Demystifying ROS 2 Networking

Hilary Luo
Luis Camero
Roni Kreinin
Tony Baltovski

Agenda

- Introduction to ROS Networking
- Networking Fundamentals
- DDS
- RMW Implementations
- Fast DDS Simple Discovery
- Fast DDS Discovery Server
- RMW Zenoh
- Troubleshooting

ROS 2 Networking

Intro to ROS

The Robot Operating System (ROS) is an open-source framework that provides a structured set of tools and libraries designed to help developers build and program robotic systems. It provides:

- Middleware
- Communication Infrastructure
- Libraries and Tools
- Simulation
- Modularity
- Community and Ecosystem

ROS 1 vs ROS 2

	ROS 1	ROS 2
Middleware	Utilizes custom TCP/IP and UDP	Utilizes the Data Distribution Service (DDS)
Architecture	Relies on a central "master" node for coordination	Does not rely on a central master node, uses DDS discovery methods
Communication and Data Handling	Uses ROS-specific messaging protocols and tools like rosbag for logging data. Communication between nodes is managed by the ROS Master, which can introduce latency and dependency issues.	Uses the DDS standard for communication, which supports more sophisticated QoS (Quality of Service) settings.
Design	Designed with real-time capabilities and built-in security features using DDS	Additional development needed for real-time capabilities and security

ROS concepts

- **Nodes**: Modular software components that perform computation and communicate with other nodes in a ROS 2 system
- **Topics**: Communication channels that allow nodes to exchange messages asynchronously
- **Publishers**: Nodes that send messages over a topic, sharing data with any nodes subscribing to that topic
- **Subscribers**: Nodes that listen to messages on a specific topic, receiving data from publishers
- **Services**: Provide synchronous communication where one node sends a request and waits for a response from another node
- **Actions**: Allow asynchronous communication where a node can request a long-running task, receive feedback, and get the result once it's completed

Networking is important for ROS?

- **Enables Distributed Systems**: Networking allows ROS nodes to communicate across different devices and platforms, enabling real-time data exchange between components
- **Connects Nodes**: Networking links Publishers and Subscribers over topics, and facilitates Services and Actions across nodes, whether they are on the same machine or distributed across a network
- **Scalability**: With networking, ROS systems can scale from small, single-robot setups to large, multi-robot systems with nodes distributed across multiple machines
- **Modular Collaboration**: Nodes can be developed independently and deployed across different hardware, making robotics systems more flexible and easier to update

Networking Fundamentals

What is Networking

- What is Networking?
  - Definition: Networking involves connecting multiple computers or devices or **robots** to share resources and information
  - Purpose: To facilitate communication and resource sharing
- Network Types
  - Local Area Network (LAN):
    - Definition: A network that covers a small geographic area, like a home or office
    - Example: Wi-Fi in a home
  - Wide Area Network (WAN):
    - Definition: A network that covers a large geographic area, such as a city, country, or even global
    - Example: The Internet

Basic Network Components

- **Modem**: Converts digital data from a computer into analog signals for transmission over other media such as cable or fiber
- **Router**: Directs data between devices on a network and external networks
- **Firewall**: Controls incoming and outgoing network traffic based on security rules
- **Switch**: Connects multiple devices within a LAN and manages data traffic between them
- **Wireless Access Point**: Converts wired connections to wireless signals

IP addressing

- Definition: An IP address is a unique identifier for a device on a network.
  - IPv4: 32-bit address (e.g., 192.168.1.1).
  - IPv6: 128-bit address (e.g., 2001:0db8:85a3:0000:0000:8a2e:0370:7334).
- Dynamic vs. Static IP Addresses:
  - Dynamic: Assigned by DHCP, changes over time.
  - Static: Manually assigned, remains constant.
- Private vs. Public
  - Pubic: On the internet or WAN
  - Private: Local or LAN
- Subnetting
  - Definition: The practice of dividing a network into smaller, more manageable sub-networks (subnets).
  - Example: 192.168.1.1/24 or 10.0.0.0/8

Networking Ports

- Definition: Ports are virtual endpoints used for network communication. They help direct data to specific processes or services on a device.
- Port Numbers:
  - Well-Known Ports (0-1023): Reserved for common services (e.g., HTTP - 80, HTTPS - 443).
  - Registered Ports (1024-49151): Used by applications or services (e.g., MySQL - 3306 or ROS 1 - 11311).
  - Dynamic/Private Ports (49152-65535): Used for temporary connections (e.g., by applications).
- Purpose: Facilitate the routing of data to the correct application or service on a device.

DHCP

DHCP (Dynamic Host Configuration Protocol)

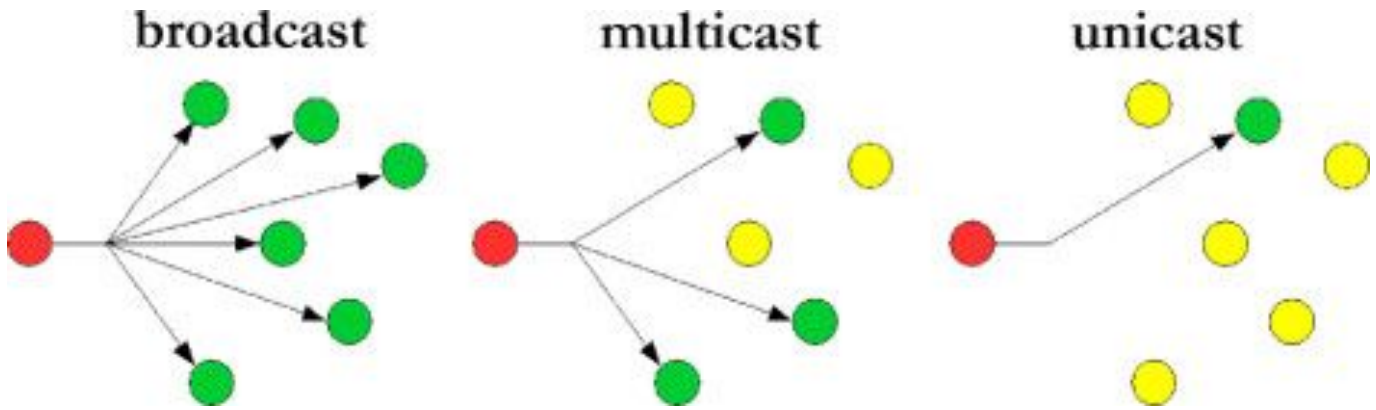
- Function: Automatically assigns IP addresses and network settings to devices.
- How It Works:
  - Discovery: Device searches for a DHCP server.
  - Offer: Server provides an IP address and settings.
  - Request: Device requests the offer.
  - Acknowledge: Server confirms the IP address and settings.
- Purpose:
  - Simplifies network setup.
  - Avoids IP address conflicts.
  - Automatically updates network settings.

DNS

- DNS (Domain Name System)
- Translates domain names (ie www.robot.com) into IP addresses (ie 1.2.3.4).
- How It Works:
  - Request: Your computer asks a DNS server for the IP address of a domain.
  - Query: The DNS server checks its cache or asks other servers if needed.
  - Response: The IP address is sent back, allowing your computer to connect to the website.
- Purpose: Makes it easier to access hosts using readable names rather than numbers.

Transmission types

- **Unicast**: One-to-one communication where data is sent from a single sender to a single receiver.
- **Multicast**: One-to-many communication where data is sent from one sender to a specific group of receivers.
- **Broadcast**: One-to-all communication where data is sent from one sender to all devices on a network.



Network Protocols

- Definition: A set of rules that determine how data is transmitted over a network.
- Common Protocols:
  - TCP (Transmission Control Protocol):
    - Reliable: Ensures that data is delivered accurately and in the correct order. It uses acknowledgments (ACKs) and retransmissions to handle packet loss, duplication, or corruption.
    - Error Checking: TCP performs error checking and correction.
    - Uses: Web browsing, file transfers
  - UDP (User Datagram Protocol):
    - Unreliable: Does not guarantee delivery, order, or error-free transmission. Packets may be lost, duplicated, or received out of order without any automatic recovery.
    - Error Checking: UDP includes a checksum for error detection, but there is no error recovery or correction.
    - Live streaming, online gaming, VoIP

Network Protocols (con’t)

- Common Protocols:
  - NTP (Network Time Protocol): A protocol used to synchronize the clocks of computers over a network to ensure accurate and consistent timekeeping across systems.
    - Uses: Syncing clocks between computers/devices

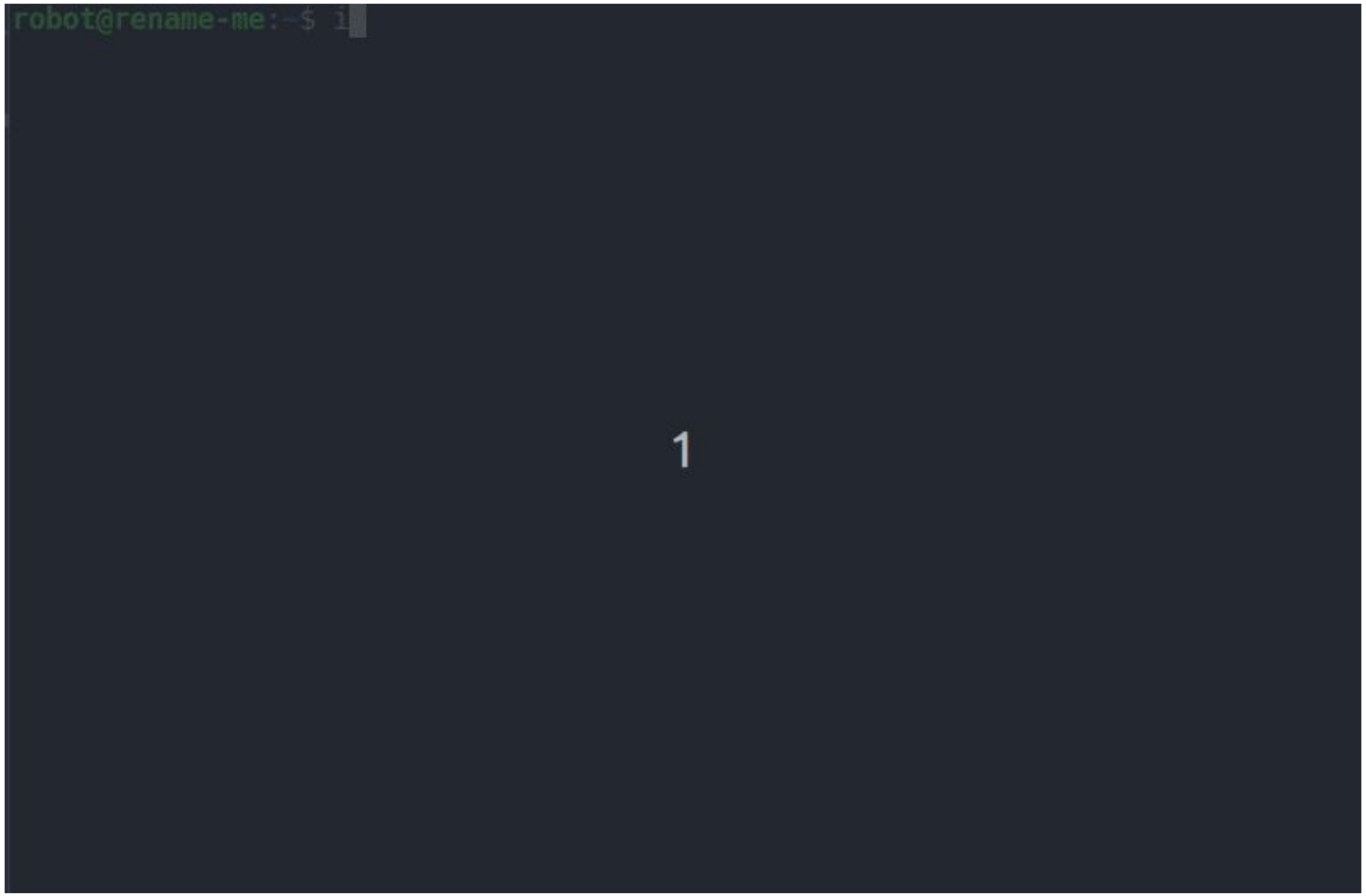
Host types

- Bare Metal
  - Definition: Physical servers.
  - Features: Direct hardware access, high performance, no virtualization overhead.
- Virtual Machines (VMs)
  - Definition: Virtualized computers with their own OS.
  - Features: Full OS per VM, more resource usage, strong isolation.
- Containers
  - Definition: Lightweight units with app and dependencies.
  - Features: Shared OS kernel, efficient, fast, ideal for microservices.
- WSL2 (Windows Subsystem for Linux 2)
  - Definition: Linux running on Windows.
  - Features: Full Linux kernel in a lightweight VM, integrates with Windows.

