89800

Technical Report UMTRI-97-15 April, 1997

A Description of the UMTRI Driving Simulator Architecture and Alternatives

Alan Olson and Paul Green

umtri
HUMAN FACTORS

UMTRI The University of Michigan Transportation Research Institute

Technical Report Documentation Page

1. Report No. UMTRI-97-15	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle **A Description of the UMTRI Driving Simulator Architecture and Alternatives**		5. Report Date April, 1997
		6. Performing Organization Code
7. Author(s) Alan Olson and Paul Green		8. Performing Organization Report No. UMTRI-97-15
9. Performing Organization Name and Address The University of Michigan Transportation Research Institute (UMTRI) 2901 Baxter Rd, Ann Arbor, Michigan 48109-2150		10. Work Unit no. (TRAIS)
		11. Contract or Grant No.
12. Sponsoring Agency Name and Address none		13. Type of Report and Period Covered
		14. Sponsoring Agency Code

15. Supplementary Notes
The development of the software described was funded by the Transportation Research Board's Innovations Deserving Exploratory Analysis (IDEA) program.

The UMTRI driving simulator family consists of six simulators used for research on in-vehicle devices (e.g., cellular phones, collision warnings displays), medical considerations (e.g., impairments due to alcohol and Alzheimer's disease, along with individual differences due to age and sex), steering system dynamics, and simulator design characteristics.

This report describes the Driver Interface Research Simulator, a network of Macintosh computers that generate the road scene, instrument panel graphics, sound, and traffic. This report examines the following basic questions:

1. Why should a networked simulator be used to present traffic?
2. How should tasks be allocated among computers?
3. Which network should be used?
4. How should simulators on different computers be coordinated?
5. How should simulators track the locations of vehicles simulated on other computers?

Decisions were made with an eye towards performance, flexibility, and ease of implementation. The solutions chosen were:

- 10Base-T Ethernet using AppleTalk protocol
- distributed coordination
- dead reckoning

Future directions of network development are also described.

17. Key Words ITS, human factors, ergonomics, driving, driving simulators, computer networking		18. Distribution Statement No restrictions. This document is available to the public through the National Technical Information Service, Springfield, Virginia 22161	
19. Security Classify (of this report) none	20. Security Classify (of this page) none	21. No. of pages 28	22. Price

Form DOT F 1700 7 (8-72) Reproduction of completed page authorized

A Description of the UMTRI Driving Simulator Architecture and Alternatives

A3 summary **UMTRI Technical Report 97-17**
Alan Olson and Paul Green
University of Michigan, Ann Arbor, Michigan, USA

Why Use a Networked Configuration?

goal: low cost but reasonable performance

• networking avoids overloading single processor with traffic and scene computations, thus decreasing frame rate
• networking allows for greater code reuse: most of new code is user interface
• use of PCs for processing reduces capital outlay
• PCs are highly reliable, readily repaired, and easy to replace
• low cost, commodity software used
• using the same type of computer for all aspects of the system reduces support cost

Which Network Should Be Used?

Assumptions 1. Token ring and ATM too expensive or not available for Mac
2. otherwise hardware cost is unimportant (<$100), so issue is software

Which Physical and Data Link Layers Should Be Used?

Option	Availability on Mac	Advantages/Disadvantages
serial	2 IEEE-422 ports; can form daisy-chained network (no limit)	least bandwidth; custom protocols; difficult to implement; now used for 1-way connection to IP; needs software to forward messages between ports
LocalTalk	standard	broadcast network, message forwarding not needed
Ethernet √ **(chosen)**	standard; 10Base-T allows up to 8 Macs w/o hub	most bandwidth, broadcast network, message forwarding not needed

Which Protocol Should Be Used?

Option	Advantages	Disadvantages
TCP/IP	widely used (protocol used by the Internet); well documented	no way for simulators to locate one another; drivers regarded as unreliable
EtherTalk √ (AppleTalk on an Ethernet network)	previously used for sound computer; Name Binding Protocol (not in TCP/IP) allows simulators to locate each other	small message size (586 bytes)

How Should Simulators Be Coordinated?

Coordination	Advantages	Disadvantages
Centralized	conceptually simple; less variance in time between state transitions	difficult to implement; sends more messages
Distributed √	easy to implement; sends fewer messages	more variance in time between state transitions; more broadcasted messages

How Should the Position of Vehicles Be Tracked?

Tracking Mechanism*	Summary	Advantages
Continuous updates	each simulator broadcasts an update each frame (30 time/second)	each message is shorter
Dead reckoning √	each simulator broadcasts an update when the actual position begins to differ from that estimated by dead reckoning. In between broadcasts, each simulator uses dead reckoning (using last position and velocity) to estimate the current location of each vehicle.	• generates much lower network load (fewer messages) • smoother motion of traffic

* uncertain which approach generates least processor load

Future Developments

- Multiple vehicles on a single traffic simulator
- Open Transport
- Apple's new operating system

References

Green and Olson, (1997). A Technical Description of the UMTRI Driving Simulator Family-1996 Implementation, Ann Arbor, MI: University of Michigan Transportation Research Institute.

Yoo, H., Hunter, D., and Green, P. (1996). Automotive Collision Warning Effectiveness: A Simulator Comparison of Text vs. Icons (Technical Report UMTRI-96-29), Ann Arbor, MI: The University of Michigan Transportation Research Institute.

PREFACE

This report describes the design rationale for the UMTRI driving simulator. Those interested in the experimental work conducted as a result of the simulator enhancements should see Yoo, Hunter, and Green (1996).

This report is written for developers of other simulators who are considering adding a network capability to their simulators, potential sponsors who need information on UMTRI's capabilities, computer scientists interested in distributed networks, and interested parties at the National Academy of Sciences, Transportation Research Board (project sponsor). This report may also be used by new UMTRI personnel to provide them an introduction to the simulator and by programmers interested in enhancing the current network.

