HLA RTI Performance in High Speed LAN Environments

***Richard Fujimoto, PhD*[1]**
College Of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
fujimoto@cc.gatech.edu

Peter Hoare, PhD, CEng
Defence Evaluation and Research Agency
St. Andrews Road
Malvern, WORCS UK WR14 3PS
peteh@signal.dera.gov.uk

Keywords:·
High Level Architecture, Runtime Infrastructure, Cluster Computing, Time Management.

ABSTRACT: *This paper presents recent results concerning the realization of HLA RTIs in high-speed LAN environments. Specifically, the UK-RTI and a second, simplified RTI implementation were realized on a cluster of Sun workstations using Myrinet, a gigabit, low-latency interconnection switch developed by Myricomm Inc. The performance of these implementations was compared with the UK-RTI and version 1.3 of the DMSO RTI using UDP and TCP/IP on an Ethernet LAN. The Myrinet implementations utilize a software package called RTI-Kit that implements group communication services and time management algorithms on high-speed interconnection hardware. Results of this study demonstrate the technical feasibility and performance that can be obtained by exploiting high performance interconnection hardware and software in realizing HLA RTIs. In particular, in most experiments the RTIs using Myrinet achieved one to two orders of magnitude improvement in performance relative to the UK-RTI and DMSO version 1.3 RTI using UDP/TCP in attribute update latency time, and the wallclock time required to perform a time management cycle.*

1. Introduction

The High Level Architecture (HLA) has been mandated by the U.S. Department of Defense (DoD) as the standard technical architecture to be used by DoD modeling and simulation programs [1]. As such, the HLA must span a broad range of applications with diverse computational and communication requirements, as well as a broad range of computing platforms with widely varying cost and performance characteristics.

An important, emerging class of distributed computing platforms are cluster computers, i.e., high performance workstations and/or personal computers inteconnected with a high bandwidth, low latency interconnection switch. Cluster computers offer the potential for much higher communication performance than classical networking solutions. This is typically accomplished by by-passing traditional networking protocols, and providing low level communication services directly to the application. This typically yields message latency times on the order of tens of microseconds or less, compared to hundreds of microseconds using traditional local area networking hardware and software. Work in developing industry wide communication standards for cluster computers called the Virtual Interface Architecture (VIA) standard is under way, and will help to ensure the widespread availability of this technology [2]. Unlike most supercomputers, cluster computers can be incrementally upgraded by replacing existing CPUs with newer models, and/or including additional workstations and expanding the interconnection switch.

Most work to date has focused on realization of the High Level Architecture (HLA) Runtime Infrastructure (RTI) on conventional networking hardware using well-established protocols such as UDP and/or TCP. While such implementations are

[1] Work completed while on leave at the Defence Evaluation and Research Agency, Malvern, UK.

sufficient for large portions of the M&S community, many applications require higher communication performance than can be obtained utilizing these interconnection technologies. Cluster computers offer a solution to this problem, but to date, little experience has been reported concerning the realization of HLA RTIs exploiting these high performance interconnection switches.

In conjunction with the UK-RTI development[14], a prototype library was developed to explore the implementation of a Run-Time Infrastructure in cluster computing environments composed of workstations interconnected with high speed, low latency switches. This paper describes the RTI-Kit objectives, implementation and performance.

2. RTI-Kit

RTI-Kit is a collection of libraries designed to support development of Run-Time Infrastructures (RTIs) for parallel and distributed simulation systems, especially distributed simulation systems running on high performance cluster computing platforms. Each library is designed so it can be used separately, or together with other RTI-Kit libraries, depending on what functionality is required. It is envisioned that these libraries will be embedded into existing RTIs, e.g., to add new functionality or to enhance performance by exploiting the capabilities of a high performance interconnect. Alternatively, the libraries can be used in the development of new RTIs.

This "library-of-libraries" approach to RTI development offers several important advantages compared to more traditional RTI design approaches. First, it enhances the modularity of RTI designs because each library within RTI-Kit is designed as a stand alone component that can be used in isolation of other modules. Modularity enhances maintainability of the software, and facilitates optimisation of specific components (e.g., time management algorithms) while minimising the impact of these changes on other parts of the RTI. This design approach facilitates technology transfer of the results of this work to other RTI development projects because utilising RTI-Kit software is not an "all or nothing" proposition; one can select modules such as the time management or multicast communication software, and utilise only those libraries while ignoring other libraries of secondary importance to potential users.

Multiple implementations of each RTI-Kit library are planned that are targeted at different platforms, ranging from workstations interconnected with conventional networking technologies, to compute clusters using high speed interconnects, to shared or distributed memory multiprocessors. The current implementation utilises cluster computing platforms composed of Sun workstations interconnected via Myrinet switches.

