Demonstrating DDS-XRCE

eXtremely Resource Constrained Environments

Angelo Corsaro, PhD

Chief Technology Officer
ADLINK Tech. Inc.
Co-Chair, OMG DDS-SIG
angelo.corsaro@adlinktech.com

Erik Boasson

Senior Technologist
ADLINK Tech Inc.
erik.boasson@adlinktech.com

PRISMTECH™
AN ADLINK COMPANY

THE PROBLEM

Sensing in **Smart Cities**
has to deal with **Memory**,
Bandwidth and **Power**
Constraints

Sensing in **Smart Farming** has to deal with **Memory**, **Bandwidth** and **Power Constraints**

Cost of connectivity is an issue in Oil Exploration platforms as sensing 1MByte costs over $8

Cost of connectivity is an issue in Smart Grids as the operator has to pay for the 2G/3G/4G data-link

LOW-POWER NETWORK-CONNECTIVITY LANDSCAPE

Local/Personal-Area Networks

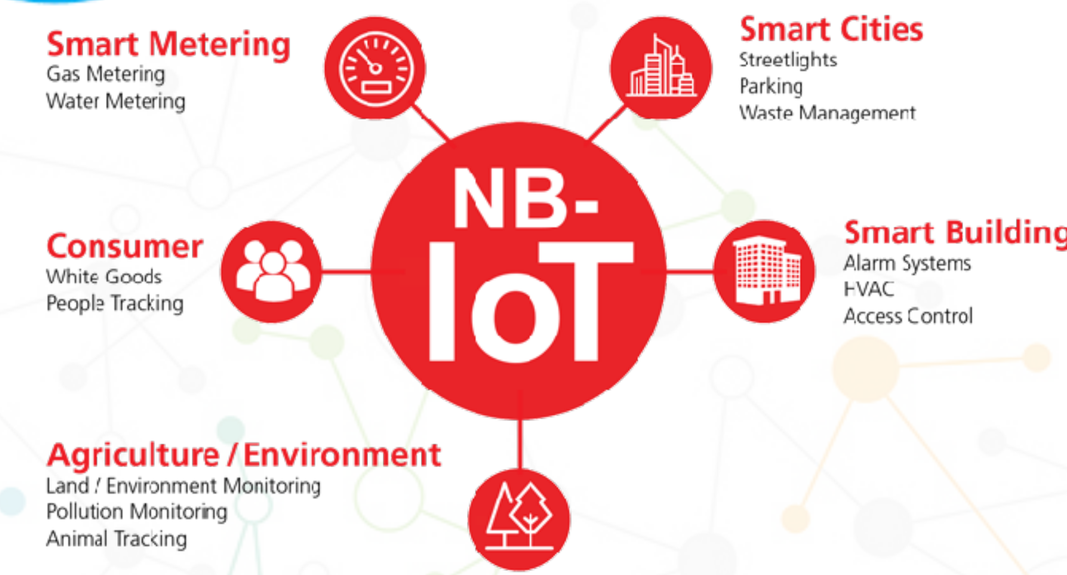
Low Power Local/Personal Area Network have been an area of active innovations in the past several years

While proprietary solution are still widely deployed, standards are starting to gain more and more market traction

Low-Power Wide-Area Networks

Low-Power Wide Area Network technologies have initially emerged in unlicensed spectrum, e.g. LoRa, sigfox, and has a result have some limitations on duty-cycle

LTE NB-IoT leverages 2G and 4G-gap bandwidth to provide low-power non-duty-cycle-constrained connectivity



DATA-SHARING / MESSAGING STANDARDS LANDSCAPE

IoT/IIoT Standards

Data Sharing Standards recommended in IoT/IIoT References Architectures and used in deployed systems **were not designed for extremely resource constrained environments**

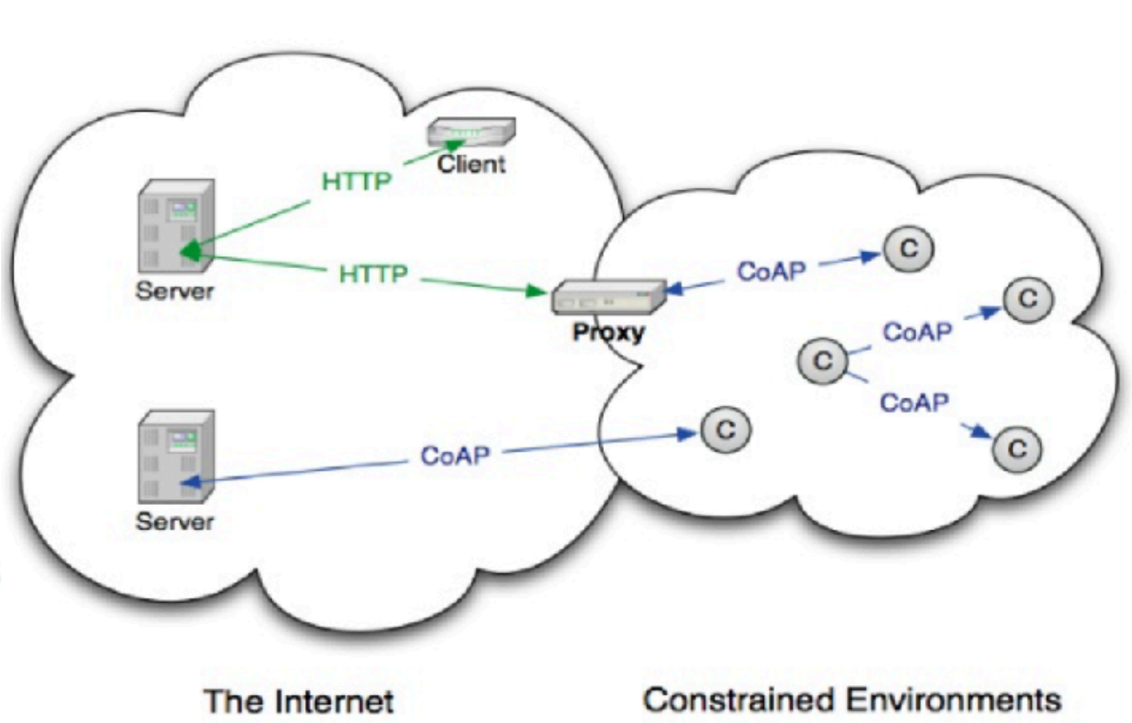
IoT/IIoT Standards

Some of the key limitations are the **wire-overhead** (all), **dependency on TCP/IP** (AMQP, MQTT, OPC-UA) **Push-model** (all), **etc.**

Constrained Application Protocol

CoAP is a Resource-Oriented protocol designed to deal with resource constrained environments

CoAP has been designed to nicely map to HTTP

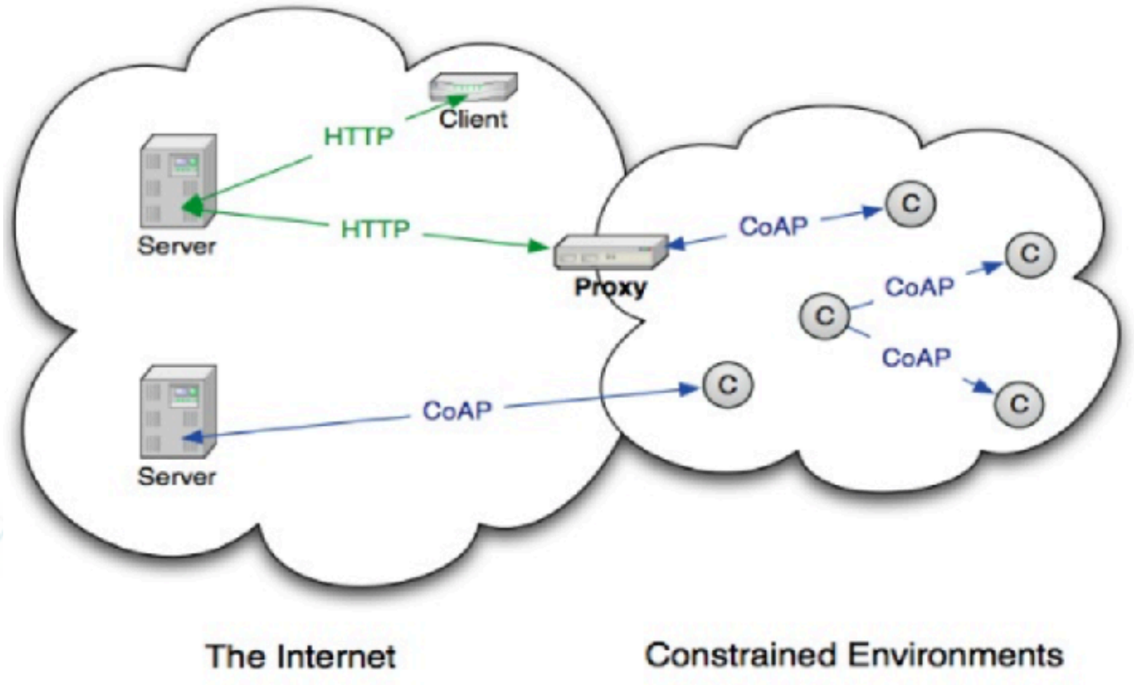


CoAP Limitations

CoAP **lacks temporal decoupling** and the **publish/subscribe** model was added as an **after-thought** to the protocol

CoAP has **limited dynamic discovery** and **only** supports **UDP/IP**

CoAP provides **at most reliable un-ordered delivery**

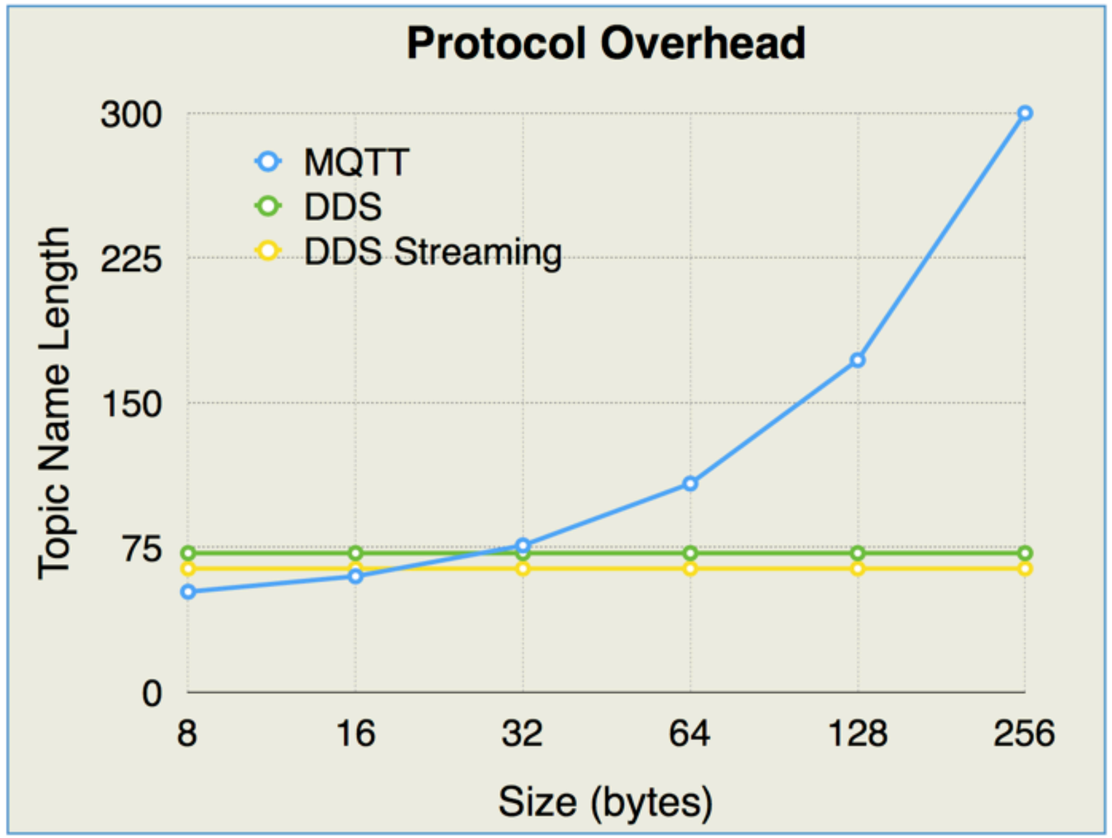


LOW-POWER NETWORK-CHALLENGES

Small MTU

When compared with traditional network technologies, Low-Power networks have very small MTU, for instance 802.15.4 has **127 bytes**

