ORTE - open source implementation of Real-Time Publish-Subscribe protocol

Petr Smolik, Zdenek Sebek, Zdenek Hanzalek
Czech Technical University
Faculty of Electrical Engineering, Dept. of Control
Technicka 2, Praha 6, CZ-16000
xsmolik,sebek,hanzalek@fel.cvut.cz

Abstract

The Ocera Real-Time Ethernet (ORTE) is open source implementation of Real-Time Publish-Subscribe (RTPS) communication protocol developed under Linux. RTPS is an application layer protocol targeted to real-time communication area, which is build on the top of standard UDP stack.

1. Introduction

The communication protocols (such as IP, TCP, UDP, HTTP, FTP or DCOM) are very well suited for standard Internet applications but none of them is suited for real-time distributed applications, imposing deadlines on the data delivery. Ethernet in general is not deterministic due its CSMA/CD MAC method and its behavior under transient overload is not sufficient for any real-time application. On the other hand, when the applications have predictable and bounded number of requests, behavior of Ethernet is "nearly" real-time (very low probability of delayed data delivery) [2] due to the reasonably low number of accesses compared to the high performance of this communication media [3]. Therefore Real-Time Publish-Subscribe (RTPS) [1] compliant middleware is used to satisfy different deadline requirements of applications on one side and to determine a number of accesses on the other side. This is quite natural since distributed control applications have big portion of synchronous requests (usually related to periodic tasks) and small portion of asynchronous requests (usually related to asynchronous events).

Since there are many TCP/IP stack implementations under many operating systems and RTPS protocol does not have any other special HW/SW requirements, it should be easily ported to many HW/SW target platforms. Because it uses only UDP protocol, it retains control of timing and reliability.

2. Real-Time Publish-Subscribe protocol

Real-time applications require more functionality than the one provided by traditional publish-subscribe semantics. Real-Time Publish-Subscribe protocol (RTPS) adds publication and subscription timing parameters and properties so that the application developer can control different types of data flows and therefore the application's performance and reliability goals can be achieved – see fig. 1, 2.

Publication parameters: topic & type - identifies a specific publication, strength - relative weight of publication compared to the publications of the same topic and type, persistence - specifies how long time an issue is valid.

Subscription parameters: topic & type - identifies a specific publication, minimum separation - period during which no new issue is accepted, deadline - specifies how long time a new issue is expected.

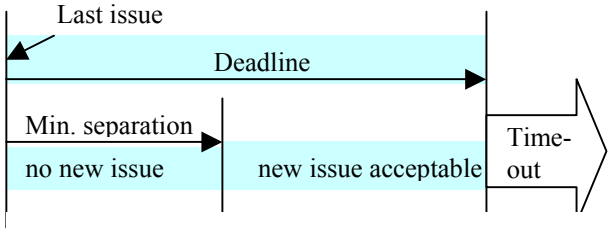

Figure 1. Subscription parameters.

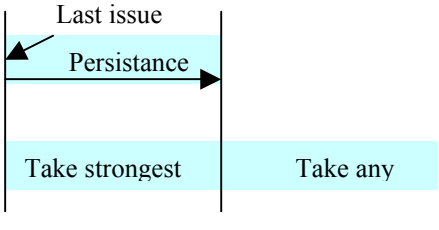

Figure 2. Publisher parameters

2.1. RTPS Architecture overview

The RTPS protocol is implemented as a set of objects. Objects are of the following types:

- Manager (M)
- ManagedApplication (MA)
- Writers (Publication, CSTWriter)
- Readers (Subscription, CSTReader)

The Manager is an independent process, which is created during application startup. It is a special

Application that helps applications to automatically discover each other on the Network. Every Manager keeps track of its managees and their attributes. To provide this information on the Network, every Manager has the special CSTWriter writerApplications. The Composite State (CS) provided by the CSTWriter writerApplications are the attributes of all the ManagedApplications the Manager manages (its managees). Whenever the Manager accepts a new ManagedApplication as its managee, whenever the Manager loses a ManagedApplication as a managee or whenever an attribute of a managee changes, the CS of the writerApplications changes. Each such change creates new instance of CSChange, which has to be transferred to all network objects (Managers and ManagedApplications) by means of CST protocol.

The Publication is used to publish issues to matching Subscription. The CSTWriter and CSTReader are the equivalent of the Publication and Subscription, respectively, but are used solely for the state-synchronization protocol.

A ManagedApplication is an Application that is managed by one or more Managers. Every ManagedApplication is managed by at least one Manager. TheManagedApplication has a special CSTWriter writerApplicationSelf. The Composite State of the ManagedApplication's writerApplicationSelf object contains only one NetworkObject - the application itself. The writerApplicationSelf of the ManagedApplication must be configured to announce its presence repeatedly and does not request nor expect acknowledgements. A Manager that discovers a new ManagedApplication through its readerApplications must decide whether it must manage this ManagedApplication or not. For this purpose, the attribute managerKeyList of the Application is used. If one of the ManagedApplication's keys (in the attribute managerKeyList) is equal to one of the Manager's keys, the Manager accepts the Application as a managee. If none of the keys are equal, the managed application is ignored. At the end of this process all Managers have discovered their managees and the ManagedApplications know all Managers in the Network.