The principal components of RTI-Kit that have been developed are MCAST, a communication library supporting group (multicast) communications among a collection of federates, and TM-Kit which implements time management algorithms. Other libraries that have been developed provide services such as buffer management, priority queues, and random number generation. The latter two libraries were adapted from code originally used in the Georgia Tech Time Warp (GTW) software [3].

3. RTI-Kit Architecture

An architecture diagram showing how RTI-Kit might be embedded into an existing RTI (e.g., the UK-RTI) is shown in Figure 1. The existing RTI will generate calls to MCAST to utilise the group communication services. Group communications means messages are in general sent to a collection of destinations rather than to a single receiver. This is analogous to Internet newsgroups where each message sent to a newsgroup is transmitted to all subscribers to the group. Here, the Distribution Manager (DMAN) object in Figure 1 will call MCAST primitives to create, join, and leave groups, as well as to send and receive messages. If it wishes these communication services to operate under a time management protocol, it will also generate calls to TM-Kit to initiate LBTS (lower bound on time stamp) computations, and to provide TM-Kit with the information it needs to compute LBTS. For example, TM-Kit must be notified whenever a time stamped message is sent or received, in order to account for transient messages. For buffer allocation, the existing RTI can utilise its own memory management services by embedding callbacks into MCAST, or it can use a module called MB-Lib for buffer allocation.

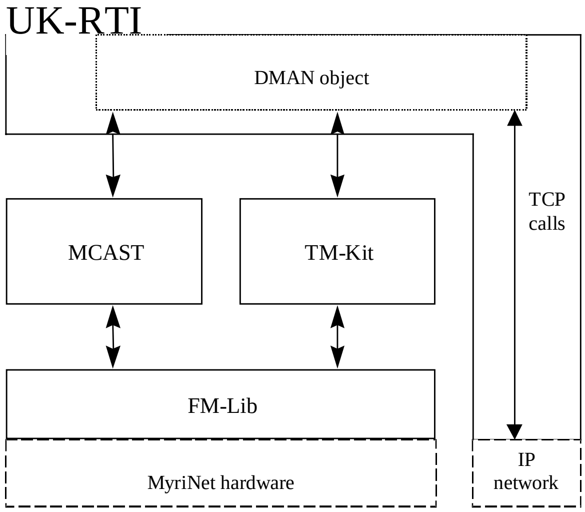

Figure 1. RTI architecture using RTI-Kit.

MCAST and TM-Kit both require the use of basic communication services, defined in a module called FM-Lib. This communication layer software acts as a multiplexer to route messages to the appropriate module. If the existing RTI wishes to bypass MCAST for certain communications that it requires, it can either utilise FM-Lib directly, or use an altogether different mechanism. The latter option assumes, of course, this alternate mechanism can co-exist with FM-Lib. The current implementation of FM-Lib utilises an API based on the Illinois Fast Messages (FM) software [4] for its basic communication services, and provides only slightly enhanced services beyond what FM provides.

The RTI-Kit architecture is designed to minimise the number of software layers that must be traversed to realise communication and time management services. For example, TM-Kit does not utilise the MCAST library for communication, but rather directly accesses the low-level primitives provided in FM-Lib. This is important in cluster computing environments because low level communications are very fast, on the order of ten to twenty microseconds latency for short messages, compared to hundreds of microseconds or more when using conventional networking software such as TCP/IP. Thus, if not carefully controlled, overheads introduced by RTI-Kit software could severely degrade performance in cluster environments, whereas such overheads would be insignificant in traditional networking environments where the time required for basic communication services is very high. Measurements indicate the overheads introduced by RTI-Kit are indeed small, on the order of 10% when using Myrinet switches.

4. MCAST Library

Two goals in the MCAST design are (1) to support potentially large numbers of multicast groups (e.g., thousands) with frequent changes to the composition of groups, and (2) to minimise the overhead of frequently used operations such as sending and receiving messages. It has been reported that large distributed simulation exercises typically require thousands of multicast groups, and rapid (no more than a few milliseconds) changes to group composition [8]. Further, as discussed earlier, communication overheads must be kept low to enable full exploitation of the fast communication services provided in cluster environments.

A variety of techniques are used to achieve these goals. Groups in MCAST are light weight. Storage required to implement each group is kept to a minimum; the bulk of the storage is used to hold lists indicating group membership. Hashing functions are used to rapidly access table entries. Group membership information is cached at processors sending messages to the group to reduce interprocessor communication. Message copying overheads are avoided by allowing the application to specify where incoming messages are to be written, as opposed to more traditional approaches where each incoming message is written into a temporary message buffer and then copied it to an application defined buffer after the application has been notified that the message has been received. Joining or leaving groups is optimised to reduce the amount of interprocessor communication that must take place to realise these operations

Each MCAST group contains a list of subscribers to that group. Whenever a message is received for a group, each subscriber is notified by a call to a procedure (a message handler) defined by the subscriber. A single processor may hold multiple subscriptions to a group, in which case it receives a call back for each one when a message for the group is received. Any processor can send a message to any group; in particular, one need not be subscribed to the group in order to send a message to it. If the processor sending a message to the group is also subscribed to the group, it will still receive a call back for each subscription, i.e., processors can send messages to themselves using this service. Of course, the message handler can simply ignore messages sent by a processor to itself if this is the desired semantics.