As a consequence **(1)** wire-protocol overhead has to be reduced to the bare minimum to leave sufficient space to application data, **(2)** The wire overhead of IP is major and TCP/IP is unacceptable **(3)** Fragmentation is necessary

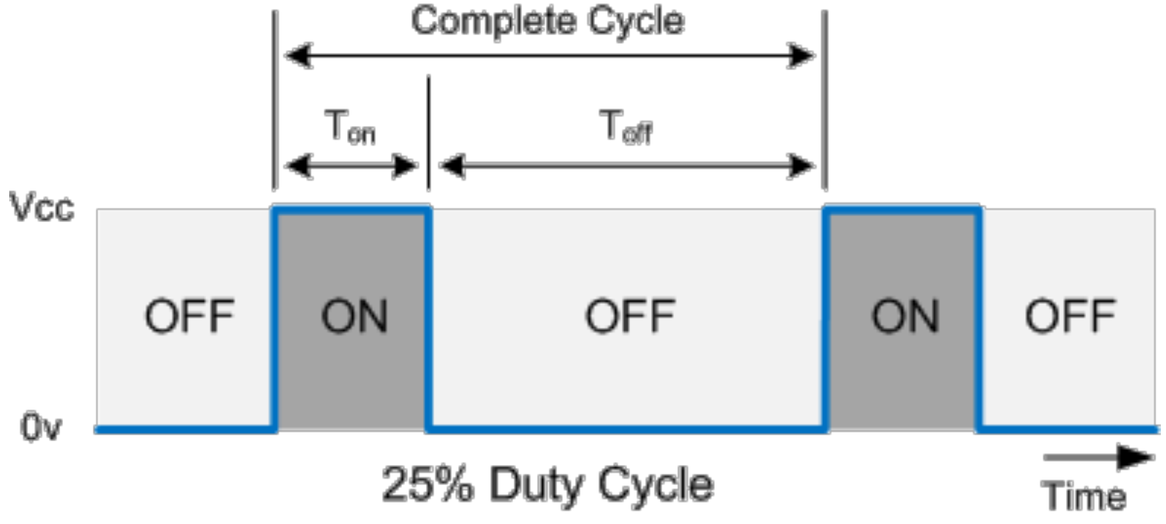


[Ref: A Comparative Study of Data-Sharing Standards for the Internet of Things, Cutter Journal, Dec 2014

Low Duty Cycle

Battery powered applications are often characterised by extremely low duty cycles to ensure that the battery can last decades

This means that the ***traditional*** **push does not work** and ***traditional*** **pull** incurs in **unnecessary I/O** and thus **energy waste**



Reliability

As Low Power WAN networks will become more widely deployed and used in IoT the need for **reliable, ordered delivery** will be driven by applications like **firmware update**

Likewise, **temporal decoupling** will be essential to facilitate the provisioning of large scale systems, i.e. don't need to wait for the device to be online for provisioning it, just assert what's the new state is and let the system eventually propagate it

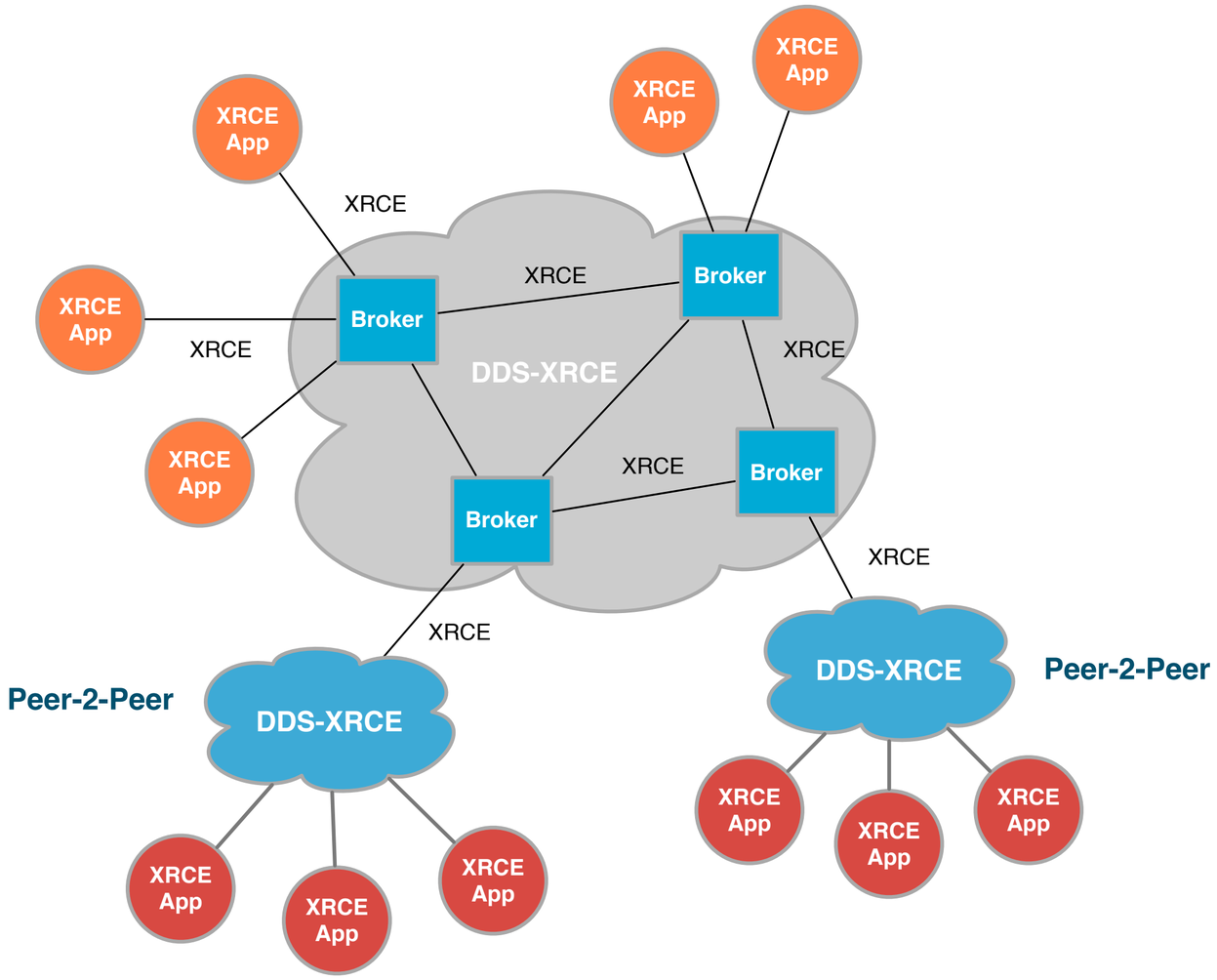
DDS-XRCE

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**.

Architectural Model

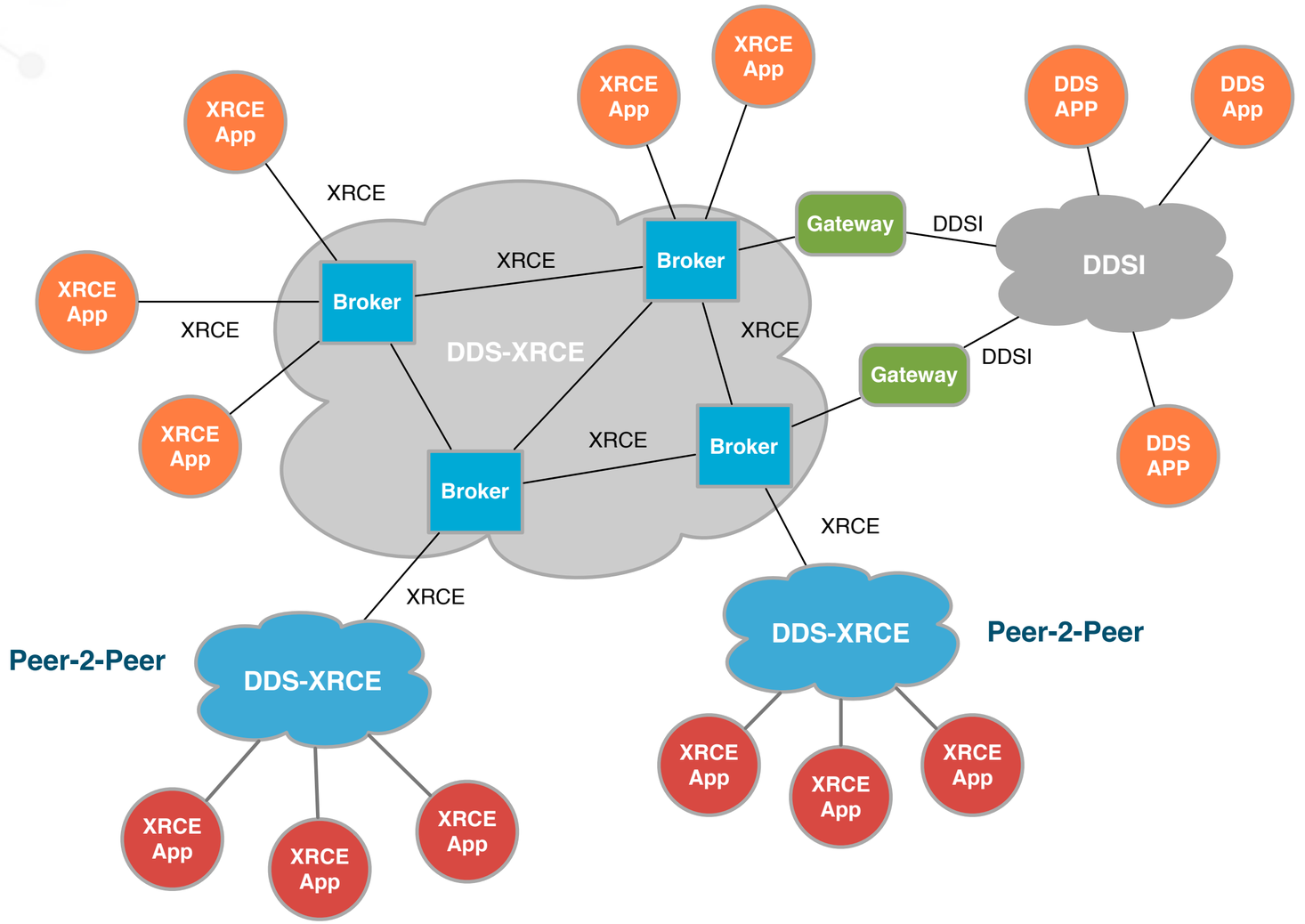
The XRCE global data space can be implemented by peers or by a network or brokers or both depending on deployment network and device constraints



