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

Joe Hoffert, Douglas Schmidt, and Aniruddha Gokhale

Institute for Software Integrated Systems, Dept. of EECS,
Vanderbilt University, Nashville, TN, USA 37203
{jhoffert,schmidt,gokhale}@dre.vanderbilt.edu
www.dre.vanderbilt.edu

Abstract. Real-time event stream processing (RT-ESP) applications must synchronize continuous data streams 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 flexible enough to tune the existing transport protocol or switch to a transport protocol better suited to the 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 ReLate2, which is an evaluation metric that combines packet latency and reliability to evaluate transport protocol performance. Second, we use the ReLate2 metric to quantify the performance of various transport protocols integrated with the OMG's Data Distribution Service (DDS) QoS-enabled pub/sub middleware standard using our FLEXible Middleware And Transports (FLEXMAT) prototype for experiments that capture performance data. Third, we use ReLate2 to pinpoint configurations involving sending rate, network loss, and number of receivers that show the pros and cons of the protocols.

Key words: Pub/Sub Middleware, Data Distribution Service, Transport Protocols, Metrics

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.

* This work is supported in part by the AFRL/IF Pollux project and NSF TRUST.

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 [18, 8].

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 continue to increase, the communication infrastructure must be able to coordinate these streams so that publishers and subscribers are connected appropriately. Flexible communication infrastructure must adapt to fluctuating demands for various event streams and environment changes to maintain acceptable levels of service.

• Certain types of large-scale RT-ESP applications require a flexible communication infrastructure due to their dynamic and *ad hoc* nature. These application environments incur fluctuations in resource availability 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 regional disasters.

Several pub/sub middleware platforms have been developed to support large-scale data-centric distributed systems, such as the Java Message Service, Web Services Brokered Notification, and the CORBA Event Service. These platforms, however, do not support fine-grained and robust QoS. Some large-scale distributed system platforms, such as the Global Information Grid and Network-centric Enterprise Services, 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 routers quickly alternating destinations (*i.e.*, route flaps).

Solution approach → A FLEXible Middleware And Transports (FLEXMAT) Evaluation Framework. Developing such a flexible communication infrastructure is hard because it must have a detailed understanding of the capabilities that the underlying transport protocols provide. The infrastructure must also understand how these protocols behave under different operating conditions stemming from both the application-imposed workload changes, as well as system dynamics, such as failures and network congestion. Building on this understanding, QoS-enabled pub/sub middleware can help alleviate the complexity of managing multiple event streams and maintaining real-time QoS for multiple event streams in highly dynamic environments.

This paper describes the design and capabilities of the *FLEXible and Integrated Middleware and Transport Evaluation Framework* (FLEXMAT) to address these requirements. To evaluate the impact of various transport protocols that can lead

to the realization of a QoS-enabled pub/sub middleware we developed *ReLate2*, which is a composite metric for FLEXMAT that considers both reliability and latency. This paper uses ReLate2 to evaluate the reliability and latency of transmitted data for various experimental configurations involving parameters such as sending rate, network loss, and number of receivers.

To facilitate the empirical benchmarking environment, and collection of the Relate2 metrics, FLEXMAT integrates and enhances the following capabilities:

• The *Adaptive Network Transports* (ANT) framework, which provides infrastructure for composing transport protocols that builds upon properties provided by the scalable reliable multicast-based Ricochet transport protocol [12]. Ricochet enables trade-offs between latency and reliability, which are needed qualities for pub/sub 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 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.

We apply the ReLate2 metric across various commonly used and custom FLEXMAT transport protocols. We then empirically quantify the results and analyze the pros/cons of various transport protocol configurations in the context of FLEXMAT. By capturing the insights gained from this effort, our goal is to enhance the development and validation of QoS-enabled pub/sub middleware.

Paper organization. The remainder of this paper is organized as follows: Section 2 describes a representative RT-ESP application to motivate the challenges that FLEXMAT is designed to address; Section 3 examines the structure and functionality of FLEXMAT and the ReLate2 metric we created to evaluate FLEXMAT and its adaptive transport protocol framework; Section 4 analyzes the results of experiments conducted by applying the ReLate2 metric to FLEXMAT; Section 5 compares FLEXMAT with related work; and Section 6 presents concluding remarks.

2 Motivating the Need for FLEXMAT

This section describes a representative RT-ESP application to motivate the challenges that FLEXMAT addresses.