Key contributors to the development of the simulator have been the following:

Paul Green (UMTRI-Human Factors)	served as the project leader
Alan Olson (UMTRI-Human Factors)	implemented the network
Matt Reed (UMTRI-BioSciences)	developed the original graphics
Amitaabh Malhotra (now with Lucent Technologies)	implemented the torque motor control algorithm
Brian Davis (U of Michigan, Electrical Engineering & Computer Science)	developed sound module
Patrick Wei (now with Hewlett Packard)	started development of the torque motor controller
Charles MacAdam (UMTRI-Engineering Research Division)	developed initial vehicle dynamics model
Pat Waller (UMTRI Director)	served as the project advocate
Greg Goetchius (Chrysler, NVH Group)	provided sound recordings of a Chrysler LH
Erik Arthur (University of Minnesota now with Microsoft)	provided ideas concerning vibration

TABLE OF CONTENTS

Introduction

For several years, the Human Factors Division at the University of Michigan Transportation Research Institute (UMTRI) has used a driving simulator to assist in conducting studies of driver behavior. The simulator is capable of showing winding two-lane roads with a dashed centerline, signs, and other objects (as shown in Figure 1), and collecting a variety of driver performance measures (steering wheel angle, throttle position, speed, lane position). Figure 1 shows a typical road scene. Readers interested in a more detailed description of the simulator hardware and software (other than items specific to networking) should see Green and Olson (1997) or MacAdam, Green, and Reed (1993). The Green and Olson (1997) report also lists some of the studies that have been conducted using the simulator.

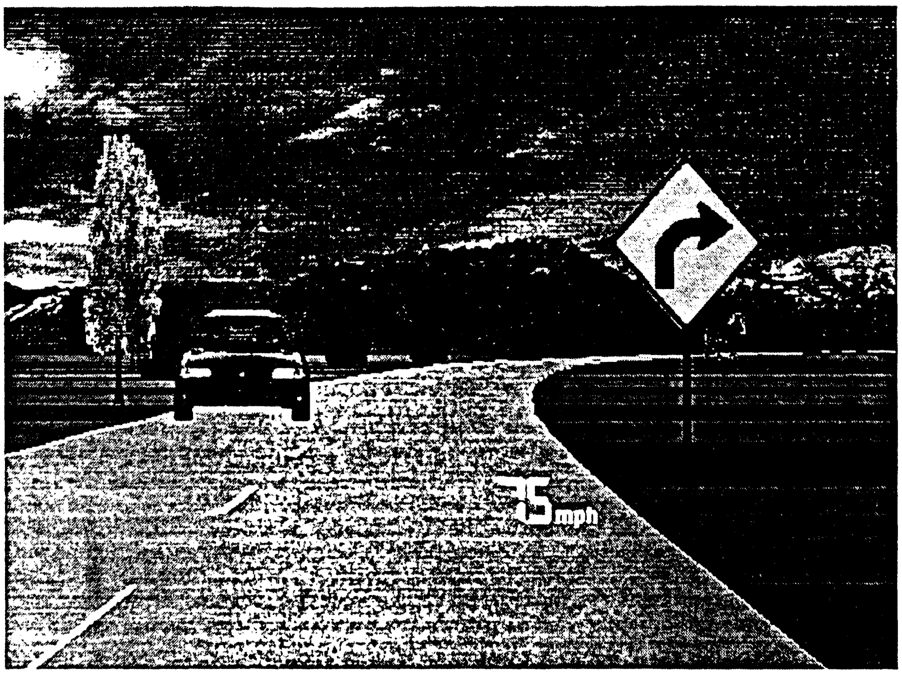

Figure 1. Example road scene with the Head-up display shown.

The simulator was originally developed as a task loader, providing a workload and task structure similar to that of driving. As such, it was used for studies of in-vehicle devices (Reed and Green, 1995), driving workload (Green, Lin, and Bagian, 1993), and simulator design (Davis and Green, 1995). However, both for those studies and for studies of collision avoidance (Yoo, Hunter, and Green, 1996), it was readily apparent the simulator would be improved considerably by the addition of traffic. While previous versions of the simulator did support other vehicles in the scene, the vehicles were static. (For a brief period of time, there was a non-networked version of the simulator with a single lead vehicle with minimal functionality.) Since most crashes involve collisions with other vehicles that are moving, adding traffic to the simulator was a high-priority enhancement. This upgrade greatly expanded the range of useful studies that could be conducted.

Specifically, consideration of the types of studies to be conducted led to the following requirements:

1. The simultaneous simulation of multiple vehicles.

   Traffic often consists of more than just one lead vehicle. Further, vehicles may be oncoming or following. However, simulating more than four or five vehicles is of secondary importance, because interactions with large numbers only occur on expressways with a very large number of lanes under highly congested conditions, situations only experienced by commuters in big cities.

2. The other vehicles are visible in each vehicle's view.

   The subject in an experiment cannot interact with other vehicles unless these vehicles are visible to him or her. Also, in cases where an experimenter is driving one of the other vehicles, the experimenter must be able to see the subject's vehicle to interact with it properly.

3. The view from any vehicle may be displayed.

   It is very difficult to control a vehicle accurately without seeing the view from the vehicle. Thus, the subject must be able to see the view from his or her vehicle, and the experimenters must be able to see the view from any vehicle which he or she might wish to control. It is also possible that some experiments might require that the experimenter monitor the view from a particular vehicle.

4. The views from multiple vehicles may be displayed simultaneously.

   The subject must be able to see the view from his or her vehicle. At the same time the experimenters must be able to see the views from the subject vehicle and the vehicles he or she is controlling.

5. Each vehicle may be controlled either manually or by automatic pilot.

   While most experiments would involve test subjects responding to traffic following a preprogrammed script, there may be experiments in which the interactions of drivers in connected simulators might be of interest.

The remainder of this report answers five questions that influenced the design of the simulator.

1. Why should a networked simulator be used to present traffic?
2. How should tasks be allocated among computers?
3. Which network should be used?
4. How should simulators on different computers be coordinated?
5. How should simulators track the locations of vehicles simulated on other computers?

Why Use a Networked Simulator?

