# RTI DDS

# QOS策略和特性

Recommend
火龙果软件

# 服务质量策略 (QoS)

- 定义:
  - 应用程序可以使用QOS策略来控制、管理和优化在网络中传输的数据流。
- 功能:
  - **DDS标准提供的 QOS参数可以对数据流进行常规控制：**
    - 数据的持久度
    - 数据的传输方式
    - 容错性
  - **DDS提供商可以提供扩展的QOS策略以实现更深层次的控制**
    - 对发现过程控制
    - 对系统使用的资源控制（如限制使用资源的大小，对传输途径配置等）

Recommend 火龙果软件

© 2009 Real-Time Innovations, Inc.

# QOS的策略

- 可靠性，Reliability
- 数据的持久度，Durability
- 数据的历史记录，History
- 周期数据的超时，Deadline
- 基于时间的过滤，Time-based filter
- 数据的有效期，Lifespan
- 数据的所有权，Ownership
- 分区，Partition
- 资源限制，Resource limits
- 等等……

Recommend
火龙果软件

© 2009 Real-Time Innovations, Inc.

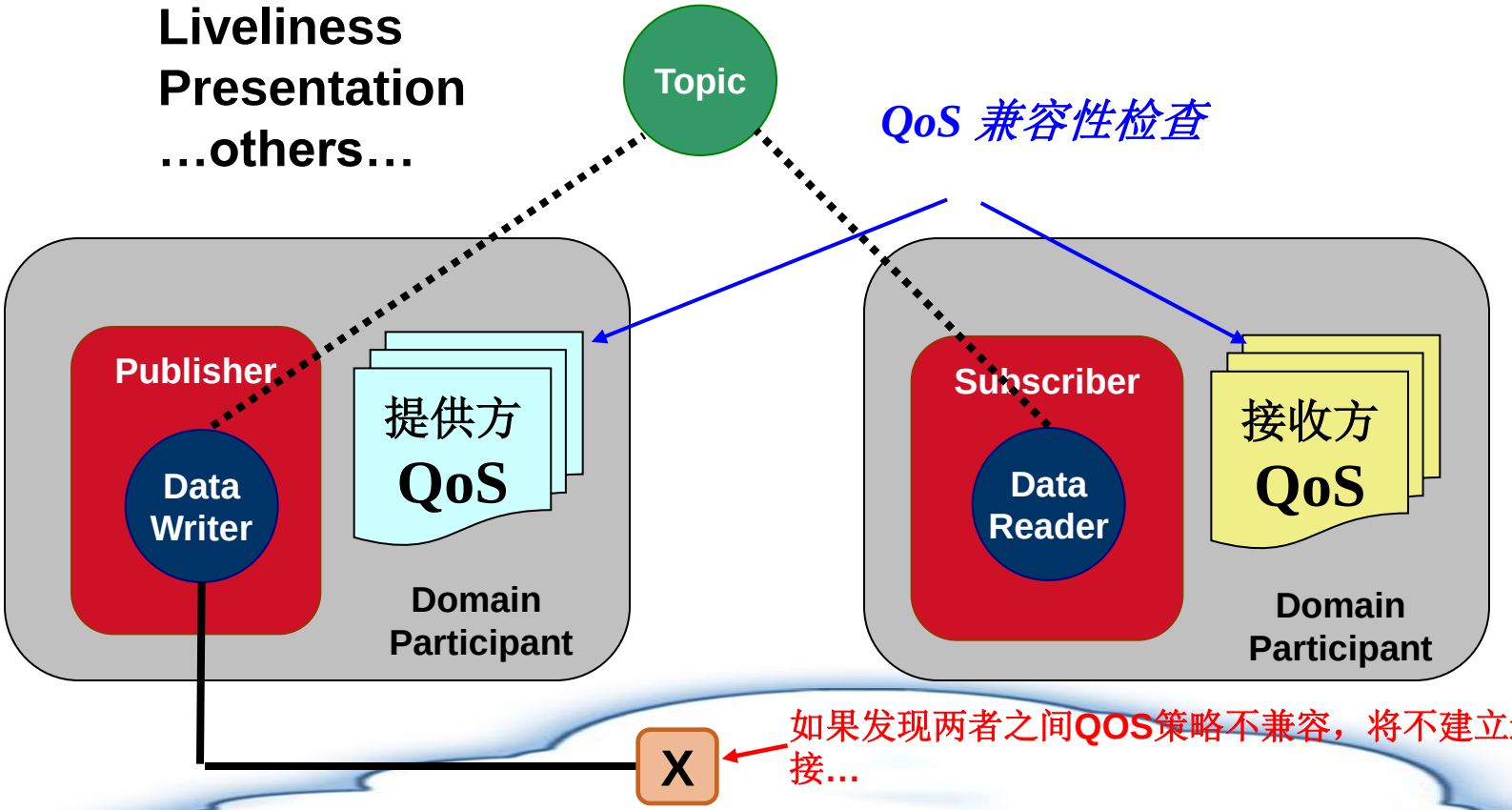
- Deadline
- Destination Order
- Durability
- Entity Factory
- Group Data
- History
- Latency Budget
- Lifespan
- Liveliness
- Ownership
- Ownership Strength
- Partition
- Presentation
- Reader Data Lifecycle
- Reliability
- Resource Limits
- Time-Based Filter
- Topic Data
- Transport Priority
- User Data
- Writer Data Lifecycle

Recommend
火龙果软件

# QOS可以理解为数据提供者和接收者之间的合约

**Reliability**
**Durability**
**Deadline**
**Liveliness**
**Presentation**
**…others…**

必须确保收发双方的QOS策略相互兼容




© 2009 Real-Time Innovations, Inc.

# 不同的设计模式考虑

- 按数据传输的类型考虑（Data Delivery）
  - 周期性数据（Periodic）
  - 状态和位置数据（State and Status）
  - 时间和命令（Events and Commands）
- 按高有效性考虑（High Availability）
  - 冗余性(Redundancy)
  - 从故障中恢复(Failover)
  - 确保的信息数据(Guaranteed messaging)
- 按安全性考虑（Security）
- 按可测量性考虑（Scalability）

Recommend 火龙果软件

© 2009 Real-Time Innovations, Inc.

# 设计模式-根据被传递的数据特性

- 持续的传感器数据
  - 数据定期更新
  - 有多个订阅者接收相同的数据
  - 在存在冗余数据出版者情况下，可以无缝地从故障中恢复
- 状态和位置信息
  - 偶尔会发生变化的连续的数据
  - 新加入网络的订阅者需要获取最后发布的数据以明确当前的状态或位置信息
- 命令和时间数据
  - 是异步的消息
  - 需要高可靠性，不能出错或误报

Recommend 火龙果软件

© 2009 Real-Time Innovations, Inc.

# 数据传递 – 连续的传感器数据

*服务质量策略*

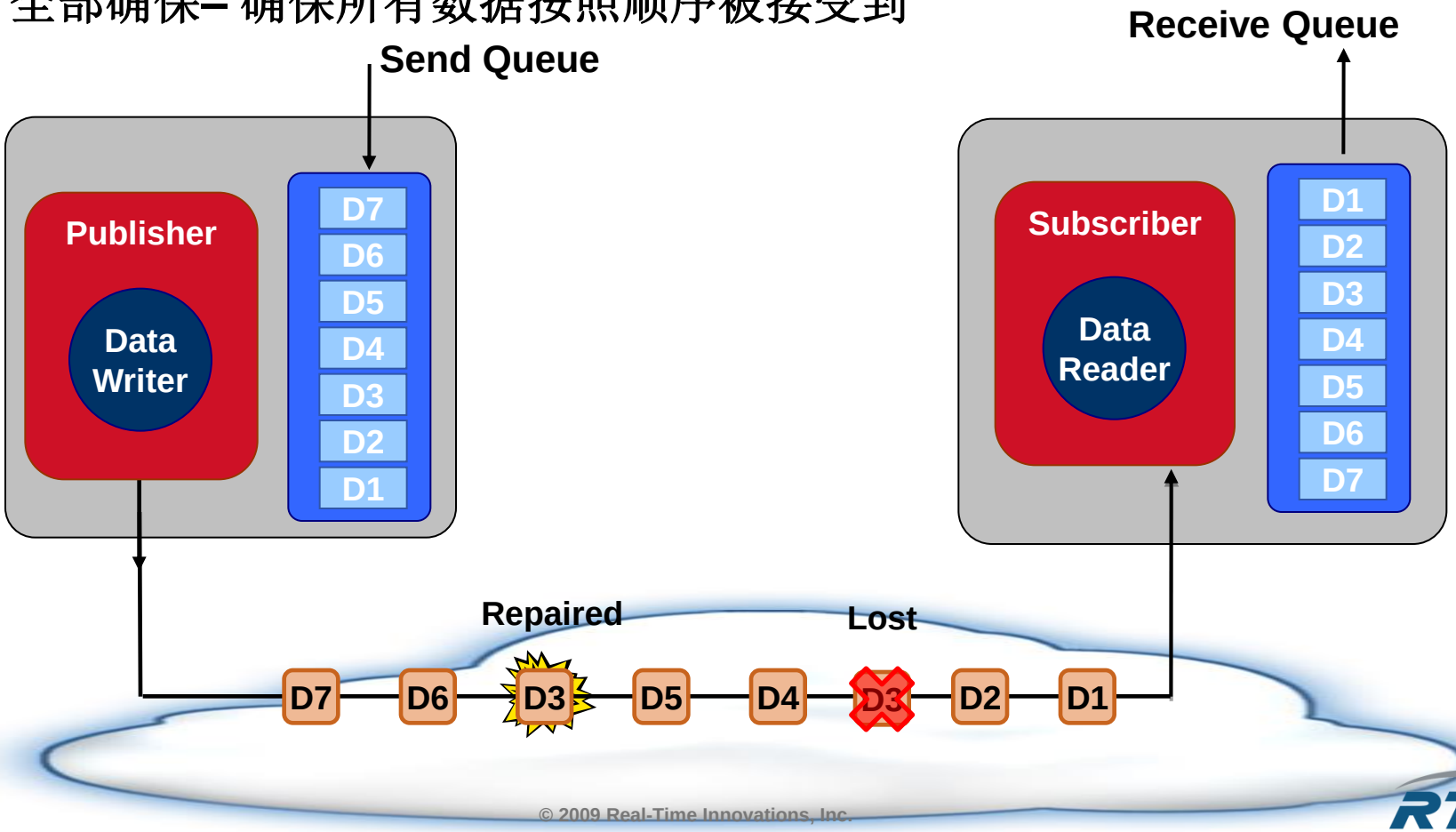
- 最可靠与最高效（Reliability - Best Effort）
  - 在没有高可靠性需求的应用中，减少额外的开销
- 截止时间（Deadline）
  - 确保期望得到的数据和实际收到的数据在*时间有效期*上是相符的
- 基于时间的过滤（Time-based Filter）
  - 在数据流中采样，而不是全部发送或接收
- 多地址广播方式发送（Multicast addressing）
  - 数据可以被所有订阅者接收

Recommend 火龙果软件

© 2009 Real-Time Innovations, Inc.

