In Proceedings of the IASTED Telehealth 2005, Banff, Canada, June 19-21, 2005.

GiPSiNet: A MIDDLEWARE FOR NETWORKED SURGICAL SIMULATIONS *

Vincenzo Liberatore
Division of Computer Science
Case Western Reserve University
10900 Euclid Avenue, Cleveland, Ohio 44106
United States
vincenzo.liberatore@case.edu

M. Cenk Çavuşoğlu
Division of Computer Science
Case Western Reserve University
10900 Euclid Avenue, Cleveland, Ohio 44106
United States
cavusoglu@case.edu

Qingbo Cai
Division of Computer Science
Case Western Reserve University
10900 Euclid Avenue, Cleveland, Ohio 44106
United States
qingbo.cai@case.edu

Abstract

Virtual environments are a promising new medium among the simulation methods for surgical education. The network extension of surgical virtual environments will enable continuing education and advanced training over wide geographical areas. However, the network settings also pose several non-trivial problems in terms of bandwidth limits, delays, packet losses, *etc*. for distributed surgical virtual simulations.

In our previous work [6], we presented *GiPSi* (General Interactive Physical Simulation Interface), an open source/open architecture framework for developing surgical simulations. GiPSi works on individual workstations and, in our ongoing development, we extend GiPSi to a network environment. This network extension involves the development of a middleware module (*GiPSiNet*) to remediate for the lack of network *QoS* (Quality of Service) and to enhance the user-perceived quality (fidelity and realism) of a networked simulation. In this paper, we introduce the design of the GiPSiNet middleware and describe the techniques to provide timely data delivery over the network. We also provide the evaluation measures for the performance of user-perceived simulation quality in the presence of network dynamics.

Keywords
Surgical simulation, Virtual reality, Network, Middleware.

1 Introduction

Medical education can be improved through the use of simulations whenever possible [1]. Among the various simulation methods for surgical education, virtual environments are a promising new medium. With virtual environments, a user can perform surgery on a simulated patient by manipulating simulated organs using simulated surgical instruments (haptic devices). An example of a surgical simulator is shown in Figure 1.

The accessibility of surgical virtual environments can be substantially extended by network communications. The resulting networked simulations will enable continuing education and advanced training over wide geographical areas. However, the network settings also pose several non-trivial problems for distributed surgical virtual environments because the physical communication platform introduces constraints in terms of delays and bandwidth. Moreover, haptic data traffic cannot expect any special handling within a best-effort network such as the Internet, and consequently can be subject to delays or packet losses due to congestion. Since the simulator should sense a user's input (e.g., the position of a haptic device) and respond (e.g., with haptic force feedback) to the user in real-time, the issues such as the network delays can greatly impair the user-perceived simulation performance. Thus, the quality of a networked surgical simulation critically depends on methods to enable an effective remote interaction.

In our previous work [6], we presented *GiPSi* (General Interactive Physical Simulation Interface), an open source/open architecture framework for developing surgical simulations. GiPSi works on individual workstations and, to enable it to operate in a network environment, we

*URL: `http://vorlon.cwru.edu/~vxl11/`. The presentation was made possible in part by School of Graduate Studies at Case Western Reserve University.

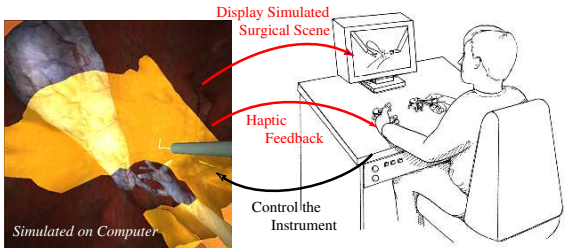

Figure 1: Surgical simulator concept. Simulation snapshot courtesy of F. Tendick [14].

have designed an open source and open architecture middleware module *GiPSiNet*. GiPSiNet acts as an intermediary between applications and the network and takes actions to remediate for the lack of network *QoS* (Quality of Service) [13]. In our ongoing project, we develop, deploy, and evaluate the GiPSiNet middleware for the distributed surgical virtual simulations.