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, our FLEXMAT work is motivated in the context of 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 can 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 to a

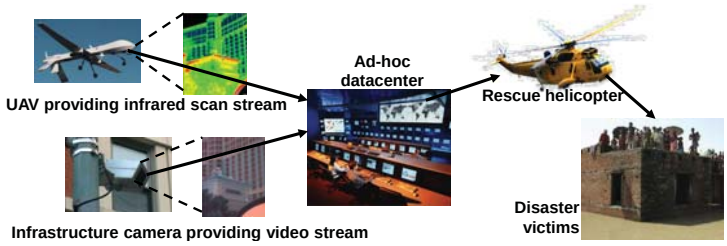

Fig. 1. Search and Rescue Motivating Example

datacenter, where they are processed by fusion applications to detect survivors. Once survivors are 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 expending scarce resources such as rescue workers, rescue vehicles, and data center coordinators unnecessarily.

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 FLEXMAT addresses to support the communication requirements of the SAR operations presented above.

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. Section 3.1 describes how FLEXMAT addresses this challenge by incorporating transport protocols that balance reliability and low latency.

Challenge 2: Managing Subscription of Event Data Streams Dynamically SAR operations must seamlessly incorporate and remove particular event

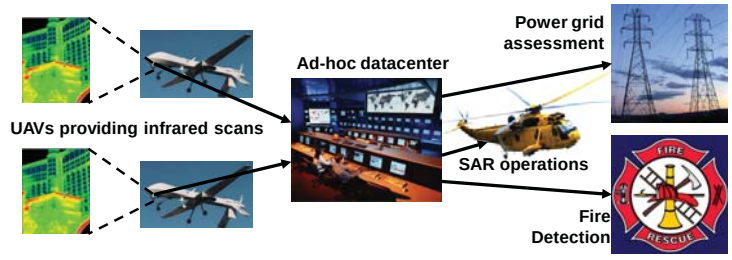

Fig. 2. Uses of Infrared Scans during Disaster Recovery

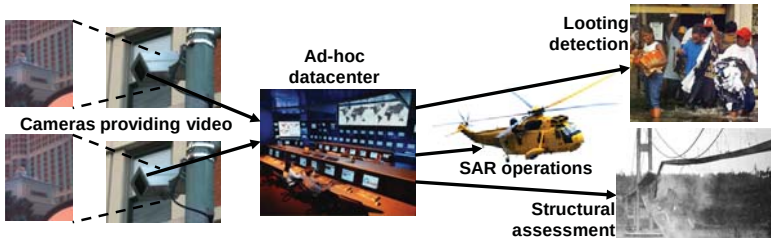

Fig. 3. Uses of Video Stream during Disaster Recovery

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.

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 may not perform as needed 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.

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 use them to provide higher fidelity or more expansive coverage. Section 3.1 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 FLEXMAT and ReLate2

This section presents an overview of FLEXMAT, including the OpenDDS and ANT transport protocols it uses. We then describe the ReLate2 metric created to

evaluate the performance of FLEXMAT in various environment configurations to support RT-ESP application requirements for data reliability and timeliness.

3.1 Design of FLEXMAT and Its Transport Protocols

FLEXMAT integrates and enhances QoS-enabled pub/sub middleware with adaptive transport protocols to provide the flexibility needed by RT-ESP applications. FLEXMAT helps resolve Challenge 2 in Section 2.2 by providing anonymous publication and subscription via the OMG Data Distribution Service (see Sidebar 1 for a brief summary of DDS). FLEXMAT is based on the OpenDDS implementation of DDS and incorporates several standard and custom transport protocols.

We chose OpenDDS as FLEXMAT's DDS implementation due to its (1) open source availability, 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 [7] and Component Configurator [6]) to provide flexibility and delegate responsibility to the protocol only when applicable.

Sidebar 1: Overview of DDS

The OMG Data Distribution Service (DDS) specifies 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. This paper focuses 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.

OpenDDS currently provides several transport protocols. Other protocols for the FLEXMAT prototype are custom protocols (described below) that we integrated with OpenDDS using its pluggable transport framework.

OpenDDS Transport Protocols. By default, OpenDDS provides four transport protocols in its transport protocol framework: TCP, UDP, IP multicast (IP Mcast), and a NAK-based reliable multicast (RMcast) protocol, as shown in Figure 4. OpenDDS TCP is a reliable unicast protocol, whereas UDP is an unreliable unicast protocol. IP Mcast can send data to multiple receivers.

