Evaluating Transport Protocols for Real-time Event Stream Processing Middleware and Applications

ABSTRACT

Real-time event stream processing (RT-ESP) applications must synchronize continuous data streams, e.g., infrared scans and video monitoring for survivor detection after a natural disaster, despite fluctuations in resource availability. Satisfying these needs of RT-ESP applications requires predictable QoS from the underlying publish/subscribe (pub/sub) middleware. If a transport protocol is not capable of meeting the QoS requirements within a dynamic environment, the middleware must be able to adapt the transport protocol at runtime to react to changing operating conditions.

Realizing such adaptive RT-ESP pub/sub middleware requires a thorough understanding of how different transport protocols behave under different operating conditions. This paper makes three contributions to work on achieving that understanding. First, we define a composite evaluation metric that combines packet latency and reliability to evaluate transport protocol performance. Second, we use this metric to quantify the performance of various transport protocols integrated with the OMG's Data Distribution Service (DDS) QoS-enabled pub/sub middleware standard. Third, we use our metric to pinpoint configurations involving sending rate, network loss, and number of receivers that exhibit the pros and cons of the protocols.

Our results show that a NAK-based reliable multicast protocol and a lateral error correction protocol provide the best performance. Moreover, the reliable multicast protocol provides lower average latency with small number of receivers and small network loss, whereas the lateral error correction protocol provides greater consistency in overall performance and the flexibility to trade-off lower average latency with network bandwidth usage.

Categories and Subject Descriptors

D.2.8 [**Software**]: SOFTWARE ENGINEERING—*Metrics, Performance measures*

DEBS 2009, July 6-9, 2009, Nashville, TN, USA

Keywords

Pub/sub Middleware, Event-based Distributed Systems, Real-time Event Based Streaming, Transport Protocol Evaluation, Data Distribution Service

1. INTRODUCTION

Emerging trends and challenges. *Real-time Event Stream Processing* (RT-ESP) applications support mission-critical systems (such as collaboration of weather monitoring radars to predict life-threatening weather [4]) by managing and coordinating multiple streams of event data that have (possibly distinct) timeliness requirements. Streams of event data may originate from sensors (*e.g.*, surveillance cameras, temperature probes), as well as other types of monitors (*e.g.*, online stock trade feeds). These continuously generated data streams differ from streaming the contents of a data file (such as a fixed-size movie) since the end of RT-ESP data is not known *a priori*. In general, streamed file data demand less stringent delivery and deadline requirements, instead emphasizing a continuous flow of data to an application.

RT-ESP applications require (1) *timeliness* of the event stream data and (2) *reliability* so that sufficient data are received to make the result usable. Moreover, RT-ESP applications encompass multiple senders and receivers, *e.g.*, multiple continuous data streams can be produced and multiple receivers can consume the data streams. With the growing complexity of RT-ESP application requirements (*e.g.*, large number of senders/receivers, variety of event types, event filtering, QoS, and platform heterogeneity), developers are increasingly leveraging pub/sub middleware to help manage the complexity and increase productivity [14, 7].

To address the complex requirements of RT-ESP applications, the underlying pub/sub middleware must support a flexible communication infrastructure. This flexibility requirement is manifest in several ways, including the following:

- Large-scale RT-ESP applications require flexible communication infrastructure due to the complexity inherent in the scale involved. As the number and type of event data streams continues to increase the communication infrastructure must be able coordinate these streams so that publishers and subscribers are connected appropriately. Flexible communication infrastructure must adapt to (1) fluctuating demands for various event streams and (2) environment changes to maintain acceptable levels of service.
- Certain types of large-scale RT-ESP applications aggravate the demands of flexible communication infrastruc-

ture due to their dynamic and *ad hoc* nature. The environments for these applications cause fluctuations in available resources as they include mobile assets with intermittent connectivity and underprovisioned or temporary assets from emergency responders. Examples of *ad hoc* large-scale RT-ESP applications include tactical information grids, *in situ* weather monitoring for impending hurricanes, and emergency response networks in the aftermath of a regional or national disaster.

Achieving a flexible communication infrastructure requires an understanding of the capabilities that the underlying transport protocols provide. Building on this understanding, pub/sub middleware can help ameliorate the complexity of managing multiple event streams. QoS-enabled pub/sub middleware can then maintain real-time QoS for multiple event streams in highly dynamic environments.

Several pub/sub middleware platforms have been developed to support large-scale data-centric distributed systems, such as the Java Message Service (`java.sun.com/products/jms`), Web Services Brokered Notification (`docs.oasis-open.org/wsn/wsn-ws_brokered_notification-1.3-spec-os.pdf`), and the CORBA Event Service (`www.omg.org/technology/documents/formal/event_service.htm`). These platforms, however, do not support fine-grained and robust QoS. Some large-scale distributed system platforms, such as the Global Information Grid (`www.nsa.gov/ia/programs/global_industry_grid`) and Network-centric Enterprise Services (`www.disa.mil/nces`), require rapid response, reliability, bandwidth guarantees, scalability, and fault-tolerance. Moreover, these systems are required to perform under stressful conditions and over connections with less than ideal behavior, such as latency and bandwidth variability, bursty loss, and route flaps.

Solution approach → ADAptive Middleware And Network Transports (ADAMANT). ADAMANT is a QoS-enabled pub/sub middleware platform that provides reliable and timely delivery of data for RT-ESP applications by integrating and enhancing the following technologies:

- The *Adaptive Network Transports* (ANT) framework, which provides an infrastructure for composing transport protocols that builds upon the properties provided by the scalable reliable multicast-based Ricochet transport protocol [20]. Ricochet provides a trade-off between latency and reliability which are desirable qualities for middleware supporting RT-ESP applications. Ricochet also supports modification of parameters to affect latency, reliability, and bandwidth usage.
- OpenDDS (`www.opendds.org`), which is an open-source implementation of the OMG Data Distribution Service (DDS) standard (`www.omg.org/spec/DDS`) that enables applications to communicate by publishing information they have and subscribing to information they need in a timely manner. OpenDDS provides support for various transport protocols, including TCP, UDP, IP multicast, and a reliable multicast protocol. OpenDDS also provides a pluggable transport framework that allows integration of custom transport protocols within OpenDDS.

This paper describes how ADAMANT's integration of—and enhancements to—ANT and OpenDDS provide fine-grained QoS control and standardized QoS-enabled pub/sub middleware atop a multicast protocol framework that enables runtime modification of transport protocols.

To evaluate the impact of various transport protocols on the ADAMANT QoS-enabled pub/sub middleware we developed the *ReLate* composite metrics to evaluate the reliability and latency of received data for various experimental configurations involving parameters such as sending rate, network loss, and number of receivers. This paper applies the ReLate metrics across various ADAMANT transport protocols. It then empirically quantifies the results and analyze the pros and cons of various transport protocol configurations in the context of ADAMANT.

Paper organization. The remainder of this paper is organized as follows: Section 2 describes a representative RT-ESP application to motivate the challenges that ADAMANT is designed to address; Section 3.1 examines the structure and functionality of ADAMANT and the ReLate metrics we created to evaluate ADAMANT and its adaptive transport protocol framework; Section 4 analyzes the results of experiments conducted by applying the ReLate metrics to ADAMANT; Section 5 compares ADAMANT with related work; and Section 6 presents concluding remarks.

2. MOTIVATING THE NEED FOR ADAMANT

This section describes a representative RT-ESP application to motivate the challenges that ADAMANT is designed to address.

2.1 Search and Rescue (SAR) Operations for Disaster Recovery

To highlight the challenges of providing timely and reliable event stream processing for RT-ESP applications, we focused our ADAMANT work on supporting search and rescue (SAR) operations. These operations help locate and extract survivors in a large metropolitan area after a regional catastrophe, such as a hurricane, earthquake, or tornado. SAR operations use unmanned aerial vehicles (UAVs), existing operational monitoring infrastructure (*e.g.*, building or traffic light mounted cameras intended for security or traffic monitoring), and (temporary) datacenters to receive, process, and transmit event stream data from various sensors and monitors to emergency vehicles that can be dispatched to areas where survivors are identified.