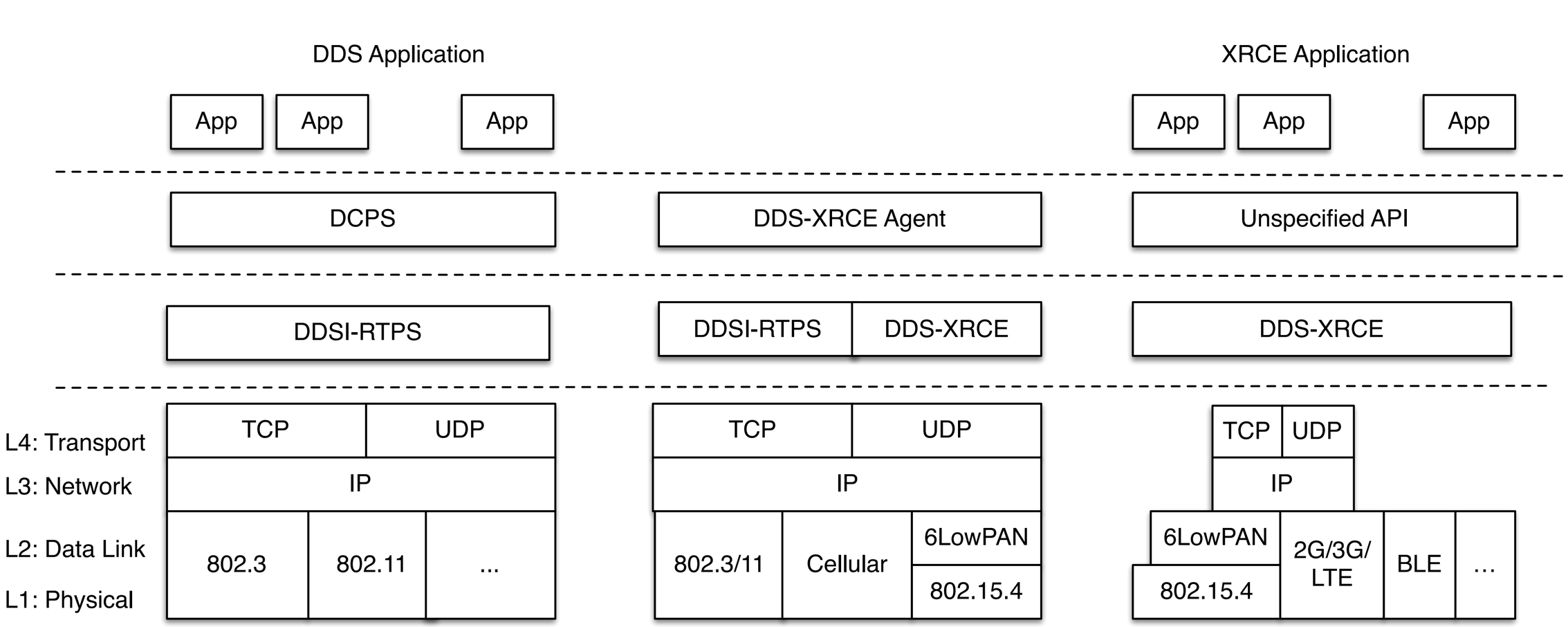
DDS Interoperability

XRCE has been designed to have a trivial mapping to DDS.

Thus Protocol Gateways will be easy to build in SW or HW and very efficient



Scope



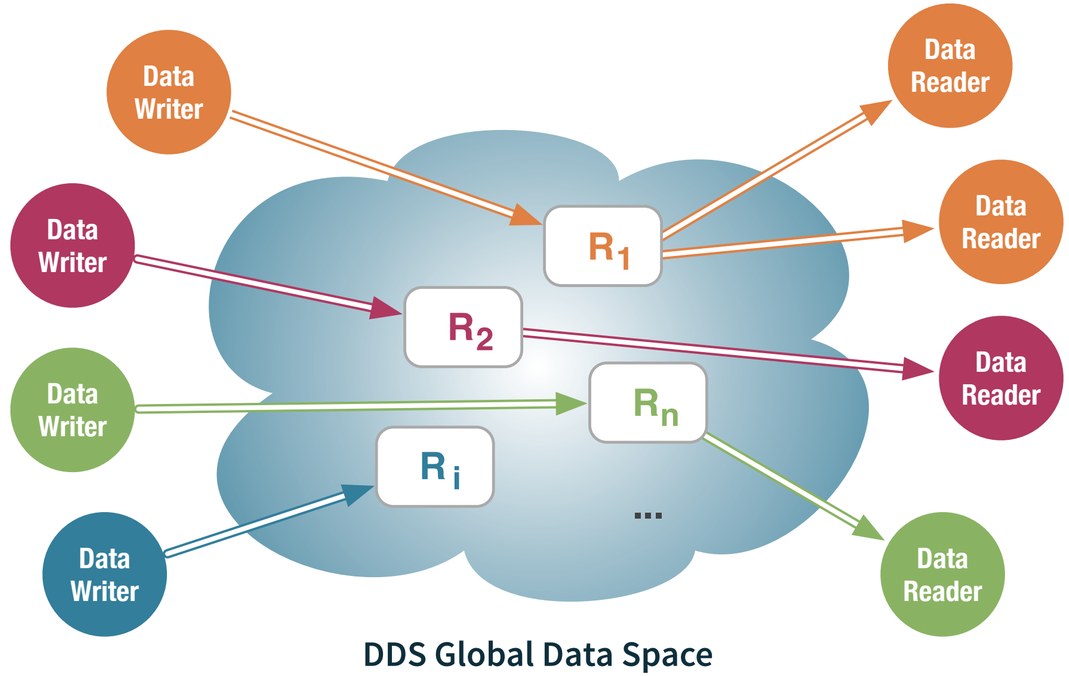
XRCE-Abstractions

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 DDS topic or a DDS instance.



Resource Definition

An **XRCE Resource** is a **closed description for a set of named values**.

If the **cardinality** of the set is **one** then we call it a **Trivial Resource**.

It is **described** by means of a **URI** which may only **include path expansions**.

A Trivial Resource thus corresponds to a URI without wildcards.

```
-- These are Resources
xrce://myhouse/**/bedroom/*/LightStatus
xrce://myhouse/**/musicroom/LightStatus
xrce://myhouse/**/LightStatus
xrce://myhouse/**

-- These are Trivial Resources
xrce://myhouse/floor/1/musicroom/LightStatus
xrce://myhouse/floor/2/musicroom/LightStatus
xrce://myhouse/floor/2/bedroom/erik/LightStatus
```

Note, for compactness we take the liberty of representing trivial resources by the single element that constitutes the set as opposed to the single element set.

Resource Definition

An **XRCE Resource** also has a **set** of **properties**, which are **name-value pairs**. These are used for specifying QoS and for extensibility.

For a resource *X*, there is a number of **predefined meta-data resources** for **discovering child resources** and **properties**. These are named ***X*$ls**, ***X*$props**, &c., and are handled automatically by the XRCE agents.

```
Given these resources:

xrce://myhouse/floor/1/musicroom/LightStatus
xrce://myhouse/floor/2/musicroom/LightStatus
xrce://myhouse/floor/2/bedroom/erik/LightStatus
```

For example:

```
Reading xrce://myhouse/floor/2$ls yields:
  {musicroom,bedroom}
```

Selection

An **XRCE Selection** is the **conjunction** of a **Resource**, and a **Predicate over** the **resource content** and **properties**. The selection's query is separated from the resource by a "?".

A Trivial Resource has a value, therefore a selection on a Trivial Resource yields a single value or none at all. A query on a non-trivial resource may yield many values.

XRCE does not prescribe a format for the value of a resource, and therefore the meaning of a query is dependent on the chosen format. If a query cannot be evaluated on the value of some resource, the result is empty.

For example: these are legal XRCE Queries

```
xrce://myhouse/floor/2/bedroom/jeanclaude/LightStatus?
xrce://myhouse/floor/*/bedroom/*/LightStatus?{luminosity>0}
xrce://myhouse/floor/*/bedroom/*/LightStatus?{property.durability = transient && luminosity>0}
xrce://myhouse/floor/1/musicroom/LightStatus/ID12345?{inRange(property.location, someLocation,
radius)}
xrce://myhouse/floor/1/musicroom/LightStatusπ{deviceId, property.location}
```

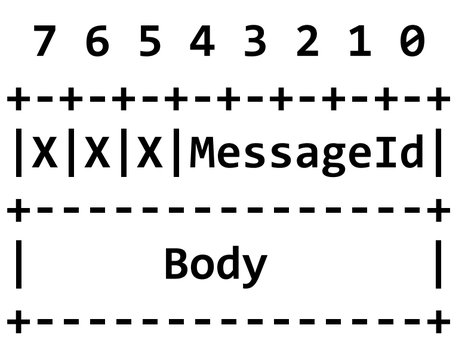
XRCE Protocol

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: When present, this is the reliability bit and is set to 1 for message that are intended to be reliable

S: When present, this is the synchronicity bit and is set to 1 to indicate synchronous messages requiring the receiving party to respond immediately

X: An extension bit used by some messages to indicate additional fields are present

Message Markers

Messages can be “decorated” by prefixing them with certain markers.

A message together with its decoration shall be treated as an atomic unit on the network.