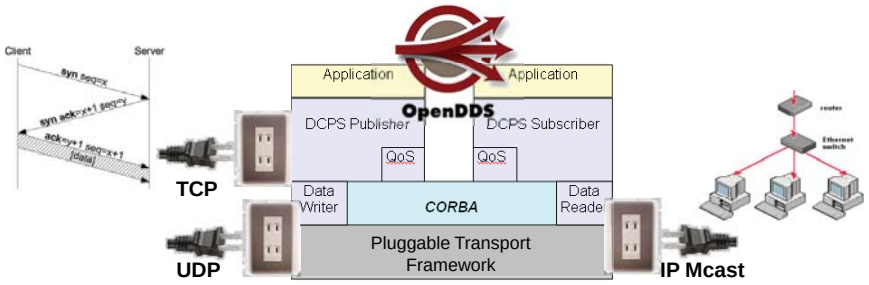

Fig. 4. OpenDDS and its Transport Protocol Framework

While TCP, UDP, and IP Mcast are standard protocols, RMcast warrants more description. It is a negative acknowledgment (NAK) protocol that provides reliability. For example, the sender sends four data packets, but the third data packet is not received by the receiver. The receiver realizes this 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 by providing reliability and timeliness for certain environment configurations.

Adaptive Network Transport Protocols. The ANT transport protocol framework supports various transport protocol properties, including multicast, packet tracking, NAK-based reliability, ACK-based reliability, flow control, group membership, and membership fault detection. These properties can be composed dynamically at run-time to achieve greater flexibility and support adaptation.

The ANT framework originally was developed from the Ricochet [12] 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.

We included ANT's Ricochet transport protocol, ANT's NAKcast protocol, which is a NAK-based multicast protocol, and ANT's baseline transport protocol in FLEXMAT. The ANT Baseline protocol mirrors the functionality of IP Mcast as described in Section 3.1. 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. Sender-based FEC protocols have the sender send redundant information, as shown in Figure 5. In contrast, receiver-based FEC (a.k.a. Lateral Error Correc-

tion (LEC)) have receivers send each other redundant information as shown in Figure 6. The Ricochet protocol we employ in FLEXMAT is an example of an LEC protocol.

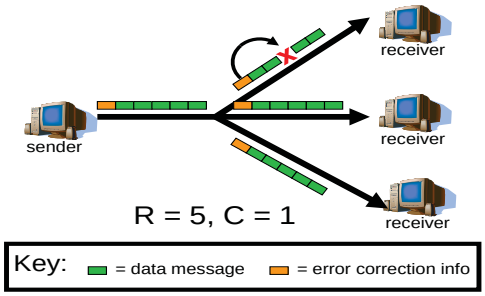

Fig. 5. FEC Reliable Multicast Protocol - Sender-based

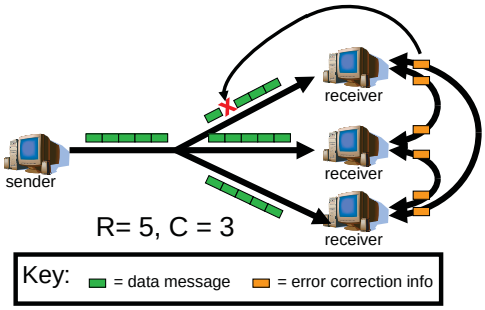

Fig. 6. FEC Reliable Multicast Protocol - Receiver-based (LEC)

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 the 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 *receiver*, 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 by providing high probabilistic reliability and low latency error detection and recovery. Ricochet also helps resolve Challenge 4 in Section 2.2 by supporting tunable parameters that effect reliability, latency, and bandwidth usage. We designed the ANT framework so that different transport protocols can be switched dynamically.

3.2 Evaluation Metric for Reliability and Latency

We now describe considerations for evaluating FLEXMAT's latency and reliability. We present guidelines for unacceptable percentages of packet loss for multimedia applications. We also introduce the *ReLate2* metric used to evaluate FLEXMAT empirically in Section 4.

One way 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. Moreover, depending upon the environment configuration the average data latency between protocols differs as well. To compare results, the level of reliability must also be quantified.

