Galileo: a Tool for Simulation and Analysis of Real-Time Networks

Edward W. Knightly* and Giorgio Ventre*†

TR-93-008

March 1993

Abstract

Galileo is a flexible tool for simulation of heterogeneous real-time communication networks and for development and verification of network protocols. Galileo provides several unique features that make it particularly suitable for the simulation and analysis of networks that provide quality-of-service guarantees. First, its object-oriented programming environment provides the means for a modular, hierarchical, heterogeneous description of networks. Second, its multimedia device interface provides the tools for a qualitative analysis of network protocols. Finally, Galileo's network interface provides interaction with actual networks to access real data and simulate realistic multimedia scenarios.

*The Tenet Group, Computer Science Division, Department of EECS, University of California, Berkeley and International Computer Science Institute, Berkeley. E-mail: {knightly,ventre}@icsi.berkeley.edu

†On leave from Dipartimento di Informatica e Sistemistica, Universitá degli Studi di Napoli "Federico II", Napoli, Italy.

1 Introduction

In this paper we present Galileo, a tool designed and implemented to address several problems unique to the simulation and analysis of real-time communication networks. The tool is based on Ptolemy [4], an object-oriented software environment that serves as a foundation for simulation of heterogeneous, hierarchical systems. Galileo has several unique features that distinguish it from the vast number of existing network simulators such as Netsim, REAL, and NEST.

First, Galileo is object-oriented. This feature, inherited from Ptolemy, effectively exploits the modular and hierarchical nature of communication network protocols and facilitates their rapid prototyping and testing. The simulation environment, together with Ptolemy's graphical interface, provides a block diagram representation of the network where modules may be encapsulated into higher level modules. This representation allows easy configuration of networks and network protocols as well as a hierarchical view of the network. The object-oriented paradigm extends to the protocol code (written in C++) and facilitates an easy exchange of a protocol's algorithms. The modular description of networks also provides the capability of simulating the heterogeneity of internetworks. Because future networks will offer a wide range of communication architectures and services in a rapidly changing scenario, Galileo provides easy integration of new network models into the existing ones, and supports modularity in their development.

The second feature of Galileo is quality-of-service simulation with an emphasis on multimedia applications. The primary goal of the real-time network simulator is to show the quantitative effects of network management decisions on the traffic, and to verify both the correctness and efficiency of the resource allocation schemes. In addition, Galileo facilitates the qualitative evaluation of protocols by interacting directly with multimedia devices and applications. For example, Galileo supports the sending of video packets across the simulated network showing the effects of the network on transmitted video streams. This feature is especially useful to the network designer because it provides the capability of analyzing the impact on the quality of the communication provided to end-users.

Finally, Galileo provides the tools for interacting with the network itself. For example, if a user wishes to simulate a video conference with another node on the network, Galileo reads the actual information on the current state of the network by interacting with the network protocols. With this information, Galileo is able to simulate the proposed video connection and to show the impact of the network on the communication. Thus, this feature provides the means for the study of realistic scenarios. Indeed, a tool for simulation and analysis of networks should not be isolated from the external world. The availability of actual data from the network components can dramatically improve the quality of the simulation results, particularly for multimedia applications, where precise models for traffic sources still need to be developed.

This paper is organized as follows. First, some issues unique to real-time networking and simulation are examined along with the Tenet protocol suite (under development by the Tenet Group at the University of California at Berkeley and the International Computer Science Institute). Second, Galileo's architecture is described in terms of its functional modules. Next, Galileo's main modules are described: the Simulation Module, the Multimedia Interface Module, and the Network Analysis Module. Finally, we present an example simulation and conclude.

2 Real-Time Communication

Real-time communication protocols generally present mechanisms for the reservation and control of the network's resources. In this section, the general principles of real-time communication are presented along with the Tenet protocols, a real-time protocol suite.

2.1 General Principles

The simulation of real-time (i.e., guaranteed performance) communication networks has two primary components. First, real-time protocols must partition the network's resources among various clients. That is, the protocols need an algorithm to allocate the network's bandwidth and buffer space to its clients according to their corresponding quality-of-service (QoS) requirements (as well as some pricing scheme). In the Tenet resource management protocol, RCAP (Real-time Channel Administration Protocol) [1], admission control is performed by a series of tests to check if sufficient resources are available to meet the client's QoS requirements. The calculations involve transforming the client's requirements for end-to-end delay, delay jitter, and loss probabilities into local node parameters so that each node of the network guarantees local performance, thus ensuring global performance. The client's traffic characteristics (minimum and average packet interarrival times along with an averaging interval) are also used in the calculations. The real-time channel may be viewed as a contract between the client and the network. If the channel is accepted, the network guarantees the requested performance bounds provided that the client obeys its specified traffic characteristics.