Ex. 1a Ping

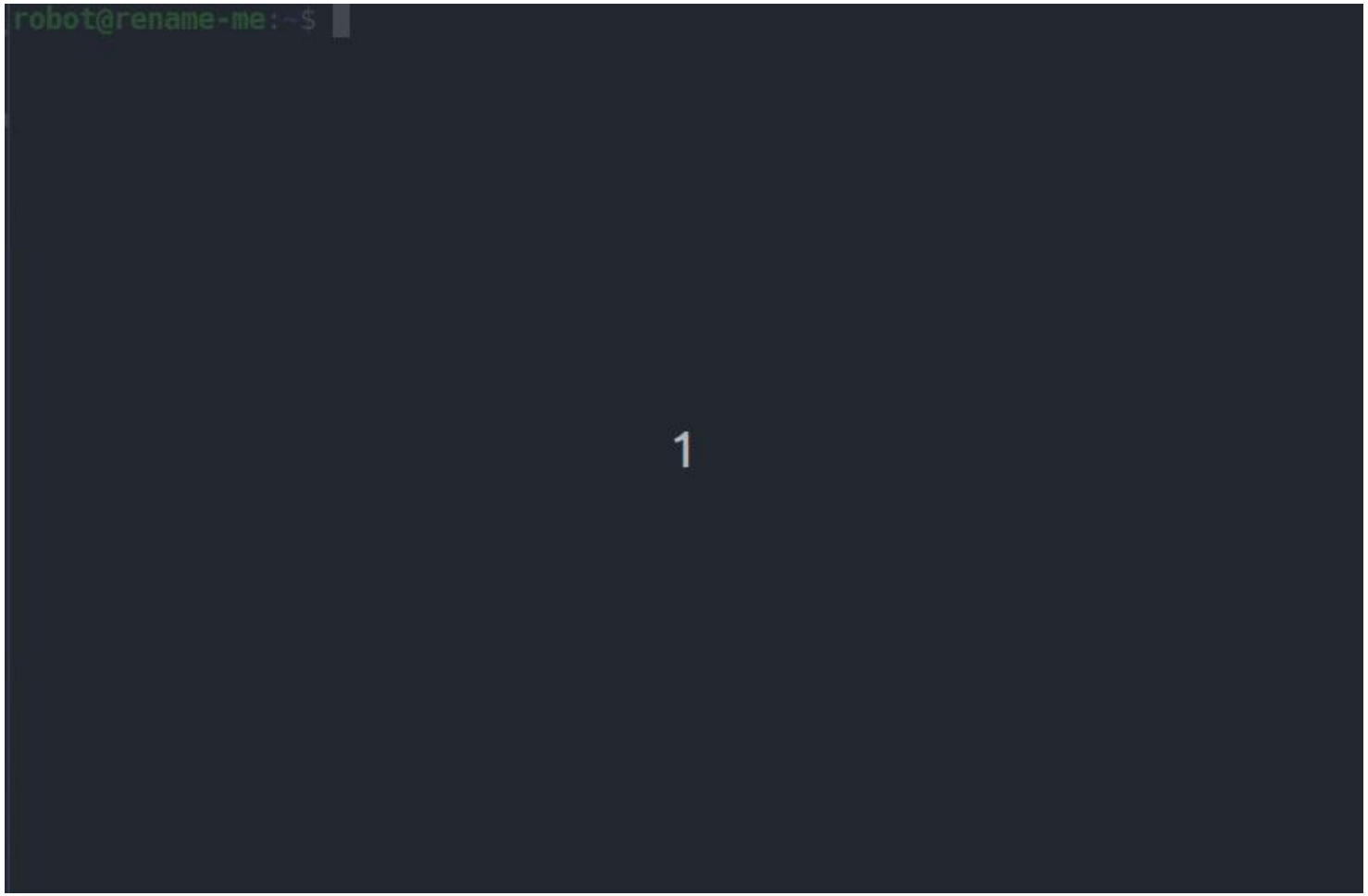
Determine your IP address using

```
ip a
```



Ex. 1a Ping

Ping your neighbour
`ping <partners_ip>`



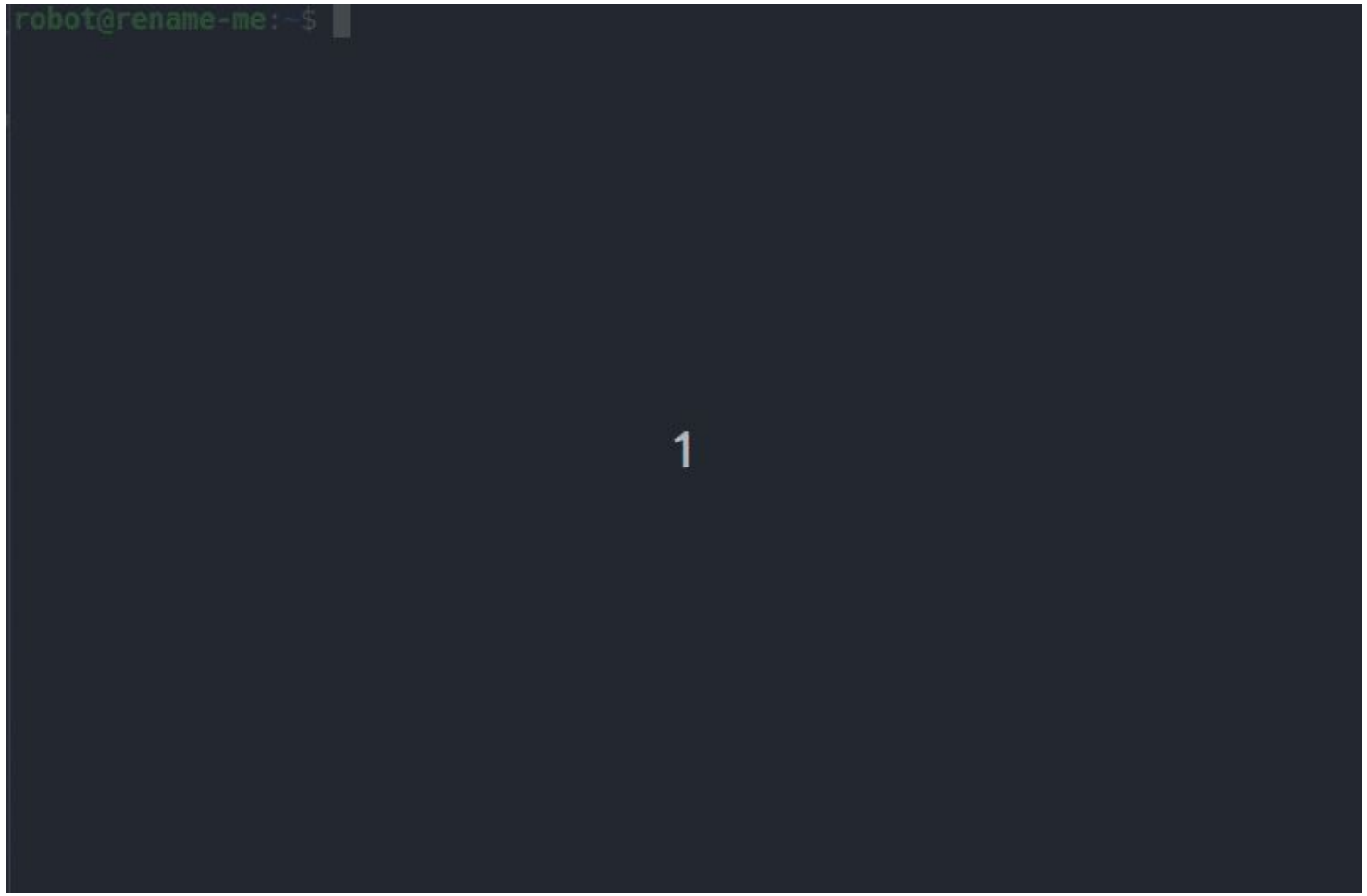
Ex. 1 - Linux Networking

Determine your IP address using
`ip a`
Ask your neighbour for their IP address and ping it using
`ping <partners_ip>`

