APPLICATION OF RELIABLE HOST-BASED MULTICAST TO LARGE SCALE SIMULATIONS

By

Richard Stephen Grandy

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

WASHINGTON STATE UNIVERSITY
School of Electrical Engineering and Computer Science
December 2007

To the Faculty of Washington State University:

The members of the Committee appointed to examine the thesis of RICHARD STEPHEN GRANDY find it satisfactory and recommend that it be accepted.

_____

Chair

_____

_____

ACKNOWLEDGEMENT

I would like to thank my advisor, Dr. Dave Bakken, for the encouragement, guidance and assistance he has provided over the course of the development of my thesis.

I would also like to thank my committee members, Dr. Carl Hauser and Dr. Sirisha Medidi for their time and support during this defense.

I would like to thank Dr. Ken Birman, Cornell University for access to the CS cluster and Krzys Ostrowski for access to his QuickSilver Scalable Multicast code.

Finally, I would like to thank my wife, Patty, and sons Michael and Ryan, for their patience and support of me in this quest of completing a Masters of Science in Computer Science while balancing work and family responsibilities

APPLICATION OF RELIABLE HOST-BASED MULTICAST TO LARGE SCALE SIMULATIONS

ABSTRACT

by Richard Stephen Grandy, M.S.
Washington State University
December 2007

Chair: David E. Bakken

In support of both the development of new tactics and ongoing training, military and other organizations rely on simulations. To support such requirements, the High Level Architecture (HLA) was developed. HLA provides a standard for the development and deployment of distributed simulations. As the complexity and scale of the events being simulated increases, the underlying HLA simulation likewise increases in scale and complexity.

A key constraint in the performance of large-scale distributed HLA simulations is the ability of the distributed simulation components to communicate simulation state and coordinate their actions in simulated time. A key enabler for this communication and coordination is robust and reliable network communications. While the HLA standard provides simulation components both a best effort and a reliable communications transport, simulation developers often limit use of the reliable transport due to concerns about latency and throughput performance.

The contribution of this thesis is to examine the performance of a specific high-performance HLA implementation, RTI-s, under controlled benchmark conditions to document the simulation performance tradeoffs of best effort and reliable transport under controlled, representative loads. A key part of this examination was the integration of the QuickSilver Scaleable Multicast protocol, developed at Cornell University, into the high-performance HLA implementation. Research was done on the nature of HLA simulations and the characteristics of reliable multicast protocols. Controlled benchmark experiments were conducted and analysis provided highlighting the tradeoffs between use of best effort and reliable protocols for HLA simulations. Finally, considerations for both an “ideal” HLA benchmark and an “ideal” HLA reliable transport are offered.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

In support of both the development of new tactics and ongoing training, military and other organizations rely on simulations. To support such requirements, the High Level Architecture (HLA) was developed, HLA provides a standard for the development and deployment of distributed simulations. As the complexity and scale of the event being simulated increases, the underlying HLA simulation likewise increases in scale and complexity.

A key constraint in the performance of large scale distributed HLA simulations is the ability of the distributed simulation components to communicate simulation state and coordinate their actions in simulated time. A key enabler for this communication and coordinate is robust and reliable network communications. While the HLA standard provides simulation components both a best effort and a reliable communications transport, simulation developers often limit use of the reliable transport due to concerns about latency and throughput performance.

The question is, are those concerns well founded? Are they still valid? The HLA standard was established approximately 10 years ago [DFW97, DKW98] with some of HLA the network components dating back perhaps as long [VM96, WBV+97]. During that interval, research into reliable multicast has continued [BHO+99, JGJ+00, CRS+02, GMS+03, BPB05, OBP06, BBP+07]. If current-

generation reliable multicast is applied to HLA, will it open the door for greater use of reliable multicast? This is the fundamental question this research seeks to answer.

The paper proceeds as follows: Section 2 provides an overview of HLA simulation and some of the key HLA concepts that are germane to the research. Section 3 provides a brief overview of reliable multicast protocols and introduces QuickSilver Scalable Multicast (QSM). Section 4 discusses the task of integrating QSM into a high-performance implementation of HLA. Section 5 considers benchmarks to measure the performance of the HLA implementation. Section 6 presents the experimental results and analysis. Section 7 discusses related work. Section 8 provides concluding thoughts including ideas for more capable benchmarks and what an ideal HLA reliable might include.

2 HIGH LEVEL ARCHITECTURE (HLA)

2.1 HLA Architecture

The United States military makes extensive use of simulations for the purposes of training as well as tactics, strategy and doctrine development. The High Level Architecture (HLA) is the standard for these simulations, replacing the previous Distributed Interactive Simulations (DIS) standard. Several papers provide an overview of the HLA, the rationale behind it and the standardization process [DFW98, DKW98]. Note that the initial development of HLA by the United States Defense Modeling Simulation Office (DMSO), HLA has been standardized as the IEEE 1516 standard. For the sake of simplicity and consistency with earlier documents, this thesis uses HLA to refer to both the original HLA as well as the current, standardized IEEE HLA.

Note also that the HLA standard is a general-purpose architecture for simulation that is applicable to a wide range of simulations. Most prominent in the literature are simulations involving military branch applications. Thus while this thesis, and the vast majority of the references, use military examples, the HLA standard is not limited to such simulations.

The (DMSO) describes three broad categories of simulations [DMSO04]:

Live: A simulation involving real people operating real systems (e.g., airplanes, …);

Virtual: A simulation involving real people operating simulated systems. Virtual simulations inject human-in-the-loop in a central role by exercising motor control skills (e.g., flying an airplane), decision skills (e.g., committing fire control resources to action), or communication skills (e.g., as members of a C4I team);

Constructive Model or Simulation: Models and simulations that involve simulated people operating simulated systems. Real people stimulate (make inputs) to such simulations, but are not involved in determining the outcomes.

In the context of the HLA, a federation is a set of simulations, a common Federation Object Model (FOM) and supporting runtime infrastructure, that together to form a larger model or simulation. A federate is a member of a federation; one point of attachment to the infrastructure. A federate could represent (i.e., simulate) a single platform, such as a tank, or an aggregate simulation such as a battalion of tanks. A federation execution is a session of a federation executing together.

The HLA is an event-based and component-based architecture that supports distributed components, and thus distributed simulations. HLA also supports the

concept of data abstraction: federates can use HLA to share data elements without HLA knowing what types the data elements represent.

The HLA standard is defined in three parts:

> *HLA Rules*: principles and conventions that must be followed to achieve proper interaction of federates during a federation execution.

> *Object Model Template (OMT)*: a meta-model used to describe a federate's Simulation Object Model (SOM) which describes the data the federate can produce or consume; and a FOM that defines what part of the union of all the federates' SOMs will be used in the federation. Note that the SOMs (and thus the FOM) only define those data elements that will be shared between federates. The FOM can contain both HLA objects and HLA interactions. Objects contain are used for simulated entities (such as a ship or tank) and have persistent state data expressed as attributes. Interactions are used for simulated occurrences (such as an explosion) and do not have persistent state. Each HLA object or interaction can have one or more attributes associated with it. The FOM defines the object and interaction classes – federates create object instances using these classes.

Interface Specification: defines the interface between federates and the Runtime Infrastructure (RTI). The RTI is middleware that allows a federation to execute together.

2.2 HLA Run Time Infrastructure (RTI)

For the purposes of this thesis, the RTI is the key. The RTI is the middleware layer that individual federates (i.e., simulation programs) connect to receive simulation management services and to publish and subscribe to data. Note that the HLA specifies the interface to the RTI as a standard but does not prescribe the actual implementation. The RTI interface describes six services.

Federation Management services include definition of a federation execution and membership and federation-wide operations such as synchronization and checkpoint/restore.

Declaration Management services are used by federates to declare their intent to publish or subscribe to data.

Object Management services are used by federates to register new instances of objects with the RTI and to update their attribute values (i.e, exchange data with other federates). Note that compliant RTI-s must support both “best effort” and “reliable” transport types. Other types are allowed for, but not specified, in the HLA standard [IEEE00].

Ownership Management services provide for the management of instance attributes. At any one time, only one federate owns (is responsible for) a

given instance attribute. This ownership can be shared or transferred among federates. Note that federates own attributes, not instances of objects. Thus an instance of object airplane that has attributes of lat, long, altitude and ice_loading could have attributes lat, long and altitude owned by one federate, that simulates the airplane's movements, and another federate, that simulates weather-induced effects, could own ice_loading.

Time Management services allow federates to advance their logical time and controls the delivery of time-stamped events. [Fuj03] provides more detail on time management in HLA.

Data Distribution Management services, used in conjunction with Declaration Management, provides a finer grained publication and subscription definition capability. It allows federates to associate their data publications and subscriptions to abstract regions called routing spaces. The goal of these services is to optimize use of federation network bandwidth by providing information to the RTI that allows it to route data only to interested federates.

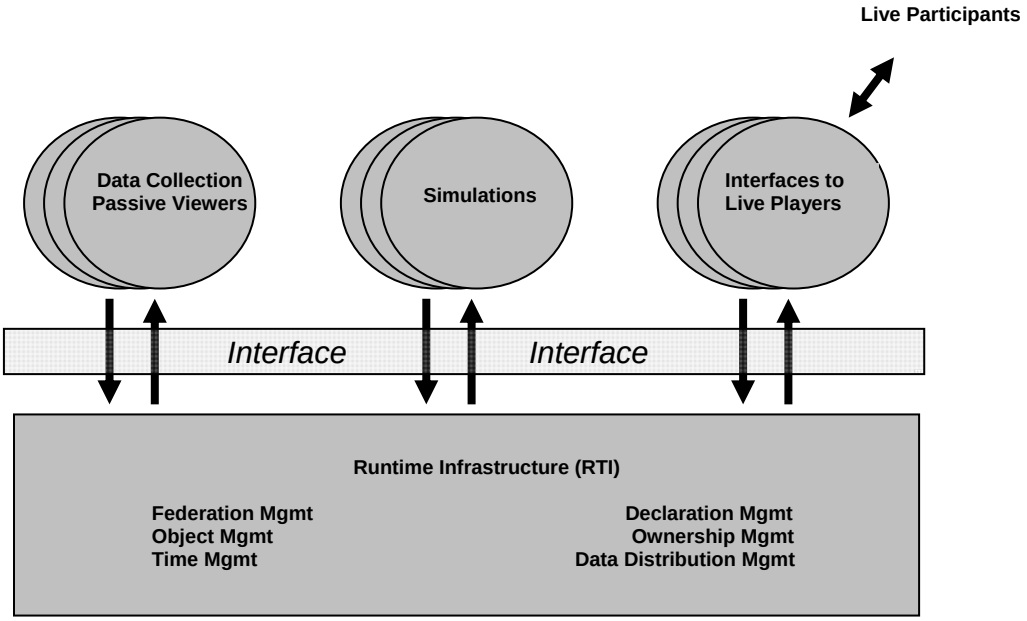

Figure 2.1 Functional view of an HLA Federation from [DFW98]

The distributed, object-oriented nature of the HLA invites comparisons with other distributed, object-oriented middleware standards. [KWD99] briefly compares HLA to CORBA while [BJ98] provides an overview of HLA, CORBA and RMI and then notes the following key differences:

- While both CORBA and HLA support multiple language bindings, HLA places the burden of language interoperability on the RTI implementer. RMI is Java-based and thus essentially not cross-language
- HLA is oriented towards simulation and thus provides time management features that CORBA and RMI do not
- HLA supports the transfer of object ownership between federates, a capability that CORBA and RMI lack

2.3 Group Communication in HLA

The HLA standard defines several concepts and mechanisms similar to the properties used to formally describe a group communication system [CKV01]. Relative to formal group communication systems, HLAs communication properties are generally inferred from the standard and less rigorously defined.