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. [14] define consistent packet loss above 2% as unacceptable for videoconferencing. We use these values as guidelines to develop the *ReLate2* metric that balances reliability and latency

The 10% loss unacceptability for multimedia is due to the interdependence of packets. For example, 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 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 any metric value generated. We therefore developed our ReLate2 metric to multiply 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 packets received,
l_i = latency of packet i,
and t = total number of packets sent.

We add 1 to the percent packet loss to normalize for any loss less than 1% where the metric would otherwise yield a value lower than the average latency, specifically the value 0 where all packets are delivered. This adjustment produces a ReLate2 value equal to the average latency when there is no packet loss which still accommodates meaningful comparisons for protocols that deliver all packets. Section 4.2 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 FLEXMAT in a representative RT-ESP environment. The experiments include FLEXMAT 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 and ISISlab due to the availability of computing platforms.

As outlined in Section 2, we are concerned with the distribution of data for SAR datacenters, where network packets are dropped at end hosts [11]. 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. 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, we next determined the protocols that balanced latency and reliability well, namely RMcast, ANT NAKcast, 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 protocols. We obtained up to 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 utilizing OpenDDS version 1.2.1: (1) one DDS data writer wrote data, variable number of DDS data readers read data, (2) the data writer and each data reader ran on its own computing platform, and (3) the data writer sent 12 bytes of data 20,000 times at a specified sending rate. To account for experiment variations 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.

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 3.1, 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 1 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 1. Emulab Variables

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 5 %

Table 2. ISISlab Variables

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 ANT NAKcast and Ricochet protocols since from the initial experiments these protocols showed the ability to balance latency and reliability. At times, OpenDDS RMcast showed the ability to balance reliability and low latency. Since its behavior was erratic for a NAK-based protocol, however, we excluded it from the detailed experiments. Table 2 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 readers respectively. All nodes were configured in a LAN as was done for Emulab. We ran experiments using Ricochet's R value of 8 and 4, as explained in Section 4.2.

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 by characterizing the performance of the transport protocols for various environment configurations.

The Baseline Emulab Experiments. The initial set of experiments for the FLEXMAT prototype included all the OpenDDS protocols as enumerated in Section 3.1. 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.

As shown in Figures 7 and 8, the latency at times was lowest with protocols that do not provide any reliability, *i.e.*, OpenDDS UDP, OpenDDS IP Mcast, and ANT Baseline). 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.

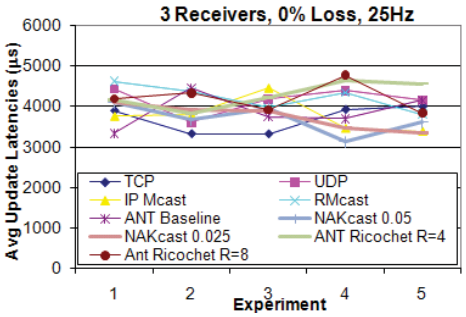

Fig. 7. Emulab: 3 readers, 0% loss, 25Hz

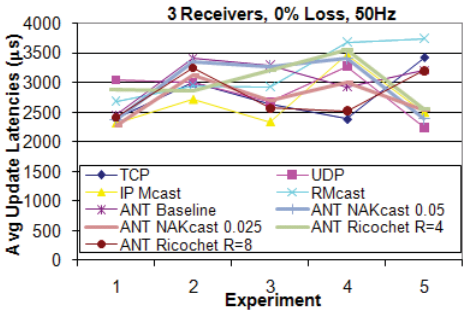

Fig. 8. Emulab: 3 readers, 0% loss, 50Hz

The next set of experiments added 1% network packet loss for the receiving end hosts. We do not include figures for the 50 Hz update rate as the data are comparable to that seen with a sending rate of 25 Hz. As shown in Figure 9, there is a clear delineation between the protocols that provide reliability and those that do not.

TCP received all updates sent, whereas ANT NAKcast and ANT Ricochet received a high percentage of updates with ANT NAKcast receiving all updates except for one experiment run where it received 59,999 out of the 60,000 updates. Both configurations of ANT Ricochet delivered a consistently high percentage of updates between 99.95% and 99.99%. UDP, IP Mcast, and ANT Baseline group together in the figure with low reliability.

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, *i.e.*, 1% loss. We are confident this calculation

does not invalidate the values seen and used for OpenDDS IP Mcast as the values for ANT's version of IP Mcast, *i.e.*, ANT Baseline, produces similar results.

Figure 10 shows the erratic behavior of RMcast. At times RMcast received all updates and other times it received all updates only up to a certain number and then received no additional updates. The cause of this problem was not explained by the RMcast developers. We therefore removed RMcast from further consideration.