# QoS: 可靠性（Reliability）

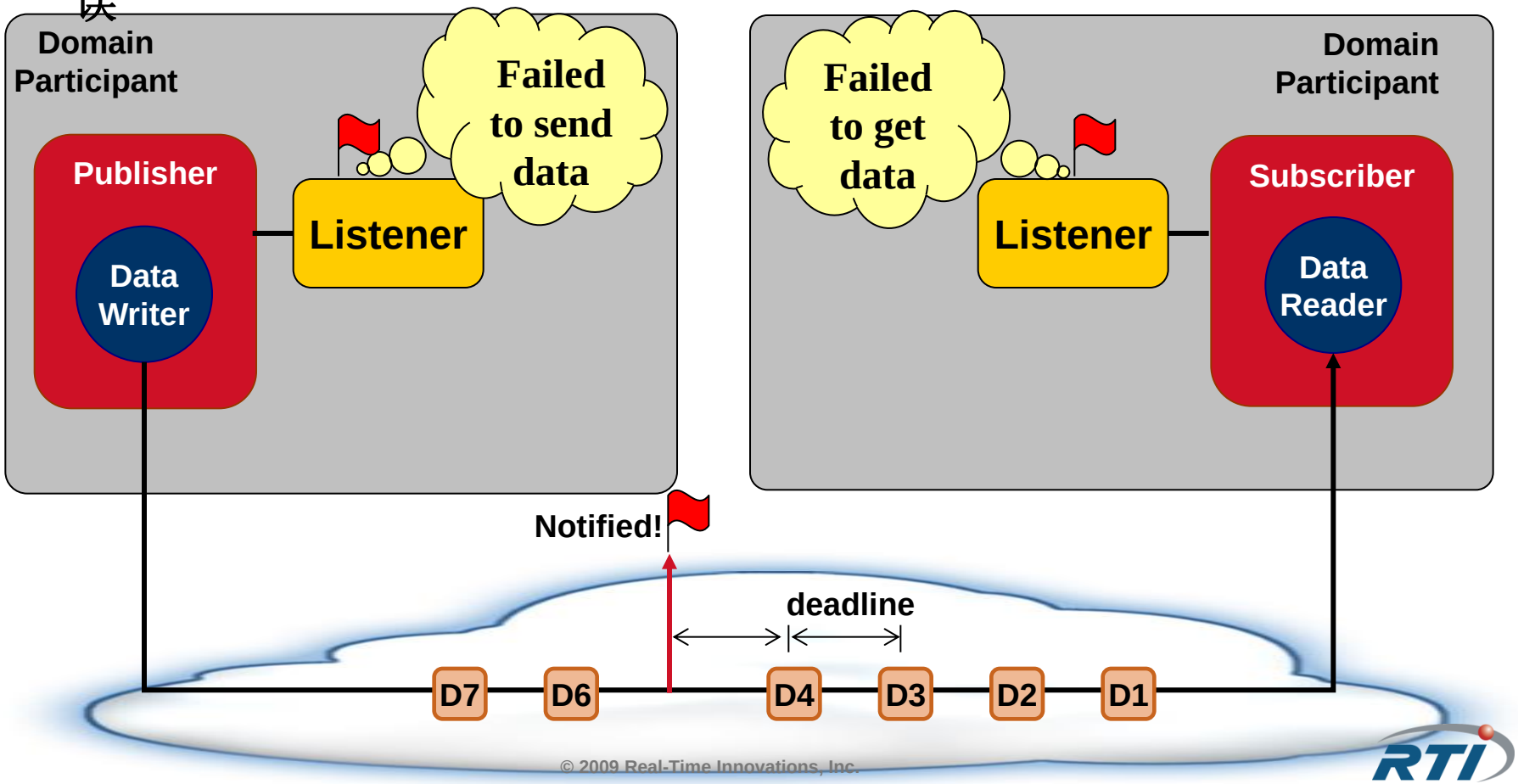
- 最高效 –不保证数据是否能被接收到（但是可以确保数据到达的先后顺序）
- 确保最后N个数据 – 发送所有数据，但仅仅确保最后N个数据被收到
- 全部确保– 确保所有数据按照顺序被接受到




© 2009 Real-Time Innovations, Inc.

# QoS: 底线/截止时间（Deadline）

- 数据发送者承担在每一个数据周期的截止时间内发送数据的责任
- 数据接收者希望在在每一个数据周期的截止时间内接收到数据
- 保证接收者的deadline >= 发送者的deadline，否则产生不兼容错误





© 2009 Real-Time Innovations, Inc.

# QoS: 生命期/寿命（LIFESPAN）

- 防止传输过期数据
- 写数据者，每个数据都附有保质期（expiration time）
- 读数据者，直接删除过期数据
- 要求发送者与接收者时钟足够同步。如果服务发现不同步，数据读取者可以用接收时间戳计算保质期，而不是用源时间戳。

Recommend
火龙果软件

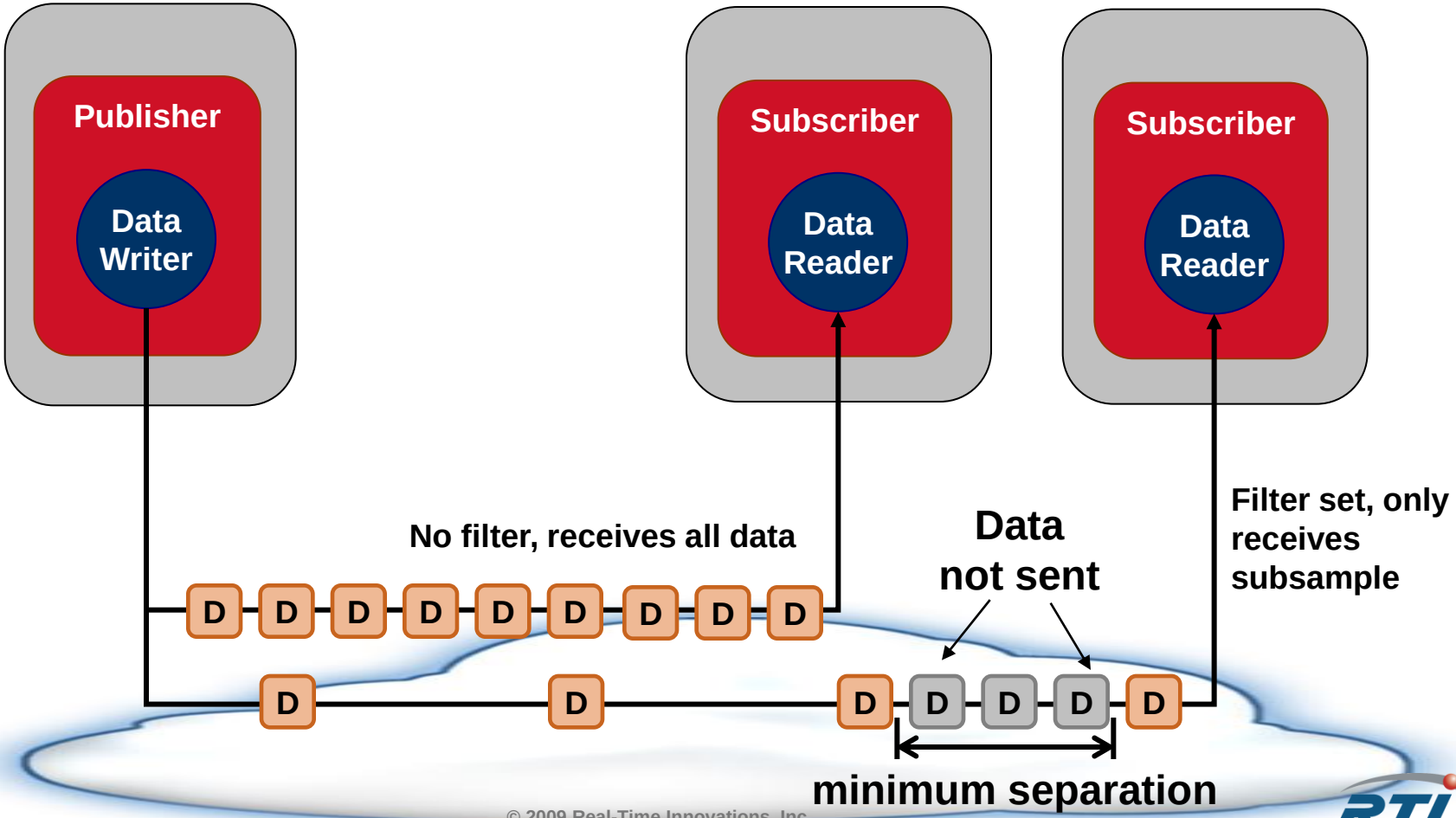
# QoS：资源限制（RESOURCE_LIMITS）

- 用服务控制资源以达到应有和其他QoS的强制需求
- 如果写数据者发送速度大于读数据者读取速度，中间件就会与QoS强制资源限制冲突。
- 取决于RELIABILITY QoS
  - BEST_EFFORT：允许放弃样本。（drop）
  - RELIABLE：阻塞写数据者，或者丢弃样本。（discard）

Recommend
火龙果软件

# QoS: 基于时间的过滤（Time-Based Filter)

- 数据接收者设定期望得到新数据的最小时间间隔
- DDS在满足上述条件的情况下，传送最小量的数据

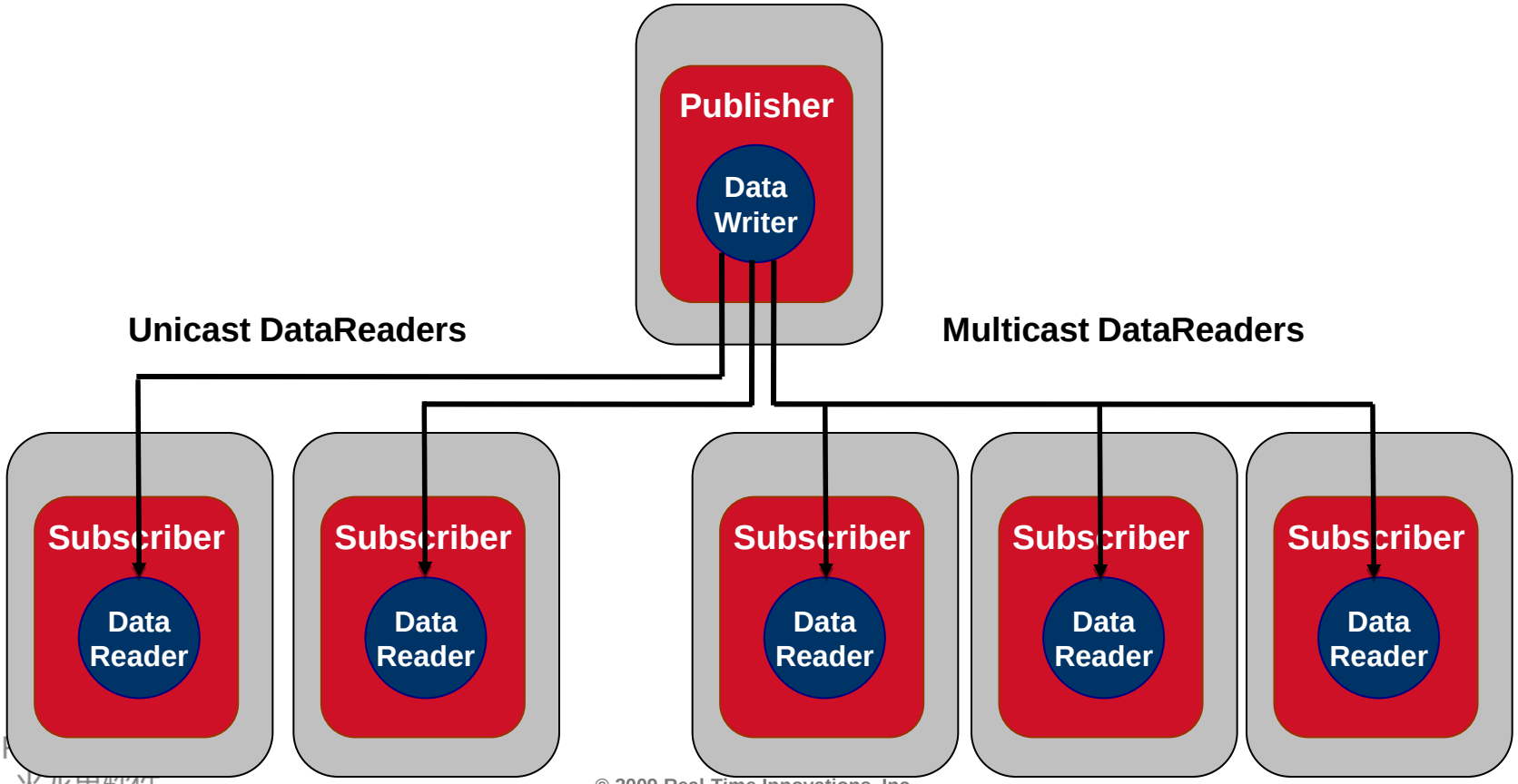




© 2009 Real-Time Innovations, Inc.

# QoS: 多播方式传送（Multicast Addressing）

- 在多个应用程序订阅相同的主题时，达到网络上传输最少数据包的效果
- 数据接收者定义地址
- 数据发送者可以支持以混合方式（同时单播和多播）发送数据





火龙果软件
© 2009 Real-Time Innovations, Inc.

# 数据传送 – 位置和状态

*服务质量策略*

- 历史记录（History）
  - DDS保存多少个历史数据
- 可靠性– 确保最后一个数据可靠（KEEP LAST 1)
  - 最后的数据是最重要的
- 持久性（Durability）
  - 当新的订阅者加入时，自动将之前最后的状态数据发送给新订阅者
- 数据自动保存到相关的数据库中


© 2009 Real-Time Innovations, Inc.

