Real-Time Event Channel Performance on a Submarine Communications Network

SUMMARY

This project evaluated the real-time event channel of the TAO software when running under the Sun Solaris 8 operating system. We modeled the communications channel among the RTS, NAV, and SONR units taken from the shipboard network architecture design for Virginia-class submarines. We established a baseline set of performance measurements for throughput, latency and jitter, communicating over UDP, TCP, and CORBA, running over Fast Ethernet , Gigabit Ethernet, and ATM OC3. Then we evaluated the performance of the CORBA event channel in the NAV data distribution scenario, including tests of the Real-Time Event Channel and the real-time features of Solaris 8. We conducted hundreds of simulations under a variety of conditions, including process-to-process communication on one machine, process-to-process with network loopback, machine-to-machine over the three networks, with and without highly loaded CPUs, with and without additional network traffic, and with and without utilizing the real-time event channel. Our most significant results were the documentation of CORBA overhead, the cost of CORBA data marshaling, and the throughput and latency measurements, using both loaded and unloaded CPUs, over Fast Ethernet, Gigabit Ethernet, and ATM OC3 networks.

Real-Time Event Channel Performance on a Submarine Communications Network

Scott W. Talbert
Department of Computer Science
University of Virginia
151 Engineer's Way
Charlottesville, VA 22904 U.S.A.
weaver@virginia.edu

Alfred C. Weaver
Department of Computer Science
University of Virginia
151 Engineer's Way
Charlottesville, VA 22904 U.S.A.
talbert@virginia.edu

ABSTRACT. This project evaluated the real-time event channel of the TAO software when running under the Sun Solaris 8 operating system. We modeled the communications channel among the RTS, NAV, and SONR units taken from the shipboard network architecture design for Virginia-class submarines. We established a baseline set of performance measurements for throughput, latency and jitter, communicating over UDP, TCP, and CORBA, running over Fast Ethernet , Gigabit Ethernet, and ATM OC3. Then we evaluated the performance of the CORBA event channel in the NAV data distribution scenario, including tests of the Real-Time Event Channel and the real-time features of Solaris 8. We conducted hundreds of simulations under a variety of conditions, including process-to-process communication on one machine, process-to-process with network loopback, machine-to-machine over the three networks, with and without highly loaded CPUs, with and without additional network traffic, and with and without utilizing the real-time event channel. Our most significant results were the documentation of CORBA overhead, the cost of CORBA data marshaling, and the throughput and latency measurements, using both loaded and unloaded CPUs, over Fast Ethernet, Gigabit Ethernet, and ATM OC3 networks.

I. IMPORTANCE OF REAL-TIME DATA

In a naval combat environment, there are many different interacting entities, including submarines, surface ships, and satellites. These entities must coordinate in real-time to make split-second decisions to defend from an attack or to launch an offensive. Inside a naval submarine, for example, the ship's navigation information (position, heading, speed, etc.) must be known consistently and to a high degree of accuracy throughout the ship. Otherwise, a missile launch could be compromised or the ship may be unable to position its antenna to communicate with the rest of the battle group or the central command. In a Virginia-class submarine, the navigation information is distributed periodically via a computer network system. Therefore, it is essential that we have prior knowledge of the exact communications characteristics of the underlying system, including not only the properties of the network itself, but also the characteristics of the CPUs and software involved in the communications.

In a Virginia-class submarine, the navigation information is generated in the NAV (navigation unit, see Fig. 1) and sent to the RTS (Real-Time Subsystem) unit for distribution to the rest of the ship. The RTS subsystem uses a CORBA Event Channel to distribute the navigation messages periodically. Seventeen destinations receive NAV messages at 16 Hz, while another eighteen destinations receive NAV messages at 1 Hz. There is a design constraint that NAV messages cannot be received more than 50 milliseconds late; otherwise, a missile launch may be compromised, or an antenna may not be properly aligned.

