Week_2

5장 ROS2의 중요 컨셉트

6장 ROS2의 특징 (ROS1과의 차이점을 중점으로)

7장 ROS2와 DDS

8장 DDS의 QoS

5장 ROS2의 중요 컨셉트

ROS2의 주요 특징

- ROS2는 ROS1의 10년 경험을 바탕으로 처음부터 산업용 소프웨어로 개발되었으며 ROS2의 설계, 개발 및 프로젝트 관리는 업계 이해 관계로부터 얻은 요구 사항을 기반으로 하고 있다.

1.벤더 선택 기능

- ROS2는 로봇공학 라이러리와 응용 프로그램을 통신 기능으로부터 분리하여 추상화 작업을 하였다.

2.오픈소스 라이브러리 채택

- ROS2 코드는 Apache 2.0라이센스를 기본 라이센스로 사용하여 넓은 범위에 사용할 수 있다.

3.글로벌 커뮤니티

- ROS는 10년 이상 ROS 소프트웨어에 기여하는 수십만 명의 개발자와 사용자를 기반으로 하는 시스템이고, ROS2는 이러한 에코 시스템이라 할 수 있다.

6장 ROS2의 특징 (ROS1과의 차이점을 중점으로)

1.ROS 개발의 필요성

ROS 1의 제한 사항은 다음과 같다.

- 단일 로봇
- 워크스테이션급 컴퓨터
- 리눅스 환경
- 실시간 제어 지원하지 않음
- 안정된 네트워크가 필요함
- 아카데믹 연구 용도

이러한 ROS 1의 제한 사항을 개선하기 위해 요구되는 새로운 로봇 개발 환경은 다음과 같다.

- 두 대 이상의 로봇
- 임베디드 시스템에서의 ROS사용
- 실시간 제어
- 불안정한 네트워크에서도 효과적으로 동작할 수 있어야 함
- 리눅스, 윈도우, macOS에서도 지원함
- DDS등 여러 최신 기술 지원
- 상업용 제품에 적용, 최적화

이러한 새로운 요구 사항을 적용하려면 대규모 API변경이 필요하다.

- 하지만 이러한 새로운 기능들을 추가하면서 호환성을 유지하는것은 쉽지 않으며
- 기존 ROS1의 API에 큰 변화를 주는것은 위험부담이 크다.

2.ROS2의 개선점

- Platform : Windows도 지원하게 되었다.
- Real-time의 측면 : External framefork 사용에서 →

  RTOS(Real-Time Operation System) 을 사용해서 node들을 Real-Time으로 제어할 수 있게 되었다.
- Security & Communication : DDS 통신과 DDS 보안을 지원하게 되었고,

  보안 측면에서 로봇 시스템에서 발생할 수 있는 Robotic Systems Threat Model도 지원하게 되었다.
- Node life cycle을 지원하게 되었다.
- 한 process에 Multiple node를 지원할 수 있게 되었다.
- mROS → microROS

3.ROS2의 특징

1. Platform적 특징
   - (Canonical은 리눅스 배포판을 개발하고 있는 회사이다.)
   - Canonical의 우분투 진영이 ROS 2 의 TSC(Technical Steering Commitee)의 일원이여서

     즉, ROS 2 의 로드맵을 제시하고 / 기술적인 결정을 하고 / 개발 방향성 결정하는 일원들이 리눅스 배포판을 개발하고 있는 회사의 일원이기 때문에

     리눅스 이용자라면 Canonical에서 연재하는 참고 자료를 활용한다면 ROS2 개발에 도움이 될 수 있다.

---

2. Real-time 지원

- ROS2는 실시간성을 지원한다.
- 단, 하드웨어, RTOS 사용, DDS의 RTPS(Real-time Publish-Subscribe Protocol)과 같은 프로토콜을 조건으로 한다.

---

3. DDS 채택으로 향상된 Security
   - ROS 1에서는 보안이 문제였지만, 개발을 더 우선으로 발전시켯으며 이 방향은 옳았다고 할 수 있다.

   - 하지만 이 점은 상용 로봇에 ROS를 도입할 수 없도록 만드는 요인이 되었고, 이를 개선하기 위해서 ROS2에서는 설계 단계부터 이를 고려하였다.
   - TCP통신은 산업에서 사용중인 DDS(Data Distributin Service)를 채택하였으며

     이에 따라 자연스럽게, DDS에서 사용하고 있던 DDS-Security를 이식하여 보안 문제를 해결할 수 있게 되었다.