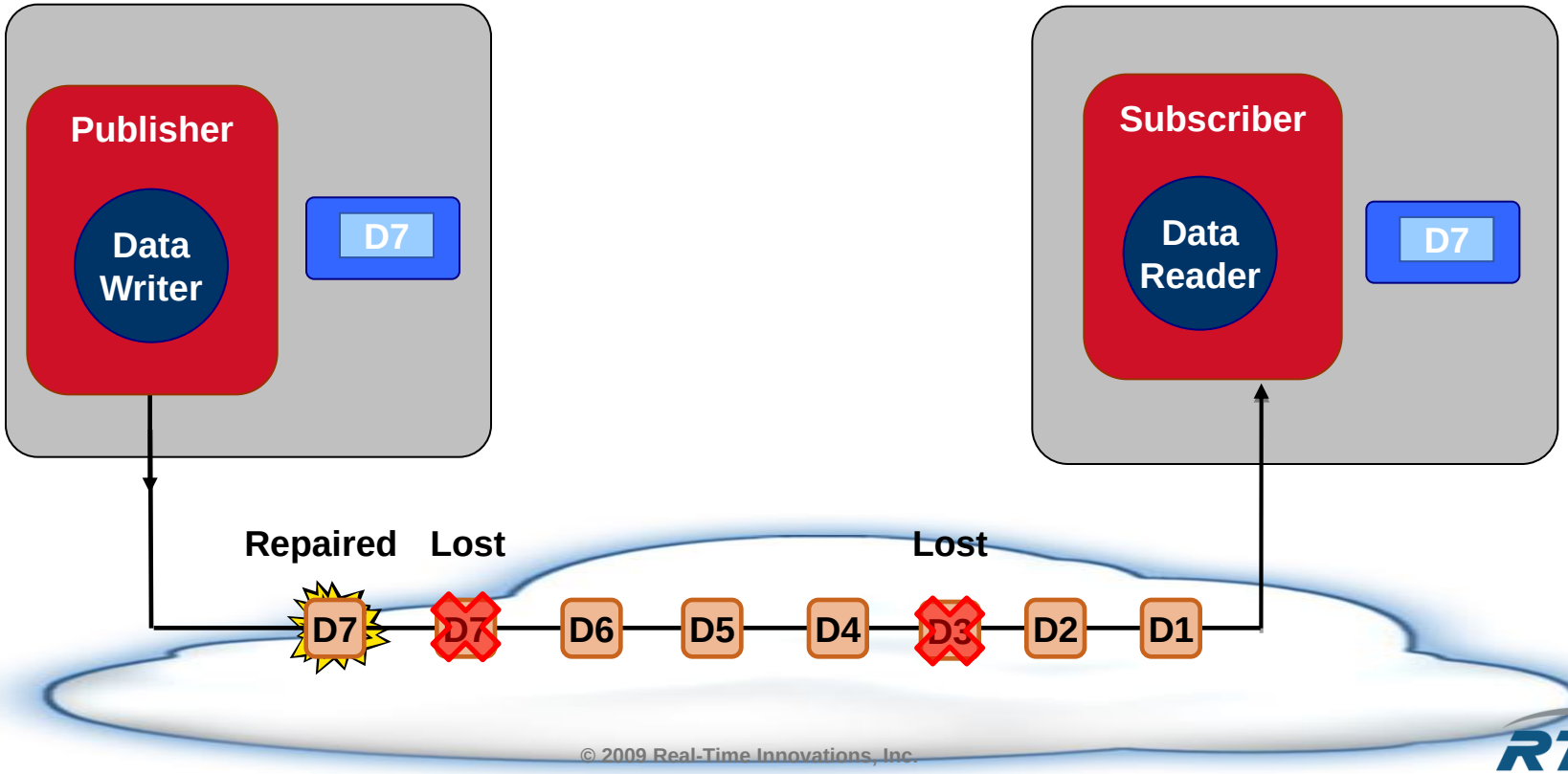
# QoS: 历史记录（History）

- 应用于数据发送者或接收者
- 是设定如何使用发送/接收队列的策略
  - 全部保持（KEEP_ALL）：在可用资源的限制内，在队列中保存所有的数据
  - 保持最后N个（KEEP_LAST N）:这是队列可以看成一个大小为N的循环缓冲区（ring buffer）
- 可靠性（ Reliability）和持久性（ Durability）QOS策略可以相互配合
- 使用可靠（ RELIABLE）和全部保持（ KEEP_ALL ）， 数据在发送队列中保持直到数据被全部数据接收者确认接收
  - 在发送队列满的情况下，数据发送者再发送数据时会被阻塞直到之前的数据被确认接收
- 使用可靠（RELIABLE） 和保持最后N个（KEEP_LAST N ） …
  - 仅仅最后N个数据会被确保可靠地传送
  - 数据发送者不会被阻塞


© 2009 Real-Time Innovations, Inc.

# QoS: 可靠性+ 保证最后1个数据

- 可靠性及保证最后一个数据的历史记录策略
  - 所有的数据都会被发送
  - 最后一个数据确保可靠地传送
  - 其他的数据使用最高效的方式传送

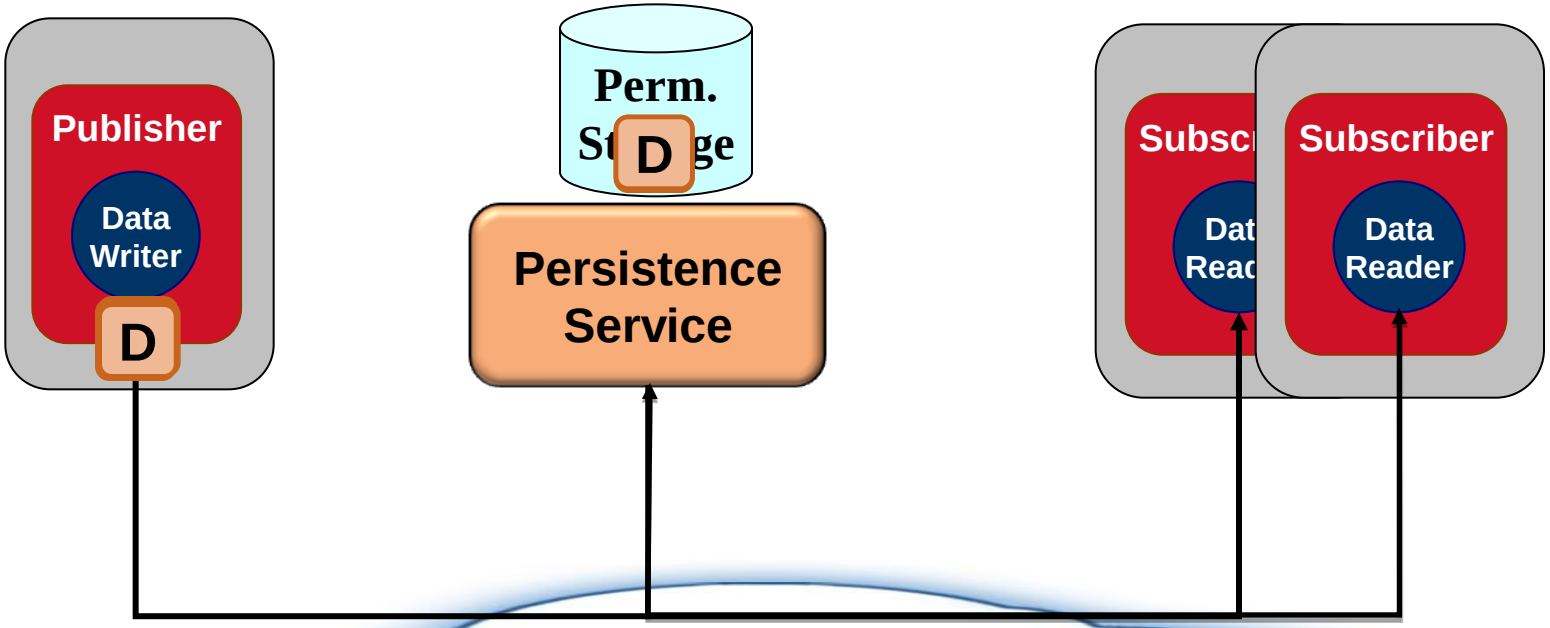



© 2009 Real-Time Innovations, Inc.

# QoS: 持久性（Durability）

易失的（VOLATILE）– 发送的数据不被保存
本地临时保存（TRANSIENT_LOCAL）– 数据发送者自己保存发送过的数据
临时保存（TRANSIENT）– 发送过的数据被保存在内存中
永久保存（PERSISTENT）– 发送过的数据被保存在永久性的存储器中

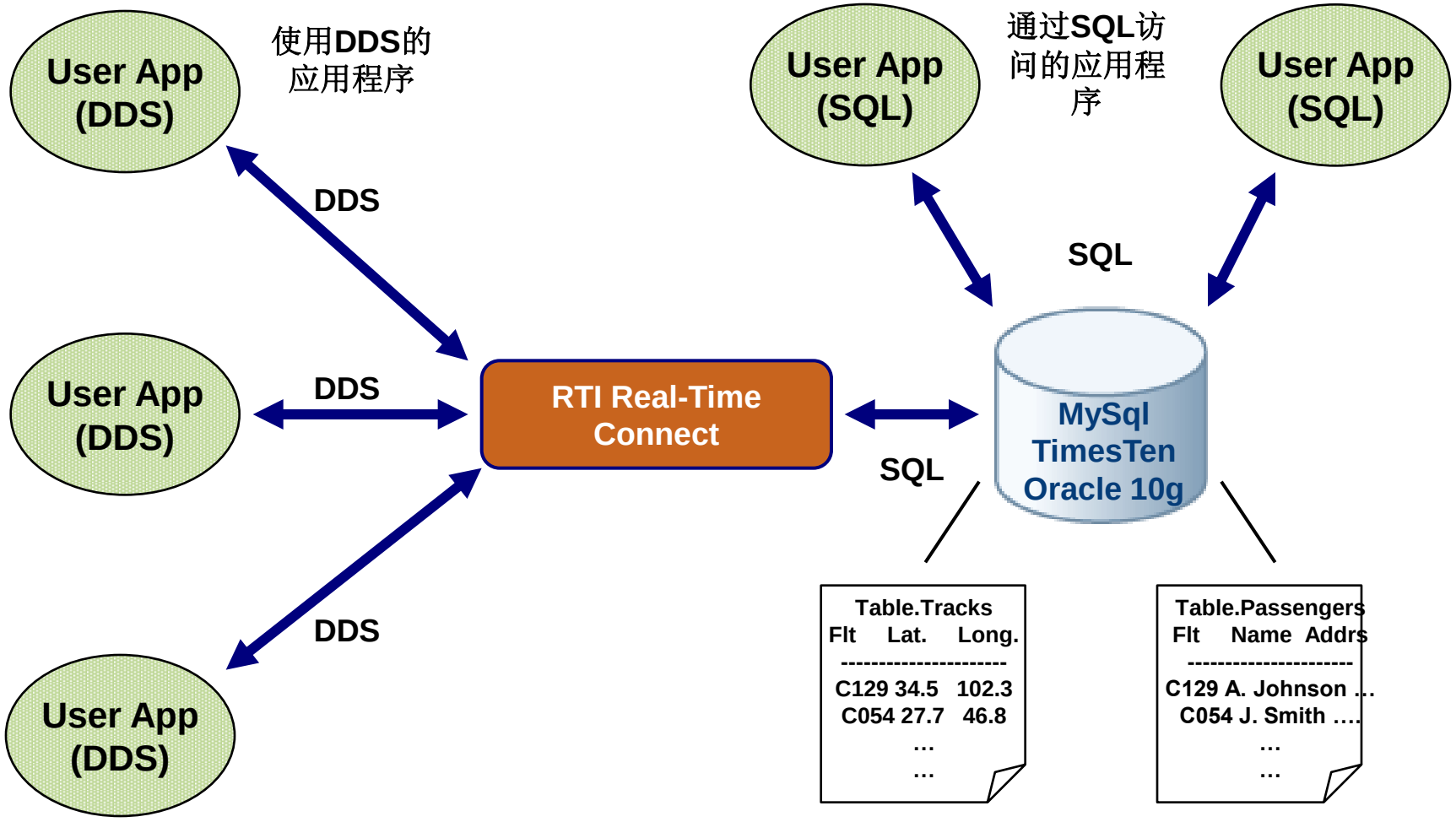
*给新的数据接收者提供一份之前已经发送过的数据拷贝*




持久性 = 永久保存策略


© 2009 Real-Time Innovations, Inc.

# RTI Real-Time Connect – 实现DDS与数据库的双向集成




**Real-Time**

**Web/Enterprise**

Recommend 火龙果软件



© 2009 Real-Time Innovations, Inc.