The second component of real-time communication is scheduling. A scheduling algorithm is needed that can provide the performance guarantees promised by the network. There are several options available, with the underlying requirement that packets are scheduled with some manner of fairness or priority. This is necessary to ensure that a non-real-time channel does not degrade the performance of a real-time channel and that a malicious real-time channel (one with a source sending at a rate greater than its agreed peak rate) does not degrade the performance of other real-time channels. The Tenet network layer protocol is RTIP (Real-Time Internet Protocol) [7]. RTIP currently uses a deadline-based scheduling algorithm called multiclass EDD (Earliest Due Date) [3], where an incoming real-time packet is queued according to its local node deadline as calculated by RCAP. In addition to real-time traffic, EDD supports traditional, best-effort traffic in a separate, lower priority, FCFS queue.

2.2 The Tenet Protocols

The Tenet protocol suite consists of RMTP (Real-Time Message Transport Protocol), CMTP (Continuous-Media Transport Protocol), RTIP, and RCAP. The logical structure of the protocols is shown in Figure 1. Currently, Galileo simulates RTIP and RCAP. The primary functions of RTIP and RCAP are as follows.

- RTIP is a simplex, sequenced, unreliable, guaranteed-performance packet service providing rate control, jitter control, packet scheduling, and data transfer.
- RCAP performs the admission control tests to provide mathematically provable guarantees for throughput, end-to-end delay, delay jitter, and loss rates.

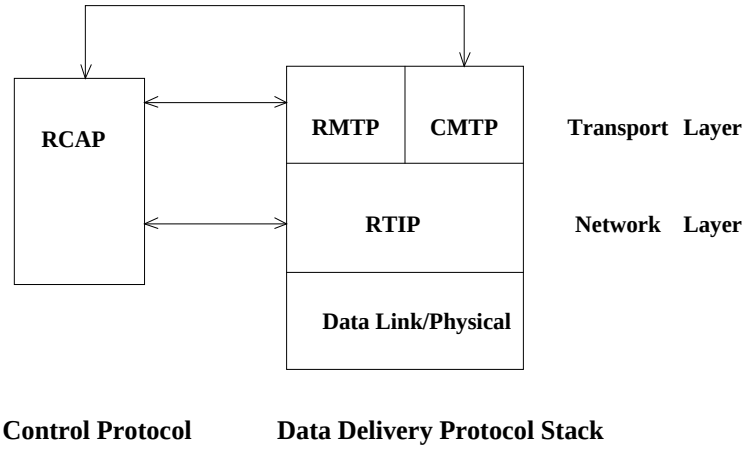

Figure 1: Tenet Protocol Suite

2.3 The Tenet Guarantees

Although Galileo is flexible enough to support a variety of communication protocols (both real-time and non-real-time), the Tenet real-time protocols are those in the current implementation of the simulator. As mentioned above, the Tenet protocols are based on a contract between the client and network. If RCAP accepts a real-time channel, the network guarantees the client's requested performance bounds provided that the client sends packets at rates not exceeding the promised ones. If a channel is accepted, the client promises to obey (x_{min}, x_{ave}, I, s_{max}), the minimum and average packet interarrival time over an interval I with maximum size s_{max}. In return, network guarantees (D, J, Z, U, W), the maximum end-to-end packet delay and delay jitter, guaranteed with probability Z and U respectively. W is the probability that a packet is not dropped due to buffer overflow.

If Z = U = W = 1, then the guarantees are absolute and the channel is called a deterministic channel. If any one of these parameters is less than one, then the channel is called statistical. If all probabilities are zero, then no guarantees are made and the channel is called best-effort. Thus non-real-time traffic may coexist with real-time traffic via the best-effort class of traffic.

A channel is established in the following manner. The requesting client generates an RCAP control message containing its requested performance guarantees and its promised traffic characteristics. This message will make a round trip between the source and the destination along the fixed route given by the source. On the forward part of the trip, sufficient resources are reserved for the new channel. At the destination, the client's end-to-end requirements are compared with the aggregate performance bounds offered by the nodes along the route and the connection is appropriately accepted or denied. If accepted, on the reverse part of the trip each node's resources are relaxed to the minimum level so that the local performance bounds may be guaranteed. The details of the admission control algorithms are found in [3].