Figure 1 shows an example SAR scenario where infrared scans along with GPS coordinates are provided by UAVs and video feeds are provided by existing infrastructure cameras. These infrared scans and video feeds are then sent

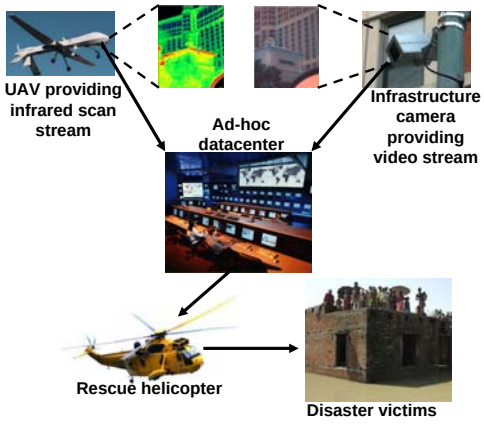

Figure 1: Search and Rescue Motivating Example

to a datacenter, where they are processed by fusion appli-

cations to detect survivors. Once a survivor is detected the application will develop a three dimensional view and highly accurate position information so that rescue operations can commence.

A key requirement of the data fusion applications within the datacenter is tight timing bounds on correlated event streams such as the infrared scans coming from UAVs and video coming from cameras mounted atop traffic lights. The event streams need to match up closely so the survivor detection application can produce accurate results. If an infrared data stream is out of sync with a video data stream the survivor detection application can generate a false negative and fail to initiate needed rescue operations. Likewise, without timely data coordination the survivor detection software can generate a false positive tying up scarce resources such as rescue workers, rescue vehicles, and data center coordinators in an unwarranted effort.

2.2 Key Challenges in Supporting Search and Rescue Operations

Meeting the requirements of SAR operations outlined in Section 2.1 is hard due to the inherent complexity of synchronizing multiple event data streams. These requirements are exacerbated since SAR operations often run in tumultuous environments where resource availability can change abruptly. These changes can restrict the availability of resources (*e.g.*, data stream dropouts and subnetwork failure due to ongoing environment upheaval) as well as increase them (*e.g.*, network resources being added due to the stabilization of the regional situation). The remainder of this section describes four challenges that ADAMANT must address to support the communication requirements of the SAR operations presented above.

2.2.1 Challenge 1: Maintaining Data Timeliness and Reliability

SAR operations must receive sufficient data reliability and timeliness so that multiple data streams can be fused appropriately. For example, the SAR operation example described above highlights the exploitation of data streams (such as infrared scan and video streams) by several applications simultaneously in a datacenter. Figure 2 shows how fire detection applications and power grid assessment applications can use infrared scans to detect fires and working HVAC systems respectively. Likewise, Figure 3 shows how security monitoring and structural damage applications can use video stream data to detect looting and unsafe buildings respectively.

Sections 3.2.1 and 3.2.2 describe how ADAMANT addresses this challenge by incorporating transport protocols that balance reliability and low latency.

2.2.2 Challenge 2: Managing Subscription of Event Data Streams Dynamically

SAR operations must seamlessly incorporate and remove particular event data streams dynamically as needed. Ideally, an application for SAR operations should be shielded from the details of when other applications begin to use common event data streams. Moreover, applications should be able to switch to higher fidelity streams as they become available. Section 3.1 describes how we address this challenge by using anonymous QoS-enabled pub/sub middleware that seamlessly manages subscription and publication of data streams as needed.

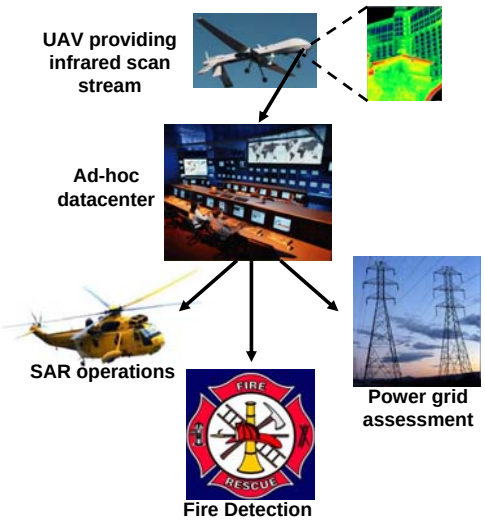

Figure 2: Uses of Infrared Scans during Disaster Recovery

2.2.3 Challenge 3: Providing Predictable Performance in Dynamic Environment Configurations

In scenarios where there is much variability and instability in the environment, such as with regional disasters, the performance of SAR operations must be known *a priori*. SAR operations tested only under a single environment configuration might not perform as needed if at all when introduced to a new environment. The operations could unexpectedly shut down at a time when they are needed most due to changes in the environment. Section 4.2 describes how we determine application performance behavior for dynamic environments.

2.2.4 Challenge 4: Adapting to Dynamic Environments

SAR operations not only must understand their behavior in various environment configurations, they must also adjust as the environment changes. If SAR operations cannot adjust then they will fail to perform adequately given a shift in resources. If resources are lost or withdrawn, the SAR operations must be configured to accommodate fewer resources while maintaining a minimum level of service. If resources are added, the operations should take advantage of these to provide higher fidelity or more expansive coverage. Section 3.2.2 describes how we are incorporating adaptable transport protocols that can be adjusted for reliability, latency, and/or network bandwidth usage.

3. THE STRUCTURE AND FUNCTIONALITY OF ADAMANT AND RELATE

This section presents an overview of ADAMANT, including the OpenDDS and ANT transport protocols used. We then describe the ReLate metrics created to evaluate the performance of ADAMANT in various environment configurations to support RT-ESP application requirements for data reliability and timeliness.

3.1 Overview of ADAMANT

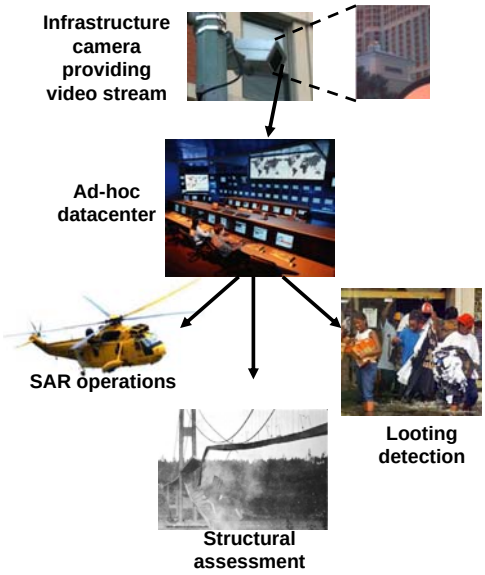

Figure 3: Uses of Video Stream during Disaster Recovery

We have developed ADAMANT to integrate and enhance QoS-enabled pub/sub middleware with adaptive transport protocols to support RT-ESP applications. ADAMANT helps resolve Challenge 2 in Section 2.2.2 by providing anonymous publication and subscription via the OMG Data Distribution Service. (Eee Sidebar 1 for a brief summary of DDS.) ADAMANT is based on the OpenDDS implementation of DDS and incorporates several standard and custom transport protocols.

We chose OpenDDS as ADAMANT's DDS implementation due to its (1) source code being freely available, which facilities modification and experimentation, and (2) support for a *pluggable transport framework* that allows RT-ESP application developers to create custom transport protocols for sending/receiving data. OpenDDS's pluggable transport framework uses patterns, *e.g.*, Strategy [6] and Component Configurator [17], to provide flexibility and delegate responsibility to the protocol only when applicable.