4.Communication 방식의 개선

- ROS 1에서는 TCPROS와 같은 통신 라이브러리를 import해서 사용하였지만

  ROS 2에서는 DDS를 채택하였다.
- DDS에서는 노트 간의 자동 감지 기능을 지원하고 있기 때문에 ROS Master이 없어도 다른 노드들 간의 통신이 가능하다.
- DDS 프로토콜에서 제공되는 QoS(Quality of Service)를 사용하면

  → 사용자의 요구사항에 맞게 메시지의 중요도, 전송 속도, 신뢰성, 지연 시간 등을 제어할 수 있다.
  - QoS 프로파일의 종류
    - `SYSTEM_DEFAULT`
      : 시스템의 기본 설정 값을 사용한다.
    - `SENSOR_DATA`
      : 센서 데이터와 같이 손실이 발생하더라도 신뢰성이 높은 데이터를 전송한다.
    - `SERVICES_DEFAULT`
      : 서비스 요청과 응답에 대한 신뢰성과 지연 시간을 고려하여 QoS를 설정

한다.

- `PARAMETER_EVENTS`
  : ROS 2 파라미터 변경 이벤트를 전송한다.
- `PARAMETERS`
  : ROS 2 파라미터를 전송한다.

- TCP처럼 데이터 손실을 방지하거나
- UDP(User Data Protocol)처럼 통신속도를 최우선하여 사용하는 것이 가능하다.

5.Ros Middleware (RMW) 의 지원 방식

- DDS 벤더는 10여 곳이 있는데, 이 중에서 ROS2를 지원하는 곳은 5군데이다.

- 이중에 GurumNetworks(구름 네트워크)는 ROS 2 DDS를 순수 국산 기술로 개발해서 상용화에 성공하였다.

> 💡 ROS2에서 DDS는 RMW(로봇 미들웨어, Robot Middleware)의 형태로 사용할 수 있다.

6.Node manager의 형식 변화

- ROS1의 필수 실행 프로그램은 roscore이 있고,

  이를 실행시키면 ROS Master, ROS Parameter Server, rosout logging node가 실행되었는데

  → ROS Master의 연결이 끊기거나 죽는 경우 모든 시스템이 마비되는 단점이 있었다.

- ROS 2에서는 roscore의 역활이 3 가지의 독립적인 기능으로 완전히 대체되었다.
  - ROS Master →DDS의 기능으로 대체되었고
  - + 노드들은 DDS의 Participant로 취급하게 되었다.

- Dynamic Discovery를 이용하여 노드를 검색하여 연결할 수 있게 되었다.

  (DDS 미들웨어를 통해 사용한다.)

> 💡 ROS2에서 Node manager은 중앙집중적 형태에서 DDS를 활용한 분산형 형태가 되었으며, 이에 따라 마스터 노드의 연결이 끊기거나 불안정한 상태에서도 더욱 잘 작동할 수 있게 되었다.

---

7. Build System 개선
   - 빌드 시스템도 개선되었다.

     ROS1에서의 catkin → ROS2에서는 ament를 사용하게 되었다.

   - catkin은 CMake만을 지원했지만
   - ament는 CMake뿐만 아니라 CMake를 사용하지 않는 파이썬 패키지도 지원할 수 있게 되었다.

   - ROS1에서는 파이썬 코드가 있는 패키지가 setup.py의 사용자 로직으로 처리되었지만
   - ROS2에서는 파이썬 패키지가 독립을 이루게 되어서, 순수 파이썬 모듈과 동일하게 사용할 수 있게 되었다.

---

8. Build tool 과 Build option의 변화
   - ROS1은 여러 가지 빌드 도구가 지원 되었고, ROS2에서는 알파,베타, ardent 릴리즈까지 ament_tools가 지원되어왔고 지금은 colcon build tool을 추천하고 있다.

   →ROS2에서는 빌드 도구들이 변경되면서 빌드 옵션에도 변화가 생겼으며 이는 세 가지 특징을 가진다.

   - Muiltiple workspace : ROS1에서는 catkin_ws과 같은 하나의 워크스테이션에서 모든 작업을 진행하였으나

- ROS2에서는 복수의 독립된 워크스페이스를 사용하여 작업 목적과 패키지 목적별로 관리할 수 있게 되었다.