The RTI provides methods for federates to join and leave a federation and maintains a list of all federates that have joined a federation but there is no expectation of failure detection and thus no guarantee that this group view is current or accurate at a point in time. The RTI also tracks all federates publications and subscriptions, but again, there is no guarantee this information is correct. Given this, federates publish object attributes to a "channel" without any knowledge or assurance that two consecutive publications will be received by the same group of subscribed federates. Thus HLA does not support strong group membership properties such as group view liveness [CKV01] (i.e., HLA can not guarantee an accurate group view.).

The HLA standard calls for two transport methods, "HLA best effort" and "HLA reliable." "The IEEE standard defines "HLA best effort as "*makes an effort to deliver data in the sense that UDP provides best-effort deliver*" and "HLA reliable" as "*provides reliable delivery of data in the sense that TCP/IP delivers its*

data reliably." An RTI user can infer that "reliable" data will be uncorrupted and retransmitted, if necessary, by the RTI, and that duplicates will not be delivered.

The HLA standard also calls for two ordering types: "receive order" and "time stamp order (TSO)." The standard describes "receive order" as having "*no ordering guarantee*", that messages "*will be received in an arbitrary order by the respective joined federate*", i.e., unordered. "Time stamp order" is defined as "*Messages having different time stamps are said to be delivered in TSO if for any two messages M1 and M2 (time stamped with T1 and T2, respectively) that are delivered to a single joined federate where T1 < T2, then M1 is delivered before M2*." , i.e., per federate casual delivery[CKV01]. Note that this is timestamp is based on a logical time.

Note that both of HLA's ordering definitions are on a per receiving federate basis. This is consistent with HLA's lack of a rigorous definition for group membership and lack of a group view concept. Given this, the standard provides no atomicity guarantees – for instance, no assurance that all federates will receive a given message.

2.4 Typical HLA Usage and Requirements

The HLA provides a general-purpose simulation infrastructure that is used by a wide range of simulations and thus presents the RTI with a wide range of requirements. Virtual simulation exercises may include 10 federates representing

10 tanks located at several dispersed geographical locations. Similar to a computer first person shooter (FPS) game, the ability of the federation to provide appropriate response times to the human participants is important. This human-in-the-loop class of simulation requires the simulation to be paced by wall clock time – where low predictable delays are desired, and thus time stamp ordered (TSO) message transport is avoided [Fuj03].

Constructive simulations, by contrast, are similar to real time strategy (RTS) computer games – while humans interact with the simulation, they do not provide "real time" inputs that determine the outcome of the simulation. The scale of constructive simulations can be the key challenge as exercises can include hundreds of nodes, millions of simulated entities and thousands of multicast groups [HTW03, HWT01, McG98]. In these simulations, network bandwidth and throughput considerations become constraints.

HLA simulation requirements have been described [Pul99a, McG98] as follows:

- Up to 100,000s objects per federation execution
- Typical message size of 200 to 270 bytes per message
- Typical message creation rates between .2 and 15 packets per second per simulator
- Peaks of 500 packets per second (assumed to be messages per second) per group

- Packet delivery loss of less than 2%
- Use of up to 3,000 multicast groups to represent routing spaces
- Join latencies of less than 1 second
- Message delivery latencies on the order of a few hundred milliseconds, maximum
- 90% of group members both send and receive messages

Note that recent constructive simulations report object counts 1.5 million using 100,000 multicast groups while running on over 100 compute nodes [HTW03, GAD05].

2.5 Routing Spaces and Network Performance

DMSO sponsored the development of HLA to promote reuse and interoperability of simulations as well as to address limitations of the previous standard, Distributed Interactive Simulations (DIS), specifically in the context of the goal of supporting larger simulations. Central to the challenges faced by both DIS and HLA is the volume of data generated by large simulations where network bandwidth becomes the limiting constraint. RFC 2502 [Pul99a] provides a brief overview. Several papers from the early development of HLA [CCM+97, RSM97, VC98, VM96, VRC96] provide a detailed look into the design tradeoffs considered in meeting this data delivery problem. The problem has both the potential to satu-

rate network connections as well as to require excessive federate processor resources to filter out, at the receiver, unwanted data.

A key approach in implementation of the Data Distribution Management (DDM) services was the concept of an abstract routing space [VRC96]. Publishers and subscribers could then specify spaces of interest within a routing space called a region. One method to map publishers to interested subscribers was to map federate publishing and subscribing interests onto region grids [RV96]. These grids can then be mapped to multicast groups. [HWT+01] points out that in large scale, (i.e., constructive simulations with tens or hundreds of thousands of simulated objects) static mappings of grids into multicast groups allows the simulation to offload much of the data distribution filtering to the network hardware. Other papers [CDW00, McG98, McP03] provide additional experience of RTI performance in large-scale simulations. [MBD00] discuses the potential complexity of such a mapping and demonstrates the benefits of a dynamic multicast mapping scheme. [Woo02] provides a description of a recent, Commercial off the Shelf (COTS) implementation of DDM. [BD01] provides performance comparisons of several different DDM strategies.

Consider the following tradeoff. A high resolution grid with a high cell count allows for finely-grained pub/sub interest mapping but can quickly use all available IP multicast groups. Federates whose pub/sub interest mapping changes rapidly – such as a simulated high-speed airplane flying over a physical area mapped as a

high resolution grid – can experience extremely high rates of interest joins and leaves.

DDM routing spaces are identified in the FOM and passed to the RTI via a .FED file. For example:

```
(class tank
  (attribute latitude reliable receive Ground)
  (attribute longitude reliable receive Ground)
)
```

defines a tank with attributes of latitude and longitude. Both attributes are transmitted using HLA's "reliable" transport method and use HLAs "receive order" ordering. Both attributes are also bound to the "Ground" routing space. Federate run time RTI calls define a region within the routing space by defining the dimensions and bounds. Calls to the RTI UPDATE_ATTRIBUTES_WITH_REGION enable the federate to publish these attributes to all subscribers joined to that region.

DDM and its ability to define routing spaces and regions that are then mapped to IP Multicast groups is a significant tool in support of scaling HLA simulations up such as in large-scale constructive simulations.

2.6 RTI-s Implementation of the HLA RTI

The RTI-s implementation of the HLA RTI standard was developed in support of the requirements of a specific class of simulations: large scale constructive simulations. Specifically, the DARPA Synthetic Theater of War (STOW) simulation exercise [CCM+97]. This class of simulation required large numbers of federates, high numbers of simulated entities and corresponding high amounts of network traffic. Thus, RTI-s was built with scalability as a key design goal.

Support for robust and diverse networking capabilities has been a key driver in the design and implementation of RTI-s since it's inception [VM96, CCM+97, BV99]. The design featured a Transport Manager layered on top of multiple transport services layered on top of a Stream Manager. Below the Stream Manager were multiple communication managers (UDP, TCP, ..). Subsequent redesign of RTI-s' communications infrastructure layers three message services below the transport services and combined some Stream Manager functionality with that of the communication managers into a set of chained protocol modules, configurable at runtime, similar in architecture to that used by Hours [VBM96].

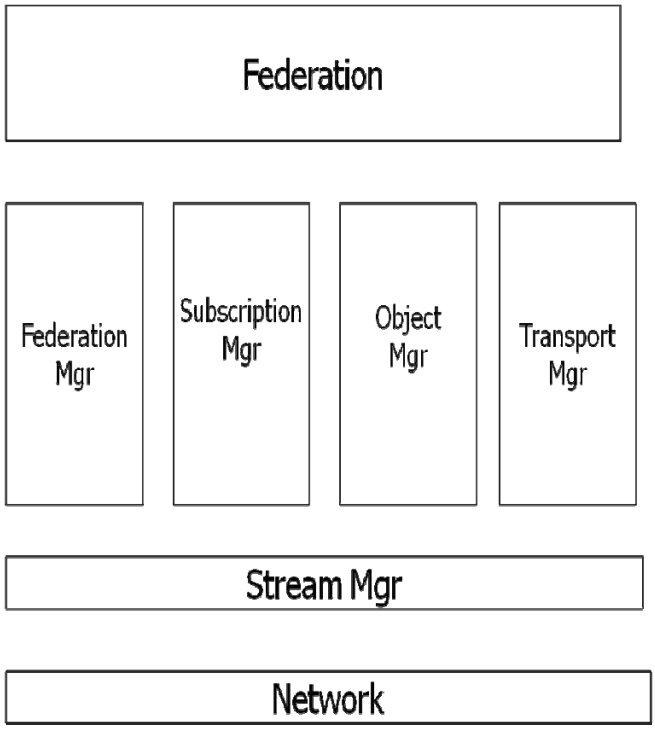

Figure 2.2 HLA RTI-s Architectural Overview

Transport Manager – manages the individual transport services and provides marshalling and de-marshalling of HLA object/attribute pairs into RTI's message buffer. Transport Services supported are:

Minimum Rate Service – provides ongoing transmission of messages – in essence a "keep alive" heartbeat. Uses the Best Effort message service.

State Consistent Service – uses the Best Effort message service and a Consistency Protocol (CP) to maintain a consistent state of object attributes across multiple federates [VCS95].

Simple Service - uses the One Shot, Best Effort and Reliable message services to transmit messages.

The following Message Services are supported:

Best Effort – uses underlying UDP protocols to transmit messages.

One Shot – queues messages internally for transmission at a future point in logical time using UDP protocols.

Reliable – Uses a NAK-based protocol to reliably send messages using the UDP protocols.

The following chained protocol modules are supported:

MulticastEmul – provides multicast emulation.

Bundler – bundles and unbundles messages, using an interrupt timer, to achieve higher throughput at the cost of increased latency.

Ratelimit – limits the rate of transmission.

Fragment – provides message fragmentation and defragmentation to match the requirements of the underlying communications network.

UDP conn – maps to underlying UDP network protocol.

TCP conn – maps to underlying TCP network protocol.

IP multicast – maps to underlying IP multicast protocol.

Note that RTI-s uses the construct of a Stream identifier which is an identifier for publish/subscribe interest groups. The mapping between Stream identifier and

underlying communication paths is determined by runtime configuration files. Typically, RTI-s maps DDM regions in the routing space to IP Multicast groups. Transport and message services and chained protocols all address messages in terms of RTI-s Stream identifiers. An exception to this is MulticastEmul that uses RTI's internal database of publication and subscription information to provide multiple UDP unicasts to emulate IP multicast.

RTI-s supports configuration of many networking parameters at federate start up via the .RID file. The makeup and order of the chained protocols are specified there as well as the transport services available to the federate. A variety of network parameters such as socket poll interval or minimum rate heartbeat interval, as well as various non-network RTI configuration parameters, are set there.

RTI-s is single threaded utilizing it's own internal scheduling mechanism. The simulation application (federate) periodically makes a call to the RTI::Tick(min, max) method specifying a minimum and maximum time RTI's internal scheduler is allowed to use for internal processing such as the receipt of incoming messages. During the Tick processing time, RTI-s performs various internal management functions such as polling all open sockets, building an event queue of sockets with data awaiting processing, and initiating call backs to the simulation program.

RTI-s comprises several hundred C++ and several dozen C modules and runs on Windows, Linux and several varieties of Unix systems. RTI-s provides C++, C, Java and ADA APIs and supports two different versions of the pre-IEEE HLA standard.