For example, to plug in a custom protocol developers subclass from the *TransportSendStrategy* class to determine how the protocol should send data when a data writer writes out topic data. Likewise, developers subclass from the *TransportReceiveStrategy* class to determine how data should be handled once it is received. The Component Configurator pattern allows application developers to specify which transport protocols should be included into the application at runtime. These protocols can be included either statically when the application is built or dynamically when the application is loaded or while it is running.

Transport protocols are associated with publishers and subscribers since these are the DDS entities that handle sending and receiving data. The OpenDDS pluggable transport framework requires application developers to manage the coordination of a data reader or data writer with the publisher or subscriber, respectively, that provides the desired transport properties. For example, if the application requires that a data writer use reliable communication, application developers must manually associate the data writer with the publisher using a reliable transport. Thus, the transport protocol used determines the reliability of the data transmission.

Sidebar 1: Overview of DDS

The OMG Data Distribution Service (DDS) is standards-based anonymous QoS-enabled pub/sub middleware for exchanging data in event-based distributed systems. It provides a global data store in which publishers and subscribers write and read data, respectively. DDS provides flexibility and modular structure by decoupling: (1) *location*, via anonymous publish/subscribe, (2) *redundancy*, by allowing any numbers of readers and writers, (3) *time*, by providing asynchronous, time-independent data distribution, and (4) *platform*, by supporting a platform-independent model that can be mapped to different platform-specific models.

The DDS architecture consists of two layers: (1) the *data-centric pub/sub* (DCPS) layer that provides APIs to exchange topic data based on specified QoS policies and (2) the *data local reconstruction layer* (DLRL) that makes topic data appear local. The focus of this paper is on DCPS since it is more broadly supported than the DLRL.

The DCPS entities in DDS include *Topics*, which describe the type of data to be written or read; *Data Readers*, which subscribe to the values or instances of particular topics; and *Data Writers*, which publish values or instances for particular topics. Various properties of these entities can be configured using combinations of the 22 QoS policies. Moreover, *Publishers* manage groups of data writers and *Subscribers* manage groups of data readers.

3.2 Overview of Transport Protocols Used in ADAMANT

OpenDDS currently provides several transport protocols. Other protocols for the ADAMANT prototype are custom protocols that we integrated with OpenDDS using the Pluggable Transport Framework. We describe these transport protocols below.

3.2.1 OpenDDS Transport Protocols

OpenDDS provides four transport protocols that can be used with its transport protocol framework, including the Transmission Control Protocol (TCP), the User Datagram Protocol (UDP), and the Internet Procotol (IP) multicast (IP Mcast), and a NAK-based reliable multicast (RMcast) protocol, as shown in Figure 4. OpenDDS TCP is a unicast protocol that provides several properties for data transmission including (1) flow control, where TCP will reduce the number of data packets sent if positive acknowledgments are not received in a timely manner, and (2) reliability, where TCP will resend data until a packet is acknowledged by the receiver. Conversely, UDP is a unicast protocol that makes no attempt to provide flow control or reliability, *i.e.*, after a data packet is sent out on the network UDP's responsibilities end regarding QoS. IP Mcast provides the same properties as UDP except that IP Mcast is used to send data to multiple receivers.

While TCP, UDP, and IP Mcast are standard protocols, RMcast warrants more description. RMcast is essentially a negative acknowledgment (NAK) based protocol. Figure 5 shows how a NAK-based protocol provides reliability. In this example, the sender sends four data packets, but the

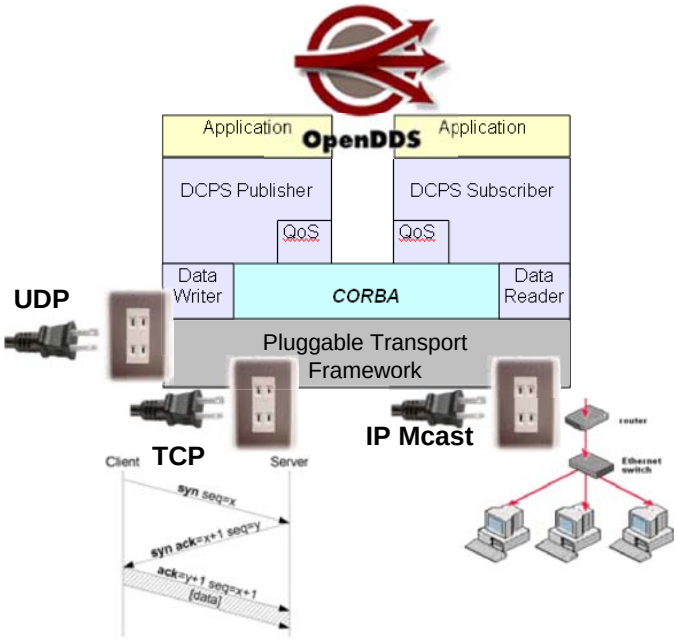

Figure 4: OpenDDS and its Transport Protocol Framework

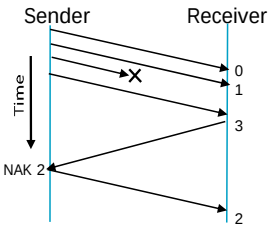

Figure 5: A NAK Based Protocol Discovering Loss

third data packet (*i.e.*, packet #2) is not received by the receiver. The receiver realizes that the third data packet has not been received when the fourth data packet is received. At this point the receiver sends a NAK to the sender and the sender retransmits the missing data packet. The receiver sends a unicast message to the sender for loss notification and the sender retransmits the missing data packet to the receiver.

In addition to providing reliability, the RMcast protocol orders data packets. When the protocol for a receiver detects a packet out of order it waits for the missing packet before passing the data up to the middleware. The receiver must buffer any packets that have been received but have not yet been sent to the middleware. RMcast helps resolve Challenge 1 in Section 2.2.1 by providing reliability and timeliness for certain environment configurations.

3.2.2 Adaptive Network Transport Protocols

The ANT framework originally was developed from the Ricochet [20] transport protocol. Ricochet uses a bi-modal multicast protocol and a novel type of forward error correction (FEC) called lateral error correction (LEC) to provide QoS and scalability guarantees. Ricochet supports (1) time-critical multicast for high data rates with strong probabilistic delivery guarantees and (2) low-latency error detection along with low-latency error recovery.

ANT is a transport protocol framework developed to support various transport protocol properties. These properties include NAK-based reliability, ACK-based reliability, several FEC codes (*e.g.*, XOR, Reed-Solomon, Tornado), and specification of FEC at the sender, the receiver, or within a multicast group. These properties can be composed dynamically at run-time to achieve greater flexibility and support autonomic adaptation.

We included ANT's Ricochet transport protocol (ANT R) and the Baseline transport protocol (ANT B) in ADAMANT. The ANT Baseline protocol mirrors the functionality of IP Mcast as described in Section 3.2.1. Using ANT's Baseline protocol helps quantify the overhead imposed by the ANT framework since similar functionality can be achieved using the OpenDDS IP Mcast pluggable transport protocol.

Forward Error Correction (FEC). Ricochet is based on the concepts of FEC protocols. FEC protocols are designed with reliability in mind. They anticipate data loss and proactively send redundant information to recover from this loss. There are two subcategories of FEC protocols: (1) sender-based and (2) receiver-based.

Sender-based FEC protocols such as Packet Level FEC [22] and Parity-based Loss Recovery [18], have the sender send redundant corrective data along with the normal data, as shown in Figure 6. For every r data packets sent c repair packets are also sent. The r and c parameters together constitute the rate of fire. The rate of fire is tunable, giving systems flexibility in the amount of reliability delivered versus the amount of network bandwidth used.

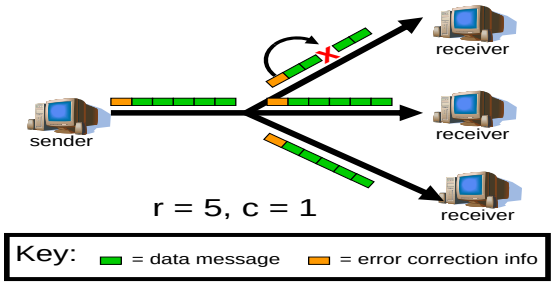

Figure 6: FEC Reliable Multicast Protocol - Sender-based

FEC reliable multicast protocols also can be receiver-based. This type of protocol is also known as Lateral Error Correction (LEC). For these protocols, such as the Ricochet protocol we employ in ADAMANT, the data receivers send each other redundant error correction information, as shown in Figure 7.