The unique requirements of simulating real-time networks are apparent. The simulator must provide quality-of-service validation in addition to performing admission control calculations. For real-time applications, this QoS validation must be qualitative as well as quantitative. Furthermore, the need for an object-oriented programming environment is also evident. With the current variety of algorithms for network functions such as admission control and scheduling, an object-oriented

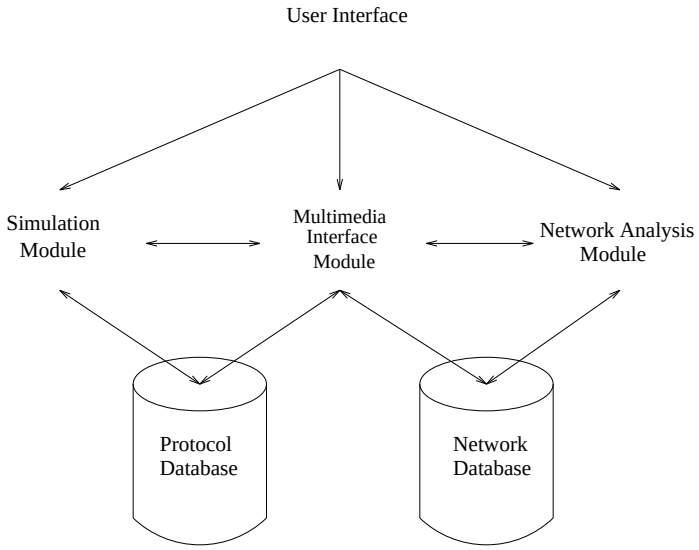

Figure 2: Galileo Architecture

environment allows comparison of these algorithms with minimal programming difficulty. In addition, these algorithms must coexist in order to simulate heterogeneous networks. The sections below describe how these simulation needs are met.

3 Galileo's Architecture

Galileo's architecture is shown in Figure 2. At the highest level is the graphical user interface shown in Figure 3. The interface, provided by Ptolemy, allows networks to be created graphically by placing and connecting icons. The User Interface interacts with three modules: the Simulation Module, the Multimedia Interface Module, and the Network Analysis Module.

The Simulation Module consists of a library of programming modules such as protocols and network architectures. These programming modules are encapsulated and interconnected to simulate an entire network. The software environment is object-oriented — a feature inherited from Ptolemy and C++.

The Multimedia Interface Module provides the means for qualitative analysis of network protocols. For example, Galileo shows the effects of the network on a video stream's delay, delay-jitter, and packet loss. These effects may be qualitatively analyzed for different network topologies, loads, and protocols to provide a more thorough study of how proposed algorithms affect multimedia traffic.

The Network Analysis Module provides the means for better simulating real networks by providing an interface to the network via the network's control protocols. The Network Analysis Module gets the most recent information on the current state of the network from the local host. The local host will be required to have some information about the state of the network in order to make routing decisions. If the information provided by the host is insufficient for the required simulation, the Network Analysis Module will interact with the control protocol to probe the network for further

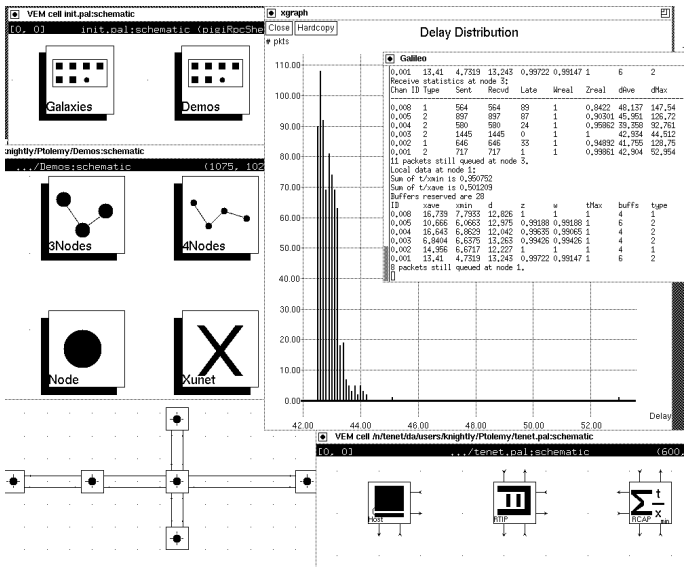

Figure 3: Graphical User Interface

state information.

The final components of Galileo's architecture are the Protocol Database and the Network Database. The Protocol Database provides the foundation for the simulations by defining the protocol algorithms. The Network Database stores the information obtained by the Network Analysis Module.

4 Object-Oriented Simulation Module

Galileo's software foundation, Ptolemy, provides an object-oriented programming environment. Ptolemy performs most of the underlying functions needed in any discrete event simulation such as a graphical interface, discrete event scheduling, and efficient handling of packets. On top of these, Galileo provides functions specific to the simulation of communication networks. An example of the difference in the functions that are performed by Galileo and Ptolemy is the following. To send a packet, Galileo defines the packet's structure and when and where the packet is sent. Ptolemy schedules the packet (as an event in the simulation), provides reference counting for the packet, provides the means for sending the packet among protocols and nodes, and ensures efficient use of memory as packets are created and destroyed. This environment allows the Galileo programmer to be concerned with network issues such as channel establishment routines rather than with simulation issues such as event scheduling. C++, the programming language of Galileo and Ptolemy, provides the underlying support for object-oriented programming.