- No non-isolated build : ROS1에서는 하나의 Cmake파일로 여러 개의 패키지를 동시에 빌드할 수 있었지만 모든 패키지의 종속성을 신경써야 하고 + 패키지 빌드 순서를 고려해야 한다.
- 하지만 ROS2에서는 격리 빌드만을 지원하여, 모든 패키지를 별도로 관리하고 빌드할 수 있게 되었고, 패키지 종속성에 들이는 노력을 줄이고 / 패키지 빌드 순서를 고려하지 않아도 되게 되었다.

- No devel space : catkin은 빌드 후에 devel폴더에 코드를 저장하여 패키지를 설치하지 않고도 사용할 수 있도록 지원했는데, 이 기능은 패키지 관리의 복잡성을 크게 증가시켰다.
- ROS2에서는 패키지를 빌드하고 설치해야만 사용할 수 있도록 바뀌었는데,
- - -symlink -install : 빌드된 패키지를 시스템에 설치할 때 심볼릭 링크를 사용하는 옵션이다.
    - 원본 파일이나 디렉토리가 이동되어도 링크가 유효하다.

(Symbolic link : 파일 시스템에서 파일이나 디렉토리를 가리키는 포인터이다.)

9.Version control system

- ROS2에서는 버전 관리 시스템을 제공한다.

  ROS1에서는 wstool을 사용했고 → ROS2에서는 vsctool로 통합해 주었다.
- vcs는 YAML 파일 형식으로 작성된 ROS패키지와 + 종속성을 설명하는 vcs파일로 구성된다.
    - vcs파일에는 패키지의 저장소 위치 / 버전 등이 포함된다.

- 다음은 ROS2에서 vcstool을 사용하여 패키지를 사용하는 예시이다.
    - cd ~/ros2_ws/src →디렉토리 이동
      vcs import < path/to/package_repositories.yaml →vcs를 이용하여 YAML에 저장된 패키지와 종속성을 다운로드한다.

- cd ~/ros2_ws

  colcon build

  디렉토리 이동 후 빌드한다.

10.Client library

ROS 구조에서

- User land는 개발자가 Node, Topic, Service, Action등의 ROS기능을 이용하는 사용자 정의 프로그램을 작성하는 영역이다.
- ROS Client Library는 ROS의 핵심 기능을 제공하는 라이브러리이며 사용자 정의 프로그램과 함께 작동되고, Node간의 통신, Message 및 Service의 호출, 로봇 시스템의 환경 설정 등의 기능을 수행하고 조금 더 기반에 가까운 동작을 하는 라이브러리이다.

  ROS Client Library는 rospy, roscpp, rosnodejs등 여러 언어로 제공된다.

11.Lifecycle

- 로봇 개발에서 현재 상태 파악과, 현재 상태에서다른 상태로의 변경을 추적하는 것은 수십 년간 주요 연구 주제로 다루어졌다.

- ROS1에서는 SMACH와 같은 독립적인 패키지를 사용하여 이를 구현하였는데,

  → ROS2에서는 이 기능을 Lifecycle의 형태로 클라이언트 라이브러리에 포함시켰다.

12.Messages

- ROS2에서는 단일 데이터 구조를 메시지라고 정의하고, 정해지거나 / 사용자가 정의한 메시지를 사용할 수 있다. 그리고 고유한 이름으로 이 메시지들을 식별할 수 있다.

- ROS2에서는 OMG(Object Management Group)에서 정의한 IDL(Interface Description Language)를 사용하여 메시지를 더 쉽고 포괄적으로 다룰 수 있다.
    - Object Management Group(OMG)은 객체 지향 기술 및 분산 시스템 관련 기술을 연구, 개발 및 표준화하는 비영리 국제 표준화 단체이다.
    - Interface Description Language(IDL)은 객체 지향 프로그래밍에서 사용되는 인터페이스를 정의해주고, 중간 언어를 생성하는 언어이다.
        - →이를 이용해서 메시지,서비스,액션 등의 ROS2인터페이스를 정의한다

ROS2에서는 기존의 msg, srv, action이외에도 IDL을 지원한다.

13.Launch

- ROS의 실행 시스템은 run과 launch가 있다.