Lateral Error Correction (LEC). LEC protocols have the same tunable r and c rate of fire parameters as sender-based FEC protocols. Unlike sender-based FEC protocols, however, the recovery latency depends on the transmission rate of receivers. As with gossip-based protocols, LEC protocols have receivers send out to a subset of total number of receivers to manage scalability and network bandwidth. Moreover, the r and c parameters have slightly different semantics for LEC protocols than for sender-based FEC protocols.

The r parameter determines the number of packets a *re-*

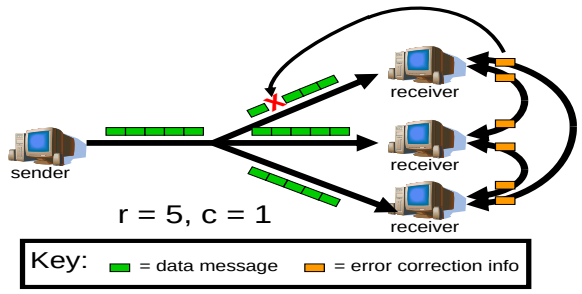

Figure 7: FEC Reliable Multicast Protocol - Receiver-based (LEC)

ceiver, rather than the sender, should receive before it sends out a repair packet to other receivers. The c parameter determines the number of receivers that will be sent a repair packet from any single receiver. As described in Section 4.2, we hold the value of c constant (*i.e.*, the default value of 3) while modifying the r parameter.

The Ricochet protocol helps resolve Challenge 1 in Section 2.2.1 by providing high probabilistic reliability and low latency error detection and recovery. Ricochet also helps resolve Challenge 4 in Section 2.2.4 by supporting tunable parameters that effect reliability, latency, and bandwidth usage. The ANT framework is architected so that different transport protocols can be switched dynamically as needed. Table 1 presents a summary of all the protocols we included in our experiments.

Protocol	Integrator	Functionality
TCP	OpenDDS	unicast, reliable, packet ordering, flow control
UDP	OpenDDS	unicast, unreliable
IP Mcast	OpenDDS	multicast, unreliable
RMcast	OpenDDS	multicast, reliable, packet ordering
ANT Baseline	ANT	multicast, unreliable
ANT Ricochet	ANT	multicast, probabilistically reliable

Table 1: Transport Protocols Evaluated

3.2.3 *Evaluation Metrics for Reliability and Latency (ReLate)*

We now describe considerations for evaluating latency and reliability for ADAMANT. We present guidelines for unacceptable percentages of packet loss for multimedia applications. We also introduce the *ReLate* metrics used to evaluate ADAMANT empirically in Section 4.

A naive attempt to evaluate the effect of transport protocols with respect to both overall latency and reliability would be simply to compare the latency times of protocols that provide reliability. Since some reliability would be provided these protocols would presumably be preferred over protocols that provide no reliability. The reliability provided by the reliable protocols in our experiments, however, deliver different percentages of reliability. The level of reliability must be quantified to be useful in comparing results.

We initially designed the *ReLate* metric to account for both latency and reliability in a fairly straightforward manner. ReLate divided the average latency of data updates for an experiment using a particular protocol by the percentage of updates received. This metric then accounted for reliability and latency. In particular, if latencies were equal between two protocols then the protocol that delivered the most updates would have the lowest value. The formula for ReLate is defined as:

$$ReLate_p = \frac{\sum_{i=1}^{r} l_i}{r} \div \frac{r}{t}$$

where p is the protocol being evaluated,
r = number of updates received,
l_i = latency of update i,
and t = total number of updates sent.

The initial ReLate metric is helpful only for evaluating protocols that balance reliability and latency. This metric does not help us evaluate all the protocols that we have currently used, in particular the protocols that provide no reliability. Using this initial metric produces values that are lower than those for the reliable multicast and Ricochet protocols even with a significant percentage of network loss, *e.g.*, 3%.

For example, using the values from one of our experiments outlined in Section 4.2.1, the ReLate metric produces the lowest values for OpenDDS UDP, OpenDDS IP Mcast, and ANT Baseline even with 3% packet loss. None of these transport protocols with the lowest ReLate value provide reliability. Figure 14 in Section 4.2.1 presents these results in the context of ADAMANT.

For RT-ESP applications involving multimedia, such as our motivating example of SAR operations in Section 2, over 10% loss is generally considered unacceptable. Bai and Ito [1] limit acceptable MPEG video loss at 6% while stating that a packet loss rate of more than 5% is unacceptable for Voice over IP (VoIP) users [2]. Ngatman et al. [11] define consistent packet loss above 2% as unacceptable for videoconferencing. We use these values as guidelines to modify the ReLate metric.

The 10% loss unacceptability for multimedia is due to the interdependence of packets. As shown in Figure 8, MPEG frames are interdependent such that P frames are dependent on previous I or P frames while B frames are dependent on both preceding and succeeding I or P frames. The loss of

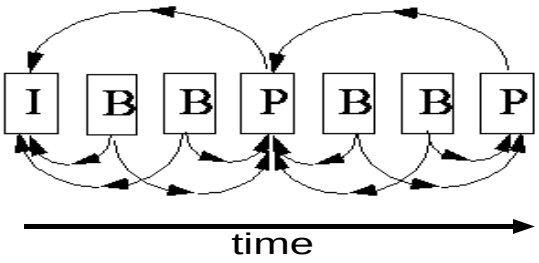

Figure 8: MPEG Frame Dependencies

an I or P frame therefore results in unusable dependent P and B frames, even if these frames are delivered reliably and in a timely manner.

We conservatively state that a 10% packet loss should result in an order of magnitude increase in the metric value generated. We therefore modified our ReLate metric to multiple the average latency by the percent packet loss as follows:

$$ReLate2_p = \frac{\sum_{i=1}^{r} l_i}{r} \times (\frac{t-r}{t} \times 100 + 1)$$

where p is the protocol being evaluated,
r = number of updates received,
l_i = latency of update i,
and t = total number of updates sent.

We add 1 to the percent packet loss to normalize for 0% loss and any loss less than 1% where the metric would yield the value 0 or a value lower than the average latency respectively. Section 4.2.1 uses the ReLate2 metric to determine the transport protocols that best balance reliability and latency.

4. EXPERIMENTAL SETUP, RESULTS, AND ANALYSIS

The section presents the results of experiments we conducted to determine the performance of ADAMANT in a representative RT-ESP environment. The experiments include ADAMANT using multiple transport protocols with varying numbers of receivers, percentage data loss, and sending rates as would be expected with SAR operations in a dynamic environment as described in Section 2.1.

4.1 Experimental Setup

We conducted our experiments using two network testbeds: (1) the Emulab network emulation testbed and (2) the ISISlab network emulation testbed. Emulab provides computing platforms and network resources that can be easily configured with the desired computing platform, OS, network topology, and network traffic shaping. ISISlab uses Emulab software and provides much of the same functionality, but does not (yet) support traffic shaping. We used Emulab due to its ability to shape network traffic. We used ISISlab due to the availability of needed computing platforms.

As outlined in Section 2, we are concerned with the distribution of data for SAR datacenters. As presented in [19], network packets are expected to be dropped at the end hosts. The Emulab network links for the receiving data readers were configured appropriately for the specified percentage loss. The experiments in ISISlab were conducted with modified source code to drop packets when received by data readers since ISISlab does not yet support network traffic shaping.

The Emulab network traffic shaping was mainly needed when using TCP. Unlike some middleware experimentation platforms, OpenDDS does not support programmatically dropping a percentage of packets in end hosts for TCP. We therefore used network traffic shaping for TCP which only Emulab provides.