Specifically, the objectives of the GiPSiNet middleware are to address the following issues:

- Abstraction for heterogeneous haptic devices and data representations.
- Modularity through encapsulation and data hiding.
- Customizability to accommodate diverse surgical virtual simulations.
- Efficient utilization of network bandwidth and other system resources. GiPSiNet should be lightweight and efficient for data communication via techniques such as data compression.
- Compensation strategies for networks with delay, jitter, and congestion. A main objective of GiPSiNet is to enable high quality (fidelity and realism) of remote simulation in the absence of network QoS.
- Qualitative and quantitative measures for a comprehensive performance evaluation.

Organization. In Section 2, we describe the architectural design and modules of GiPSiNet. Section 3 introduces the techniques that GiPSiNet adopts to ensure timely data delivery over the network. In Section 4, some measures are given for the performance evaluation of GiPSiNet. In Section 5, we introduce the related work. Section 6 concludes this paper.

2 Architectural Design of GiPSiNet

GiPSi [6] is an open source/open architecture framework for developing organ level surgical simulations, and works on individual machines. To enable GiPSi to operate in a network environment, we develop the GiPSiNet middleware. A technical objective is to develop an organization of the software and enforce a precise abstraction of the communication flow.

The GiPSi framework currently provides an API between the simulation kernel and I/O (haptic I/O, visualization). In order to extend the kernel-I/O API to a network environment, a middleware layer (GiPSiNet) is added between the kernel and the I/O module; such a middleware encapsulates all networking functionality (Figure 2). The resulting simulator involves two communication endpoints: (1) the *client*, which interacts with the end-user (e.g., surgeon, trainee) through the haptic I/O and visualization interfaces, and (2) the *server*, where the simulation kernel runs and the physical models are numerically simulated. The middleware transparently incorporates the complex adaptive methods that are described in the rest of the paper. Moreover, the networked framework inherits from GiPSi its flexibility and potential for extensibility to a variety of surgical simulations.

2.1 Communication Model

The development of a middleware necessitates a precise agreement between the client side and the server side. GiPSiNet uses the following *contract* between the client and the server. The client sends the server: (1) an input x_i, which represents state derived from user actions, such as a scalpel position, and (2) an integration interval Δt_i. The server replies to the client with a linearized approximation V_{i+1}, which is the system state after the integration interval Δt_i assuming that the input x_i is applied continuously during such interval. Figure 3 shows the canonical way for the client to exploit its contract with the server: the client sends $(x_i, \Delta t_i)$ to the server, which replies to the client after a computation interval with the new state V_{i+1}. The communication is structured around the concept of *data unit*, which is the basic information block exchanged between the two communication end points. The two data unit types in GiPSiNet represent the information flowing from the client to the server $(x_i, \Delta t_i)$ and from the server to the client (V_{i+1}). In these terms, the information exchange follows a classical on-demand protocol: the client sends a data unit (request) to the server, which replies with another data unit (response). In an ideal scenario, the client sends its requests at regular intervals of length Δt_i (say, 100ms, as in the non-networked GiPSi simulator), and the server reply is delivered before Δt_i. GiPSi views each data unit as a foundational building block for its operations. However, each "unit" requires several operations, e.g., segmentation and compression, to be executed in the GiPSiNet middleware.

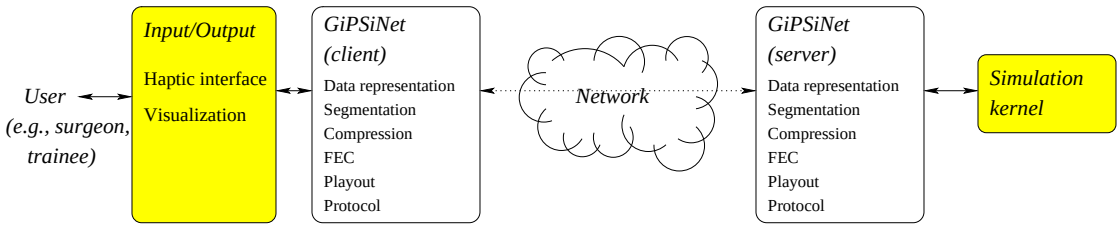

Figure 2: The GiPSiNet software architecture. Shaded module are part of the existing GiPSi platform, clear modules are part of the GiPSiNet middleware.