1. run은 단일 프로그램 실행하는 명령어이다.
2. launch는 사용자가 지정한 프로그램들을 실행하는 명령어이다.

   launch는 사용자가 기술한 설정에 맞추어 프로그램을 실행하며,

   사용자는 실행할 프로그램 / 실행할 위치 / 전달할 인수 등을 설정할 수 있다.

7장 ROS2와 DDS

1.DDS란?

DDS는 middleware protobol과 API(Application programming Interfate) stndard for data-centric connectivity이다.

- DDS는 미들웨어 프로토콜과 API 표준을 제공하며, 데이터 중심 연결을 위한 기술로 사용된다.

주요 특징

1. 언어 독립

- DDS는 미들웨어이므로, 사용자 코드 레벨에서 DDS를 사용하기 위해서 기존에 사용하던 프로그래밍 언어를 변경할 필요가 없다.
- ROS2에서는 이 특징을 살려 DDS를 ROS middleware으로 추상화하였고, 그 위에 ROS Client Library를 지원한다.

3.Dynamic Discovery

- DDS는 Dynamic Discovery(동적 검색)을 지원한다.

  이 기능을 활용하면, 검색을 해서

  노드 실행 정보

  토픽 메시지

  서비스 요청 및 응답

  액션목표 및 상태

  ROS2미들웨어

  를 추적할 수 있다.

- ROS2에 와서 마스터는 더 이상 노드 간의 중매 역활을 하지 않아도 되고,

노드를 DDS의 participant로 취급하면서, Dynamic Discovery를 이용하여 이를 연결할 수 있게 되었다.

2.ROS에서의 사용법

1.RMW의 변경

두 명령어를 실행하자

- $ ros2 run demo_nodes_cpp listener
- $ ros2 run demo_nodes_cpp talker

이 두 노드가 실행되었다는 것은 DDS, 즉 RMW를 사용하고 있다는 것이다.

이 경우에 ROS 2 humble의 default RMW인 **rmw_fastrtps_cpp**를 사용하여 빌드가 된 것이다.

RMW를 변경하여 사용할 수 있는데, 그 때에는

- $ export RMW_IMPLEMENTATION=<RMW 이름> 을 입력하여 변경할 수 있으며

  이 때의 이점은
  - C++이나 파이썬 등으로 지원하는 언어를 변경할 수 있음
  - DDS 기반 데이터 교환 rmw_cyclonedds나

    브로드캐스팅 기반 rmw_fastrtps으로 변경하거나 TCP/IP나 UDP등으로도 변환할 수 있다.
  - 성능 / 메시지 교환의 신뢰성과 안정성을 바꿀 수 있다

2.Domain의 변경

- ROS2는 UDP 멀티캐스트로 통신이 이루어지기 때문에 별도의 설정을 하지 않으면 동일 네트워크의 모든 노드가 연결된다.

- 이를 방지하기 위해서는 ROS namespace를 사용하거나, 다른 네트워크를 사용하거나, DDS의 Domain을 변경해야 한다.

Domain은 0에서 101까지의 정수를 사용할 수 있으며

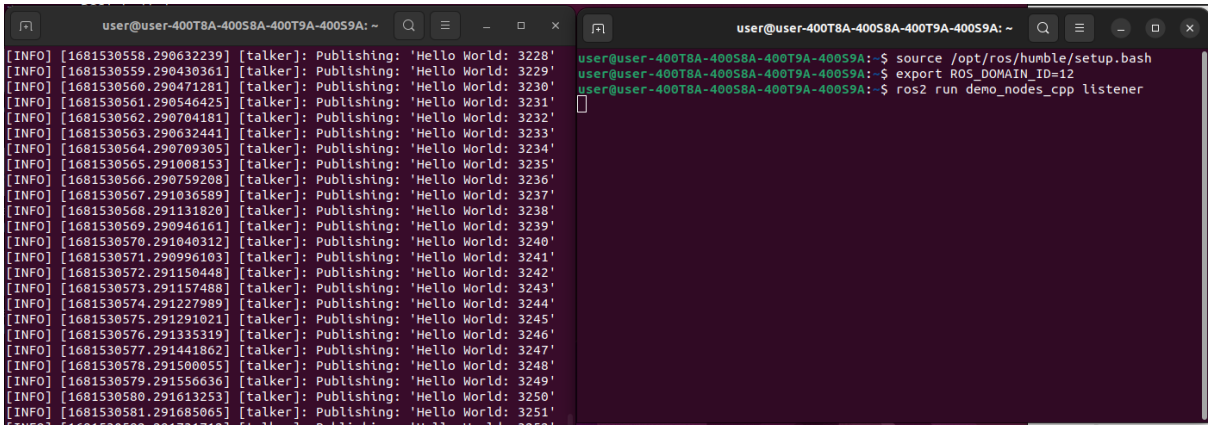
이는 ROS_DOMAIN_ID라는 환경 변수로 설정하여 사용할 수 있다.