Ex. 1b Nmap

Use nmap to find other devices on the network

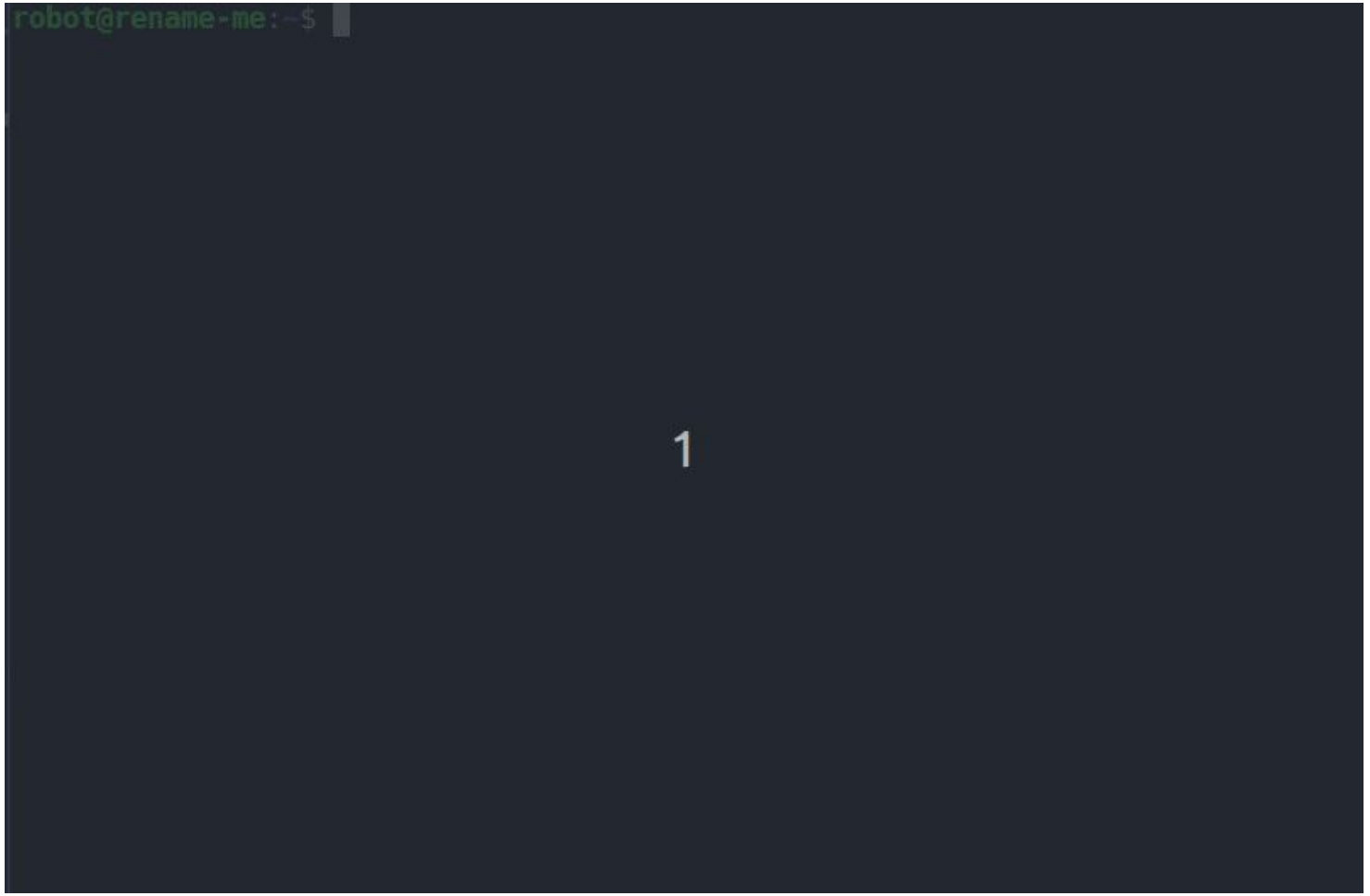
```
nmap -sP 192.168.0.1/24
```



Ex. 1c Performance

- Pair up again
  - One person be the server and the other person be the client

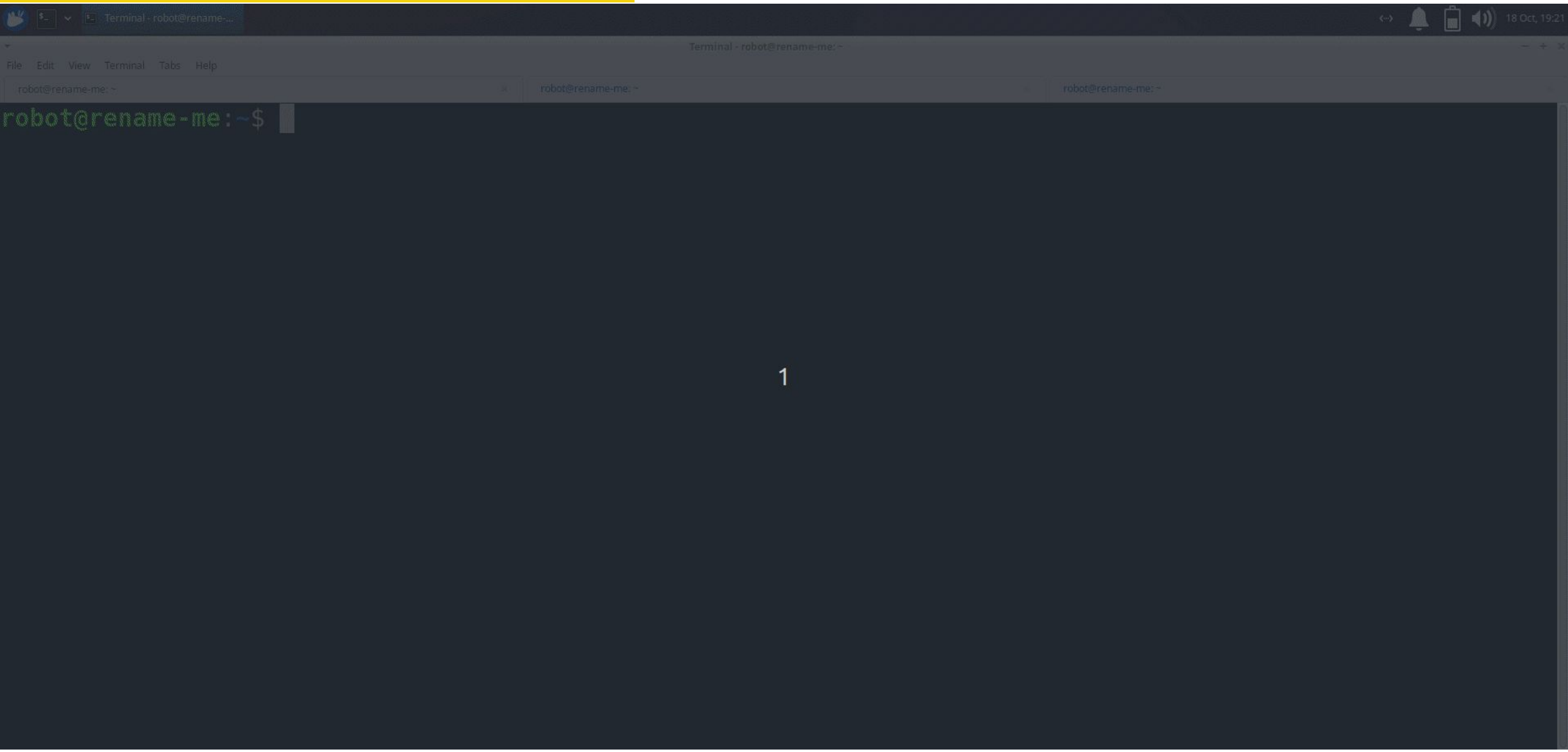
If you are the server, run
`iperf3 -s`
If you are the client, run
`iperf3 -c <partners_ip>`



Ex. 1b Wireshark

Run
`sudo wireshark`

Ex. 1b Wireshark



Data Distribution Service (DDS)

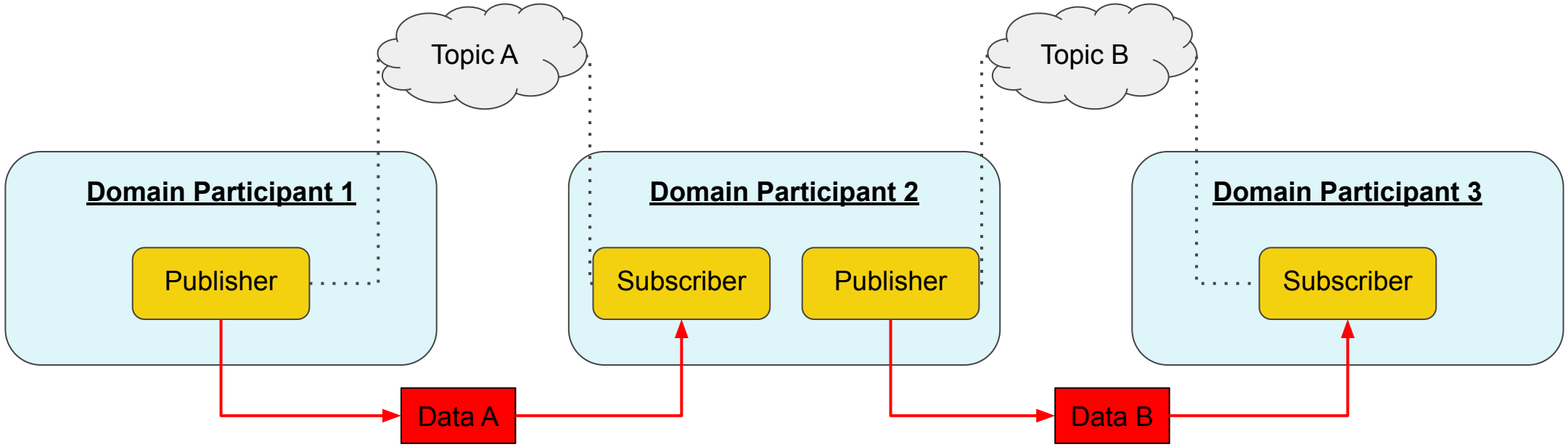
Purpose of DDS

- Real-time applications, in robotics and other industries, require scalable, predictable data distribution with minimal overhead.
- Classic shared memory distribution works but is difficult to implement over a network and does not scale.
- Before the DDS specification, there were commercially available products that were proprietary and did not have standardized interfaces.
- ***“The purpose of the DDS specification is to define the standardized interfaces and behaviors that enable application portability.”***

What is DDS?

- DDS is a specification describing a Data-Centric Publish Subscribe (DCSP) model for distributed application communication.
- Specification requirements:
  - **high performance**: pre-allocate resources to minimize dynamic resource allocation.
  - **predictable**: avoid properties that may use unbounded or hard-to-predict resources.
  - **efficient**: minimize copies of the data.
- In the DCSP model, there is a “global data space”:
  - data is identified by a “topic” and a “type”.
  - applications that contribute information are “Publishers”.
  - applications that access information are “Subscribers”.
  - a middleware handles data propagation from “Publishers” to “Subscribers”.

What is DDS?



Overview of DDS specification

Quality of Service

- QoS are controllable parameters that affect predictability, overhead, and resource allocation of a topic communication behaviour.
- Depending on the QoS, topic data can be sent as UDP, for low latency applications, or as TCP for applications that require reliable data delivery.
- QoS policies can be customized and set on specific publishers or subscribers, without having to deal directly with the middleware implementation directly.

QoS Policies

Policy	Settings
History: number of samples to store.	*Keep last*: store up to **N** samples. *Keep all*: store all samples up to middleware limits.
Depth: size of the queue	*Queue size*: if policy was set to Keep last, store this number of samples
Reliability: guarantees on data delivery.	*Best effort*: attempt to delivery, but may lose samples. *Reliable*: guarantee that sample is delivered, but may require multiple tries.
Durability: persistent samples to deliver.	*Transient local*: publisher keeps persistent samples for new subscriptions that missed previous message. *Volatile*: no persistent samples.

QoS Policies

Policy	Settings
Deadline: time between subsequent messages.	*Duration*: expected amount of time between subsequent messages.
Lifespan: amount of time to delivery.	*Duration*: maximum time between message publishing and reception before it is considered expired.
Liveliness: process to determine state of publisher.	*Automatic*: system will consider all of a node's publishers to be alive for a lease duration of time after any of the node publisher has published a message *Manual by topic:* each publisher will be considered independently alive for a lease duration after it publishes a message.
Lease Duration: amount of time to maintain liveliness.	*Duration*: maximum amount of time a publisher has to indicate that it is alive.