# 数据传送 – 时间和命令类型

*可用的QOS策略*

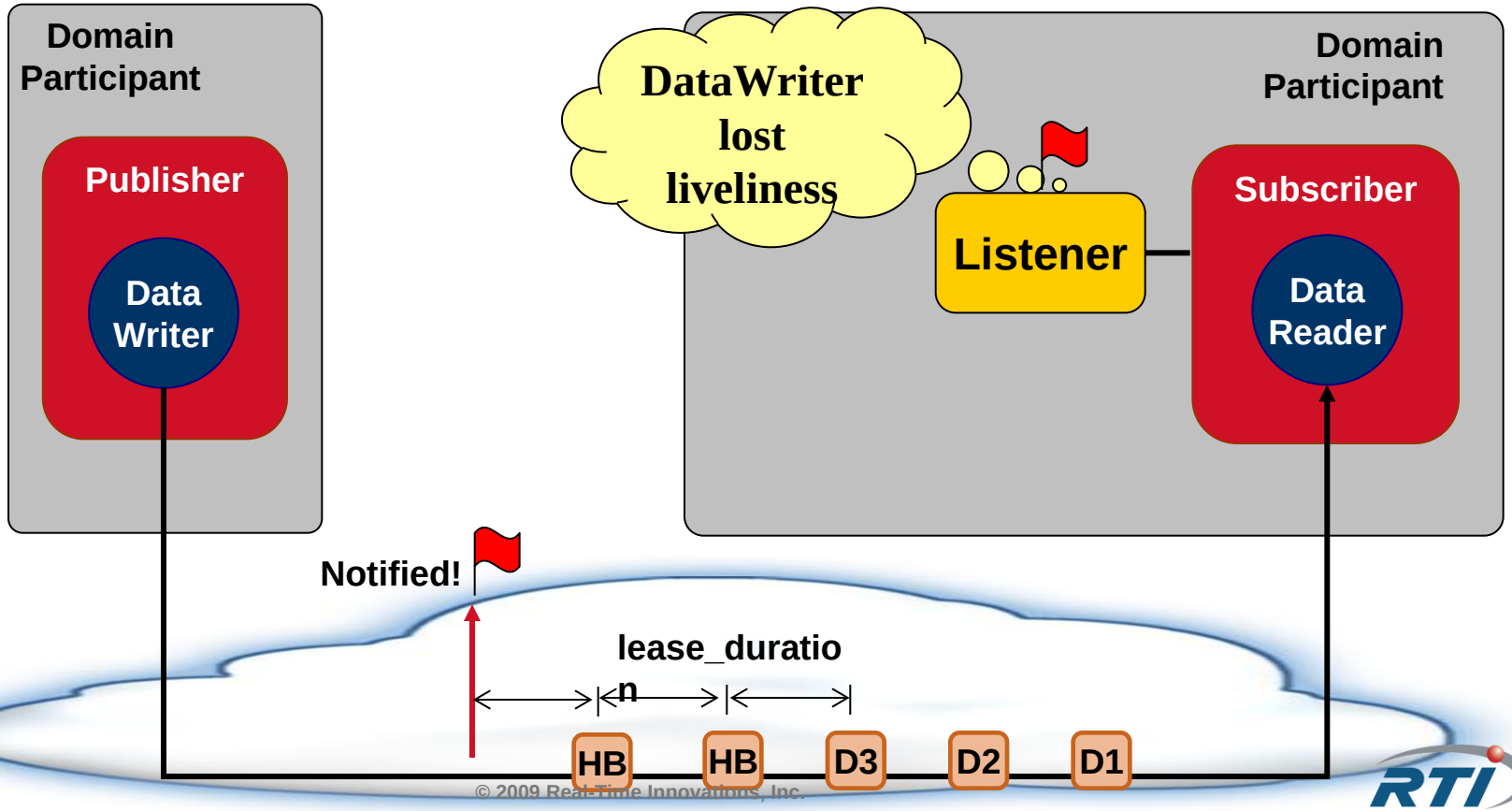
- 可靠性（最可靠）+ KEEP_ALL的历史记录，确保所有的数据均被可靠的传送
- 存在性（Liveliness）
  - 必须要能及时检测出数据发送者何时失效，那怕是在数据发送者不发送任何数据的情况下

Recommend 火龙果软件

© 2009 Real-Time Innovations, Inc.

# QoS: 存在性（Liveliness）

- 是约束数据发送者在约定周期内发送数据包或存在性包（心跳包）的合约
- 数据发送者的存在性包不需要数据发送者主动地发送（DDS发送）





© 2009 Real-Time Innovations, Inc.

# 设计模式- 高有效性

- 冗余
  - 同时使用多个传输途径或网络接口卡（NICs)
  - 重复的数据会被丢弃
- 从故障中自动恢复
  - 数据主题的所有权 （Ownership of Topics）
  - 根据所有权的权重来仲裁
  - 当主题拥有者失效时，自动恢复
- 被保证的消息
  - 在数据传输前，使用永久性的存储设备保存消息
  - 在失效的进程重新工作后，之前没有被确认的消息会被再次发送
  - 由外部服务来储存和转送


© 2009 Real-Time Innovations, Inc.

# 高有效性 – 冗余

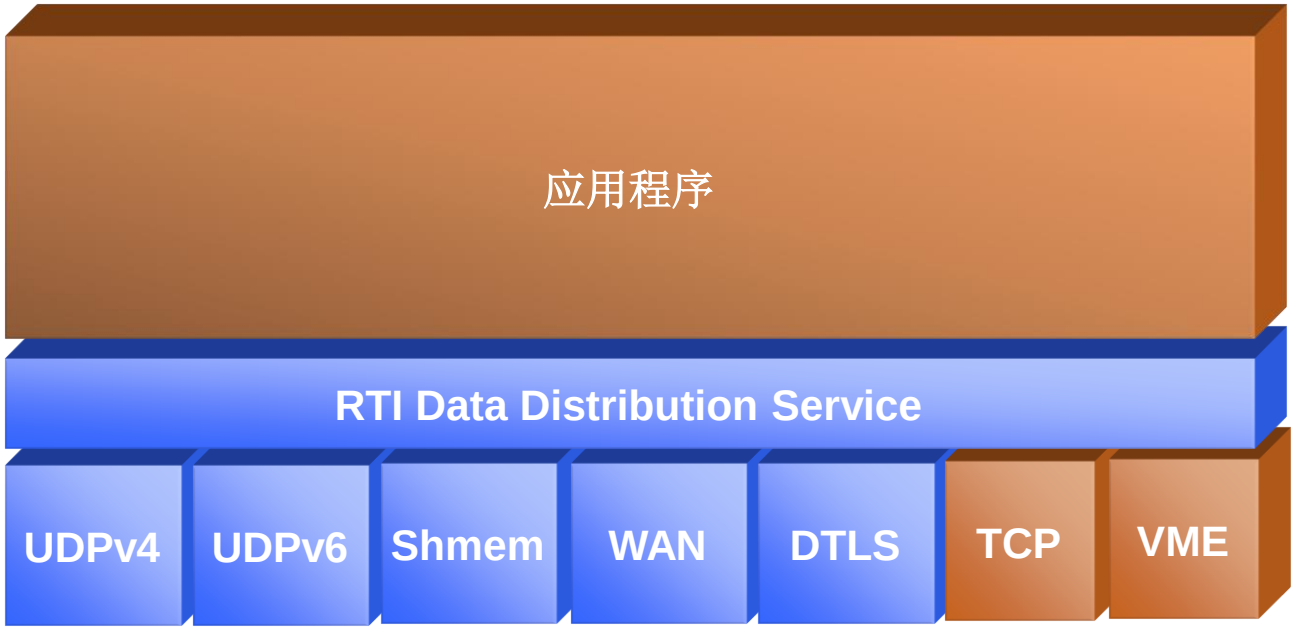
*可用的QOS策略*

- 传输途径
  - 可以使用多个传输途径同时传输
  - 可以同时使用多网络接口
- 所有权
  - 对于同一个主题，可以存在多个数据发送者


© 2009 Real-Time Innovations, Inc.

# QoS: 传输途径

- DDS 可以同时使用多个传输途径
  - UDPv4, UDPv6, Shared memory, WAN, DTLS等等
- 用户可以创建并安装自己的传输途径

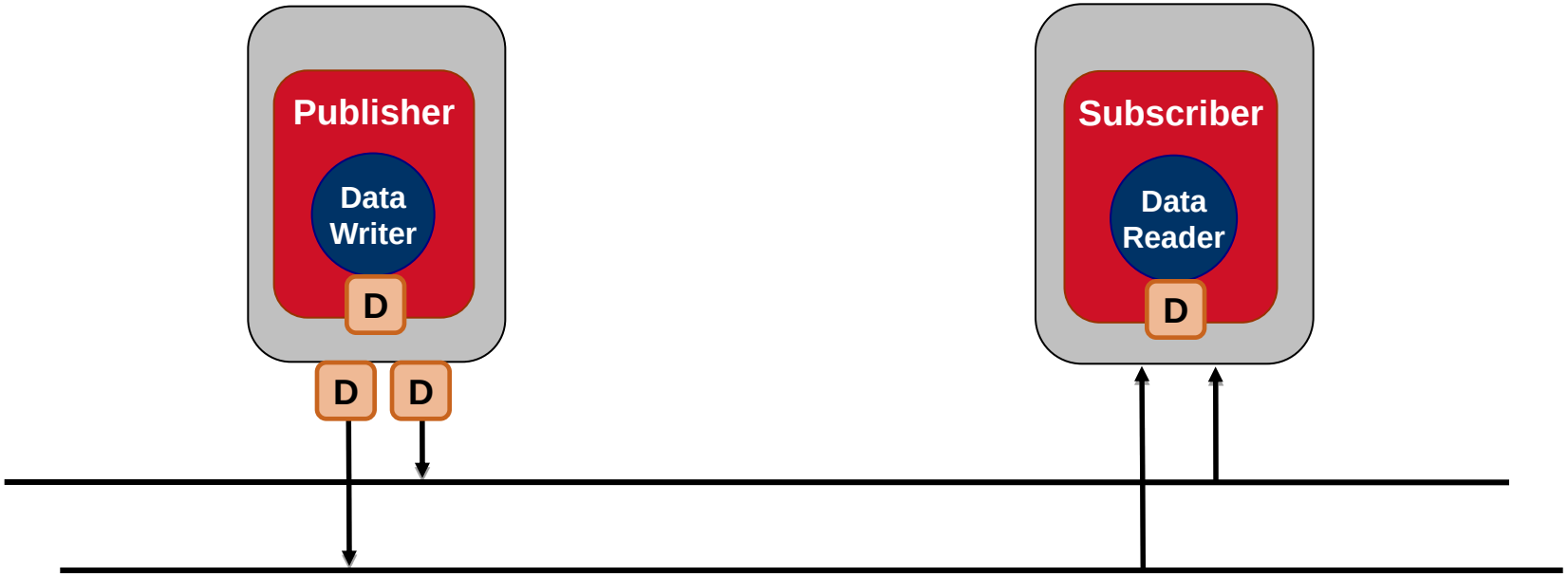




Recommend 火龙果软件
© 2009 Real-Time Innovations, Inc.

# QoS: 传输途径

- 对于基于UDP/IP的网络来说，可以同时使用多个网络接口



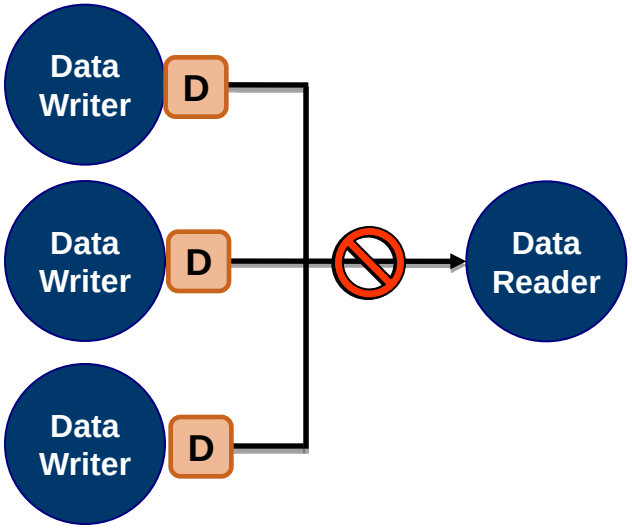

重复的数据被订阅方程序过滤掉


© 2009 Real-Time Innovations, Inc.

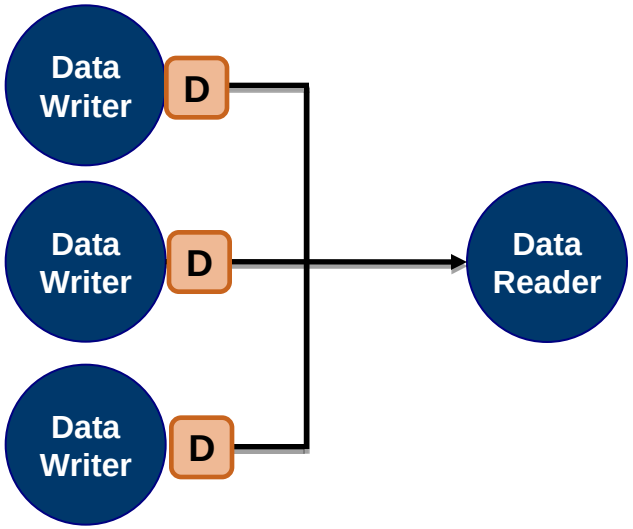
# QoS: 所有权

用来指定数据接收者是否可以同时从多个数据发送者中接收数据

所有权= 排他




所有权= 共享




快速，健壮，透明的失效恢复和岗位替换机制


Recommend 火龙果软件
© 2009 Real-Time Innovations, Inc.

