

Benchmarking Publish/Subscribe Middleware for Radar Applications

Andrew S. Rhoades, Glenn Schrader, Paul Poulin
{rhoades, gschrad, ppoulin}@ll.mit.edu, May 2007
MIT Lincoln Laboratory, Lexington, MA 02420

Introduction

At MIT Lincoln Laboratory (MIT/LL), the need for portability, upgradeability, and expandability in future radar systems is leading to modular system design. In turn, the modularity leads to the use of middleware solutions for communications. However, the communication flows within a system may have requirements that cannot be adequately met with a single middleware solution, for reasons such as needed features, platform availability, or interface semantics. The communications requirements of these systems may include high message rates, high message bandwidths, and "hard" real-time constraints, meaning messages are guaranteed to be received within an allotted time.

MIT/LL is developing a communications middleware layer with publish/subscribe [1] semantics for use in these radar applications. It is built on top of other communications middlewares and provides a consistent communications abstraction to software engineers while allowing system engineers to use different underlying middlewares as needed.

This middleware layer is expected to be part of the core infrastructure for future MIT/LL radar systems, carrying command, status, and data messages among subcomponents of the systems. To test and verify the suitability of middleware for communications in such systems, a benchmarking application has been developed. The middleware, benchmarking application, and some performance results are described below.

Communications Middleware

The communications middleware that MIT/LL is developing is a thin abstraction layer that provides a simple C++ application programming interface (API) built on top of other communications middlewares. This layer is "thin" because it seeks to minimize any performance impact by avoiding additional memory copying (zero-copy) and by not adding features to the middlewares it sits on top of. It is known as RTCL, for Radar Thin Communications Layer.

The idea is to create a single API that supports our application domain but isolates application components from specific middleware details, both in the coding and in the building process. Software written to this API is not tied to any single communications middleware, so that choosing the underlying middleware(s) is a system-engineering task, not a software-engineering task. Multiple underlying middlewares may be also used together in one system. How the

communications of various components of a system are plumbed together is also done as part of the system engineering via run-time configuration files, rather than via compile-time code changes.

The publish/subscribe paradigm provides location independence for the software code of system components, whose location is a system-engineering issue; they can be on the same machine or different machines, on the same OS or a mixed set of OSes, with no change to the software. The RTCL publish/subscribe semantics are of a Data Distribution Service (DDS) [2] flavor and include Quality of Service (QoS) concepts.

Initial development of RTCL has focused on two middleware options underneath: the commercial publish/subscribe middleware DDS from Real Time Innovations and a home-grown shared-memory middleware. Development operating systems are Linux, Solaris, and VxWorks.

Benchmark Test Cases

Figure 1 below lists four fundamental cases representing communication patterns relevant to our application.





1-to-n: single sender, multiple receivers, single topic	
n-to-1: single receiver, multiple senders, separate topics	
m-to-n: multiple receivers, multiple senders	
pipeline: messages sent from task to task	

Figure 1: Fundamental communication patterns.

The first two cases are actually special cases of the more general third case, but they are listed separately because they commonly occur in our application.

These fundamental patterns are tested over the following parameters: message size, message rate, and number of benchmarking application instances. For initial experiments, parameter ranges of interest were message sizes up to tens of kilobytes, at rates up to several kHz, using up to a dozen instances on up to eight computers.

The primary measured item is the time from the start of transmission to the end of reception. Though we call it "latency", this is really the latency plus the time to transmit the message content, also known as transmission delay. For our purpose this is sufficient and also more relevant to the applications.

This work is sponsored by the Department of the Air Force under Air Force contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the authors and are not necessarily endorsed by the United States Government.

Benchmarking Application

An application has been developed for benchmarking, as well as verifying, the RTCL middleware, known as ICA for Instrumented Communications App. Test cases involve running two or more instances of ICA on one or more computers according to a configuration described in a test configuration file. Individual process instances of ICA have one or more publishers and subscribers. Publishers send messages of a specified size at a specified rate on a specified topic. Subscribers receive messages on a specified topic and may republish them on another.

Measurements of the time to send a message are done using timestamps in the message and can be either one-way, where the subscriber computes the time, or two-way where a round-trip time is computed by the original publisher upon receiving a republished copy of the original message (the time to copy the contents is not included in the measurement). When a publisher and subscriber are on different computers, special timing hardware is used for one-way timing. ICA processes write histograms of the measurements to files at the conclusion of test runs.

