PRISMTECH™
AN ADLINK COMPANY

DDS-XRCE

The Protocol for eXtremely Resource Constrained Environments

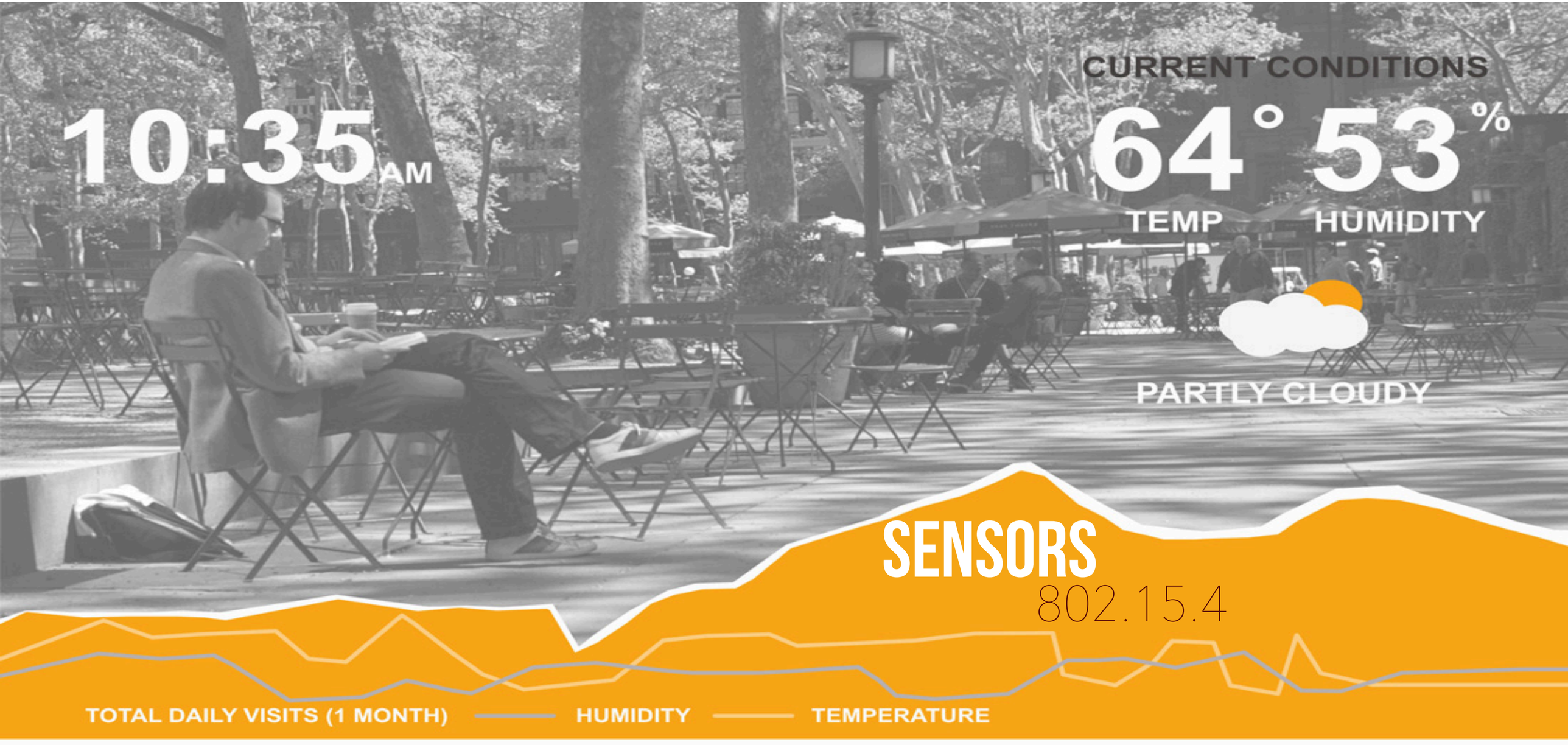
Angelo Corsaro, PhD
CTO, ADLINK / PrismTech
angelo.corsaro@adlinktech.com

OBJECTIVE

The goal of DDS-XRCE is to bring DDS connectivity to devices that are constrained with respect to the node resources, such as computational resources and power, as well as the network.