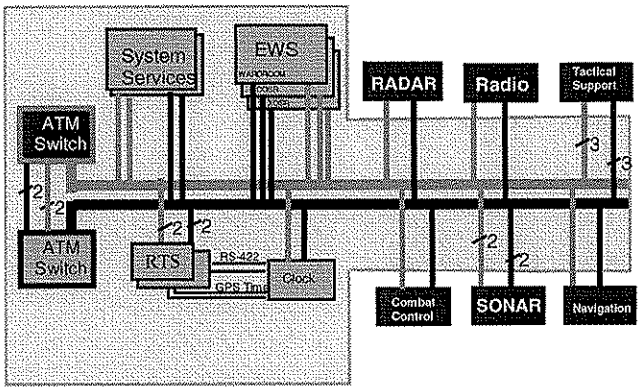

Fig. 1. Simplified NAV Architecture

In this project, our task was to investigate the communications characteristics of shipboard network technologies, including The ACE ORB (Object Request Broker), Solaris 8, and Fast Ethernet / Gigabit Ethernet / ATM-OC3 in the context of the NAV data distribution scenario. From the perspective of the Navy, it is highly desirable that these technologies are constructed from commercial, off-the-shelf hardware and software [1]. First, we established a baseline set of measurements to understand the fundamental properties of the systems. We measured latency, throughput, and jitter for each of the communications protocols (UDP, TCP, and CORBA) and networks (Fast Ethernet, Gigabit Ethernet, and ATM) employed. Next, we quantified the impact of CORBA's data structure marshalling by measuring latency and throughput for transactions using various data structures. Finally, we evaluated the real-time features of The ACE ORB (TAO) and Solaris 8 by comparing real-time scenarios with non-real-time scenarios using assorted network configurations and CPU loads.

II. NETWORK HARDWARE AND SOFTWARE ENVIRONMENT

During the project, we explored three key technologies: TAO, Sun's Solaris 8 operating system, and ATM-OC3/Gigabit Ethernet off-the-shelf networks. Widely used in industry for distributed applications, TAO [2] is an open-source C++ CORBA ORB developed at Washington University in St. Louis. It is compliant with the OMG CORBA 2.6 specification [3] and the Real-Time CORBA 1.0 specification [4]. TAO version 1.1 was used for the baseline experiments, while version 1.2 was used for the real-time experiments. Solaris 8 is a general-purpose UNIX operating system from Sun Microsystems. It is not a real-time operating system, but it does include some real-time extensions that we evaluated in this project. Finally, we used commercial off-the-shelf Gigabit Ethernet and ATM OC3 network equipment in our testing: a 3Com 9300 Gigabit Ethernet switch and a Fore Systems ASX2000 ATM switch. The current Virginia-class submarine architecture uses a hybrid ATM network, as shown in Fig. 1, and Lockheed Martin is currently evaluating Gigabit Ethernet to replace ATM in some instances. In the original plan for our project, Lockheed Martin was to build a Gigabit Ethernet test network, similar to the ATM network in the diagram, where we would run our tests and simulations. Unfortunately, due to the recent economic downturn, this plan was delayed, so instead we used existing university resources to build a subset of the shipboard network. We used two Sun Enterprise 250 Servers (each with dual 400 MHz processors, 512MB of RAM, and Sun Gigabit Ethernet and ATM network interface cards), along with the ATM switch and the Gigabit Ethernet switch. From this equipment we constructed a simulation of the sonar and RTS units that run on the real combat network.

III. BASELINE PERFORMANCE MEASUREMENTS

In performing what we called our *baseline* experiments, we sought to understand the fundamental properties of the underlying networks, protocols, and systems of the shipboard network. By doing so, we created a data book that can be used for comparison with all future experiments. In conducting the baseline experiments, we began with a single machine, first measuring each of its properties, and then moved to multiple machines connected via a network. For each network (Fast Ethernet, Gigabit Ethernet, and ATM OC3) and protocol (UDP, TCP, and CORBA), we measured message latency, throughput, and jitter for various message sizes. While these experiments resulted in several hundred different performance graphs, we report here only the three most interesting subsets. First, we compared the performance (latency and throughput) of Fast Ethernet, Gigabit Ethernet, and ATM networks. Second, we quantified the overhead associated with using CORBA (specifically TAO) as a middleware layer. Third, we documented the impact of data structure marshalling in CORBA applications for different types of data structures.