Each group has two unique names that are used to identify it: (1) an ASCII string name, and (2) an internal handle that serves as a pointer to the group. The internal name is required to send/receive messages to/from the group. A name server is provided that maps the ASCII names of groups to their internal name.

In the current implementation, multicast communication is implemented by a sequence of unicast sends, with one message send to each processor holding one or more subscribers to the group. If the processor holds several subscriptions to the group, only a single physical message is sent to the processor.

Destination lists for groups are distributed among the processors participating in the simulation. Specifically, the destination list for each group is maintained at a "home" location for that group. Message senders for a group need not reside at the home location for that group. In order to avoid communication with the home location with each message send, MCAST includes a caching mechanism to provide a local copy of the destination list to each sender. The destination list is only loaded into the sender's cache when the sender obtains a handle for the group. This avoids the need to load destination lists into caches for processors that do not have a handle (and thus cannot send messages) to that group. MCAST includes a set of cache directories in order to update copies of the destination list when the composition of the group changes.

MCAST is designed to minimise message copying, thereby eliminating a major source of overhead in traditional communication systems. Specifically, each processor specifies a memory allocator, called a WhereProc procedure that indicates where incoming messages are to be written. Different WhereProc procedures may be specified for different groups. This allows MCAST to directly write incoming messages into the location specified by the user, rather than writing it into an internal buffer, which would typically require the application to later copy the message to its desired location.

5. TM-Kit Library

The services in TM-Kit provide the basic primitives necessary for time management in distributed simulations. At the heart of TM-Kit is an algorithm for computing LBTS information, i.e., a lower bound on the time stamp of future messages that can be later received by each processor. The other procedures, for the most part, are included to provide enough information to compute LBTS, and do not actually perform any other service. For example, the "communication primitives" TM_Out() and TM_In() do not actually send and receive messages. They only inform the TM software that messages have been sent/received, and provide the TM software the opportunity to piggyback some of its own information onto the message.

To use TM-Kit, users must make the following additions to an existing RTI. First, calls must be added to initiate LBTS computations, and handlers must be defined that are called when the LBTS computation is complete. Calls must be made to TM-Kit whenever time stamp ordered messages are sent or received, so that TM-Kit can properly account for transient messages (messages sent but not yet received). Applications define abstract data types and associated operators for operations dealing with time values (e.g., comparisons), allowing TM-Kit to be used with arbitrary representations of time (integers, floating point values, priority fields to break ties, etc.). TM-Kit also requires that certain information be piggy-backed onto messages, so mechanisms are provided to add this information to outgoing messages, and to remove it from messages that are received. Finally, the TM-Kit interface supports exploitation of federate topology information, i.e., information considering which federates send messages to which other federates, though time management algorithms to exploit this information have not yet been implemented.

TM-Kit does not directly handle time stamped messages. Thus, message queuing and determining when it is "safe" to deliver time stamped messages is performed outside of TM-Kit.

The underlying computation model for the simulation is viewed as a collection of processors that communicate through message channels. A parallel computer could be designated as a single processor if it provides its own internal time management mechanism (e.g., a Time Warp implementation), or each processor within the parallel computer can be viewed as a separate processor, in which case TM-Kit provides time management among processors in the parallel machine.

TM-Kit computes LBTS by using a "butterfly" like interconnection graph. This allows a global minimum to be computed, and the result distributed to all processors in log N time (with N processors) and N log N messages, without use of a centralised controller. TM-Kit can be configured to use different "fanouts" for the butterfly interconnection scheme.

The butterfly scheme is similar to well-known parallel prefix and barrier algorithms (e.g., see [11]), and works as follows. Assume an N-processor reduction is performed, and the processors are numbered 0, 1, 2, …, N-1. Each processor holds a local time value. The goal is to compute the global minimum among these N values, and notify each processor of this global minimum. To simplify the discussion, assume N is a power of 2; the TM-Kit software allows arbitrary values of N. The communication pattern among processors for this barrier mechanism for the case of eight processors is shown in Figure 2(a). Each processor executes a sequence of log N pairwise minimum operations with a different processor at each step. A pairwise minimum operation between processors i and j is accomplished by simply having i (or j) send a message to j (i) containing the minimum value it has computed thus far, then wait for a similar message from j (i). When this message is received, a new local minimum is computed as the minimum of the current value and the new value received from j (i). In the first step, processors whose binary addresses differ only in the least significant bit perform a pairwise minimum, e.g., processors 3 (011) and 2 (010). In the second step, processors whose addresses differ only in the second least significant bit, e.g., processor 3 (011) and 1 (001) exchange messages. In general, in step k processor i communicates with the processor whose address differs in only the kth bit (where bits are numbered 1, 2, ... log N from least significant to most significant). These pairwise minimum operations continue for log N steps until all of the address bits have been scanned. This communication pattern is referred to as a butterfly. Each processor has the global minimum once it has completed the log N pairwise minimum operations.