SMART LIGHTBULBS

96Kbytes Memory



TOTAL VISITS TODAY
622 ↑

CURRENT CHAIR OCCUPANCY
65% ↑

OCCUPANCY @ TABLES
43% ↓

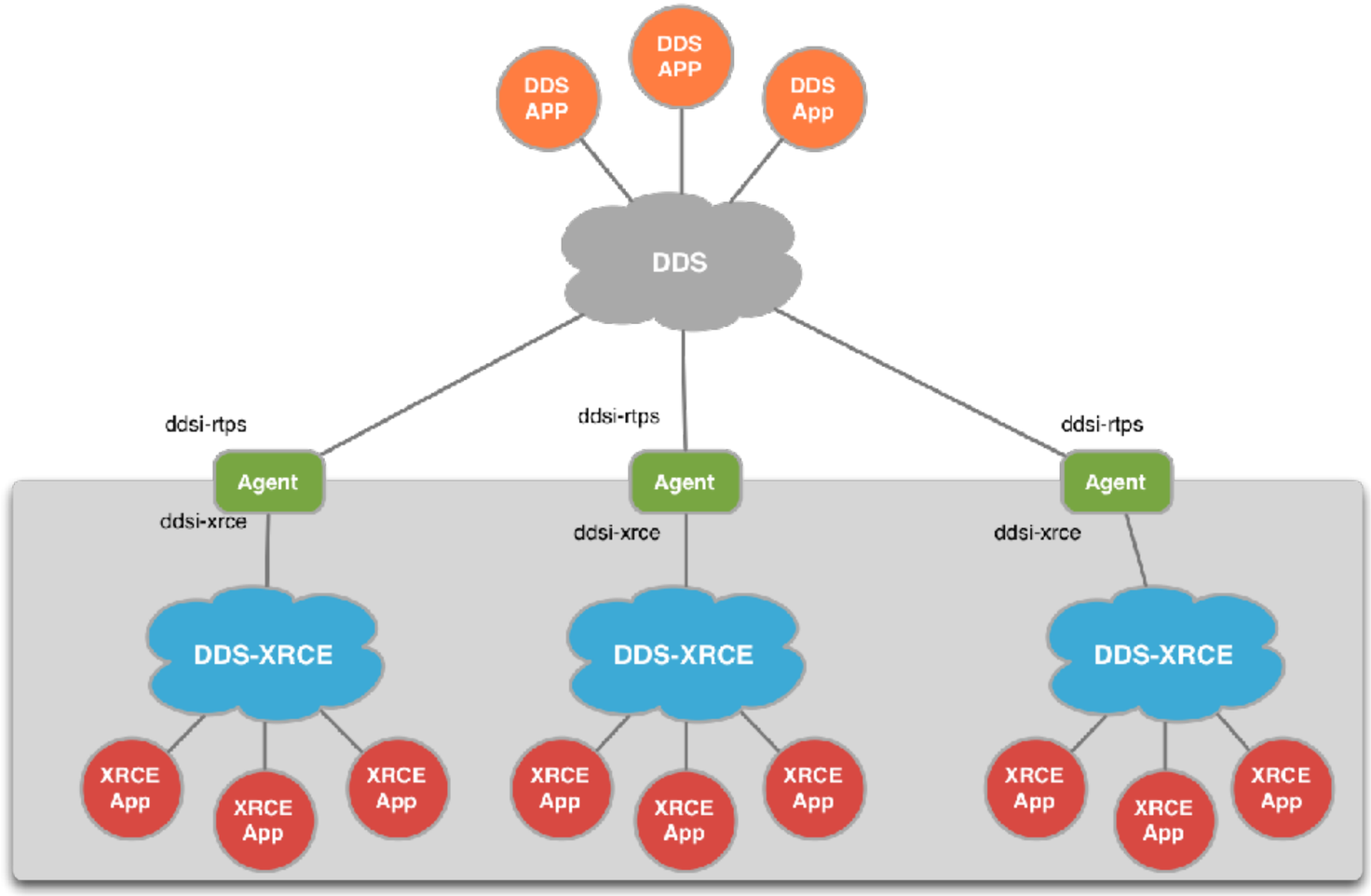
OCCUPANCY @ BENCHES
23% ↑

CONNECTED MEDICAL DEVICES

Low Protocol Overhead

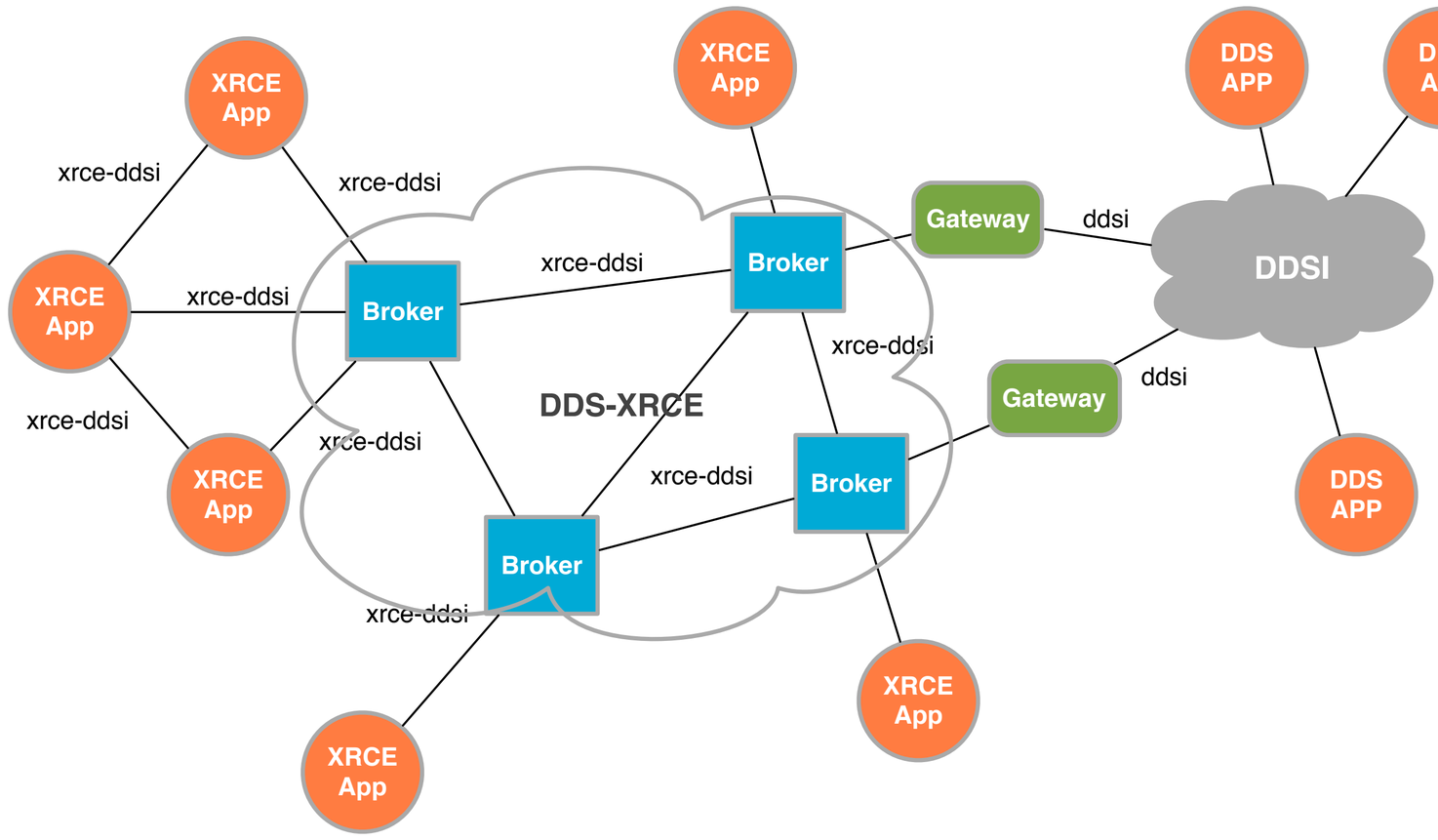
ARCHITECTURAL MODEL

In DDS-XRCE devices constrained by local resources and or network are brokered into DDS by a DDS-XRCE agent.

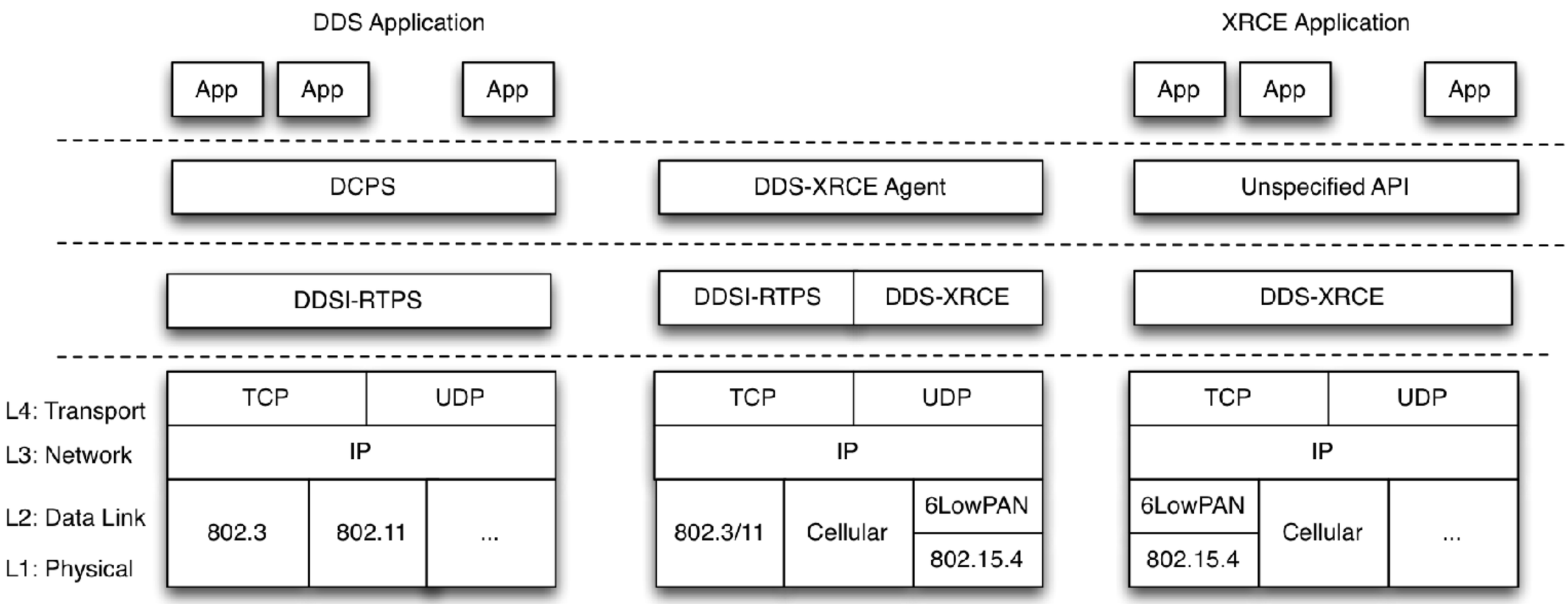


ARCHITECTURAL MODEL

The XRCE global data space can be implemented by peers or by a network or brokers or both depending on deployment network and device constraints