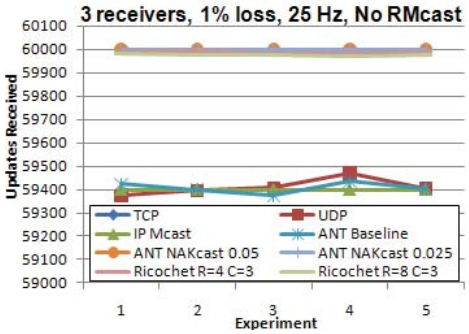

Fig. 9. Emulab: 3 readers, 1% loss, 25Hz, no RMcast

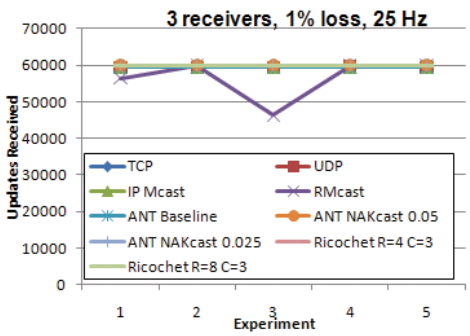

Fig. 10. Emulab: 3 readers, 1% loss, 25Hz

Figure 11 highlights the latency overhead incurred by TCP. This latency is due to TCP's use of positive acknowledgments. Moreover, TCP's latency overhead increases as the amount of loss increases. All other protocols are fairly comparable with respect to latency for this environment configuration.

Figure 12 shows the ReLate2 values for all the protocols considered. We see that using ReLate2 splits the protocols that support both reliability and low latency from those that do not. The separation of the protocols using ReLate2 is more pronounced with higher levels of network loss and number of receivers.

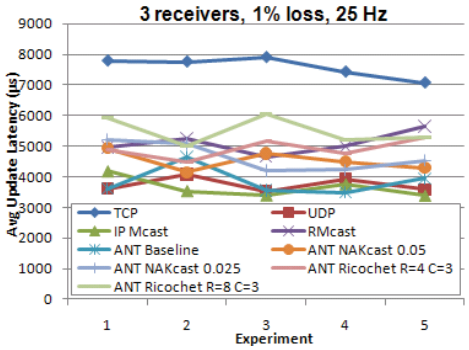

Fig. 11. Emulab: 3 readers, 1% loss, 25Hz

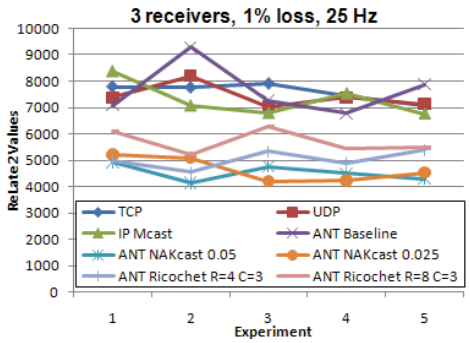

Fig. 12. Emulab: 3 readers, 1% loss, 25Hz

We now analyze the results of the Emulab experiments, which involved all the transport protocols presented in Section 3.1. We utilize the ReLate2 metrics defined in Section 3.2 to evaluate the results from the initial Emulab experiments.

The results show that ANT NAKcast and ANT Ricochet always produced the lowest ReLate2 values even for multiple configurations of the protocols, *i.e.*, NAKcast timeout values of 0.05 and 0.025 and Ricochet R values of 4 and 8. The protocols that support reliability but unbounded latency and the protocols that support low latency but no reliability are clearly separated from the protocols that support both low latency and reliability.

Moreover, the ReLate2 value is equal to the average latency when there is no loss, as is the case for TCP and the majority of cases for NAKcast. When NAKcast does not receive all updates, it is only missing some of the very last updates which could not be detected since no packets were received after them. The data and figures show that the ReLate2 metric is useful for evaluating protocols that balance reliability and latency.

The NAKcast and Ricochet Experiments. Our next set of experiments focused on the protocols that are best suited for balancing reliability and latency based on the ReLate2 metric (*i.e.*, ANT NAKcast 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. We note that if RMcast's behavior would stabilize it would also be a protocol worth evaluating for reliability and low latency.