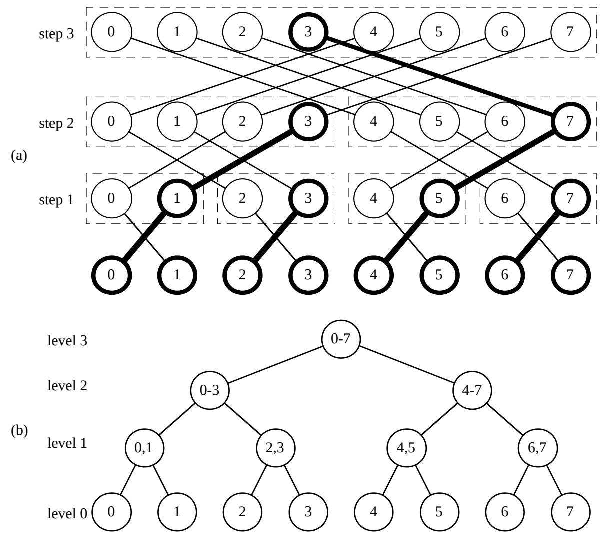

Figure 2. Eight processor Butterfly barrier. (a) communications pattern, and illustration of barrier from the perspective of processor 3. (b) tree abstraction of barrier mechanism.

The operation of the barrier mechanism from the perspective of a particular processor is shown in Figure 2. The highlighted nodes and arcs in Figure 2(a) illustrate the barrier from the perspective of processor 3. After step 1 has completed, processor 3 has computed the minimum of the values stored in processors 2 and 3. This is illustrated by the dashed box around processors 2 and 3 in step 1 of Figure 2(a). In step 2 processor 3 receives a message from processor 1 containing the minimum between processors 0 and 1. By computing the minimum of this value with its own local minimum, processor 3 computes the minimum among processors 0, 1, 2, and 3. This is represented by the dashed box around these four processors in step 2. Continuing this analysis, in step 3 processor 3 receives a message from processor 7 that contains the minimum among processors 4, 5, 6, and 7. After minimising this value with its local minimum, it now has the global minimum value over all eight processors. In effect, as shown in Figure 2(b), a tree is constructed in bottom-up fashion with processors at the leaves. Intermediate nodes of the tree indicate the set of processors whose minimum values have been computed when that step of the algorithm has been completed.

The above algorithm is complicated somewhat by the fact that the LBTS computation must take into account transient messages, i.e., messages that have been sent, but have not yet been received while the LBTS computation is taking place. Failure to take into account these messages could result in an incorrect (i.e., too large) LBTS value to be computed. To address this problem, one must (1) have a mechanism to detect the presence of transient messages in the system since the LBTS computation cannot be completed until all such messages have been accounted for, (2) detect when a transient message has been received, and (3) provide a means for including the time stamp of transient messages into the global minimum computation. Detecting the presence of transient messages is accomplished by maintaining two counters in each processor indicating the number of messages the processor has sent, and the number it has received. Actually, TM-Kit only maintains a single counter indicating the difference between these two values, but we assume two counters are used to simplify the explanation. When all processors have initiated an LBTS computation and the sum of the message send counters across all of the processors in the system is equal to the sum of the message receive counters, there are no transient messages remaining in the system. These counters are reset when the processor initiates (or detects that another processor has initiated) a new LBTS computation. The global minimisation operation is augmented to also sum the two counters in each step, enabling each processor to detect when there are no more transient messages. Specifically, if the counters do not match, then this indicates one or more transient messages has not been included in the global minimum, so completion of the LBTS computation is delayed.

Receipt of a transient message is detected by a colouring scheme [12]. Each processor is initially some colour, say red. When a processor enters into a new LBTS computation, it changes colour, e.g., to green. Each message sent by a processor is tagged with the sending processor's current colour. Therefore, if a green processor receives a red message, it is known that that message is a transient message, i.e., one sent before the sender entered into the LBTS computation, but received after the receiver had entered into the computation and reported its local minimum. When a processor receives a transient message, the time stamp of this message is reported up through the butterfly network, as well as the fact that the receive count for that processor has increased by one, in exactly the same way as other butterfly operations as discussed previously. Thus, when the last transient message is received the sum of the send counters and the sum of the receive counters will now match, indicating the true LBTS value has been computed and the computation has been completed.