A. Throughput and Latency

Our first result is a comparison of the performance of Fast Ethernet, Gigabit Ethernet, and ATM networks. Fig. 2 depicts CORBA message latency, the amount of time needed to send a message of the specified size through the CORBA ORB. Each data point shown is the average of 1,000 trials. On average, the Fast Ethernet transaction took approximately 300 microseconds longer than the Gigabit Ethernet and ATM transactions. The Gigabit Ethernet's somewhat erratic behavior is due to other (non-experiment) traffic on our public Gigabit Ethernet network.

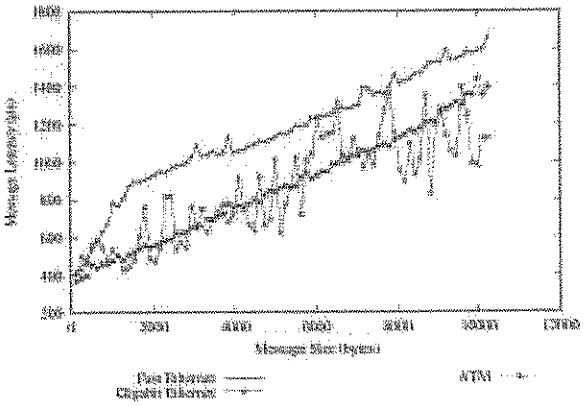

Fig. 2. CORBA Message Latency Over Three Networks

Fig. 3 shows the TCP throughput for the three networks. The ATM and Fast Ethernet networks nearly achieve their maximum theoretical throughputs at 130 (out of 155) Mbps and 90 (out of 100) Mbps, respectively. On the other hand, the Gigabit Ethernet network is only able to achieve roughly 27% (270 out of 1000 Mbps) of its theoretical maximum throughput. This occurred because the CPUs of the machines involved simply could not process data any faster.

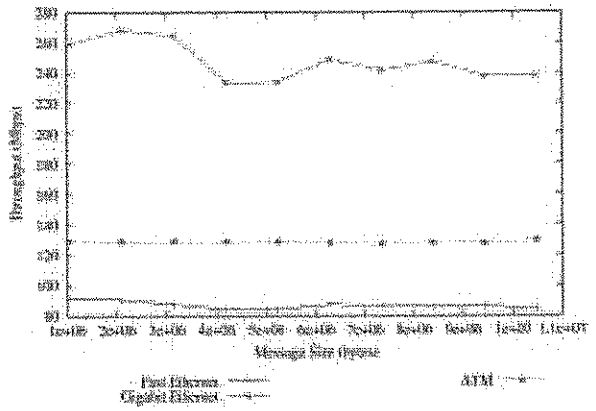

Fig. 3. TCP Throughput Over Three Networks

Also, the small frame size (1,500 bytes) of the Gigabit Ethernet network was another limiting factor; although the frame size of 1,500 bytes was chosen for system

compatibility reasons, a larger frame size would have proven more efficient.

Fig. 4 displays the CORBA throughput for the three networks. Here we see significant reductions from the previous throughput levels in Fast Ethernet (85 vs. 90 Mbps), ATM (115 vs. 130 Mbps), and Gigabit Ethernet (235 vs. 270 Mbps).

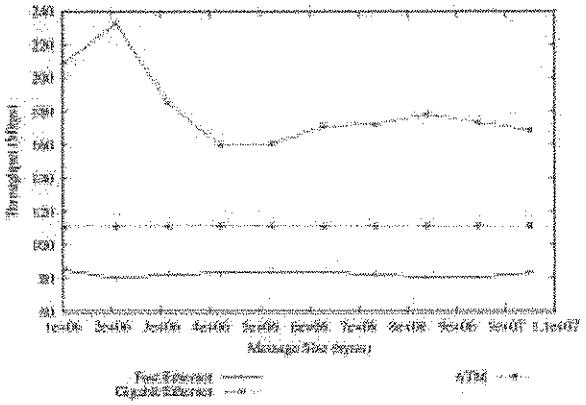

Fig. 4. CORBA Throughput Over Three Networks

B. CORBA Overhead

In our next set of baseline results, we quantified the overhead associated with using the TAO CORBA ORB as a middleware layer by comparing CORBA transactions with UDP and TCP transactions. Fig. 5 shows UDP and CORBA message latency for various message sizes. On average, the CORBA transaction takes about 300 microseconds longer than a UDP transaction of the same size.