In this version of the protocol, the only markers are Migrate, Conduit and Frag

When a message is decorated with multiple markers, these markers shall occur in the order:

- **Migrate**
- **Conduit**
- **Frag**

Encoding

XRCE always uses little-endian for its data and primarily uses variable-length encoding (VLE) to save in the common cases

VLE is a value-based encoding, agnostic to the word size used to represent the number in memory and encodes 7 bits per byte

$0 .. 2^7-1$	⇔	$[00_{16} .. 7f_{16}]$
$2^7 .. 2^{14}-1$		$[80_{16} .. ff_{16}] ; [00_{16} .. 7f_{16}]$
$2^{14} .. 2^{21}-1$		$[80_{16} .. ff_{16}] ; [80_{16} .. ff_{16}] ; [00_{16} .. 7f_{16}]$
&c.		*&c.*

Addressing

The **source address** of **each message** is **assumed** to be a **unique address of the sender**.

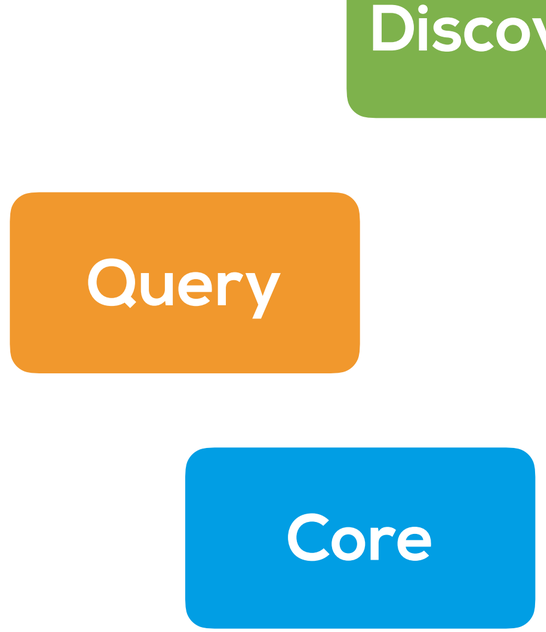
Data sent to such an address will in principle arrive at the sender (or be lost).

Each node has a unique ID (e.g., a UUID) that remains the same even when it changes network address.

The ID is only exchanged for session establishment.

Protocol Modules

XRCE is a modular protocol that allows implementation to decide which subset of the protocol to implement so to minimise resource usage and retaining interoperability



Protocol Modules

Discovery. The discovery module allows application to dynamically discover each other and exchange entities declarations.

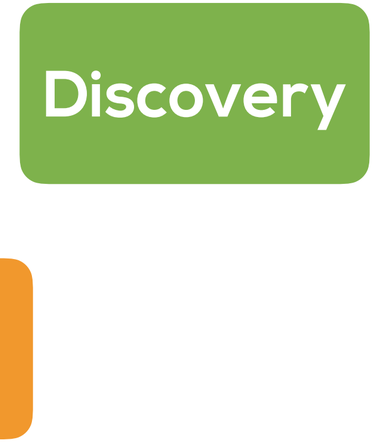
- SCOUT, HELLO, KEEPALIVE, PING, PONG
- DECLARE: PUB, SUB, COMMIT, RESULT

Query. The query profile allows applications to execute queries over resources.

- DECLARE: RESOURCE, SELECTION, BIND

Core. The core profile defines the session establishment and the data exchange.

- OPEN, ACCEPT, CLOSE,
- WDATA, SDATA, BDATA
- PULL



XRCE Protocol

SCOUTING

Scouting

Scouting is the discovery of other nodes in the network.

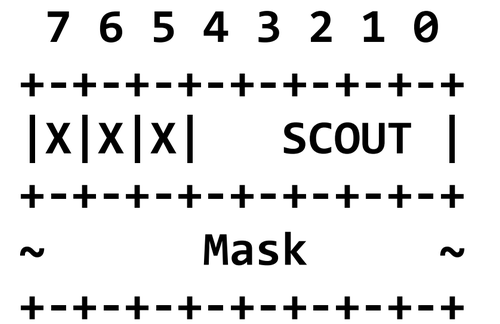
If the transport has its own discovery (Bluetooth for example), the scouting phase may be skipped.

Scout and Hello

Scout is a request for other nodes in the network to send a Hello. MASK encodes the type: broker, durability, peer, client, … it is requesting a response from.

Hello informs nodes of locators, properties. MASK encodes the type of the node itself. The locators allow a node to specify additional addresses at which it is reachable, e.g., multicast addresses.

When a client sends "SCOUT broker" brokers would respond with HELLO. Armed with this knowledge, the client can try to open a session with a broker.



P is set iff properties present

XRCE Protocol
SESSION MANAGEMENT

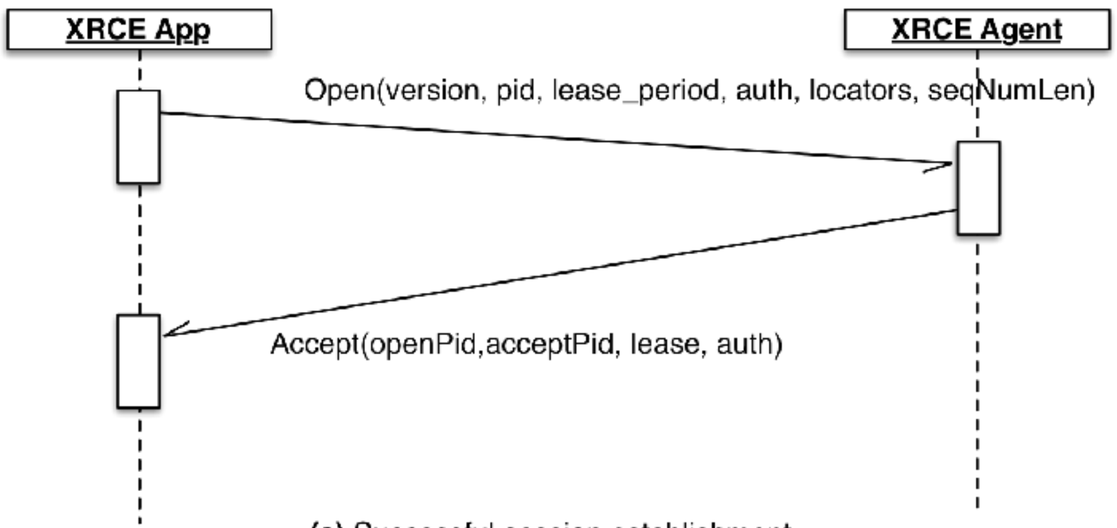
Session Management

For two nodes to communicate in XRCE, they **need** to establish a **session**. With a session comes the ability to publish or subscribe. Included in the **session** are **conduits**, pairs of logical **reliable** and **best-effort channels** for data.

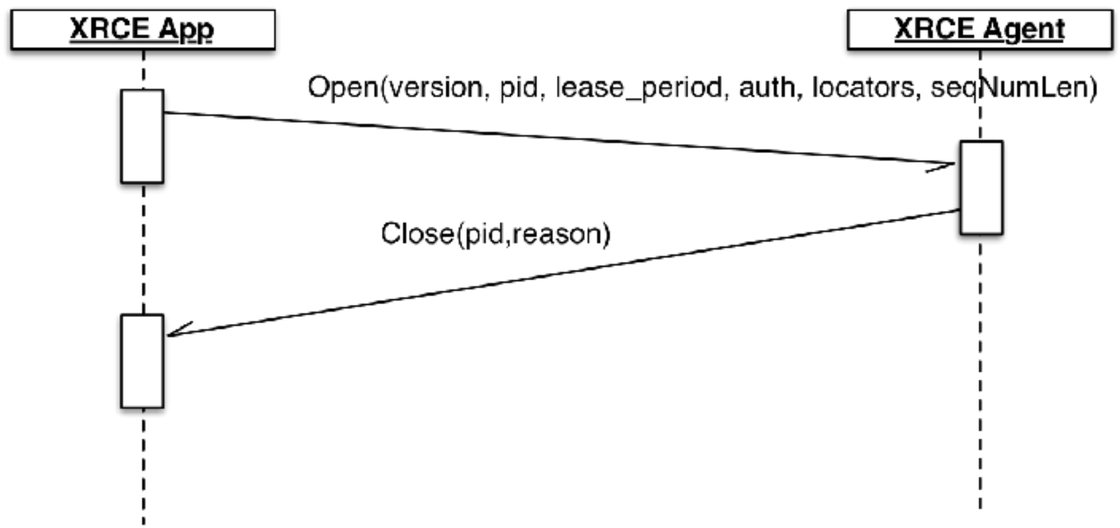
A "client" typically only needs a single session, with its broker. In mesh networks, it is expected that brokers would typically need sessions with other brokers as well.

The **conduits** can be implemented over any transport, e.g., TCP (would make sense in client/broker over Ethernet), unicast/multicast UDP (e.g., peer-to-peer), Bluetooth, …

Opening a Session



(a) Successful session establishment.



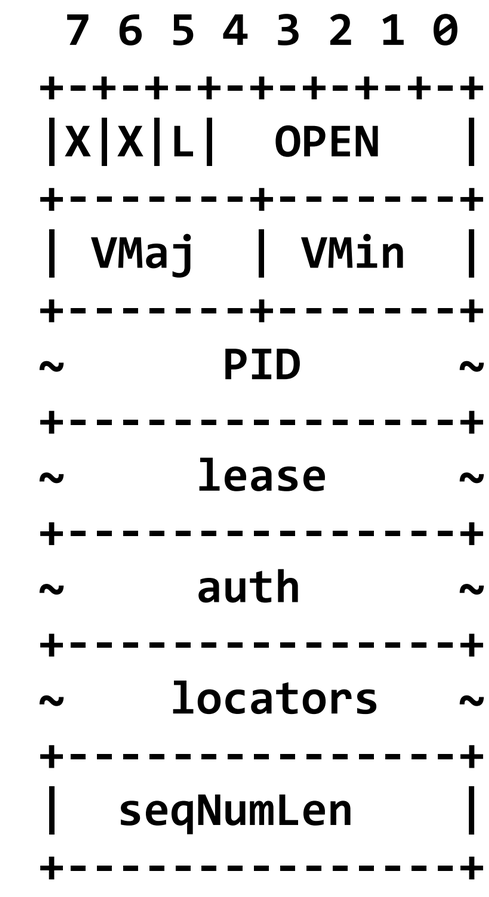
(b) Unsuccessful session establishment.

Opening a Session

Open is a request to establish a session.

- **VMAJ/VMIN** is protocol version
- **PID** is the requesting node's unique id (PID is short for peer id)
- **lease_period** is the lease duration expression in units of 100ms
- **auth** is a sequence of bytes for authentication data
- The **locators** provide additional locators useful for asymmetric discovery

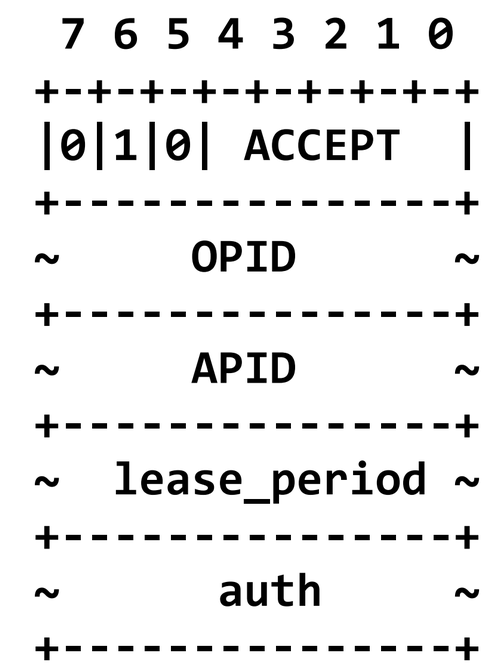
- **seqNumLen** is present when L=1 and represents the number of bytes used to represent sequence numbers for this session



Opening a session

Accept is the successful response to an **Open**.