The butterfly scheme described above uses pairwise minimum operations. In general, one can perform F-wise minimum operations at each step where each processor sends its local minimum to F-1 other processors. F is referred to as the fanout. The TM-Kit software implements this functionality, and allows arbitrary fanout (up to the number of processors in the system) to be used. TM-Kit uses a default value of 2 as the fanout, and this value is used in all of the experimental results reported later.

TM-Kit allows any processor to initiate an LBTS computation at any time by invoking a TM-Kit primitive. If two different processors simultaneously begin a new LBTS computation, the two computations are automatically merged, and only one new LBTS computation is actually started. If a processor detects another processor has initiated an LBTS computation, but none has been started locally, the processor can then either participate in this new LBTS computation, or defer participation until some later time; in the latter case, the processor must eventually participate in this computation or it will never complete. Each processor is notified when an LBTS computation has completed by a call-back from RTI-Kit that indicates the new LBTS value.

The approach used in TM-Kit for time management offers several attractive properties. First, the time complexity of the LBTS computation grows logarithmically with the number of processors, enabling it to scale to large machine configurations. Because any processor can initiate a new LBTS computation at any time, RTI developers are given great flexibility concerning how they initiate these computations. For example, each processor might asynchronously start a new computation whenever it needs a new LBTS value to be computed; as discussed earlier, the TM-Kit software automatically merges multiple, simultaneous initiations of new LBTS computations by different processors into a single LBTS computation. This approach is used in the UK-RTI and BRTI implementations discussed later. Alternatively, a central controller could be used to periodically start a new LBTS computation at fixed intervals of wallclock time, or using some other criteria. TM-Kit allows multiple different LBTS computations to be in progress at one time, i.e., a processor can start a second LBTS computation even though one that it had previously initiated has not yet completed (note this is different from simultaneous initiation of LBTS computations by *different* processors, which are automatically merged into one computation). This allows a processor to immediately begin a new LBTS computation when it has obtained new information (e.g., a new local minimum), without having to wait for a previously initiated LBTS computation to complete. The current implementation of TM-Kit allows up to K LBTS computations to be in progress at one time, where K is a parameter specified during configuration.

6. Other Libraries

MB-Lib is a simple library providing memory allocation services. This can be coupled with MCAST's where-procedures to allocate memory for incoming messages, or applications using RTI-Kit can utilise their own memory management facilities.

Heap-Lib provides software to implement priority queues. One or more priority queues can be created, with arbitrary types of information stored in each node of the queue. An in-place heap data structure with O(log N) insertion and deletion times, where N is the number of elements in the heap, is used in the current implementation

Rand-Lib provides a collection of random number generators for a variety of probability distributions. This and the heap library were derived from the Georgia Tech Time Warp (GTW) software distribution. Rand-Lib allows one to draw random numbers following the uniform, binomial, exponential, gamma, geometric, normal, or Poisson distributions. Facilities are provided to support generation of multiple independent streams of random numbers.

The current implementations of TM-Kit and MCAST interface to the communications system through FM-Lib. FM-Lib is built on top of, and is only slightly different from the Illinois Fast Messages (FM 2.0) API, as described in [5]. The RTIKit software described here uses FM "as is", facilitating portability to other platforms since one need only recreate the FM interface. An alternative approach involving modification of the Myrinet switch firmware is described in [13].

7. The Myrinet System

The first implementation of RTI-kit has been written on top of the Myrinet networking hardware [4] running on a network of Sun UltraSPARC workstations. The Myrinet hardware is a high performance networking interconnect designed to enable individual workstations to be built into cluster computers. It provides low latency, high bandwidth communications; the system currently in use has a bandwidth of 640 Mbit/s but now Myricom are shipping a 1.2 Mbit/s version. Because the experiments described here do not utilize the fastest available switches, we consider the performance measurements presented later to be conservative.

Whilst the actual speed of the interconnect is not that much faster than 100Base-T Ethernet the main difference between using the Myrinet and TCP/IP is that the overhead of the protocol stack can be removed. The Myrinet device is accessible via an area of shared memory in the process's address space and data can be read and written to the device without a context switch. This gives a dramatically lower latency than traditional TCP/IP communications, especially TCP where there are multiple memory copies as the data moves from user space, into the kernel and down onto the Ethernet device.

Another feature of the Myrinet which is extremely useful is that the network is built out of low latency cut through switches which route packets of arbitrarily long length through the network between source and destination hosts. This removes the fragmentation of TCP packets forced by the 1500 byte IP limit and also provides an almost perfect error free connection. The Myrinet does not however directly support either broadcast or multicast which needs to be implemented in software as multiple point to point communications.

8. RTI Implementations

Two separate RTI implementations were developed using RTI-Kit. First, the UK-RTI [14], an HLA RTI originally implemented using TCP and UDP/IP, was modified to use the RTI-Kit libraries. In addition, a separate RTI built directly on top of RTI-Kit was developed (in contrast to the UK-RTI which was an existing implementation modified to use RTI-Kit), providing an additional point of comparison. This implementation is referred to as the BRTI ("baby" RTI). The BRTI software is written in C, and implements a modest subset of the HLA Interface Specification services. In particular, BRTI implements key time management services (e.g., NextEventRequest, TimeAdvanceRequest, TimeAdvanceGrant) as well as certain object management and declaration management services.