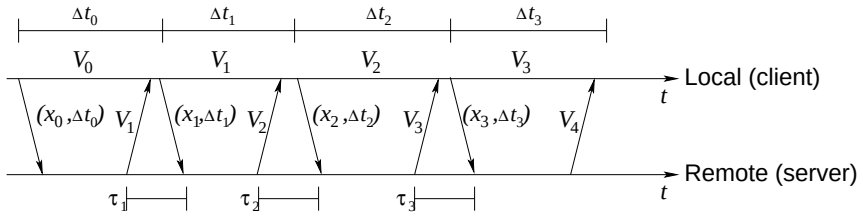

Figure 3: Timeline of data unit exchanges between client and server.

2.2 GiPSiNet Modules

Data representation module. This module creates, implements, and documents a format to represent the data exchange implied by the kernel-I/O API.

Protocol module. In the protocol module, a light-weight protocol is implemented to establish a connection between client and server. The emphasis is on light-weight methods that highlight the network effect of various software design choices. However, more complex solutions, such as SOAP, CORBA, HLA, or RTP framing can be incorporated after the interaction of network and simulation is better understood. The protocol is based on a connectionless transport (UDP), which is better suited to real-time traffic.

Segmentation module. The data size of each server data unit (the linearized model V_{i+1}) is currently of the order of 11,664 bytes. The client data unit (the local state x_i) is much smaller and of the order of 12 bytes. Meanwhile, Internet traffic is based on packets whose size cannot exceed certain bounds, typically 1500 bytes (the Ethernet Maximum Transmission Unit), including UDP/IP headers. If a data unit is longer than the given bound, it is divided up among multiple packets (*segmentation*). Data unit segmentation is made necessary by the datagram nature of the UDP protocol and by the need for fast, light-weight communication that is controllable by and interacts with the simulator. In a best-effort network, each individual packet can be delivered out-of-order or lost. Therefore, the middleware and the associated protocol implement *segmentation handling* to patch individual packets together into a simulation data unit (or to detect that a data unit was lost) by keeping an appropriate data structure of packet sequence numbers.

Compression module. GiPSiNet can benefit from a short data unit: short data exchanges require fewer packets, which in turn decreases the overhead of keeping track of multiple segments. Shorter data units also decrease the loss probability. Furthermore, a short data unit has a shorter transmission time given the same amount of end-to-end available bandwidth. Data unit size can be reduced by resorting to compression. A data compressor is implemented in the GiPSiNet middleware immediately upstream of the network code. The data compression algorithm is based on the standard Burrows-Wheeler algorithm [2], which is simple, light-weight, and leads to better compression rates than other widespread techniques. Moreover, the algorithm can compress each data unit independently of others, which renders compression resilient to losses.

Section 3 introduces the GiPSiNet modules that ensure timely data delivery.

3 Timely Data Delivery

Most of today's networks are based on the best-effort model, which provides low level of QoS. This leads to the dropping or loss of data units, and consequently the degradation of the simulation realism and effectiveness. An approach to enhance the user experience is to add appropriate modules within the GiPSiNet middleware, without altering the remaining modules.