# 高有效性 – 故障恢复

*可用的QOS策略*

- 所有权（ Ownership）
  - 排他的设定保证了仅有单个数据源有效
- 所有权的权重（Ownership strength）
  - 可以区分主要的/活泼的与次要的/后备的数据发送者
- 存在性和截止时间（Liveliness and deadline）
  - 如果存在性不存在或截止时间到期情况下，会自动启动失效-恢复机制

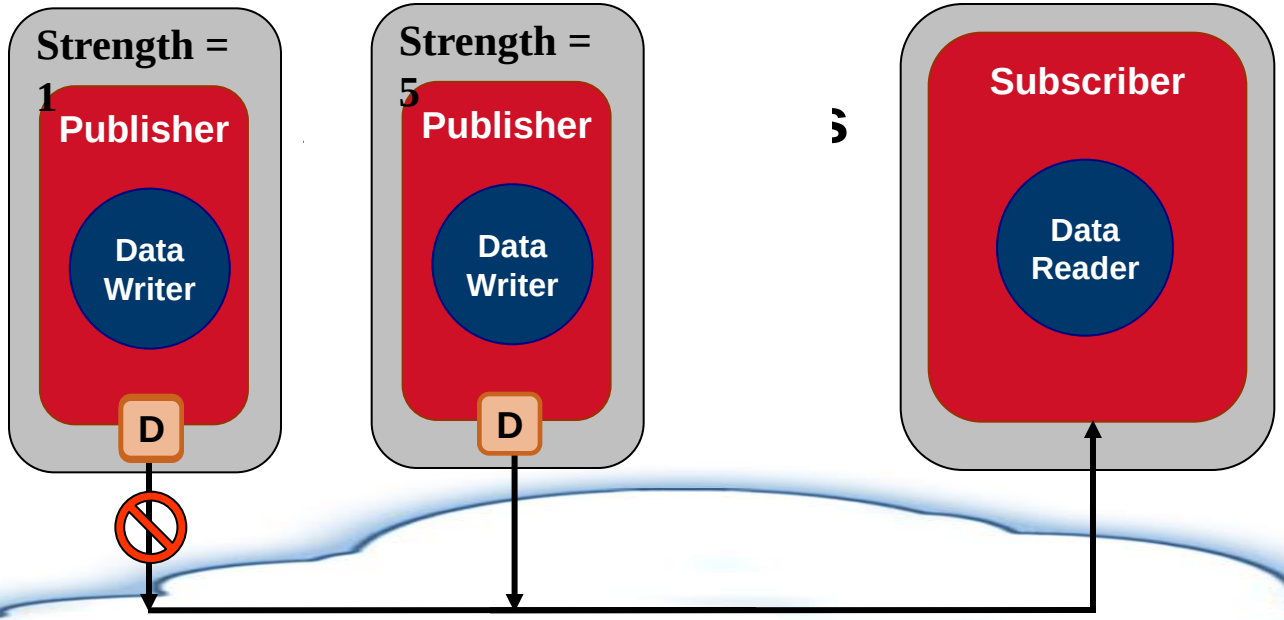
Recommend 火龙果软件

© 2009 Real-Time Innovations, Inc.

# QoS: 所有权的权重

这个策略决定众多数据发送者中哪一个数据发送者可以将数据发送给数据接收者

对于排他的所有权策略中，权重值最高的数据发送者拥有数据发送权





© 2009 Real-Time Innovations, Inc.

# 高有效性 – 被保证的消息（Guaranteed Messaging ）

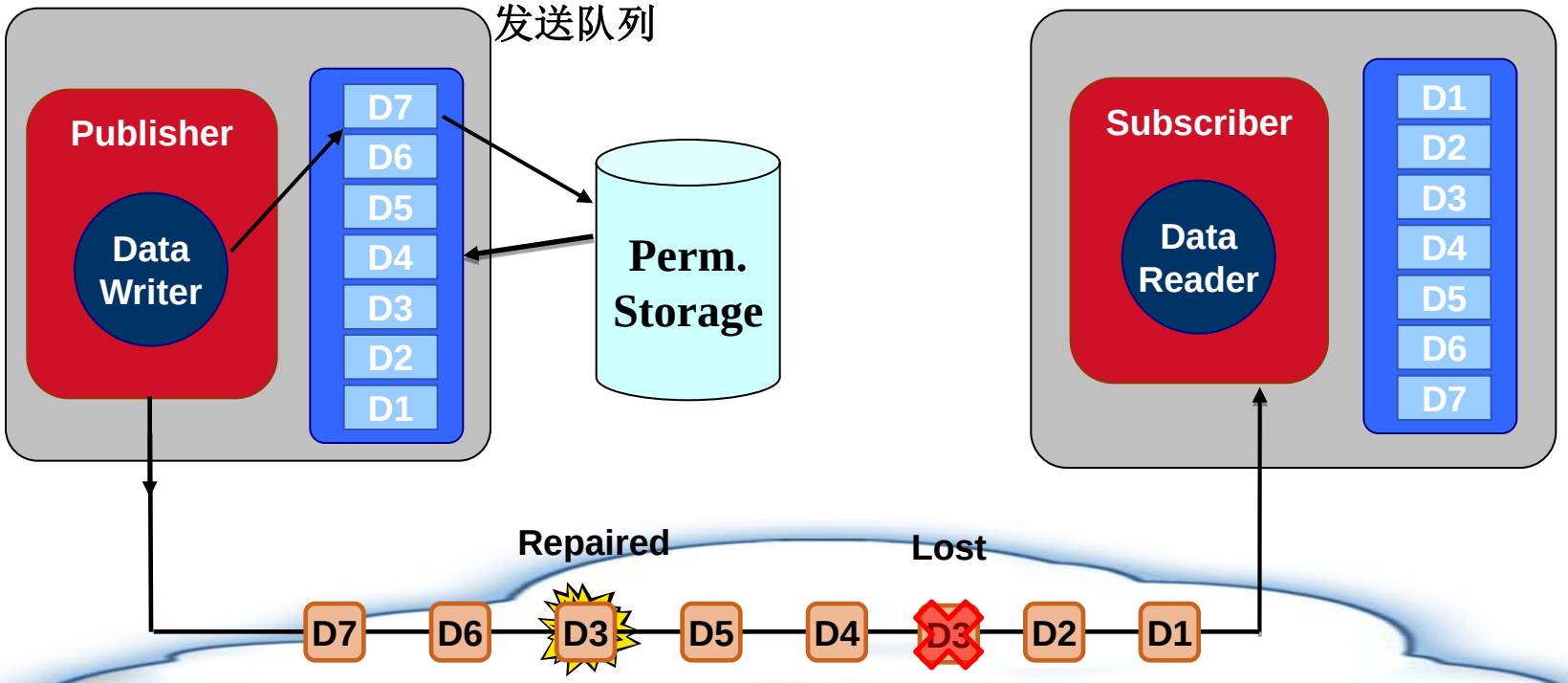
## *可用的QOS策略*

- 持久的发送历史记录（DurableWriterHistory）
  - 数据发送者与可靠相关的状态和发送队列被保存在本地磁盘中
- 持久的接收状态（DurableReaderState）
  - 数据接收者与可靠相关的状态被保存在本地磁盘中
- 永久服务(Persistence Service)
  - 保存数据并确保在数据发送者失效时数据依然可以被可靠地传输


Recommend 火龙果软件
© 2009 Real-Time Innovations, Inc.

# QoS:持久的发送历史记录（DurableWriterHistory）

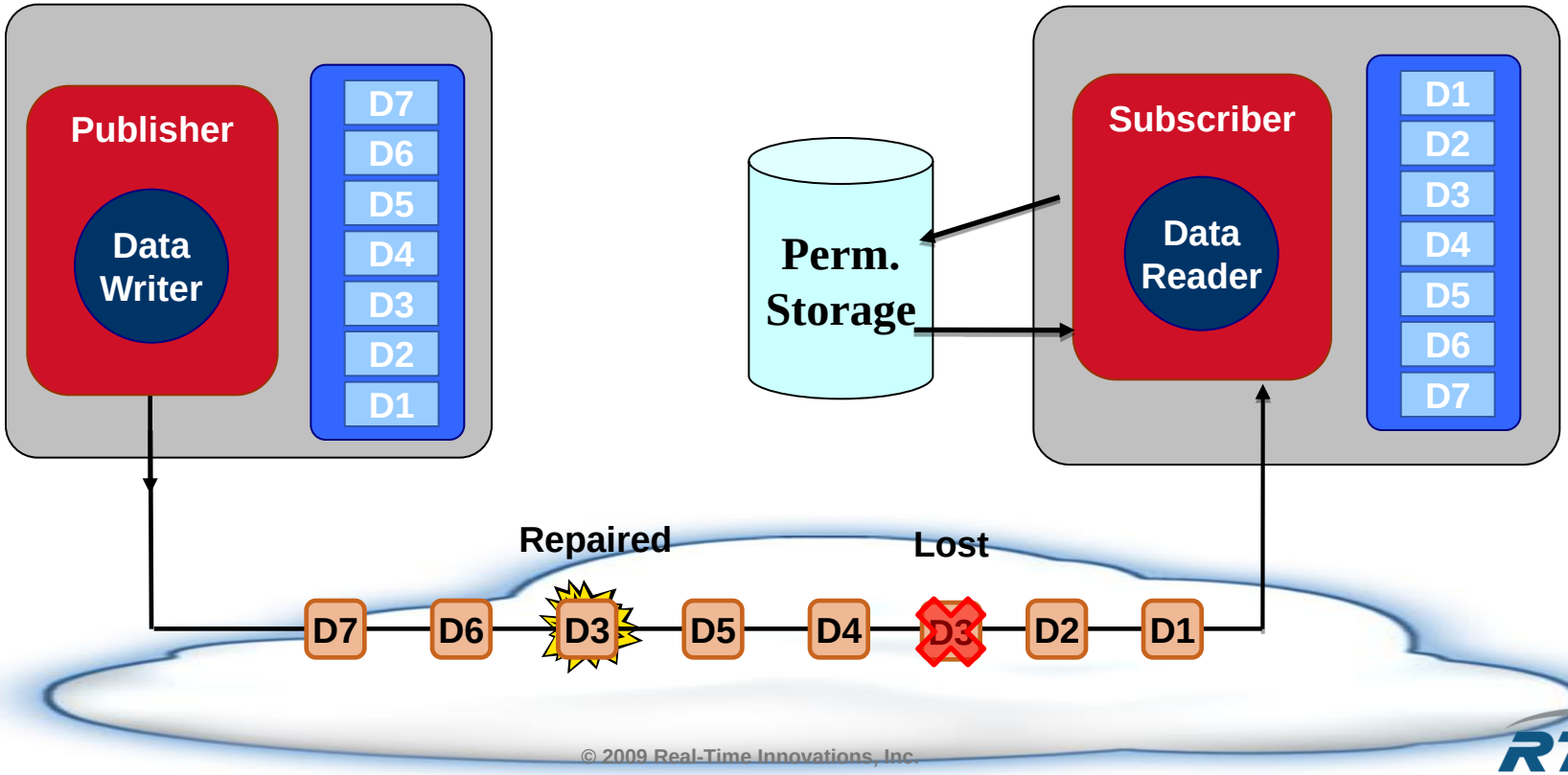
- 与实现可靠性协议相关的状态被保存在磁盘中
  - 发送队列
  - 可靠的数据接收者列表
  - 哪个数据需要采用应答机制
- 保存的状态将在数据发送者崩溃重启后被恢复




© 2009 Real-Time Innovations, Inc.

