes, depending on the needed information about the object position, orientation, identification within the virtual world, etc. There are also other protocol messages (ack, presencemanager, audiomanager, etc.) that contribute very little to the TCP traffic. Although all this information can be get from the Wonderland documentation, it is not easy to model the actual workload in a real setting as it depends on what kind of movements or how often the avatar moves. There may also be some differences depending on the configuration of the client platform. That is why we performed several experiments to try to confirm the actual network workload using different Operating Systems and different number of clients. One important conclusion that can be drawn from our experiments is that when GNU/Linux is used on the client side, synchronization messages are *not* limited to five updates per second. For instance, Figure 4 shows a trace of Wonderland TCP traffic with only one client running on a Linux system. We asked the only user to continuously move her avatar in order to generate as many synchronization messages as possible. Both the client and the server run on the same machine, so that what we see is the traffic due only to Wonderland operation. At this stage, we were not interested in any other issue but to model the actual network workload. It can be seen that the vast majority of TCP traffic is due to object synchronization messages (movement requests and/or confirmation), while other messages (presencemanager, audiomanager, ack, etc.) contribute very little, as expected. The peak traffic is about 10 times higher than would be expected if the limit of 5 updates per second were observed. This fact was confirmed in the Wonderland forum, and seems to be due to an issue related with key repeat in Linux [14]. If the same configuration is measured on a Windows-based client, the 5 updates per second limit works, providing an upper-bound on the mean object synchronization traffic. Figure 5 compares the TCP traffic generated by one Wonderland client in these two cases: one GNU/Linux based client and one Windows-based client. The results are shown in bytes per second.

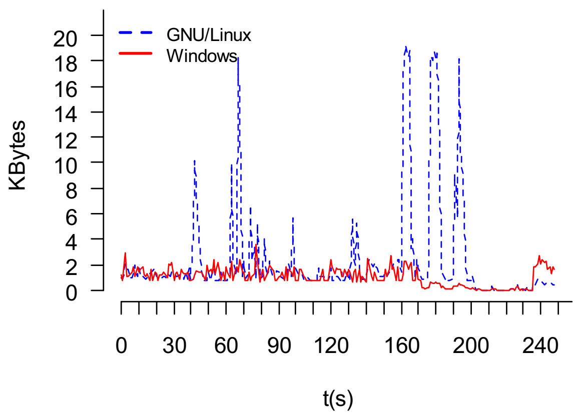

Figure 5. Outgoing TCP traffic by one Wonderland client with different operating systems (bytes/s)

It can be seen that the difference in generated traffic is dramatic. Of course, these data are obtained for a user generating as many object synchronization messages as possible, moving her avatar as much as possible. But it is apparent how important it is to guarantee the limit of 5 updates per second in a real Wonderland deployment, as it will avoid saturation when a high number of users simultaneously move in the virtual world. Note that allowing users to interact with other users is more important in this kind of systems than to quickly respond to their movements. In other words, in a virtual world system performance is perceived not only as system latency, but also as system throughput (defined as the maximum number of users that the system can simultaneously support) [15].

B. *Voice*

In Wonderland, the standard conversation allows users to speak and hear each other depending on the distance between them. But in addition, users can also initiate a voice chat session with other users. This conversation can be private but can also be made public. The voice functionality can be extended depending on the application. For instance, in [16] a system for virtual meetings is described, where voice communications between users are supported even if a user is not present in the virtual world (using a Virtual Phone for real-time communication). In our system, we currently use the standard Wonderland voice functionality offered by jVoiceBridge, which uses the standard SIP (Session Initiation Protocol) and RTP (Real-time Transport Protocol) protocols to transmit voice data from the Wonderland server to the various clients. jVoiceBridge uses a single UDP port for all control data, and an additional two UDP ports per call connected. The UDP control port is used by SIP, and by default is UDP port 5060 [17].