3 RELIABLE MULTICAST PROTOCOLS

3.1 Survey

A review of reliable multicast protocols can be considered in how they answer the following three questions: What is the goal of the design? What are the key features of the design? What is the performance of the resulting protocol?

Common design goals may include a focus on one or more group communication properties such as safety, ordering and reliability and liveness [CKV01]. Other considerations include scalability and performance, typically in terms of throughput and latency. A goal of the design may be support for large numbers of multicast groups and for multiple senders per group. Key features of the design may include peer to peer based-design or one with proxy/infrastructure nodes. Caching or logging of messages, by someone other than the sender, may be provided. Damaged messages may be replaced and/or repaired. The protocol may be layered on top of unreliable IP multicast or totally standalone. The design features answer key questions such as: Who detects loss - sender, receiver, other? Who performs repair? Performance considerations include throughput, latency, time to detect loss, time to repair loss, protocol overhead under non-loss conditions, protocol overhead under loss conditions.

[BH0+99] describes the bimodal multicast protocol based on gossip or epidemic algorithms [DGH+87]. Bimodal multicast's design goals seek to strike a balance

between support for strong reliability properties on the one hand and performance and scalability on the other. Key features include a peer-to-peer design with cache distributed among all nodes. Following initial message transmission by an unreliable multicast such as IP multicast, individual nodes randomly select other peers to exchange digests of received messages. Nodes that thus detect a loss can request missing packets form the sender of the digest. Performance of the protocol includes excellent throughput, scalability and stability under a range of loss conditions. Horus [VBM96] design goals include a focus on providing a system that could be easily reconfigured to support a range of design requirements, from the rigorous virtually synchronous model to other models with less guarantees but potentially better performance. A key design feature of Horus is a modular protocol stack design that can be reconfigured at runtime to meet a range of requirements.

The Narada protocol [CRS+02] design goals are focused on addressing the weaknesses of IP multicast: scalability, lack of reliability guarantees, and the requirement for changes to network infrastructure to enable deployment. Narada uses a self organizing, tree-based, overlay network that is self-improving and adapts to dynamic characteristics of the underlying network. Performance is described in terms of throughput and latency for a range of group sizes. Overcast [JGJ+00] design goals also focus on ability to deploy over existing networks and support for large-scale and reliability. Overcast supports a single-source/single-sender. Key features include a self-organizing overlay tree and a number of internal

Overcast nodes spread throughout the network to data replication. Overcast uses TCP for connections between nodes thus providing ordering and reliable node-to-node communications and leverages the cached data to efficiently support late-joiners.

The design goals of Scalable Reliable Multicast (SRM), as the name suggests, is to provide reliable multicast for large scale deployments. [FJL+96] describes the protocol and its use in a large-scale shared whiteboard collaboration application. Key features include its use of IP multicast for initial delivery with end-nodes responsible for detecting loss and sending a NAK back over the multicast group to request repair. Hosts also having the same loss will suppress their NAK upon hearing a NAK for the same data. Any host on the multicast group can respond repair requests. Pragmatic General Multicast (PGM) [GMS+03] also focuses on scalability as the key design constraint. Also like SRM, PGM design goals do not include total ordering or acknowledged delivery to a known group. PGM features a proxy/infrastructure-based design that introduces PGM-capable Network Elements (NE), typically network routers, and separate Designated Local Repairer (DLRs) nodes. Initial packet delivery is via an unreliable multicast such as IP multicast. Request for repair are unicast up to a node's parent the PGM tree with repairs multicast back down. The tree structure enables more aggressive NAK surpression and constrained forwarding of repairs (limited to sub trees in need of the repair). PGM also uses Forward Error Correction (FEC) to enable multiple packets to be repaired with a single FEC repair packet. PGM provides congestion

control information back to the source to allow the source to adjust its sending rate but no flow control mechanism to avoid overflow of the receiver's buffer.

The Slingshot [BPB05] design is focused on the requirements of datacenter-based multicast: high throughput and time-critical (i.e., low latency). Design features include using IP multicast for the initial delivery and then having end-nodes exchange receiver-generated FEC packets in a random gossip style. Similar to Bimodal multicast, Slingshot imposes a constant overhead on the network in the form of, in this case, FEC repair packets. Ricochet's design goal also includes support for low-latency performance and adds support for multiple groups [BBP+07]. Key design features include a peer-to-peer design that uses IP multicast for the initial transmission and the construction of Lateral Error Correction (LEC) packets that span multiple multicast groups. Ricochet argues for the creation of multi-group regions based on those nodes with intersecting group membership. Losses that are not recovered by the LEC packets are recovered by a secondary NAK layer.

Three reliable multicast protocols were designed to support distributed simulations. The design goals of the Log-Based Receiver-reliable Multicast (LBRM) [HSC95] were support for low rate updates of information that never-the-less has a requirement for freshness (i.e, receiver's copy is up to date with the sender's) and optimization of low bandwidth tail circuits. Both of these goals were directly shaped by the needs of the Distributed Interactive Simulation (DIS) standard that

was replaced by HLA. LBRM uses distributed logging (i.e., caching) servers to support scalability and maximize tail circuit bandwidth utilization and minimize recovery latency. Selectively Reliable Transport Protocol (SRTP) is also aimed at DIS simulations and leverages best effort transmission of data to carry sequence numbers for infrequently changing reliable messages [Pu99]. This enables receivers to detect reliable message loss without additional receipt of reliable messages or overhead packets. Lastly, the RTI-s reliable protocol is based on data consistency protocols initial developed to support DIS simulations [VCS95]. RTI-s reliable uses IP multicast for initial transmission and relies upon receivers to NAK back to the multicast group for lost or damaged messages. It adds a periodic sender-to-receiver heartbeat as a secondary layer of reliability.

3.2 The QuickSilver Scalable Multicast (QSM) Protocol

QSM is a recent multicast that, like Slingshot and Ricochet, has design goals centered on high throughput, low latency performance in datacenter operations while providing scalability in number of nodes and number of groups [OBP06, OBP05, OB06, OB07, KBD07]. Like Ricochet, QSM argues for the use of multi-group regions and leverages that construct by adding recovery protocols that are region aware and localized

The concept of regions is as follows. Assume some number n of nodes who are members of some mix of multicast groups A, B and C. Let $nodes_A$ be the set of

all nodes who are members of only the A multicast group and $nodes_{AB}$ be the set of nodes who are members of only the A and B groups. Thus with the three groups A, B and C, there could be at most seven regions: $nodes_A$, $nodes_B$, $nodes_C$, $nodes_{AB}$, $nodes_{BC}$, $nodes_{AC}$ and $nodes_{ABC}$. Regions are thus non-overlapping selections of nodes that share multicast group membership and thus exhibit interest sharing. QSM leverages that by creating a new multicast group for each region thus allowing a receiving node to be part of a group with identical interests. This enables recovery, as experienced by any one node, to be based on a single multicast group. Note that this approach may be thought of as being receiver-friendly in that receiving nodes can leverage the resources of their aggregated group (region) for recover purposes. On the other hand, senders must multicast multiple copies of a message – one for each region that has that data source as a component.

QSM leverages these region groupings by caching region messages and running its recover protocol inside the region. QSM subdivides a region of n nodes into k partitions, where k = m / r and r is typically 5. Each node in a partition then caches 1/kth of the incoming messages. Each partition has a partition leader that circulates a token through the partition members in a loop. The token contains ACK/NAK information to enable, using the cache distributed through the partition, local recovery – partition members use the token to push and/or pull recovery packets from the partition cache. A region has a region leader identified that generates the region token and circulates it in a loop among partition leaders.

Thus each region uses a single token that is generated by the region leader and passed among partition leaders and by these partition leaders among partition members. Partition leaders use the ACK/NAK information stored in the token by partition members to send NAKs, as needed to senders. When the token returns to the region leader, it generates an ACK sent to the sender to allow cleaning of sender message buffers. The token also accumulates region control information such as the maximum sequence number seen in the region. Thus through interest sharing (mapping nodes to regions) and dividing regions into partitions with message cache distributed evenly among partition members, QSM seeks to perform local recovery whenever possible and to minimize NAK implosion by leveraging partition leaders to NAK for their partition. Note that the current unmanaged implementation of QSM, used by this research, only supports a configuration file read at startup time for a node to specify what multicast group(s) it wants to join.

QSM is implemented using C# as a managed Windows application. QSM runs as a single thread with its own scheduling system that includes an alarm queue to hold timer events. QSM leverages the Windows kernel-based I/O completion port by binding all send and receive sockets to this single port. A novel feature of the QSM design is its support for pulling of messages from nodes. Applications can register with QSM their intent to send data. Based on QSM's internal scheduling algorithm, when QSM is ready to send it can then pull data from the application with the goal of reducing polling overhead.

QSM performance [OBP06], on tests run on a 110 node cluster, is shown to scale well with excellent throughput and latency. Latency is reported ~ 10 – 12ms for packets of 256 bytes and smaller and 25 ms for 1024 byte packets. Using 1024 byte packets, throughput remains steady at 8K packets per second (pps) under normal (no loss) conditions. In cases of bursty loss, simulation by having a selected node drop 1 second of messages every 10 seconds, QSM reports steady throughput of ~ 7K, pps with latency rising from 25 ms to 660 ms for groups of 20 nodes and a maximum of 1.7s with 110 nodes.

The HLA standard is used for a wide range of simulations. Constructive simulations, built using the HLA standard, tend to push the limits of scalability in terms of number of entities being modeled, number of nodes and number of groups [CDW00, GAD05]. Execution of constructive simulations on datacenter clusters where high throughput and low latency are essential is common [GAD05]. QSM appears to be a good fit for these constructive simulation requirements of scalability and high networking performance and was thus selected for this research.

4 QSM INTEGRATION INTO RTI-S

Although QSM is implemented as a Windows .NET managed application, a version of QSM with an interface for unmanaged applications, such as RTI-s which runs as an unmanaged Windows application, was available. This unmanaged interface exposes an enqueue and dequeue (i.e., send and receive) method. Note the dequeue method can be either blocking or non-blocking. Joining of multicast groups is supported via an XML configuration file that is read at QSM application startup. A QSM-enabled application is started up by a QuickSilver host program. The QSM-enabled application must be a Windows .DLL file. Thus the command:

QuickSilver_h.exe –c:myapp.xml –a:myapp.dll

Starts up the QSM enabled application myapp.dll using myapp.xml as the configuration parameters. Note that QSM relies on a provided Group Management System (GMS) that must also be running. The GMS tracks membership and updates group views as QSM applications connect and leave groups.

As described above, layered below the HLA standard UPDATE_ATTRIBUTE_VALUE API call, RTI-s is architected into a number of supporting functions and layers and is driven by two configuration files, the RID and the FED file. The RID file includes parameters to configure the protocol stack and list the protocols available. The default configuration of RTI-s' uplink

(i.e., its' network connection point) uses the MULTICAST_PATH RID macro that expands into the stack of udp_multicast, rate_limit, fragment_reassembly and bundler. Udp_multicast is the lowest element of these layers and makes direct Winsock network calls. Udp_multicast exposes to the upper layers methods such as send, receive, join and leave (i.e., multicast groups) as well as various internal group subscription management methods.