QoS Profiles

Default Profile	`rmw_qos_profile_default`
History	*Keep last*
Depth	**10**
Reliability	*Reliable*
Durability	*Volatile*
Deadline	Implementation default
Lifespan	Implementation default
Liveliness	Implementation default
Lease Duration	Implementation default

Sensor Profile	`rmw_qos_profile_sensor_data`
History	*Keep last*
Depth	**5**
Reliability	*Best effort*
Durability	*Volatile*
Deadline	Implementation default
Lifespan	Implementation default
Liveliness	Implementation default
Lease Duration	Implementation default

QoS Profiles

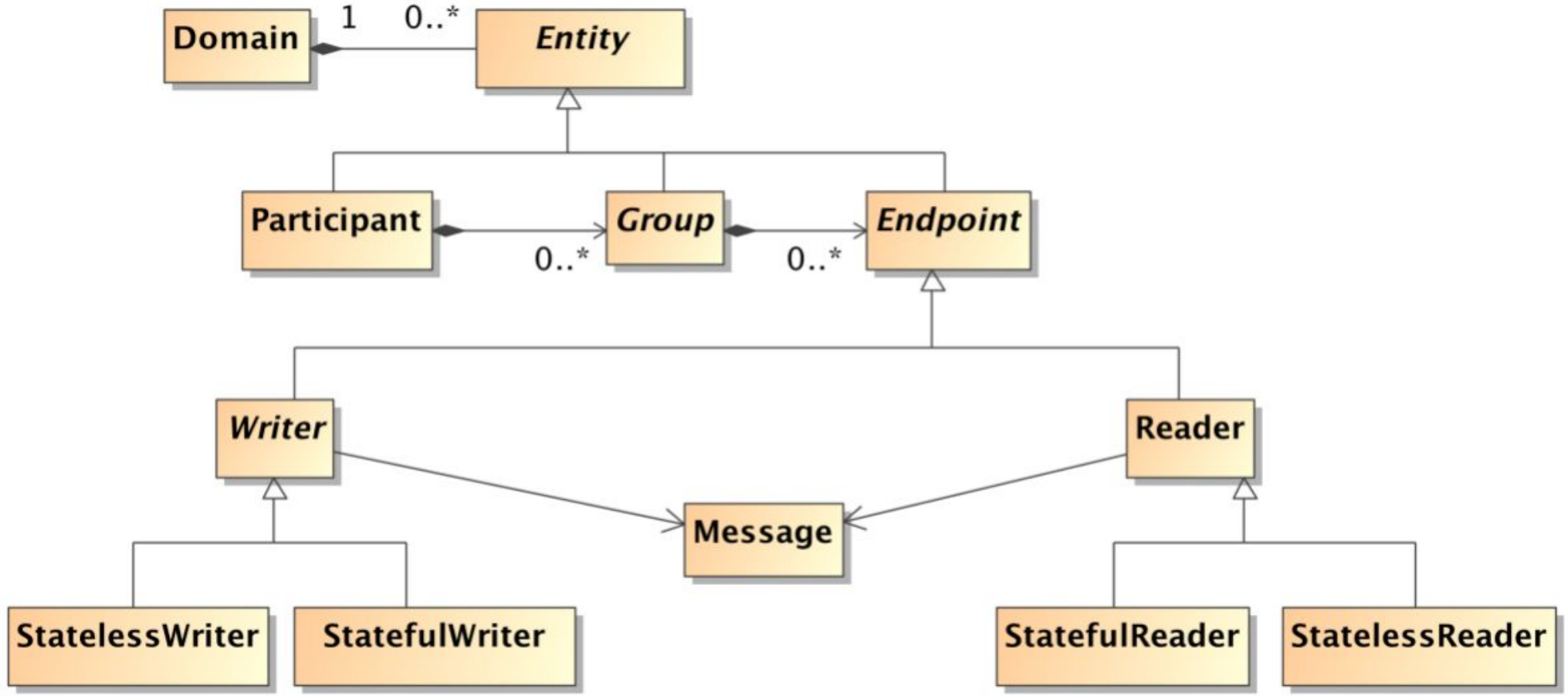
How does the robot state publisher publish the robot description?

Transient Local Profile	
History	*Keep last*
Depth	**1**
Reliability	*Reliable*
Durability	*Transient local*
Deadline	Implementation default
Lifespan	Implementation default
Liveliness	Implementation default
Lease Duration	Implementation default

Real-Time Publish-Subscribe

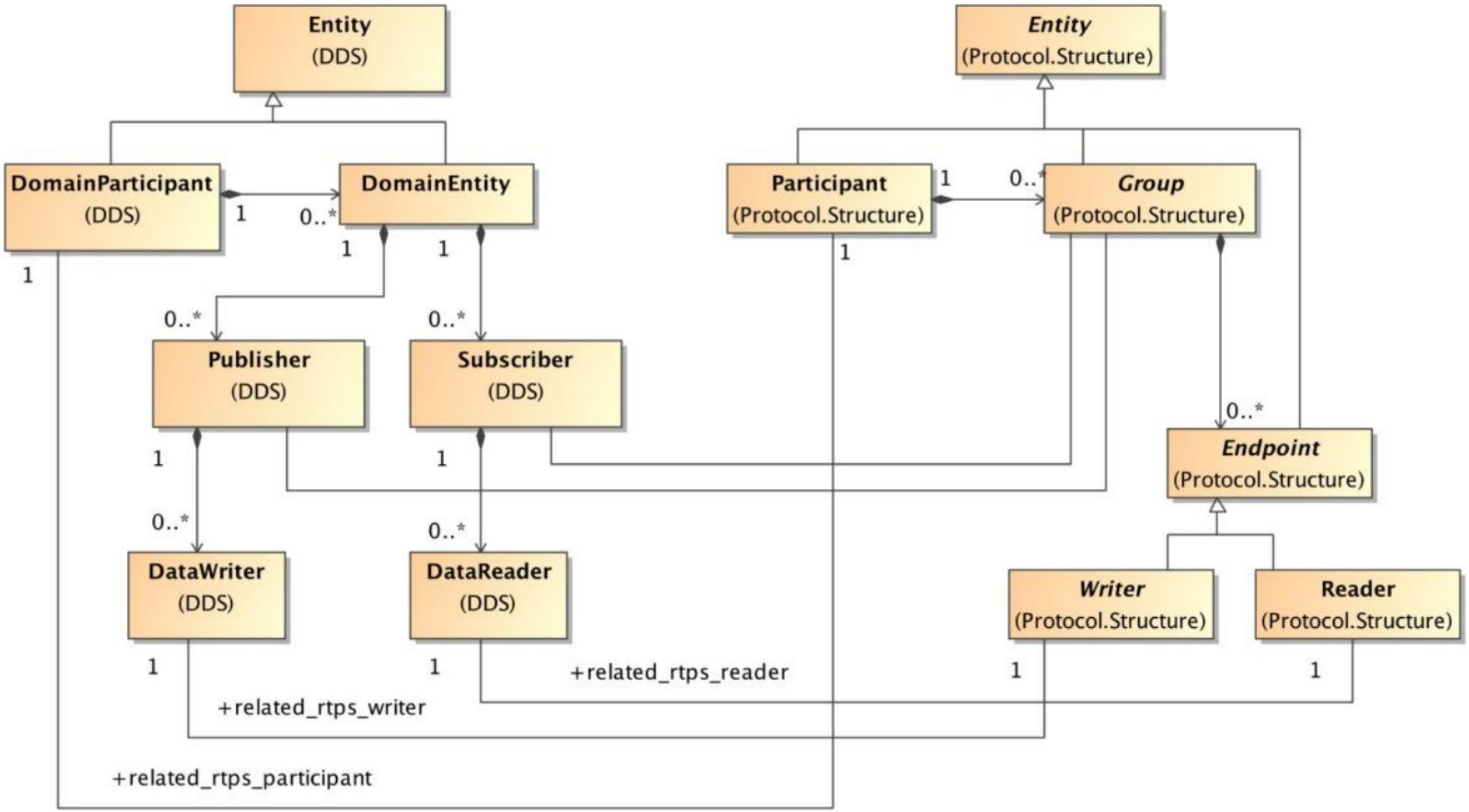
- RTPS protocol describes an implementation model of the DDS specification with the following features:
  - multicast and connectionless best-effort transports (UDP/IP).
  - creation of networks without single points of failure.
  - plug-and-play connectivity that allows applications to join and leave the network.
  - configurability to balance reliability and timeliness.
  - modularity for simple devices to participate while only needing a subset of the protocol.
  - scalability for very large networks.
  - type-safety.

RTPS Structure



RTPS Structure Module

RTPS and DDS



Correspondence between RTPS and DDS Entries

RTPS Transports

- RTPS protocol supports a variety of transports and QoS:
  - able to run on multicast and best-effort transports.
  - does not need to guarantee each packet will reach its destination or delivered in-order.
  - may deliver one message to multiple receivers.
- Transport general requirements are:
  - notion of unicast address (within 16 bytes).
  - notion of a port (within 4 bytes).
  - send a datagram (sequence of octets) to specific address/port.
  - drop messages if incomplete or corrupted.
  - means to deduce size of received messages.

RTPS Discovery

RTPS specification splits up discovery into two independent protocols:

- Participant Discovery Protocol
  - how participants discover each other.
- Endpoint Discovery Protocol
  - exchange information on the endpoints they contain.

RTPS requires that the following discovery protocols are implemented:

- Simple Participant Discovery Protocol (SPDP)
- Simple Endpoint Discovery Protocol (SEDP)

We will revisit these protocols in detail when discussing the middleware implementation.

DDS and ROS

- DDS provides an end-to-end middleware that can be abstracted and is therefore easy to maintain, swap, and allows ROS users to use the transport system while avoiding its complexity.
- DDS is a distributed discovery system; it is more fault tolerant and flexible than the ROS 1 **roscore** centralized communication node.
- DDS is used in mission critical systems, from battleships to financial systems, and is apt for robotic applications.
- In ROS 2, a few DDS implementations have been used as the data transport middleware.

RMW Implementations

Vendors and Implementations

- Since DDS is a specification, it has a wide range of implementations from different vendors that meet the transport requirements.
- eProsima's **rmw_fastrtps_cpp** is the default RMW implementation, and therefore, the one we will cover in subsequent sections.

Product	License	RMW Implementation	Tier	Status
eProsima *Fast DDS*	Apache 2	**rmw_fastrtps_cpp**	1	Default. Full support. Packaged with binary releases.
Eclipse *Cyclone DDS*	Eclipse Public License v2.0	**rmw_cyclonedds_cpp**	1	Full support. Packaged with binary releases.
RTI *Connext DDS*	Commercial, Research	**rmw_connextdds**	2	Full support. Support included in binaries, but *Connext* installed separately.
GurumNetworks *GurumDDS*	Commercial	**rmw_gurumdds_cpp**	3	Community support. Support included in binaries, but GurumDDS installed separately.

Vendors and Implementations

- The DDS specification does not specify a protocol to exchange messages; therefore, different implementations have no interoperability guarantees.
  - *Fast DDS* ***WString*** cannot be received by *Connext* on macOS.
  - *Connext* ***WString*** is not support by *Cyclone DDS*.

Setting ROS Middleware

- Assuming that the middleware is installed and that the executable was installed from binaries, then the RMW implementation can be selected at run time:

```
export RMW_IMPLEMENTATION=rmw_fastrtps_cpp
```

- If not installed from binaries, make sure to build the workspace after having installed the desired middleware.

Daemon

- As we will see later, discovering all nodes on the network is an active process.
- When first running any command line tools, like **ros2 topic list**, all nodes need to be discovered, and depending on the number of nodes, it could take several seconds and may require multiple calls to view all topics.
- The daemon is a process that runs in the background from the moment you run the first **ros2** CLI command that continuously discovers nodes on the network and stores them.
- Subsequent calls to **ros2** CLI commands are quickly executed as they reference the stored information by the daemon.