Using the Emulab environment and the *ReLate2* metric defined in Section 3.2.3, we next determined the protocols that balanced latency and reliability well, namely RMcast and ANT Ricochet. Since we could programmatically control the loss of network packets at the receiving end hosts with these protocols, we then used ISISlab due to its availability of nodes to conduct more detailed experiments involving these two protocols. We were able to obtain upto 27 nodes fairly easily using ISISlab, whereas this number of nodes was hard to get with Emulab since it is often oversubscribed.

Our experiments using Emulab and ISISlab used the following traffic generation configuration using OpenDDS version 1.2.1.

- One DDS data writer wrote data and a variable number of DDS data readers read the data.
- The data writer and each data reader ran on its own computing platform.
- The data writer updated 12 bytes of data 20,000 times at the specified sending rate.

We ran 5 experiments for each configuration, *e.g.*, 5 receiving data writers, 50 Hz sending rate, 2% end host packet loss. We used Ricochet's default c value of 3 for both Emulab and ISISlab experiments.

4.1.1 Emulab Configuration

For Emulab, the data update rates were 25 Hz and 50Hz for general comparison of all the protocols. We varied the number of receivers from 3 up to 10. We used Ricochet's default r value of 8. As defined in Section 3.2.2, the r value is the number of packets received before sending out recovery data.

We used the Emulab pc850 hardware platform, which includes an 850 MHz processor and 256 MB of RAM. We ran the Fedora Core 6 operating system with real-time extensions on this hardware platform, using experiments consisting of between 5 and 12 pc850 nodes. The nodes were all configured in a LAN configuration. We utilized the traffic shaping feature of Emulab to run experiments with network loss percentages between 0 and 3 percent. Table 2 outlines the points of variability for the Emulab experiments.

Point of Variability	Values
Number of receiving data writers	3 - 10
Frequency of sending data	25 Hz, 50 Hz
Percent end-host network loss	0 to 3 %

Table 2: Emulab Experiment Variables

4.1.2 ISISlab Configuration

We used ISISlab for experiments involving transport protocols where we could programmatically affect the loss of packets in the end hosts. By modifying the source code, we could discard packets based on the desired percentage. In particular, we focused the ISISlab experiments on the OpenDDS RMcast and the ANT Ricochet protocols since from the initial experiments these protocols showed the ability to balance latency and reliability. We also used ISISlab since it was easier to obtain a larger number of nodes needed for experiments than with Emulab. Table 3 outlines the points of variability for the ISISlab experiments.

ISISlab provides a single type of hardware platform: the pc8832 hardware platform with a dual 2.8 GHz processor and 2 GB of RAM. We used the same Fedora Core 6 OS with real-time extensions as for Emulab. We ran experiments using between 5 and 27 computing nodes which map to between 3 and 25 data writers respectively. All nodes were configured in a LAN as was done for Emulab. We modified Ricochet's r value so it was either 8 or 4, as explained in Section 4.2.2.

Point of Variability	Values
Number of receiving data writers	3 - 25
Frequency of sending data	10 Hz, 25 Hz, 50 Hz, 100 Hz
Percent network loss	0 to 3 %
Ricochet r value	4, 8

Table 3: ISISlab Experiment Variables

4.2 Results and Analysis of Experiments

This section presents and analyzes the results from our experiments, which resolves Challenge 3 in Section 2.2.3 by characterizing the performance of the transport protocols for various environment configurations.

4.2.1 The Baseline Emulab Experiments

Experiment design. The initial set of experiments for the ADAMANT prototype included all the OpenDDS protocols as enumerated in Section 3.2. These experiments used Emulab as described in Section 4.1. Our baseline experiments used 3 data readers, 0% loss, and 25 and 50 Hz update rates. As expected, all protocols delivered all data to all data readers, *i.e.*, 3 receivers * 20,000 updates = 60,000 updates.

Experiment results. As shown in Figures 9 and 10, the latency was lowest with protocols that do not provide reliability support, *i.e.*, OpenDDS UDP, OpenDDS IP Mcast, and ANT Baseline (*i.e.*, IP Multicast). The OpenDDS RMcast and ANT Ricochet protocols were the only ones that never produced the lowest overall average latency. As expected, average latency times decreased as the sending rate increased from 25 Hz to 50 Hz.

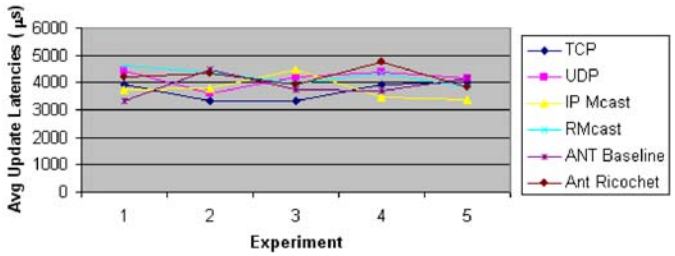

Figure 9: Baseline Emulab Experiment: 3 data readers, 0% loss, 25 Hz update rate

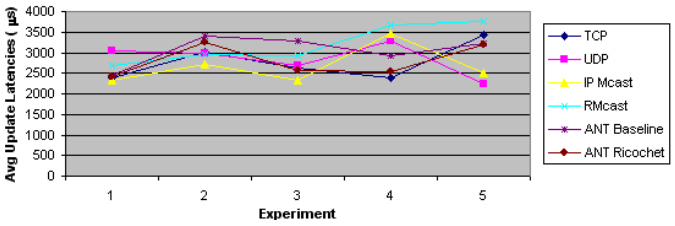

Figure 10: Baseline Emulab Experiment: 3 data readers, 0% loss, 50 Hz update rate

The next set of experiments added 3% network packet loss for the receiving end hosts. As shown in Figures 11 and 12, TCP has the highest average latency times while OpenDDS UDP, OpenDDS IP Mcast, and ANT Baseline have the lowest average latencies.

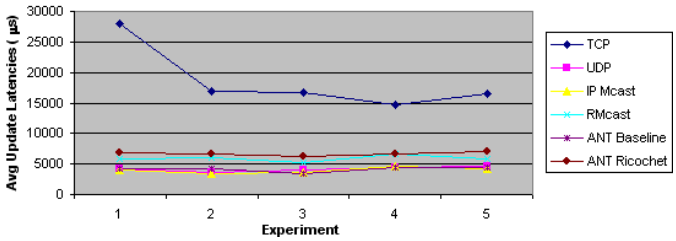

Figure 11: Baseline Emulab Experiment: 3 data readers, 3% loss, 25 Hz update rate

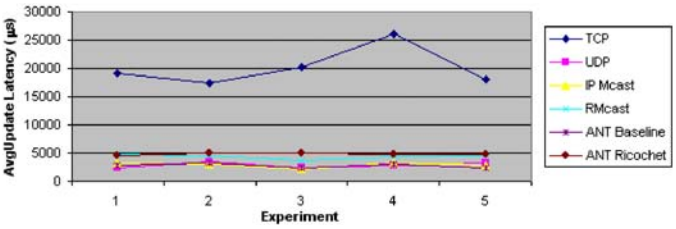

Figure 12: Baseline Emulab Experiment: 3 data readers, 3% loss, 50 Hz update rate

Figure 13 shows the number of packets received when the update rate is 50 Hz. As expected, the data readers did not receive all updates. We do not include packet loss figures for the 25 Hz update rate as it is comparable to the loss seen with a sending rate of 50 Hz.

For 3% network loss, TCP delivers all the packets for every experiment. OpenDDS Reliable IP Mcast delivers all the packets for some experiments but not all. ANT Ricochet always delivers the second most highest number of updates with the percentage delivered being between 99.5% and 99.6%.