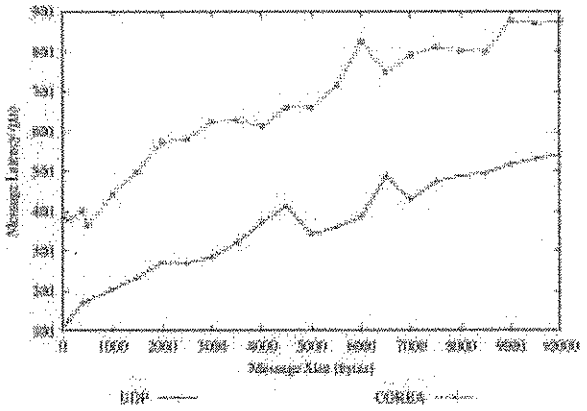

Fig. 5. UDP vs. CORBA Message Latency

Fig. 6 shows TCP and CORBA throughput for various message sizes. On average, raw TCP is 65 Mbps faster than CORBA—the CORBA transactions incur approximately a 27% reduction in throughput due to CORBA overhead.

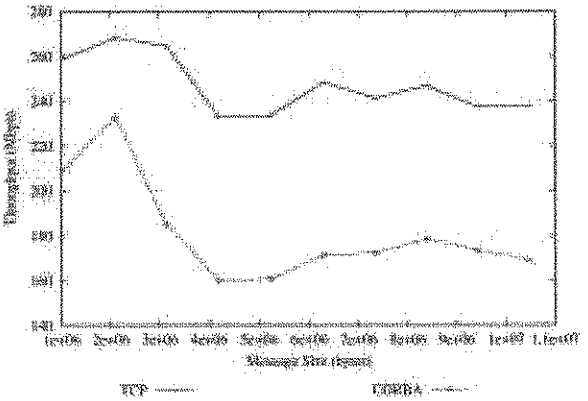

Fig. 6. TCP vs. CORBA Throughput

C. Parameter Marshaling

In our third set of baseline results, we document the impact of using different types of data structures in CORBA transactions. In these experiments, we measured the latency and throughput of CORBA transactions using two different sets of IDL, "Sequence" and "Struct," as shown in Fig. 7. The Sequence IDL is a very simple sequence of characters, while the Struct IDL, on the other hand, is a sequence of complex structures, and includes many different data types.

Simple Sequence

```
typedef sequence<char>
        SequenceMessage_;

interface Echo {
 void processSequenceMessage(inout
        SequenceMessage_ m);
};
```

Complex Struct

```
struct ComplexStruct {
  short shortVal;
  long longVal;
  float floatVal;
  double doubleVal;
  char charVal;
  string<13> stringVal;
}; // ComplexStruct: 32 bytes

typedef sequence<ComplexStruct>
        StructMessage_;

interface Echo {
void processStructMessage(inout
        StructMessage_ m);
};
```

Fig. 7. Simple vs. Complex Data Structure

Fig. 8 shows the message latency for transactions of both IDL types. On average, the Struct transaction took approximately 2,000 microseconds longer than the Sequence transaction of the same size. In addition, the Struct transaction's latency growth rate is much greater than that of the Sequence transaction.

Fig. 9 shows the throughput measurements for the two types of transactions. The Sequence message throughput was approximately 181 Mbps, while the Struct throughput was only 26.6 Mbps, a mere 15% of the Sequence throughput. Clearly, the ORB is spending many CPU cycles marshalling the parameters of the Struct transmissions and throughput is reduced to a very low level. This was a major revelation that, had it not been discovered in this project, might have escaped a communications systems designer.