QSM was integrated into RTI-s by replacing the udp_multicast class with a QSM_multicast that implemented the RTI-s send and receive methods by mapping to the QSM enqueue and dequeue methods. The default RID file uplink protocol stack configuration was replaced with a newly defined "QSM_multicast" identifier. RTI-s running of the non-QSM versions of the benchmark programs was observed to identify the multicast groups used. These groups were then added to the static XML configuration file and the RTI-s calls to join and leave were stubbed out. RTI-s' internal SteamID, which is its internal representation of a multicast group id, was then used to map to QSM's corresponding internal representation, called a channel number. RTI-s uses two multicast groups (SteamIDs 1 and 2) for various internal control purposes. In addition, RTI-s uses up to ~ 70 HLA interactions, each mapped to a unique multicast group, for various federation control purposes. In addition to these system groups, each federation uses one or more multicast groups for user-defined interactions and object attributes updates. In the case of the thruput and latency programs, ~ 10 user-defined interactions, used to coordinate the benchmark run, are mapped to multicast groups

were mapped to interactions and the actual object attributes being updated are mapped to one multicast group.

Note that the system multicast groups typically exchange messages at federation startup and shutdown were essentially quiet during benchmark runs. The benchmark programs used their interactions to coordinate runs but not during the runs themselves. Thus essentially the only network traffic from the benchmarks during their run was directed to this single multicast group used to update the benchmark object attributes. This multicast group usage was determined by observing the benchmarks running in debug mode and confirmed by watching benchmark runs using RTI-s' integrated debug console.

QSM_multicast retained udp_multicast's existing subscription management methods to enable RTI-s to continue to manage the mapping of interactions and object updates to SteamIDs. Note that the purpose of this integration was to support experimentation with reliable protocols thus the group management aspects of RTI-s and QSM (i.e., its GMS) were not integrated.

As described above, the Tick command invokes RTI-s' internal scheduler with a "time allowed" parameter. RTI-s maintains an internal event queue to process, as time allows via the Tick command, various internal events such as federate callbacks and polling of network sockets. Socket polling is also initiated by a timer. The timer interval is a configuration parameter set in the RID file with a default value of 10ms. QSM_multicast detects incoming network packets by using RTI-s

scheduled socket polling method to invoke the non-blocking QSM dequeue call. RTI-s' Buffer Manger maintains a buffer object that is exposed to all layers of RTI-s' internal stacked protocols, its Steam Manager and various components of its Transport Manager. At its lowest level, it exposes a pointer to a buffer maintained in and by the Buffer Manger as well as a size of that buffer. Udp_multicast passes that buffer pointer and size to the Winsock call. Upper layers of RTI-s later declare that buffer available for reuse. QSM, in interests of performance, does not do a "copy on write" of the application-provided buffer upon invocation of the enqueue method. The application is not free to reuse or free that buffer until receiving a subsequent response from QSM as part of its dequeue method return. The integration of QSM into RTI-s required the creation of a separate buffers to pass to QSM and subsequent freeing of this buffer. Benchmark runs of RTI-s without QSM but with a similar malloc/copy/free "handicap" showed only marginal impact from this added work. Specifically, simulation performance was essentially the same (within the range of normally observed variation between identical benchmark runs). CPU loading of the "handicapped" version of RTI-s was observed to ~ 3 - 5% higher than the normal version.

The RTI-s architecture supports the definition of only a single uplink (i.e., network connection) which is bound at application start up, via the RID file, to the lowest level of the protocol chain. By default, this lowest level chain member is udp_multicast. Thus for purposes of this experiment, one version of RTI-s was compiled and linked that had uplink bound to QSM_multicast and another that

was bound to the existing udp_multicast. Benchmark runs using QSM were via this QSM version of RTI-s while runs of RTI-s using best effort and reliable protocols were using the udp_multicast version. The benchmark programs' binding of object attributes to either best effort or reliable transport protocol is via the FED file. Thus two versions of the FED file were maintained to bind the attribute updates to either best effort or reliable.

5 HLA BENCHMARKING CONSIDERATIONS

The evaluation of performance of HLA simulations and the RTI has been reported in a number of papers [KLK+02, FMF02, JBC+00, Dor01, FH98, CDW00, GAD05]. Early in the development of the HLA standard by the DMSO, several HLA benchmark programs were developed. These programs represent the closest thing to a standard HLA benchmark. As an established, defacto standard, they were chosen for this research.

Specifically, the DMSO thruput and latency programs were used. Both represent unit testing done at the application level. As unit tests, they evaluate discrete aspects or subcomponents of the HLA simulations system. Focusing on the specific components or subsystems, essentially scopes down the test to a narrow focus, resulting in a less complex test environment and supporting better control of the test variables. Results can then be more understandable and reproducible. These benchmarks also measure performance from an application-level perspective, not from a computer system or network perspective. This combination of unit measurements at the application level allows HLA researchers and practitioners to understand the results in the context of actual HLA simulation performance. Specifically, thruput and latency measure the standard HLA data transfer API call of UPDATE_ATTRIBUTE_VALUES.

Thruput performs a series of cycles of UPDATE_ATTRIBUTE_VALUES as controlled by a number of user-supplied test parameters. Parameters include:

- Number of updates performed per cycle
- Number of cycles to perform
- Size of attribute data being updated

A single federate acts as the benchmark controller and sends all updates. The beginning and end of both the benchmark and each cycle is managed by this controller via a set of user-defined HLA interactions. The controller begins a cycle by sending an interaction with the number of updates about to be sent as parameter attached to the interaction. Thruput receivers, also known as reflectors based on the HLA REFLECT_ATTRIBUTE_VAUES method they implement, then watch for interactions until that number has been received or a timeout value has been reached. At the end of the cycle, receivers report back to the controller the actual number of updates they received. These values, received by the controlling copy of thruput from each thruput receiver, are used by the controller to calculate benchmark loss rates. The sending (controller) federate reports the number of updates sent and the time to perform the updates and the corresponding UAV (update attribute value) per second rate. Receivers separately report the number of updates received (i.e., reflected) and their corresponding RAV (reflect attribute value) rate.

These measurements are thus not end-to-end measurements. In both cases, the time measured is from the first HLA call, UPDATE_ATTRIBUTE_VALUE (sending side) or REFLECT_ATTRIBUTE_VALUE (receiving side), to the last call. The RTI may queue up the update/reflection at the sender/receiver side, or may be buffered by the operating system at either side. In an extreme case, all the updates could be queued at the either (or both!) side reducing this benchmark to only a performance measure of API invocation of an RTI implementation!

The latency benchmark does provide an end-to-end measure. Like the thruput benchmark, latency has a single copy acting as the benchmark controller. The controller uses the operating system clock to create a timestamp that becomes one of the attributes being updated (sent). Upon receipt by the receiver(s), the attributes are sent back to the controller via another UPATE_ATTRIBUTE_VALUE call invoked by the receiver. When the controller receives the attribute, its timestamp is compared with the current system clock to calculate the end-to-end (i.e., round trip time) for that packet. The latency benchmark only sends a single update and waits for its return before sending the next update.

The following considerations led to the use of both the thruput and latency benchmark for this research:

- Defacto standards cited by papers in this field
- Enables results to be compared with related research

- Unit testing approach allows good control of conditions and thus supports reproducibility
- While application-level benchmarks, they feature being measured is a fundamental action closely related to network performance and thus more meaningful in relationship to results from reliable multicast papers.

HLA was designed to support distributed simulations and in practice are run on LAN, WAN and hybrid network topologies. In this context, hybrid refers to the case of several LAN-based nodes at one geographical location acting as part of distributed simulation with other components connected via WAN. This research was conducted on a homogenous LAN-based cluster of nodes running Windows 2003 Enterprise Edition server on 1.3 GHz Intel Pentium III processors with 512MB memory. Intel 100 Mbps network adapters connected to a switched network were used. All nodes were in the same subnet with node-to-node latencies of less than 1 ms as reported by the Windows ping command. Benchmark runs were made with essentially exclusive access to the cluster and only typical operating system processes running. In between benchmark runs, the ambient or background processor and network interface adapter (NIC) load was typically reported as less than 5%. Processor and NIC utilization was observed, both during and in between runs, using the Windows Performance Monitor. The cluster used did not provide any cluster job scheduling or queuing system and the thruput and latency programs themselves are standalone console applications that take input via command line (argv) input and write results to stdout. To facilitate the research, a

script-based control program was developed to deploy and update benchmark programs and configuration files, launch runs of two or more nodes, kill hung benchmark programs and gather resulting output files.

A typically used mean message size used in HLA simulations, and in HLA benchmarks, is 128 byte messages. 128 byte messages were used for this research. RTI-s provides message bundling as an optional protocol chain element. In the case of a 128 byte message, RTI-s can bundle up ~ 10 such messages into the 802.3 MTU thus significantly reducing the number of network packets sent and potentially increasing throughput. RTI-s includes a user-controllable (via the RID file) bundle time out value and bytes remaining value to control the action of the bundler. Note that the thruput benchmark could be considered a best case scenario for such message bundling in that it sends bursts of messages to the same multicast group such that bundles would fill prior to the expiration of the bundle timer. Such a scenario would tend to maximize network bandwidth with little or no increase in latency. The current implementation of QSM does not do message bundling. That is, each message sent translates into at least one network transmission. In the interest of the comparing core protocols on an equal footing, all tests were done with RTI-s' message bundling turned off.

Note that since the benchmark programs were written, the HLA standard has added support for federate synchronization via synchronization points. This RTI-supported construct provides coarse-level synchronization, allowing multiple fed-

erates to wait on a point for other federates to join. Such a device could be used, for example, to coordinate multiple benchmark programs (federates) running a benchmark. The thruput and latency benchmarks were originally written before HLA provided this synchronization capability thus they used a set of user-defined interactions to synchronize benchmark runs. Under the initial use of these benchmarks running under RTI-s, they failed to successfully exchange these interactions and thus were unable to synchronize and perform the benchmarks. Careful tracing of the handling of these interactions down through the sending federates RTI-s call stack and up through the receiving federate's stack identified a bug in the production RTI-s code. Several of these user-defined synchronization interactions contain no attached parameters. The RTI-s Transaction Manager, upon receipt of such interactions from the RTI-s Stream Manager (i.e, as the interaction moves up the call stack to the receiving federate), intentionally drops these interactions. The HLA standard under which RTI-s was written allows for interactions without parameters thus the RTI-s is a bug and was a surprise to the RTI-s programmer. Commenting out this code in Transport Manager enabled the benchmarks to synchronize and run successfully. Note that the current IEEE 1516.2000 HLA standard has an implicit requirement that interactions include parameters.

6 EXPERIMENTAL RESULTS AND ANALYSIS

As previously stated, except as noted, all results are for 128 byte packets and all nodes are homogeneous 1.3 Ghz Pentium III running on Windows 2003 Server Enterprise edition. Unless otherwise noted, all runs were done using cycles of 1000 updates per cycle.

Figure 6.1 shows the performance of RTI-s running the thruput benchmark using best effort (i.e., UDP multicast). Recall that the benchmarks use a single sender – all other nodes are receiving or reflecting nodes. The graph shows two characteristics that were common to almost all runs of all three protocols: the thruput of the sending federate was always noticeably greater than that of the receiving federate and performance was essentially level across a range of number of nodes. Also note that thruput results are reported, as measured by the benchmarks, in application units. Thus the rates are for the HLA UPDATE_ATTRIBUE_VALUE method in UAVs per second.