사용예

- $ export ROS_DOMAIN_ID=11
- $ ros2 run demo_nodes_cpp talker

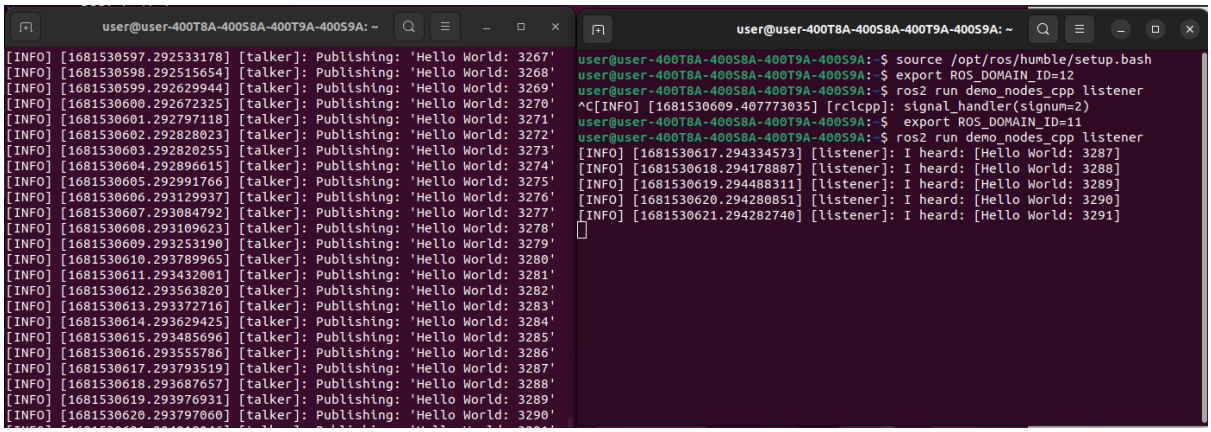
도메인 11으로 talker 실행

- $ export ROS_DOMAIN_ID=12
- $ ros2 run demo_nodes_cpp listener



반응없음

- $ export ROS_DOMAIN_ID=11
- $ ros2 run demo_nodes_cpp listener



같은 도메인이므로 talker 노드와 연결되었음

3. QoS 테스트

- 이번 테스트에서는 DDS의 QoS 설정과 관련된 테스트를 진행하였다.

  tc(traffic control)명령어를 사용하여 임의의 데이터 손실(10%)을 만든 후에 Reliability를 시험해 보았다.

1.Reliability가 RELIABLE으로 되어 있었을 때

- 명령어 입력

  $ sudo tc qdisc add dev lo root netem loss 10%

- 이 명령어는 tc(Traffic Control)을 이용하여 lo(Lookback) 인터페이스에 대한 qdisk(Queueing Discipline)을 추가하는 명령어이다.
  - tc : Traffic Control 명령어이다.

- qdisc : Queueing Discipline으로, 큐의 동작 방식을 정의한다.
- add : 새로운 qdisc를 추가한다.
- dev lo : device loopback으로, loopback인터페이스를 대상으로 qdisc를 추가한다.
- root : 가장 상위 수준의 큐를 추가한다, 여기서 상위 수준이란 먼저 처리한다는 것을 의미한다.
- netem : Network Emulation, 네트워크 패킷을 조작한다
- loss 10% : 로스율 설정