SCOPE



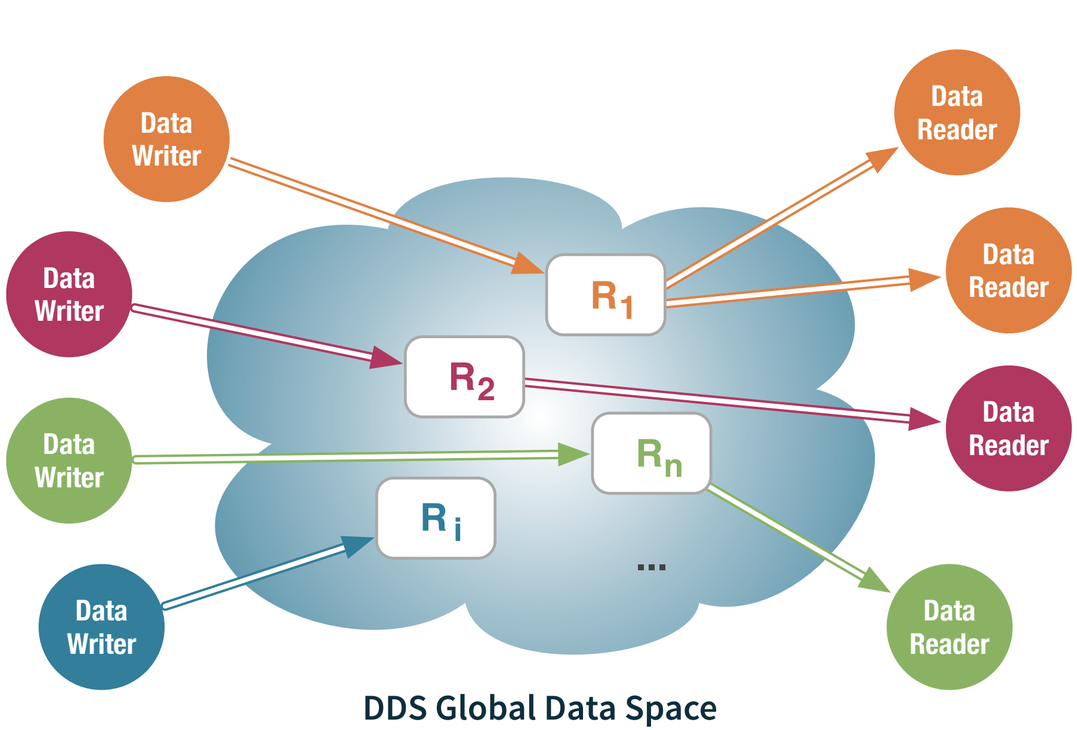
The XRCE Protocol

CONCEPTUAL MODEL

XRCE provides a data space abstraction in which applications can read and write data autonomously and asynchronously.

The data read and written by XRCE applications is associated with one or more resources identified by a URI.

Depending on the context a resource may represent a topic or an instance.



DDS Global Data Space

TOPIC DEFINITION

A Topic can be explicitly defined by the application or pre-defined as part of the system configuration.

The definition of a topic can be done as follows:

```
define_topic :: URI -> QoS -> IO(Either Ok Error)
```

READING DATA

Data is read by issuing a query defined as:

```
query :: URI -> Query -> IO([Data])
```

There is no QoS associated with a Query.

When trying to retrieve data it does not make much sense to do that in a best effort manner. Data should be "retrieved" reliably.

CONTINUOUS QUERY

A continuous data stream is created by issuing a continuos query defined as:

```
cquery :: URI -> Query -> QoS -> IO(IStream)
```

The QoS in this case allows to control whether how data will be streamed, for instance reliably or with best effort, at the production rate or down-sampled, etc.

WRITING DATA

Data is written by issuing a write operation defined as:

```
write :: URI -> QoS -> Data -> IO(Either Ok Error)
```

The QoS in this case allows to control whether how data will be streamed, for instance reliably or with best effort, at the production rate or down-sampled, etc.

WRITING DATA

Otherwise for repeated writes a stream can be created as follows:

```
stream :: URI -> QoS -> Either(OStream Error)
```

Once a stream is created data can be written as:

```
write :: Stream -> Data -> IO(Either Ok Error)
```

QoS Policies

RELIABILITY

Reliability does not make sense when issuing a query — query are reliable.

On the other hand the level of reliability can be selected when streaming data in read or write mode.

DURABILITY

DDS provides four different levels of Durability. The desired Durability has to be declared as part the Topic, the DataWriter and DataReader definitions.

XRCE simplifies this model by associating the QoS with the topic.

PARTITION

The partition in XRCE becomes part of the topic name. Thus, different topics in XRCE may map to the same topic in DDS -- but with different partition settings for the readers/writers.

Partition in XRCE are constrained to follow the URI format. We think this is good as it gives some structure and ensures that we can have efficient routing.

The following is an example of a legal topic declaration:

```
xrce://apple.com/smart-watch
```

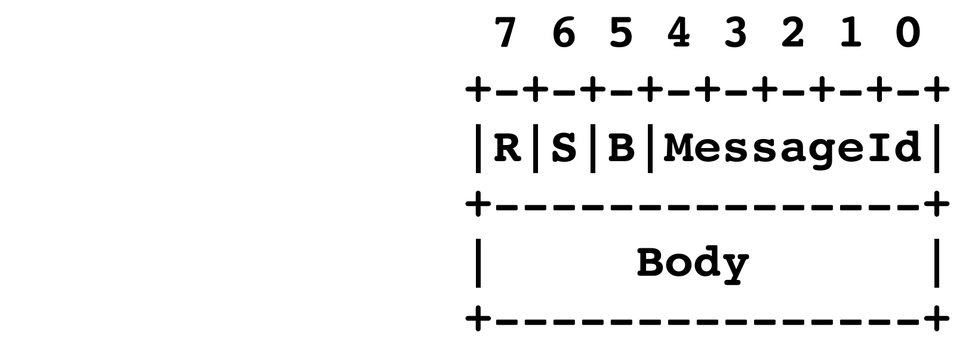
XRCE Protocol

XRCE PROTOCOL

The XRCE protocol specifies how the XRCE global data space can be implemented “on the wire”.

MESSAGE STRUCTURE

XRCE messages are all composed by a single byte header and a body.



R: This is the reliability bit and is set to 1 for message that are intended to be reliable.

S: This is the synchronicity bit and is set to 1 to indicate synchronous messages. A synchronous message requires to receiving party to respond immediately.

B: This bit is used to distinguish message coming from brokering elements such an XRCE agent or applications. It is set to 1 for messages coming from a brokering element.

ENCODING

To minimise wire overhead, XRCE uses fixed as well as variable length encoding.

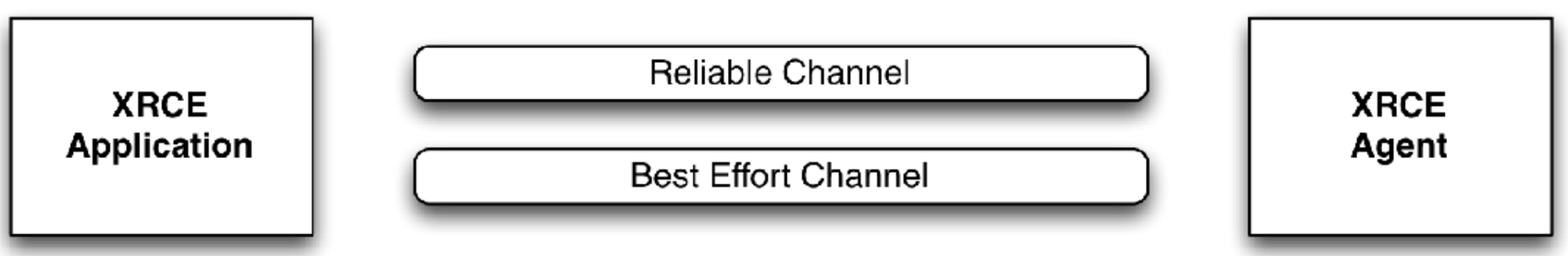
For the details of the encoding refer to the submission.

XRCE Protocol

Data Exchange

XRCE CHANNELS

All exchanges that take place between a XRCE application and the XRCE agent, from the establishment to the closure of a session, happen over two logical channels. These logical channels may be implemented over one or more transports, for instance the best effort channel could be implemented over UDP/IP while the reliable channel over TCP/IP. However, the mechanism provided to implement a XRCE reliable channel doesn't require a reliable transport



BEST-EFFORT CHANNEL SPECIFICATION

The Best-Effort Channel (BC) supports the following primitives:

$$open: (\square) \rightarrow Channel$$
$$close: Channel \rightarrow (\square)$$
$$send: Channel \times Msg \rightarrow Channel$$
$$receive: Channel \rightarrow Msg$$

The semantic of the channel operations are as follows:

open: creates a new Best-Effort Channel (BC);

close: closes the channel, in other terms no messages can be sent anymore over it;

send: given a sequence of messages m_0, m_1, … m_k, sent over the channel, using the channel primitive **send,** the channel shall deliver an ordered subsequence:

$$m_{s_0}, m_{s_1}, \dots, m_{s_h}$$

where:

$$\begin{cases} s_o < s_i < \dots < s_h \\ s_i \in \{x \in \mathbb{N}_0 : 0 \leq x \leq k\} \end{cases}$$

BEST EFFORT CHANNEL IMPLEMENTATION

The best-effort channel implementation is relatively straight forward.