The ManagedApplications now use the CST Protocol between the writerApplications of the Managers and the readerApplications of the ManagedApplications in order to discover other ManagedApplications in the Network. Every ManagedApplication has two special CSTWriters, writerPublications and writerSubscriptions, and two special CSTReaders, readerPublications and readerSubscriptions.

Once ManagedApplications have discovered each other, they use the standard CST protocol through these special CSTReaders and CSTWriter to transfer the attributes of all Publications and Subscriptions in the Network. The managedApplication is composed from seven kinds of objects.

Example of communication between two nodes (N1, N2) is shown on fig. 3. There are two applications running on each node - MA1.1, MA1.2 on node N1 and MA2.1, MA2.2 on node N2. Each node has it own manager (M1, M2).

0. Managers M1 and M2 discovere each other.
1. MA1.1 introduces itself to local manager M1
2. M1 sends list of remote managers Mx and other local applications MA1.x
3. MA1.1 is introduced to all Mx by M1
4. All remote MAs are reported now to M1.1
5. Local MAs are queried for their CS (composite state)
6. All local MAs are sending their CS
7. Remote MAs are queried for their CS
8. All remote MAs are sending their CS

The corresponding publishers and subscribers with matching Topic and Type are connected and their data communication starts.

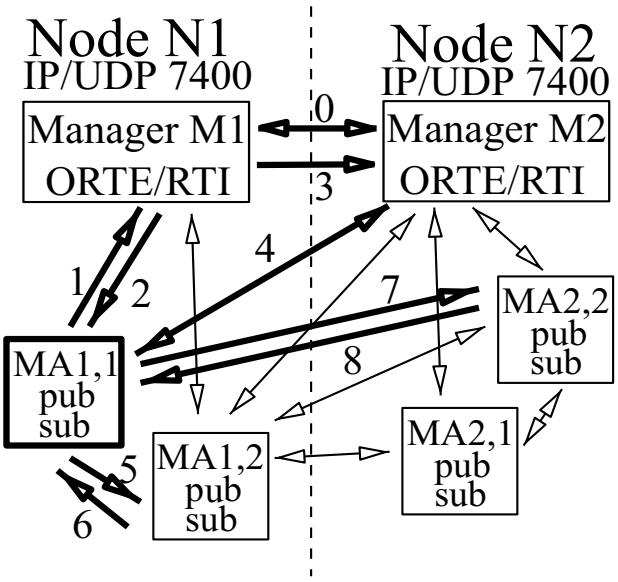

Figure 3. Network communication

2.2. ORTE Implementation

Although object concept of RTPS would be ideal for its implementation using an object-enabled programming language such as C++, we decided to implement ORTE using poor C language because it allows to transfer ORTE into kernel space easily afterwards. Initial implementation has been developed on Linux kernel 2.4, but it should be able to run on both 2.2 and 2.5 branches as well.

Internal structure of ORTE layer is shown of figure 4. There are two main objects in ORTE layer. There is one instance of Manager (M) per each node and one instance of ManagedApplication (MA) per each user's application running on such node. Object Manager is not part of any user application, it is created and

executed by independent program (ORTEManager). User application should never create any instance of object Manager. Manager is designed as a single thread handling both incoming as well as outgoing metatraffic.

ManagedApplication is object which represents user application inside of ORTE layer. It is designed as two threaded process. One thread processes metatraffic (network exploatation) and second thread processes data publication and reception of data issues from another nodes.

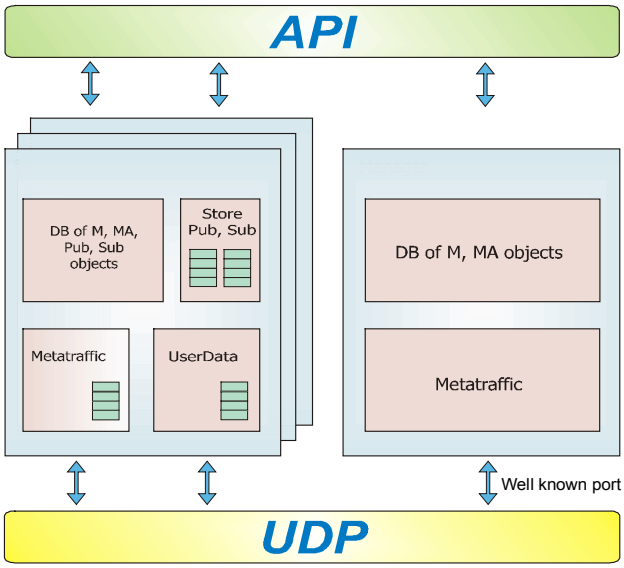

Figure 4. ORTE internals

Whole ORTE is designed as a library, which should be linked with each user application, which require usage of ORTE layer.