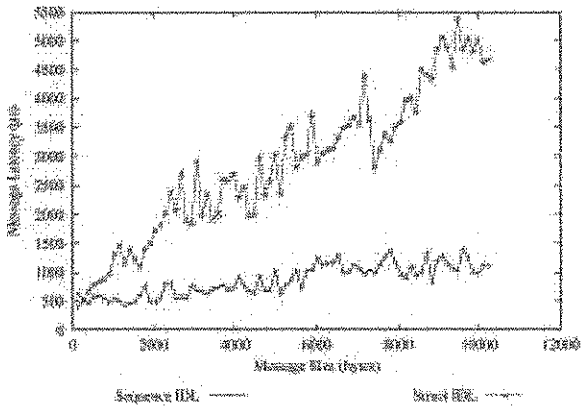

Fig. 8. Simple vs. Complex Struct Message Latency

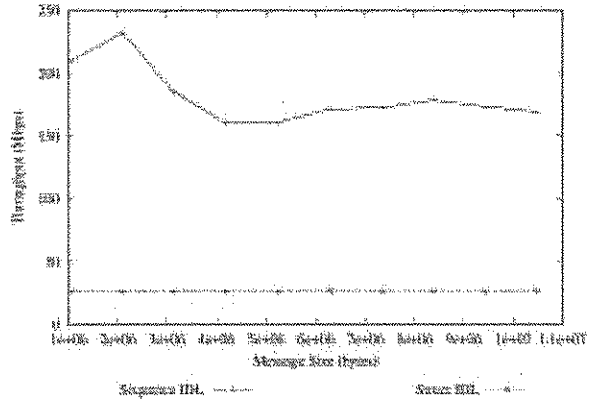

Fig. 9. Simple Sequence vs. Complex Struct Throughput

IV. REAL-TIME EVENT CHANNEL

Our main project focus was on our real-time experiments where we sought to simulate the NAV data distribution scenario and evaluate the real-time features of TAO and Solaris 8. The system was configured as shown in Fig. 10. We used a CORBA Event Channel with one supplier (the RTS) and one consumer (the SONAR system). Events were supplied every 50 ms (20 Hz). In order to avoid a mishap such as a failed missile launch, design requirements dictate that events must not arrive more than 50 ms late and that no more than 3 late events per 10,000 can be tolerated. The NAV event data consisted of a 384-byte complex structure with various data types. The Event Service and Naming Service ran on the same machine as the Supplier.

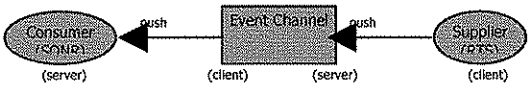

Fig. 10. Real-Time Experiment Architecture

We conducted our real-time experiments by varying a few different parameters. For the purpose of comparison, we performed experiments using the TAO COS Event Channel with the Solaris time-sharing process designation, as well as the TAO Real-time Event Channel with the Solaris real-time process designation. In addition, we conducted our experiments with "loaded" CPUs and "unloaded" CPU's. On a "loaded" system, there were 20 processes running the following line of code.

```
while (1) {for (i=0; i<500000; i++); usleep(1);}
```

which simply counts to 500,000 and then sleeps for one microsecond. On an "unloaded" system, no processes were running, other than experiment processes and system processes. Lastly, we performed our experiments beginning with a single machine (no network involved—simply process-to-process communications) and then on Gigabit Ethernet and ATM networks.

In each of these experiments, we emitted 100,000 NAV messages. Because we did not have a trusted global clock, we measured the inter-arrival time of the NAV messages on the consumer machine. Thus, in an ideal world, the inter-arrival time would always be 50ms. As stated previously, there is a design constraint that messages must not be more than 50ms late. Thus, the inter-arrival time must never be more than 100 ms to meet system requirements. The results of our real-time experiments are shown in the next two sections.

