Distributed Real-Time Applications Now Have a Data Distribution Protocol

Distributed applications can use UDP directly for simple data distribution. When more sophisticated data distribution is required, a wire protocol built on UDP can simplify development and ease communications management.

by Dr. Gerardo-Pardo Castellote and Peter Bolton,
Real-Time Innovations, Inc.

From enterprise extranets to factory-floor control subnets, the TCP/UDP/IP protocol suite has become the communications framework of choice for distributed applications. While the suite's success attests to the generality and power of the protocols, its transport-level protocols are too low level to be used directly for any but the simplest applications. Consequently, higher-level protocols such as HTTP, FTP, DHCP, DCE, RTP, DCOM and CORBA have emerged. Each of these protocols was designed to fill a niche, providing well-tuned functionality for specific purposes or application domains.

The real-time publish-subscribe (RTPS) communications model is a wire protocol designed specifically for data distribution among real-time applications and is implemented "on top of" UDP. The RTPS protocol defines the message format, interpretation and usage scenarios that underlie all messages exchanged by the applications. The protocol has already been adopted by the Interface for Distributed Automation Group (www.ida-group.org), a consortium of industrial automation companies, as the underlying communication mechanism for industrial communication over standard Internet Protocol (IP) networks built using Ethernet. Real-Time Innovations (RTI) is expecting to submit the protocol specification to the Internet Engineering Task Force (IETF) as an informational Request for Comment (RFC) in the first quarter of 2002.

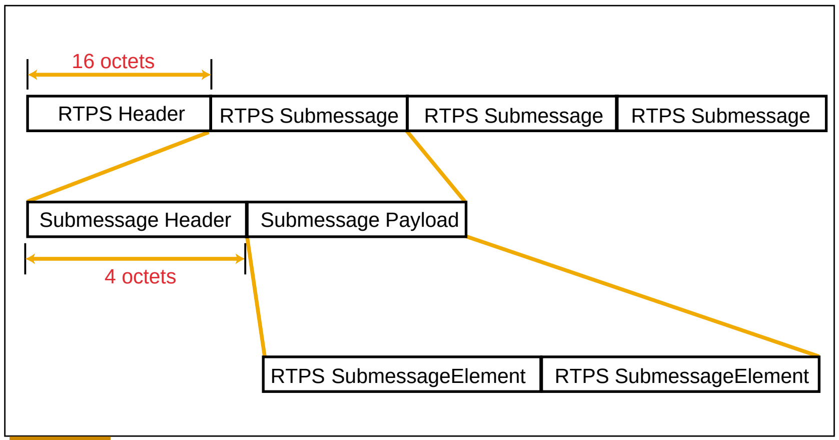

Figure 1 Modular message composition

RTPS Protocol Goals

The RTPS protocol is designed for distributed, data-centric, real-time applications. These applications have three fundamental characteristics:

- Distributed: The system is composed of individual applications running independently of each other on multiple nodes interconnected via a network.
- Data-centric: The system has communication patterns in which some applications are producing information required by other applications. This information can range from simple strings (for example, timestamp and temperature of a boiler) to complex structures composed of static and dynamic variables.
- Real-time: The sending and receiving applications have time constraints that require the network I/O operation to complete within a specific timeframe.

The publish-subscribe (PS) communications model forms the

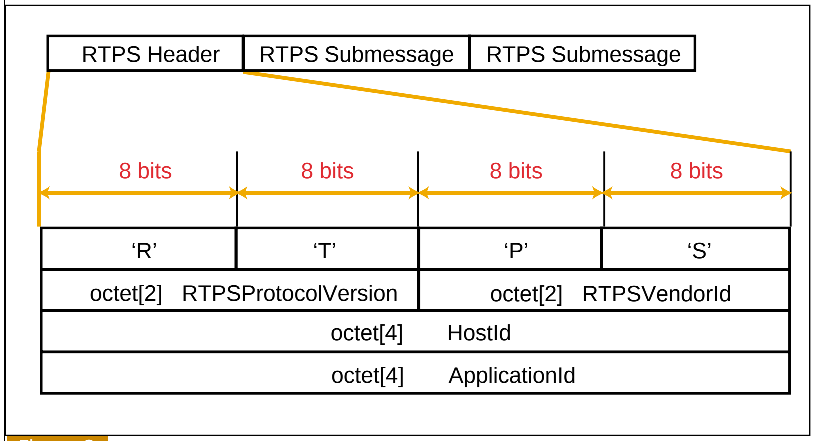

Figure 2 RTPS Header

basis of the protocol. It was selected because of its bandwidth efficiency and simple programming interface. Other goals that drove the specification include:

- Standard IP: Enables high-performance communication over standard IP networks.
- Extensibility: Allows the protocol to be extended and enhanced without breaking backward compatibility and interoperability.
- Configurability: Lets the programmer balance the requirements for reliability and timeliness for each application-to-application communications "channel."
- Real-time: Builds on UDP/IP so that applications don't get blocked by TCP retries.
- Modularity: Allows simple devices to implement a subset and still participate in the network.
- Scalability: Enables systems to scale to potentially very large networks.

