Source: DDS Best Practices

* Start by defining a data model, then map the data-model to D DS domains, data types and Topics. 일반적인 TCP/IP 또는 시리얼통신 등을 위하여 패킷을 정의할 때와 유사한 방법으로 데이터 모델을 정의할 필요가 있으며, 이렇게 정의된 데이터 모델을 도메인, 데이터 타입 및 토픽으로 매핑을 시켜야 한다.
* 이렇게 함으로써 서브 시스템을 별도의 도메인 단위로 분리할 수 있으며, 도메인 단위로 분리된 경우에 상당한 성능상의 이점을 갖게 된다.
* 데이터 모델 형식으로 정의를 한 후 UML 클래스 다이어그램과 같은 관계성을 검증할 수 있다. 다음과 같은 체크 리스트를 적용할 수 있다.
  + UML 클래스 다이어그램과 같이 자료 구조와 그 상호 관계 측면에서 데이터 모델을 표현할 수 있다.
  + 이러한 데이터 모델을 통하여 프로그래밍이나 소프트웨어 공학에 대한 지식이 없더라도 문제 영역에 대하여 논의 및 이해가 좀더 쉽게 가능하다.
  + 데이터 모델은 프로그래밍 언어, 플랫폼 등에 독립적인 UML, XML 또는 IDL과 같은 모델링 언어로 표현할 수 있다.
  + 데이터 모델은 위치 투명성을 가져야 한다. 즉, 데이터 모델은 하나의 컴퓨터에 배치될 수도 있으며, 원격지의 어디에도 배치될 수 있다.
* 토픽 정의 시 토픽의 아이템으로 사용되는 데이터 타입은 새로 정의해서 사용하는 것이 유리하다. 가능하면 string 이나 opaque byte array 타입의 사용을 지양해야 한다. 이유는 데이터의 내부 형태가 불분명하기 때문에 미들웨어가 제공하는 기능을 충분히 활용하기가 어렵기 때문이다. 예를 들어 필터링 기능이나 DBMS에 저장 기능 등 이용 시에 제약을 받을 수 있으며, marshalling 및 demarshalling 시 좀 더 복잡한 과정을 거칠 수 있다.
* 또한 가능한 경우에 키를 갖는 토픽을 정의하는 것이 여러 가지 경우에 유리하다. 예를 들어 토픽을 DBMS에 저장하고 조회하는 경우에 키를 가짐으로 인하여 속도를 높일 수 있으며, 어플리케이션끼리 샘플 송수신 시 필터링 기능을 사용하여 관심 있는 샘플만 수신이 가능하기 때문이다.

It is often useful to divide the data model into a control plane and a data plane. This allows separation of the

application and system management functions from the logic of the application components them-selves.

A checklist for a well-defined model:

• It can be fully expressed in terms of data structures and their relationships.

• It can be easily explained and understood to someone familiar with the problem domain, without

requiring the person to be knowledgeable in programming or software engineering.

• It is defined using a platform-independent modeling language, such as UML, XML or IDL. This way the

model is not tied to a specific platform, technology, or deployment scenario. The same model could be

deployed on top of different technology stacks (DDS, WSDL, message bus), programming languages

(Java, C++, C#), and configurations (platforms, networks, etc.).

• There are no location dependencies in the model. It can be deployed on a single computer or on a

network.

• The different scenarios and behaviors of the system can be expressed in terms of changes to the state of

elements in the data model.

* Configure QoS using XML Profiles. XML 파일을 이용하여 QoS설정하는 것이 QoS 관련 API를 이용하는 것보다 유연하며, 관리하기가 쉽다. 또한 QoS 값이 실행 파일 외부에 존재하기 때문에 개발이 완료된 후 재컴파일이 없이 어플리케이션 튜닝 등에 쉽게 적용할 수 있기 때문에 매우 유리하다.
* 일정 규모 이상의 프로젝트에서는 DDS API를 직접 사용하기 보다는 프로젝트 초기에 wrapper를 만들어서 사용하는 경우가 있다. 이렇게 함으로 인하여 종속성이 줄어드는 단점이 있지만 이 경우에도 주의할 점은 방대한 DDS의 특장점을 잘 이해하고 있어야 개발 중에 이미 방대해진 wrapper를 변경하는 경우가 없게 된다. 특히나 각 QoS 정책 중 주요 파라메타는 DDS에 있어서 매우 중요하면서도 샘플 송수신 시 미치는 영향이 매우 크기 때문에 특히나 주의해야 한다.