A critical difference between simple and complex simulators used for driving is whether traffic is simulated. In many simulators that provide traffic, traffic is generated on a statistical basis, rather than providing specific directions to a specific vehicle. Further, the computer to control traffic is generally different from other computers (such as those used for experiment control and graphics generation), and the system and software are specialized for modeling traffic. There is little, if any, reuse of the code from the main simulation. While this highly specialized approach provides a high performance simulator, each computer and each software module is unique, complicating maintenance, update, and system reliability. The number of support personnel can be considerable and the applications used for software development are generally not sold in volume. All of these characteristics make such simulators expensive and of diminished reliability.

An alternative approach is a *multiple-vehicle simulator* - a single simulator, running on a single high-performance computer, which simulates the actions of all vehicles. There were three areas of concern. Perhaps the greatest concern at the start of the project was that having a single simulator simulate more than one vehicle would generate too great of a load. The dynamics model for a vehicle is a fairly complex set of equations involving floating-point computations, and doubling (tripling, quadrupling, etc.). This load might cause a noticeable drop in frame rate. As it turned out, floating-point computations on PowerMacs are so fast, and network overhead so great, this concern was misplaced.

A secondary concern was the ability to see the views of and control these additional vehicles. A simulator cannot display more than one road scene without suffering an unacceptable reduction in frame rate, as updating the road scene is the most computationally intensive task in the simulation. Thus, the user can see only the view from one vehicle at a time. Manually controlling a vehicle without seeing its view is very difficult. Using a networked simulator allows the user to see the views from multiple vehicles simultaneously, without reducing frame rate. Networking also makes it possible to manually control each of them.

A tertiary concern was the programming effort required to implement a multiple-vehicle simulator as opposed to a networked simulator. Adding the ability to simulate multiple vehicles (networked or otherwise) would be relatively simple. Further, modifying the graphics routines to display these other vehicles properly in the road scene(s) would not be too difficult. But, a multiple-vehicle simulator would require a far more extensive overhaul of the simulator's user interface than would the networked simulator. Since the user interface accounts for a large part of the simulator source code, the authors concluded that modifying the user interface to make a multiple-vehicle simulator would take more time and effort than writing the network code for the networked simulator.

To provide the desired functionality, the approach selected was to modify the existing simulator to make it a networked simulator. This approach requires multiple computers connected by a suitable network. Each computer runs a simulator, and each simulator simulates one vehicle and displays the view from its vehicle. The simulators use the network to inform one another of the position of their vehicles, and to coordinate actions such as starting and stopping simulation.

To keep costs low, the plan was to use off-the-shelf personal computers with no special-purpose hardware. This meant that capital costs would be low and the components would be highly reliable. Since suitable computers were in wide use in the Division, if a computer ever failed, a replacement could be borrowed from someone's office at a moments notice. In fact, while there were some start-up problems with the simulator during earlier stages of software development, there has **never** been a simulator computer hardware failure over the last four years. This is not the case where special-purpose or high-performance computers are used.

To a large degree, UMTRI's consideration of a networked simulator was due to the distributed interactive simulation program within the U.S. Department of Defense (Simulation Interoperability Standards Organization, 1996). To date, distributed interactive simulation has been of minor interest to the automotive community. The Department of Defense work has led to a standard protocol for exchange of data (Institute of Electrical and Electronics Engineers, 1995) and the widespread use of distributed simulation for training, the evaluation of new weapon systems, battle planning, and other purposes. This program has been successful because it is much less costly to operate simulated systems than real ones, and there are no real casualties.

While simulator developers at UMTRI have adopted many of the conceptual ideas from the government Distributed Interactive Simulation program, program details have not been implemented because the scale of military simulations is much larger (hundreds of players distributed over great distances, support for weapons features (and explosions), multispectral target signatures (IR, radar, acoustic, etc.), and radio and data traffic. Furthermore, because the literature is extensive, the entry cost is high. Nonetheless, just the success of the approach proved to be an important motivator for the effort described in this report.

How Should Tasks Be Allocated?

The primary simulator tasks were:

- handling experimenter interface input and output
- sensing the steering wheel angle and sending commands to the torque motor
- sensing the accelerator and brake pedal positions
- computing the subject's position and orientation based on the steering wheel, accelerator, and brake inputs, road slope, current velocity, and vehicle dynamics
- computing the position and orientation of traffic (other vehicles) on the network
- updating the road scene to reflect the current position of the subject's vehicle, traffic, and changing terrain
- generating the appropriate sounds for the subject's environment (both from their vehicle and other vehicles)
- updating the instrument cluster display
- saving the driving performance data

At the time the simulator was being developed, precise figures on the time required to handle various tasks were not available, so decisions concerning task allocation were based on general impressions and likely future enhancements. Tasks that needed to be tightly time synchronized were put on the same computer. Tasks were distributed if, in combination, they would overload a single computer. Tasks were also grouped if their combination simplified programming.

The most computationally intensive task is updating the road scene, a task whose update rate is very sensitive to the amount of detail in the scene. In fact, at high levels of detail, update rates were unsatisfactory. Hence, to the extent possible, this task was allocated to a dedicated computer. The scene display must be tightly time synchronized with the torque motor controls, which in turn needed to be synchronized with sensing input from the subject. It was logical, therefore, that both should be handled by the same computer. The computational load for the latter tasks was expected to be low.

Processing experimenter input and output and saving the driving performance data were also allocated to the main computer to simplify programming. Since these tasks did not occur while the subject was driving, processor overload (in conjunction with the scene display software) was not a concern.

In many simulators, sound is also handled by the main computer, as this is a task of moderate processing load. However, the UMTRI driving simulator uses multiple channels of sampled stereo sounds, and varies playback rate and volume according to current conditions. The processor load imposed by sound generation is significant enough to warrant allocating a computer specifically for this task.

As a consequence of these decisions, tasks were allocated as shown in Table 1.

Table 1. Task allocation

Computer	Task
main	handle experimenter interface input and output
	sense the steering wheel angle control the torque motor
	sense the accelerator and brake pedal positions
	compute the subject's position and orientation
	update the subject's road scene
	save the driving performance data
traffic (may be more than one computer)	compute the position and orientation of other vehicles
sound	generate sounds from subject's vehicle
instrument cluster	update the instrument cluster

Figure 1 shows the simulator logical network that resulted. Additional details concerning the network design appear in the remainder of this report.