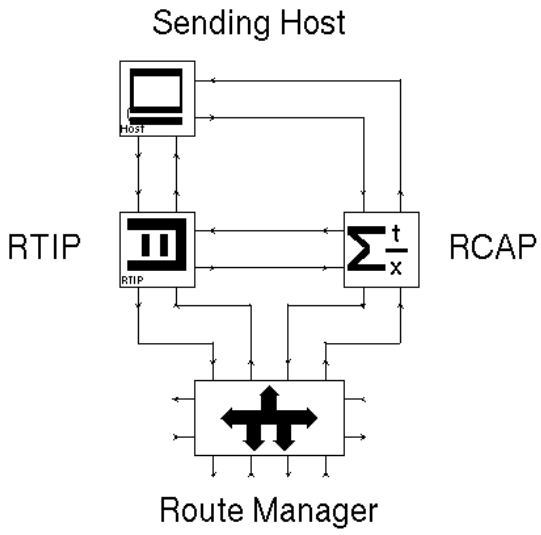

Figure 4: Tenet Node

4.1 Modular Programming

Galileo provides (via Ptolemy) a modular programming environment. This modularity allows a block diagram description of the nodes within the network as well as the protocols within the nodes. Figure 4 shows the block diagram configuration of a Tenet node. The interaction among modules in the node when creating a real-time channel can be described as follows. First, the sending host sends an establish request message to the node's RCAP. This message is sent to the RCAP in each node along the round trip as described earlier. If the request is accepted, the RCAP at each node sends its RTIP the local guarantees. The sending host then sends data packets to the network via RTIP. The function of the Route Manager is to send a received packet to the next Route Manager on the route list (assuming source routing) or to the node's appropriate protocol, RCAP or RTIP. This model of the node provides a network layer (as in the OSI model) simulation along with added control by RCAP, and a higher level host to generate the traffic. Galileo is currently being extended to do transport layer simulations by designing the appropriate transport protocol modules, RMTP and CMTP, which will be placed between the Host and RTIP.

The modular nature of Galileo provides several advantages. First, at the highest level, a network is easy to create. Figure 5 shows the Galileo schematic of a six node network. This network was created by selecting (via the graphical interface) the appropriate node modules and interconnecting them. Creating networks with new topologies is as simple as connecting a block diagram and giving each block its appropriate parameters. Second, it is relatively easy to integrate new protocols. A new resource management protocol may be tested by removing the RCAP module in Figure 4 and connecting the new protocol in RCAP's place. In a similar manner, nodes may be created with very different characteristics than the Tenet node. For example, another node in the network may provide only IP and may not be able to support real-time protocols. The effects of real-time traffic traversing non-real-time nodes may then be studied.

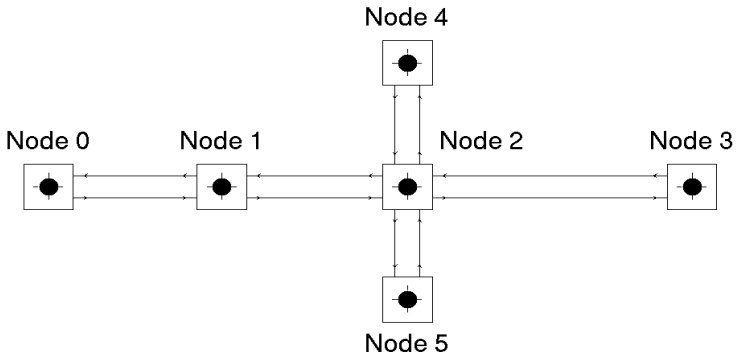

Figure 5: Network Topology

Galileo's modularity also extends to the programming environment. For example, the default scheduling algorithm in RTIP is EDD. A second algorithm, Rate-Controlled Static-Priority Queueing [6], will also be used in the future. In Galileo's object-oriented programming environment, changing the scheduling algorithm only requires changing the scheduling object. This setup allows the possibility of choosing the scheduler at run-time by utilizing Ptolemy's highly parameterized block-diagram representation. Thus, for example, two ATM switches may have two different scheduling algorithms. In this case, they will also require different channel establishment tests. Different tests will be required in other cases, for example when part of the network uses high-speed circuit switching. Such a programming environment allows heterogeneity at all levels of simulation.

In sum, the block-diagram representation of the network and of the protocols within the network provides a flexible framework for rapid development and verification of network protocols. The modular nature of the programming environment extends this flexibility to the algorithms within the protocols. Such an environment allows simulation of heterogeneous networks such as complex internetworks.