# QoS:持久的接收状态（DurableReaderState）

- 将数据接收者的可靠相关状态保存在磁盘中
  - 可靠的数据发送者列表
  - 最后接收到的数据的列表
- 保存的状态将在数据接收者重新工作后被恢复



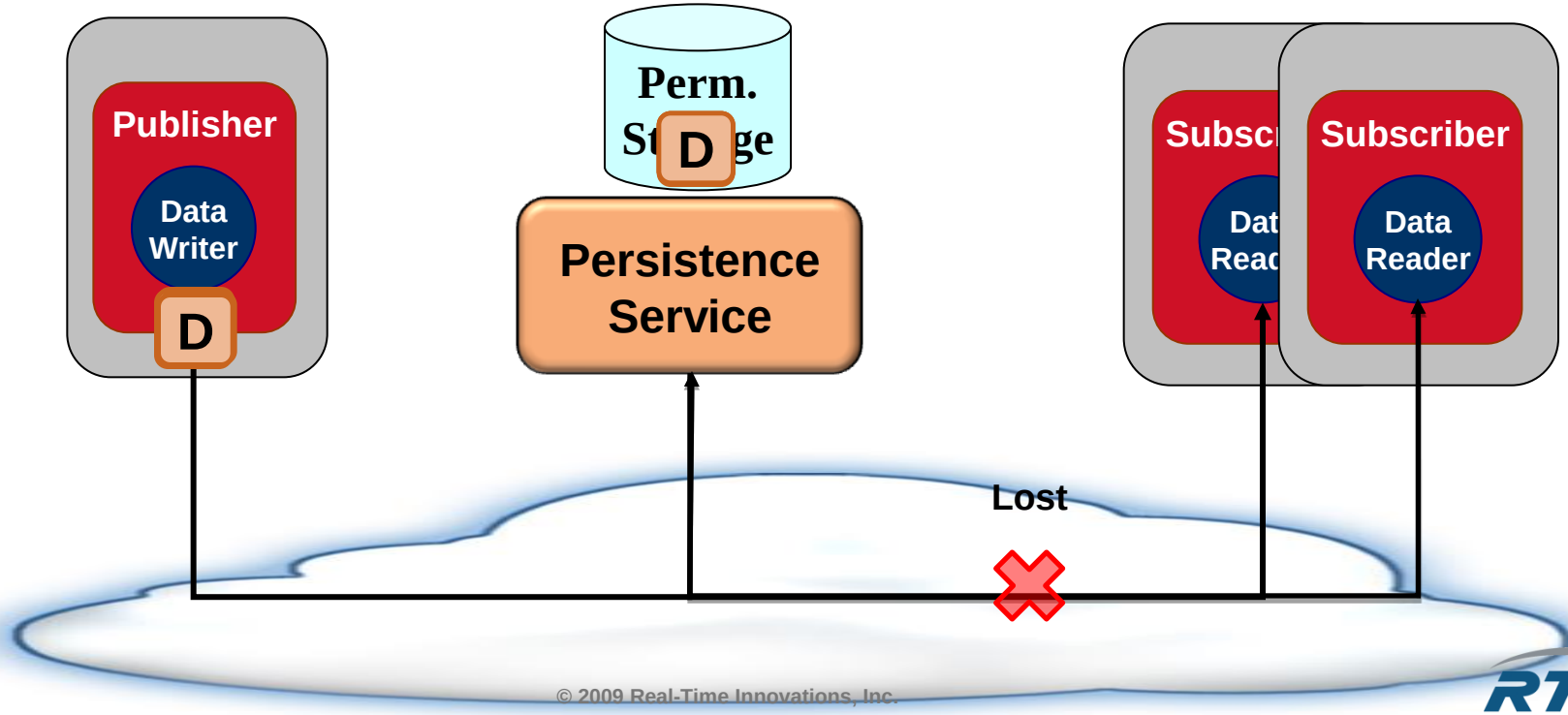


© 2009 Real-Time Innovations, Inc.

# QoS: 永久性/服务（Persistent Durability/Service）

- **RTI 永久性数据服务**
  - 保存分发过的数据并将在需要的时候发送到网络中

- **容错的优势（Fault Tolerance Benefits）**
  - 确保可靠的数据始终有效，那怕在数据发送者失效情况下
  - 在不需要数据发送者参与的情况下有效地提供数据给新的数据接收方





© 2009 Real-Time Innovations, Inc.

# 设计模式 – 安全性和可测量性（Security and Scalability）

- 认证（Authentication）
  - 通过钩子函数传递证书用于认证和授权
- 加密（Encryption）
  - 用户自定义的传输插件可以加密/解密所有发送/接收的数据
- 主题范围（Topic scoping）
  - 使用不同的域或区来设定主题的适用范围
- 关键字（Keys）
  - 在成百上千个对象中自动管理消息状态
  - 在数据库中直接映射为表中的primary key


© 2009 Real-Time Innovations, Inc.

# 安全性 – 认证

## *可用的QOS*

- 用户数据（User Data）
  - 在发现过程中传入保密证书用于认证和授权
- 发现和内置主题（Discovery and Builtin Topics）
  - 可以监听和拦截发现进程
  - 防止与没有被授权的应用程序建立连接


© 2009 Real-Time Innovations, Inc.

# QoS: 用户数据和发现（User Data and Discovery）

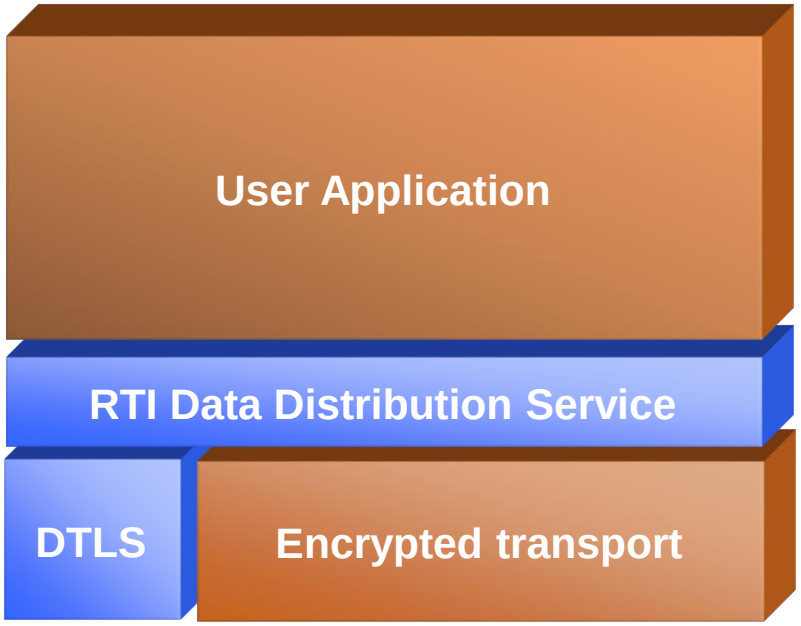
- 用户数据是附加带有发现信息的二进制数据

- 通过监听者来截获发现过程的通讯
  - 在新应用程序，新数据发送者和数据接收者被发现时调用
  - 监听者可以提取用户数据并使用标准的安全技术来认证或授权网络连接
  - 用户代码可以拒绝：
    - 来自未经授权的程序的连接
    - 未经授权的数据接收者接收数据
    - 来自未经授权的数据发送者的数据


Recommend 火龙果软件
© 2009 Real-Time Innovations, Inc.

# 安全性和可测量性 – 加密

- DTLS 的传输插件
  - 使用针对Datagrams流的标准的TLS（SSL）技术
  - 用公开/私有key签名证书进行加密
- 用户自定义的传输插件
  - 用户编写代码实现自己的私密传输





Recommend 火龙果软件
© 2009 Real-Time Innovations, Inc.

# 可测量性 – 主题范围

*可用的QOS策略*

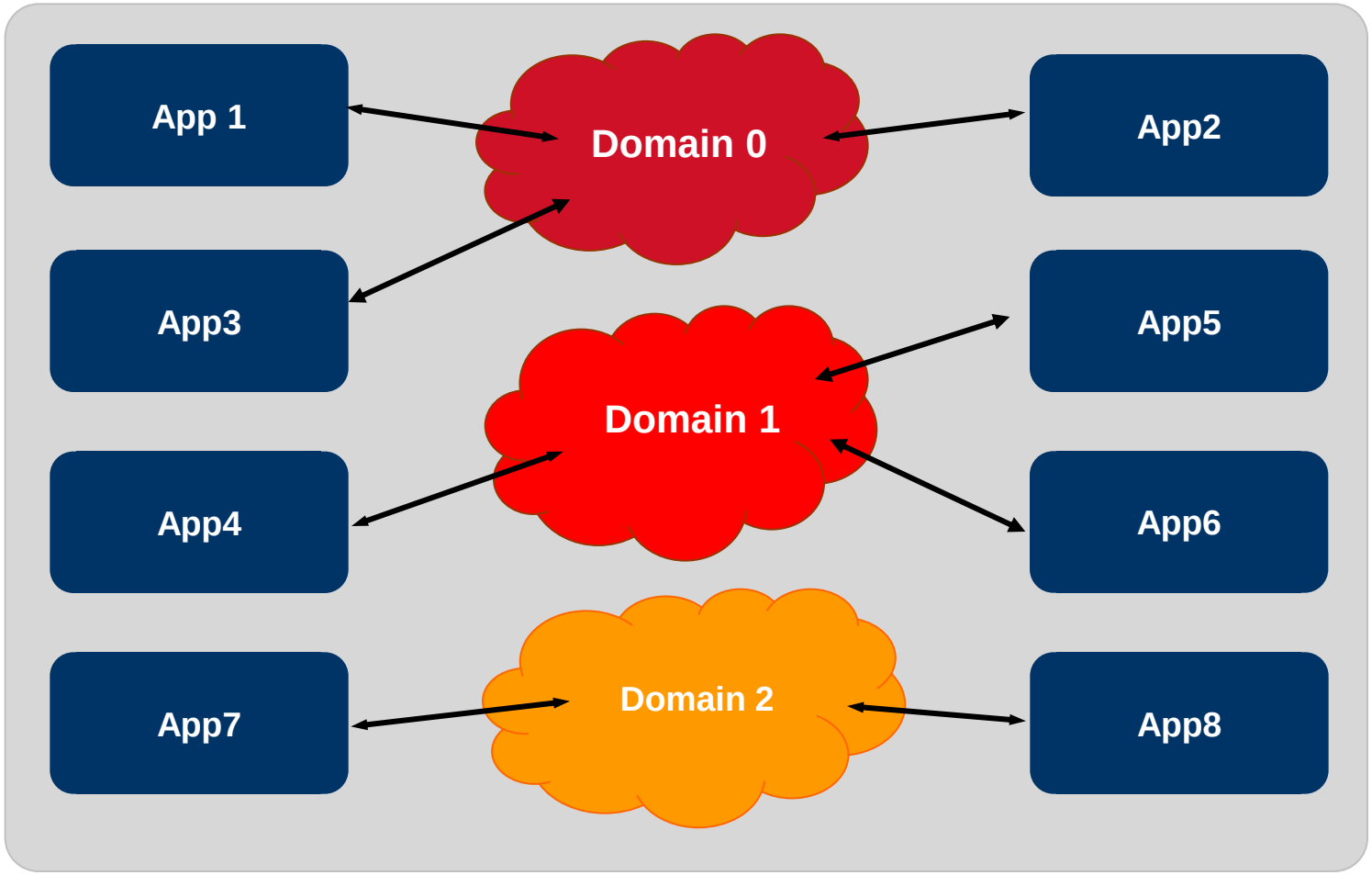
- 域（Domains）
  - 使用多个域可以用于隔离，限定范围，或测试数据传输
  - 可以将“私有”或“保护”数据流与公开数据流区分开
  - 允许多系统共享使用同一个物理网络而没有“串音”的风险

- 区（Partition）
  - 允许/阻止拥有相同主题的数据发送/接收者交换数据
  - 可以将数据发送/接收者们动态成组或取消成组

Recommend 火龙果软件

© 2009 Real-Time Innovations, Inc.

# QoS: 域（Domains）



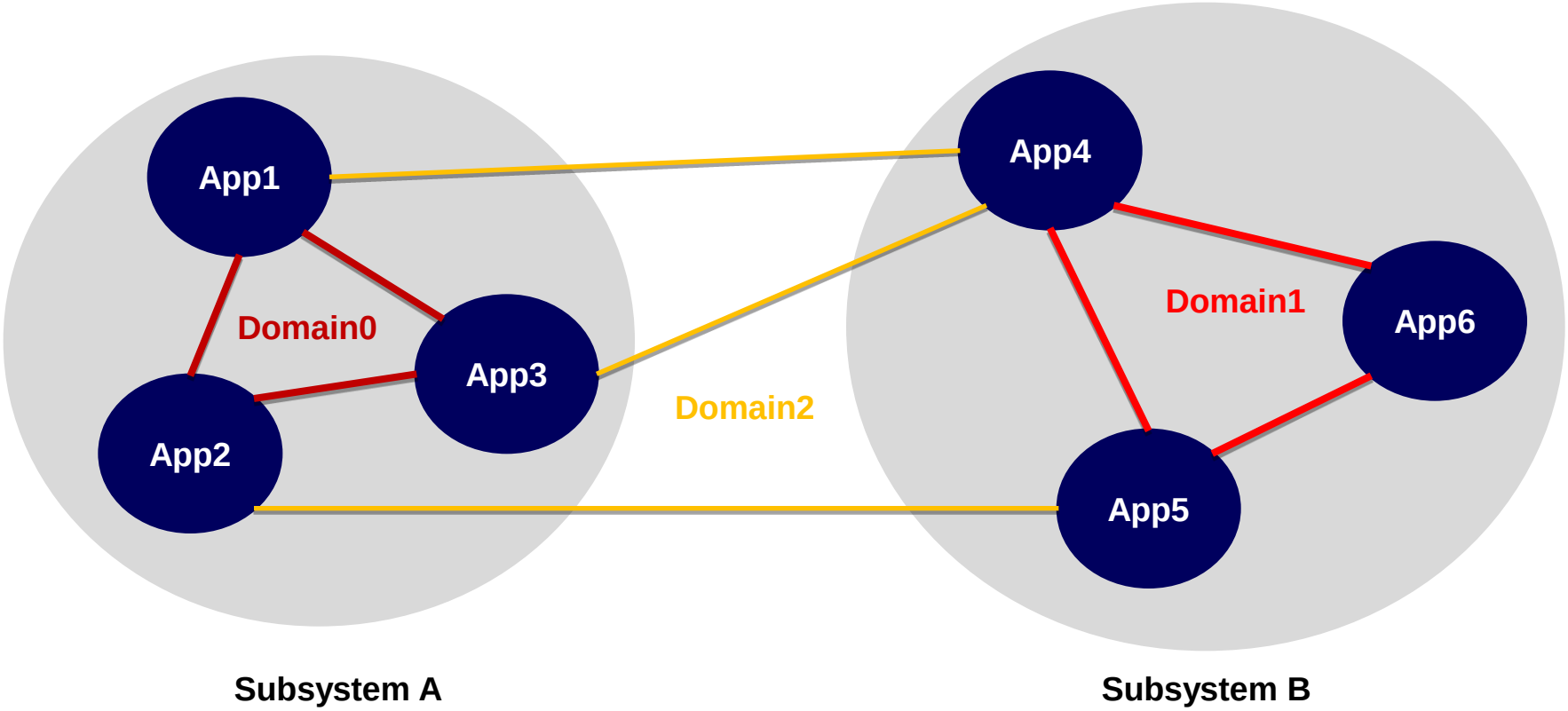

- 隔离：不同域内的程序不会交互数据

Recommend
火龙果软件



© 2009 Real-Time Innovations, Inc.