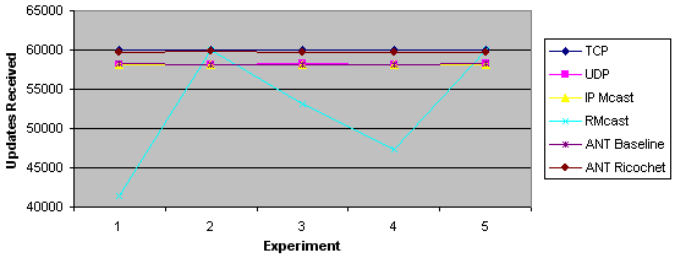

Figure 13: Baseline Emulab Experiment: Updates Received (3 data readers, 3% loss, 50 Hz update rate)

We were unable to configure OpenDDS IP Mcast to use Emulab's network traffic shaping. Instead we calculated the amount of packet loss that is comparable to the other unreliable transports. This calculation does not invalidate the values seen and used for OpenDDS IP Mcast.

Analysis of results. We now analyze the results of the Emulab experiments, which involved all transport protocols outlined in Section 3.2. We utilize the ReLate and ReLate2 metrics defined in Section 3.2.3 to evaluate the results from the initial Emulab experiments.

We compare the values from the ReLate2 metric as shown in Figure 15 with the values in Figure 14 which were only based on the original Relate metric. The results show that

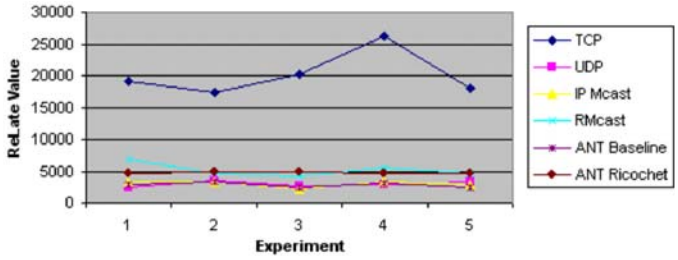

Figure 14: ReLate Metrics for Emulab Experiment: 3 data readers, 3% loss, 50 Hz update rate

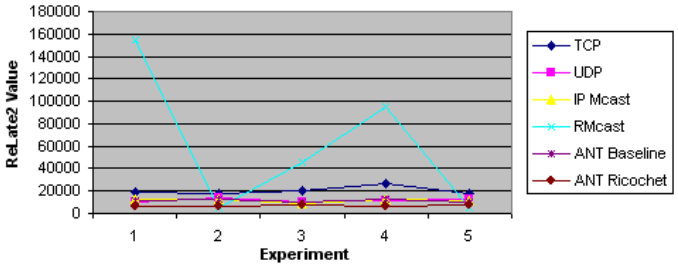

Figure 15: ReLate2 Metrics for Emulab Experiment: 3 data readers, 3% loss, 50 Hz update rate

OpenDDS RMcast and ANT Ricochet always produce the lowest ReLate2 value. Moreover, when there is no loss, the ReLate2 value is equal to the average latency as is the case for TCP. This comparison shows that the ReLate2 metric is useful for evaluating protocols that balance reliability and latency.

4.2.2 The RMcast and Ricochet Experiments

Experiment design. Our next experiment focused on the protocols that are best suited for balancing reliability and latency based on the ReLate2 metric (*i.e.*, OpenDDS RMcast and ANT Ricochet). We focus on these protocols for comparison to gain a better understanding of trade-offs between them. We provide experimental results and analyze the results.

As noted in Section 4.1, we used the ISISlab testbed for experiments involving only OpenDDS RMcast and ANT Ricochet due to the availability of a larger number of hardware nodes. We were able programmatically to induce packet loss at the end hosts for these two protocols since the source code is available for both and thus did not require Emulab's network traffic shaping capability.

As with the Emulab experiments in Section 4.2.1, we began with experiments where the number of receivers and packet loss were low. We also expanded the sending rates to include 10Hz and 100Hz along with the original rates of 25Hz and 50Hz. Adding sending rates made sense as the packet loss recovery times for both of these protocols are sensitive to the update rate.

The packet loss recovery time for the RMcast is sensitive to the update rate since loss is only discovered when packets are received. If packets are received faster then packet loss is discovered sooner and recovery packets can be requested, received, and processed sooner. Likewise, the packet loss recovery time for Ricochet is sensitive to the update rate since recovery data is only sent out after r packets have been received. When packets are received sooner, recovery data is sent, received, and processed sooner.

Experiment results. Figures 16, 17, and 18, show that for a low number of receivers, *i.e.*, 3, and a low loss percentage, *i.e.*, 1%, RMcast, in general, has lower overall latency and lower ReLate2 values.

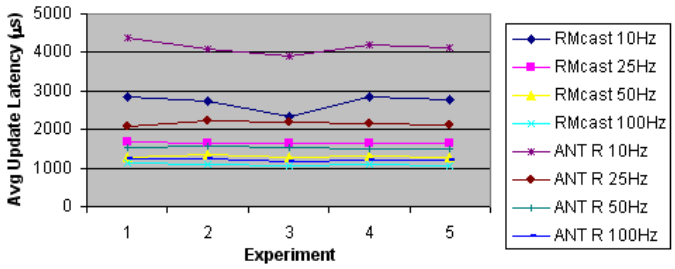

Figure 16: Latency (μs) for ISISlab Experiment: 3 data readers, 1% loss

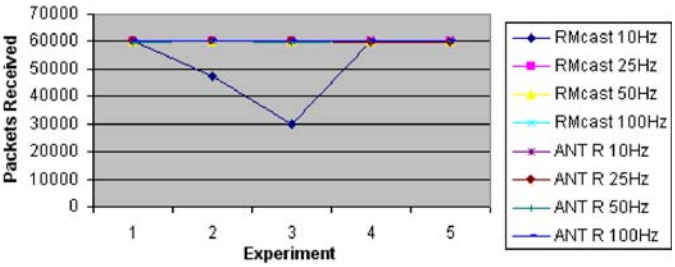

Figure 17: Packets Received for ISISlab Experiment: 3 data readers, 1% loss

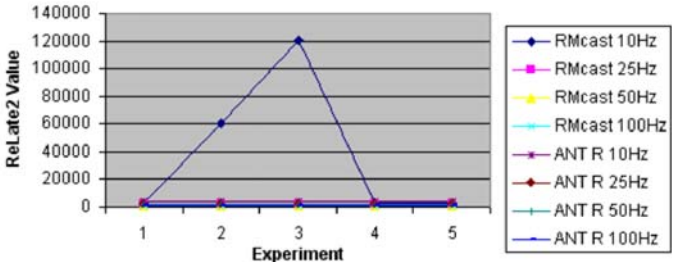

Figure 18: ReLate2 Values for ISISlab Experiment: 3 data readers, 1% loss

Analysis of results. The data in Figures 16, 17, and 18, highlight patterns we observed in our experiments. RMcast can at times fail to receive a disproportionate percentage of messages. For example, experiment 2 in Figure 17 shows that RMcast delivered only 79% of the potential packets and experiment 3 only 49%. These percentages are clearly above the 1% loss specified.

On further inspection, RMcast appears to stop delivering packets after a certain point. If there is a NAK storm (*i.e.*, where a sender is inundated with retransmission requests) we would expect latency times to increase, but still have all packets received. This loss of data by RMcast also typically happens with lower update rates. This result is counter-intuitive since we would expect a lower sending rate to pro-

vide the protocol more time to process any retransmission requests.

We found this anomaly of RMcast is common throughout our experiments with the likelihood of it increasing as the percent loss increases and to a much lesser degree as the number of receivers increases. The ReLate2 metric tends to favor the Ricochet protocol as percent loss or number of receivers increases. This result stems mainly from RMcast failing to deliver messages above and beyond the specified packet loss rather than increased average latency times.

As indicated in Figures 16, 17, and 18, we observed the pattern of Ricochet consistently delivering a high percentages of updates. The ReLate and ReLate2 values for Ricochet have been consistent, as well. We next focus our attention on the configurability of the Ricochet protocol to determine if the average latency for Ricochet can begin to match the average latency for RMcast.

