Descubrimiento de nodos en ros de distintas maquinas por wifi

Introducción

Ros2 solo permite multicast a través de DDS. Por defecto DDS usa multicast para descubrir otros nodos pero esto solo es posible si la red soporta multicast.

La Autoridad de Números Asignados de Internet (IANA) ha definido el rango de direcciones IP multicast como 224.0.0.0 a 239.255.255.255, comúnmente conocido como direcciones de Clase D. Las direcciones en el rango de 224.0.2.0 a 238.255.255.255 suelen estar disponibles para su asignación a discreción de organizaciones individuales y proveedores de servicios de Internet (ISPs). Es importante tener en cuenta que una dirección IP multicast solo especifica los destinatarios de los paquetes multicast, y la dirección IP del remitente es una dirección IP unicast estándar.

Validación de la Disponibilidad de Multicast:

Sistema A (servidor)
iperf -s -u -B 224.1.1.1 -i 1

En la segunda máquina, ejecuta iperf en modo cliente, apuntando a la misma dirección multicast:
Sistema B (cliente)
iperf -c 224.1.1.1 -u -T 32 -t 1 -i 1

La salida en el Sistema A debería parecerse a lo siguiente después de ejecutar el cliente en el Sistema B:

Servidor escuchando en el puerto UDP 5001
Uniéndose al grupo multicast 224.1.1.1
Servidor configurado en modo de tráfico de un solo cliente (por recepción multicast)
Tamaño del búfer UDP: 208 KByte (predeterminado)
---
[1] local 224.1.1.1 port 5001 connected with 192.168.0.13 port 47708
[ID] Interval Transfer Bandwidth Jitter Lost/Total Datagrams
[1] 0.0000-0.9215 sec 134 KBytes 1.19 Mbits/sec 0.986 ms 0/93 (0%)
[1] 0.0000-0.9215 sec 134 KBytes 1.19 Mbits/sec 0.986 ms 0/93 (0%)

Si la máquina del servidor recibe los paquetes enviados por el cliente, entonces la multidifusión está funcionando en tu red.

Otra forma de hacerlo es utilizando ros2 multicast test en dos máquinas separadas y verificar si el receptor puede recibir paquetes.
PC1:
$ ros2 multicast receive
PC2:
$ ros2 multicast send

Ref2:

Los middleware de DDS en ros2 nos permiten elegir qué implementación de DDS deseamos utilizar. En la versión más reciente de ros2, la implementación por defecto es FastDDS (que generalmente no es la mejor opción). La gran mayoría de usuarios suele cambiar esta implementación por Cyclone.

Para seleccionar la implementación de DDS, puedes utilizar la variable de entorno RMW_IMPLEMENTATION. Por ejemplo:

export RMW_IMPLEMENTATION=cyclonedds
export RMW_IMPLEMENTATION=rmw_fastrtps_cpp
or
export RMW_IMPLEMENTATION=rmw_fastrtps_dynamic_cpp

Cuando lanzas tu aplicación ROS 2:
RMW_IMPLEMENTATION=rmw_fastrtps_cpp ros2 run <package> <application>
o
RMW_IMPLEMENTATION=rmw_fastrtps_dynamic_cpp ros2 run <package> <application>

Pruebas:

Condiciones que nunca cambiaban:

- En mi casa alejado del router:
- Descubrimiento de nodos entre un PC y una raspberry pi4b
- Tras cada cambio de implementación de dds se reiniciaba el demonio de ros2 en ambas máquinas con: $ ros2 daemon stop && ros2 daemon start

Test1: red2G, dds por defecto y ros_domainid = 1
pc:
$ ros2 multicast receive
Raspberry
$ ros2 multicast send

Resultado: éxito

Test2: red5G, dds por defecto y ros_domainid = 1:
pc:
$ ros2 multicast receive
Raspberry
$ ros2 multicast send

Resultado: éxito

Test3: red2G, dds rmw_cyclonedds_cpp y ros_domainid = 1:
pc:
$ ros2 multicast receive
Raspberry
$ ros2 multicast send

Resultado: éxito
Test4: red5G, dds rmw_cyclonedds_cpp y ros_domainid = 1:
pc:
$ ros2 multicast receive
Raspberry

$ ros2 multicast send

Resultado: éxito
Test5: red2G, dds rmw_fastrtps_cpp y ros_domainid = 1:
pc:
$ ros2 multicast receive
Raspberry
$ ros2 multicast send

Resultado: éxito

Test6: red5G, dds rmw_fastrtps_cpp y ros_domainid = 1:
pc:
$ ros2 multicast receive
Raspberry
$ ros2 multicast send

Resultado: error red inalcanzable

Test7: reintento de test anterior:
pc:
$ ros2 multicast receive
Raspberry
$ ros2 multicast send

Resultado: éxito

Test8: red2G, dds rmw_fastrtps_cpp y ros_domainid = 1:
pc:
$ ros2 multicast receive
Raspberry
$ ros2 multicast send

Resultado: éxito

Test9: red2G raspberry y red5G el pc, dds por defecto y ros_domainid = 1:
pc:
$ ros2 multicast receive
Raspberry
$ ros2 multicast send

Resultado: éxito

Test10: red5G, dds por defecto y ros_domainid = 1:
pc:
$ ros2 multicast receive
Raspberry
$ ros2 multicast send

Resultado: éxito

Conclusiones

Sorprendentemente, usando el MRW por defecto y Cyclone, ha funcionado sin problemas. Sin embargo, al utilizar FastRTPS, solo ha funcionado con la red 2G; con la 5G, no ha funcionado a la primera. Al repetir el test, me di cuenta de que la Raspberry sufría desconexiones de forma intermitente de vez en cuando en la red 5G. Supongo que esto se debe a que la tarjeta de red no es muy buena y a la distancia que hay hasta el router.

Lo que me ha sorprendido es que, en verano cuando empecé con mi proyecto, el descubrimiento de nodos mediante Wi-Fi fallaba el 80% de las veces. Hoy me ha funcionado en todos los casos que he probado, reiniciando el demonio de ROS en cada cambio de implementación y reiniciando los dispositivos para comprobar si había sido suerte o si realmente funcionaba.

Creo que esto puede deberse a que el otro día, mientras probaba esto con el Wi-Fi de la universidad (Eduroam y Wireless_URJC), habilité el multicast en la interfaz Wi-Fi de la Raspberry y en la de mi PC (según indicaba en internet que probara). Igual es por eso, o no estoy seguro de si antes no estaban habilitados.

Igualmente después de haber realizado este test he seguido experimentando problemas de nodos que no se descubren por wifi por lo que los resultados del test realizado no deberían tomarse como una garantía de éxito.

Detalles de DDS para ROS:

Fast DDS proporciona los estándares de protocolo de cable interoperables OMG DDS 1.4 y OMG RTPS 2.2 consiguiendo una implementación sorprendentemente rápida. Dicha implementación tiene una serie de ventajas muy importantes que deberemos de tener en cuenta en el momento de seleccionar una implementación DDS. Entre estas encontramos un alto rendimiento, baja latencia, fácil integración multiplataforma, cumplimiento de estándares DDS 1.4 y RTPS 2.2, servicios personalizados, etc.

Referencias:

1:https://medium.com/@arshad.mehmood/setting-up-node-discovery-across-multiple-systems-in-ros2-infrastructure-a1a5c25f052f
2:https://docs.ros.org/en/humble/Concepts/Intermediate/About-Different-Middleware-Vendors.html
3:https://ddd.uab.cat/pub/tfg/2022/tfg_1634553/TFG_ROS2_InformeFinal.pdf