Daemon

- The daemon, if not stopped, is always running in the background, continuously discovering nodes and creating discovery traffic.
  - the more nodes on the network, the more data that will be transmitted to and from the daemon.
- Only one daemon exists, and is shared by all terminals and all command line tools.
  - the daemon is created with the environment of the terminal that ran the first **ros2** command.
  - the daemon must be restarted for changes in the environment to be propagated to it.

Ex. 2 - Env. Var. and Daemon

Open a terminal and try the following:

List all environment variables	`printenv`

The **printenv** command will print all environment variables set. To narrow down the output, we can use **grep** to display only ROS related variables.

List ROS environment variables	`printenv | grep ROS`

If the output is empty, then that indicates the ROS environment has not being sourced. Do so now:

Source ROS environment	`source /opt/ros/humble/setup.bash`

Ex. 2 - Env. Var. and Daemon

At this point, the environment is in the default state, and the daemon has not been started yet. Let's start the default demo talker, but with a unique namespace.

Namespaced demo talker node
`ros2 run demo_nodes_cpp talker --ros-args -r __ns:=/unique_namespace`

Open a new terminal. Make sure to have sourced the ROS environment. Then, run the **ros2 topic list** command.

ROS topic list command line tool	`ros2 topic list`

Once the command is executed, the daemon will start and will be running in the background, even if the terminal window is closed.

Ex. 2 - Env. Var. and Daemon

Ex. 2 - Env. Var. and Daemon

The output of the topic list command should result in at least the **/unique_namespace/chatter** from your talker. You may also see others' chatter topics.

If we wanted to avoid other devices' nodes, we can use the **ROS_LOCALHOST_ONLY** environment variable to limit communication to only nodes on the same device.

In the same terminal you ran **ros2 topic list** set the env. variable.

Local host only environment variable	`export ROS_LOCALHOST_ONLY=1`

Ex. 2 - Env. Var. and Daemon

Now, try listing the topics once again.

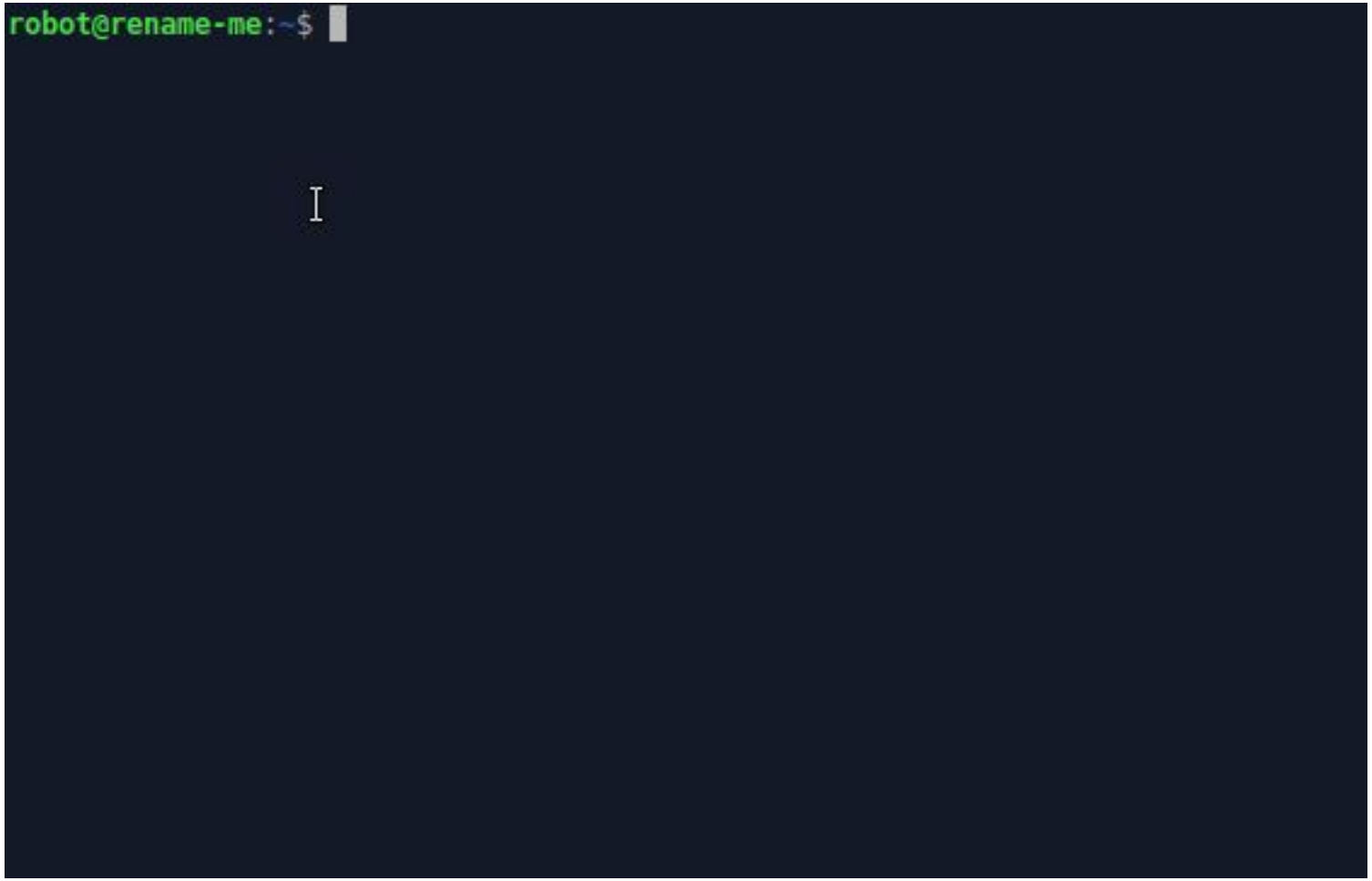
ROS topic list command line tool	`ros2 topic list`

You will notice that nothing has changed. The daemon has not been restarted with the new environment. Restart the daemon, and try again.

Stop daemon	`ros2 daemon stop`
Re-run topic list command line tool	`ros2 topic list`

Notice that there are no topics showing up now. The local host variable modifies the data transport layer, and now the daemon is unable to communicate with the node that was launched without the local host variable.

Ex. 2 - Env. Var. and Daemon



Ex. 2 - Env. Var. and Daemon

For the daemon to reliably communicate with other nodes, they need to have compatible environments. To demonstrate this, stop the talker node, set the localhost variable, and restart the talker.

Local host only environment variable	`export ROS_LOCALHOST_ONLY=1`
Restart talker node	`ros2 run demo_nodes_cpp talker --ros-args -r __ns:=/unique_namespace`

Now, try to once again see the topics in the second terminal.

ROS topic list command line tool	`ros2 topic list`

Ex. 2 - Env. Var. and Daemon

Description	CLI command
View all env. variables	`printenv`
View all env. variables with specific keyword	`printenv | grep ROS`
Source ROS	`source /opt/ros/humble/setup.bash`
Talker	`ros2 run demo_nodes_cpp talker --ros-args -r __ns:=/unique_namespace`
ROS topic command line tool	`ros2 topic list`
Local host only variable	`export ROS_LOCALHOST_ONLY=1`
Stop daemon	`ros2 daemon stop`

Fast-DDS:
Simple Discovery

What is Simple Discovery?

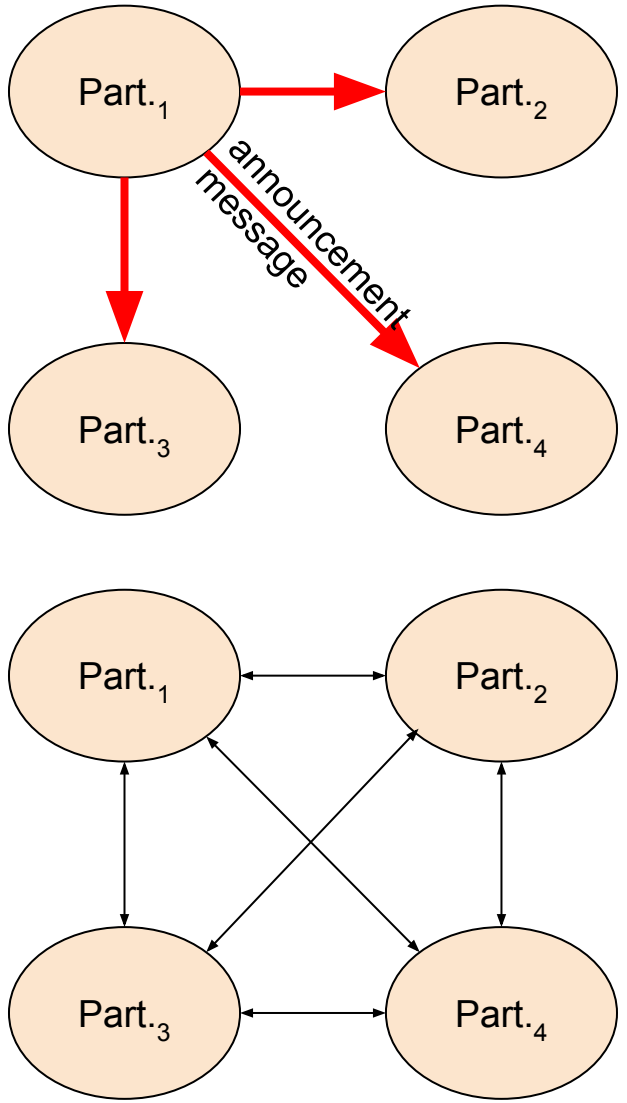
- Follows the ***Real-time Publish-Subscribe*** (RTPS) standard for ***Simple Participant Discovery Phase*** (SPDP) and ***Simple Endpoint Discovery Phase*** (SEDP).
- All publishers and subscribers acknowledge each other's existence and establish communication channels between their readers and writers.
- Best suited for small to medium sized networks.
  - Every participant will try to discover every other participant.
  - In large networks, a significant amount of traffic will be generated just to discover all nodes.

RTPS Simple Participant Discovery

- Each participant creates a built-in writer and reader endpoints.
  - ***SPDPbuiltinParticipantWriter*** and ***SPDPbuiltinParticipantReader***
- The writer sends out a single data-object containing the participant's information to a pre-configured list of locators to announce the participants existence.
  - the list and participant information is defined by the specific implementation.
  - the list may contain both unicast and multicast locators.
  - the list contains *possible* participants, but no participants are required.
- The reader receives other participants' information and now a network of endpoints can be established.