A. Timing Measurements For Non-Real-Time Configuration

We began with a single machine (process-to-process communications—no network) with unloaded CPUs, using the non-real-time event channel and the time-sharing process designation. The results are shown in Fig. 11. In this case, there are 21 out of 100,000 (0.02%) late messages (those with inter-arrival times greater than 100 ms), so the design requirements of no more than 30 out of 100,000 late messages are met.

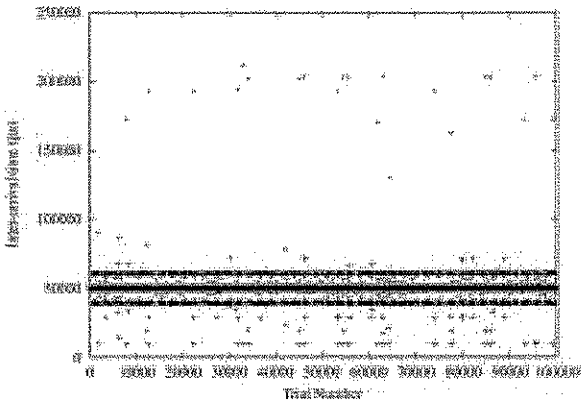

Fig. 11. Process-to-Process, Non-Real-Time Channel, Unloaded CPUs

When the same experiment is performed, except with loaded CPUs, a different picture is seen, as shown in Fig. 12. In this case, we see a drastic increase in the number of late messages. There are 883 late messages, which is approximately 0.9%. This clearly demonstrates the need for real-time services to help provide assurance that the event channel messages take priority over other activities.

In the next two experiments, we move to a two-machine configuration, connected over a Gigabit Ethernet network. We are still using the non-real-time event channel and the time-sharing process designation.

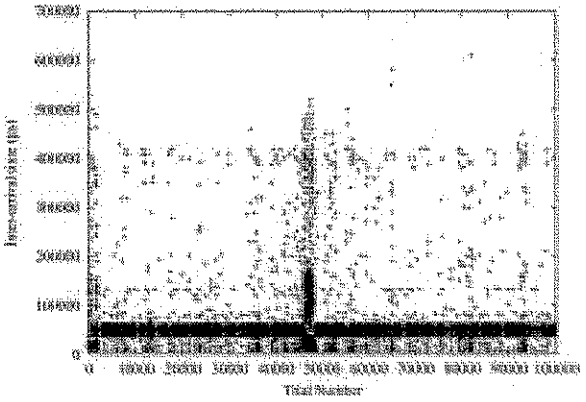

Fig. 12. Process-to-Process, Non-Real-Time Channel, Loaded CPUs

Fig. 13 shows this configuration with unloaded CPUs on both machines. In this case, we see 49 late messages, or 0.05%. Again, this does not meet our design requirements of 30 or fewer late messages per 100,000.

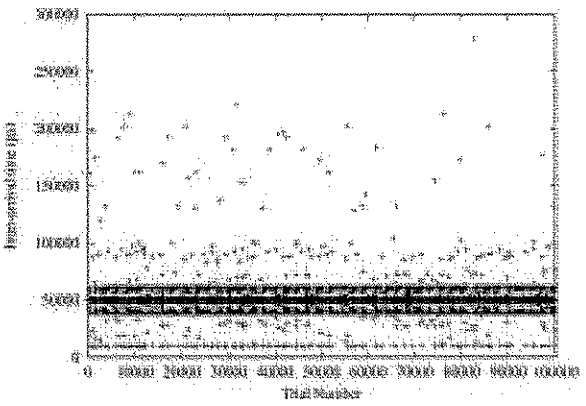

Fig. 13. Gigabit Ethernet, Non-Real-Time Channel, Unloaded CPUs

Finally, if we load both machines' CPUs on the Gigabit Ethernet network, we see the results shown in Fig. 14. Here we see an absolute blow-up. There are over 32,000 late messages (32%). Clearly, steps must be taken to avoid this situation, which could easily compromise a missile launch.