- 명령 입력

  $ ros2 run demo_nodes_cpp talker

  → **$ ros2 run demo_nodes_cpp listener**

  : 메시지 손실은 없지만 손실된 데이터를 재전송하기 떄문에 터미널 창에 지연이 생김

2.Reliability 가 BEST_EFFORT로 되어있을 때

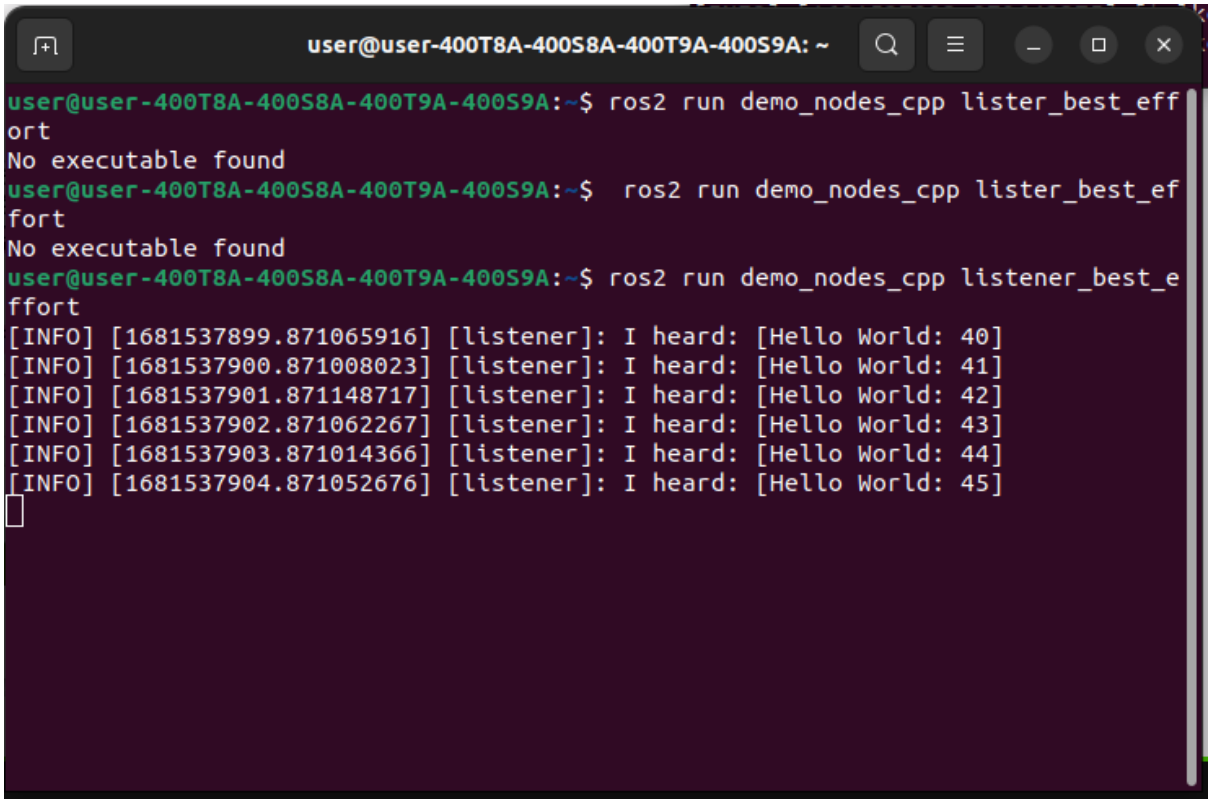
- 이번에는 demo_nodes_cpp 패키지의 lister_best_effor 노드를 사용한다
- 명령어 입력

(환경:

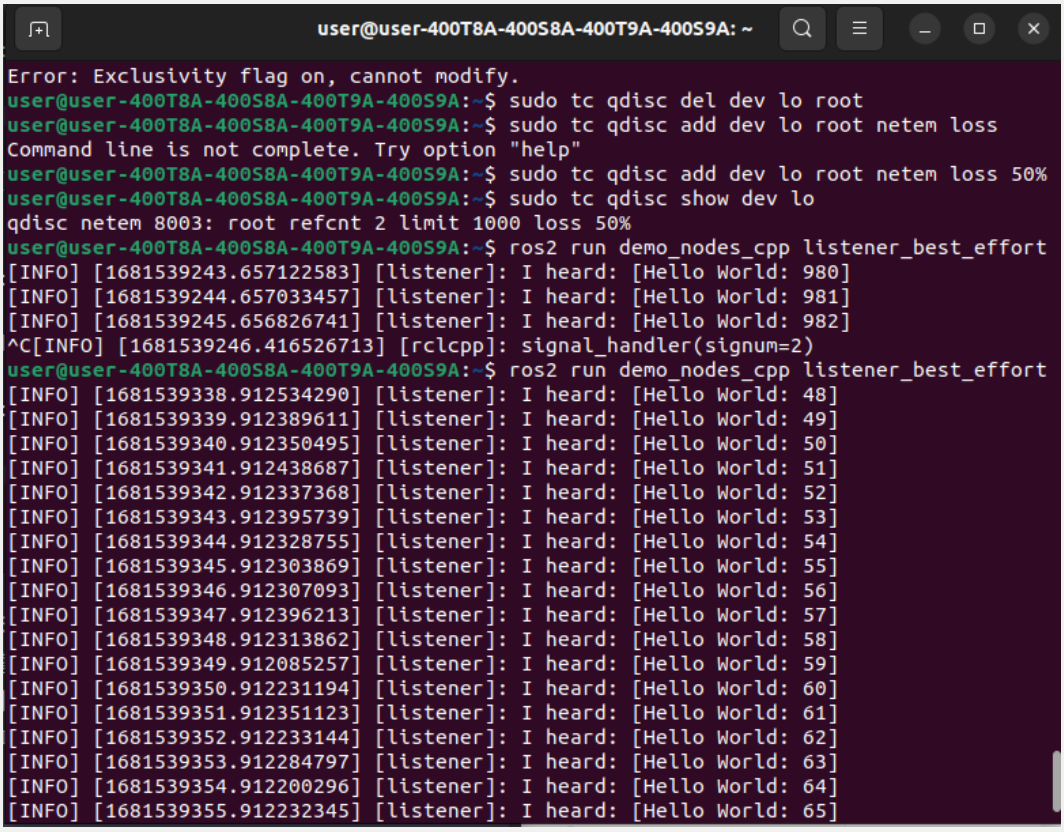
$ sudo tc qdisc add dev lo root netem loss 10%

$ ros2 run demo_nodes_cpp talker)

$ ros2 run demo_nodes_cpp listener_best_effort


user@user-400T8A-400S8A-400T9A-400S9A: ~
user@user-400T8A-400S8A-400T9A-400S9A:~$ ros2 run demo_nodes_cpp lister_best_eff
ort
No executable found
user@user-400T8A-400S8A-400T9A-400S9A:~$ ros2 run demo_nodes_cpp lister_best_ef
fort
No executable found
user@user-400T8A-400S8A-400T9A-400S9A:~$ ros2 run demo_nodes_cpp listener_best_e
ffort
[INFO] [1681537899.871065916] [listener]: I heard: [Hello World: 40]
[INFO] [1681537900.871008023] [listener]: I heard: [Hello World: 41]
[INFO] [1681537901.871148717] [listener]: I heard: [Hello World: 42]
[INFO] [1681537902.871062267] [listener]: I heard: [Hello World: 43]
[INFO] [1681537903.871014366] [listener]: I heard: [Hello World: 44]
[INFO] [1681537904.871052676] [listener]: I heard: [Hello World: 45]

오류 1



로스율을 50%로 설정해도, 로스가 하나도 발생하지 않았다.

(기존 qdisc 제거 :$ sudo tc qdisc del dev lo root)

~~Queue는 자료구조 중 하나로 목록 형태로 데이터를 저장하는 선형 자료구조이다.~~

- ~~큐는 데이터를 삽입하는 enqueue와 데이터를 인출하는 dequeue를 지원한다.~~
- ~~FIFO(선입선출, First in First Out) 원칙에 따라 동작한다.~~
- ~~선형 큐는 배열을 사용할 때 구현한다.~~
- ~~원형 큐는 배열의 처음과 끝이 연결되어 원형으로 돌아가는 구조를 보인다.~~

- ~~너비 우선 탐색(BFS, Breath First Search)알고리즘, 프로세스 스케줄링, 버퍼 등에 사용된다.~~

~~loopback은 데이터를 네트워크로 보내지 않고 내부적으로 자기 자신으로 돌려주는 인터페이스를 의미한다.~~

~~echo명령어는 문자를 출력하는 명령어이다.~~

~~셸에서 **echo "source /opt/ros/humble/setup.bash" >> ~/.bashrc**~~

~~를 입력하면 ~/.bashrc파일의 끝에 source /opt/ros/humble/setup.bash 내용이 추가된다.~~

8장 DDS의 QoS

1.QoS

- QoS(Quality of Service) 는 DDS의 서비스 품질이고,

  DDS사양서에서 설정 가능한 QoS 항목은 22가지이지만, ROS2에서는 Reliability가 대표적으로 사용된다.