In Wonderland, a *conference* has members which can talk to each other. *Calls* are the individual parts of a conference, and data from all calls in the conference are added into a "common mix". Data samples from every member are added together to create the output from the server to the clients. Each client's audio must be subtracted out of the common mix when the data is sent to that client, so that the user does not hear him/herself when he/she talks [18]. Therefore, the traffic from the server to the clients has to be sent as different streams. As a result, the voice traffic increases linearly as the number of members in a conference increase.

jVoiceBridge handles audio at 3 different fidelities:

- 8k ulaw, 8 bits per sample, 8000 samples per second. This means 64kbits/s.
- 16k pcm, 16 bits per sample, 16000 samples per second. This means 256kbits/s.
- 44.1k pcm, 16 bits per sample, 44100 samples per second. This means 705600kbits/s.

The voice system sends packets every 20ms, or 50 times per second. In addition, the client may send/receive a mono or stereo stream to/from the voice bridge. For example:

- send 256 kbit/s = 32 kbyte/s = 50 x 640 bytes/packet
- receive stereo 256 kbit/s = 64 kbyte/s= 50 x 1280 bytes/packet

Finally, packet headers should be included which add up to about 40 bytes per packet. Our experimental observations confirm these assumptions: the traffic due to one client sending messages to the server would be approximately constant and equal to 64kbytes per second, assuming stereo streams. The following experiment illustrates the voice traffic in Wonderland. Figure 6 shows the traffic generated by three clients: the first one receives a voice traffic, the second one transmits a voice traffic and the third one begins receiving a voice stream when its avatar come closer to the other clients' avatars. Figure 7 shows the traffic in the server. Note that all these events imply a proportional increase in voice traffic, with the server assuming the sum of all this traffic as it has to mix-up and resend all the voice streams to every client.

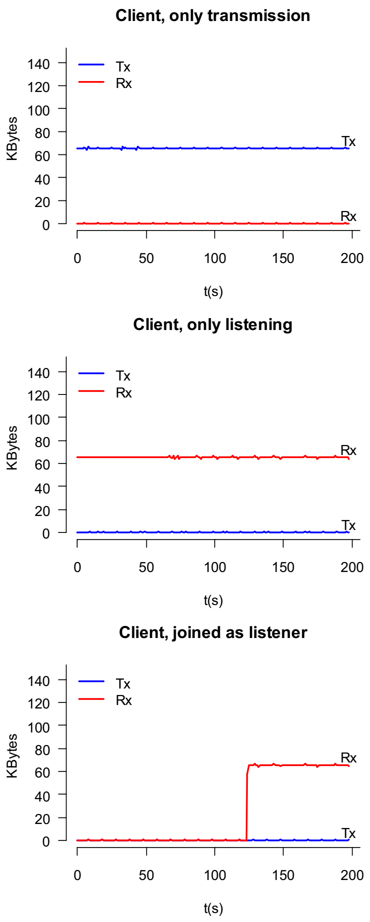

Figure 6. Voice traffic generated by several clients (Kbytes/s), in both directions.

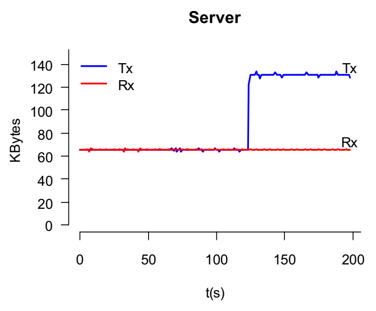

Figure 7. Voice traffic at the server (Kbytes/s).

III. SIMULATION MODEL

In order to evaluate the performance of the Wonderland communications architecture, a simulation model of Wonderland communications has been developed which is based on ns-3 [19], a discrete-even simulator conceived as ns-2's successor. Ns-3 and ns-2 share common background and concepts, but ns-3 is a new project that tries to solve or mitigate many of ns-2's well-known drawbacks as well as to apply new concepts, such as validation and software engineering techniques. For example, it has dropped ns-2's dual language design, and it emphasizes both code and model structuring as well as encouraging software engineering practices to improve code and documentation maintenance. A new architectural approach characterized by modularity, source code reuse and software pattern application is used [20].