In particular, for comparison we focused on specific configurations of NAKcast and Ricochet, *i.e.*, NAKcast with a timeout period of 0.05 seconds and Ricochet with an R value of 4. We constrained the protocols in this way because configured correctly either protocol can generally provide lower ReLate2 values than the other. However, we are interested in a relative comparison of the protocols themselves rather than reconfigurations that can make the one protocol outperform the other for a particular environment.

As noted in Section 4.1, we used the ISISlab testbed for experiments involving only ANT NAKcast 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 ANT source code is available and thus we did not require Emulab's network traffic shaping capability.

As with the Emulab experiments in Section 4.2, 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 NAKcast 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.

Moreover, our results and analysis are focused on environment configurations with relatively low (*i.e.*, 1%) and high (*i.e.*, 5%) network loss combined with relatively few (*i.e.*, 3) and many (*i.e.*, 20) receivers. While we ran experiments that ran the spectrum of configurations between these bounds, the particular experiments at these limits are useful for understanding the behavior of the protocols.

We show data collected while using 10Hz and 100Hz sending rates to highlight the behavorial distinctions of the protocols.

Figures 13 and 14 show that for a low number of receivers (*i.e.*, 3), a low loss percentage (*i.e.*, 1%), and low sending rate (*i.e.*, 10Hz), NAKcast, in general, has lower ReLate2 values. In fact, NAKcast 0.05 provided the lowest ReLate2 values for all of the ISISlab protocol configurations tried, i.e., NAKcast with timeout values of 0.05 and 0.025 seconds and Ricochet with R values of 4 and 8. Ricochet provided lower average update latency as the sending rate increases. We discuss this observation in more detail at the end of this section. The number of updates received remains constant across various sending rates for both protocols and we do not include those figures here.

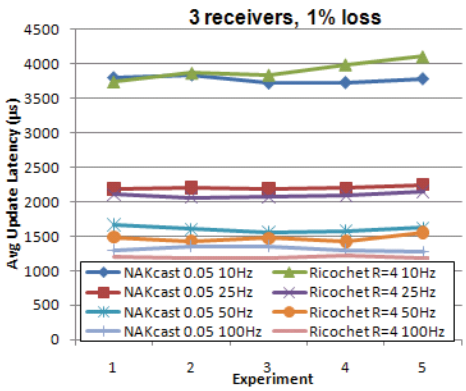

Fig. 13. ISISlab: 3 readers, 1% loss

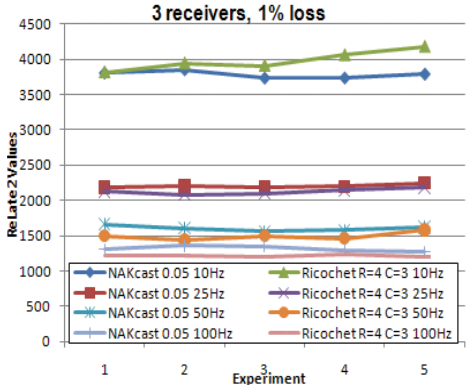

Fig. 14. ISISlab: 3 readers, 1% loss

Figures 13 and 14 also show the reliability of Ricochet at low loss rates. This reliability can be seen by comparing the figures and noticing that the graphs appear very similar. This similarity points out that Ricochet is almost as reliable as NAKcast with reliability rates ranging from 99.97% to 99.99%. This reliability is fairly constant across the different sending rates.

Figures 15 and 16 show the effect on the protocols of increasing packet loss. In this environment configuration we have changed the network loss from 1% to 5%. We see that NAKcast performed best not only for a sending rate of 10 Hz as was the case for 1% loss but also for 25 Hz. Ricochet still provided the best ReLate2 values for sending rates of 50 Hz and 100 Hz. Moreover, while Ricochet average update latency improved over NAKcast the ReLate2 values don't reflect this as Ricochet only had better ReLate2 values for sending rates of 50 and 100 Hz. This is due to Ricochet's reliability ranging from 99.42% to 99.56% which has decreased from the experiments with 1% loss.

Figures 17 and 18 show the effect on the protocols of increasing the number of receivers. In this environment configuration we increased the number of receivers from 3 to 20. We see that now Ricochet and NAKcast performed equally well at 10 Hz where NAKcast always performed best at that rate with only 3 receivers. Ricochet provided the best ReLate2 values for the other sending rates. Moreover, Ricochet's reliability is almost as high as with only 3 receivers ranging from 99.94% to 99.96% of updates received.