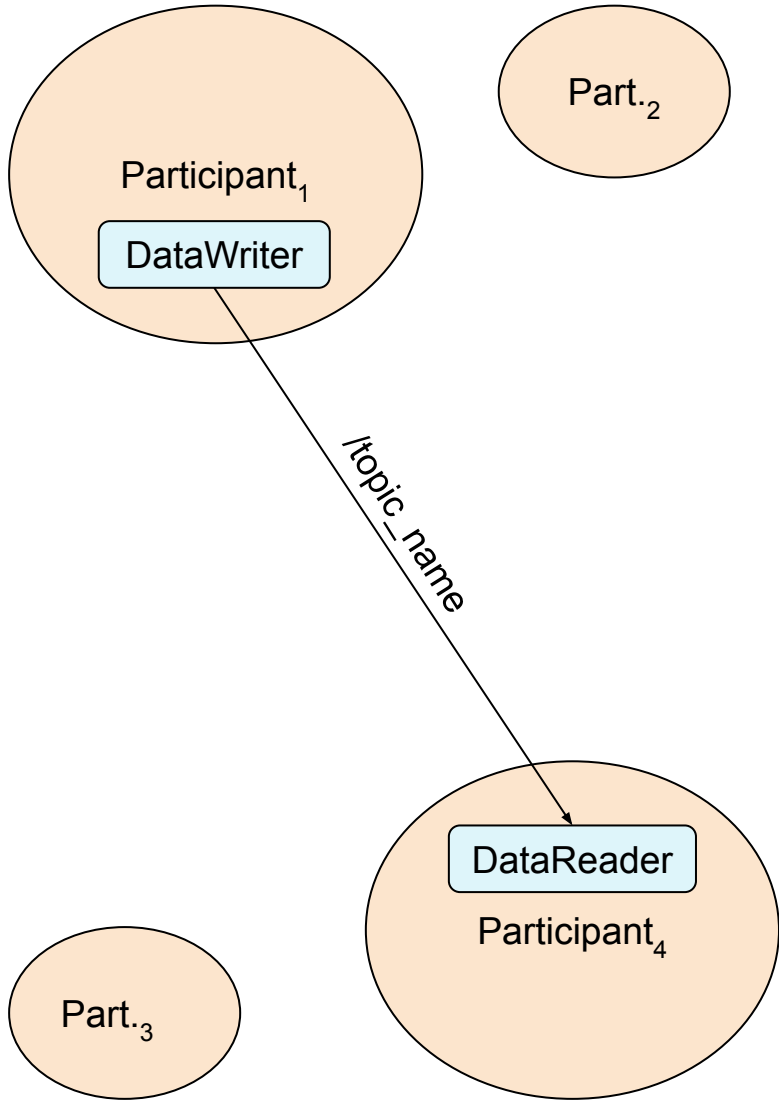
RTPS Simple Participant Discovery

- Each participant sends periodic multicast announcement messages containing the multicast and unicast addresses (IP, port) where it is listening:
  - Multicast: **PB** + **DG** * ***domainId*** + **d0**
  - Unicast: **PB** + **DG** * ***domainId*** + **d1** + **PG** * ***participantID***
  - By default, **PB** = 7400; **DG** = 250; **PG** = 2; **d0** = 0; **d1** = 10; **d2** = 1; **d3** = 11.
  - Due to UDP port limits, this enables the use of 230 domains with up to 120 participants per domain.



RTPS Simple Endpoint Discovery

- **DataWriters** and **DataReaders** acknowledge each other using information from their respective participants.
- Using the established communication channels in the previous step, endpoints are matched if they share the same data **Topic** and **Type**.
- Once matched, the endpoints are ready for sending and receiving user data traffic.



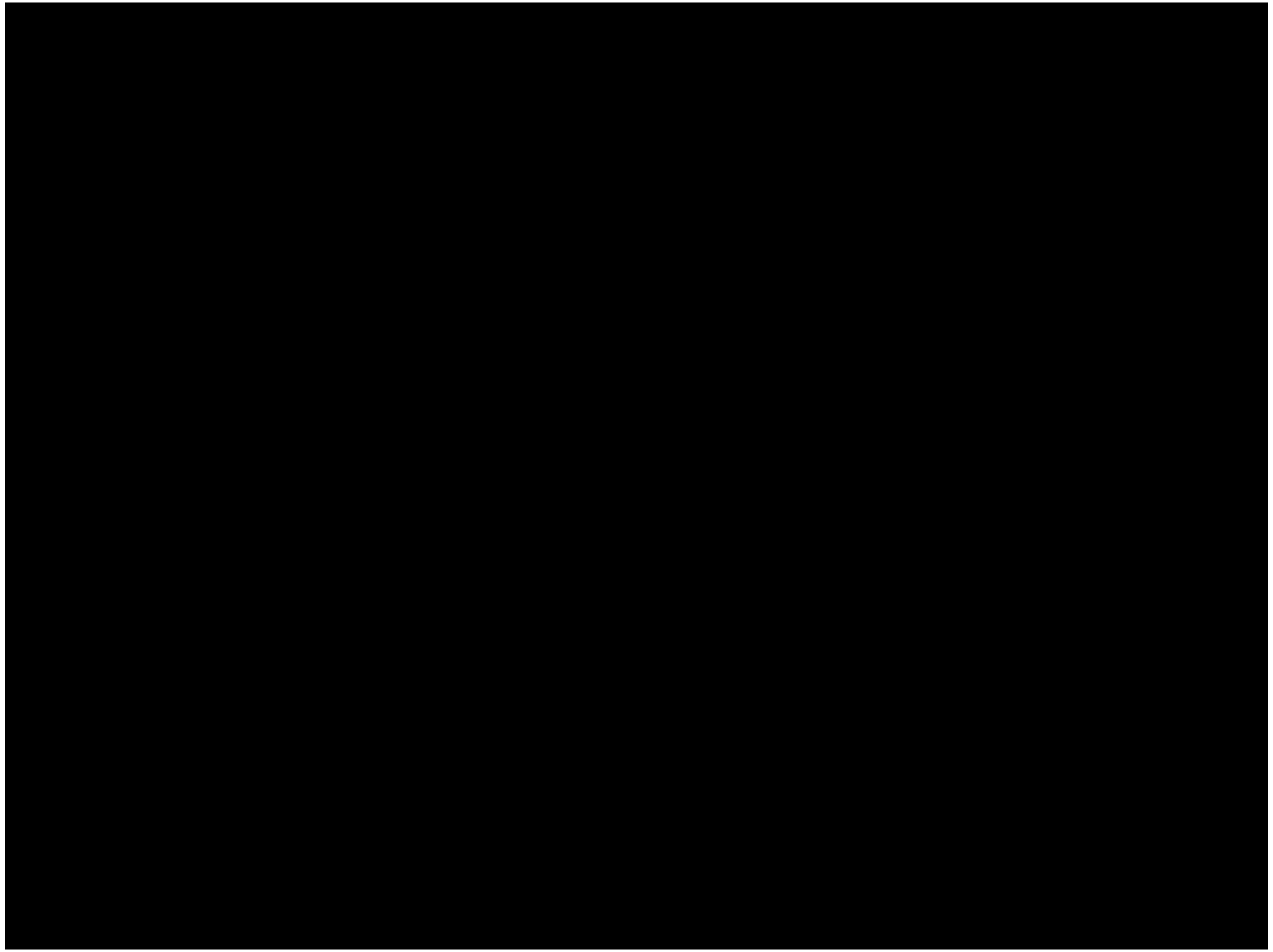
Simple Discovery Env. Variables

- In simple discovery, the only environment variable that directly influences the discovery process is **ROS_DOMAIN_ID**.
- In the previous slides, we discussed that the multicast announcement messages are directed to ports depending on the domain ID.
- Nodes on different domains are unable to establish communication with each other.
- Therefore, the domain ID is the simplest tool that can alter the network of nodes.
- In general, robotic systems that are not meant to interact should be on separate domains.

Simple Discovery Env. Variables

- Although UDP has port ranges that support up to **232** domains, not all port ranges are available.
- The Linux kernel uses ports **32768-60999** for ephemeral ports. Therefore, domain IDs **0-101** and **215-232** can be safely used, by default.
- It is also important to note that each process on a device will be assigned a process ID. If there are more than 120 processes on a single device, the unicast port will “spill over” to the next domain.
- “Spill over” is not a concern unless the next domain ID is also being used.

Discovery Traffic



Ex. 3.1 Turtlesim on Domains

Start the demo *Turtlesim*.

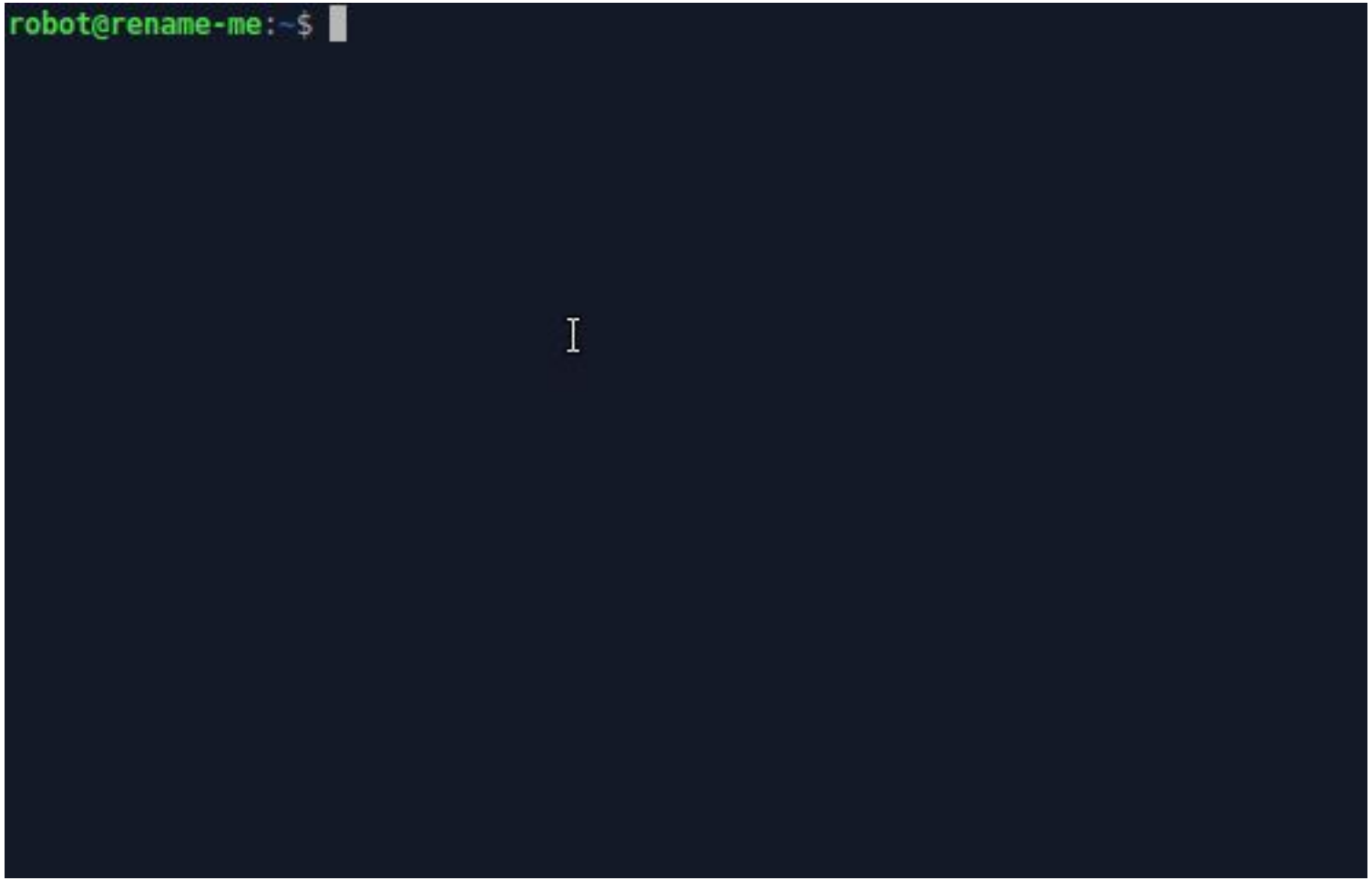
Namespaced turtlesim
`ros2 run turtlesim turtlesim_node --ros-args -r __ns:=/unique_namespace`

In another terminal, restart the daemon and try to list all of the nodes on the network.