4.2 Hierarchical Simulation

Galileo's hierarchy is also provided via Ptolemy. The lowest level in the Ptolemy hierarchy is called a star. Stars contain C++ code as well as information for the Ptolemy preprocessor. Examples of stars include all of the modules in Figure 4. Each of these stars has several parameters (e.g., node buffer space) corresponding to variables in the C++ code. In the discrete event domain, each star has a *go()* function that is called when either another star sends it a packet or the star calls its own *go()* function at a later time (via internal feedback). An example of this is found in RTIP. Whenever a packet is received from either the sending host, RCAP, or the network (via the Route Manager), RTIP's *go()* function is called and an *enqueue(pkt)* function is called (again, the queueing algorithm may be specified at run time if desired). To dequeue the packet, RTIP has an internal clock representative of the link speed. This internal clock also calls RTIP's *go()* function so that the packet is dequeued at the appropriate time according to the link bandwidth and the packet size. As stated above, Ptolemy then schedules this packet in the simulator's event queue and sends it to the appropriate module.

The next higher level of Ptolemy's hierarchy is a galaxy. A galaxy consists of interconnected stars as well as other galaxies. An example of a galaxy is each of the nodes in Figure 5. These nodes are an encapsulation of the four stars in the Tenet node found in Figure 4. Galaxies have parameters that are linked to lower-level star or galaxy parameters to link the node parameters with the protocols. This hierarchy may be extended to form higher level galaxies consisting of interconnected galaxies and stars. For example, Galileo may utilize this feature by encapsulating the six node network of Figure 5 into a single galaxy (with some added outgoing links) to represent a subnetwork in a complex internet. The parameters of this subnetwork galaxy will then reflect the effect of the entire subnetwork on a traversing packet. This hierarchical view of the network is the approach taken in [2], where the highest level of an internetwork consists of gateways that interconnect networks, and "logical links" that represent networks between the gateways. This view of the network is called a level-1 network. The level-2 network consists of the component networks of the level-1 network, where some of the level-2 networks may themselves be internetworks extending the abstraction to a third level. This process is then iterated until all of the physical links (interconnecting the actual nodes) are represented. Such an abstraction reduces an internetwork to a manageable hierarchical structure while maintaining the necessary heterogeneity. Thus, Galileo supports the encapsulation of subnetworks to provide this logical link view of networks.

In sum, Galileo provides the modularity and hierarchy needed to simulate real-time networks. Its modularity provides a good environment for testing and developing protocols as well as simulating heterogeneous networks. Its hierarchical structure provides the means for encapsulating network components in order to simulate complex internetworks.

5 Multimedia Interface Module

As communication networks begin to provide real-time guarantees, multimedia applications such as interactive video conferencing will utilize this service. The conference participants will have requirements for bandwidth, delay, delay-jitter, and loss rates. The application program may make efforts to improve the services offered by the network. For example, it may use buffering to absorb some delay-jitter or use (necessarily fast) signal processing to make up for dropped packets. The purpose of Galileo is not to simulate the final application but rather to show the raw effects of the network on multimedia traffic [1]. Such a qualitative analysis will provide the network designer with a measure of how well the protocols are supporting a practical real-time load. The application designer will also benefit by using the qualitative analysis to better analyze the underlying service the network is providing.

Galileo's multimedia interface module provides the tools for qualitative analysis of network protocols. For example, it can show the effects of the network on a video stream's delay, delay jitter, and packet loss. Currently, an interface is provided to display stored video. Work in progress includes displaying video directly from a camera and playing received audio.

[1] It should be noted, however, that Galileo's modular structure allows interaction with application software.

5.1 Video

Two methods of displaying a simulated received video stream are the following. In the first method, the simulator displays the received video in real *simulation* time. In the second method, the simulator stores the necessary information on disk, and recreates the received video stream when the simulation is complete. Currently, Galileo uses the latter approach although the former approach is also under development.

To show the received video stream in real simulation time, Galileo will send received video frames to the display window in the following manner. First, the source (of the test network) will packetize the actual video data. The packet, implemented as a C++ object, is then sent to the destination in the manner described in section 4. When the destination receives the packets it will reassemble them into frames. When the end of a frame is received the frame and its time stamp will be sent to an application program. This application will either display the frame immediately or after a delay corresponding to the difference in frame time stamps. This method will show the degradation of the video stream due to dropped packets, excessive delay, and delay-jitter. The disadvantage of this method is its severe requirements on the computer used for simulation. That is, for a received video stream to be interesting, the test network should be large and the amount of cross traffic should be heavy or at least variable. This would correspond to a computationally intense simulation. However, by displaying small black and white frames and keeping the network size reasonable, this method may still be feasible.