The DDS-XRCE runtime has to make a reasonable effort to send the message over the associated transport and ensure that messages, whilst may be dropped, shall never be delivered to receive out of order.

RELIABLE CHANNEL SPECIFICATION

The Reliable Channel (RC) supports the following primitives:

$$open: (n) \rightarrow Channel$$
$$close: Channel \rightarrow (\square)$$
$$r\_send: Channel \times Msg \rightarrow Channel$$
$$r\_receive: Channel \rightarrow Msg$$

The semantic of the channel operations are as follows:

`open:` creates a new Reliable Channel (RC);

`close:` closes the channel, in other terms no messages can be sent anymore over it;

`send:` given a sequence of messages m_0, m_1, … m_k, sent over the channel using the **r_send** primitive, the channel will deliver (on the other end) exactly the sequence:

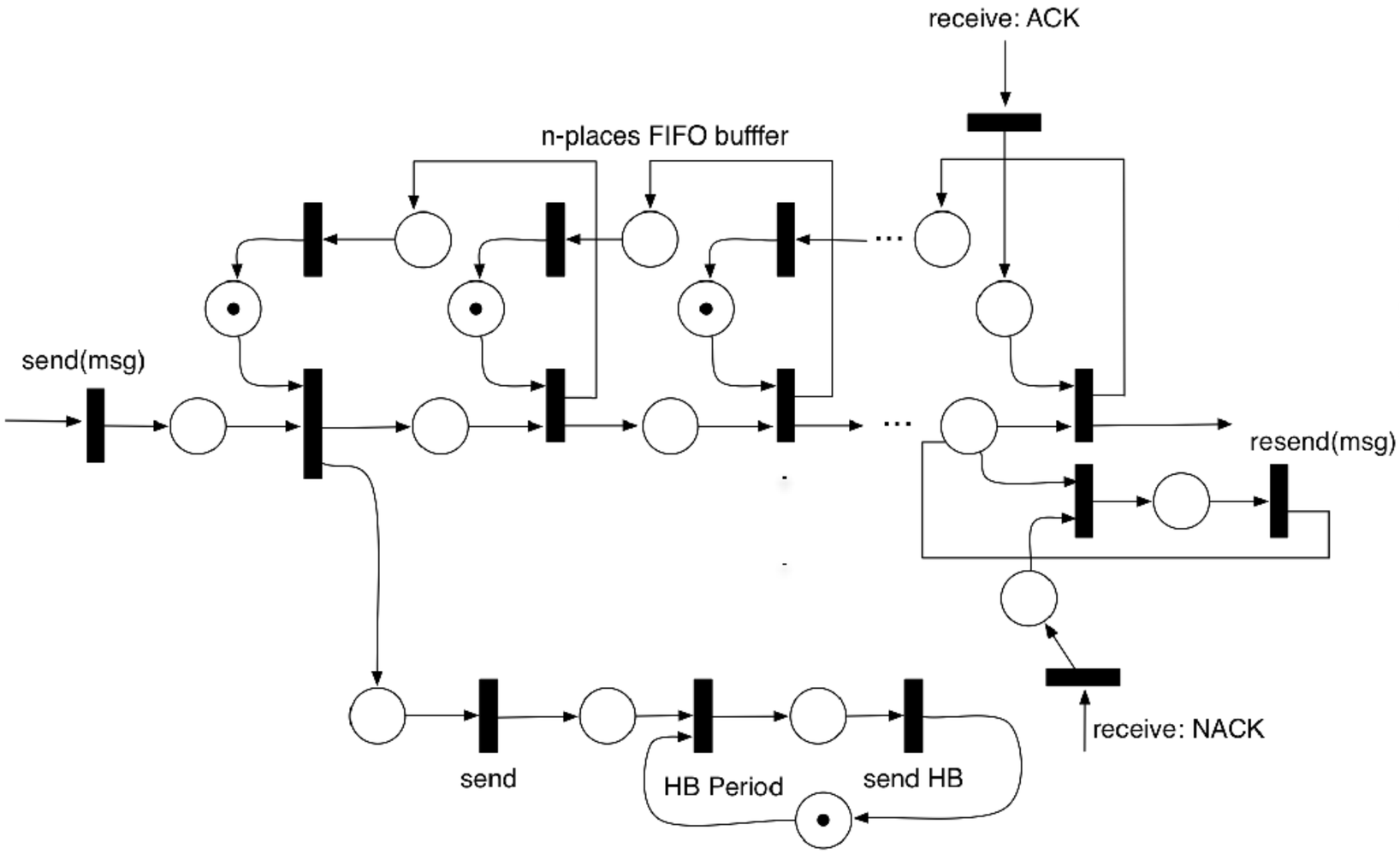
$$m_0, m_1, \ldots, m_k$$

In other terms, the XRCE RC shall not drop messages nor deliver them out of order. This semantics has to be guaranteed only under the assumption that the communicating parties are correct, i.e., don't fail. To maintain ordering the channel relies on a sequence number. The sequence number **sn** is initialized to zero at the creation of the channel and is incremented by one for every message sent on the channel.

`receive`: returns a message, previously sent on the other end of this channel, if available. Otherwise it returns nothing.

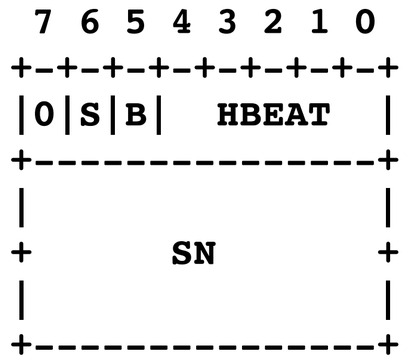
RELIABLE CHANNEL BEHAVIOUR

The Petri Net describes the behaviour of a Reliable Channel with a buffer of **n places**. It is worth to mention that the specification assumes that a NACK can only be received for the head of the queue. This may seem a restriction, but in reality it only means that when receiving a NACK for message with sequence number k the previous messages have been acknowledged.



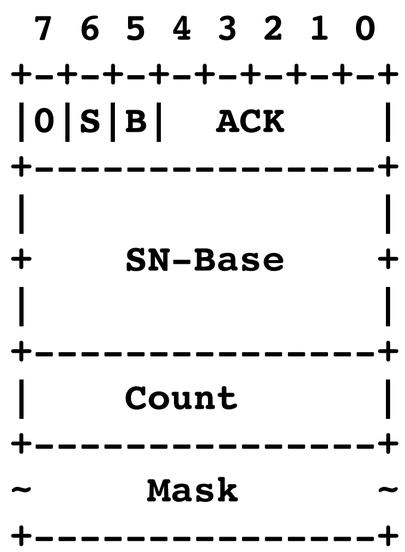
HEARTBEATS

Heartbeats are sent periodically to inform the other end of the channel of the sequence number of the latest message sent. This allows the other end to detect message loss without necessarily waiting for another regular message.



ACK/NACK

A single message is used to indicate both positive as well as potentially negative acknowledgements.



sn: is the sequence number base, meaning the sequence number represented by the first bit in the bit mask;

count: the number of bit that should be considered when parsing the mask;

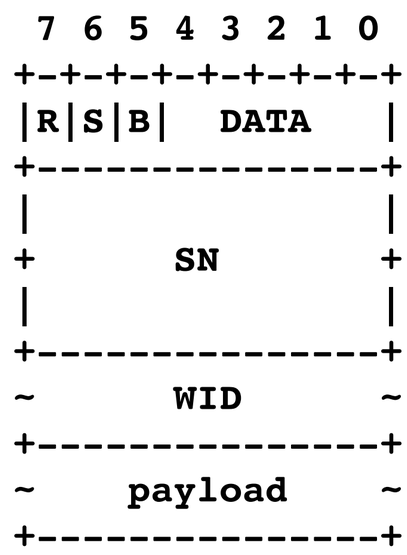
mask: a bit mask where, in the range 0 to count-1, bit set to 1 represent an ACK for the associated sequence number and a 0 represent a NACK. The associated sequence number is obtained by adding to the base sequence number the offset of the bit.