Data Exchange: Timeline. The ideal timeline schema operation is such that, when the client sends a data unit, at time $i\Delta t_i = i\delta$ (where $\Delta t_i = \delta$ is a fixed time quantity) to the server, the client simultaneously sets the simulation state to V_i. The client is then anticipated to receive the next state V_{i+1}, before the next sampling instance $(i+1)\delta$. However, the segments forming the V_{i+1} state could be lost or delayed, which in turn would cause the V_{i+1} state to be late or not forthcoming. In some instances the client's data unit could itself be lost and the client may end up waiting for V_{i+1} from the server. Hence, this timeline schema is based on ideal assumptions that do not hold in best-effort

networks. The modules to be added to the middleware endeavor to diminish the incidence of late or missing of data units.

Retransmission. There are two categories of data losses: the client data unit loss, and the server data unit loss. As a result, the client can fail to receive the next state information V_{i+1} from the server. In order to avoid this, the client would resend the input information to the server at regular intervals of length δ, irrespective of whether the server has received the data or not. Thus, regular retransmission by the client can support timely data delivery to some extent.

Forward Error Correction (FEC). The probability of loss of an n-packet data unit increases with the value of n. A general way to maintain low loss rate is to keep the value of n as small as possible, but a complementary method is to encode the data unit redundantly with FEC methods. Although FEC does not eliminate packet losses, it can nonetheless significantly reduce the probability that a longer data unit is lost, and therefore is included in the GiPSiNet middleware. Using these methods, the n-segment data unit is encoded into $m > n$ segment, such that the reception of n segments out of m would enable the reconstruction of the data unit.

Playout. The ideal time interval within which the client receives the server's reply is δ, which can differ from the actual time lag Δt_i. If $\Delta t_i < \delta$, the data unit can be buffered and used at the scheduled time, but a more severe problem occurs when the server reply is late. One way to deal with this problem is to have a large value assigned to δ, such that most data units can be received on time. However, δ cannot be too large, since it would effect the performance and the realism of the simulation. Therefore, the appropriate way is to have the value of δ depend on the Round Trip Time (RTT) and the server computation time, and should change over time to reflect changing network conditions. The GiPSiNet middleware contains a module for adaptive playout that dynamically adjusts the value of δ. The playout module borrows concepts from the Voice-over-IP (VoIP) literature and, in particular, it uses Algorithm 4 from [12], which appears to be effective, yet simple and robust. Then, the client includes the new value of δ in its request to the server, because the value of δ is dynamically changing and moreover δ is equivalent to the integration step.

4 Evaluation

A broad objective of the evaluation is to ascertain the effectiveness of the methods described in this paper and to improve upon them in critical scenarios. The first step in this direction is to develop a framework for the semi-automatic evaluation of GiPSiNet in the context of surgical simulations. The framework is used to evaluate the effectiveness of surgical simulations over various network configurations and then to improve simulation performance accordingly. It consists of hardware, software, and quantitative and qualitative results that enable the performance evaluation of GiPSiNet. GiPSi will be run on a real and emulated networks with a set of benchmark tasks and to introduce causes for deviation from the ideal behavior, such as network-induced delays, packet losses, or congestion. The quality of the software should also be evaluated from the user's perspective, for example, by expert reviews. We next describe several aspects of the evaluation details.

Benchmark Suite. In measuring performance, the first hurdle is to bridge the two worlds of network dynamics and user-perceived quality. Furthermore, the linkage should be as automated as possible so that different software design choices can be prototyped and tried in relatively short time frame. Network behaviors are related to simulation performance by executing a standard task remotely and measuring the performance on that task. The task performance can depend on network conditions and, in certain circumstances, it can even become impossible (e.g., in case of a network partition). Thus, the performance on a benchmark task gives an indirect but significant indication of the impact of network conditions on user-perceived performance. However, performance can be measured with an automatic procedure. Specifically, an automatic controller can be implemented for executing each task. For example, a state-of-the art automatic controller moves the scalpel using compliant control to perform the Fitts' task [4, 5]. The controller performance will be observed under different network configurations.