Other interesting characteristics of ns-3 are:

- Internal project policies that encourage code correction, cleanliness and exhaustive documentation, which make understanding, extending and working with ns-3 a much easier task, in spite of being such a big and complex project.
- Open source licenses such as GPL make ns-3 very convenient for developers and users who have a considerable freedom to use and adapt it to their needs, being able to run it in different context such as research or industrial environments. The open source focus is especially valuable from researches' point of view due to source code availability, which allows them to freely study and modify it.
- The own project organization as world-wide distributed effort opens a great opportunity for new developers to join and contribute with their own code, patches and bug fixes. This is an important added value for the ns-3 tool itself: users can have a powerful simulation framework plus a wide range of third-party tools and extensions that will fulfill their needs.

An ns-3 simulation divides a network model into several main entities: communication channels, network devices, protocol stacks, nodes and applications [21]:

- Channel models abstract different physical communication channel types and their characteristics: point to point links, CSMA buses, wireless channels, underwater environments, etc. These models are complemented with auxiliary models that describe different physical channel behavior: path loss, propagation, noise, etc.
- Network device models simulate the functionality of the network hardware that makes physical channel access possible. The network devices provide the rest of the entities in the simulation with access to the communication channel. Ns-3 includes classes for CSMA network devices, WIFI and WIMAX among other communication technologies.
- Protocol stack models abstract the functionality of real protocol stacks, implementing their APIs. These protocols give communication services to the applications and use the network devices to send and receive data through a communication channel. IPv4 and IPv6 are examples of protocol stacks modeled and included in ns-3 simulator by default.
- A node model is the equivalent to a computer or device connected to a network. In ns-3 the mission of the Node entity is to be the container of both hardware models (network devices) and software models (applications, protocol stacks). They are connected to each other by a communication channel and they need network devices to be able to access that channel. Their applications and protocols use these network devices to obtain communication services.
- Application modes are abstractions of the real software applications deployed on the network. These entities act as packet generators or consumers. They use the rest of entities to send or receive data through the simulated network.

Regarding the simulation of our Wonderland prototype, the model is articulated by the simulation file alice.cc, written in C++ and located in the special ns-3 folder "scratch". This file makes use of several modules, auxiliary classes and services provided by ns-3. Besides some default classes, there are several new custom classes developed as part of this work to model both audio and object synchronization traffic between Wonderland clients and servers.

In this preliminary study, the network is assumed to be based on Ethernet technology, so CSMA channel and network devices are used. The first simulation prototype defines a common CSMA bus which interconnects two nodes, a Wonderland client and server respectively. We begin with this basic set-up as we are interested in modeling the traffic and in comparing the simulation with the real traces described in the previous section.

Each node has a CSMA device and a IPv4 protocol stack. The ns-3 helpers automate the IP address assignment process given a certain IP range.

The client node acts as container of both audio and object synchronization applications which share its resources and CSMA network device. The server node contains the corresponding audio and server applications.

The simulation results are stored as traffic dump using the ns-3 tracing system which can generate several output file formats. The simulation generates its traces in PCAP format, which can be read by Wireshark among other common traffic analysis tools. This way simulation and experimental traces can be easily compared.

A. Audio client application

This class derives from ns3::Application which describes the basic common methods and attributes of an ns-3 application, a simulation entity devised for generating and consuming network traffic.

The "udp-echo-client" application has been used as template due to their similar basic functionality. The original udp-echo-client only sends a specific amount of packets, while the WonderAudioClient can stop when it reaches a certain number of sent packets, or it can send information until it reaches the stop time defined in the simulation. Any of the above modes can be chosen depending on our needs.

Like udp-echo-client, WonderAudioclient uses UDP sockets to periodically send packets to its associated Audio Server. The programmer defines the audio packet size attending to Wonderland specifications. Both 8bits and 16K x 16bits modes audio streams produce packets which don't require fragmentation and fit into a single CSMA packet with a MTU of 1500 bytes. For the 3rd and higher quality, 41k x 16bits, a basic mechanism has been implemented to avoid ns-3 lack of packet fragmentation. The total audio payload is split between several consecutive packets. There is no sequence control mechanism in the server as long as the main purpose of the simulation is to generate network traffic similar to the real modelled system.