B. Timing Measurements for Real-Time Configuration

The next set of results is from experiments using the TAO Real-time Event Channel and the Solaris real-time process designation. In the first experiment, we used a single machine with unloaded CPUs. The results are shown in Fig. 15. These results are very encouraging. Virtually all of the inter-arrival times are distributed tightly around 50 ms and there are no late messages at all.

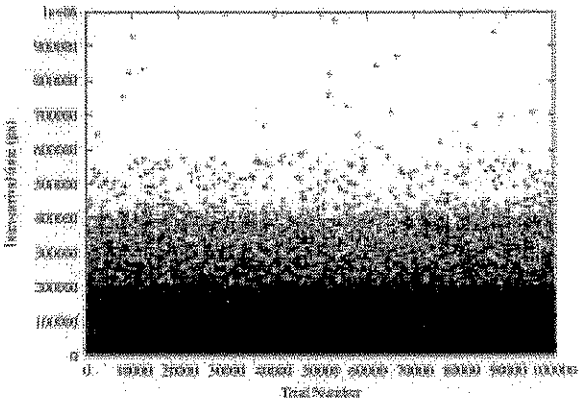

Fig. 14. Gigabit Ethernet, Non-Real-Time Channel, Loaded CPUs

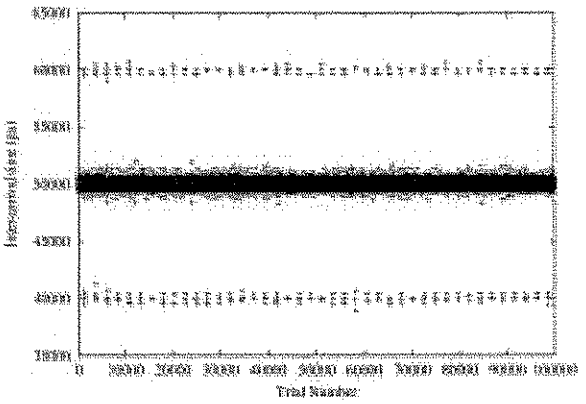

Fig. 15. Process-to-Process, Real-Time Channel, Unloaded CPUs

When this same configuration is used, except with loaded CPUs, we get the results shown in Fig. 16. In this case, there are 18 late messages (0.02%), but we are still able to meet our design requirement of no more than 30 late messages.

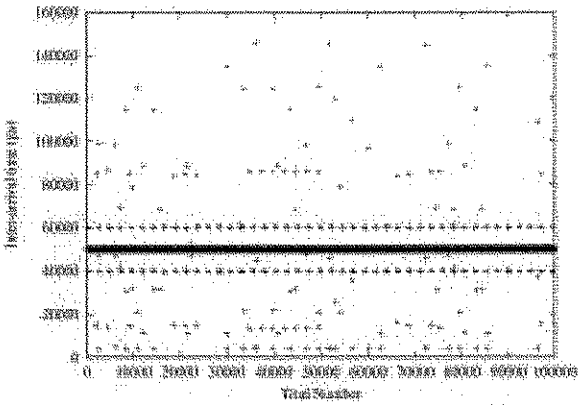

Fig. 16. Process-to-Process, Real-Time Channel, Loaded CPUs

In our next experiment, we used two machines, connected via Gigabit Ethernet, with unloaded CPUs. The results are shown in Fig. 17. Here we see 8 late messages out of 100,000, which is perfectly acceptable under our design constraints.

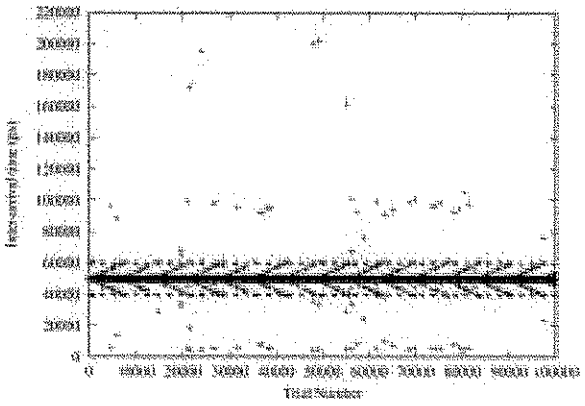

Fig. 17. Gigabit Ethernet, Real-Time Channel, Unloaded CPUs

In our last experiment (Fig. 18), we show the same configuration, except that both machines on the Gigabit Ethernet network have their CPUs highly loaded. In this graph, we see 59 late messages (0.06%). Unfortunately, this does not meet the design constraint of 30 or fewer late messages per 100,000. Further steps must be taken in this case to prevent a disaster due to lack of knowledge of the ship's precise position.