An important step in the evaluation to build and document a benchmark suite of standard surgical tasks on which performance can be evaluated. The benchmark tasks should be simple enough such that they can be programmed with control techniques of general knowledge. Meanwhile, the benchmark tasks should be representative of simple operations that a human user could do, and should enable the validation of the simulator under various network conditions. The standard Fitts' task will be used as a benchmark task, and task performance will be quantified using the Fitts' index of performance. In addition, the evaluation adopts benchmark tasks that are more general and represent other common tasks performed in typical surgery, including tasks that require tracking of simple trajectories using surgical tools and maintaining constant contact force between the surgical tool and tissue. In addition, the GiPSiNet evaluation can adopt suitable performance metrics, such as task completion time and error in trajectory tracking, and the amount of simulated damage that a controller inflicts under poor network conditions for constant force application tasks.

GiPSiNet Instrumentation. The user can perceive different levels of simulation realism depending on the computational capabilities of his own computer and on the overhead imposed by running GiPSiNet on his machine. An evaluation objective is to assess the computational requirements of the individual modules within GiPSiNet. To estimate

the running time of each module, GiPSiNet includes instrumentation software that can be selectively turned on or off at run-time.

Network Dynamics. The network behavior effects user's perception of the simulation quality. For example, a user could get quickly dissatisfied with an interactive simulation if the network introduces unacceptable end-to-end delays. Networks introduce delays between communication endpoints due to propagation delays, data unit transmission time, in-network packet processing, and queuing delays. As a result, an RTT τ_i elapses between the transmission of V_i and the receipt of x_i (Figure 3). Analogously, bandwidth limitations increase the transmission time and consequently the RTT. Moreover, packet drop-outs can cause data unit losses and degrade the simulation performance. Delays can be studied by linking the client and the server through a bridge that forwards frames after a delay. The emulator can also introduce packet losses on an emulated link and limit the bandwidth available between the two endpoints with a simple leaky bucket scheme. The emulation technology is standard and is based on a commercial off-the-shelf workstation with two network cards running dummynet over FreeBSD. Such an emulator will be installed and tested, and a convenient interface will be provided to quickly set the simulation parameters.

Another potential reason for poor user-perceived performance is that best-effort networks can (and often do) provide low QoS levels in the presence of congestions. An evaluation objective is to assess the impact of network congestion on simulation performance. The effects of congestion can be evaluated by network emulations on the benchmark tasks. An emulation network are developed to contain the end-points, possibly the network emulator, plus another two workstations that generate cross-traffic. Scripts will also be developed to inject cross-traffic according to various models (constant bit rate (CBR), Pareto, TCP-regulated bulk data transfers). Our in-lab results will be further validated on a wide-area network.

Experiments. The test bed will be used to obtain performance figures on representative tasks. The benchmark tasks are to be run under several emulated network conditions by setting a range of values for delays, losses, and cross-traffic in the emulator and in the traffic generator. Task performance numbers and GiPSiNet computational requirements will be collected, as described above. Such performance figures will be related with network conditions and type of cross-traffic, so that general trends can be isolated. In general, if network conditions are perfect (e.g., client and server running on the same machine), then the only difference between GiPSiNet and the non-networked GiPSi is the computational overhead posed by the middleware modules. At the other extreme, if network QoS is inexistent (e.g., a network partition), no communication can take place. The objective of these experiments is to explore this spectrum and to relate quantitative task performance with network conditions, as well as quantify the computational overhead of the middleware.

Expert Reviews. To obtain a more adequate evaluation from user's perspective, expert reviews are conducted by involving participants of a simulation session. Expert reviews have proved to be effective to provide useful feedback in user interface design [10]. In a simulation session, the participants will be asked to freely manipulate the software and devices attached to it, and to answer a list of questions regarding the user interface satisfaction, their experience with the haptic devices, and open-ended questions with respect to the research values and future directions of the software. The feedback is used for improve the quality, accuracy, effectiveness, and usability of the software.