At present, BRTI does not conform to the Interface Specification (IFSpec) API, e.g., the UpdateAttributeValues and ReflectAttributeValues services transmit a block of bytes to the destination, and do not pack/unpack the data into attribute handle pair sets. It does implement the callback mechanisms required in the HLA IFSpec for message transmission and time management.

9. Performance Measurements

Underlying Communication Performance

To understand the potential communcation gains of cluster computers experiments were run on the SPARCstation Ultra-1 machines to measure average one way message latencies of the different communications bearers. The machines used in the experiments were otherwise idle when the experiments were performed. We have observed that latency times vary significantly when the test program must compete with computations from other users that are simultaneously using the test machines. All experiments described below were run on a network of UltraSparc 1 machines on Solaris 2.5.

The figures shown in Table 1 have been measured on the Myrinet configuration, which has been reported yielding bandwidth as high as 550 Mbytes/sec one way. The table shows one-way message latency, measured as the round-trip time to send a message between processors divided by two. Each experiment completes 100,000 round-trip sends of four byte packets via Myrinet, socket based TCP and CORBA (OMNIBroker[7]) (the CORBA and TCP communications used a 100 Mbit network).

CORBA (OMNIbroker) over TCP	TCP (raw Socket based comms)	Myrinet
552 microsecs	168 microsecs	13 microsecs

Table 1 : Raw latency communcations performance over different bearers

As can be seen the Myrinet performance is an order of magnitude better than using CORBA and even raw TCP communications. This underlying latency is even better on the latest generation of Myrinet hardware.

Figure 3 compares the latency (half the round trip time to send and return the payload) for RTI-Kit's MCAST library with the underling FM software and TCP for different message sizes. As can be seen, MCAST introduces only a modest increase in latency beyond the underling FM system.

RTI Performance

RTI performance was evaluated using the DMSO benchmark programs [6]. Software included in the version 1.3 release was used for the DMSO and UK-RTI evaluation[2]. The benchmarks were re-implemented for evaluating BRTI performance. The studies described here focused on two benchmark programs: the "latency" and "TAR" programs.

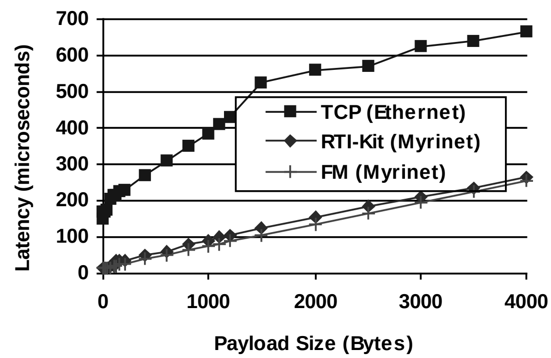

Figure 3. Message latency measurements.

The Latency Benchmark

The latency benchmark measures the latency for best effort, receive ordered communications. This program uses two federates, and round-trip latency was measured. Specifically, latency times report federate-to-federate delay from when the UpdateAttributeValues service is invoked to send a message containing a wallclock time value and a payload of N bytes until the same federate receives an acknowledgement message from the second federate (via a ReflectAttributeValues callback). The acknowledgment message contains a time stamp, but no payload.

Latency measurements were taken and averaged over 100,000 round trip transmissions. The one-way latency time is reported as the round trip time divided by two.

We remark that this seems like an unusual way to measure one-way latency since the return message does not contain a payload, but we used this approach to be consistent with the existing DMSO software and methodology discussed in the literature. As noted earlier, the "raw" one-way transmission latency for a four byte message using the low level message passing layer (FM) on the Myrinet switch was measured at 13 microseconds.

The graph in Figure 4 shows the one-way latency time

[2] The UK-RTI only supports the previous version of the IF specification so the benchmarks had to be modified slightly to compile.

on BRTI, UKRTI (using Myrinet) and RTI1.3 (using TCP and UDP/IP) in microseconds; note the logarithmic scale. The Myrinet does not actually support unreliable communications so the times for UK-RTI and BRTI are for reliable communications, while those for RTI 1.3 are best effort. Federate-to-federate BRTI latency is approximately 20-30 microseconds for small messages (up to 128 bytes). The current figures for UK-RTI are only preliminary as the RTI is being optimized. Similar tests for interactions have already shown a speedup of an order of magnitude which is beginning to approach BRTI performance.