In addition to fast, efficient communications, additional goals were set to reduce the complexity of distributed application configuration management:

- Plug-and-play: Allows applications and services to join and leave the network in any sequence, at any time.
- Fault tolerance: Allows the creation of networks without single points of failure.
- Type-safety: Prevents application programming errors from compromising the operation of remote nodes.

Real-Time Publish-Subscribe Primer

Publish-subscribe communications has gained traction in many networked applications in industrial automation, command and control systems, financial communications and web-based "push" technologies. Several features characterize publish-subscribe architectures:

- Named publications: Applications communicate by sending "issues" (samples) of user-defined publications. Publications are identified by a name and type; users do not need to specify computer addresses, routes and port numbers.
- Distinct declaration and delivery: Communications occur in two phases: (1) declaration, in which applications declare their intent to send issues of a specific publication and their interest to receive a publication's issues, and (2) delivery, in which the issues are sent from a publisher to the subscriber(s).
- Event-driven transfer: Communications occur only when there's a new issue. Data transfers are initiated by a publisher.
- One-to-many communications: There is typically more than one subscriber for each publication.

Publish-subscribe has many advantages for real-time applications, especially regarding latency and bandwidth. For example:

- PS is more efficient than distributed objects (for example, DCOM or CORBA) because subscribers do not need to send request messages, and there is no need for polling.
- PS is capable of supporting many-to-many connectivity and group subscriptions. It is thus well suited to dynamic environments with fault tolerance and applications that must scale to many nodes.
- PS maps well to connectionless protocols. This means it can take advantage of direct messaging to circumvent the TCP overhead, and multicast technology to send data to multiple subscribers simultaneously.

However, real-time applications require more functionality than is provided by commercial publish-subscribe semantics. For instance, real-time applications often require:

- Delivery timing control: Real-time applications must know when data is delivered and how long it remains valid.
- Reliability control: Since reliable delivery conflicts with deterministic timing, each time-constrained application needs to specify its particular reliability characteristics (for example, how long it is willing to wait).
- Fault-tolerance: The communications layer should not introduce any single-node points of failure. Moreover, support for "hot standby" or backup data production is often a requirement.
- Selective degradation: Each real-time logical data-channel must be protected from the others. The performance of a channel should not be affected by other channels slowing due to dropouts, network congestion, receiver CPU overload and so on.

RTPS Protocol Overview

The RTPS protocol was developed to provide applications with the anonymous distribution of publication issues, using either reliable or unreliable messaging, to multiple subscribers. In addition, RTPS includes protocols to handle the administrative chores underlying plug-and play and fault-tolerant distributed system configurations. RTPS is implemented using UDP/IP and has its own retry protocol for reliable communications.

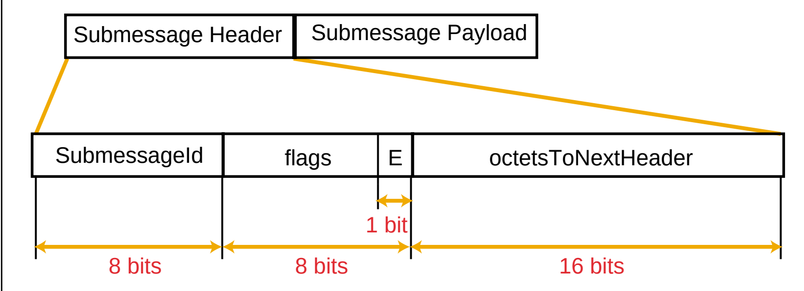

Figure 2 RTPS Submessage header

Any network protocol intended for communicating among heterogeneous computers in an open network must deal with issues such as the representation of information in a machine-independent way, the unique identification of objects and entities and the representation of time and sequence numbers (to disambiguate messages in a sequence). In addition, the broad design paradigms must address goals such as backward compatibility and extensibility. Depending on the intended use of the protocol, different design trade-offs are made.

Applied Standards

The approach taken in the RTPS protocol is to use existing standards whenever possible, so long as the standard does not introduce overhead that would be unacceptable for real-time or embedded applications. These standards then guide the implementation of the design paradigms.

The RTPS message design derives directly from the Internet Engineering Task Force (IETF) IPv6 standard. Like IPv6 all RTPS messages have fixed header sizes. Moreover, RTPS uses the concept of extension headers to add additional "header" information prior to "payloads." In addition, submessage headers contain a header-length field that allows the receiver to quickly navigate the daisy-chain or even to skip submessages.