A second approach is to run the simulation and store on disk a trace of the sizes, arrival times, and sequence numbers of the packets from the video channel. Without compression, the display application is simple. Using knowledge of the number of bits per pixel and pixels per frame of the original sequence, an application program displays the received sequence using the trace file and the original images. With compression (not yet implemented), one possible solution is the following. The original images may be read by the simulator, compressed, packetized, and sent across the simulated network. At the destination, the data is unpacketized, uncompressed and then the resulting image is stored on disk along with its arrival time. The compression and decompression algorithms can be easily added as additional modules within Galileo. The display program for this approach differs in that it uses the (stored) received images and time stamps. In the uncompressed case the received images do not need to be stored because they can be easily deduced — the receive frame is the original frame played at the appropriate time, with certain pixels missing (corresponding to dropped packets).

In either case, the most important result is to show the *raw* effects of the network on the video stream. To investigate the improvements of the application on the video stream, application modules may be added to Galileo. For example, an application module could be added that interpolates among missing missing pixels and attempts to smooth some of the network's delay jitter with buffering. In this respect, Galileo can show the received video stream at several levels from the raw network stream to the final application stream. Such features better facilitate a qualitative analysis of the effects of protocols, traffic, and network configuration on the video stream.

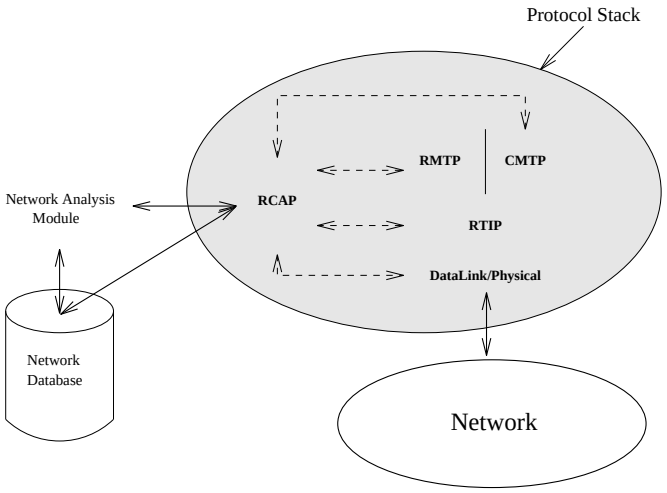

Figure 6: Network Analysis Module Interactions

5.2 Audio

Although the bandwidth requirement of an audio stream is not as stringent as that for video, a qualitative analysis is still useful. For example, the audio stream's maximum and average end-to-end delay will determine if a conversation is feasible. Also, the stream's delay-jitter will provide the destination application with information about buffer requirements and synchronization requirements. Audio streams will also be a valuable platform for analysis of probabilistic QoS guarantees. In this type of simulation, the final analysis on the acceptability of the received QoS is most easily judged when the received audio stream is played on the workstation speaker.

6 Network Analysis Module

This section describes the final component of Galileo's architecture, the Network Analysis Module. As described earlier, one of the most important features of Galileo is that it allows a complete understanding of the effects of transmission of continuous media over a real-time communication network.

This effect can be obtained through simulation in two different ways. The first is via integration of models of network components, such as gateways and hosts, with models of sources of multimedia traffic. In Galileo this is accomplished by means of the Simulation Module and the information stored in the the Protocol Database.

The second solution is to use real, updated data about the current state of the network for simulations. This data may be collected in the network itself through the Network Analysis Module. Figure 6 shows how this feature is implemented in the simulator. The Network Analysis Module was designed explicitly to allow Galileo access to the information regarding the network traffic. This information is obtained directly through interactions with the local modules of the real-time protocols available on the node where Galileo is running. The figure shows the current implementation of the Network Analysis Module developed for the Tenet real-time protocol suite.

There are two major issues related to interfacing the simulator with the network. First, which mechanisms are needed to provide the Network Analysis Module with updated information from the network? Second, how can the communication overhead be limited so that the gathering of information does not have a measurable effect on the network traffic? To solve these problems we propose a solution composed of two steps: *network database analysis* and *direct data collection*.

The Network Analysis Module accesses information concerning the state the network through RCAP, the protocol responsible for the establishment and management of real-time connections. RCAP collects information both during the establishment and entire lifetime of a connection. The analysis and collection of this information and the use of this information by Galileo is described in section 6.1. If the information collected by Galileo from the Network Database is insufficient for the required simulation, Galileo will collect further information from the network by sending a status request message to the node of the network that has the required information. The method of further probing the network is described in section 6.2.

6.1 Network Database Analysis

In the Tenet protocols, the establishment of a new real-time connection requires a number of tests in each node along the path from the source of the connection to the destination. The tests are required to verify the availability of the resources needed to guarantee the specified quality-of-service. In order to exploit the information produced during these tests and to limit the amount of state-related information to be exchanged periodically in the network, Galileo uses the information produced by the existing channel establishment and management algorithms. That is, when channels are established from a node in the network, an RCAP control message makes a round-trip from the source to destination and back (as described in section 2.3). During this round-trip, the message collects data (e.g., current utilization and reservation information) regarding the state of each node along the round trip. This information is then stored in the network's nodes so that other network mechanisms (e.g., routing schemes) may utilize it. RCAP collects data not only during the channel establishment tests but also during the entire lifetime of the connection. A summary of such information is also communicated to other interested network nodes (e.g., network gateways).