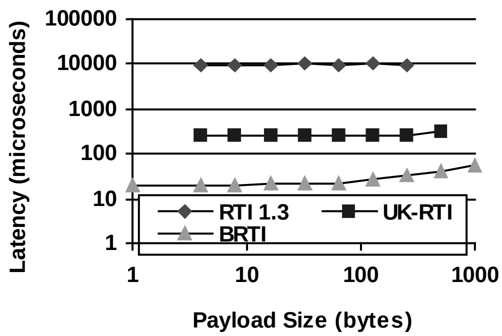

Figure 4. Update/Reflect latency on different RTIs.

It can be seen that one to two orders of magnitude reduction in latency were observed using the Myrinet cluster relative to the DMSO RTI using UDP/IP. While time did not permit comparison with the UK-RTI using UDP/IP, we anticipate a similar performance differential would exist. This is consistent with measurements of the time management benchmark that are discussed next.

The Time Advance Request Benchmark

The TAR benchmark measures the performance of the time management services, and in particular, the time required to perform LBTS computations.

This benchmark contains N federates, each repeatedly performing TimeAdvanceRequest calls with the same time parameter, as would occur in a time stepped execution. The number of time advance grants observed by each federate, per second of wallclock time is measured for up to eight processors. Performance measurements for BRTI are shown in Figure 5, and for UK-RTI (both the Myrinet and TCP implementations) and RTI 1.3 in Figure 6. The lookahead and time step size are set equal to 1.0 in these experiments. We note that substantial performance improvements can be gained by setting the lookahead to value slightly larger than 1.0, due to the nature of the definition of the time management services. As shown in Figure 5 this doubles the performance of BRTI (since a new LBTS computation only needs to be performed every other time step rather than every time step), and resulted in about a 25% increase in performance in the DMSO RTI.

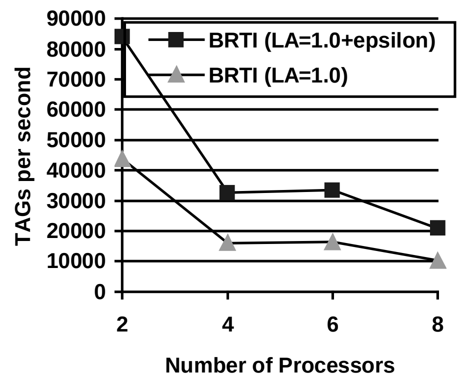

Figure 5. TimeAdvance benchmark on BRTI

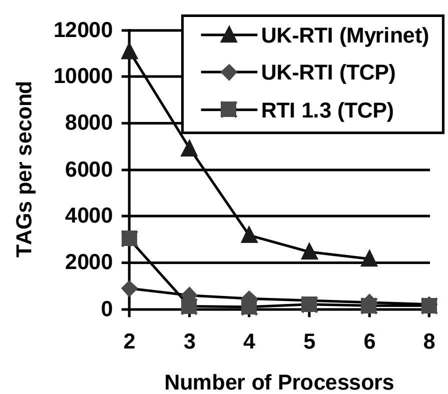

Figure 6. TimeAdvance benchmark on RTI1.3 and UK-RTI.

The number of TAR cycles per second in the DMSO RTI drops from several thousand to one to two hundred per second as the number of processors is increased beyond two. This is because in the two processor case, many TAR cycles can be performed without interprocessor communication, so require a negligible amount of time. However, when there are three or more processors, nearly all of the cycles require interprocessor communication, resulting in a substantially lower level of performance.

As can be seen in Figures 5 and 6 the RTI-Kit implementations using Myrinet achieve one to two orders of magnitude improvement in performance relative to the TCP implementations when there are three or more processors. The performance improvement in the two processor case is substantial, about a factor of 4, though not as dramatic.

These figures are expected to improve as performance tuning takes place. In particular, we believe the UK-RTI version will approach the speed of the BRTI implementation because both provide essentially complete implementations of the TimeAdvanceRequest / TimeAdvanceGrant cycle. The reason the UK-RTI performance is currently lower than that of BRTI is the UK-RTI includes additional computational overheads for queue management and memory allocation that are avoided in the BRTI implementation. While such overheads are modest compared to interprocessor communication overheads in conventional LAN environments (for which the UK-RTI was originally developed), these overheads become much more significant in high-speed LAN environments.

10. Conclusions

The experiments here have shown that it is possible to develop an RTI implementation for cluster computer hardware which significantly improves on the performance of the current generation of RTIs. Most of our measurements resulted in one to two orders of magnitude reduction in message latency and wallclock time to perform a time advance cycle compared to both the UK-RTI and DMSO RTI (version 1.3) using TCP and UDP/IP on an Ethernet LAN. As this paper was being written a presentation was given to the Architecture Management Group recommending study into non-TCP based RTIs [9] and we present this paper as the first data point in this regard. We have also shown that time management costs are substantially lower in high-speed LAN environments, helping to ensure federations of causal simulations are able to keep up and interoperate with 'real time' federations.