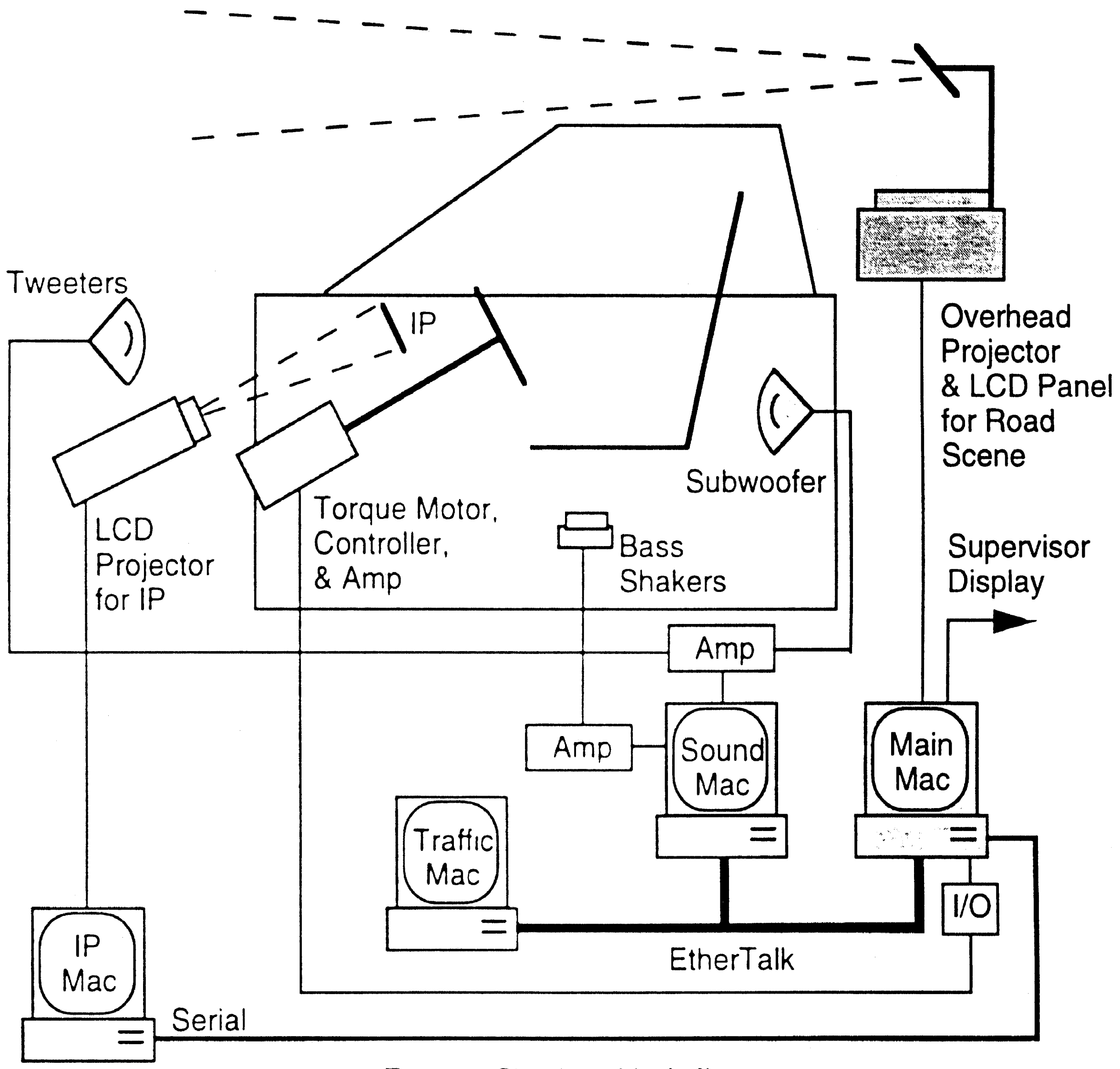

Figure 1. Simulator block diagram

Which Network Should Be Used?

What Should Be Considered in Selecting a Network?

While a network may seem like a single entity to its users, in reality it is a complex collection of interrelated parts. The traditional view is of networks as a series of layers, with each layer adding new functions and capabilities to those of the layers below it. For the purpose of explanation, the network is considered to have three layers.

- The **physical** layer, the lowest layer, consists of cables, connectors, cards, etc. This layer specifies the physical representation of data as it is transmitted (e.g., voltage levels, light pulses).
- The **data link** layer expands the physical layer's ability to transmit data into the ability to communicate between two or more computers. To do this, the data link layer must perform two functions. The first is to control who can transmit and when. For example, in Ethernet networks, a computer with data to transmit will attempt to do so as soon as it sees no other computers are transmitting. If two computers happen to transmit at the same time each will detect a collision, and will wait a random period of time before attempting to retransmit their data. The second function is to specify the format of data transmitted. For example, in LocalTalk networks data is transmitted in packets. The packet starts with a three-byte header identifying which computer sent the data, which computer is to receive the data, and the type of data. The rest of the packet is the data itself, which may be up to 600 bytes long.
- The **protocol** layer expands the data link layer's communication abilities by allowing the linking together of multiple networks, even those of different types. For example, TCP/IP links together a variety of networks all over the world to form the Internet. Protocols also commonly provide a number of high-level communication facilities, which go beyond the simple sending of messages. For example, AppleTalk provides AppleTalk Data Stream Protocol, a facility that allows a stream of bytes to be sent from one computer to another, and guarantees the bytes in the stream will arrive in the order in which they were sent.

These layers are not independent, as the choice for one layer will limit the available choices for the other layers. In particular, there is a very close association between the physical and data link layers. Therefore, the physical and data link layers are chosen as a unit, and the protocol layer is chosen separately.

In making choices, performance was given primary consideration. However, also considered were the availability and reliability of the necessary drivers, as well as any effects on software development. Cost of hardware was not a significant factor in the choice. All the alternatives considered cost less than $100 per computer, and given that software development time was expected to take several weeks at least, hardware costs were expected to be small in comparison to software costs.

Which physical and data link layers should be used?

Although multiple protocols commonly work with a particular data link, multiple data links rarely work with a particular physical layer. Therefore, the choice of physical layer usually determines the choice of data link layer.