As with the Emulab experiments we started the ISISlab experiments using Ricochet's default r value of 8. This value denotes the number of packets received before a recovery packet is sent out. For all experiments conducted using the default r value of 8, Ricochet's average update latency was never less than that of the RMcast protocol.

Leveraging the fact that the r value had a significant impact on average latency, we conducted experiments with Ricochet's r value set to 4. These experiments showed the average latency of RMcast exceed the average latency of Ricochet, as shown in Figures 19, 20, and 21,

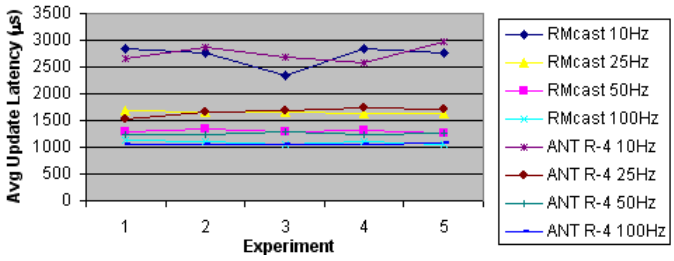

Figure 19: Latency (μs) for ISISlab Experiment: 3 data readers, 1% loss

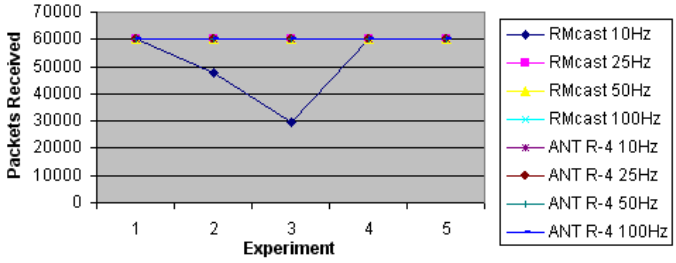

Figure 20: Packets Received for ISISlab Experiment: 3 data readers, 1% loss

Each figure represents data for latency, packets received, and ReLate2 value, respectively. In these experiments Ricochet had the lowest average latency 9 out of 20 times and the lowest ReLate2 value 11 out of 20 times.

As the number of receivers increases the average update latency of Ricochet configured with $r = 4$ decreases such that eventually it becomes consistently less than for RMcast as

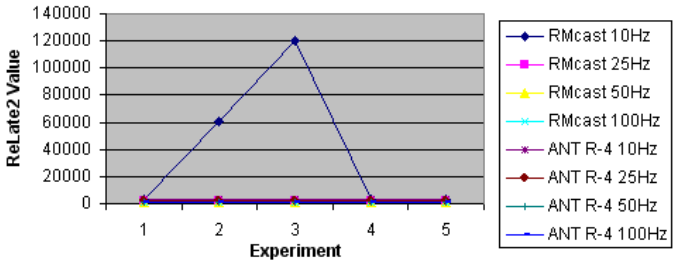

Figure 21: ReLate2 Values for ISISlab Experiment: 3 data readers, 1% loss

shown in Figure 22. The *only* time in this experiment configuration when RMcast's latency is less than ANT R's is when RMcast delivers significantly fewer packets. When ANT R's latency is calculated with the same number of packets as RMcast delivered ANT R's latency is less, *i.e.*, there is time that accumulates in the middleware as more messages are received.

The ReLate2 metric does not consistently become less than RMcast, however. This difference is due to the decrease in reliability that occurs with Ricochet as the number of receivers increases. The decrease is small with the reliability for Ricochet ranging from 99.96% to 99.99% for 3 receivers with 1% loss for all 4 update rates and ranging from 99.94% to 99.95% when the number of receivers is increased to 25.

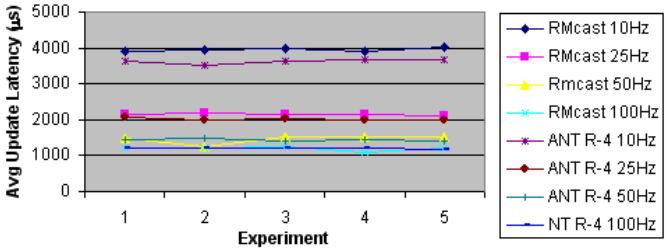

Figure 22: Latency (μs) for ISISlab Experiment: 25 data readers, 1% loss

The ReLate2 metric accounts for the changes in reliability. When RMcast is able to deliver all messages (which becomes less unlikely as the loss percentage or number of receivers increases) its ReLate2 value can be lower than that for Ricochet even when the average latency is consistently higher. Ricochet provides the c parameter which can be adjusted to increase the number of receivers to which a single receiver sends out recovery data. The flexibility of the r and c parameters highlight the suitability of Ricochet with QoS-enabled pub/sub middleware for RT-ESP applications in turbulent environments. One challenge is to manage this flexibility in a timely manner as the environment changes. We are planning further research in this area.

5. RELATED WORK

This section compares our work on performance evaluation of ADAMANT with related R&D efforts.

Performance evaluation of network transport protocols. Much work has been done evaluating various network transport protocols. For example, Balakrishnan et al. [20] evaluate the performance of the Ricochet transport

protocol with the Scalable Reliable Multicast (SRM) protocol [21]. Bateman et al. [10] compare the performance of TCP variations both using simulations and in a testbed. Cheng et al. [15] provide performance comparisons of UDP and TCP for video streaming in multihop wireless mesh networks. Kirschberg et al. [8] propose the Reliable Congestion Controlled Multicast Protocol (RCCMP) and provide simulation results for its performance. In contrast to our work on ADAMANT, these evaluations specifically target the protocol level independent of any integration of QoS-enabled pub/sub middleware.

Performance evaluation of enterprise middleware. Xiong *et al.* [12] conducted performance evaluations for three DDS implementations, including OpenDDS. That work highlighted the different architectural approaches taken and trade-offs of these approaches. In contrast, to our work on ADAMANT, however, that prior work did not include performance evaluations of DDS with various transport protocols.

Sachs *et al.* [9] present a performance evaluation of message-oriented middleware (MOM) in the context of the SPECjms2007 (`www.spec.org/jms2007`) standard benchmark for MOM servers. The benchmark is based on the Java Message Service (JMS) (`java.sun.com/products/jms`). In particular, the work details performance evaluations of the BEA WebLogic server under various loads and configurations. In contrast to our work on ADAMANT, however, that work did not integrate various transport protocols with the middleware to evaluate its performance.

Tanaka *et al.* [16] developed middleware for grid computing called Ninf-G2. In addition, they evaluate Ninf-G2's performance using a weather forecasting system. The evaluation of the middleware does not integrate various protocols and evaluate performance in this context, as our work on ADAMANT does.

Tselikis *et al.* [5] conduct performance analysis of a client-server e-banking application. They include three different enterprise middleware platforms each based on Java, HTTP, and Web Services technologies. The analysis of performance data led to the benefits and disadvantages of each middleware technology. In contrast, our work on ADAMANT measures the impact of various network protocols integrated with QoS-enabled pub/sub middleware.

Performance evaluation of embedded middleware. Bellavista *et al.* [13] describe their work called Mobile agent-based Ubiquitous multimedia Middleware (MUM). MUM has been developed to handle the complexities of wireless hand-off management for wireless devices moving among different points of attachment to the Internet. In contrast, our work on ADAMANT focuses on the performance and flexibility of QoS-enabled anonymous pub/sub middleware.

TinyDDS [3] is an implementation of DDS specialized for the demands of wireless sensor networks (WSNs). TinyDDS defines a subset of DDS interfaces for simplicity and efficiency within the domain of WSNs. TinyDDS includes a pluggable framework for non-functional properties, *e.g.*, event correlation and filtering mechanisms, data aggregation functionality, power-efficient routing capability. In contrast, our work on ADAMANT focuses on how properties of various transport protocols can be used to maintain specified QoS.