The proposed audio client model uses the UDP port 5060 for all the audio transmissions, unlike the real Wonderland clients which use this port for audio protocol purposes and as many pairs of UDP ports as needed from the 10000 to 10200 range. The audio client automatically sends UDP packets every fixed time period, set by default to 20 ns as described in Wonderland technical documentation [18].

The other audio client function is receiving audio traffic generated by other Wonderland clients and forwarded by the audio server. No extra protocol packets or ACKs have been used in this model.

B. Audio client application

Analogously to the audio client, the WonderAudioServer derives from the ns3::Application class and uses udp-echo-server as template to build up its own functionality.

The WonderAudioServer receives all the UDP traffic from the 5060 port. It also manages an IP address list of all the nodes that have sent UDP packets to the server. When a new UPD packet is received, the server extracts the sender IP address and checks it against its client list. If the address is not in the list, it is added to it.

The list is used later to know the destination of the forwarded audio packets. The server sends a copy of each forwarded message to each node, instead of using multicast mechanism.

C. Object synchronization client.

WonderMovClient class, the Wonderland object synchronization client, shares many similarities with the audio client and their source code share a common base. The main difference is that object synchronization uses TCP protocol to send packets. The configuration of the TCP sockets and IPv4 protocol were extrapolated from the ns-3 packet-sink application instead of udp-echo-client.

All the TCP traffic flows through the 1139 TCP port which is a simplification of the real Wonderland behavior. The object synchronization client also generates TCP traffic periodically, but in this case both the packet size and the sending interval depend on probability distributions. The packet size is calculated from a Uniform distribution which generates values between 280 and 400, the packet sizes that we experimentally measured for Wonderland. As for the time between object synchronization messages, they are sent every time the object moves, and it is very difficult to model the actual movements made by the user through the keyboard, joystick or the like. Therefore, we model this traffic in our simulator using for the time between synchronization messages an exponential distribution. This simple solution is in principle a good way to model the time between this kind of random and un-correlated events caused by a number of independent factors. The mean time between messages a is used as a parameter that allows us to mimic the actual mean time between synchronization messages. A limit of five updates per second is used, with the exponential distribution modeling the randomness of the user movements. Ns-3 provides the classes ns3::Uniformvariable and ns3::ExponentialVariabe classes respectively to model the behavior of these probability distributions.

The object synchronization client sends object position updates to the server, and it receives as well object synchronization updates from the server, which are originated by other clients. The client sends a 66 bytes ACK packet to the server for each received update.

D. Object synchronization server.

The WonderMovServer class is a redefinition of the WonderAudioServer, adapting it to the object synchronization traffic which includes the use of TCP protocol and ACK packets to confirm the reception of updates.

Like its audio counterpart, the server listens the incoming TCP connections from the port 1139, and also maintains an IP address list of all the simulated Wonderland clients which have sent at least one packet. The server also forwards a copy of each update packets to the rest of clients to keep their virtual world instances synchronized, and waits for the corresponding ACKs of the clients.

IV. SIMULATION RESULTS

In this section, we present some preliminary results that show that our simulator is able to mimic the behavior of a real Wonderland setting, therefore becoming a useful tool to study different configurations. First, we focus on the results for object synchronization messages as this traffic is the more difficult to model. From the previous sections we know that Wonderland clients limit object synchronization updates to 5 per second. This reduces the amount of traffic avoiding saturation when a high number of users simultaneously move in the virtual world. GNU/Linux Wonderland clients cannot meet this constraint and as a result the generated traffic shoots up. Of course this fact can be considered a bug so the main focus should be on the "regular" case with up to 5 updates per second.

The traffic has been modeled using an exponential distribution with a=0.4 and it has been truncated to avoid values lower than 0.2 (that would break the restriction of maximum 5 updates per second). A mean value of 0.4s performs well when comparing the simulated results with the real ones, as shown in Figure 8. Note that the results are similar to the experimental results shown in Figure 4, at least the peak values which are the most interesting ones.