The process began with the selection of three physical/data link combinations that were available for Macintosh computers, and which were thought to meet the simulator's needs. These alternatives were then compared on the basis of bandwidth and whether they supported the protocols of interest. One was then selected on the basis of this comparison.

Serial versus LocalTalk versus Ethernet

Three possibilities for the physical and data link layers were considered: serial lines, a LocalTalk network, and an Ethernet network. Other options, such as Token Ring and ATM, were either too expensive or unavailable for the Macintosh.

Using serial lines involves connecting Macintosh computers via their IEEE422 serial ports. Each Macintosh has two such ports, and a network of any number of computers may be daisy-chained together. The physical layer is the serial lines themselves, and the data link layer is either custom built or a standard such as SLIP or PPP. The protocol layer must handle the task of forwarding messages between serial lines when necessary. A serial line was already used for communication between the main simulator computer and the instrument panel computer. Using serial lines for communication between simulators would require this existing implementation be modified and expanded.

LocalTalk is a simple network standard established by Apple Computer and used primarily on Macintosh computers. Any Macintosh may be connected to a LocalTalk network via a LocalTalk transceiver connected to one of its serial ports. The physical layer is either LocalTalk cables and transceivers developed by Apple (which use a custom three-wire cable), or PhoneNet cables and transceivers (which use standard phone cord). There is no difference in performance between them. The data link layer is the LocalTalk Link Access Protocol (LLAP), which is a part of MacOS.

Ethernet is a network standard used with a wide variety of computers. All high-end Macintosh computers, and all those depicted in Figure 1, come with Ethernet on their motherboards. The physical layer may be any of a number of standards, with a variety of performance characteristics. 10Base-T was selected for consideration for three reasons: 10Base-T was already used for communication between the main and sound computers, a daisy-chainable 10Base-T transceiver was available which allowed up to eight computers to be networked without the need for a hub, and the UMTRI building network uses 10Base-T. The data link layer is the EtherTalk Link Access Protocol (ELAP), which is a part of MacOS.

Bandwidth Comparison

Bandwidth (how quickly data may be transmitted) is a primary consideration when selecting the physical/data link layers. They must be able to handle the expected message load, and should have some extra capacity for the inevitable upgrades and new features. Of the three options, serial lines had the least bandwidth and Ethernet the most.

Macintosh serial ports are normally only capable of speeds up to 57,600 bits per second (bps), but newer models with serial DMA (Direct Memory Access) can achieve serial speeds of 115,200 bps or even 230,400 bps. However, these high speeds place a considerable demand on the software, and great care must be taken to produce an efficient implementation. The effective bandwidth is also somewhat less than these theoretical maxima due to delays encountered when a message must be forwarded between serial ports.

LocalTalk transmits at a rate of 230,400 bps. However, carrier sense, collisions, and protocol overhead reduce the effective bandwidth from this maximum. LocalTalk is a broadcast network, so any transmission is seen simultaneously by all computers on the network. Therefore, there is no need for any computer to forward messages, as with serial connections.

Ethernet on 10Base-T transmits at a rate of 10,000,000 bps. However, carrier sense, collisions, and protocol overhead reduce the effective bandwidth, just as with LocalTalk. A general rule of thumb with Ethernet is that effective maximum bandwidth is one-third theoretical maximum bandwidth. Ethernet is also a broadcast network, so there is no need for forwarding of messages.

Protocol Support

Different types of networks support different protocols. Of specific interest were AppleTalk (a protocol suite developed by Apple Computer), and TCP/IP (the protocol suite used on the Internet), since MacOS offers support for both of these protocols.

Serial communications generally use custom protocols. All parties agree on how data is formatted, and the format is specific to the task at hand. It is possible to use TCP/IP with serial lines with the assistance of either SLIP or PPP. IP routing could then be used to solve the problem of forwarding messages between serial lines. However, at the time this approach was first considered there were few reliable Macintosh implementations of SLIP or PPP, and Macintosh implementations of TCP/IP did not allow the computer to serve as a router.

LocalTalk was originally designed to support AppleTalk, but can also support TCP/IP. There was some concern over LocalTalk's relatively small, 600-byte packet size. The protocol must break up long messages into smaller pieces which can fit into a LocalTalk packet. The process of breaking up long messages (called packetization) and reassembling them at the receiving end (called de-packetization) adds considerable overhead. However, AppleTalk always breaks up messages into pieces no larger than 586 bytes, and TCP/IP implementations are not required to accept messages longer than 576 bytes. So any message longer than a LocalTalk packet would likely be packetized by both AppleTalk and TCP/IP, no matter what physical and data link layers were being used.

Ethernet is often used with TCP/IP, but it can be used with many different protocols, including AppleTalk. AppleTalk running on Ethernet is usually called EtherTalk in order to distinguish it from AppleTalk running on LocalTalk.

Why Ethernet Was Selected

The most important points in favor of Ethernet were high bandwidth and support for the protocols of interest. Also, a 10Base-T Ethernet could be plugged into the UMTRI network to allow access to the outside world. This would make file transfers between office and simulator computers possible, and would allow for backups of simulator computers.

LocalTalk was thought to be marginal, at best, in terms of bandwidth, and it offered nothing that was not available with Ethernet. Finally, while both Ethernet and serial lines were already in use, there was no existing LocalTalk network. Adding a LocalTalk network would introduce yet another type of hardware/cable, complicating support.

Serial lines had too little bandwidth, and also were difficult to implement. Using serial communications for all networking would also require revising the existing communications software linking the main simulator computer and instrument panel computer. Finally, there were considerable doubts about the availability of either SLIP or PPP, meaning a custom data link (and probably a custom protocol) would have to be written, greatly increasing development time.

Which Protocol Should Be Used?

Protocols, like networks, are built in layers. The lowest layer, called (confusingly) the network layer, interacts with the data link layer below it and provides a basic service for sending, receiving, and routing messages. In the layers above the network layer are various services, each building on one or more of the services in layers below it.

The process of selecting a protocol began by identifying two protocols that were thought capable of doing what was required and which were supported by MacOS. These protocols were then compared on the basis of the services they offered, and their stability and reliability under MacOS. One was then selected on the basis of this comparison.