6. CONCLUDING REMARKS

Developers of *Real-time Event Stream Processing* (RT-ESP) systems face a number of challenges when developing their applications for dynamic environments. To address these challenges, we have developed ADAMANT to integrate and enhance QoS-enabled pub/sub middleware with adaptive transport protocols to support RT-ESP applications. This paper defines metrics that empirically measure the reliability and latency of ADAMANT as a first step to having QoS-enabled pub/sub middleware autonomically adapt transport protocols as the changing environment dictates.

The following is a summary of lessons learned from our experience evaluating ADAMANT's performance with various transport protocols:

• **Exploring a configuration space for trade-offs requires a disciplined approach with analysis to guide the exploration.** Depending on the number of dimensions involved in the search space there can be many configurations to explore. In our case, we had multiple variables, *e.g.*, update rate, % loss, number of data readers, and Ricochet's r value. Since the number of potential experiments was large, we found it helpful to make coarse-grained adjustments for initial experiments. We would then analyze the results to guide areas of refinement to find trade-offs between transport protocols. For example, varying Ricochet's r value (see Section 4.2.2) occurred as a result of analyzing early experimental results.

• **Integrating pub/sub middleware with transport protocols exacerbates the challenge of pinpointing the source of problems and anomalies.** Certain experiments incurred unexpected behavior, such as RMcast at times only providing a small percentage of updates. With the integration of middleware and transport protocols, determining where deficiencies lie can be hard since problems could be in the middleware, the protocol, or the combination of both. In addition to individually testing protocols and the middleware, therefore, it was helpful to compare the anomalous behavior of a protocol with other protocols keeping the same configuration environment. For example, Section 4.2.2 described how we used these comparisons to determine unexpected behavior coming from RMcast rather than the OpenDDS transport protocol framework or pub/sub middleware.

• **The manual integration of QoS with pub/sub middleware and transport protocols is tedious and error-prone.** Currently, pub/sub middleware and transport protocols integrators must manually manage QoS properties specified in the middleware with QoS properties provided by a transport protocol. For example, an integrator could mistakenly select a transport protocol with no reliability support even though application developers specified reliable communication. The middleware does not help in determining the mismatch between QoS properties and transport protocol properties. Our future work is investigating ways to manage this complexity via domain-specific modeling languages (DSMLs) that provide profiles for certain types of applications, such as RT-ESP applications. Once a profile is selected, the DSML could automatically generate correct implementation artifacts for the application.

• **Specifying unacceptable loss for RT-ESP is hard to generalize.** The amount of acceptable loss is specific to a particular application or application type. However, a general acceptability guideline of 10 % loss or less for multimedia applications has been helpful in making initial eval-

uations of protocols that balance reliability and latency.

• **Flexible transport protocols make manual management and tuning of the protocols hard.** Our experiments show the adaptability of the Ricochet transport protocol. Modifying Ricochet's r value affects the average overall latency, as shown by our results in Section 4.2.2. Likewise, the modification of Ricochet's c value can affect the percentage of recovered packets with a corresponding impact on bandwidth.

Keeping protocol parameter settings optimized in a turbulent environment can quickly become overwhelming if done manually. Reaction time needed can swiftly surpass those of humans. We are researching the use of machine learning to automatically adjust parameter settings appropriately based on the environment and the QoS specified by the application. We anticipate our experimental data to be used for (supervised or unsupervised) machine learning to dynamically optimize parameter settings.

• **Multicast with NAK-based reliability and LEC protocols balance reliability and latency.** After conducting the experiments and using our ReLate metrics we determined that when combining low latency and reliability, multicast with NAK-based reliability and LEC protocols deliver the best performance. NAK-based protocols have fairly low overhead and low bandwidth usage for low loss rates since only the detected loss of a packet triggers recovery actions. Moreover, we found that Ricochet is consistently reliable with a high probability. Ricochet also provides consistent bandwidth usage for r and c settings which can be important for network constrained environments.

7. REFERENCES

[1] Y. Bai and M. Ito. A new technique for minimizing network loss from users' perspective. *Journal of Network Computing Appllications*, 30(2):637–649, 2007.

[2] Yan Bai and M.R. Ito. A Study for Providing Better Quality of Service to VoIP Users. In *20th International Conference on Advanced Information Networking and Applications (AINA 2006)*, Lecture Notes in Computer Science, vol. 3410, pages 799–804, April 2006.

[3] P. Boonma and J. Suzuki. Middleware support for pluggable non-functional properties in wireless sensor networks. *Services - Part I, 2008. IEEE Congress on*, pages 360–367, July 2008.

[4] B. Plale *et al.* CASA and LEAD: Adaptive Cyberinfrastructure for Real-Time Multiscale Weather Forecasting. *Computer*, 39(11):56–64, 2006.

[5] C. Tselikis *et al.* An evaluation of the middleware's impact on the performance of object oriented distributed systems. *Journal of Systems and Software*, 80(7):1169 – 1181, 2007. Dynamic Resource Management in Distributed Real-Time Systems.

[6] E. Gamma *et al. Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

[7] G. Eisenhauer *et al.* Publish-subscribe for high-performance computing. *Internet Computing, IEEE*, 10(1):40–47, Jan.-Feb. 2006.

[8] J. Kirschberg *et al.* Rccmp: reliable congestion controlled multicast protocol. In *1st EuroNGI COnference on Next Generation Internet Networks Traffic Engineering*, april 2005.

[9] K. Sachs *et al.* Performance Evaluation of Message-oriented Middleware using the SPECjms2007 Benchmark. *Performance Evaluation*, 2009. to appear.

[10] M. Bateman *et al.* A comparison of tcp behaviour at high speeds using ns-2 and linux. In *CNS '08: Proceedings of the 11th communications and networking simulation symposium*, pages 30–37, New York, NY, USA, 2008. ACM.

[11] M. Ngatman *et al.* Comprehensive study of transmission techniques for reducing packet loss and delay in multimedia over ip. *International Journal of Computer Science and Network Security*, 8(3):292–299, 2008.

[12] Ming Xiong *et al.* Evaluating Technologies for Tactical Information Management in Net-Centric Systems. In *Proceedings of the Defense Transformation and Net-Centric Systems conference*, Orlando, Florida, April 2007.

[13] P. Bellavista *et al.* Context-aware handoff middleware for transparent service continuity in wireless networks. *Pervasive and Mobile Computing*, 3(4):439 – 466, 2007. Middleware for Pervasive Computing.

[14] V. Kumar *et al.* Distributed stream management using utility-driven self-adaptive middleware. *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, pages 3–14, June 2005.

[15] X. Cheng *et al.* Performance evaluation of video streaming in multihop wireless mesh networks. In *NOSSDAV '08: Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 57–62, New York, NY, USA, 2008. ACM.

[16] Y. Tanaka *et al.* Design, Implementation and Performance Evaluation of GridRPC Programming Middleware for a Large-Scale Computational Grid. In *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 298–305, Washington, DC, USA, 2004. IEEE Computer Society.

[17] D. Schmidt et al. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, Volume 2*. Wiley & Sons, New York, 2000.

[18] J. Nonnenmacher et al. Parity-based loss recovery for reliable multicast transmission. volume 6, pages 349–361, Piscataway, NJ, USA, 1998. IEEE Press.

[19] M. Balakrishnan et al. Slingshot: Time-critical multicast for clustered applications. In *Proceedings of the IEEE Conference on Network Computing and Applications*, 2005.

[20] M. Balakrishnan et al. Ricochet: Lateral error correction for time-critical multicast. In *NSDI 2007: Fourth Usenix Symposium on Networked Systems Design and Implementation*, Boston, MA, 2007.

[21] S. Floyd et al. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Trans. Netw.*, 5(6):784–803, 1997.

[22] Christian Huitema. The case for packet level fec. In *TC6 WG6.1/6.4 Fifth International Workshop on Protocols for High-Speed Networks V*, pages 109–120, London, UK, 1997.