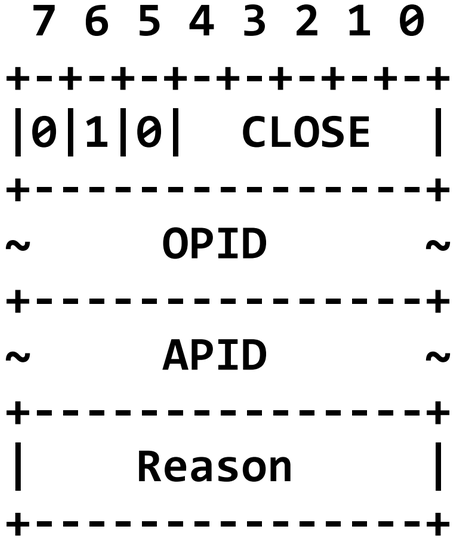
- **OPID** is the requesting node's unique id
- **APID** is the accepting node's unique id
- **lease_period** is the lease duration expression in units of 100ms
- **auth** is a sequence of bytes for authentication data



Rejecting or closing a session

To reject a request for opening a session and to explicitly close an existing session, a **Close** message is used.

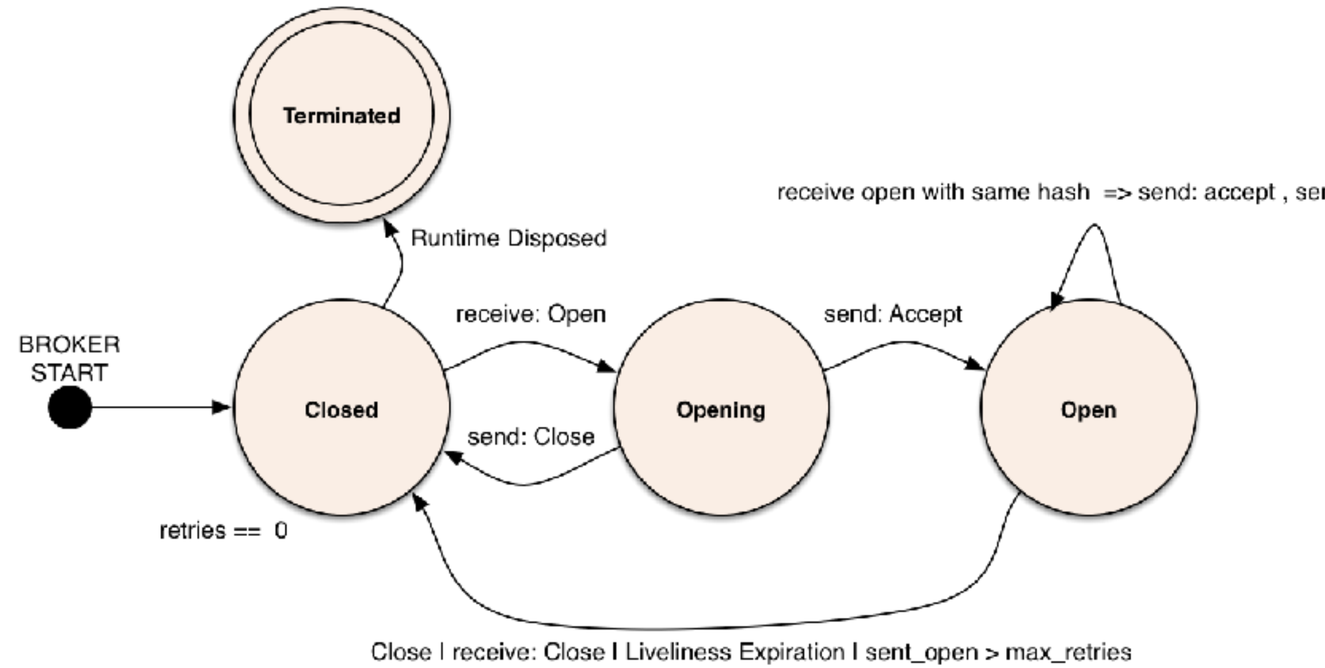
- **OPID** is the sending node's unique id
- **APID** is the other node's unique id
- **Reason** is a one byte code for the reason:
  - 0: success
  - 1: invalid authentication data
  - 2: unsupported protocol (version)
  - 3: out of resources
  - 4: unsupported sequence number length
  - 5: unsupported parameter (e.g., lease, locators, peer id, subscription mode)
  - 6: incompatible pre-committed declaration
  - 255: generic error

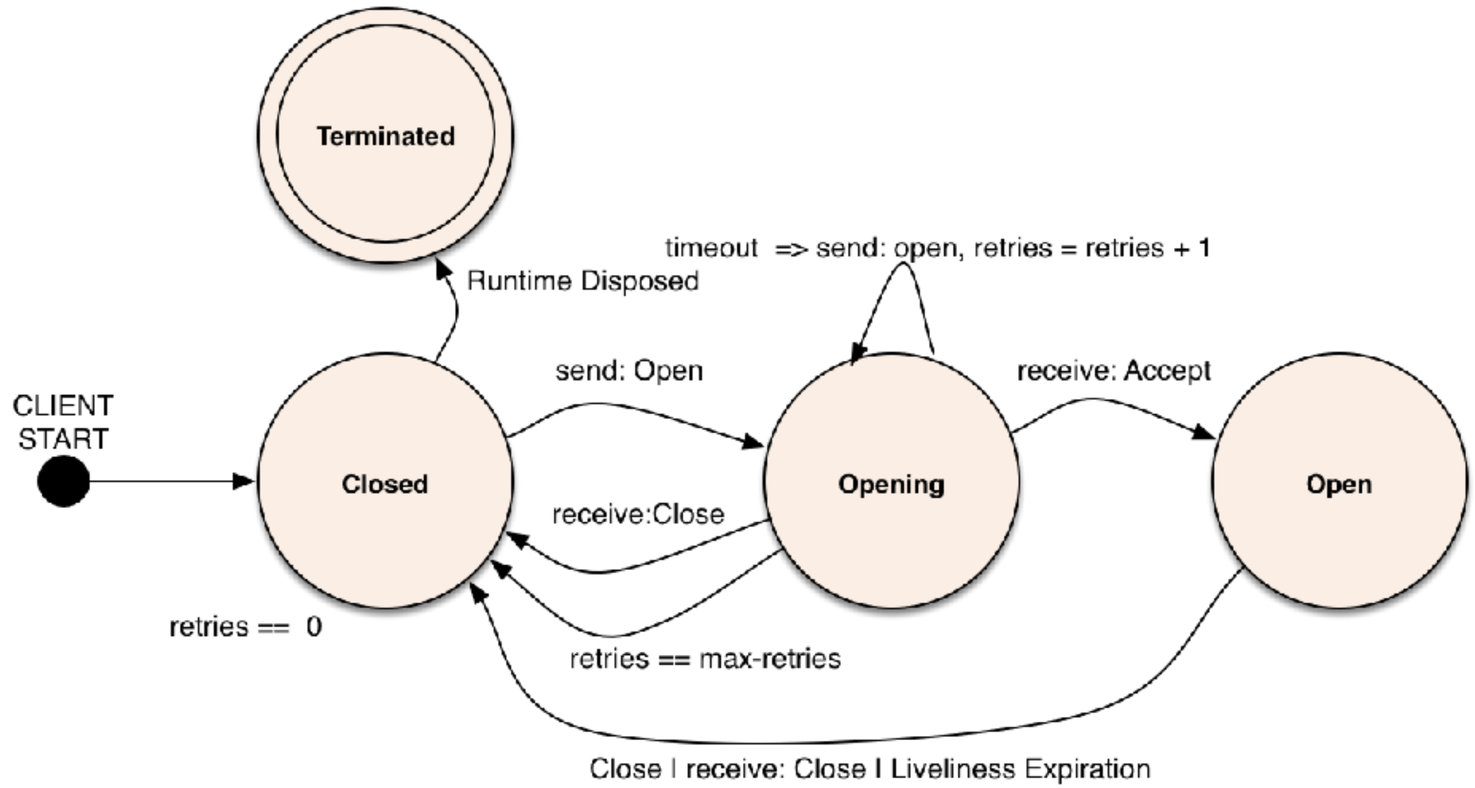


Session FSM

To reject a request for opening a session and to explicitly close an existing session, a Close message is used.

- PID is the sending node's unique id
- Reason is a one byte code for the reason





XRCE Protocol
DECLARATIONS

Declarations

Resources, **publications**, **subscriptions**, **selections** all need to be **declared** before one can publish or read/receive data.

Declarations are buffered until committed — unless a commit flag is specified.

Resources may be declared multiple times, but they may also be declared once and then used by everyone.

Combined with the numerical ids for the resources, this allows small clients to use only a few small messages to declare their publications/ subscriptions.

Resource IDs

Resources and **Selections** are **uniquely identified** by **numerical ids** to save space on the wire.

The relationship between a Resource ID and a Resource is established when declaring a resource. **Resource IDs are global**, everyone in the XRCE domain must use the same mapping of ID to Resource.

RIDs are positive, even integers.

Selection IDs

Subscribing to a trivial resource can be done using the Resource ID.

Subscribing to a more complicated selection requires choosing a Selection ID.

Applications can each define their own selections, so these are in principle local to the session.

This allows a broker to send just the data matching the selection to a client, relieving the client from knowing all the matching resource ids and performing the matching itself.

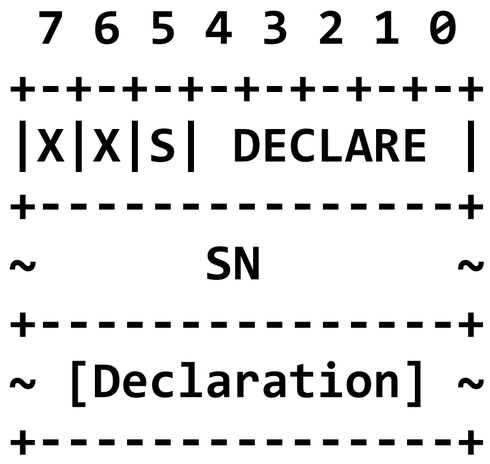
When a broker is pushing data to multiple clients the Selection IDs may be different even for the same selection. The broker would then have to send the data multiple times, even if the clients **support multicast**. Hence **we allow rebinding Selection IDs**.