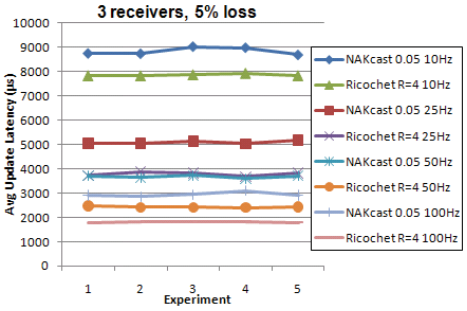

Fig. 15. ISISlab: 3 readers, 5% loss

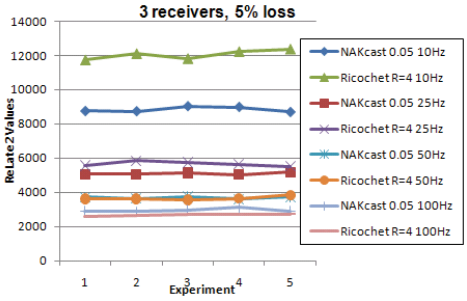

Fig. 16. ISISlab: 3 readers, 5% loss

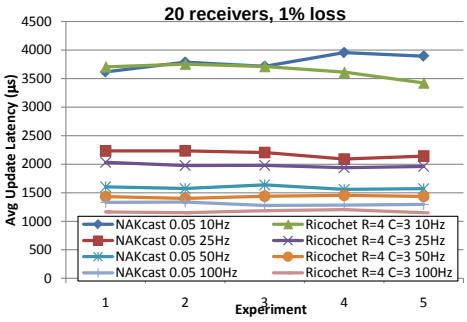

Fig. 17. ISISlab: 20 readers, 1% loss

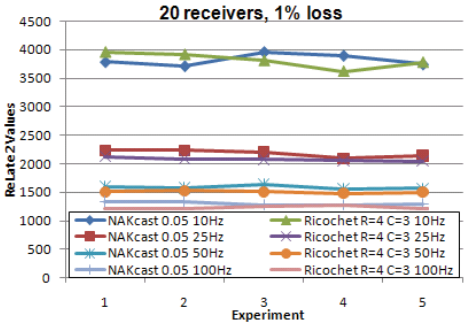

Fig. 18. ISISlab: 20 readers, 1% loss

Finally, Figures 19 and 20 show the effect on the protocols of increasing the number of receivers and loss rate. In this environment configuration we had 20 receivers and 5% network loss.

We see that while Ricochet had a noticeable improvement in average update latency compared to NAKcast, NAKcast offset this discrepancy with its higher reliability. For higher rates, *i.e.*, 25, 50, and 100 Hz, the ReLate2 values for Ricochet and NAKcast are comparable. NAKcast always provided the lowest ReLate2 values for 10 and 25 Hz while Ricochet always provided the lowest ReLate2 values for 50 and 100 Hz. Moreover, Ricochet's reliability is in the same range as for 3 receivers with 5% loss ranging from 99.46% to 99.55% of updates received.

The results above show that for a set protocol configuration there are performance trade-offs between NAK-based and LEC protocols. In general, NAK-based protocols performed better with a lower network loss percentage, lower sending rates, and few receivers. In this environment configuration there is no concern for NAK storms where receivers flood the sender with requests for retransmissions. Moreover, NAK-based protocols only needed to receive one update that is out of sequence to determine loss whereas LEC protocols need to receive R updates before error detection and correction information is sent among the receivers. NAK-based protocols also delivered consistently high reliability, at the cost of higher latency for higher sending rates.

LEC protocols, however, provided better performance when network loss was higher and sending rates increased. LEC protocols did not incur increasingly more network usage as network loss and number of receivers increased. LEC protocols