Source: best practices repeat success

Architectural Best Practices

* 어떤 상태가 표현돼야 하는지 조사하라. 예를 들어 화물 운송 회사의 경우 트럭 및 화물에 대한 정보를 송수신한다고 가정하자. 이 경우 트럭의 위치를 나타내는 GPS 정보, 트럭의 기름의 양, 화물 목록, 정비관련 정보 등등이 있을 수 있다. 이러한 정보들은 주기적인지 비주기적인지 또는 전송 주기 등등이 다르다. 따라서 이러한 정보 들은 각 정보의 특징을 감안하여 토픽으로 매핑을 해야 한다.
* 타입이 있는 데이터를 사용하라. Opaque 타입이나 단순 바이너리 타입이 아닌 명확한 타입의 데이터를 사용하면 많은 이점이 있다. DDS 미들웨어는 내부적으로 이런 타입 정보를 이용하여 discovery 또는 타입 정보 교환에 사용한다. 여기 내용은 위의 내용과 합한다.
* 키가 있는 데이터를 사용하라.
  + DDS 미들웨어는 QoS를 이용하여 토픽 인스턴스의 행동을 제어할 수 있다. 예를 들면, deadline QoS를 이용하면 특정 시간 동안 수신이 안되는 인스턴스를 찾아 낼 수 있다.

Application Design Best Practices

* 가능한 적은 domain participant를 생성하라. DomainParticipant가 내부적으로 수행하는 일은 다음과 같다.
  + Discovery를 수행한다. 각 어플리케이션은 디스커버리 프로토콜을 이용하여 네트워크내의 다른 어플리케이션을 discovery 하게되는데 이때 기준의 DomainParticipant 기준이다.
  + 따라서 발견된 타 DomainParticipant에 대한 내부 정보를 유지하게된다.
  + DomainParticipant 기준으로 쓰레드를 생성하여 데이터의 송수신과 DDS 미들웨어 내부의 이벤트를 처리한다.

따라서 DomainParticipant가 사용될수록 보다 많은 리소스(메모리 및 쓰레드)가 사용된다. DomainParticipant가 생성될 때마다 네트워크 내의 어플리케이션은 이를 discovery 해야 함으로 네트워크 bandwidth 또한 소비된다. 따라서 Domain 당 하나의 DomainParticipant 만 생성하는 것이 좋다.