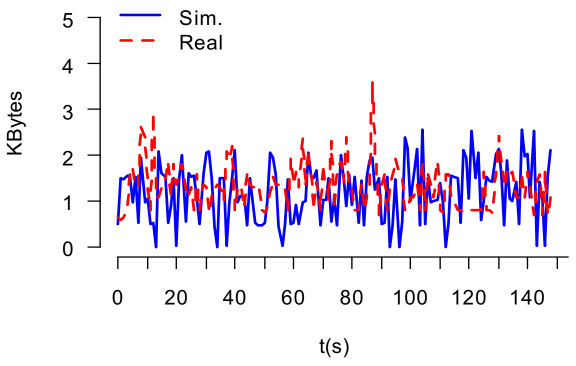

Figure 8. Simulated Object Synchronization traffic (bytes/s) for one Wonderland client (with the 5 updates/s limit).

Finally, the simulator also allows us to study several interesting configurations. For instance, it allows us to study how the traffic scales as the number of clients increases. Figures 10 and 11 show the total UDP (audio) and TCP (object synchronization) traffic at the server for the case of 10 and 20 clients, respectively. In the left side of these figures the number of clients gradually increases, showing how the traffic increases accordingly. The results for the audio traffic show a linear increase with the number of clients N, as expected. Clearly, in situations with scarce bandwidth resources clients may switch the audio off to reduce this traffic source. On the other hand, the object synchronization traffic increases with N^2. This is also an expected result. When the position of an object changes, for every MOVE_REQ message sent by the client to the server, the server sends a MOVED message to every other clients. Ignoring other messages like ACK, if P is the mean packet length in bytes ($P \approx 400$*bytes*), then given the restriction of 5 updates per second the mean traffic would be approximately (in bytes per second): $N^2 * 5 * P$, where N corresponds to the number of clients and P the size in bytes of the packet. . Therefore, as the number of clients increases the object synchronization traffic may reach values similar to those of the audio traffic. It can be imagined what would happen when the number of GNU/Linux clients increase which do not limit the number or object synchronization messages.

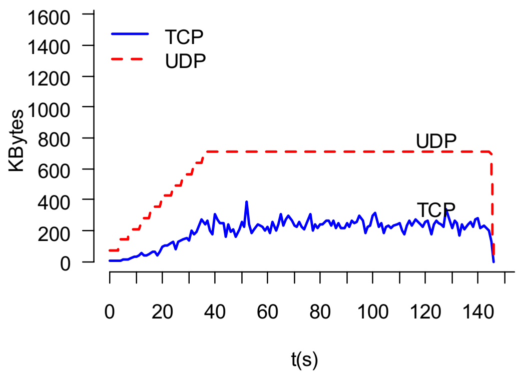

Figure 9. Simulated total UDP (audio) and TCP (object synchronization) traffic for the case of 10 clients (bytes/s).

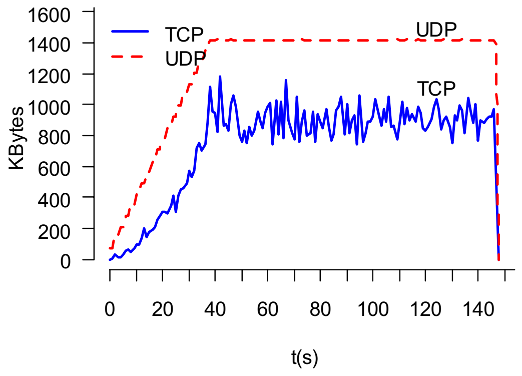

Figure 10. Simulated total UDP (audio) and TCP (object synchronization) traffic for the case of 20 clients (bytes/s).

V. CONCLUSIONS

In this paper, we try to model two of the main network traffic sources in a Wonderland-based networked virtual world. First, object synchronization which allows all users to have a coherent view of the virtual world (including moving objects like avatars), and second voice traffic intended to support communications among users. Some experimental measures were made in order to correctly model this traffic. Some interesting conclusions were drawn from this experimental study, which were not described in the Wonderland documentation. Particularly, the fact that GNU/Linux clients do not meet the limit of five updates per second imposed to object synchronization traffic from the clients.

The obtained traffic model is used as input of an ns-3 based simulation model which can be used to study different Wonderland configurations. Some preliminary simulation results are presented. Future work will make extensive use of this simulator with different communications links to study issues like scalability, latency, bandwidth, etc.