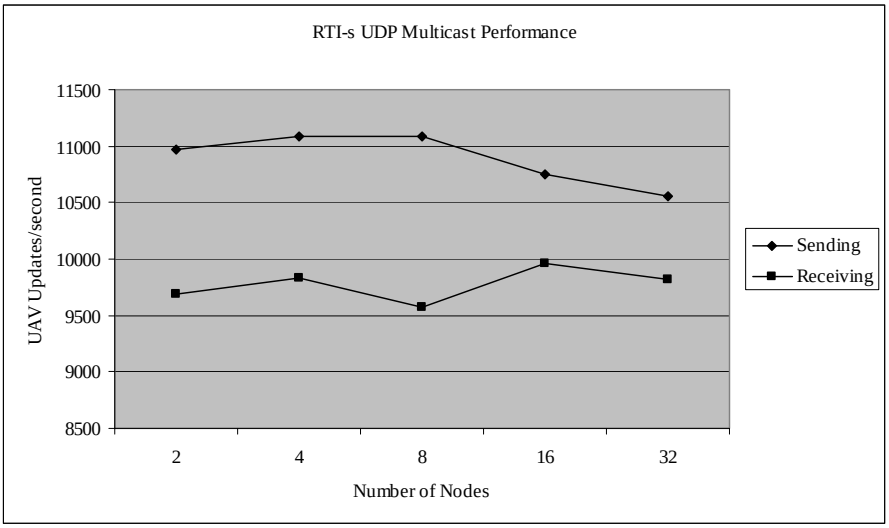

Figure 6.1 RTI-s UDP Multicast Performance

Note that initial runs of thruput exhibited peak send rates on the order of 30K – 50K UAV/s with mean receive throughputs similar to figure 6.1. Under such conditions the benchmarks, specifically the receiving federates exhibited wide swings in receive performance of an order of magnitude or more. This was more pronounced in runs with larger numbers of updates per cycle. Update volumes of 1, 4, 8, 16, and 100K were run. Review of the performance, the RTI-s code itself and QSM papers (as well as correspondence with the QSM designer/implementer) identified that none of the three protocols tested (UDP multicast, RTI-s reliable or QSM) provided a meaningful flow control mechanism. The conclusion was reached that such runs were more a measure of the protocols' (and/or kernel's) handling of receive buffer overflows than the core protocols operation themselves. While runs with various sizes of bursts of updates could result in identification of where the protocols apparently dropped messages given various buffer

sizes, without out effective flow control such results were of limited interest in real work operational simulations.

After the initial runs, the benchmark programs' use of the RTI-s Tick command was adjusted to give more time to RTI-s for internal processing during UAV bursts. This resulted in a reduction of peak send rates to the levels shown in Figure 6.1 while not impacting the mean receive rates. The receive rates did exhibit more steady behavior. In either case, the benchmark programs were much more likely to not run to completion with greater volume of update bursts. Under such conditions, the benchmark programs also exhibited another unexpected and unhelpful behavior: receivers would report a wide range of updates received per cycle, including frequent reception of more updates than were actually sent in the cycle. Reviews of the benchmark and RTI-s code identified shortcomings in the architecture of the benchmarks given the nature of the RTI-s implementation. Specifically, the non-blocking nature of the UPDATE_ATTRIBUTE_VALUE invocation, combined with the ability of RTI-s to queue up events for future processing, overtaxed the benchmarks' simplistic synchronization implementation. Specifically, the sending benchmark would complete all UAV invocations and send an interaction indicating the cycle was complete and requesting receiver federates submit their cycle report. Receiving federates would in some cases timeout on reception and report that not all updates were received. At a future benchmark cycle the receiver would receive updates from the timed out as well as the current cycle and thus report incorrect receive counts.

Based on the fragility of the benchmark architecture, and the previously mentioned desire to not just measure buffer overflow in the absence of flow control, all future runs were done with the revised Tick configuration and limited to 10 cycles of 1000 updates. Unless otherwise stated, the results are show for mean of the sender throughput across all cycles run and the mean of all receivers across all cycles run.

Figure 6.2 demonstrates the baseline performance of the RTI-s reliable and the QSM reliable protocol running the thruput benchmark. QSM exhibits generally better receive performance than RTI-s. Again send performance is markedly better than receive and remains generally flat across a range of nodes. RTI-s was able to synchronize when running on 32 nodes but was unable to complete all cycles. QSM was unable to synchronize and thus unable to begin a 32 node run.

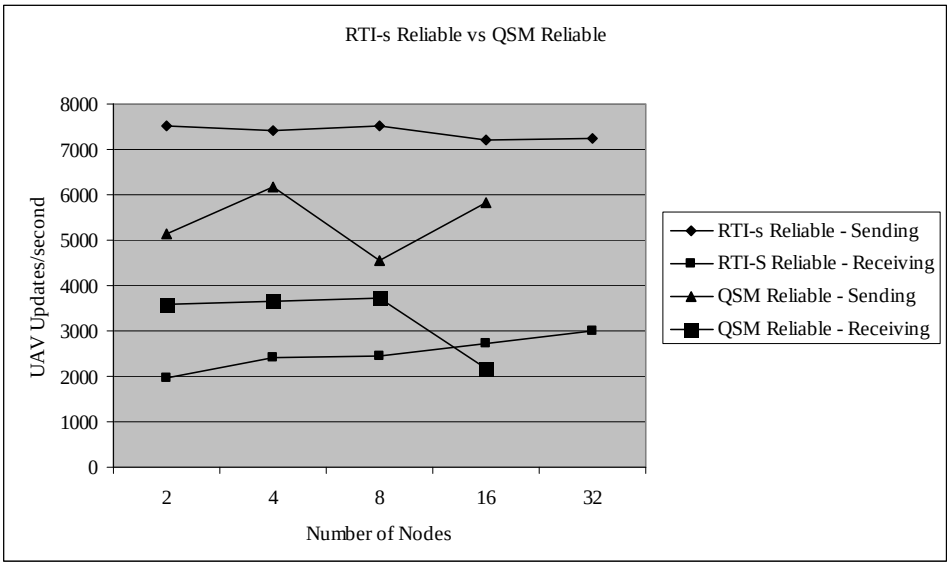

Figure 6.2 RTI-s Reliable protocol performance compared to QSM Reliable performance.

The comparison of the performance of RTI-s UDP and Reliable protocols, as shown in figures 6.1 and 6.2, shows that a simulation does pay a significant performance penalty for using reliable. RTI-s UDP throughput, specifically for receive, is consistently 3x to 4x that of RTI-s' reliable. QSM reliable receive performance penalty is less but it still runs approximately 1/3 that of RTI-s UDP. Note that all these throughput rates represent unloaded or unstressed nodes – runs for which essentially no loss occurs.

While Figure 6.2 demonstrates relatively consistent performance for each protocol across a range of nodes, the detailed results on a cycle by cycle and node by node basis demonstrate a significant difference between RTI-s and QSM in the stability of throughput as measured by the receiver. Figure 6.3 provides that detailed data and demonstrates a characteristic of RTI-s and QSM that was apparent across most of the benchmark runs: RTI-s demonstrated significant throughput instability while QSM stability was stable. Discussions with the RTI-s programmer and review of RTI-s papers underscores the fact that RTI-s was designed with peak performance and high scalability in mind. Future research into the granularity of the RTI-s event scheduling system and its interaction with the Tick command would appear to be an appropriate line of investigation to better understand and perhaps modify this behavior.

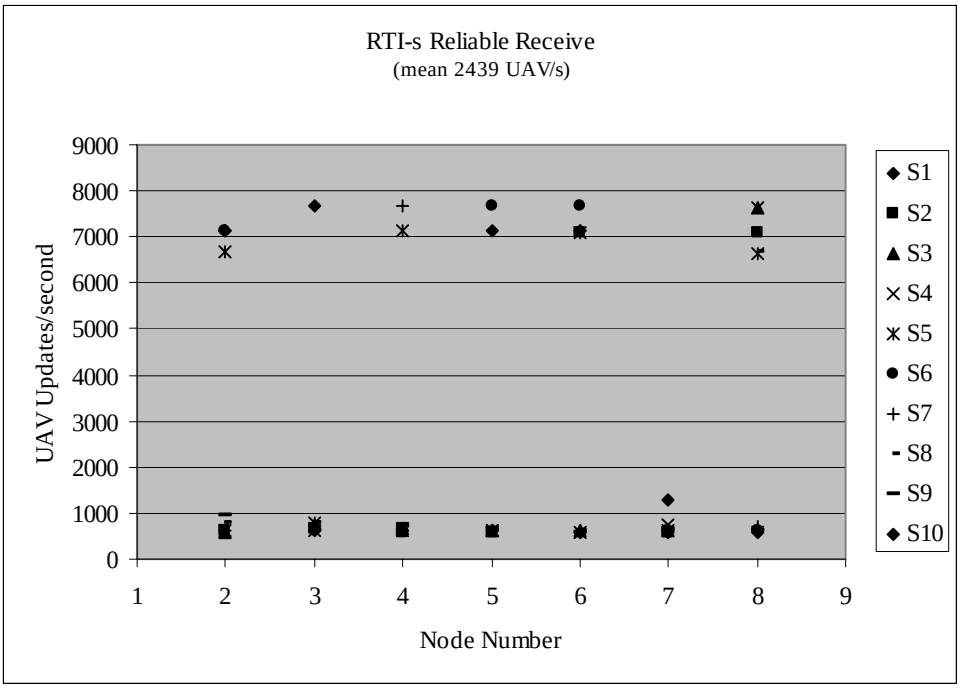

Figure 6.3a RTI-s Reliable receive performance for 7 receive nodes and 10 series of runs.

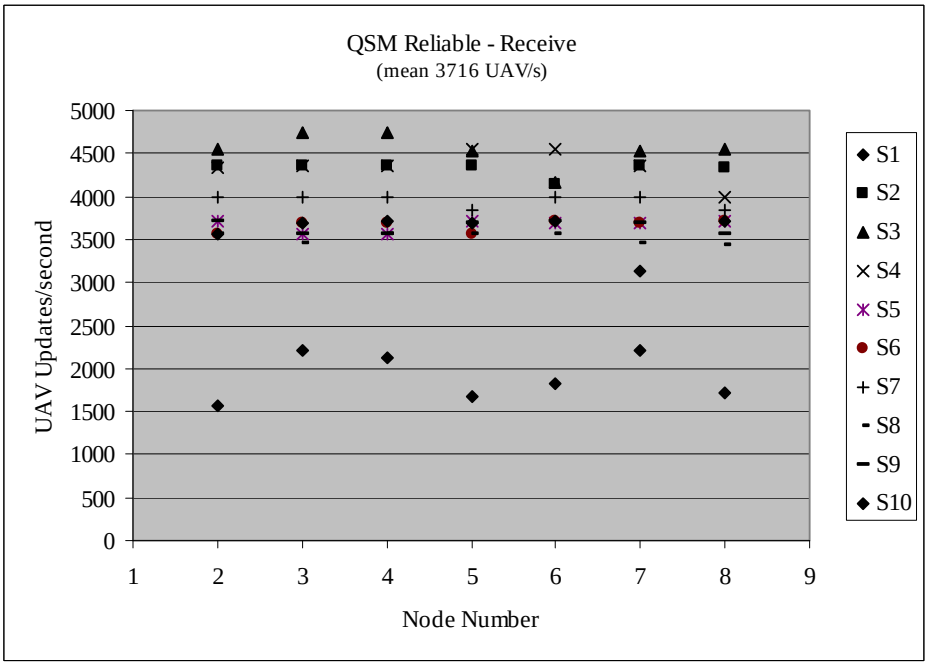

Figure 6.3b QSM Reliable receive performance for 7 receive nodes and 10 series of runs.