  그 외의 QoS항목은 다음과 같다

  - 통신 상태에 따라 정해진 만큼의 데이터를 보관하는 History
  - Subscriber생성 이전의 데이터를 사용할지 폐기할지 결정하는 Durability
  - 정한 주기 내에 데이터 발신/수신이 없으면 이벤트 함수를 실행시키는 Deadline
  - 정한 주기의 데이터가 아니면 삭제하는 Lifespan
  - 정한 주기동안 노드/토픽의 생사를 확인하는 Liveliness

2.QoS 옵션들

1.**History**옵션

- Values
  - KEEP_LAST : 정해진 메시지 큐 크기만큼 데이터를 보관한다.
  - KEEP_ALL : 모든 메시지를 보관한다.
- RxO(Requested by Offered)

  RxO는 상대방이 요청한 신뢰성 수준에 따라 메시지를 전송하도록 설정하는 것이다.
  - 해당X
- Examples

  [RCLCPP]
  - rclcpp::Qos(rclcpp::KeepLast(10));

  [RCLPY]
  - qos_profile = QoSProfile(hostory=QoSHistoryPoilcy.KEEP_LAST, depth = 10)

2.Reliability옵션

- Values
  - BEST_EFFORT : 데이터 송신에 집중한다. (유실이 발생할 수 있다.)
  - RELIABLE: 데이터 수신에 집중한다. (재전송을 통해 수신을 보장한다.)
- RxO(Requested by Offered)

PUB \ SUB	BEST_EFFORT	RELIABLE
BEST_EFFORT	BEST_EFFORT	불가
RELIABLE	BEST_EFFORT	RELIABLE

  PUB: RELIABLE / SUB:BEST_EFFORT 상황에 →BEST_EFFORT로 응답하였다.

  PUB: BEST_EFFORT / SUB: RELIABLE 상황에 → RELIABLE로 응답하였다.
- Examples

  [RCLCPP]
  - rclcpp::QoS(rclcpp::KeepAll).best_effort();

  [RCLPY]
  - qos_profile = QoSProfile(reliability=QoSliabilityPolicy.BEST_EFFORT)

3.Durability

- Values
  - TEANSIENT_LOCAL :Subscription이 생성되기 전의 데이터도 보관한다.
  - VOLATILE : Subscription이 생성되기 전의 데이터는 무효로 한다.
- RxO(Requested by Offered)

PUB \ SUB	TEANSIENT_LOCAL	VOLATILE
TEANSIENT_LOCAL	TEANSIENT_LOCAL	VOLATILE
VOLATILE	불가	VOLATILE

- Examples

[RCLCPP]

- rclcpp::QoS(rclcpp::KeepAll).best_effort();

[RCLPY]

- qos_profile = QoSProfile(reliability=QoSliabilityPolicy.BEST_EFFORT)

4.Deadline

- Values
  - deadline_duration : Deadline을 확인하는 주기이다.

    : Deadline은 정해진 주기 안에 데이터가 발신 및 수신되지 않으면 EventCallback을 실행시킨다.
- Examples
  - rclcpp::QoS(10).deadlind(500ms)
  - qos_profile = QoSProfile(depth =10, deadline=Duration(0.1))

그 외에는 Lifespan / Liveliness가 있다.

3.rmw_qos_profile

ROS2에서는 QoS의 설정을 쉽게 하기 위해서 설정을 세트로, RMW QoS Profile의 형태로 표현해 놓았다.

- 이는 목적에 따라 6가지의, Data / Service / Action Status / Patameters / Parameter Events 로 구분되고

- 각각의 4가지 속성인 Reliability / History / Depth / Durability를 설정할 수 있다.

4. 유저 QoS 프로파일

3번에서, 6가지로 구분되는 형태 외에도 다른 형태의 QoS의 프로파일을 유저가 작성하여 사용할 수 있다.

이를 위해서 rclpy.qos로부터 각각

- QoSDurabilityPolity / QoSHistoryPolicy / QoSProfile / QoSReliabilityPolicy를 임포트한 다음에

  Reliability / History / Depth / Durability을 설정한 후에

  create_publisher 메서드를 사용할때 사용자 정의 QoS 프로파일을 사용하면 된다.