TCP/IP versus EtherTalk

Given the choice of Ethernet for the network, there are two protocols to choose from: TCP/IP and EtherTalk. EtherTalk is used for communication with the sound computer, but since a single Ethernet network can carry both TCP/IP and EtherTalk traffic, this was not a major factor in the decision.

TCP/IP is the protocol used by the Internet. TCP/IP works well in both large and small networks. TCP/IP is widely used and well documented.

EtherTalk is the name given to AppleTalk running on an Ethernet network. AppleTalk is limited to networks of a few thousand computers, and is designed for small LANs. EtherTalk is used primarily by Macintosh computers, and Apple Computer is the main source of documentation.

Protocol Services

The services the simulator requires from the protocol are fairly basic: fast delivery of messages with minimal overhead, and some means of locating and contacting other simulators on the network. Since the network is small, and there are no routers or gateways, there is little chance of message loss, and therefore no need for guaranteed message delivery and its associated overhead.

User Datagram Protocol (UDP) is the TCP/IP service that best fits the message delivery needs of the simulator. It is not the network level service (IP is), but it has reasonably low overhead. The simulator must allocate a UDP port, and all messages for the simulator are sent to that port. There is no way to communicate with the simulator unless its port number is known. Therefore, all simulators must agree on what port number they will use, and each must be able to allocate that port number on its computer. Each simulator must then broadcast to that port number to see what other simulators are currently running. If another program allocates the port number used by the simulator, it will not only prevent the simulator from running on the same computer, it will likely be confused by the messages from other simulators.

Datagram Delivery Protocol (DDP) is the AppleTalk service that best fits the message delivery needs of the simulator. It is AppleTalk's network level service, and is therefore low level and low overhead. The principle drawback of DDP is its maximum message size of 586 bytes. The simulator must allocate a DDP socket (analogous to a UDP port), to which all messages are sent. However, unlike TCP/IP, AppleTalk provides a service that allows one to determine which sockets on which computers are assigned to which programs. By registering with the Name Binding Protocol (NBP), a simulator can advertise its network address and socket number to other simulators on the network. Simulators still try to get the same socket number so they can use broadcasting, but they can still locate and cooperate with one another even if they can't.

Protocol Stability and Reliability

Support for a particular protocol is not enough. The software that provides this support must be relatively stable and bug free. An unstable implementation may change each time the operating system is updated, causing previously working programs to fail. A bug-riddled implementation can lengthen software development time as work-arounds for the various bugs are developed. It can also halt development entirely if a critical service is not working.

At the time this work was started, Apple was in the process of creating a new network framework for MacOS. TCP/IP support, which had always been troublesome, had suffered greatly under this new framework, and there were numerous reports of bugs and unimplemented services.

AppleTalk support was also fit into Apple's new network framework. However, AppleTalk came off far better than TCP/IP, perhaps because the AppleTalk software was simply ported to the new framework, while the TCP/IP software was largely rewritten from scratch. There were a few reported bugs, but none that affected services the simulator used.

Why EtherTalk Was Selected

The most important reasons for selecting EtherTalk were the availability of NBP and fears about the stability of TCP/IP. Using EtherTalk also allowed for some code reuse between the functions that communicated with the sound computer, and those that communicated with other simulators. Finally, an EtherTalk implementation allows simulators to proceed even though they cannot allocate the desired socket number, and does not need to use broadcasts to find other simulators. This is not of major importance in the isolated lab environment, but is much more robust in the office environment where simulator development and testing are done.

TCP/IP suffered primarily from a buggy implementation. There were also concerns about the procedures necessary to locate other simulators. While these procedures are unlikely to cause problems in the lab, they could cause problems in an office environment.

How Should Simulators Be Coordinated?

Coordination is the problem of synchronizing the starting and stopping of simulations on different simulators. Synchronization of the start of simulation is necessary to ensure that the elapsed times recorded by different simulators are comparable. Synchronization of the stopping of simulation relieves the experimenter of the task of stopping each simulation manually.

The process of selecting a coordination method began by generating a careful definition of what it meant to be coordinated and how coordination was to be specified. Then, two promising coordination methods were selected. They were then compared on various performance measures and ease of implementation. One of the methods was then selected on the basis of this comparison.

How is Coordination Defined?

To better define coordination, it is necessary to divide simulator operation into three states (Table 2), which the simulator moves between.

Table 2. Simulator States

State	Description
Stopped	The simulator is not running its simulation.
Waiting-to-run	The simulator wishes to run its simulation, but is waiting until other simulators are either waiting to run or running their simulations.
Running	The simulator is running its simulation.

The problem of coordination is therefore the problem of ensuring the simultaneous transition of simulators from waiting-to-run to running, and from running to stopped. In addition, some nonsimultaneous transitions need to be allowed for as well. (e.g., a simulator that starts running after all the others and stops before they do.)

To control coordination of transitions between these states, a simple scheme was developed. Four properties were defined, of which a simulator may have all, none, or any subset. The user decides which simulators have which properties, and careful selection of which simulators have which properties can produce practically any desired behavior. The properties and how they affect transitions follow (Table 3).

Table 3. Transition Properties

Property	Description
Start independence	The simulator immediately makes the transition from <u>waiting-to-run</u> to <u>running</u> upon being told to do so by the user.
Start veto	The simulator prohibits other simulators which do not have start independence from making the transition from <u>waiting-to-run</u> to <u>running</u> until the simulator is <u>waiting-to-run</u> or <u>running</u>.
Stop independence	The simulator makes the transition from <u>running</u> to <u>stopped</u> only when told to do so by the user.
Stop control	The simulator forces other simulators which do not have stop independence and are <u>running</u> to immediately make the transition to <u>stopped</u> when the simulator makes the transition from <u>running</u> to <u>stopped</u>.

In most cases, all simulators will start and stop their simulations simultaneously, and therefore all simulators will have start veto and stop control and, no simulators will have start independence or stop independence. A simulator that starts its simulation before other simulators needs start independence. A simulator that starts its simulation after other simulators should not have start veto. A simulator that stops its simulation after other simulators needs stop independence. A simulator that stops its simulation before other simulators should not have stop control.