What has become abundantly clear during the performance tuning of this implementation of RTIKIT in the UKRTI is the importance of memory allocation and deletion. Using C++ which encourages object based design can, when used naively, result in unintentional and costly copying and creation of new objects. When dealing with Myrinet the cost of communications decreases to a level where even copying data once becomes a significant overhead. However the RTI is forced to make a certain unpredictable number of memory copies to cope with changes in publication and subscription and storing timestamped updates prior to release. This concern was also a focus of [10].

In the longer term it is interesting to note that the technology used in Myrinet is now becoming mainstream with the introduction of the Virtual Interface Architecture (VIA)[2] standard from Intel and Compaq. This is a server interconnect standard which can utilise fast networking hardware and removes the classical protocol stack overheads just as Myrinet does. Myricom is already able to support this standard and other vendors have products in the pipeline.

11. Acknowledgments

This work was funded by the UK Ministry of Defence Corporate Research Programme, Technology Package 10 Research Objective 6 - Technology for the Synthetic Environment and the DARPA ASTT program. Thanks also go to Duncan Clark and Gerry Magee for their implementation of UK-RTI.

12. References

[1] Defense Modeling and Simulation Office Web Site, http://hla.dmso.mil.

[2] The Virtual Interface Architecture standard, http://www.viarch.org

[3] R. Fujimoto, S. Das, and K. Panesar, "Georgia Tech Time Warp User's Manual," Technical Report, College of Computing, Georgia Institute of Technology, July 1994 http://www.cc.gatech.edu

[4] The Myrinet - A brief technical overview http://www.myri.com

[5] S.Pakin, M.Lauria, M.Buchanan, K.Hane, L. Giannini, J. Prusakova, A.Chein "Fast Message (FM) 2.0 User documentation", Dept of Computer Science University of Illinois at Urbana Champaign.

[6] Judith Dahmann, Richard Weatherley et al, "High Level Architecture Performance Framework", ITEC, Lausanne, Switzerland April 1998

[7] The OMNIBroker CORBA ORB http://www.ooc.com

[8] D.Miller A brief history of Distributed Interactive simlation (presentation notes) Distributed

Interactive Simulation IP/ATM multicast symposium May 1995.
[9] Jim Calvin "HLA/RTI Communcations experiments" Presentation to the 25th AMG June 1998 http://hla.dmso.mil
[10] Stephen Bachinsky, Larry Mellon, Glenn Tarbox and Richard Fujimoto "RTI 2.0 Architecture" SIW Fall 97 Orlando USA.
[11]D. Nicol, "Non-Commital Barrier Synchronization," Parallel Computing, vol. 21, 1995 pp 529-549.
[12] F. Mattern, "Efficient Distributed Snapshots and Global Virtual Time Algorithms for Non-FIFO Systems," Journal of Parallel and Distributed Computing, vol. 18, no. 4, August 1993, pp. 423-434.
[13] S. Srinivasan, M. Lyell, P. Reynolds, Jr., J. Wehrwein, "Implementation of Reductions in Supoort of PDES on a Network of Workstations," Proceedings of the 12th Workshop on Parallel and Distributed Simulation, May 1998, pp116-123.
[14] P. Hoare, G. Magee, and I. Moody, "The Development of a Prototype HLA Runtim Infrastructure [RTI-Lite] Using CORBA," 1997 Summer Computer Simulation Conference, July 1997, pp 573-578.

Author Biographies

DR. RICHARD FUJIMOTO is a professor in the College of Computing at the Georgia Institute of Technology. He received the Ph.D. and M.S. degrees from the University of California (Berkeley) in 1980 and 1983 (Computer Science and Electrical Engineering) and B.S. degrees from the University of Illinois (Urbana) in 1977 and 1978 (Computer Science and Computer Engineering). He has been an active researcher in the parallel and distributed simulation community since 1985 and has published over 70 technical papers in refereed journals and conference proceedings on parallel and distributed simulation. He lead the definition of the time management services for the DoD High Level Architecture (HLA) effort. Fujimoto is an area editor for ACM Transactions on Modeling and Computer Simulation. He also served on the Interim Conference Committee for the Simulation Interoperability Workshop in 1996-97 and was chair of the steering committee for the Workshop on Parallel and Distributed Simulation (PADS) in 1990-98.

DR. PETER HOARE is a Principal Scientist in the Parallel and Distributed Simulation section of the UK Defence Evaluation and Research Agency in Malvern. He received a BSc(Hons) and PhD degrees from Royal Holloway, University of London in 1988 and 1993 respectively. He has been working in parallel and distributed simulation since joining DERA in 1993. Currently he is the technical lead of several programmes in the UK researching the use of the DoD High Level Architecture including Project FlasHLAmp a joint UK MoD/Industy simulation technology transfer programme. Also he is currently a Drafting member of the HLA Standards Development group developing the IEEE standards documents.