Declare message

The general **Declare** message is a container for a list of Declarations.

- it is always reliable
- SN is sequence number

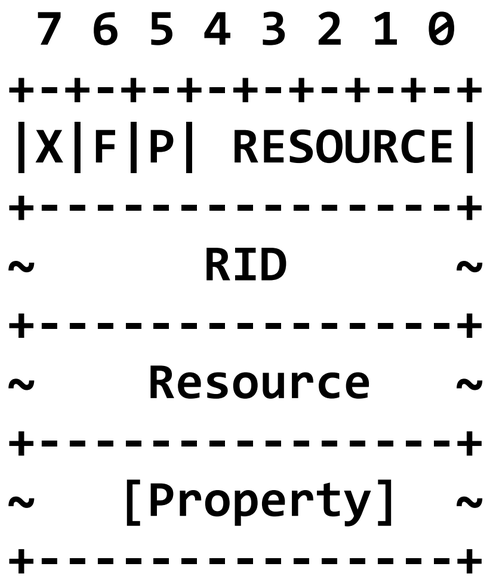
Note that it is very similar to regular data. The use of a separate message kind avoids the use of predefined resource ids.



Resource Declaration

This declares that (numerical) resource id RID corresponds to Resource. Resource need not be trivial.

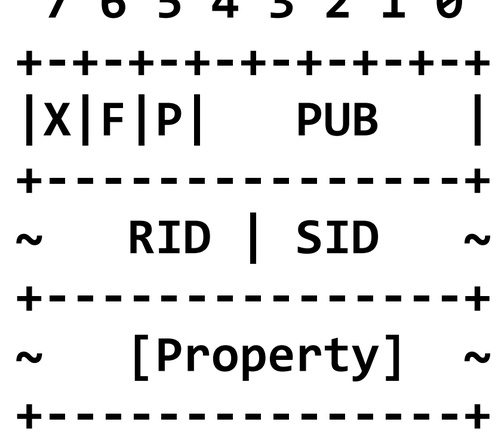
- **P=1** iff properties present
- **F=1** implies the resource should be forgotten
- The properties are a sequence of name-value pairs
- In the absence of properties, the resource has a default QoS, corresponding to DDS reliable, keep-last-1, volatile



Publication Declaration

This declares the sender as a publisher of values for the Resource/Selection identified by RID/SID, or cancels a previous declaration.

- **P** is set iff properties are included
- **F** is 0 if it is a declaration, 1 if it is the cancellation of a declaration



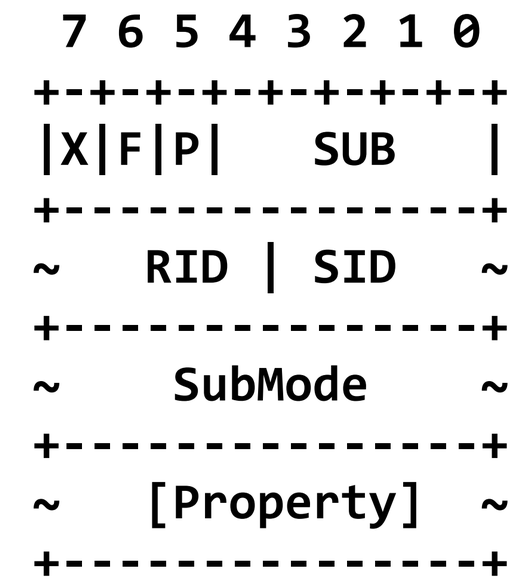
Subscription Declaration

This declares a subscription to the resource/selection identified by RID/SID.

SubMode determines whether it is a pull/push/ periodic pull/periodic push/push-pull subscriber.

Periodic push tells the publisher that the subscriber will listen for data every X seconds; periodic pull that it will request data every X seconds. Push-pull means a notification of data is pushed, but the data itself is not.

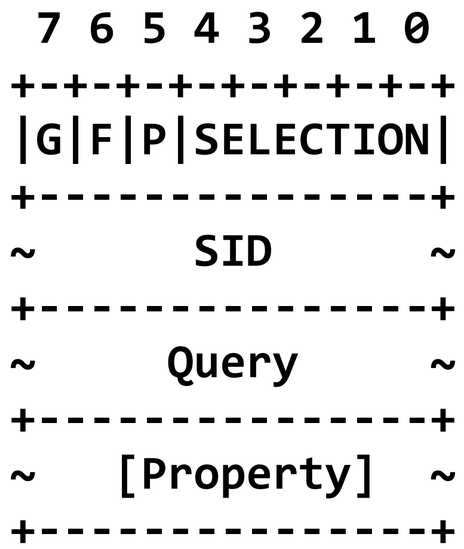
- P is set iff properties are included
- C is 0 if it a subscription is taken, 1 if it is being cancelled



Declare: Selection

This declares a selection (i.e., a query) and associates it with a Selection ID. To actually subscribe to a selection, one must still send a **Sub** declaration.

- P is set iff properties are given
- G is set iff SID is a global identifier for this selection, rather than one bound to this session

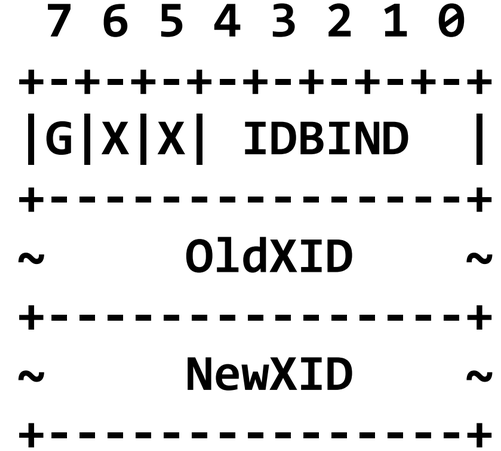


Declare: Selection ID binding

This defines a new binding for the given selection, i.e., an alias for the old selection ID.

This allows the sender to use NewXID instead of OldXID. Its primary purpose is to allow multicast push to clients that chose different IDs for the same selection.

- G is set iff the NewXID is a global id (similar to G in **Selection** declaration).



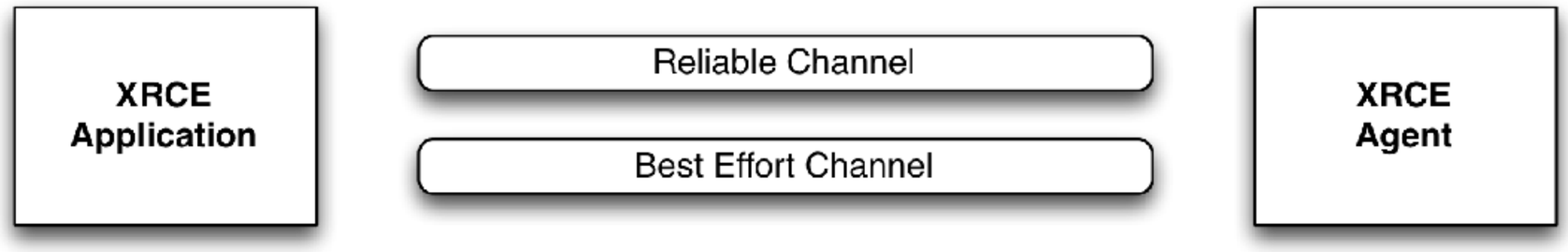
XRCE Protocol
Data Exchange

Conduits

All data messages and declarations are transported over a conduit, which is a pair of a reliable and a best-effort channel in a session.

Multiple conduits may be used in parallel, to avoid head-of-line blocking and to allow multicasting across session in addition within a session.

The default conduit 0 is used unless messages are prefixed with a special **Conduit** message.



Best-Effort Channel Specification

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

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

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:

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

where:

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

Best-effort Channel Implementation

The best-effort channel implementation is relatively straight forward.

The 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:

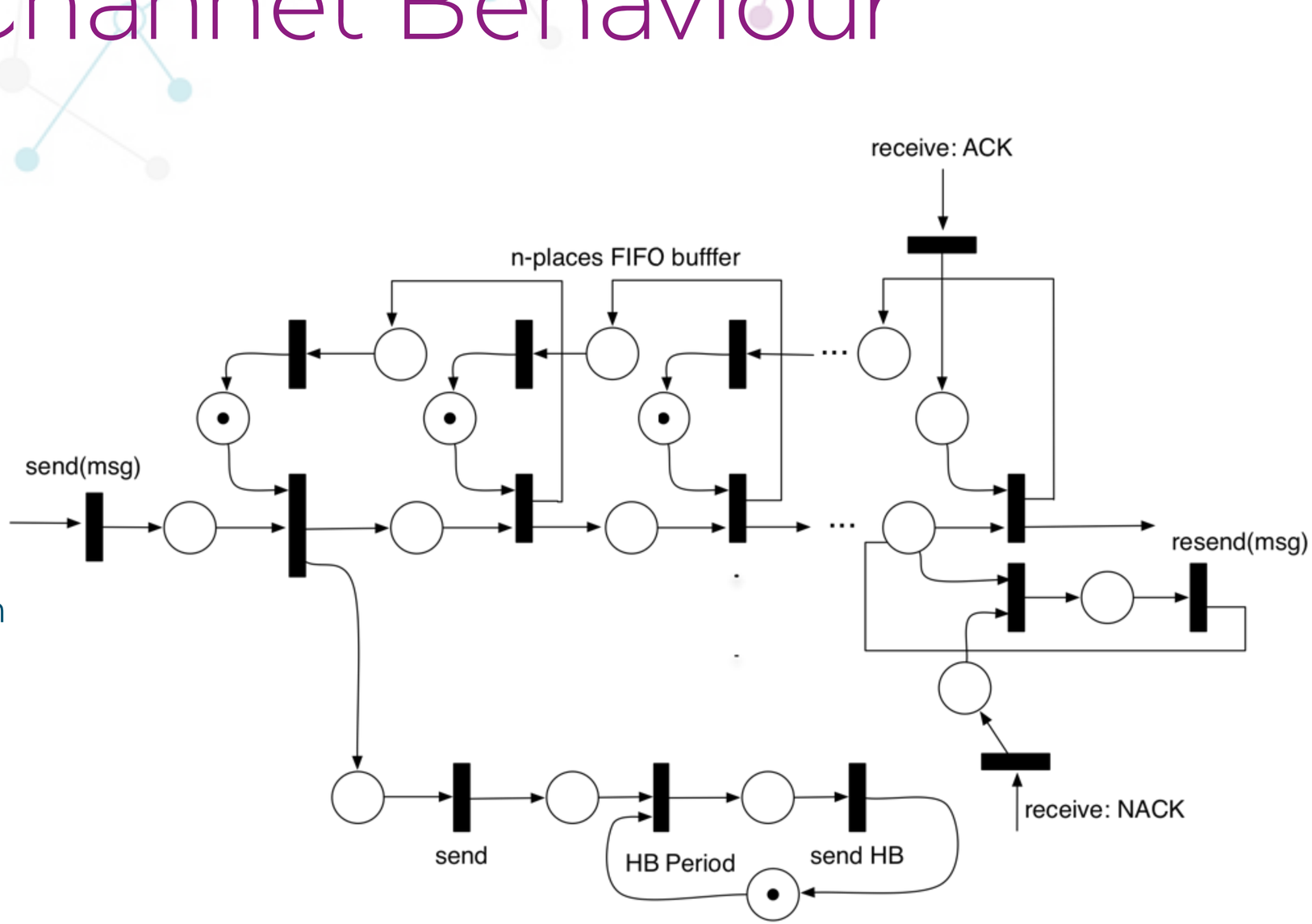
$$m_0, m_1, \ldots, m_k$$

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

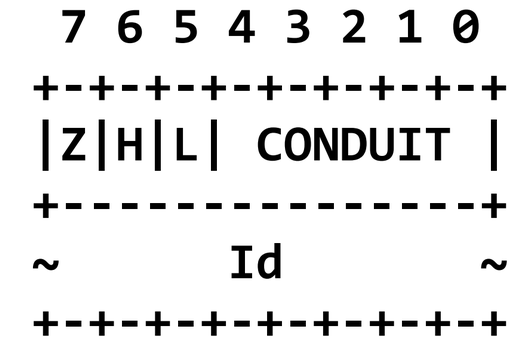
The Petri Net describes the behaviour of a Reliable Channel with a buffer of **n places**. It is worth to mention that the specification assumes that a NACK can only be received for the head of the queue. This may seem a restriction, but in reality it only means that when receiving a NACK for message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