Another significant performance difference not illustrated by the mean throughput graphs is the CPU utilization of the nodes while under test. As the Table 6.1 indicates, while the CPU loading for RTI-s reliable vs RTI UDP multicast is higher, the CPU loading for QSM vs RTI-s reliable is significantly higher. All results are approximate as measured by the Windows System Monitor. In addition to significantly lower CPU utilization, the RTI-s CPU utilization tended to peak, on both the sending and receiving side, for few seconds around the sub-second time to send the 1000 updates. The QSM utilization, on the other hand, generally remained at essentially 100% CPU utilization for the duration of a benchmark run. As mentioned above, the design of QSM, as exposed using the unmanaged interface, does not copy a sending buffer upon invocation of the enqueue call. RTI-s, with QSM integrated, added an additional malloc, buffer copy and free, to each update. To isolate the impact of this as compared to RTI-s reliable that does not have to maintain a dedicated send buffers, a “handicapped” version of RTI-s was created that included an added malloc/copy/free cycle. This handicapped version of RTI-s reliable does show a lower NIC utilization (and hence lower fundamental network throughput) and a higher CPU utilization than the non-handicapped version. But the increase in CPU utilization of ~ 3 - 5% is too low to account for QSM’s high CPU numbers.

	Sending CPU	Receiving CPU	Sending NIC	Receiving NIC
QSM	100%	100%	70%	35%
RTI-s Reliable	12-16%	12-16%	55%	45%
RTI-s UDP	4-8%	4-8%	30%	25%
RTI-s Reliable Handicapped	15%	20%	45%	30%

Table 6.1 Node CPU and NIC loading running various protocols**.**

The comparison of the results between Figure 6.3 and Table 6.1 perhaps indicates that, relative to RTI-s reliable, QSM invests more CPU cycles in scheduling and controlling the delivery of messages to the application with a resulting cost to CPU utilization but improved throughput stability. Again, detailed analysis of RTI-s' and QSM's scheduling subsystems would help illuminate the cause of this observed behavior.

As described above, the latency benchmark sends a single UAV per cycle. Thus even when run with multiple cycles, the benchmark does not generate a significant load on the nodes under test. As indicated in Table 6.2, RTI-s reliable displayed latencies approximately twice that of RTI-s UDP multicast. Multiple latency tests with QSM displayed a range of latencies on the order of 100 – 200 ms. The wide range of measured latencies combined with the general lack of sophistication of the latency benchmark program itself calls into question the validity of these QSM latency numbers. For that reason they were not included in the table

below. Note that non-RTI-s QSM latency has been observed in [OBP06] as being on the order of 12 – 25 ms for the range of packet sizes indicated in Table 6.2 Note that the Windows ping command reported node to node latencies of <1 ms for this cluster.

	128 Byte	256 Byte	1024 Byte
RTI-s UDP	5.3 ms	5.5ms	5.3 ms
RTI-s Reliable	10.5 ms	11.0 ms	10.9 ms

6.2 Latency of RTI-s UDP and Reliable protocols

With the goal of trying to induce loads onto the benchmarked nodes such as might be experienced by operational simulations running in a datacenter/cluster environment, three series of experiments were performed. In the first experiment, the sending node was targeted with approximately 15 Mbps of additional network traffic created by the ping command. In the second experiment, the same load was targeted to a receiving node. Figure 6.4 provides the results of these two runs. In both cases, the added network load had essentially no impact on the application throughput as measured by the thruput benchmark program.

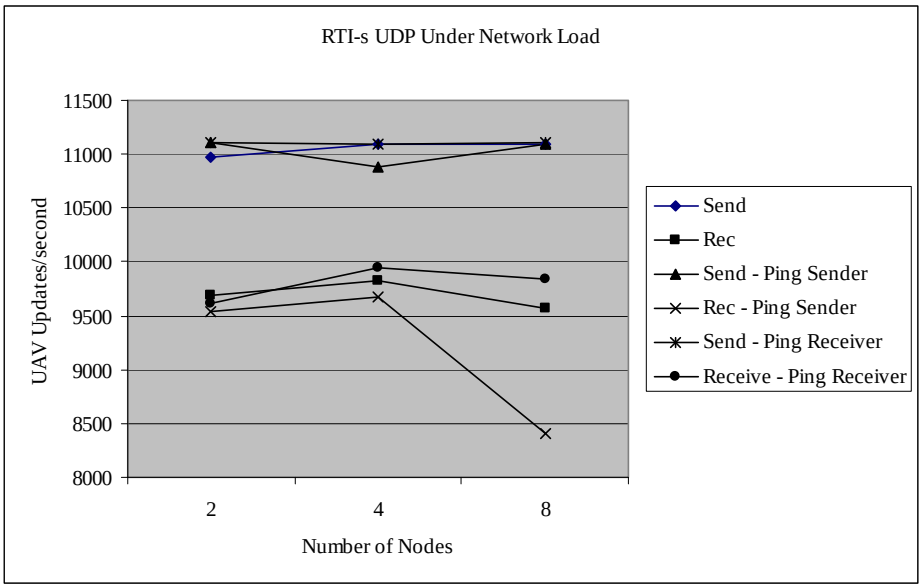

Figure 6.4a RTI-s UDP protocol under network load.

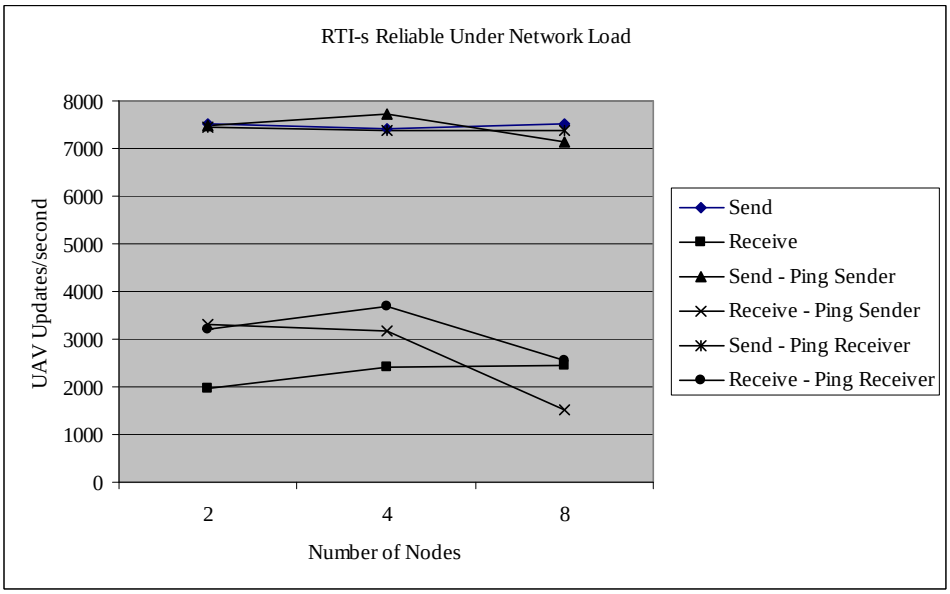

Figure 6.4b RTI-s Reliable protocol under network load.

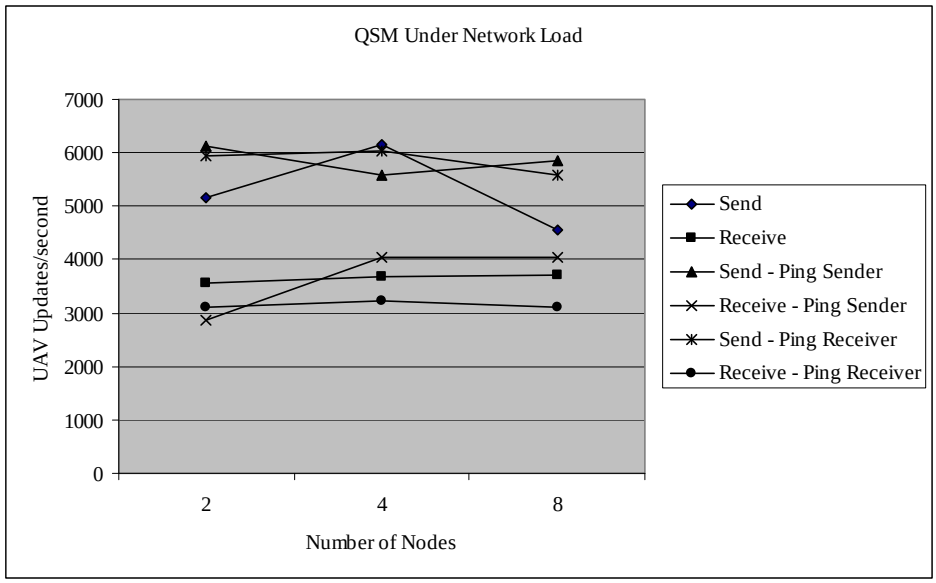

Figure 6.4c QSM protocol under network load.

In the third experiment, one of the receiving nodes was loaded with a high computation load such as might be experienced by a federate with a large number of simulation calculations to perform. The load was provided by the Linpack floating point benchmark which was run continuously on one receiving node throughout the series of tests, generating essentially 100% CPU loading. Note that while this ran as a separate process from the benchmark program both ran at the same operating system priority level as they competed for CPU cycles. CPU monitoring during this series showed a steady 100% CPU load. As indicated by Figure 6.5, this significant CPU load had essentially no impact on the performance of RTI-s UDP multicast or reliable protocols. QSM, on the other hand, was unable to run the benchmark with this load. Given the previously reported high CPU utilizations of QSM running on an unloaded node, it would appear that the conten-

tion for CPU cycles with the addition of the Linpack workload incapacitated QSM.

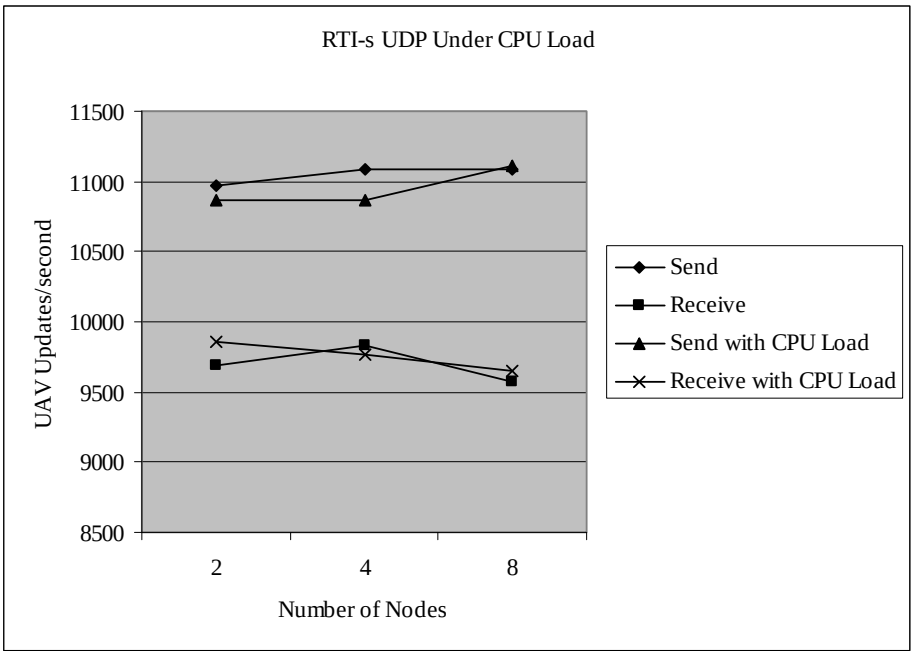

Figure 6.5a RTI-s UDP protocol under receiver-side CPU load.