Deciding when a simulator starts running its simulation is simple. A simulator makes the transition from stopped to waiting-to-run when the user tells it to start its simulation. If the simulator has start independence it immediately makes the transition from waiting-to-run to running. Otherwise, a check is done to see if any other simulators have start veto. If none do, or if all that do are waiting-to-run or running, the simulator makes the transition from waiting-to-run to running. Otherwise, the simulator waits until the preceding condition is satisfied.

Deciding when a simulator stops running its simulation is also simple. A simulator always makes the transition from running to stopped if told to do so by the user. If a simulator does not have stop independence it must also monitor the states of other simulators. If it sees a simulator with stop control make the transition from running to stopped, then it also makes the transition from running to stopped.

Centralized versus Distributed Coordination

The problem of coordination is then one of ensuring that state transitions that should be simultaneous are simultaneous (or nearly so). Two coordination methods were considered: centralized coordination and distributed coordination.

Centralized coordination is, on the surface, the simplest solution. A single simulator is the master, and all the remaining simulators are slaves. Simulators make the transition from stopped to waiting-to-run when told to do so by the user, but only make transitions from waiting-to-run to running and running to stopped when told to do so by the master. Simultaneity is achieved through broadcasted messages from the master ordering state changes for one or more simulators.

Distributed coordination is more complex. Each simulator is responsible for deciding for itself when to change state. Simulators make the transitions from stopped to waiting-to-run and from running to stopped when told to do so by the user. Simulators also broadcast their state changes to the rest of the simulators so that each simulator is aware of the current state of the others. Each simulator uses this information to determine when to make the transition from waiting-to-run to running, and whether it should make the transition from running to stopped. Simultaneity is achieved by the broadcasted change of state messages.

Performance and Implementation

Centralized and distributed coordination achieve the same ends by two very different means. The choice between them was made on the basis of how well they did their job, how much network load they generated, and how much programming effort would be required. To this end, the two approaches were compared in three areas: simultaneity of state transitions, network traffic generated, and ease of implementation.

The maximum time difference between two "simultaneous" state transitions is the same for either centralized or distributed coordination, although the variance in time difference is slightly higher for the distributed case. This is because in both cases the state transitions are in response to a broadcasted message. Simulators check for new messages once each frame and there are 30 frames per second, so each simulator will see the message and perform the state transition within 1/30th of a second (assuming message send time is negligible). With centralized coordination the state transitions occur randomly throughout the 1/30th-of-a-second interval. With distributed coordination one of the state transitions occurs at the beginning of the interval, and the remaining occur randomly throughout, resulting in slightly more variance in the time difference between state transitions.

The total network traffic is somewhat greater for centralized coordination than it is for distributed coordination. However, centralized coordination generates fewer broadcasted messages, so most simulators have fewer messages to process. This is because the only broadcasted messages with centralized control are those ordering state transitions, while with distributed control all state transitions result in a broadcasted message. Since each broadcast from the master can order more than one state transition, centralized coordination has fewer broadcasts, and slaves will process fewer messages than they would with distributed coordination. Centralized coordination produces more messages, however, because change of state messages are still sent, but only the master must process them.

Centralized coordination is significantly more difficult to implement than distributed coordination. The first problem is selecting the master, and informing all the slaves who the master is. The master must check for messages constantly, so any lengthy operation must be interrupted regularly. Finally, there must be two procedures developed for determining state changes: one for the master and one for the slaves. With distributed coordination all simulators use the same procedure to determine state changes. Furthermore, this procedure is easier to implement than the procedure for the master simulator, since it must decide for only one simulator instead of every simulator.

Why Distributed Coordination Was Selected

The primary reason for selecting distributed coordination was ease of implementation. The greater variance in time difference between simultaneous state transitions was of some concern, but it was deemed small enough to be acceptable. The increase in message broadcasts was not considered significant, since they only occur at state transitions, and state transitions are relatively rare.

Centralized coordination was judged to be too difficult to implement. Since software development was expected to be the major expense (both in time and money) for this project, anything that would make software development more difficult would have to be justified by significant performance advantages. Centralized coordination had no significant performance advantages.

How Should the Position of Vehicles Be Tracked?

Tracking is the problem of a simulator knowing at all times the current position of not only its own vehicle, but of the vehicles of other simulators as well. Without this information, each simulator would be unable to display these vehicles in its view of the world, and users of the simulator would be unable to interact with them.

The principle problem in tracking the position of another simulator's vehicle is that a simulator may only know where the vehicle was, not where it is. As time passes, knowing where the vehicle was is less and less helpful because it has almost certainly moved and is now somewhere else. Tracking, then, comes down to the problem of using what is known about where a vehicle was to determine where it is, at least in approximation.

The process of selecting a tracking method began by identifying two tracking methods commonly used in distributed simulations. These were then compared on the basis of several performance measures and ease of implementation. One was then selected.

Continuous Updates versus Dead Reckoning

Either of the two general approaches commonly used in distributed simulations for tracking vehicle position, continuous updates and dead reckoning, could be easily adapted for use with the UMTRI driving simulator. Continuous updates operates on the principle that if where a vehicle was just a very short time ago is known, then the difference between where it was and where it is too small to matter. Once each frame, immediately after the simulator has updated a vehicle's position, the simulator broadcasts the new position to every other simulator. The position of other simulator's vehicles is taken to be their most recently reported position.

Dead reckoning operates on the principle that if one knows where the vehicle was and which direction the vehicle was going, one can make a good estimate of where the vehicle is. Each simulator not only calculates the current position of its vehicle, but also an estimated position. When the distance between the actual and estimated positions exceeds some threshold, the simulator broadcasts a message that contains not only the vehicle's actual position, but dynamics information such as speed, acceleration, heading, turn rate, etc. All simulators (including the one that sent the message) use this information to estimate the current position of the vehicle.

Performance and Implementation

Both continuous updates and dead reckoning have been used successfully in other programs. Continuous updates is used by most multiplayer action games (e.g., Doom.) Dead reckoning is used in Distributed Interactive Simulation (DIS), the distributed simulation standard developed and used by the US military. To evaluate alternative approaches for tracking vehicles for use with the UMTRI driving simulator, they were compared in four areas: network load, processor load, how smoothly vehicles appear to move, and implementation.

