# Supporting Large-scale Continuous Stream Datacenters via Pub/Sub Middleware and Adaptive Transport Protocols [*]

Joe Hoffert, Douglas Schmidt
Vanderbilt University, EECS Department
Nashville, TN
{jhoffert, schmidt}@dre.vanderbilt.edu

Mahesh Balakrishnan, Ken Birman
Cornell University, CS Department
Ithaca, NY
{mahesh, ken}@cs.cornell.edu

## Abstract

*Large-scale datacenters that handle continuous data streams require scalable and flexible communication infrastructure. The scalability of publish/subscribe (pub/sub) middleware coupled with fine-grained quality-of-service (QoS) support and adaptive transport protocols constitutes a promising area of research to address the challenges of these types of large-scale datacenters. This paper describes how we are integrating pub/sub middleware with an adaptive transport protocol framework to support composable functionality for properties that—coupled with the fine-grained QoS middlware support—can meet the required QoS of data conferencing applications that coordinate and manage multiple continuous data streams.*

## 1 Introduction

Modern datacenters are an important computing platform for large-scale applications in many domains, such as homeland security, online stock trading, humanitarian relief missions, and weather monitoring. The installed base of datacenter-class inexpensive commodity servers has increased dramatically in recent years, whereas the number of relatively expensive mainframe servers has noticeably declined. Modern datacenters are service-oriented and heavily virtualized, running heterogeneous systems and software.

Today's datacenters must also increasingly support applications with *real-time* QoS requirements. These real-time datacenters process events and recover from failures within seconds. Commodity datacenters are turbulent environments comprised of low-cost components and eliciting many failure modes, ranging in scope from the loss of packets to node crashes to large-scale facility, regional, and national system failures.

Real-time datacenters require scalable and flexible communication infrastructure. The requirement for scalability inherently comes from the datacenter being large-scale and needing to support numerous types of data for a plethora of data providers and consumers. The requirement for flexibility manifests itself in several ways, including the following:

- Large-scale datacenters need flexible communication infrastructure due to the many failure modes and complexity inherent in the scale involved. Flexible communication infrastructure can adapt to fluctuating demands and environment changes to maintain acceptable levels of service.
- Certain types of large-scale datacenters exacerbate the need for flexible communication infrastructure due to their dynamic and *ad hoc* nature. Examples of *ad hoc* large-scale datacenters include tactical information grids within a battlefield or emergency response networks in the aftermath of a regional or national natural disaster.

Many publish/subscribe standards and technologies (*e.g.*, Web Services Brokered Notification [9], the Java Message Service [12], and the CORBA Event Service [6]) have been developed to support large-scale data-centric distributed systems. These standards and technologies, however, do not provide fine-grained and robust quality of service (QoS) support. Some large-scale distributed systems (*e.g.*, the Global Information Grid and Network-centric Enterprise Services) require rapid response, scalability, bandwidth guarantees, fault-tolerance, and reliability. These systems also need to function under stressful conditions and over connections with less than ideal properties, such as bursty loss, latency concerns, and route flaps.

In addition, real-time datacenters often need to support *continuous data streams* that constantly generate data. Continuous data streams can be generated from sensors (*e.g.*, surveillance cameras, temperature probes) as well as other types of monitors (*e.g.* online stock trade feeds). These streams differ from streamed file data (*e.g.*, streaming the contents of a movie) since the start and end of streamed file data are known *a priori*. Streamed file data generally have less stringent deadline and delivery requirements, instead focusing on presenting a consistent flow of data to an application. For example, distributed movie players can buffer video to avoid pauses or skips.

[*]This work is supported in part by the NSF TRUST and AFRL Castor and Pollux projects.

We call applications that synchronize multiple continuous data streams *data conferencing* applications. These types of applications have requirements for (1) *timeliness*, which is inherent in continuous data streams and (2) *reliability*, which involves receiving enough data so that they are usable. Moreover, conferencing applications imply multiple senders and receivers since more than one continuous data stream is sent and potentially many receivers receive the data streams. A challenge for supporting data conferencing applications is to develop technologies that are compatible with commercial-off-the-shelf (COTS) middleware technologies and yet can also achieve the required level real-time QoS.

To address this challenge we are combining QoS-enabled middleware with adaptive transport protocols to provide timely and reliable data delivery. In particular, we are developing the *ADAptive Middleware And Network Transports* (ADAMANT) platform, which integrates the Ricochet++ transport protocol framework with OpenDDS [5].

Ricochet++ is a framework for composing transport protocols built upon the properties provided by the Ricochet transport protocol [1], a scalable reliable multicast protocol that was recently developed by Cornell. OpenDDS is an open-source implementation of the Data Distribution Service (DDS) [7] middleware that enables applications to communicate by publishing information they have and subscribing to information they need in a timely manner. By integrating Ricochet++ and OpenDDS, ADAMANT provides fine-grained QoS control and a powerful standardized pub/sub atop a multicast protocol that is more scalable and efficient than the standard DDS Real-Time Publish/Subscribe (RTPS) [8] protocol.