Stop daemon	`ros2 daemon stop`
Node list command line tool	`ros2 node list`

You should be able to see others' *Turtlesim* nodes.

Ex. 3.1 Turtlesim on Domains



Ex. 3.1 Turtlesim on Domains

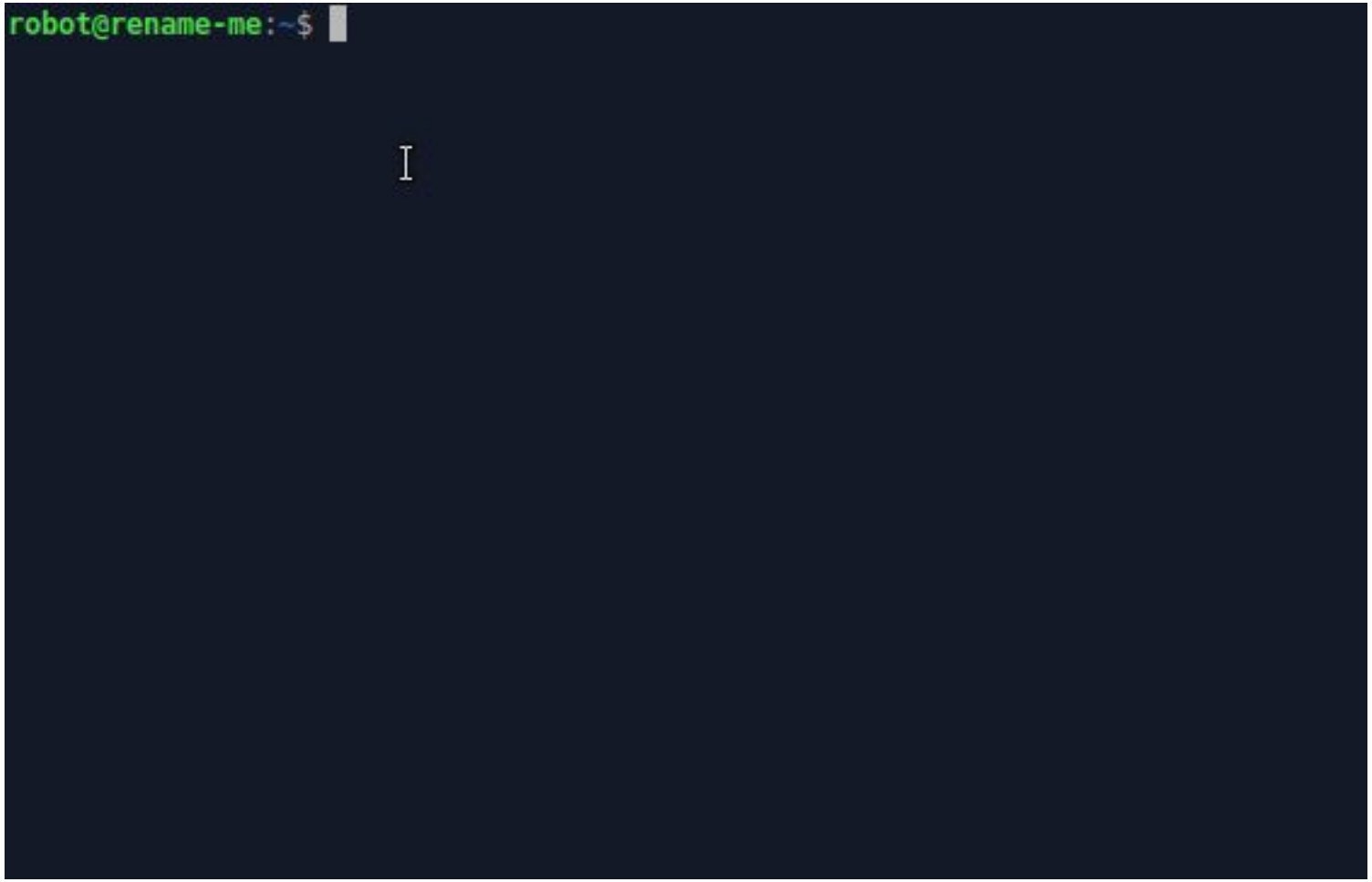
Get into pairs. Each pair will be its own domain. Set your domain ID, restart the daemon, and make sure that no nodes are on that domain.

Set domain ID	`export ROS_DOMAIN_ID=<DOMAINID>`
Stop daemon	`ros2 daemon stop`
Node list command line tool	`ros2 node list`

Now, we are ready to begin. Stop the *turtlesim* node, set the domain ID on that terminal, and start it back up.

`export ROS_DOMAIN_ID=<DOMAINID>`
Namespaced turtlesim
`ros2 run turtlesim turtlesim_node --ros-args -r __ns:=/unique_namespace`

Ex. 3.1 Turtlesim on Domains



Ex. 3.1 Turtlesim on Domains

Make sure that you can now see your partner's and your own node.

ROS node list command line tool	`ros2 node list`

We can also drive each other's nodes.

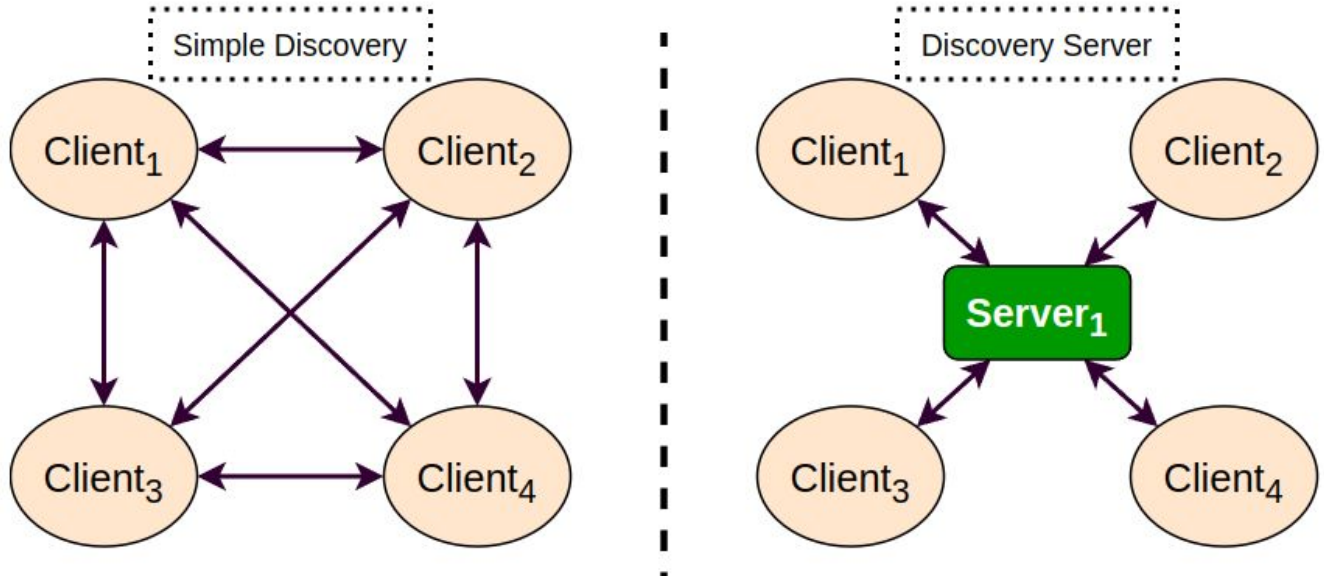
Namespaced Teleop
`ros2 run turtlesim turtle_teleop_key --ros-args -r __ns:=/unique_namespace`

Switch the namespace to drive the robot in your partner's simulator.

Fast-DDS:
Discovery Server

What is Discovery Server?

- Alternative discovery method using a Client-Server architecture
- Intermediate server(s) manage all discovery information and distribute it appropriately to each client
- Multiple discovery servers can run on the same network
- Clients only receive discovery information that they require



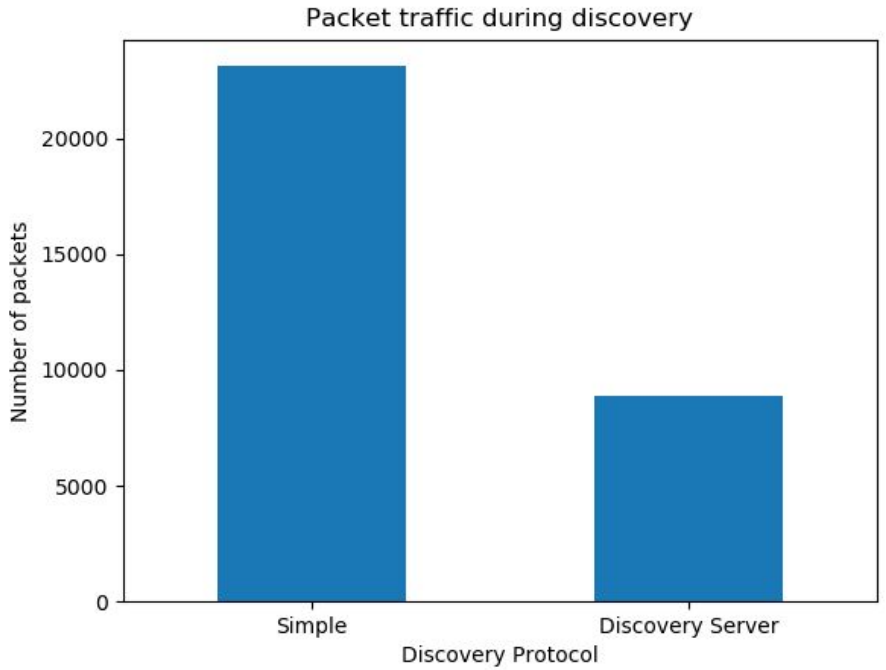
Pros and Cons of Discovery Server

Pros:

- Reduces overall discovery traffic
- Scales well with the number of nodes on the network
- Allows for server redundancy
- Does not use multicasting

Cons:

- Requires additional setup of servers
- Each client must be manually configured to use the discovery servers
- Network topology can become complex



Running a Discovery Server

Each Discovery Server must specify:

- IP Address (Default: 0.0.0.0)
- Port (Default: 11811)
- Server ID

Discovery Server Type	CLI command	IP Address	Port	Server ID
Default	`fastdds discovery -i 0`	0.0.0.0	11811	0
Unique server ID, IP Address, and Port	`fastdds discovery -i 1 -l 127.0.0.1 -p 11876`	127.0.0.1	11876	1
Multiple local interfaces	`fastdds discovery -i 0 -l 192.168.0.10 -p 11876 -l 10.27.0.5 -p 48390`	192.168.0.10, 10.27.0.5	11876, 48390	0
Default with backup file	`fastdds discovery -i 0 -b`	0.0.0.0	11811	0
Redundant servers	`fastdds discovery -i 0 -l 127.0.0.1 -p 11811` `fastdds discovery -i 1 -l 127.0.0.1 -p 11812`	127.0.0.1 127.0.0.1	11811 11812	0 1