WRITING DATA

sn: the sequence number associated with this sample. As explained in Section , DDS-XRCE uses a logical reliable channel per session. The sequence number is the next sequence number associated wit hthe reliable channel;

wid: this is the period for which the application should assume the broker to be is alive, even if no traffic has been received. A lease period of 0 (zero), represents a never ending lease;

payload: the data provided to the DDS-XRCE protocol by the level n+1.

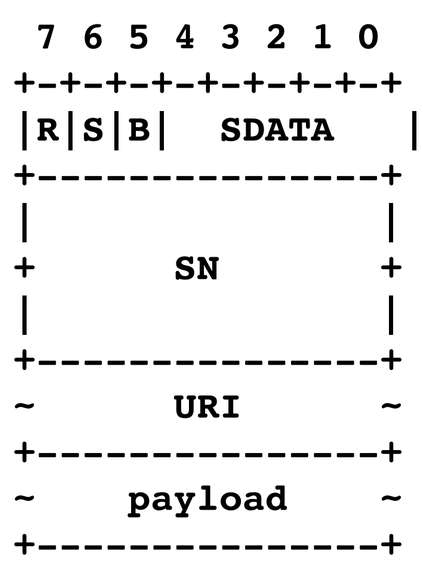


SCOPED WRITE

sn: the sequence number associated with this sample. As explained in Section , DDS-XRCE uses a logical reliable channel per session. The sequence number is the next sequence number associated wit hthe reliable channel;

URI: the URI identifying the resource that should be considered for this sample.

payload: the data provided to the DDS-XRCE protocol by the level n+1.

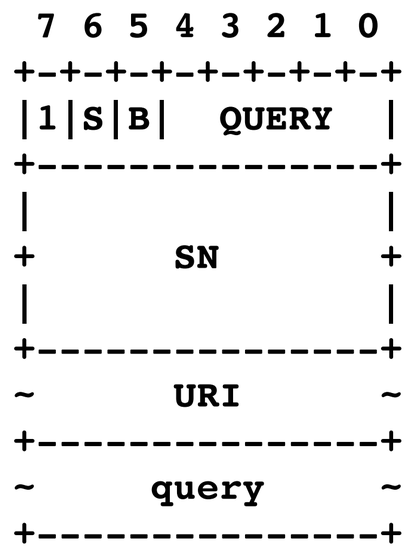


READING DATA

sn: the sequence number of this message.

rid: the identifier of the reader whose data has to be queried.

query: the query to be executed.



XRCE Data Space Implementation

MAPPING

Assuming a session is already established, the mapping between the abstract operations defined for the XRCE global data space are mapped on protocol messages should be as shown in the table

Operation	Protocol Message
define-topic	EDECL (Topic)
query	QUERY
cquery	EDECL (Reader)
write	SDATA
stream	EDECL (Writer)

XRCE vs. CoAP

XRCE vs. CoAP

- 1 byte header including message code
- Message pipelining and batching
- In order delivery
- Batched ACK/NACK
- Durability
- Secure Multicast (from DDS-Security)
- Can negotiate Push/Pull/Push-Pull/ Periodic Push / Periodic Pull
- Will run w/o IP

- 1 byte header + 1 byte message code
- ///
- Unordered delivery
- A ACK per message
- ///
- No Secure MCast
- Pull
- Requires IP

XRCE vs. CoAP

- Can leverage multicast to use only one message to share data regardless of the number of interested parties

- 1 message per observer, as a consequence the load on the server and the network grows linearly

SUMMING UP

Our DDS-XRCE proposal constrains DDS abstractions to reduce resource usage and simplify utilisation

The protocol proposed to implement the abstraction is extremely efficient and has only **4 bytes overhead** for data messages