Conduit selection

Conduit is used to specify that the next messages in the packet (or the next message for non-packet-based transports) are sent over conduit CID instead of over the default conduit.

Z=1 implies a compact representation where **HL** bits encode **Id-1**, i.e. 00 => Id = 1



Conduit Migration

```
 7 6 5 4 3 2 1 0
+-+-+-+-+-+-+-+-+
|X|X|X| MIGRATE |
+---------------+
|      cid      | -- the old conduit id
+-+-+-+-+-+-+-+-+
~    lastSN     ~ -- the sequence number
+-+-+-+-+-+-+-+-+    of the last sample for
                     the resource sent on the
                     old conduit
```

The migrate market is used to migrate a resource from a conduit to another while maintaining reliability and ordered delivery

Synchronising a Conduit

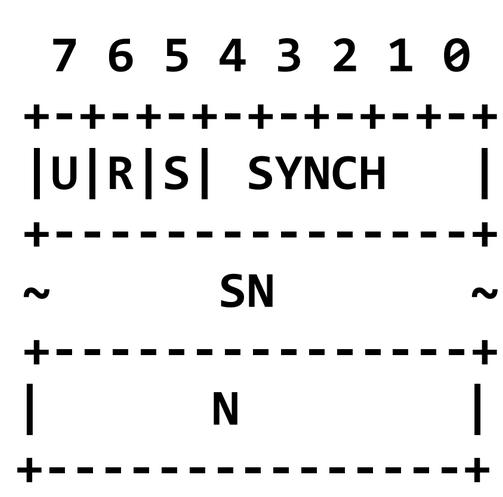
Synch is used to inform the recipient of the range of sequence numbers available for retransmission on the reliable channel of the conduit. It also allows informing a late joiner of the current sequence number in the channel.

- **R=1** indicates the reliable channels, otherwise the best-effort channel

- **SN** is the next sequence number used for a message on this channel.

- If **U=1** then **N>0** is the number of messages available

If **S=1** is set, requires an immediate **ACK/ACKNACK** response.



ACK/NACK

ACKNACK with **M=0** (no mask) is used to acknowledge all reliable messages on conduit CID up to but not including SN.

ACKNACK with **M=1** (mask included) is used to acknowledge all reliable messages on conduit CID up to but not including SN, and furthermore request a retransmission of message number SN and those for which a bit is set in Mask. Bit 0 in Mask corresponds to SN+1, Bit 1 to SN+2, &c.

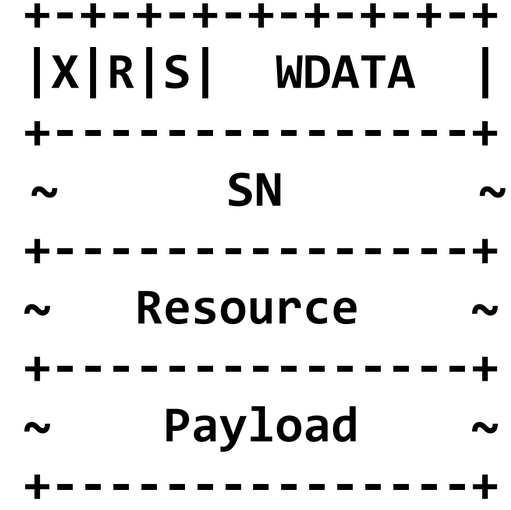
```
 7 6 5 4 3 2 1 0
+-+-+-+-+-+-+-+-+
|0|0|M| ACKNACK |
+---------------+
~       SN      ~
+---------------+
~      Mask     ~
+---------------+
```

Writing Data

One Shot Write

XRCE allows writing a resource w/o any kind of prior registration or discovery

The cost of this abstraction is that the full-resource name has to be sent on the wire



Streaming Write

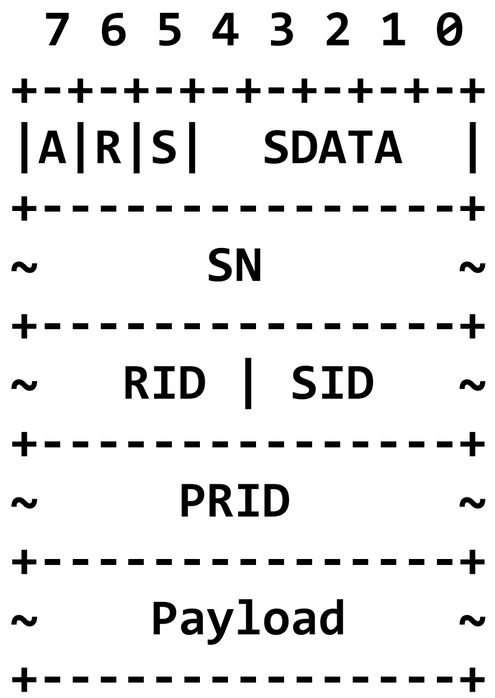
XRCE provides built-in support for very efficiently sending a sample or a batch of samples. Notice that for a single sample the **minimum overhead** is of only **3 bytes protocol headers and 1 byte message length => 4 bytes minimum overhead**

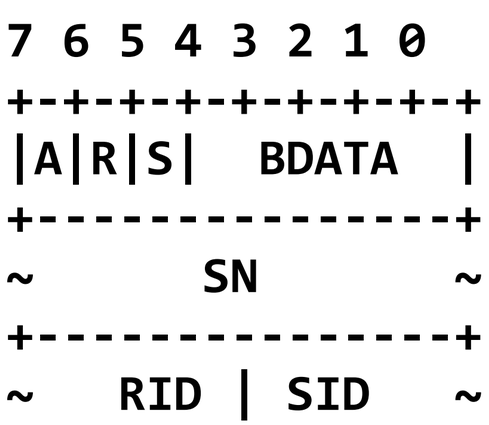
- **A** is set iff PRID is present

- **PRID** is the ID of the trivial resource for which this is a value, for cases where a selection is used, so that the subscriber knows the resource the data belongs to

for BDATA, P set => a sequence of (RID, Payload) pairs, else just a sequence of Payloads.

Note that these forms are the only messages for streaming data, and are also used for the response to a pulling read.



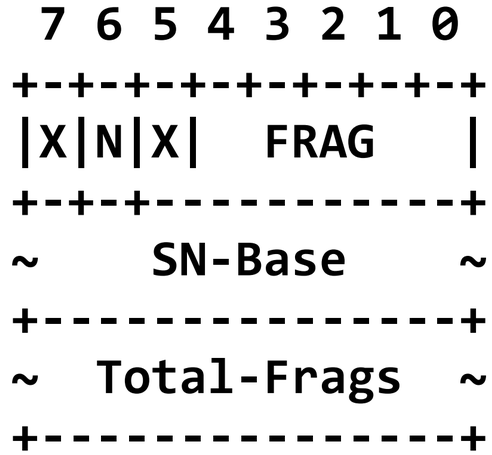


Fragmentation

XRCE needs to support small MTUs, therefore, it needs to automatically fragment larger payloads.

Fragmentation in XRCE is done by prefixing a **WDATA** or **SDATA** with a small message (in the same packet)

- SN-Base is first sequence number of fragmented payload, present only if N=1
- N=1 indicates that the total number of fragments following this packet is known.
- fragments must be ~MTU large