The information already available from the RCAP control messages represents the initial data to be included in Galileo's Network Database. The data is updated automatically whenever it is required, i.e., each time a new connection is established or modified and requires a different resource allocation in a node.

Thus, the first step in data collection involves Galileo utilizing information that is already available. The advantage of this limited distribution approach is that it reduces the communication overhead for maintaining updated information. However, this approach does influence the availability and the reliability of the data stored in Galileo's Network Database. When a simulation with actual data is required, Galileo's Network Analysis Module accesses the Network Database to get the information needed for the simulation. If the data concerning all the nodes in the simulation is available, Galileo can begin the simulation immediately. Otherwise, the second step in the network data collection has to be performed. This step involves the *direct* collection of data by Galileo.

6.2 Direct Data Collection

In this mechanism, Galileo tries to access the information produced by the protocols stored in nodes other than the node on which Galileo is running. When the Network Analysis Module requires information regarding the status of certain nodes, a message is sent to the local instance of the channel administration protocol (i.e., RCAP) to request the transmission of such data. The local RCAP forwards this message to the remote nodes and stores the additional data received in the Network Database.

This mechanism appears to be very efficient, since the transmission of information related to the state of a node is required only if a request is made. Furthermore, the request is limited to the nodes whose status is of interest to the Network Analysis Module. However, in this approach, the amount of time needed to gather the information is greater than the time required to access only the data in the Network Database, and it explicitly requires additional communication in the network for the collection of the data.

Even though the implementation alternatives presented here are dependent on the algorithms and the architectural solutions adopted in the Tenet real-time scheme, it should be noted that the same ideas can be applied to other real-time network architectures. The only condition is that the protocols available in such architecture are able to produce and gather reliable information about the current load in the network components. In order to improve the capability of Galileo to cope with networks adopting different approaches to guaranteed QoS in real-time communication, we plan to extend the Network Analysis Module to interact successfully with networks adopting different schemes for controlling and managing the resources allocated to real-time traffic.

7 Example

A simple six node network is used to demonstrate Galileo's capability to perform QoS simulation. Figure 5 shows the topology of the network. In this simulation, channels (both real-time and non-real-time) are established between nodes zero and three and nodes four and five. When a sending host receives an establish accept message from the network, the host begins to send packets according to a bursty two-state Markov process that obeys the client's agreed traffic characteristics, (x_{min}, x_{ave}, I).

7.1 Quantitative Analysis

To demonstrate the aspects of quantitative QoS simulation, Figure 7 shows how the network's utilization is related to the client's requested QoS. For this experiment, statistical channels with identical parameters are requested between nodes zero and three until no more requests are accepted. These requests are made with pre-existing (fixed) real-time cross-traffic between nodes four and five. The requests for real-time connections between nodes zero and three are all made with $x_{ave} = 50$ and $D = 80$. The figure shows how both x_{min} and Z affect the number of channels accepted. Moreover, the figure shows that, regardless of x_{min} and Z, no more than 20 channels may be accepted even as x_{min} is further increased or Z is further decreased. This demonstrates that scheduler saturation has occured and only increasing D will increase the number of channels accepted. Such information is

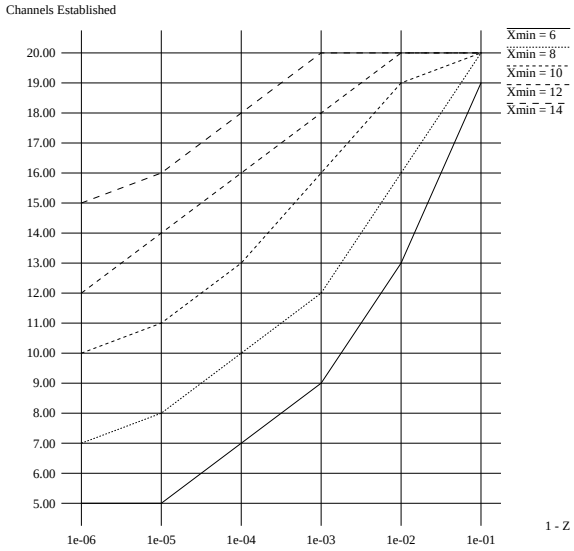

Figure 7: Channels Established vs. Delay Violation Probability

valuable in a flexible client interface to RCAP as in [5]. If the network denies a client's request, the client may make an intelligent modification of its QoS parameters so that the second request will be accepted. Moreover, the network itself may use such information to make a counter-proposal to the failed request, so that the the client will have the option of accepting a channel with less than the requested QoS.