The remainder of this paper is structured as follows: Section 2 presents a motivating example of a representative data conferencing application; Section 3 details our solution approach for ADAMANT; and Section 4 presents concluding remarks.

## 2 Motivating Example: A Data Conferencing Application for Search-and-Rescue Missions

To motivate ADAMANT, this section describes a data conferencing application that detects and locates survivors as part of disaster relief efforts. The application needs to coordinate and synchronize a video data stream from one platform (*e.g.*, a videocamera mounted atop a building) and a thermal scan data stream from another platform (*e.g.*, a thermal imaging camera attached to an unmanned aerial vehicle). Not only are there multiple senders of continuous data, but there are also potentially many receivers of the data, *e.g.*, a helicopter performing rescue operations, local disaster relief headquarters, and Federal Emergency Management Agency (FEMA) headquarters.

The senders and receivers of data for a particular search-and-rescue mission must compete for infrastructure support with other search-and-rescue missions and with other tasks. These tasks occur in a coordinated and concurrent manner as part of the disaster relief efforts. The coordination and concurrency emphasizes the need for scalability. The timeliness and reliability requirements for processing continuous data streams emphasizes the need for flexible and fine-grained QoS support.

Other data conferencing applications with similar data stream requirements include tracking the prices of multiple stocks within a stock trading system, synchronizing medical telemetry data for a wireless medical emergency room within a combat support hospital used during humanitarian missions, and monitoring national or global weather via data streams from multiple sensors and applications.

## 3 Solution Approach: ADAMANT

This section describes how ADAMANT integrates OpenDDS with the latest version of the Ricochet++ transport protocol framework. In addition, it provides overviews of Ricochet++, the DDS standard, and the OpenDDS open-source DDS implementation. By combining DDS's extensive QoS policies and standardized API with Ricochet++'s scalable multicast transport protocol, ADAMANT provides a powerful and flexible development platform for pub/sub-based data conferencing applications.

### 3.1 Overview of the Ricochet++ Transport Protocol Framework

The Ricochet++ transport protocol framework originally started as the Ricochet transport protocol developed at Cornell University. 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.

Ricochet++ 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.

Architectural enhancements have since been made to Ricochet to define the Ricochet++ composable transport protocol framework. The framework provides low-level transport protocol properties (*e.g.*, NAK-based vs. ACK-based, sender-side forward error correction vs. receiver-side forward error correction) that can be selected and combined to meet the QoS needs of a particular application.

## 3.2 Overview of the Data Distribution Service

DDS is an Object Management Group (OMG) specification that defines a standard architecture for exchanging data in distributed pub/sub systems. DDS supports a logical global data store in which publishers and subscribers write and read data, respectively. Moreover, 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. Examples of these platform-specific models include C++ running on VxWorks or Java running on Real-time Linux.

| **DDS QoS Policy** | **Description** |
|---|---|
| Durability | Determines if data outlives the time when written or read |
| Deadline | Determines rate at which periodic data is refreshed |
| Latency Budget | Sets guidelines for acceptable end-to-end delays |
| Liveliness | Sets liveness properties of topics, data readers, data writers |
| Time Based Filter | Mediates exchanges between slow consumers and fast producers |
| Reliability | Controls reliability of data transmission |
| Transport Priority | Sets priority of data transport |
| Lifespan | Sets time bound for “stale” data |

**Table 1. ADAMANT DDS QoS Policies**

In addition, DDS provides a rich set of QoS policies to provide fine-grain control. DDS provides 22 QoS policies with most QoS policies having attributes with a large number of possible values, *e.g.*, an attribute of type long or character string to support even finer-grained control. Table 1 summarizes the DDS QoS policies most relevant to ADAMANT.

## 3.3 Overview of OpenDDS

OpenDDS is an open-source implementation of DDS that supports a pluggable transport framework. This framework allows OpenDDS to use custom transport protocols for data transport. We chose OpenDDS for our DDS implementation due to (1) the source code being freely available and (2) support for incorporating customized transport protocols via its pluggable transport framework (PTF). OpenDDS is built on top of the ADAPTIVE Communication Framework (ACE) [10] and shares some functionality (such as an IDL compiler) with The ACE ORB (TAO) [11]. OpenDDS’s PTF uses design patterns (such as Strategy [3] and Service Configurator [4]) to provide flexibility and delegate responsibility to the protocol only when applicable.

For example, a custom protocol subclasses from the *TransportSendStrategy* class to determine how the protocol should send data when a data writer writes out topic data. Likewise, a custom protocol subclasses from the *TransportReceiveStrategy* class to determine how data should be handled once it is received. The Service Configurator pattern allows application developers to specify the transport protocols that should be included in the application. The 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. Currently within the OpenDDS PTF application developers must 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 developers must manually associate the data writer with the publisher using a reliable transport.

A better solution is to leverage the DDS reliability QoS policy to specify reliable communication. The middleware would then automatically select a transport protocol with the desired reliability properties, *e.g.* TCP. Custom transport protocols could register their properties with the framework so that they could be selected appropriately.