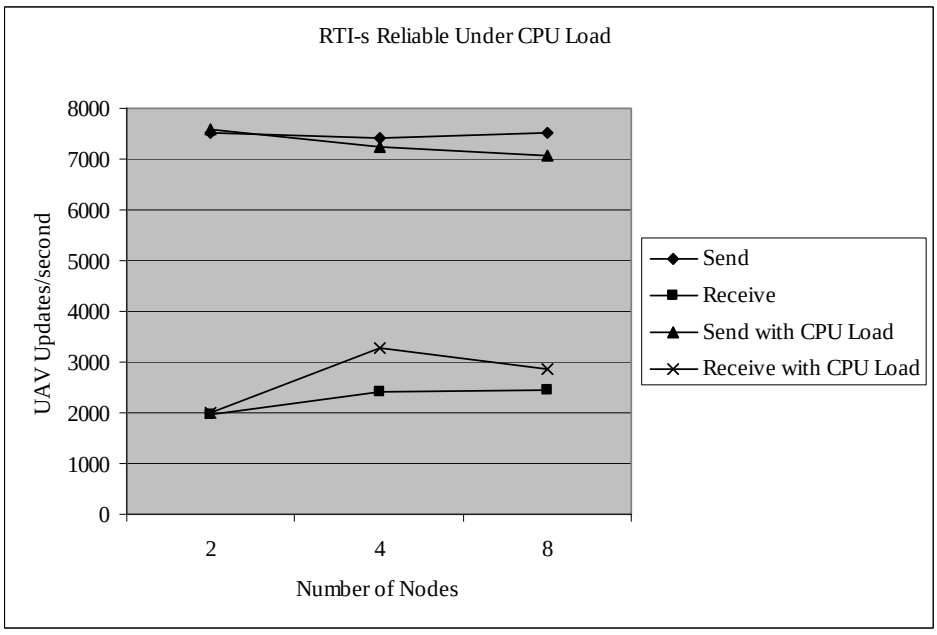

Figure 6.5b RTI-s Reliable protocol under receiver-side CPU load.

As stated above, the design goal of this experiment was to exercise HLA simulations under induced load (i.e, via the benchmarks) and conditions similar to what might be expected to be experienced by actual operational simulations running in modern datacenters. The induced load provided a high CPU loading on a receiving node and bursts of network traffic directed at sender or receiver nodes, did not demonstrate degraded performance. The notable exception to this is QSM's performance under the CPU load which rendered it unable to participate in the benchmark run.

On the one hand, the experiment's inability to cause the reliable protocols to drop packets and engage recover mechanism, with the anticipated result of reduced performance, can be viewed as a shortcoming. On the other hand, to the degree that this represented conditions that might be encountered by actual operational simulations this experiment highlights the fact that perhaps message loss is rarer problem than was anticipated and HLAs fundamental message passing functions are robust in the face of load.

7 RELATED WORK

A number of papers are available that discuss a range of HLA benchmarking and performance measurement. Papers vary on the benchmarks used, the protocols measured and the scale of the benchmarks. Only one paper includes a reliable protocol in the results but it is not evaluated under message loss conditions.

Several papers describing HLA benchmarking results using the DMSO standard benchmarks exist.

[FMF02] provides throughput, latency and HLA time advance results as well as discusses the impact of message bundling. Discussion of an improved benchmark that includes RTI overhead is provided. This work is limited to 16 nodes and a single group.

[JBC+00] provides latency and HLA time advance benchmarks with the scale limited to 5 nodes and 4 multicast groups. These benchmarks use a host-based multicast and Pseudo-Reliable Multicast Protocol (PRMP) NAK-based reliable multicast similar to SRM.

[FH98] provides latency results for three different protocol stack/network hardware configurations: TCP/Ethernet, RTI-Kit/Myrinet and FM/Myrinet. HLA time advance results for three different RTI implementations are provided.

[KLK+02] is perhaps the most extensive. It provides throughput and latency results for three different RTI implementations under a range of conditions. Measurements for Object and Interactions are included. Results from a range of attribute size, attribute count and Interaction parameter size are provided. A good discussion and graphical representation of the sources of latency is provided. Group and node scalability is not discussed – all runs were apparently on a single pair of nodes running a single group. Note that [KLK+02] uses a benchmark that "essentially measures RTI latency and through put in the same manner as the DMSO benchmark."

Several papers use Computer Generated Forces (CGF) systems to generate the simulation workload of much higher simulation entity counts and/or node counts than used by DMSO benchmarks. Note that CGF systems are typically used in large-scale constructive simulation exercises.

[CDW00] uses a scaled ClutterSim, a scaled down version of the Joint Semi-Automated Forces (JSAF) CGF system that only creates background traffic simulations entities (for example, non-combatant ground vehicles). Using 17 nodes, this system measured the startup time for creation of between 30,000 and 40,000 stationary and moving entities.

[GAD05] work extends the RTI-s networking architecture by adding tree and mesh host-based routers that utilize interest management techniques including sender-side squelching to minimize unnecessary traffic. CGF-generated load performance results for a 128 node cluster are provided as well as results for a DMSO-style benchmark.

[DMB05] provides DMSO time advance benchmarks for several WAN latencies of 50, 100 and 200 ms. Interestingly, as the number of federates increased from 2 to 8, the time advance "grants per second" performance for the best case (50 ms) dropped off more rapidly than for the 100 and 200 ms runs to the point where the 50 ms performance is only ~ 2x that of the 200 ms latency run.

Finally, [MKL05] and [WGW06] discuss the performance of HLA simulations in a broader context, not limited to RTI message transport performance.

8 CONCLUSION

The goal of this research was use defacto standard benchmarks to investigate the performance of HLA simulations in their use of best effort (UDP multicast) and reliable protocols. It was hoped that such an investigation, using existing and current-generation reliable protocols would both highlight the tradeoffs of these choices as well as demonstrate significantly better reliable performance by the current-generation protocol – enabling simulation designers to make greater use of reliable protocols.

The research did uncover a number of tradeoffs, some clearly not anticipated. The instability of the RTI-s reliable reception throughput and the high CPU loading observed in the QSM RTI-s benchmarks are specific examples. The instability of the throughput performance of RTI-s was another unexpected result.

On the other hand, the performance observed with the unmanaged version of QSM, while better for the more meaningful receive side, was not dramatically higher. It is not anticipated that these results will launch a rush for HLA simulation designers to increase their use of reliable protocols. If QSM represents one of the highest performing reliable multicast protocols developed in the last 10 years it would appear that a reliable protocol that can lure designers away from best effort IP multicast remains an unsolved problem.

Another unexpected outcome of the research was the observed fragility and immaturity of the defacto standard HLA benchmarks. Given the scarcity of HLA benchmarks, and the limitations of the defacto standards used here, there appears to be room for development of more robust and meaningful HLA benchmarks.

The experience with these less-than-ideal benchmark programs as a result of this research generated numerous ideas for what an “ideal” HLA benchmark system would provide. Given the large amount of resources that organizations such as the US military commit to HLA-based simulation exercises, and the apparent lack of sophisticated HLA benchmarks, such an “ideal” system would appear to be of value.

First, a benchmark system must be more robust and more scalable than the existing programs. While DMSO thruput and latency provide some support for multiple federates (i.e.,nodes), the unsophisticated nature of their interaction-based coordination protocol coupled with the bursty nature of RTI-s traffic caused numerous benchmark hangs. As described above, runs of 16 or 32 nodes often failed to complete and no runs of 64 nodes were successful. An HLA benchmark system should also integrate the throughput and latency measures. The existing latency program measures latency, ping-link, in the absence of other HLA-generated interactions or updates (i.e., RTI “traffic). Throughput and latency are often linked with changes to improve one measure resulting in deterioration of the other. While it would not be practical or meaningful to timestamp and return all updates

sent by a throughput benchmark, a trace amount of time stamped updates, returned by receiving federates, could be injected into the throughput stream. This would enable integrated throughput and latency measures; specifically the measurement of latency under known HLA loading. Ideally, this would be measured on a per-federate basis so, for instance, the latency from a cluster to nodes on two separate WAN-connected LANs could be individually measured.

Coordination of the existing benchmarks on different nodes is problematic. An ideal system would provide a benchmark control console and utilize the existing HLA distributed messaging system to coordinate benchmarks. This would provide benchmark programs the same fine-grained control operational simulation programs enjoy while not requiring a separate out-of-band coordination network. Obviously a tradeoff here is that this coordination traffic would compete with the benchmark itself for resources (network, RTI processing) and would be inoperable if RTI itself crashes. The console would maintain detailed benchmark configuration files to enable automation of complex, regression testing style benchmarks. Results would be collected back to the console and stored in standard format for subsequent analysis. Real time or near-real time graphing of results would allow quick adjustments. Such a system could be used as an integral component in the configuration and testing of complex simulations for operational purposes.

The ideal system would combine both unit tests as well as support diverse and complex benchmarks more representative of actual HLA use. Explicitly support for many-to-many communication patterns and fine-grain control of publication and subscriptions would be required to map real-world complexity.

Such a benchmark system that interfaces at the HLA API standard level would work with any standard HLA RTI implementation. The addition of operating system-specific hooks to catch network traffic at the socket level, would allow the ability to simulate a WAN topologies by delaying packets as well as to simulate packet loss by dropping packets based on user-specified loss distributions.

Using this capability to store and execute complex benchmark configurations would also enable the establishment of standard a benchmark such as the database benchmark promulgated by the Transaction Processing Council. Results from such standards would enable practitioners to better assess anticipated performance before a configuration is established.

This research and its reported results have also prompted considerations for what the design for an “ideal” HLA reliable protocol might be. As mentioned above, HLA has requirements that are challenging in themselves and different than the design goals of many reliable multicast systems. Specifically, HLA has requirements for large numbers of multicast groups, high group join/leave rates, high data rates and the complex many-to-many communication patterns. On the other

hand, the HLA standard has what might be considered lax standards by group communication researchers: limited ordering and group membership requirements. The practice of deploying HLA simulations across both LAN, WAN and hybrid topologies places further challenges on a HLA reliable protocol.

Use of HLA across WAN links gives rise to careful management of the typically limited WAN bandwidth (at least compared to 100 Mbps and Gigabit bandwidths available with commodity LAN-based networking). Thus, caching of messages at the local WAN-connected site would enable local recovery. For example, caching 10 seconds of OC-12 (622 Mbps) WAN traffic represents ~ 600 MB of node memory and only a nominal incremental cost given current hardware costs. A local cache, either centralized or distributed, would thus support local processing of NAKs and ACKS resulting in better utilization of WAN links due to the ability to perform local recovery as well as due to reduced time to repair and thus reduce latency under loss conditions.

An ideal HLA reliable protocol would capture performance metrics, including congestion measures that the RTI could expose to simulation programs via an extension to the current standard. This would enable simulation programs to make informed decisions on attribute updates and minimize the chance that the underlying RTI or its protocols, or the underlying operating system and network components are forced to drop packets as buffers are exhausted or when CPU cycles for processing packets are limited.

Similar to the RID configuration file of the existing RTI-s system, an ideal protocol would be configurable and tunable to enable performance to be matched to the specific simulation. Ideally, the RTI would expose to the simulation program, again through an extension to the standard, an API for dynamic adjustment at runtime. Such an extension could allow the simulation program to communicate its networking values in a high level manner that was not protocol-specific. Thus values such as high throughput, low latency or consistent throughput or latency could be communicated allowing the underlying RTI and protocols to make informed decisions about resource allocation.