The dynamic host configuration protocol (DHCP) also influenced the design of the message formats. Like DHCP, RTPS encodes parameters using tuples for key, length and value. This encapsulation allows new parameter keys to be introduced in future versions of the protocol without breaking backward compatibility. It also allows applications to efficiently constrain themselves to the subsets of parameters they care about.

RTPS uses the Object Management Group (OMG) Common Data Representation (CDR) encoding to represent data in a machine-independent way. CDR specifies a set of primitive types with agreed-upon lengths (for example, shorts are 2 octets, doubles are 8 octets). In addition, RTPS uses the standard Network Time Protocol (NTP) representation of time. This representation uses a signed 32-bit integer representing seconds and a second unsigned 32-bit integer representing fractions of seconds in units of $1/2^{32}$ seconds.

Message Formats

The overall structure of an RTPS message is a single header followed by a variable number of submessages. Each submessage is aligned on a 32-bit boundary with respect to the start of the message. The set of RTPS message formats is very rich. All messages can be viewed as composition of a smaller set of well-known submessages, each of which can be individually described and understood. Each submessage is aligned to a 4-octet boundary so that it can be manipulated independently of the previous one (Figure 1). This design is based on the definition and reuse of a set of submessage elements such as sequence numbers, object IDs, bitmaps and time stamps. The reuse of elements simplifies the understanding and construction of processing algorithms.

The entire context required to interpret a submessage must

Submessage Type	Description
ACK	Acknowledges the receipt of all messages up to and including the designated sequence number
GAP	Communicates a range of no longer relevant sequence numbers
HEARTBEAT	Communicates the sequence numbers of available data (publication issue or object parameter state)
ISSUE	Communicates publication issue data with an optional sequence number
VAR	Communicates parameter state data for a specific network object (RTPS objects have parameter sequences that encapsulate the object's properties; each VAR submessage includes the sequence number)
INFO_TS	Communicates a timestamp applicable to the submessages that follow in the message.
INFO_SRC	Indicates the address, HostID and ApplicationID of the source of the subsequent submessages in the message
INFO_REPLY	Indicates the address and port to which to send replies to subsequent submessages in the message

Table 1 RTPS Submessage Types

appear ahead of the submessage. This allows incremental processing of the submessages. The protocol uses standard messages and elements for the meta-traffic used during the declaration phase. The reuse of mechanisms simplifies the logic and reduces code size.

RTPS uses detection to assure consistency. There are two basic techniques to assure consistency: prevention and detection. Prevention makes sure the system never gets into an inconsistent state. Detection relies on extra information that allows inconsistencies to be detected and corrected before they affect the state of the system. Detection typically allows for higher-performance implementations because it can optimize the common, error-free path.

All RTPS messages start with a 16-octet header (Figure 2). The first four octets contain the characters RTPS. The protocol version is composed of two octets (major version and minor version). The VendorId consists of two octets that allow discriminating among different providers of protocol implementations. The HostId is 4-octets and must be generated to be unique among all hosts in the RTPS network; the ApplicationID is 4-octets and must be unique among all applications within a host. Figure 2 illustrates the RTPS header. Note that the RTPS header can be interpreted independently of machine endianess because all information is processed as octets.

All submessage headers have a fixed format encoding the submessage type, optional submessage elements, and a link to the next submessage. This allows a message processor to identify an individual message without needing to interpret the message contents (Figure 3). There are eight types of submessages, each identified by a unique SubmessageId, used for all communications, including data distribution and configuration management (Table 1).

Network Objects

The RTPS administers the configuration of applications and services using a set of special, built-in network objects. These objects use a composite state transfer (CST) protocol for application and service discovery, and state communication. The CST protocol conveys information about the creation, destruction and attributes of applications and their services among nodes using the VAR, ACK, GAP, and HEARTBEAT submessages.

The full description of the submessage contents, reliable and unreliable publish-subscribe protocols and CST protocols will be contained in the full specification, which will be released to the IETF in the first quarter of 2002.

The RTPS protocol is one part of RTI’s two-part policy to establish standards for data distribution services for real-time applications. The other effort is a set of data-centric publish-subscribe application services. This standards effort is currently in the form of an Object Management Group (OMG) Request for Proposal (RFP) issued in September, 2001. More information on this RFP can be found at http://www.omg.org/techprocess/meetings/schedule/Data_Distribution_Service_RFP.html ◢

Real-Time Innovations,
Sunnyvale, CA.
(408) 734-4200.
[www.rti.com].

Real-Time Innovations, Inc.
155A Moffett Park Drive
Sunnyvale, CA 94089
phone: (408) 734-4200
fax: (408) 734-5009
web site: www.rti.com