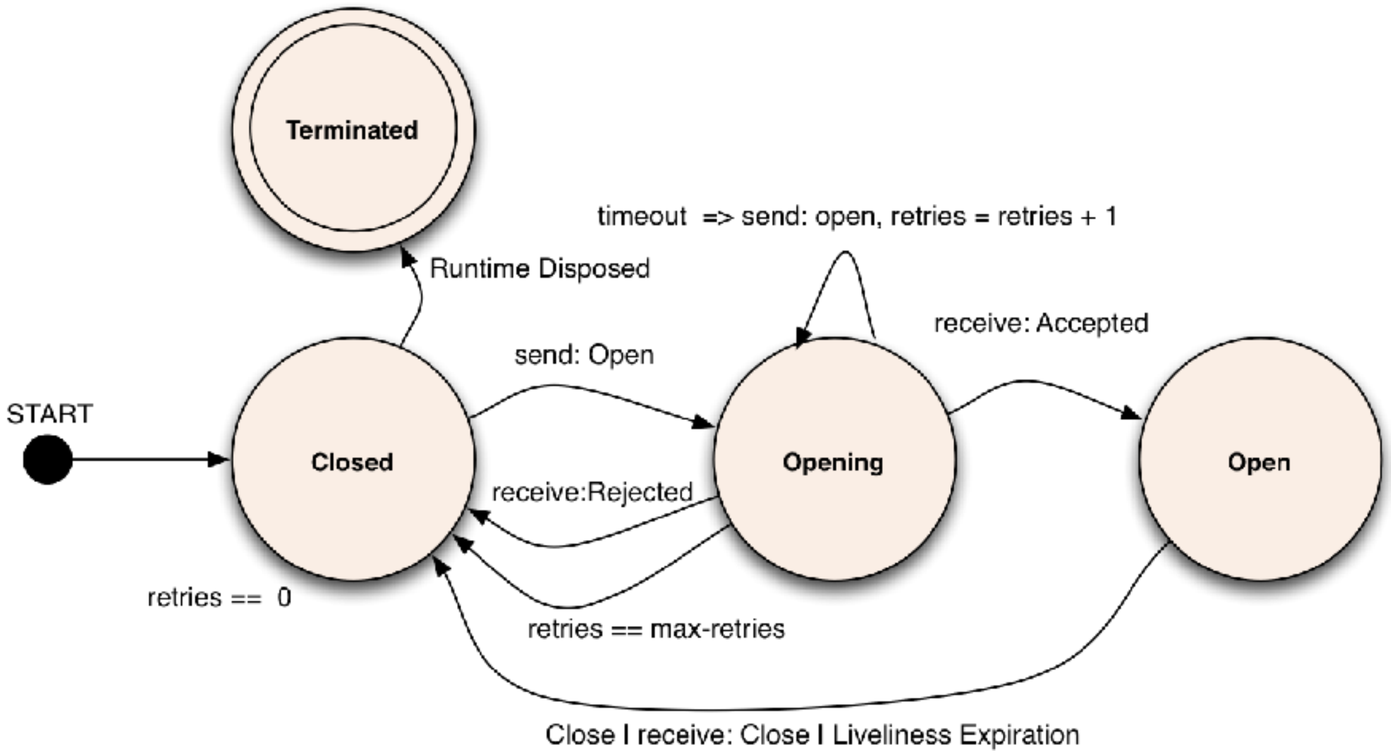
Appendix

XRCE Protocol

Session Management

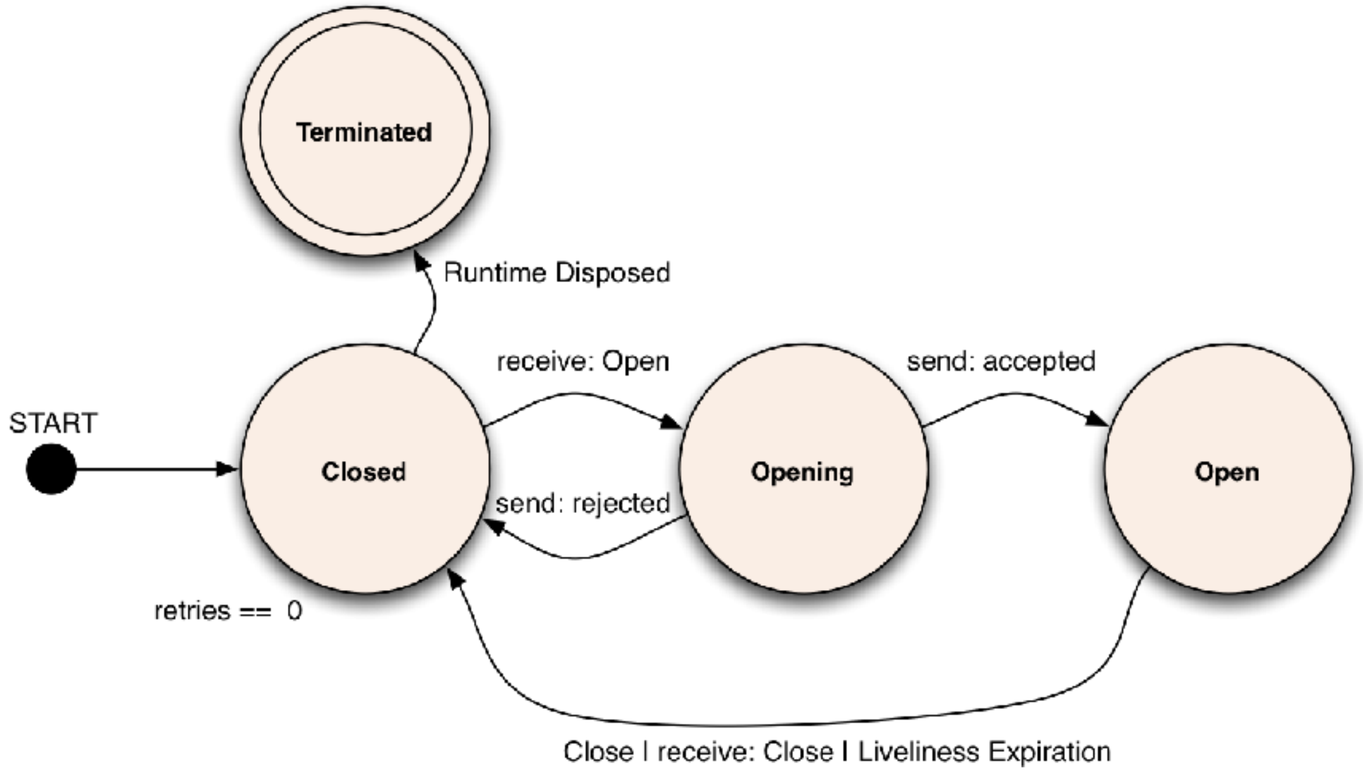
SESSION MANAGEMENT

A newly started application does not have any session, thus starts in the closed session state. The session management protocol is then used to open as well close sessions.



SESSION MANAGEMENT

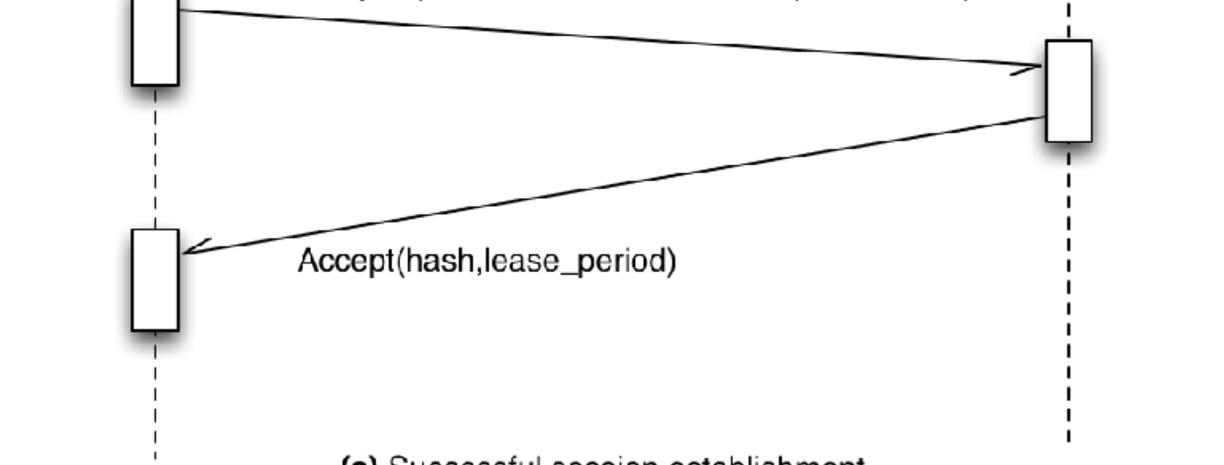
This is the session management state machine for the agent



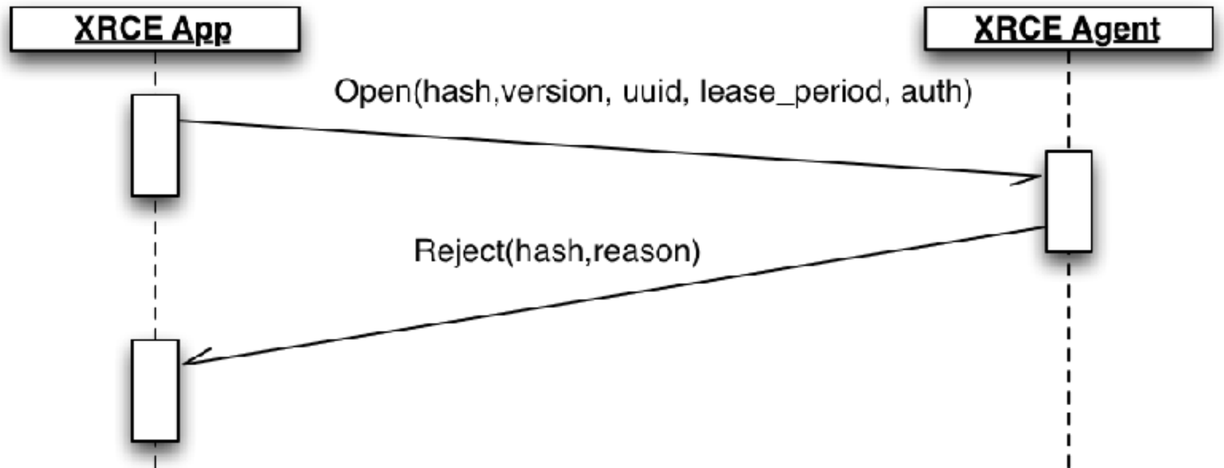
SESSION ESTABLISHMENT

The protocol exchange for opening a session between a DDS-XRCE application and an agent is described in the sequence diagram.

As represented in Figure 5a, to establish a session an XRCE application sends an Open message to the agent which in turn responds with an Accept if the section can be established or with a Reject otherwise (see Figure 5b).



(a) Successful session establishment.



OPEN MESSAGE

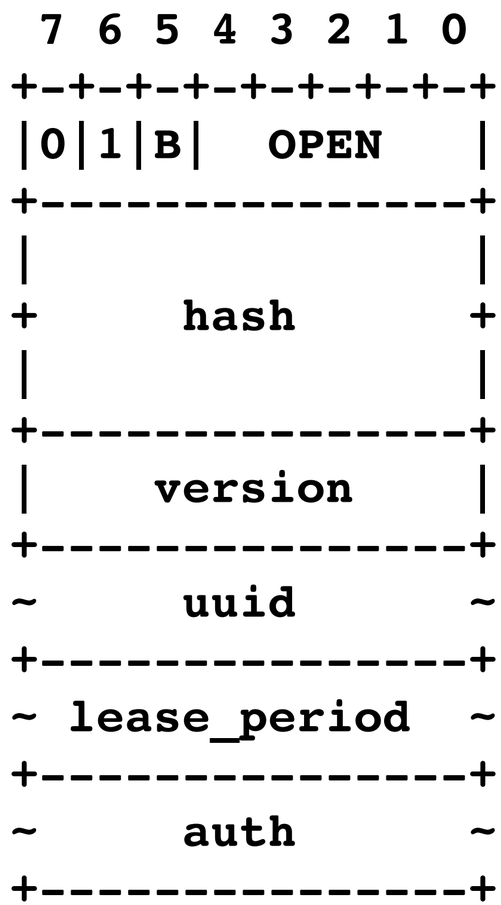
hash: A randomly generated 16bit hash used to uniquely identify the request to open a session.

version: The protocol version.

uuid: A VLE universally unique identifier, this is used to uniquely identify the client.

lease_period: This is the period for which the broker should assume the application is alive, even if no traffic has been received. A lease period of 0 (zero), represents a never ending lease.