2.3. ORTE API

ORTE API consists of three parts:

- Data structures holding parameters and status of network objects
- Set of functions providing means for all node management tasks such as ORTE application and ORTE manager initialization and deinitialization, creation of publication or subscription, its management and deletion.
- Support macros for data type conversion etc.

2.4. Example application

The skeleton of an ORTE application is very simple:

```
#include <orte.h>
int main(int argc, char *argv[])
{
  ManagedApp *app1;
  ORTEAppCreate(&app1);
  /*
    here is your application dependent code
  */
}
```

In order to exchange user data, the application must create the publications of its variables. The application which wants to receive issues from a publication must create corresponding subscription. Properties of publication and subscription contain specification of Topic and TypeName, which specify an application variable within whole network. It is allowed to have more publications of same Topic and TypeName. If it subscribes to such publication, it will receive issues from all publications of the same Topic and TypeName. A publication will be created by function *ORTEAppPublAdd*. Once the publication is created, it is ready to publish data using function *ORTEAppPublSend*.

```
int h_pub;
NtpTime timePersistence;
long strength;
char msg[128];
u_long i=0;
NtpTimeAssembFromMs(timePersistence, 5, 0);
/* this issue is valid for 5 seconds */
strength=1; /* strength of this publication */

h_pub=ORTEAppPublAdd(app1,
                     "HelloWorld", /* Topic */
                     "HelloWorldData", /* TypeName */
                     &timePersistence,
                     strength);
while (1) {
  sprintf(msg,"Hello World count:%li\n",i);
  ORTEAppPublSend(app1,h_pub,msg,strlen(msg)+1);
  ORTESleepMs(1000); /* sleep for 1 second */
  i++;
}
```

Subscribing application needs to create a subscription with publication's Topic and TypeName. A callback function will be then called when new issue from publisher will be received.

```
ManagedApp *app1;
int h_sub;
NtpTime minimumSeparation,deadline;
NtpTimeAssembFromMs(minimumSeparation, 0, 0);
NtpTimeAssembFromMs(deadline, 5, 0);
h_sub=ORTEAppSubsAdd(app1,
                     "HelloWorld", /* Topic */
                     "HelloWorldData", /* TypeName */
                     &minimumSeparation,
                     &deadline,
                     rcvCallBack); /* callback fn. */

while (1) {
  ORTESleepMs(1000);
}
```

The callback function is shown in the following example:

```
void rcvCallBack(ORTERcvInfo *rcvInfo,u_char status)
{
  switch(status) {
    case 0: /* Issue */
         printf("%s",rcvInfo->data);
         break;
    case 1: /* Deadline */
         printf("\ndeadline\n");
         break;
  }
}
```

There must be the Manager process running on each network node. This manager must be started manualy before any other ORTE-enabled application.

3. Test tools

There are several tools already developed or under development. The most important tools are Real-Time Ethernet Analyzer and Object inspector.

3.1. Real-Time Ethernet Analyzer

Real-Time Ethernet Analyzer is not a standalone program. It is a plug-in module which adds support for RTPS protocol into Ethereal network analyzer [6],. Ethereal is a free network protocol analyzer for Unix and Windows. It allows you to examine data from a live network or from a capture file on disk. You can interactively browse the capture data, viewing summary and detail information for each packet. This tool is already available [4].

3.2. Object inspector

Object inspector is a Java application, which will allow to browse whole network, inspect object's parameters. It will also provide means how to edit some object's parameters remotely. Since it will be written in Java, it will be platform independent. This tool is not yet available.

4. Conclusion

Further development of ORTE will be targeted to several areas. First is to achieve full compatibility with current or any new updated version of RTPS protocol specification [1]. Second is to convert ORTE from user space into kernel space including RTLinux support. Third area is to add support for Microsoft Windows NT/2000 platform.

References

[1] Real-Time Publish-Subscribe Wire Protocol Specification, Protocol Version 1.0, Draft Document version 1.17, February 2002, by Real-time innovations http://www.rti.com/products/ndds/literature.html

[2] Stan Schneider, Gerardo Pardo-Castellote, Mark Hamilton, Can Ethernet be Real-Time, by Real-time innovations http://www.rti.com/products/ndds/literature.html

[3] NDDS Performance Paper, by Real-time innovations http://www.rti.com/products/ndds/literature.html

[4] Overall OpenSource project information, http://sourceforge.net/projects/ocera/

[5] The official OCERA home page is at http://www.ocera.org/

[6] The Ethereal home page at http://www.ethereal.com