9 BIBLIOGRAPHY

[BBP+07] Balakrishnan, Mahesh and Birman, Ken and Phanishayee, Amar and Pleisch, Stefan, "Ricochet: Lateral Error Correction for Time-Critical Multicast", To Appear in Proceedings of the 4th USENIX Symposium on Networked Systems Design & Implementation (NSDI 07). Cambridge, MA. April 2007

[BD01] Boukerche, Azzedine and Dzermajko, Caron, "Performance Comparison of Data Distribution Management Strategies" Proceedings of the Fifth IEEE International Workshop on Distributed Simulation and Real-Time Applications (DS-RD 1001), 2001, Pages 67-75.

[BHO+99] Birman, Kenneth p. and Hayden, Mark and Ozkasap, Oznur and Xio, Zhen, "Bimodal Multicast", ACM Transactions on Computer Systems, Vol 17, No. 2, May 1999, Pages 41-88.

[BJ98] Buss, Arnold and Jackson, Leroy., "Distributed Simulation Modeling: A Comparison of HLA, CORBA and RMI", Proceedings of the 1998 Winter Simulation Conference

[BPB05] Balakrishnan, Mahesh and Pleisch, Stefan and Birman, Ken, "Slingshot: Time-Critical Multicast for Clustered Applications", Fourth IEEE International Symposium on Network Computing and Applications", July 2005. Pages 205-214.

[BV99] Boswell, Steven B. and Van Hook, Daniel J., “Support for Heterogeneous Communication Infrastructures in the HLA RTI”, Simulation Interoperability Workshop, Spring 1999.

[CCM+97] Calvin, James, O. and Chiang, Carol J. and McGarry, Stephen M. and Rak, Steven J. and VanHook, Daniel J., “Design, Implementation, and Performance of the STOW RTI Prototype (RTI-s)”, Simulation Interoperability Workshop, Spring 1997.

[CKV01] Chockler, G. V. and Keidar, I. and Vitenberg, R. “Group Communication Specifications: A Comprehensive Study”. ACM Computer Surveys, 33(4):1-43, Dec. 2001.

[CDW00] Civinskas, Wayne and Dufault, Brett and Wilbert, Deborah, “RTI-NG Performance in Large-Scale Platform-Level Federations”, Simulation Interoperability Workshop, Fall 2000.

[CRS+02] Chu, Yang-Hua and Rao, Sanjay and Srinivasan, Seshan and Zhang, Hui “A Case for End System Multicast”, IEEE Journal on Selected Areas in Communications, Vol 20, Issue 8, Oct 2002, Pages 1456-1471

[DFW97] Dahmann, Judith S. and Fujimoto, Richard M. and Weatherly, Richard M., “The Department of Defense High Level Architecture”, IEEE/ACM Winter Simulation Conference Washington DC, December 13-16th, 1997.

[DGH+87] Demers, Alan and Greene, Dan and Houser, Carl and Irish,

Wes and Larson, John and Shenker, Scott and Sturgis, Howard and Swinchart, Dan and Terry, Doug, “Epidemic Algorithms for Replicated Database Maintenance”, Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing, Vancouver, British Columbia, Canada, 1987, Pages 1-12.

[DKW98] Kahmann, Judith S. and Kuhl, Frederick and Weatherly, Richard, “Standards for Simulation: As Simple As Possible But Not Simpler The High Level Architecture for Simulation”, *Simulation*, Volume 71, Number 6.

[DMB05] Dillman, Bradford and Murphy, Jason and Bleichman, Dan., “Effects of High Latency Wide Area Networks in Distributed Simulation”, Simulation Interoperability Workshop, Fall, 2005

[DMSO04] US Department of Defense, Defense Modeling and Simulation Office website: https://www.dmso.mil/public/

[Dor01] Dorsch, Matthew D., “Achieving Scalability within High Level Architecture Run Time Infrastructures”, Simulation Interoperability Workshop, Fall 2001.

[FH98] Fujimoto, Richard and Hoare, Peter, “HLA RTI Performance in High Speed LAN Environments”, Simulation Interoperability Workshop, Fall 1998.

[FJL+97] Floyd, Sally and Jacobson, Van and Liu, Ching-Gung and

McCanne, Steven and Zhang, Lixia. “A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing”, IEEE\ACM Transactions on Networking, Vol 5, Num 6, Pgs 784-803, 1997

[FMF02] Fitzgibbons, Brad and McLean, Thom and Fujimoto, Richard, “RTI Benchmark Studies”, Simulation Interoperability Workshop, Spring 2002.

[Fuj03] Fujimoto, Richard M., “Distributed Simulation Systems”, IEEE/ACM Winter Simulation Conference New Orleans, LA, December 7-10, 2003

[GAD05] Gottschalk, Thomas D. and Amburn, Philip and Davis, Dan M., “Advanced Message Routing for Scalable Distributed Simulations”, The Society for Modeling and Simulation International, JDMS, Vol 2, Issue 1, January 2005, Pages 17-28.

[GMS+03] Gemmell, Jim and Montgomery, Todd and Speakman, Tony and Bhaskar, Nidhi and Crowcroft, Jon, “The PGM Reliable Multicast Protocol”, IEEE Network, January/February 2003.

[HSC95] Holbrook, Hugh W. and Singhal, Sandeep K. and Cheriton, David R., “Log-based Reliable-Receiver Multicast for Distributed Interactive Simulation”, ACM SIGCOMM Computer Communications Review, Volume 25, Issue 4, Pages 328-341, 1995.

[HTW03] Helfinstine, Bill and Torpey, Mark and Wagenbreth, Gene, "Experimental Interest Management Architecture for DCEE", Interservice/Industry Training, Simulation and Education Conference (I/ITSEC), 2003.

[HWT+01] Helfinstine, Bill and Wilbert, Debora and Torpey, Mark and Civinskas, Wayne, "Experiences with Data Distribution Management in Large-Scale Federations", Simulation Interoperability Workshop, Fall 2001.

[IEEE00] IEEE Std 1516.1-2000, "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-Federate Interface Specification.

[JBC+00] Ji-Beng, Alex Koh and Bu-Sung, Francis Lee and Cai, Wentong and Turner, Stephen J., "Multicasting Fast Messages in RTI-Kit", Simulation Interoperability Workshop, Spring 2000.

[JGJ+00] Jannotti, John and Gifford, David K. and Johnson, Kirk L. and Kaashoek, M. Frans and O'Toole, James W. Jr, "Overcast: Reliable Multicasting with an Overlay Network", USENIX/ACM/IEEE Fourth Symposium on Operating Systems Design and Implementation (OSDI), October 2000, San Diego, CA, USA.

[KBD07] Ostrowski, Krzysztof and Birman, Ken and Dolev, Danny, "Live Distributed Objects", IEEE Internet Computing, No-

vember 2007, Pages 72 – 78.

[KLK+02] Knight, Pamela and Liedel, Ron and Kilnner, Melanie and Steele, Jacqueline and Drake, Ray and Agarwal., Paul and Nunez, Edwin and Espinosa, Mario and Giddens, Jessica and Jenkins, Carol, “Analysis of Independent Throughput and Latency Benchmarks for Multiple RTI Implementations”, Simulation Interoperability Workshop, Fall 2002.

[KWD99] Kuhl, Frederick and Weatherly, Richard and Dahmann, Judith, “Creating Computer Simulation Systems: An Introduction to the High Level Architecture”, ISBN 0130225118, 1999, Prentice Hall

[MBD00] Morse, Katherine L. and Bic, Lubomir and Dillencourt, Michael and Tsai, Kevin., “Multicast Grouping for Dynamic Data Distribution Management”, Simulation Interoperability Workshop, Fall 2000.

[McG98] McGarry, Stephen., “An Analysis of RTI-S Performance in the STOW 97 ACTD”, Simulation Interoperability Workshop, Spring 1998.

[MKL05] Moller, Bjorn and Karlsson, Mikael and Lofstrand, Bjorn., “Common Federation Performance Bottlenecks”, Simulation Interoperability Workshop, Fall 2005

[OB06] Ostrowski, Krzysztof and Birman, Ken. “Scalable Publish-

Subscribe in a Managed Framework." Cornell University Technical Report, TR2007-2086. November, 2006.

[OB07] Ostrowski, Krzysztof and Birman, Ken. "Implementing High Performance Multicast in a Managed Environment." Cornell University Technical Report, TR2007-2087. March, 2007.

[OBP05] Ostrowski, Krzysztof and Birman, Ken and Phanishayee, Amar. "The Power of Indirection: Achieving Multicast Scalability by Mapping Groups to Regional Underlays." Cornell University Technical Report, TR2006-2064. November, 2005.

[OBP06] Ostrowski, Krzysztof and Birman, Ken and Phanishayee, Amar. "QuickSilver Scalable Multicast." Cornell University Technical Report, TR2006-2063. April 2006.

[OMG03] Object Management Group, "OMG Document" (i.e., Data Distribution Service Standard), http://www.omg.org/cgi-bin/doc?ptc/2003-07-07, July 7th, 2003.

[Pul99] Pullen, J. Mark., "Reliable Multicast Network Transport for Distributed Virtual Simulations", Proceedings of the 1999 IEEE Distributed Interactive Simulation and Real-Time Systems Workshop, College Park Maryland, March 23-24th, 1999.

[Pul99a] Pullen, M., "Limitations of Internet Protocol Suite for Distributed Simulation in the Large Multicast Environment", RFC 2502, February 1999,

http://www.ietf.org/rfc/rfc2502.txt?number=2502.

[RSM97] Rak, Steven, J. and Salisbury, Marnie and MacDonald, Robert S., “HLA/RTI Data Distribution Management in the Synthetic Theater of War”, Simulation Interoperability Workshop, Fall 1997.

[RV96] Rak, Steven J. and Van Hook, Daniel J., “Evaluation of Grid-Based Relevance Filtering for Multicast Group Assignment”, Simulation Interoperability Workshop, Spring 1996.

[vBM96] van Renesse, Robbert and Birman, Ken P. and Maffeis, Silvano. “Horus: A Flexible Group Communication System”, Communications of the ACM, 39(4):76--83, April 1996

[VC98] Van Hook, Daniel J. and Calvin, James, O. , “Data Distribution Management in RTI 1.3”, Simulation Interoperability Workshop, Spring 1998.

[VCS95] Van Hook, Daniel J. and Calvin, James, O. and Smith, J., “Data Consistency Mechanisms to Support Distributed Simulations”, Simulation Interoperability Workshop, Spring 1995.

[VM96] Van Hook, Daniel J. and McGarry, Stehpen M. , “A Prototype Approach to the Data Communications Component of the RTI”, 15th DIS workshop (September 1996). Precursor conference to SIW.

[VRC96] Van Hook, Daniel J. and Rak, Steven J. and Calvin, James,

O., “Approaches to RTI Implementation of HLA Data Distribution Management Services”, 15th DIS workshop (September 1996). Precursor conference to SIW.

[WBV+97] Wolfson, Harry and Boswell, Steven B. and Van Hook, Daniel J. and McGarry, Stephen M., “Reliable Multicast in the STOW RTI Prototype”, Simulation Interoperability Workshop, Spring 1997.

[WGW06] Watrous, Ben and Granowetter, Len and Wood, Douglas, “HLA Federation Performance: What Really Matters?”, Simulation Interoperability Workshop, Fall 2006.

[WYD05] Wagenbreth, Gene, and Yao, Ke-Thia and Davis, Dan M. and Lucas, Robert F and Gottschalk, Thomas D., “Enabling 1,000,000-enity Simulations on Distributed Linux Clusters”, Proceedings of the 2005 Winter Simulation Conference.

Note that the majority of references to HLA simulation are from the Simulation Interoperability Workshop (SIW) series organized by the Simulation Interoperability Standards Organization (SISO). SIW conference papers are located at http://www.sisostds.org/index.cfm.