## 3.4 ADAMANT

As shown in Figure 1, application-/domain-specific transport protocols can be composed using Ricochet++ and used by OpenDDS via its PTF. For example, application developers can create a custom transport protocol that uses acknowledgments from the receiver, forward error correction (FEC) information from the sender, and XOR encoding for FEC. Alternatively, developers can create a custom transport protocol that uses negative acknowledgments from the receiver, FEC information from receiver to other receivers, and Reed-Solomon encoding for FEC.

Ricochet++ also provides tunable settings within the composable modules. For example, settings for the FEC module include the number of packets sent for error correction ($r$) and the number of packets to be corrected in case of loss ($c$) as typical of FEC protocols. The FEC module can also be tuned by the amount of interleaving ($i$) it provides

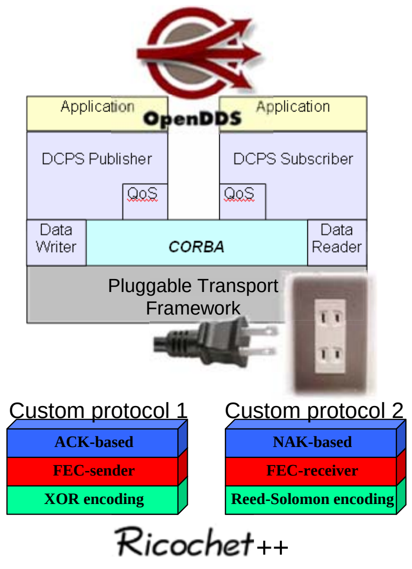

**Figure 1. ADAMANT Overview**

which makes the protocol resilient to burstiness. Settings for the NAK-based reliability module include timeouts and window size for the sender.

## 4 Concluding Remarks

We are currently evaluating the behavior and performance characteristics of various configurations of DDS and its QoS policies with the Ricochet++ transport protocol framework. These configurations will include the standard DDS transport protocol (RTPS) and the protocols composed using Ricochet++. To date, the Ricochet++ framework has been implemented with support for rudimentary composable modules, *e.g.*, IP multicast, NAK-based reliability. Additional modules are being developed to supply the transport protocol properties described in Section 3.1. We have developed and tested the OpenDDS classes necessary to plug Ricochet++ into OpenDDS along with example applications that send and receive text messages, as well as MPEG video data.

We are incorporating ADAMANT into the DDSBench benchmarking environment [2] to expedite the evaluation of various properties such as throughput and latency. We are using Emulab software (`www.emulab.net`) to define network topology and link properties for environments applicable to large-scale datacenters. We are also classifying various ADAMANT configurations based on the observed behavior. This classification will guide the development of autonomic adaptive behavior so that ADAMANT can automatically modify its configuration based on application needs and changes in application environments.

## References

[1] Mahesh Balakrishnan, Ken Birman, Amar Phanishayee, and Stefan Pleisch. Ricochet: Lateral error correction for time-critical multicast. In *NSDI 2007: Fourth Usenix Symposium on Networked Systems Design and Implementation*, Boston, MA, 2007.

[2] C. Esposito, S. Russo, and D. Di Crescenzo. Performance assessment of omg-compliant data distribution middleware. In 22$^{nd}$ *IEEE International Parallel and Distributed Processing Symposium*, Miami, Florida, USA, April 2008.

[3] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

[4] Prashant Jain and Douglas C. Schmidt. Service Configurator: A Pattern for Dynamic Configuration of Services. In *Proceedings of the* 3$^{rd}$ *Conference on Object-Oriented Technologies and Systems*. USENIX, June 1997.

[5] Object Computing Incorporated. OpenDDS. `http://www.opendds.org`, 2007.

[6] Object Management Group. *Event Service Specification Version 1.1*, OMG Document formal/01-03-01 edition, March 2001.

[7] Object Management Group. *Data Distribution Service for Real-time Systems Specification*, 1.2 edition, January 2007.

[8] Object Management Group. *The Real-time Publish-Subscribe Wire Protocol DDS Interoperability Wire Protocol Specification*. Object Management Group, OMG Document formal/2008-04-09 edition, April 2008.

[9] Organization for the Advancement of Structured Information Standards. *Web Services Brokered Notification Version 1.3*, OASIS Document wsn-ws_brokered-_notification-1.3-spec-os edition, October 2006.

[10] Douglas C. Schmidt and Stephen D. Huston. *C++ Network Programming, Volume 2: Systematic Reuse with ACE and Frameworks*. Addison-Wesley, Reading, Massachusetts, 2002.

[11] Douglas C. Schmidt, Bala Natarajan, Aniruddha Gokhale, Nanbor Wang, and Christopher Gill. TAO: A Pattern-Oriented Object Request Broker for Distributed Real-time and Embedded Systems. *IEEE Distributed Systems Online*, 3(2), February 2002.

[12] SUN. Java Messaging Service Specification. `java.sun.com/products/jms/`, 2002.