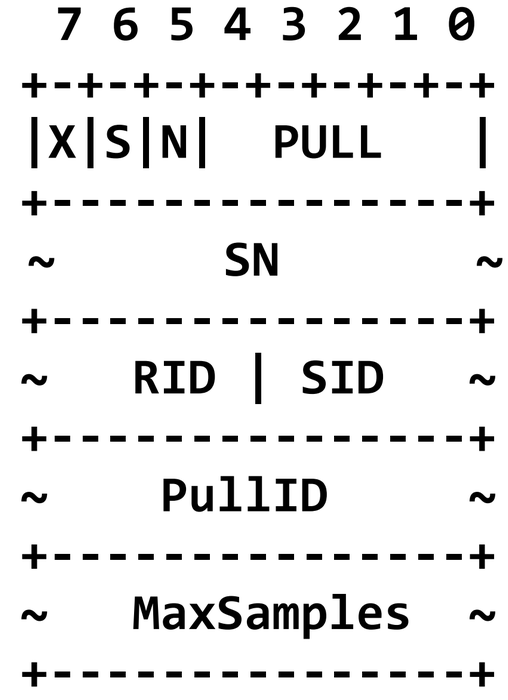
Reading Data

Pulling Data

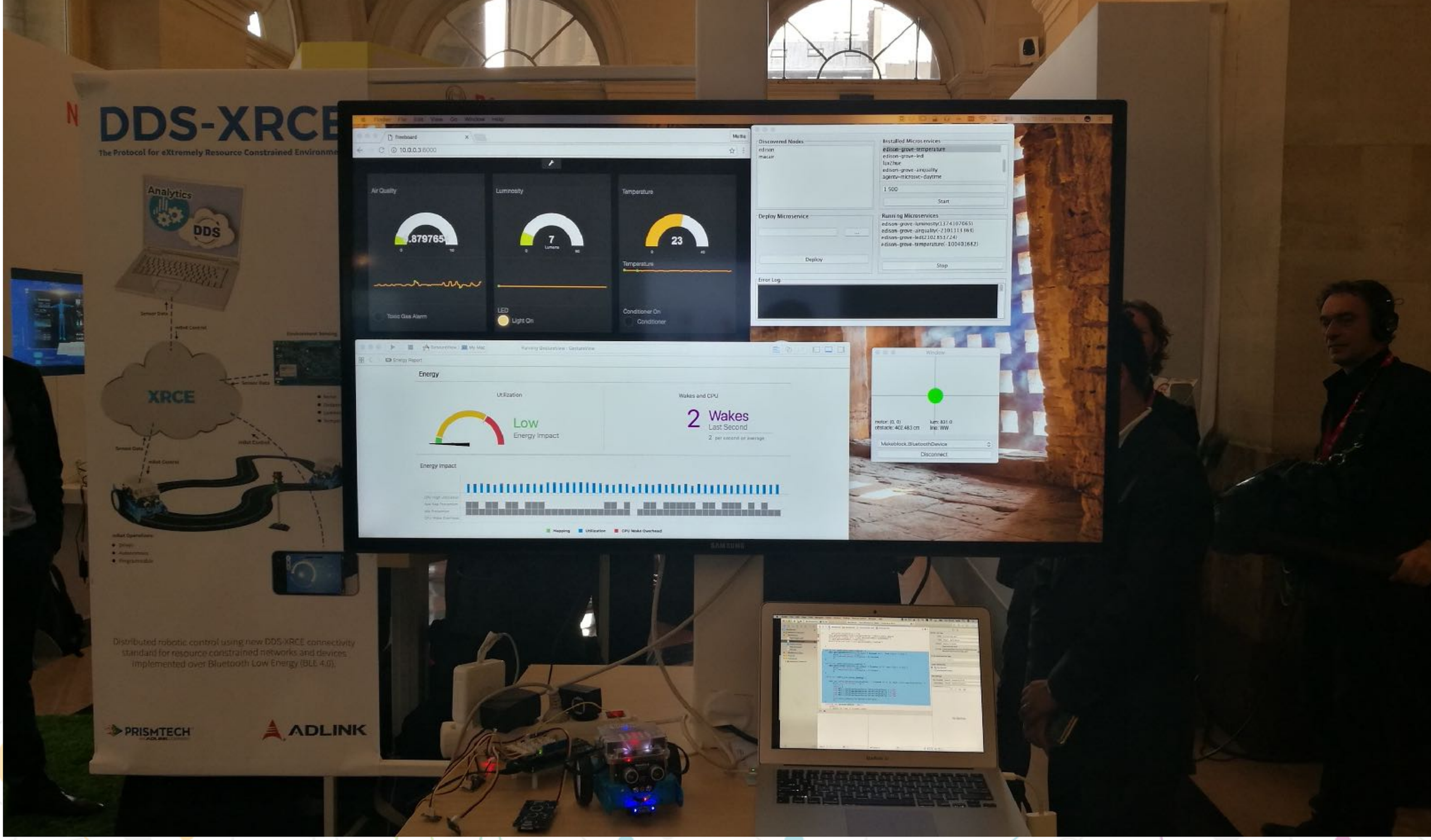
XRCE supports both push and pull modes. In a pull mode, the broker maintains the data and the client explicitly pulls the data towards it.

- **Pull** is always reliable
- N is set iff PullID and MaxSamples present
- PullID is “like” a cursor in an RDMS, MaxSamples is the maximum number of samples to be returned at a time. A second **Pull** with the same ID will continue enumerating data from the set.

The combination of PullID and N allow allowing iterating through large data sets in nested loops.



XRCE @ Work


DDS-XRCE
The Protocol for eXtremely Resource Constrained Environme
Analytics
DDS
XRCE
Distributed robotic control using new DDS-XRCE connectivity standard for resource constrained networks and devices implemented over Bluetooth Low Energy (BLE 4.0).
PRISMTECH
ADLINK
Air Quality
.879765
Luminosity
7
Lumens
Temperature
23
Toxic Gas Alarm
LED
Light On
Conditioner On
Conditioner
Discovered Nodes
edison
macair
Installed Microservices
edison-grove-temperature
edison-grove-led
lux7hor
edison-grove-airquality
agent-microsvc-daytime
1 500
Start
Deploy Microservice
Deploy
Running Microservices
Stop
Error Log
Energy
Utilization
Low
Energy Impact
Wakes and CPU
2 Wakes
Last Second
Energy Impact
Disconnect

Environment Sensing

- Noise
- Distance
- Luminosity
- Temperature

Toxic Gas Alarm

Luminosity

11 Lumens

LED

Light

Temperature

23

Conditioner On

Conditioner

Discovered Nodes

edison

macair

Deploy Microservice

Deploy

Error Log

GestureView My Mac

Running GestureView : GestureView

Energy Report

Energy

Utilization

Low

Energy Impact

Wakes and CPU

4 Wakes

Last Second

2 per second on average

Utilization

CPU Wake Overhead

SAMSUNG

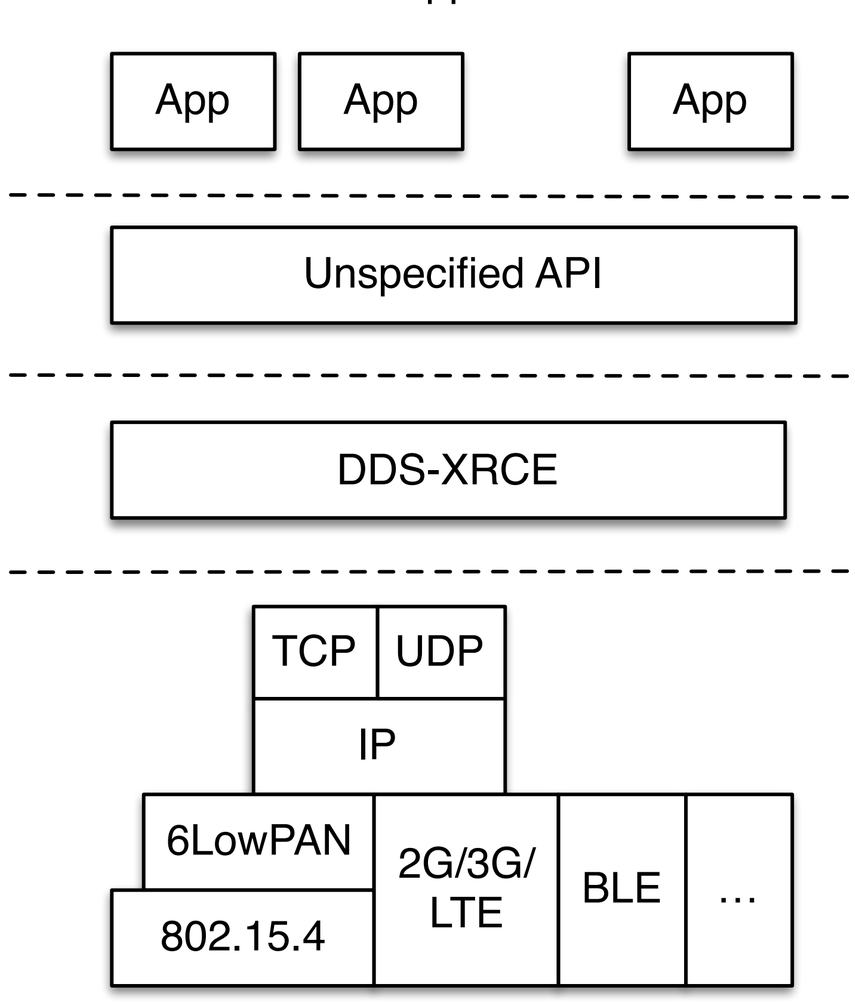
Live Demo!

Summing Up

DDS-XRCE

eXtremely Resource Constrained Environments DDS (DDS-XRCE)

Defines the **most** wire/power/memory **efficient protocol in the market** to provide DDS **connectivity to extremely constrained targets**

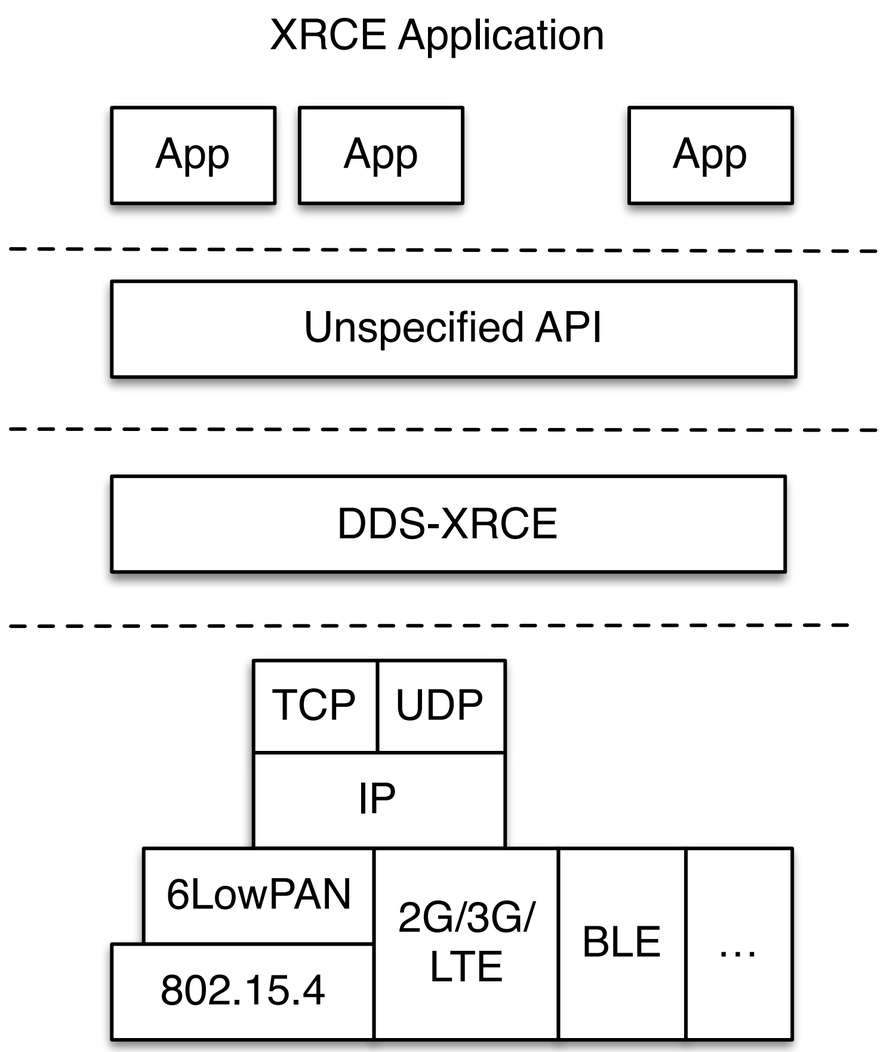


DDS-XRCE

Supports peer-to-peer as well as broker-based communication

Provides reliability and fragmentation above packet-oriented best-effort transports

Can leverage multicast



DDS-XRCE

Current prototype runs on **8-bit micro-controllers** and takes in **1 KByte of RAM** and has **wire-overhead** of **4 bytes** for data samples