* DDS entity를 가능한 초기에 생성하라. Discovery 프로세스 중에 로컬 어플리케이션의 DataReader 또는 DataWriter와 매칭되는 원격지 어플리케이션의 DataReader와 DataWriter를 매칭시키게 된다. 이러한 discovery 프로세스가 완료되기 전에는 일반적인 QoS(Reliable QoS 포함하여)인 경우에도 데이터의 전송이 이루어지지 않는다. 즉 개발자는 일반적으로 DataWriter를 생성하자 마자 정보를 전송하고자 하지만 매칭되는 DataReader 쪽에서의 discovery 절차가 완료되지 않는 이상 전송이 이루어지지 않게 된다. 따라서 discovery 프로세스가 일찍 완료될 수 있도록 DDS 엔티티는 가능한 초기에 만드는 것이 좋다.
* Callback 함수 내에서 block되지 않도록 하라. DDS listener 클래스는 특정 이벤트가 발생할 때마다 호출되는 콜백 함수를 제공하고 있다. 이러한 listener는 DataReader, DataWriter, DomainParticipant, Publisher 또는 Subscriber에 모두 존재한다. Listener 클래스를 통하여 수신된 데이터를 사용하거나, 수신이 지연되거나 또는 데이터가 분실 됐음을 알 수 있다. 이런 listener 콜백 함수는 DDS 미들웨어 쓰레드에 의해서 호출되며, DDS 미들웨어 쓰레드는 socket으로부터 데이터를 수신하고, 이를 전송하거나 이벤트가 발생했는지를 확인하는 등의 다양한 작업을 수행한다. 만약 listener 콜백 함수 내에서 block이 되는 경우 DDS 미들웨어 쓰레드가 역시 block이 되기 때문에 위에 언급한 DDS 미들웨어의 다양한 작업이 방해를 받게 되는 일이 발생한다. 따라서 listener 콜백 함수 내에서 sleep 이나 긴 시간이 소요되는 작업을 하지 않는 것이 좋다. 보통의 경우는 queue를 이용하여 수신된 데이터를 저장하여 다른 쓰레드에 의해서 처리하도록 하고 즉시 리턴하는 것이 좋다.
* Listener 대신 WaitSet을 사용하라. 일반적으로 극단적인 성능이 필요한 경우가 아니면 listener 보다는 WaitSet을 사용하는 것이 보다 안전하다. 이유는 WaitSet의 경우 사용자 쓰레드를 이용하여 특정 이벤트의 발생을 기다리기 때문에 DDS 미들웨어에 영향을 미치지 않기 때문이다. WaitSet의 단점은 사용자 쓰레드를 이용하기 때문에 DDS 미들웨어 쓰레드로부터 context switching에 따른 수신 지연이 발생한다는 점이다.

Network and QoS Best Practices

* 올바른 Reliable QoS를 사용하라. 많은 개발자들이 모든 데이터가 항상 반듯이 전송이 되어야 한다고 생각하지만 반듯이 그렇지만은 아닌 경우도 있다. 예를 들어, 1초 마다 주기적으로 발생하는 센서의 상태 정보는 GPS 위치 정보와 같이 중요하지 않은 정보일 수도 있으며, 센서의 상태 정보는 급박하게 변경되지 않으며, 또한 주기적으로 송신되기 때문에 UDP 프로토콜의 특징에 따라 분실이 되더라도 마지막에 수신된 정보는 유지되기 때문이다. DDS 미들웨어는 Reliable QoS를 사용하는 경우에 DataWriter 내부에 큐를 유지하며 DataReader가 데이터를 수신하지 못하는 경우 재전송을 하게 된다. 이런 과정에서 내부 큐가 꽉 차는 경우에 구현 방법에 따라 블록되는 경우도 있게 된다.
* 전송 버퍼는 최대 크기를 사용하라. UDP v4의 경우에 헤더를 감안하면 그 크기는 다소 줄어들지만 일반적으로 최대 전송 데이터 크기는 64K 정도이다. 전송 처리율을 최대로 하고자 하는 경우에 가장 먼저 고려해야 할 부분은 바로 이 전송 패킷의 크기를 최대로 늘리는 것이다.
* 가능한 멀티캐스트를 이용하라. 송신은 한 곳에서, 수신은 여러 곳에서 하는 경우에 가능하다면 멀티캐스트를 사용하면 데이터 송수신 시 bandwidth를 절약할 수 있을 뿐만 아니라, DataWriter가 한번 만 전송하면 되므로 CPU 사이클도 절약할 수 있다.
* 스위치의 IGMP 스누핑을 설정하라. 많은 스위치가 멀티캐스트를 브로드캐스트와 같이 취급하고 있다. 즉, DataWriter가 멀티캐스트 주소를 데이터를 전송하면 스위치는 이 데이터를 모든 어플리케이션으로 브로드캐스트한다. 따라서 스위치의 IGMP snooping 옵션을 활성화 시켜서 스위치가 정상적으로 멀티캐스트로 동작하도록 하여 네트워크 혼잡 방지를 할 수 있다.
* OS를 튜닝하여 처리량을 높여라. 만약 시스템이 높은 처리량의 데이터를 가지고 있다면, 운영체제의 설정을 변경하여 처리량을 높이는 방법도 있다는 것을 알아야 한다. 기본적으로 설정된 Windows는 많은 UDP 데이터 처리에 최적화되어 있지 않다. 따라서 Windows의 레지스티리나 리눅스의 시스템 설정 파라메터를 변경하여 처리량을 높일 수 있다. 예를 들어, 수신 오버로드가 큰 경우에 UDP 어플리케이션은 데이터 수신 시 오류가 발생할 수 있다. 콘솔 박스에서 netstat –s로 통계 데이터를 조회하면 아래와 같은 값들을 볼 수 있다.