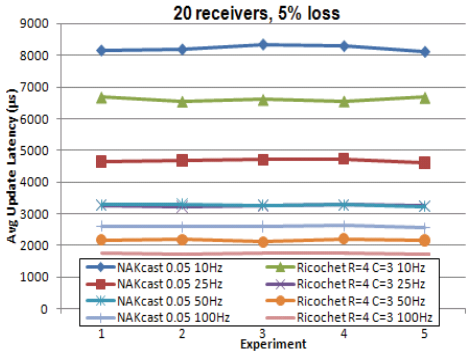

Fig. 19. ISISlab: 20 readers, 5% loss

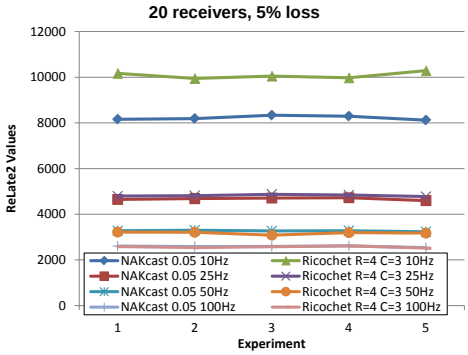

Fig. 20. ISISlab: 20 readers, 5% loss

scaled well in the number of receivers and in network loss. LEC protocols also generally provided lower latency at the cost of small decreases in reliability.

NAKcast 0.05 provided the lowest ReLate2 values and lowest average latency for 3 receivers, 1% loss, and 10 Hz sending rate. The data make sense since the sending rate was less than the timeout period and the loss rate and number of receivers were low. If the network drops a packet the packet is as likely to be discovered in the same amount of time by NAKcast with a timeout of 0.05 as it is with a higher timeout. The sending rate is so low that increasing the NAKcast timeout to 0.025 seconds provided no benefit and indeed added overhead as timeouts are generated and checked more frequently.

5 Related Work

This section compares our work on FLEXMAT with related R&D efforts.

Performance evaluation of network transport protocols. Much prior work has evaluated network transport protocols, *e.g.*, Balakrishnan *et al.* [12] evaluate the performance of the Ricochet transport protocol with the Scalable Reliable Multicast (SRM) protocol [17]. Bateman *et al.* [13] compare the performance of TCP variations both using simulations and in a testbed. Cheng *et al.* [19] provide performance comparisons of UDP and TCP for video streaming in multihop wireless mesh networks. Kirschberg *et al.* [9] propose the Reliable Congestion Controlled Multicast Protocol (RCCMP) and provide simulation results for its performance. In contrast to our work on FLEXMAT, 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.* [15] 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 FLEXMAT, however, that prior work did not include performance evaluations of DDS with various transport protocols.

Sachs *et al.* [10] present a performance evaluation of message-oriented middleware (MOM) in the context of the SPECjms2007 standard benchmark for MOM servers. The benchmark is based on the Java Message Service (JMS). In particular, the work details performance evaluations of the BEA WebLogic server under various loads and configurations. In contrast to our work on FLEXMAT, however,

that work did not integrate various transport protocols with the middleware to evaluate its performance.

Tanaka *et al.* [20] 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 FLEXMAT 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 FLEXMAT measures the impact of various network protocols integrated with QoS-enabled pub/sub middleware.

Performance evaluation of embedded middleware. Bellavista *et al.* [16] 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 FLEXMAT 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 FLEXMAT focuses on how properties of various transport protocols can be used to maintain specified QoS.

6 Concluding Remarks

Developers of RT-ESP systems face a number of challenges when developing their applications for dynamic environments. To address these challenges, we have developed FLEXMAT to integrate and enhance QoS-enabled pub/sub middleware with flexible transport protocols to support RT-ESP applications. This paper defines the ReLate2 metric to empirically measure the reliability and latency of FLEXMAT as a first step to having QoS-enabled pub/sub middleware autonomically adapt transport protocols as the changing environment dictates.

The latest information and source-code for FLEXMAT and related research can be obtained at `www.dre.vanderbilt.edu/~jhoffert/ADAMANT`.

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. D. Schmidt *et al. Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, Volume 2.* Wiley & Sons, New York, 2000.
7. E. Gamma *et al. Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, Reading, MA, 1995.
8. G. Eisenhauer *et al.* Publish-subscribe for high-performance computing. *Internet Computing, IEEE*, 10(1):40–47, Jan.-Feb. 2006.
9. J. Kirschberg *et al.* Rccmp: reliable congestion controlled multicast protocol. In *1st EuroNGI COnference on Next Generation Internet Networks Traffic Engineering*, april 2005.
10. K. Sachs *et al.* Performance Evaluation of Message-oriented Middleware using the SPECjms2007 Benchmark. *Performance Evaluation*, 2009. to appear.
11. M. Balakrishnan *et al.* Slingshot: Time-critical multicast for clustered applications. In *Proceedings of the IEEE Conference on Network Computing and Applications*, 2005.
12. 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.
13. 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.
14. 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.
15. 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.
16. 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.
17. 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.
18. 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.
19. 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.
20. 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.