Figure 8 shows a delay histogram automatically generated from a Galileo simulation of the six node network. The figure shows quantitatively the effects of the jitter control scheme. That is, it shows the small delay-jitter received by a real-time channel even in the presence of heavy, bursty, cross-traffic.

7.2 Qualitative Analysis

Multimedia services such as video and voice will be an important application for real-time protocols. Although a quantitative analysis is essential for validation of the protocols, a qualitative analysis is important for understanding how the real-time service will affect multimedia communication.

Figure 9 shows the effect of sending a best-effort video frame from node zero to node three in heavy traffic. In this simulation, a packet of unit size contains 48 bytes of data as in an ATM cell. Each pixel is 1 byte of data, so that a dropped packet causes 48 pixels to be lost. In this simulation, the deterministic and statistical channels both received all of their packets on time so that the received frame is identical to the sent frame. The degraded frame of the best-effort channel represents a non-real-time connection such as IP. Of course, an IP connection may resend the lost packets so that eventually a full image is received. However, if the application has performance requirements (e.g., it is interactive video), this excessively delayed information must be considered lost.

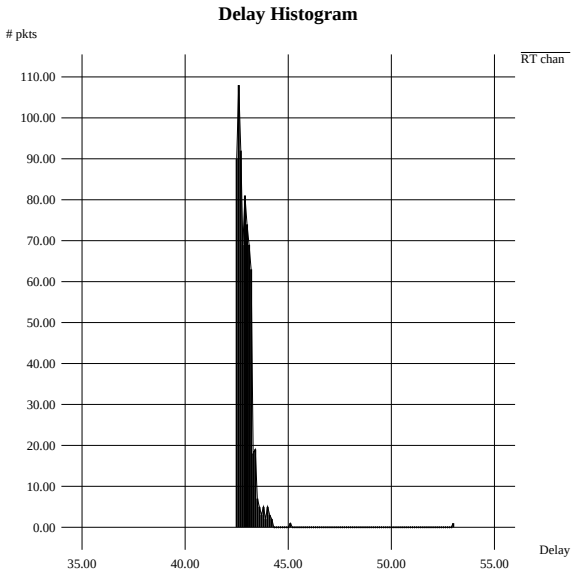

Figure 8: Delay Histogram for a Real-time Channel

Original Image
(a)

Received Image
(b)

Figure 9: "Before" and "After" Frame of a Best-Effort Connection

8 Conclusions

Galileo is an object-oriented tool designed and implemented to simulate real-time communication networks and protocols. We believe that Galileo provides several unique features necessary for analysis of real-time networks. First, the simulator is object-oriented. This provides a modular description of networks and network protocols and allows new protocols and algorithms to be easily redesigned and tested. Second, Galileo provides the multimedia interface required for a qualitative study of the network. For example, to analyze the quality-of-service of a video channel, it is essential to show the received video stream and observe it's delay, delay-jitter, and loss rates. Finally, Galileo provides an interface to the network itself. This feature facilitates a realistic analysis of the network by using the networks current state as simulation parameters.

Acknowledgments

The authors are especially grateful to Domenico Ferrari, and to all members of the Tenet Group for their ideas, comments, and criticisms.

This research is supported by the National Science Foundation and the Defense Advanced Research Projects Agency (DARPA) under Cooperative Agreement NCR-8919038 with the Corporation for National Research Initiatives, by AT&T Bell Laboratories, Digital Equipment Corporation, Hitachi, Ltd., Hitachi America, Ltd., Pacific Bell, the University of California under a MICRO grant, and the International Computer Science Institute.

References

[1] A. Banerjea and B. Mah. The real-time channel administration protocol. In *Proceedings of the Second International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 160–170, Heidelberg, Germany, November 1991. Springer-Verlag.

[2] D. Ferrari. Real-time communication in an internetwork. *Journal of High Speed Networks*, 1(1):79–103, 1992.

[3] D. Ferrari and D. Verma. A scheme for real-time channel establishment in wide-area networks. *IEEE Journal on Selected Areas in Communications*, 8(3):368–379, April 1990.

[4] E. Lee and D. Messerschmitt. *The Almagest: Manual for Ptolemy*. U. C. Berkeley Deptartment of EECS, 1992.

[5] J. Ramaekers and G. Ventre. Quality-of-service negotiation in a real-time communication network. Technical Report TR-92-023, International Computer Science Institute, Berkeley, California, April 1992.

[6] H. Zhang and D. Ferrari. Rate-controlled static priority queueing. In *Proceedings of INFOCOM'93*, San Francisco, California, March 1992.

[7] H. Zhang, D. Verma, and D. Ferrari. Design and implementation of the real-time internet protocol. In *Proceedings of the IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems*, Tucson, Arizona, February 1992.