Results

The initial test platform was Linux, admittedly not a real-time operating system (RTOS) but nonetheless a suitable starting point for "soft" real-time experiments and software development. For real-time measurements, the middleware and test application are built on other Unix-like OSes with real-time capabilities, such as Solaris (or real-time Linux variants), or on dedicated RTOSes such as VxWorks; results from these will be shown in the presentation.

An example of Linux results is shown in the graphs, which summarize a set of tests of the 1-to-n case, over different numbers of subscriber instances (x-axis) and using several message sizes (colored curves). Each instance was on a separate Linux server, connected via gigabit Ethernet. The underlying middleware was RTI DDS, using reliable delivery, multicast for the primary publisher, and unicast for the return publishers needed for round-trip timing (so this really is the primary 1-to-n case plus an n-to-1 case created by the responses). The test was run at 200 Hz, and the measured latencies are shown on the y-axis in microseconds.

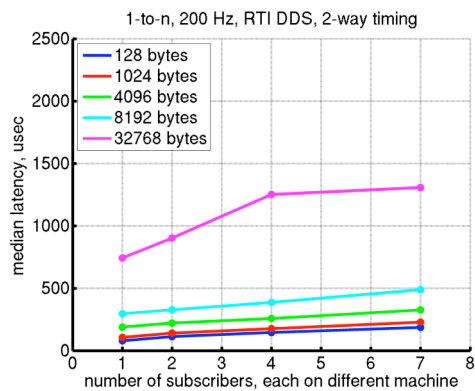


Figure 2: Median measured latencies.

Note that 200 Hz corresponds to 5000 microseconds between published messages. Figure 2 shows median latencies and Figure 3 shows the maximum measured latencies.

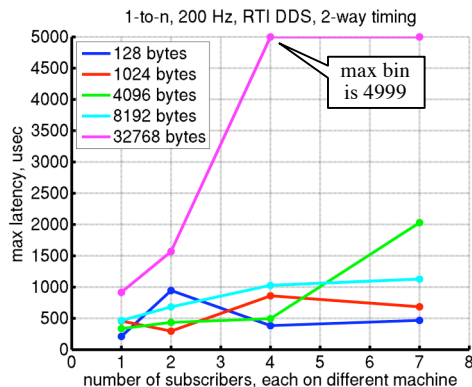


Figure 3: Maximum measured latencies.

Looking at Figure 4, which shows the histogram of all the latencies from the case with 32-kilobyte messages and four subscribers, we see that the majority of messages are clustered around the median but that there are also a significant number of outliers (see detail). Note that the histogram's maximum bin includes measurements greater than 4999.

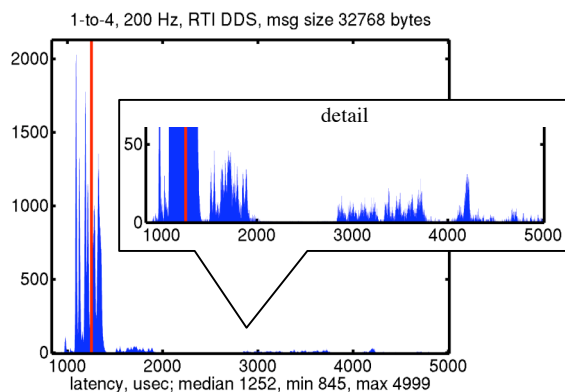


Figure 4: Latency histogram for 1-to-4 32-kilobyte messages.

The ICA can be tested in configurations that are representative of particular systems to assist system design. The example results shown here could be analyzed to assess the suitability of RTCL using DDS on a Linux cluster, for instance. From these results it could be concluded that using smaller messages or smaller numbers of subscribers is supported adequately, though the relatively high maximum latency observed in the 1-to-2 case with 128 byte messages suggests variability that might warrant further investigation. However, if larger messages were needed, this shows that 32-kilobyte messages to four or more subscribers will sometimes fall behind in this configuration, so further testing with other configurations would be needed to find an adequate solution in that case.

References

- [1] P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec, "The Many Faces of Publish/Subscribe", *ACM Computing Surveys*, Vol. 32, No. 2, June 2003, pp. 114-131.
- [2] "Data Distribution Service for Real-time Systems", specification from Object Management Group (omg.org).