|  |
| --- |
| IP Statistics  Packets Received = 687750865  Received Header Errors = 0  Received Address Errors = 1  Datagrams Forwarded = 0  Unknown Protocols Received = 0  Received Packets Discarded = 0  Received Packets Delivered = 574556088  Output Requests = 596445524  Routing Discards = 0  Discarded Output Packets = 0  Output Packet No Route = 0  Reassembly Required = 118445100  Reassembly Successful = 5234431  **Reassembly Failures = 107976783**  Datagrams Successfully Fragmented = 7642664  Datagrams Failing Fragmentation = 0  Fragments Created = 15285321 |

위의 내용을 살펴 보면 IP 헤더 리어셈블리 오류가 매우 큰 값을 나타내고 있음을 볼 수 있다. 이런 오류를 방지하기 위하여 Windows 레지스트리를 변경하여 이런 오류를 개선할 수 있다.

|  |
| --- |
| 1. **시작**을 누른 다음 **실행** 을 클릭하십시오. 2. **열기** 상자에 **regedit** 를 입력한 다음 **확인** 을 누릅니다. 3. 찾은 후 다음 레지스트리 키를 누릅니다.   HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters   1. 오른쪽 창에서 **MaximumReassemblyHeaders** 두 번 클릭하십시오. 2. **값 데이터** 상자에 IP 리어셈블리 헤더의 최대 수를 설정할 숫자를 입력한 다음 **확인** 을 누릅니다. 100-65535 사이의 임의의 정수를 입력할 수 있습니다. 100은 기본값이며 성능 향상을 위하여 가능한 큰 값을 설정합니다. 3. 레지스트리 편집기를 종료하고 시스템을 재부팅합니다. |

리눅스의 경우에는 커널의 설정을 변경할 필요가 있다. 예를 들어, 커널의 소켓 버퍼의 크기는 늘리면 성능 향상에 많은 도움이 된다.

|  |
| --- |
| [root]# /sbin/sysctl –a | grep core  net.core.rmem\_default = 109568  net.core.wmem\_default = 109568  net.core.rmem\_max = 131071 <- receive socket buffer size  net.core.wmem\_max = 131071 <- send socket buffer size |

위와 같이 설정된 값을 아래와 같이 늘려준다.

|  |
| --- |
| # Increase max. send/recv socket buffer limits  # for higher network performance  net.core.rmem\_default = 65536  net.core.rmem\_max = 2097152  net.core.wmem\_default = 65536  net.core.wmem\_max = 1048576 |

아래와 같이 소켓 백로그 값이나 IP 프래그먼트 리어셈블 값을 늘려 줄 수도 있다. 일반적으로 기본값은 약 300 정도이다. 1Gbit 이상의 네트워크 환경에서는 이 값을 늘려주는 것이 좋다. 이 값이 커지면 패킷 분실이 줄어들게 된다.

|  |
| --- |
| # Increase max. recv packets queued at the interface  # for higher network performance  net.core.netdev\_max\_backlog = 30000  # Increase memory size used to reassemble IP fragments  # for higher network performance  net.ipv4.ipfrag\_high\_threshold = 8388608 |

이 외에도 NIC의 MTU 크기를 늘리거나 인터럽트 요청 시간 간격을 늘려서 하나의 인터럽트 시 처리 가능한 패킷의 크기를 늘리는 방법(Interrupt coalescing) 등도 생각해 볼 수 있다.