# QoS:域（Domains）

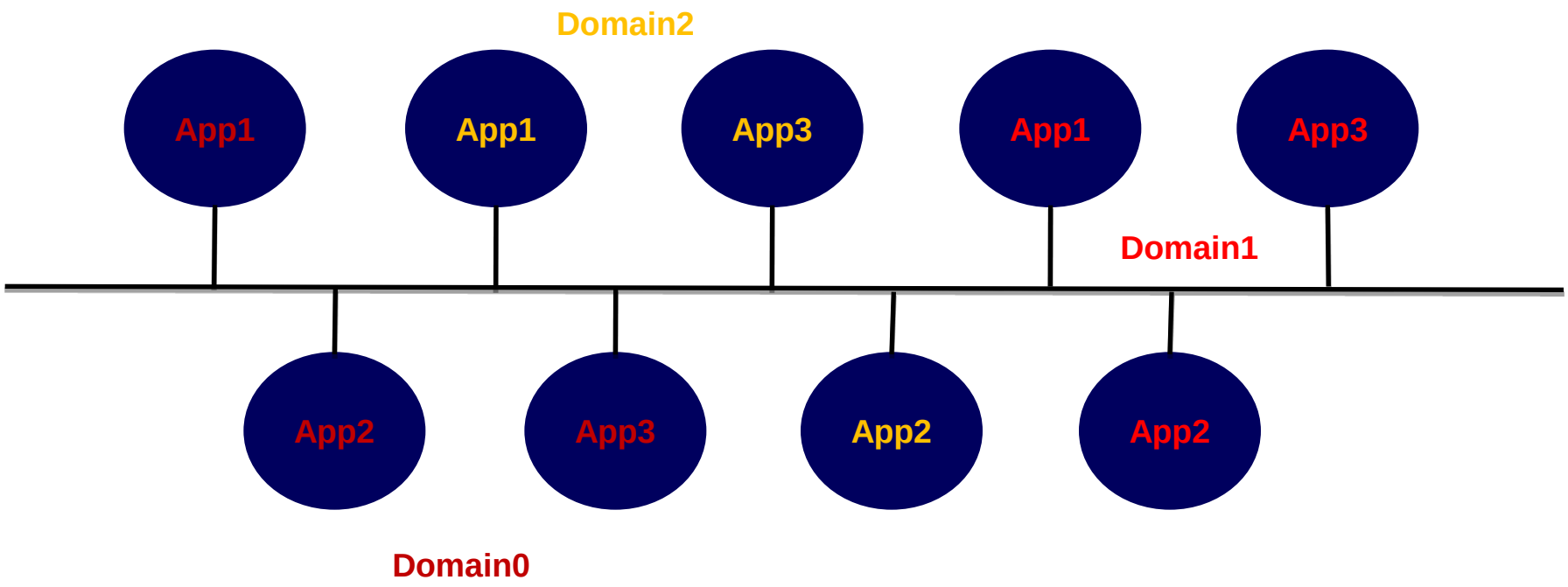



- 范围限定: 私有子系统的消息使用内部的域，使用外部的域作为公共接口
- 应用程序可以同时属于多个域

Recommend 火龙果软件

© 2009 Real-Time Innovations, Inc.

# QoS:域（Domains）

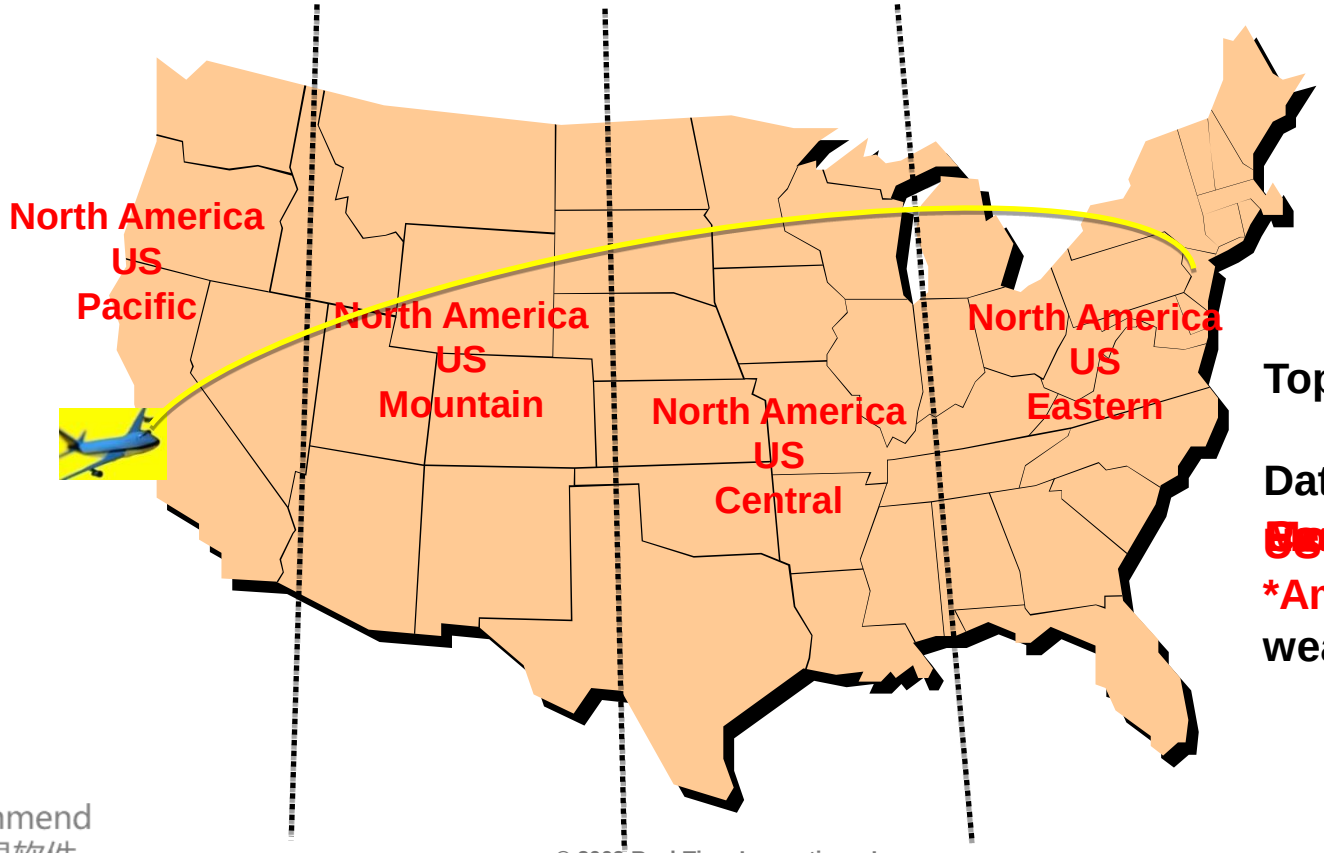



- 在同一个物理网络中建立虚拟的数据网络
- 允许开发者之间共享网络在同一个程序上进行独立的测试


© 2009 Real-Time Innovations, Inc.

# QoS: 区（Partition）

- 是在数据发送/接收者上附加的字符标签
- 仅当数据发送/接收者的区标签匹配才能交换数据
- 数据发送/接收者可以同时属于多个区
- 区可以使用通配符来匹配检验




Topic: Weather

DataReader Partition:
US – all US weather
*America – all weather in Americas


Recommend 火龙果软件
© 2009 Real-Time Innovations, Inc.

# 可测量性 – 关键字（Keys）

主题可包含“关键字”段

唯一的关键字值定义了一个唯一的主题实例

可以用千万个实例代表千万个对象

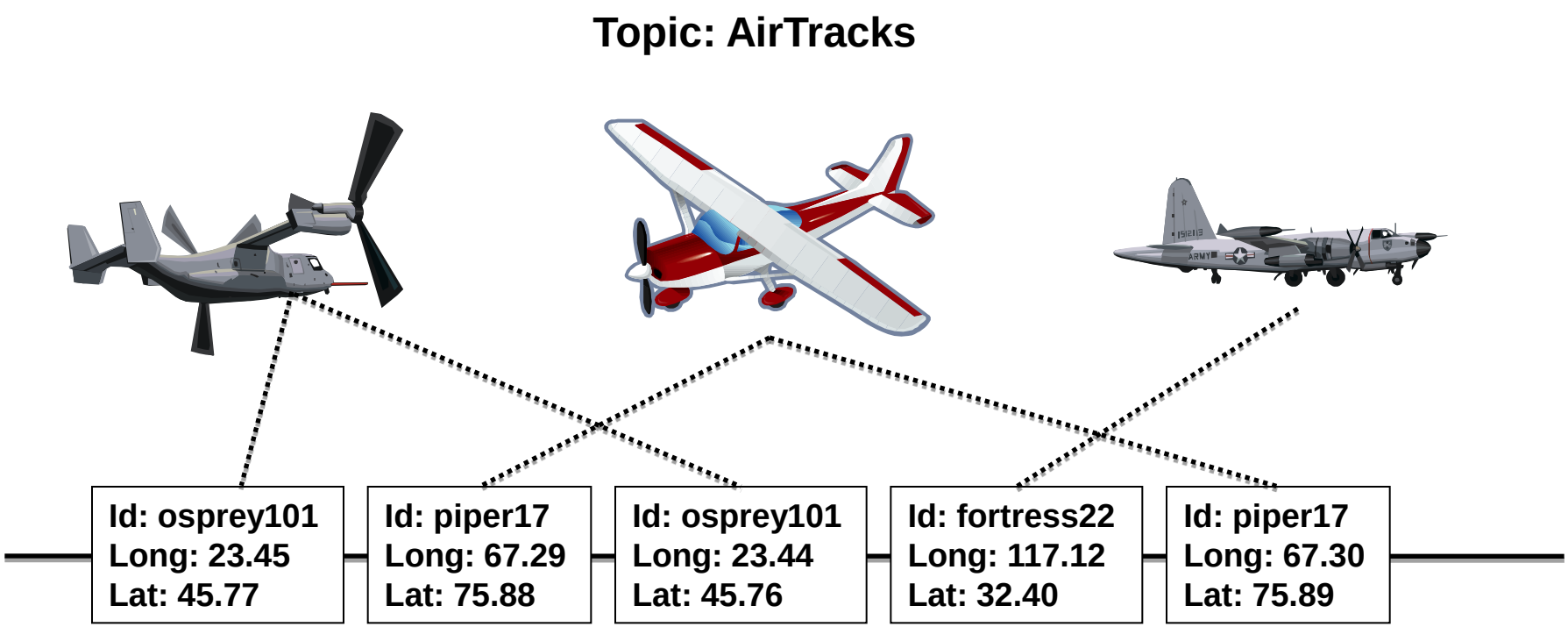
*每一个实例可以应用下列QOS策略*

- 所有权（Ownership）
- 截至时间（Deadline）
- 历史记录（History）


© 2009 Real-Time Innovations, Inc.

# 关键字（Keys）

- 单个数据流（也就是主题）可以包涵不同的对象（实例）的数据
- 关键字(特定数据字段的唯一值)决定那个数据给那个对象
- 下图中：Id就是Keys





© 2009 Real-Time Innovations, Inc.