ACKNOWLEDGMENTS

This work has been partially supported by contract Vulcano: TEC2009-10639-C04-02 as well as by the Telefonica Chair "Intelligence in Networks" of the University of Sevilla

(Spain). We would also like to thank the people from the Open Wonderland Forum who answered some of our questions.

REFERENCES

[1] Fogg, BJ.; "Persuasive Technology: using computers to change what we think and do," Morgan Kaufmann Series in interative technologies, ISBN 1-55860-643-2, Morgan Kaufmann 2003.

[2] S. J. Romero, et al., Open source virtual worlds and low cost sensors for physical rehab of patients with chronic diseases. P. Kostkova (Ed.): eHealth 2009, LNICST 27, pp. 84–87, 2010.

[3] Cascado, D.; Romero, S.J.; Hors, S.; Brasero, A.; Fernandez-Luque, L.; Sevillano, J.L.; , "Virtual worlds to enhance Ambient-Assisted Living," Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE, Aug. 31 2010-Sept. 4 2010

[4] N. Yankelovich et al. Meeting central: making distributed meetings more effective. Proceedings of the 2004 ACM conference on Computer supported cooperative work. Pp. 419-428. Chicago, IL (USA), Nov. 2004.

[5] Gardner M, Sheaffer W (2009). Designing and building immersive education spaces using Project Wonderland: from pedagogy through to practice. University of Oregon, Immersive Education Days, 18th – 20th August 2009.

[6] Open Wonderland: http://www.openwonderland.org/ Accessed 15/03/2011

[7] Wonderland Tutorial. Available at http://www.openwonderland.org/

[8] Denis Gracanin, Yunxian Zhou, and Luiz A. DaSilva. Quality of Service for Networked Virtual Environments. IEEE Communications Magazine. Pp. 42-48. April 2004

[9] T. Henderson and S. Bhatti, "Networked Games: A QoS-Sensitive Application for QoS-Insensitive Users," Proc. ACM Int'l Conf. Applications, Technologies, Architectures, and Protocols for Computer Comm. (SIGCOMM '03), pp. 141-147, 2003.

[10] T. Fritsch, H. Ritter, J. Schiller. The Effect of Latency and Network Limitations on MMORPGs. NetGames'05, October 10.11, 2005, Hawthorne, New York, USA.

[11] Enrique Asensio, Juan M. Orduña, Pedro Morillo Analyzing the Network Traffic Requirements of Multiplayer Online Games. The Second International Conference on Advanced Engineering Computing and Applications in Sciences. Valencia (Spain), Sept. 2008.

[12] Wonderland Communications Architecture. Available at http://www.openwonderland.org/

[13] J.C.S. Lui, Constructing Communication Subgraphs and Deriving an Optimal Synchronization Interval for Distributed Virtual Environment Systems. IEEE Transactions on Knowledge and Data Engineering, Vol. 13, No. 5, pp. 778-792. September/October 2001

[14] Open Wonderland Forum. http://groups.google.com/group/openwonderland

[15] P. Morillo, S. Rueda, J.M. Orduña, and J. Duato "A Latency-Aware Partitioning Method for Distributed Virtual Environment Systems". IEEE Transactions on Parallel and Distributed Systems, vol. 18, no. 9, pp. 1215-1226. September 2007.

[16] Mirza Hadžić. Supporting Distributed Software Teams with 3D Virtual Worlds. Master's Thesis. Institute for Information Systems and Computer Media (IICM), Graz University of Technology (Austria), 2010.

[17] Firewall Configuration. Available at http://www.openwonderland.org/

[18] jVoiceBridge Developer Documentation. Available at http://www.openwonderland.org/

[19] Ns-3 overview. http://www.nsnam.org/docs/ns-3-overview.pdf, August 2010.

[20] J.L. Font et al., Analysis of source code metrics from ns-2 and ns-3 network simulators, Simulat. Modell. Pract. Theory 19 (2011), pp. 1330-1346.

[21] Ns-3 Manual: 4.1 Object Model. http://www.nsnam.org/docs/release/manual.html#Objectmodel