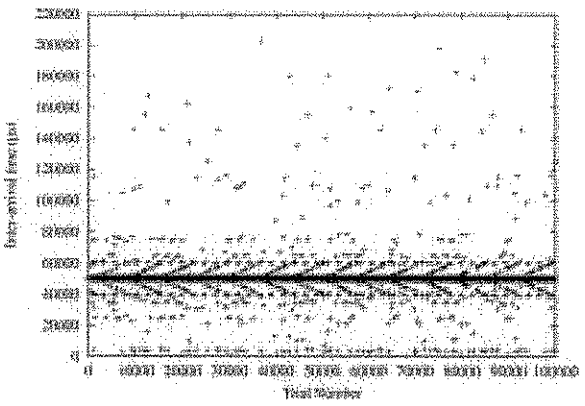

Fig. 18. Gigabit Ethernet, Real-Time, Loaded CPUs

V. CONCLUSIONS AND FUTURE WORK

In conclusion, we note that the design requirements (no more than 3 out of 10,000 navigation messages may arrive more than 50 ms after they were sent) can be met using the TAO Real-time Event Channel and the Solaris Real-time Process Designation for all but the last case: a heavily-loaded set of machines connected via a Gigabit Ethernet network. As we have seen, the real-time features of TAO and Solaris cannot provide hard guarantees for delivery of navigation messages. However, they certainly provide assurance that messages will arrive on time with a high probability.

The real-time features of TAO are still in development and are not yet well documented. In addition, these features do not integrate seamlessly with Solaris. The current real-time features of TAO are geared more towards event filtering and thread scheduling within the ORB. For example, the Real-time Event Channel is useful where there are multiple suppliers and multiple consumers, and any given consumer does not care to receive all events. The Real-time Event Channel can filter out unwanted events so that a consumer does not have to waste CPU time processing events that it does not need. Also, the real-time features of the ORB are designed more toward thread scheduling within the ORB, rather than between the ORB and other processes running on the same machine.

A number of additional experiments could be conducted to further the research in this area. First, the QoS features of Gigabit Ethernet and ATM networks could be used to help provide additional assurances for timely delivery of navigation messages. Second, a more realistic NAV distribution scenario could be used. When the Lockheed Martin lab network is complete, a NAV distribution experiment with all 35 destinations could be performed to determine if all 35 messages are received on time, every time. Also, the tests could be performed again when the TAO real-time features have been enhanced. Lastly, experiments where the network is heavily loaded could be performed to determine how the navigation data distribution performs in the face of varying network congestion.

VI. ACKNOWLEDGEMENTS

We gratefully acknowledge the technical and financial support of the Naval Electronics and Surveillance Systems Division of Lockheed Martin in Manassas, Virginia. Our project director was Pat Watson, with additional technical input from Brian Womble and John Toloczko.

VII. REFERENCES

[1] G.H. Thaker and P.J. Lardieri, "In Search of Commercial Off-the-Shelf, Hard Real-time, Distributed Object Computing Middleware," 3rd International Symposium on Distributed Objects and Applications, Rome, Italy, September 2001, pp. 146-154.

[2] The ACE ORB. Center for Distributed Object Computing, Department of Computer Science, Washington University in St. Louis. http://www.cs.wustl.edu/ TAO.html

[3] CORBA 2.6 Specification. Object Management Group. http://www.omg.org/cgi-bin/doc?formal/01-12-35

[4] Real-Time CORBA 1.0 Specification. Object Management Group. http://www.omg.org/cgi-bin/doc? formal/01-09-34 (Chapter 24)