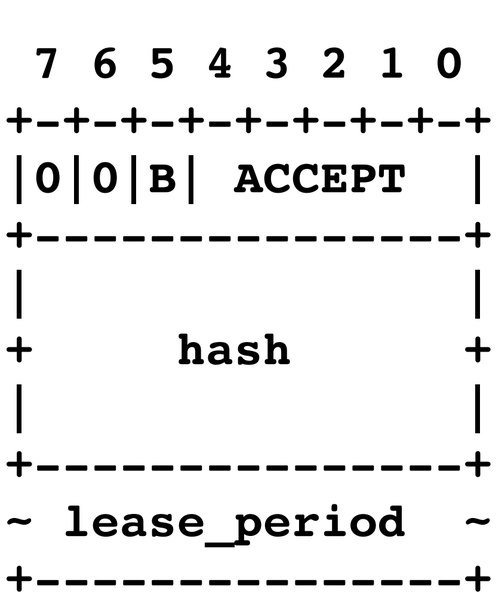
auth: This is a string containing some authentication information, such as a user name and password, etc. The content is not specified and left to the implementation.



ACCEPT MESSAGE

hash: The hash corresponding to the Open message being accepted.

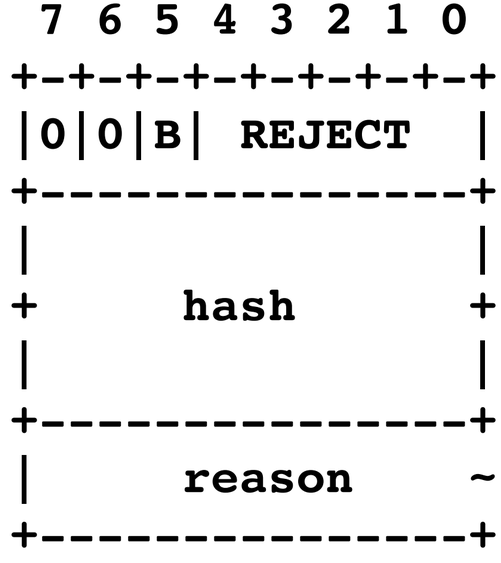
lease_period: This is the period for which the application should assume the broker to be is alive, even if no traffic has been received. A lease period of 0 (zero), represents a never ending lease.



REJECT MESSAGE

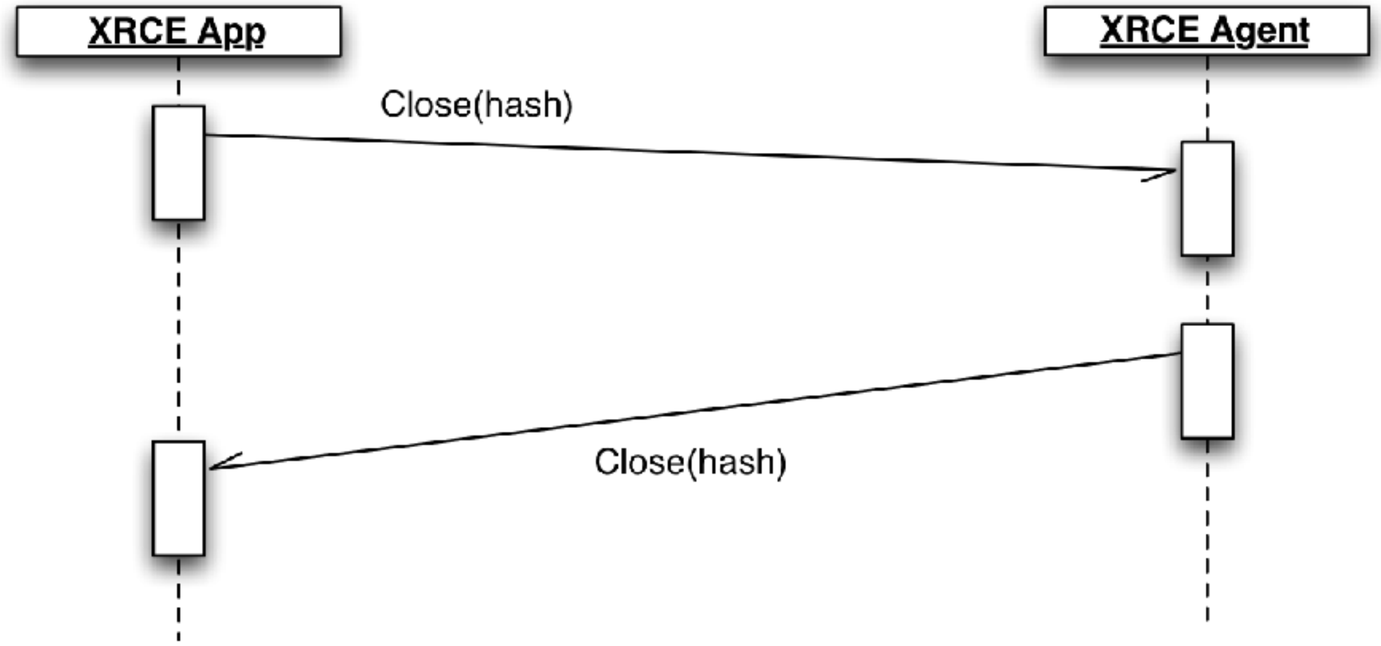
hash: The hash corresponding to the Open message being accepted.

reason: The error code explaining why the request to open a session was rejected.



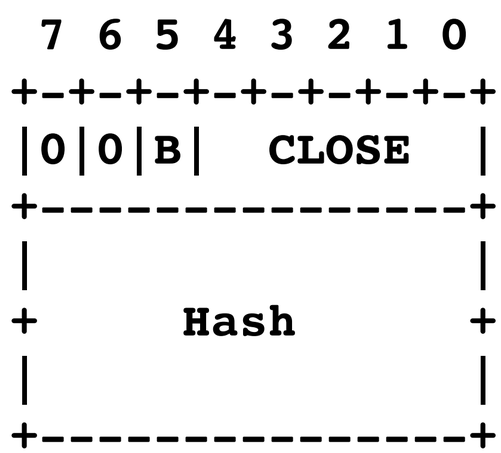
SESSION TERMINATION

The protocol exchange for gracefully closing a session between a DDS-XRCE application and an agent is described in the sequence diagram



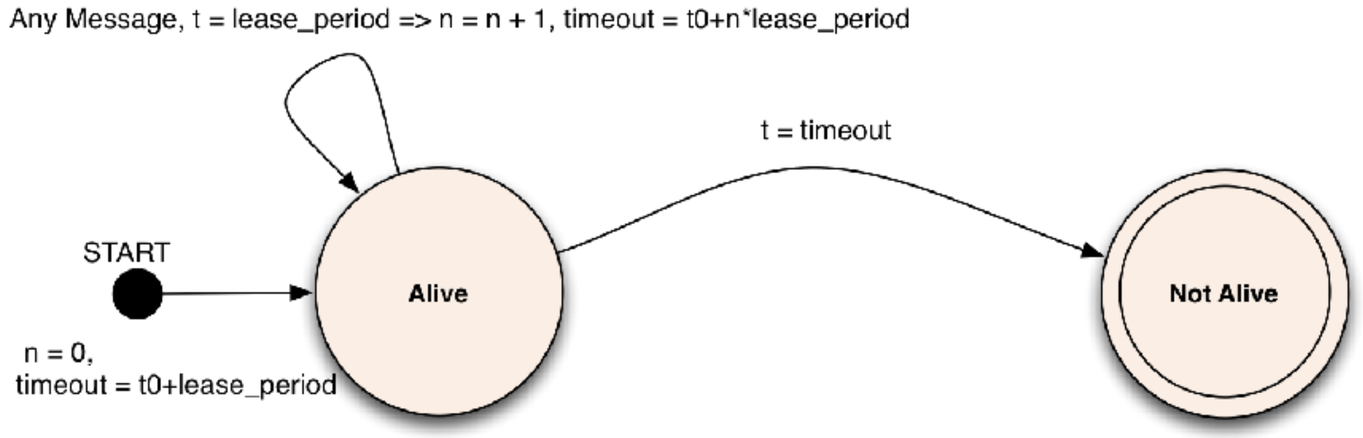
CLOSE MESSAGE

hash: The hash associated with the session, this is the has used for the initial **`Open`** message.