Continuous updates generates a much higher network load than dead reckoning. Continuous updates require each simulator broadcast once each frame, or 30 times a second. Dead reckoning broadcasts longer messages, but usually only has to do so a few times a second. Approximate message size can be computed to get a feel for the difference in network load. Both methods have to send messages containing a five-byte DDP header plus 16 bytes for vehicle position and heading. In addition, dead reckoning must send speed, acceleration, and turn rate values, all 4-byte quantities. Therefore, continuous updates sends 21-byte messages versus 33-byte messages for dead reckoning. Since dead reckoning was expected to reduce the number of messages by a factor of a least five, it reduces network load significantly over continuous updates.

Whether continuous updates or dead reckoning causes a greater load on the processor is uncertain. Continuous updates require no computation to determine the position of other simulator's vehicles, while dead reckoning requires that a new estimated position be computed each frame. However, continuous updates requires the processing of far more messages, each requiring processor time.

The apparent movement of vehicles is not as smooth with continuous updates as it is with dead reckoning. Simulator frame rates vary somewhat, and frames are not synchronized between simulators. Therefore, with continuous updates a simulator may not receive any position updates from another simulator during one frame and could receive two updates during the next frame. The result is an apparent "freeze" in the motion of the vehicle, followed by a "jump." With dead reckoning the estimated position of the vehicle is calculated for each frame and thus appears to move smoothly. There is some danger of a discontinuity when a new position update is sent out, but this is minimized by sending out position updates before the error in estimated position gets large.

The continuous updates approach is easier to implement than dead reckoning. A continuous updates approach only requires that position updates be broadcast each frame and that the most recently updated positions be used for each vehicle. Dead reckoning requires dynamic information be sent with each position update. Further, each frame the position of each vehicle must be estimated. A simulator must also compare the estimated position of its vehicle with the actual position calculated from the dynamics model, and send out updates if the estimated position is too far off.

Why Dead Reckoning Was Selected

The primary reasons for selecting dead reckoning were reduced network load and smoother movement, a critical characteristic if useful data on driver performance and behavior is to be obtained. The added implementation effort was deemed justified in light of these advantages.

Continuous updates was expected to use too much network bandwidth. This was seen as especially critical early in development, before it became clear how much bandwidth was available and how much message traffic each simulator could handle.

Summary

The need to study the interaction between multiple vehicles, especially for collision avoidance research, prompted the development of a networked version of the UMTRI Driving Simulator. In this version, simulators running on separate computers connected by a network interact to produce the illusion of a single simulation involving multiple vehicles.

In producing this version, problems faced included what network and protocol to use, how to coordinate the simulators, and how simulators were to track the vehicles of other simulators. Several solutions were considered for each of these problems. Choices were made with an eye towards performance, flexibility, and ease of implementation. The solutions chosen were:

- 10Base-T Ethernet using AppleTalk protocol
- distributed coordination
- dead reckoning

The networked simulator has been in operation since fall 1995. The simulator has worked well, both on the isolated network in the simulator laboratory and on the office network where there is a great deal of other traffic (where development takes place.) This experience validates the choices made.

Future Developments

Since network upgrade began there have been several changes to the Macintosh operating system, and with Apple Computer itself, which may affect future work.

- *Multiple-vehicle traffic simulator.* While allowing the main simulator to simulate more than one vehicle was originally considered as an alternative to a networked simulator, the idea was rejected because of processor load concerns. However, future upgrades will allow any simulator to simulate multiple vehicles. A primary concern is cost reduction: the number of computers needed is no longer equal to the number of vehicles. The restrictions on multiple vehicle simulators mentioned above still apply: only one vehicle's view may be displayed, and all but one vehicle must be on automatic pilot. Rarely is more than one manually controlled vehicle needed.

- *Open Transport.* When this latest simulator upgrade began Apple was in the process of replacing its network framework with Open Transport. Open Transport was not used for the simulator because it was considered unstable. It has now stabilized, and if this project had been started in 1997 instead of 1995, Open Transport would have been used instead of the "classic" Macintosh networking.

- *Apple's new operating system.* At the end of 1996, Apple Computer acquired NeXT Inc., and announced plans to produce a new Macintosh operating system based on NeXT's OpenStep. OpenStep is Unix-based and comes with a TCP/IP stack, but does not currently support AppleTalk. An AppleTalk stack is supposed to be ready by the time the new operating system ships, but given the time constraints it is doubtful the AppleTalk stack will be reliable, if it ships on time. These concerns about future AppleTalk support might have led to a decision to use TCP/IP for a network protocol.

References

Davis, B.T. and Green, P. (1995). Benefits of Sound for Driving Simulation: An Experimental Evaluation (Technical Report UMTRI-95-16), Ann Arbor, MI: The University of Michigan Transportation Research Institute.

Green, P. and Olson, A. (1996). Practical Aspects of Prototyping Instrument Clusters, (SAE paper 960532), Warrendale, PA: Society of Automotive Engineers.

Green, P. and Olson, A. (1997). A Technical Description of the UMTRI Driving Simulator Family (Technical Report UMTRI-97-12), Ann Arbor, MI: University of Michigan Transportation Research Institute.

Institute of Electrical and Electronics Engineers (1995). Standard for Distributed Interactive Simulation - Application Protocols (IEEE Standard 1278.1-1995), New York: Institute of Electrical and Electronics Engineers.

MacAdam, C.C., Green, P.A., and Reed, M.P. (1993). An Overview of Current UMTRI Driving Simulators, UMTRI Research Review, July-August, 24(1), 1-8.

Simulation Interoperability Standards Organization (1996). 15th Workshop on Standards for the Interoperability of Defense Simulations (Technical report IST-CF-96-01.1), Orlando, FL: University of Central Florida, Institute for Simulation and Training.

Yoo, H., Hunter, D., and Green, P. (1996). Automotive Collision Warning Effectiveness: A Simulator Comparison of Text vs. Icons (Technical Report UMTRI-96-29), Ann Arbor, MI: The University of Michigan Transportation Research Institute.