5 Related work

A scalable network architecture has been proposed for distributed virtual environments with dynamic QoS over IPv6 [3]. Later, a real time platform middleware was developed in the LAN environment to provide an architecture for prototyping realtime multimodal I/O projects [11]. In [15], a queuing and packet forwarding algorithm was proposed and evaluated for distributed virtual environment applications. Haptic collaboration over the network is under intensive investigation [7, 9, 8]. In [7], the authors examine the influences of network latency (RTT) on the quality of haptic collaboration. It was shown that the object controllability, the feeling of touch, and the performance of a task are very sensitive to RTT. However, to the best of our knowledge, our project is the first to systematically develop and evaluate a middleware for distributed surgical simulations.

6 Conclusion and future work

In this paper, we described the design, techniques, and evaluation plan of the GiPSiNet middleware project, which extends GiPSi to a network environment and enhance the quality of networked surgical virtual simulations. In the future, we will deploy GiPSiNet and run experiments for a better understanding of the interaction between network and the virtual environments. We will also systematically evaluate the qualitative and quantitative performance of GiPSiNet.

Acknowledgement

The authors gratefully acknowledge Technology Opportunities Program (TOP) of Department of Commerce, NASA NNC05CB20C, NSF CCR-039910, and the Virtual Worlds Laboratory and School of Graduate Studies at Case Western Reserve University.

References

[1] First, do no harm. Technical report, Institute of Medicine, 1999.

[2] M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Systems Research Center, Digital, 1994.

[3] M. Eraslan, N. D. Georganas, J. R. Gallardo, and D. Makrakis. A scalable network architecture for distributed virtual environments with dynamic QoS over ipv6. In *8th IEEE international symposium on computers and communications*, 2003.

[4] P. M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47:381–391, 1954.

[5] P. M. Fitts and J. R. Peterson. Information capacity of discrete motor responses. *Journal of Experimental Psychology*, 67:103–113, 1964.

[6] T. Goktekin, M. C. Cavusoglu, F. Tendick, and S. S. Sastry. Gipsi: A draft open source/open architecture software development framework for surgical simulation. In *the International Symposium on Medical Simulation*, pages 240–248, 2004.

[7] S. Matsumoto, I. Fukuda, H. Morino, K. Hikichi, K. Sezaki, and Y. Yasua. The influences of network issues on haptic collaboration in shared virtual environments. In *Proceedings of the Fifth PHANTOM Users Group Workshop*, 2000.

[8] M. McLaughlin, G. Sukhatme, W. Peng, W. Zhu, and J. Parks. Performance and co-presence in heterogeneous haptic collaboration. In *11th IEEE Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (HAPTICS)*, 2003.

[9] M. L. McLaughlin, J. P. Hespanha, and G. S. Sukhatme. *Touch in virtual environments: Haptics and the design of interactive systems*. Prentice Hall, 2002.

[10] Nielsen, Jacob, Mack, and Robert (Editors). *Usability Inspection Methods*. John Wiley and Sons, 1994.

[11] G. Pava and K. E. MacLean. Real time platform middleware for transparent prototyping of haptic applications. In *Proceedings. 12th International Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, 2004.

[12] R. Ramjee, J. F. Kurose, D. F. Towsley, and H. Schulzrinne. Adaptive playout mechanisms for packetized audio applications in wide-area networks. In *INFOCOM*, pages 680–688, 1994.

[13] J. Smed, T. Kaukoranta, and H. Hakonen. Aspects of networking in multiplayer computer games. In *Proceedings of the International Conference on Applications and Development of Computer Games in the 21st Century*, pages 74–81, 2001.

[14] F. Tendick, M. Downes, T. Goktekin, M. C. Çavuşoğlu, D. Feygin, X. Wu, R. Eyal, M. Hegarty, and L. W. Way. A virtual environment testbed for training laparoscopic surgical skills. *Presence*, 9(3):236–255, June 2000.

[15] Q. Zhou, H. Yu, D. Makrakis, N. D. Georganas, and E. Petriu. Quality of service support of distributed interactive virtual environment applications in ip-diffserv networks. In *IEEE International Workshop HAVE (Haptic Virtual Environments and Their Applications)*, pages 97–102, 2002.