# 映射到相关的数据库中

- DDS IDL ←→ table schema
- Data fields ←→ table columns
- Field names ←→ column names
- Fields types ←→ column types
- DDS Keys ←→ Primary Keys
  - Keys can be multiple data fields
  - Multiple columns can be primary keys

```
struct tracks {
 string Id; //@key
    float Long;
     float Lat;
       };
```

| Id* | Long | Lat |
|---|---|---|
| osprey101 | 23.48 | 45.72 |
| piper17 | 67.33 | 76.01 |
| fortress22 | 117.53 | 32.58 |


Recommend 火龙果软件
© 2009 Real-Time Innovations, Inc.

# 使用Key的好处

- 很多QOS可以加在每个单独的关键字值（实例）上
  - 截至时间（Deadline）
  - 历史记录（History）
- DDS可以管理实例的生存周期
  - 在新实例（keys）出现时给出提醒
  - 在现存的实例（keys）消失时给出提醒
- DDS使用key值管理内部资源
  - 在实例之间公平地分配资源
- 根据key值高效地过滤
- 可以对高达1000～100000对象（实例）进行管理


© 2009 Real-Time Innovations, Inc.

# 更多的特性……

- 发送数据
  - 成批模式(Batch)
  - 流控制(Flow Controller)
  - 多通道（Multi-channel）
- 接收数据
  - 可基于内容过滤的主题
- 数据类型
  - 程序执行中动态定义数据类型
  - 稀疏类型的数据类型


Recommend 火龙果软件
© 2009 Real-Time Innovations, Inc.

# 发送数据：采用成批模式的发送者

- 对数据发送者通过QOS配置
- 当发送小数据（<2K）时，CPU性能得不到充分利用
- 在同样数据总量的情况下，发送少次大量数据比发送多次少量数据更有效率，可达到更高的吞吐量
- RTI仅在下列情况下才发送一个网络包
  - 数据数 > max_samples（batch模式下一次发送的最大数据量），或
  - 累计数据总量 > 最大的max_data_bytes，（最大系列化后的累计数据长度）或
  - 从需要发送的第一个数据开始计算的时间间隔 > max_flush delay
- 网络吞吐量与数据延时的之间的折衷
- 效率从7万包每秒到150万包每秒的提高 (每包200 字节的数据)

Recommend 火龙果软件

© 2009 Real-Time Innovations, Inc.

# 数据发送: 流控制

- 使用异步的发送线程
  - 数据不是在用户线程中通过调用write()发送
- 可以使用内置或用户自定义的“流控制器”
- 流控制器
  - 规划网络的使用率
  - 例如，限制数据发送者可以使用的最大带宽， 保留的缓冲区大小和按照配置的最高速率发送
  - 内置的流控制器使用 tokens and leaky buckets


Recommend 火龙果软件
© 2009 Real-Time Innovations, Inc.

# 发送数据:多通道发送者 （Multi-Channel DataWriter）

- 数据发送者根据数据的不同发送给不同的多播组
  - 例如，主题是“股票报价”，不同公司的数据根据股票的代码区分：

Symbols [A-C]* sent to 224.0.0.1

Symbols [D-F]* sent to 224.0.0.2

…

Symbols [Y-Z]* sent to 224.0.0.9

将股票分成9组

  - 当用户发送报价时，RTI根据报价的符号将之直接发送到相应的多播组


Recommend 火龙果软件
© 2009 Real-Time Innovations, Inc.

# 发送数据:多通道发送者 （Multi-Channel DataWriter）

- 使用场合？
  - 多个订阅者订阅相同的主题
  - 每一个订阅者仅仅想要得到全部数据中的一部分数据
  - 数据量很大，所以利用多播方式减少数据在网路上的重复传播
  - 减少应用程序接收和处理不需要的数据

Recommend 火龙果软件

© 2009 Real-Time Innovations, Inc.

# 接收数据：主题的内容过滤

- DDS仅仅传送内容被过滤器过滤后数据，该过滤器是由订阅方指定
  - 例如：订阅数据 “Air Tracks”，仅仅接收目的地是本机场或在100公里范围内的航空器轨迹信息
- 使用主题的内容过滤
  - 内置的过滤器使用SQL中“where clause”一样的语法
    - “X > 8 AND Z < %1”, %1 是个可以改变的参数
  - 用户可以安装自定义的过滤器以决定数据是否被传送，过滤器可包含任意逻辑或算数计算
- 过滤可以在发送方执行（没有通过过滤的数据不被发送），也可以在接收方执行


© 2009 Real-Time Innovations, Inc.

# 数据类型：动态定义数据类型

- 不需要定义文件，不需要代码生成，没有任何问题！可以使用动态类型定义
- 通过API函数可以在运行时定义数据类型
  - 不需要IDL，XML，或XSD文件定义数据类型
  - 不需要编译链接来产生类型支持代码
- 可以操作包含动态定义数据类型的对象
- 与IDL/XML/XSD中定义的数据类型配合使用
- 但是不能做到：
  - 动态改变数据类型
  - 在以前存在的数据类型中添加或去除字段
  - 不能是一个自描述的数据类型


© 2009 Real-Time Innovations, Inc.

# 数据类型: 动态定义

- Can create a local data object from a data type that was found during discovery (recall type codes)
  - Useful in creating tools/bridges that are data-type agnostic, can work with any data type (that have type codes found during discovery)

```
my_typecode = typecode_factory->create_struct_tc("MyType", …);
my_typecode->add_member("ID", …, long_typecode, …);
my_typecode->add_member("X", …, float_typecode, …);
my_typecode->add_member("Y", …, float_typecode, …);
my_typecode->add_member("msg", …, string64_typecode, …);
type_support = new DDSDynamicDataTypeSupport( my_typecode, …);
dynamic_data = type_support->create_data();
dynamic_data->set_long("ID", …, 7);
dynamic_data->set_float("X", …, 3.1415f);
...
dynamic_data->get_long(id, "ID", …);
...
```


© 2009 Real-Time Innovations, Inc.

# 数据类型: 稀疏类型（ Sparse Types）

- 仅仅发送主题数据段中的子集
  - 主题需要使用新的**稀疏类型**
  - 稀疏类型不能与动态定义类型或IDL/XML/XSD定义的类型一起工作
- 稀疏类型可以通过API在代码中定义
  - 当发送数据时，仅仅用户定义了的字段才被发送
  - 通过仅发送值发生变化的字段数据来减少网络带宽使用(而不是发送全部数据)
- 稀疏类型是可扩展的，因此数据类型的改变可以向上/向下兼容


© 2009 Real-Time Innovations, Inc.

# RTI Data Distribution Service 常见问题回答

© 2009 Real-Time Innovations, Inc.

# 是否可以在流数据应用（例如音视频）中通过RTI DDS传输数据?

- 当然可以，而且已经有用户这样做了
- 音视频数据可以看成字节流
  - 可以使用内置的byte buffer数据类型或者用户自定义的数据类型并转换为byte buffer类型
- 可以采用多播方式进行一对多的高效传输
- 使用“最有效”的可靠性策略
  - 可能会有数据的丢失从而导致图像丢帧或音频跳音的现象发生
  - 但是不会发生延时现象，因为DDS不会试图修补丢失的数据包
- RTI的DDS不直接支持音视频的解码，这部分工作需要用户的应用程序完成


© 2009 Real-Time Innovations, Inc.

# DDS是否支持认证，加密或保密传输？

- 可以，就像以前已经讨论过那样。有很多方法确保用户可以保密通讯
  - 可选的DTLS (SSL for UDP) 传输插件
    - 使用证书方式进行验证，授权和加密
    - 所有的数据交互均会被加密，包括发现过程，心跳帧，应答帧和所有的用户数据
  - 用户也可以编写并安装自己的传输插件以实现保密的通讯协议
  - 用户可以通过API中钩子函数安装插件
    - 对用户数据进行加密，序列化/反序列化
    - 认证/授权模块和发现模块内置的主题监视器通过使用用户的QOS策略来接收或拒绝应用程序（端点）的连接请求

Recommend 火龙果软件

© 2009 Real-Time Innovations, Inc.

# 如何使用DDS来实现客户/服务器（像RMI那样的设计模式）方式的通讯？

- 可以，而且很多客户在这样做
- 有很多方法。通常使用一个主题作为客户端的请求，一个或多个主题作为服务端的应答
  - 每个客户端有一个DataWriter来发送请求，一个DataReader来接受回应
  - 每个服务端有一个DataReader接受请求，一个或多个DataWriter来发送回应
    - 可以使用“分区”或“基于内容的过滤”QOS策略来确保回应数据仅仅被发送请求的应用程序接受
    - 可以针对单个请求客户端，动态生成专门的DataWriter


© 2009 Real-Time Innovations, Inc.

# RTI的DDS可以在广域网/英特网上使用吗？

- 可以。主要的工作在于处理防火墙/网络地址转换（NAT）协议，而与保密无关
  - RTI提供可选的WAN插件（实际上就是 DTLS 插件）和可以穿越大多数防火墙通讯的WAN服务器
  - WAN的解决方案使用了STNU协议（也在Skype等程序中使用）和外部服务器程序来穿越防火墙并保存与处理地址映射
  - 不是适用于所有场合 (目前不能在symmetric firewalls下工作)

- RTI目前正在支持基于TCP/IP方式的传输方式。这样可以透过任意防火墙/NAT，并且能够借助TLS(SSH)协议来加密通讯数据

Recommend 火龙果软件

© 2009 Real-Time Innovations, Inc.

# 可以在RF-wireless或其他的网络中使用RTI的DDS？

- 可以。RTI已经帮助很多客户开发不同物理传输介质的传输插件
  - VME backplane (DMA)
  - TCP
  - RS232
  - Starfabric
- RTI DDS提供一个正式的传输插件
  - 仅仅用于发送/接受基于字节的缓冲（buffers of bytes）
  - RTPS 可靠性协议在传输层之上，所以可靠性是与传输途径无关的
  - DDS可以同时利用多个传输途径来传输数据
  - 通常而言，开发插件需要RTI来提供更专业化的技术支持服务

Recommend 火龙果软件

© 2009 Real-Time Innovations, Inc.

# RTI的DDS是否支持DLRL？

- “为什么你的系统中需要使用DLRL？什么样的需求导致你需要使用DLRL？”
  - 这是针对这个问题的标准答案
  - 不，RTI不支持DLRL，当前也没有计划去支持它
  - 就我们所知，目前世界上仅仅只有一个DDS用户（Thales）使用DLRL，他们在DDS成为标准之前就已经有了DLRL。Thales也是DDS规范的定义者之一，是它将DLRL写入了DDS规范。
  - 世界上99.9%的DDS用户仅仅使用由RTI支持的DCPS（出版/订阅方式）
- 有一份完整的资料来描述DLRL同时说明为什么RTI不实现DLRL


© 2009 Real-Time Innovations, Inc.

# 是否RTI与OMG的DDS规范兼容？

- 是的，RTI与DDS的1.2版兼容（很快是1.3版）
  - RTI兼容规范中的Minimum Profile， Persistence Profile, 和 Ownership Profile
  - RTI 支持除了MutiTopic和QueryConditions 之外的Content-Subscription Profile
  - RTI不支持Object Model Profile (也就是DLRL)
- RTI支持2.1版的DDS协同工作协议，也就是RTPS
- 而且，RTI的DDS还提供了规范之外的一些扩展特性
  - 很多扩展特性已经或将要被添加到OMG的DDS规范中


© 2009 Real-Time Innovations, Inc.

# RTI的DDS是否支持传输优先级的QOS策略？

- 是的。对于UDPv4传输，传输优先级QOS策略会在IPv4帧中设置TOS (type of service)标记位
  - 在windows或Linux操作系统中，必须给予应用程序修改该位的权限

- 该QOS策略是非常粗略的
  - 在每一个版本的DDS规范中，均是可选的QOS策略
  - 由每个规范具体实施者自己决定如何实现
  - 对是否能达到效果不保证，那怕是设置了TOS位。因为在不是RTI开发的网络部件中，像网络交换机/路由器等等网络设备，可能不理睬TOS位的设定。这种情况下，这个“优先度”是否设置将不会产生任何可以测量到的区别。


© 2009 Real-Time Innovations, Inc.

# RTI DDS的开销有多大?

- 这让我们怎么说呢？非常小。肯定比用户自己编写带有QOS功能的代码来实现发送/接受数据的产生的开销要小
  - 在收/发UDP帧时，用户数据可能需要被拷贝入/出内部的缓冲区
  - 仅仅在收/发两端平台字节顺序(Endian Mode)不同时，才对字节顺序进行转换
  - RTPS在用户数据上除添加IP帧头，UDP帧头之外还添加自己的帧头
  - RTI会创建最少4个线程并占用一些内存创建自己的内部数据结构
  - 占据硬盘1-2兆字节，运行时使用 1-2兆字节内存，并随用户在程序中使用DDS实例的增加而线性增加


Recommend 火龙果软件
© 2009 Real-Time Innovations, Inc.