SESSION LIVELINESS

The session is kept alive as far as there is at least a protocol message sent per lease period. If the lease expires without the reception of any message, the session loses its liveliness.



XRCE Protocol

Entity Declaration

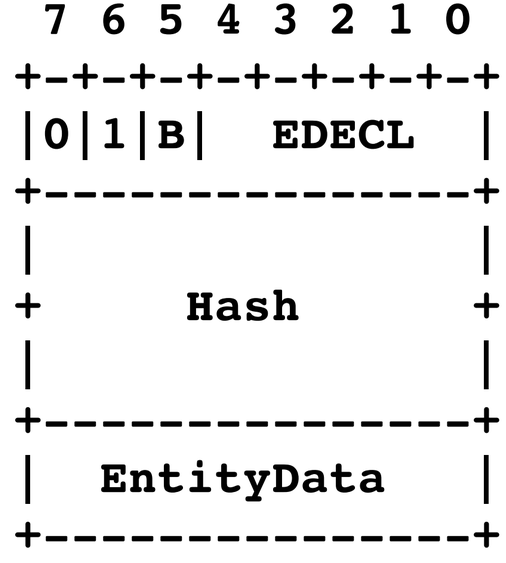
ENTITIES DECLARATION

Once a session has been established, entities such as topics, readers and writers can be declared. The DDS-XRCE wire protocol relies, as for the case of session establishment, on the exchange of best-effort messages in order to declare entities. In general the use of reliable data is minimised to reduce the amount of state that has to be kept by the endpoints.

ENTITY DECLARATION MESSAGE

hash: A randomly generated 16bit hash used to uniquely identify the request to declare an entity.

entity_data: Information concerning the entity to be declared.



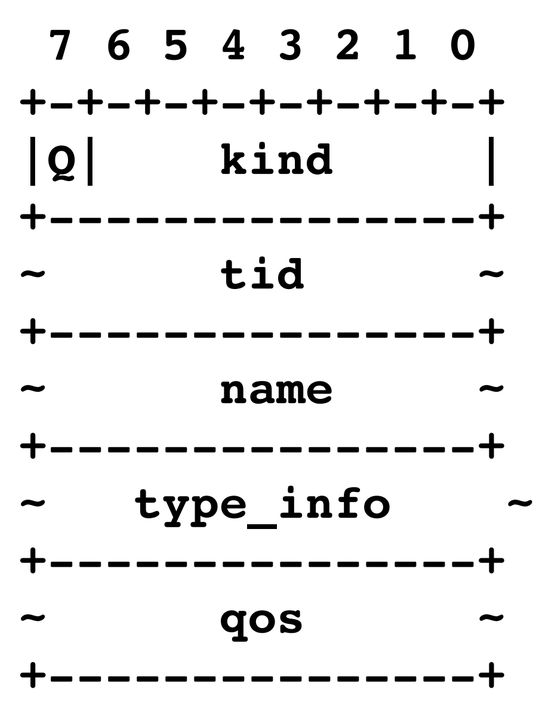
TOPIC DATA

tid: The numeric identifier associated to this topic. This identifier will be used within this session to refer to this topic when declaring data readers and writers.

name: The URI representing the topic name.

type_info: A string representing the type information. This shall minimally be the type name.

qos: The topic qualities of service, only present if Q is set to 1.

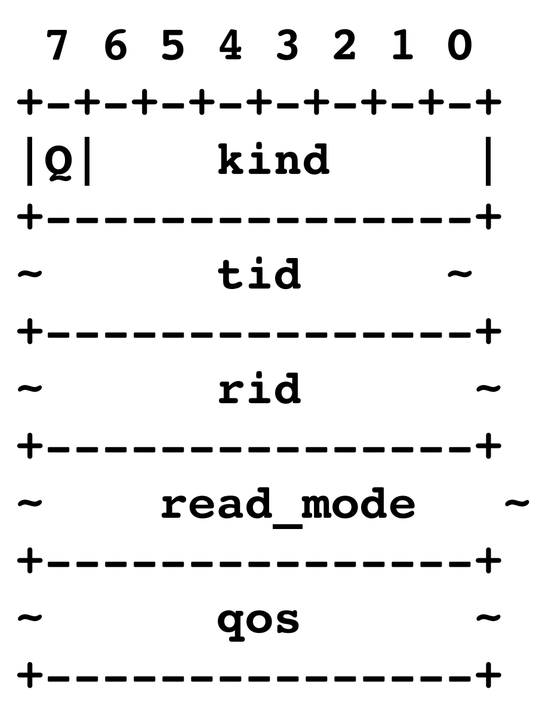


READER DATA

The read mode indicates how data will be read

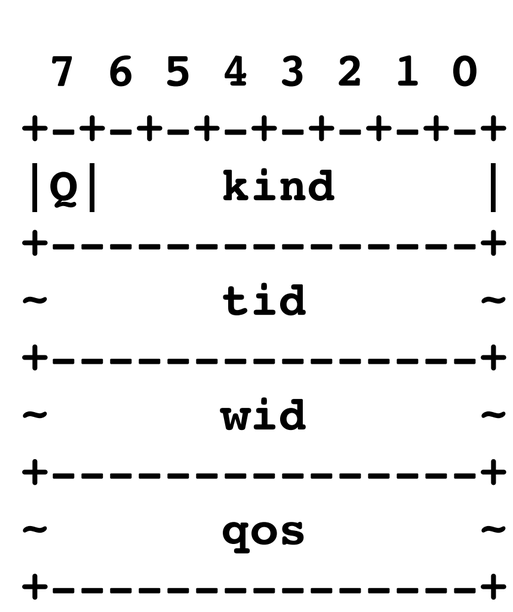
```
enum ReadModeKind {
   PUSH          = 0x01,
   PULL          = 0x02,
   PERIODIC_PUSH = 0x03,
   PERIODIC_PULL = 0x04,
   PUSH_PULL     = 0x05
};

struct ReadMode {
   ReadModeKind kind;
   Option<duration_t> read_period;
};
```



WRITER DATA

The **wid** is the identified assigned to this writer.



QOS REPRESENTATION

The bit-mask will have a bit set only for the QoS policies that are present. Policies are encoded as follows:

Ownership/Ownership Strength. **This policy is encoded as an integral type in which 0 represent shared ownership, while a non-zero value represents an exclusive ownership with the given strength.**

History. The history is encoded as an integral type where 0 represent keep-all, while a positive number represent keep last, with the depth being represented by this number.

Deadline. The deadline is encoded as duration.

```
 7 6 5 4 3 2 1 0
+-+-+-+-+-+-+-+-+
|      Mask     |
+---------------+
|  Own+Strenght |  // integral type
+---------------+
~    History    ~  // integral type
+---------------+
~    Deadline   ~  // duration_t
+---------------+
~    TFilter    ~  // duration_t
+---------------+
~    CFilter    ~  // string
+---------------+
|  Durability   |  // byte
+---------------+
```

QOS REPRESENTATION

The bit-mask will have a bit set only for the QoS policies that are present. Policies are encoded as follows:

Time Filter. The time filter is represented with a duration.

Content Filter. The content filter is represented with a string. The actual syntax of the filter is not specified, it may be SQL, JavaScript or anything else vendor may deem supporting.

Durability. The durability policy is represented by encoding as a single byte the durability kind as defined by the DDS specification.

```
 7 6 5 4 3 2 1 0
+-+-+-+-+-+-+-+-+
|       Mask      |
+---------------+
|  Own+Strenght |  // integral type
+---------------+
~    History    ~  // integral type
+---------------+
~    Deadline   ~  // duration_t
+---------------+
~    TFilter    ~  // duration_t
+---------------+
~    CFilter    ~  // string
+---------------+
|  Durability   |  // byte
+---------------+
```

REGISTERED MESSAGE

hash: The hash corresponding to the EntityDecl message