Configuring a Client

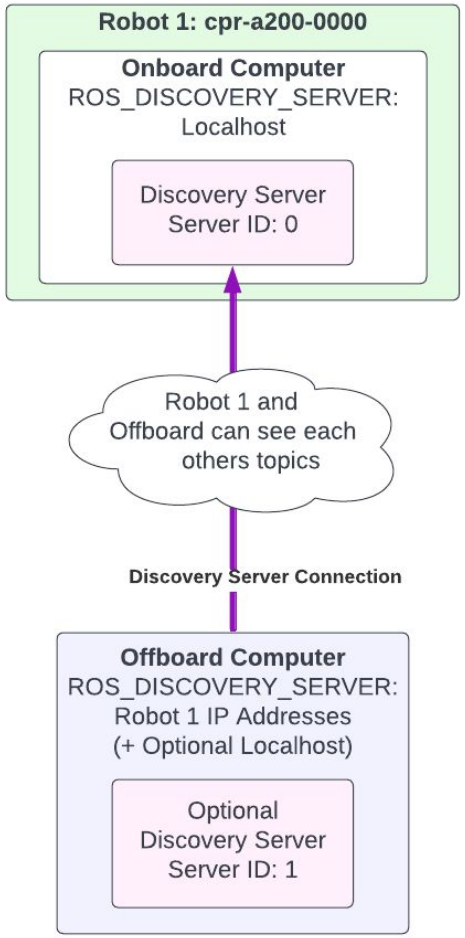
- Each discovery client must configure the servers it wants to use with the **ROS_DISCOVERY_SERVER** environment variable
- Servers are specified by their IP address and port, and delimited with semicolons
- The position of the server in the string indicates the server ID

Discovery Server Type	IP Address	Port	Server ID	ROS_DISCOVERY_SERVER*
Default	0.0.0.0	11811	0	`export ROS_DISCOVERY_SERVER="127.0.0.1:11811"`
Unique server ID, IP Address, and Port	127.0.0.1	11876	1	`export ROS_DISCOVERY_SERVER=";127.0.0.1:11876"`
Multiple local interfaces	192.168.0.10, 10.27.0.5	11876, 48390	0	`export ROS_DISCOVERY_SERVER="192.168.0.10:11876" OR` `export ROS_DISCOVERY_SERVER="10.27.0.5:48390"`
Default with backup file	0.0.0.0	11811	0	`export ROS_DISCOVERY_SERVER="127.0.0.1:11811"`
Redundant servers	127.0.0.1 127.0.0.1	11811 11812	0 1	`export ROS_DISCOVERY_SERVER="127.0.0.1:11811;127.0.0.1:11812"`

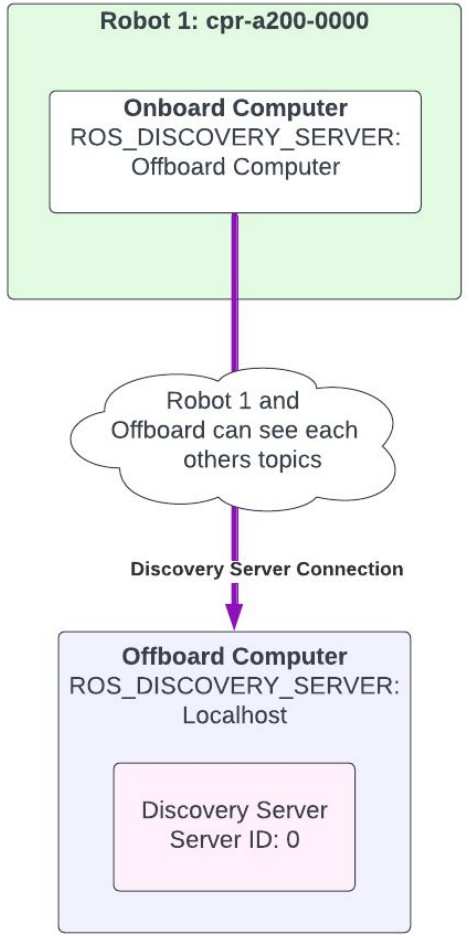
*Note: This environment variable must be exported on every terminal that will run a ROS 2 node

Discovery Server networks

Single Robot

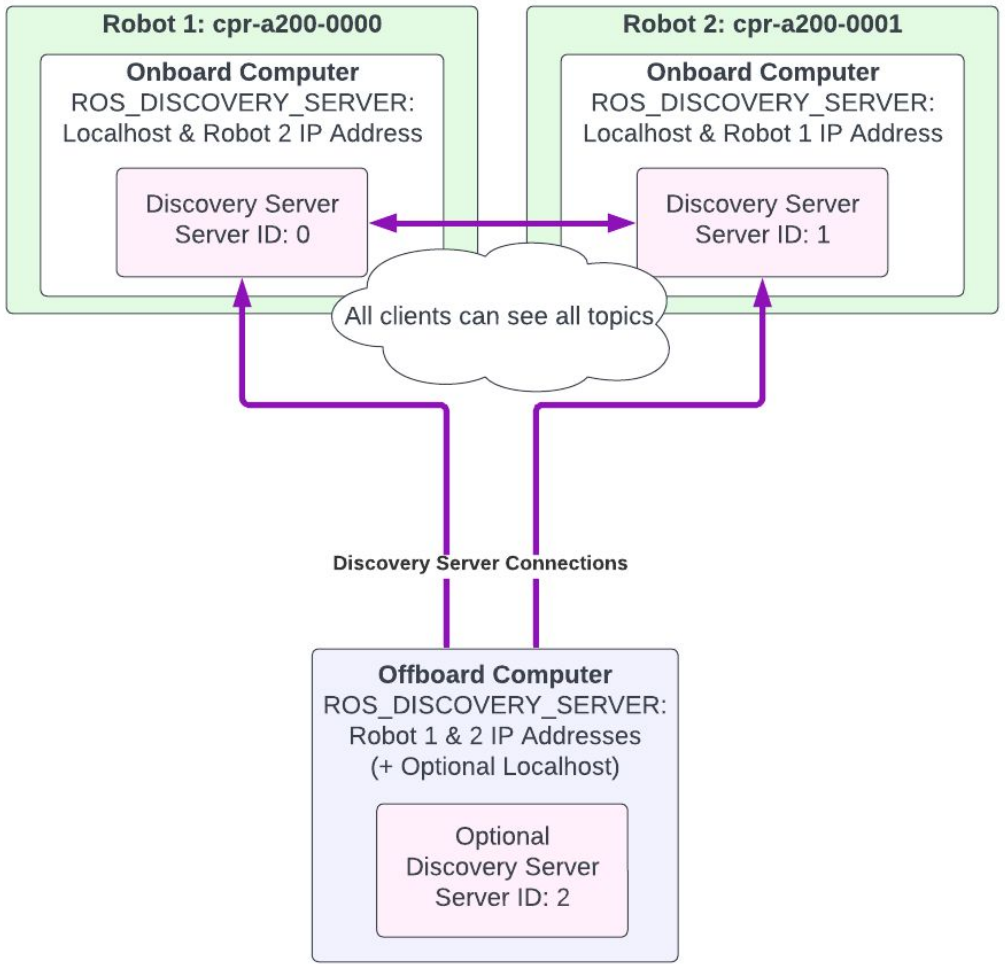


Single Robot Remote Server

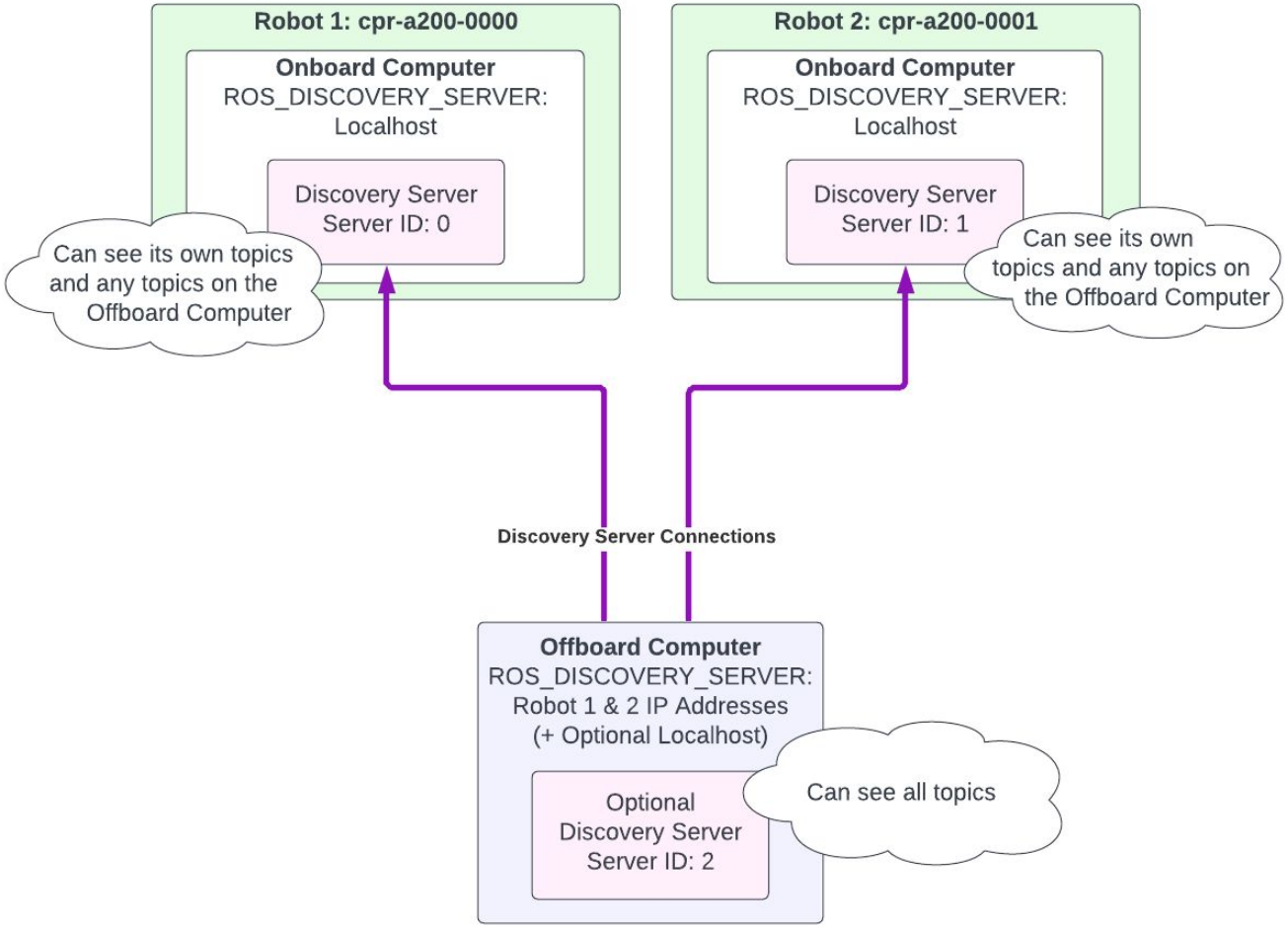


Discovery Server networks

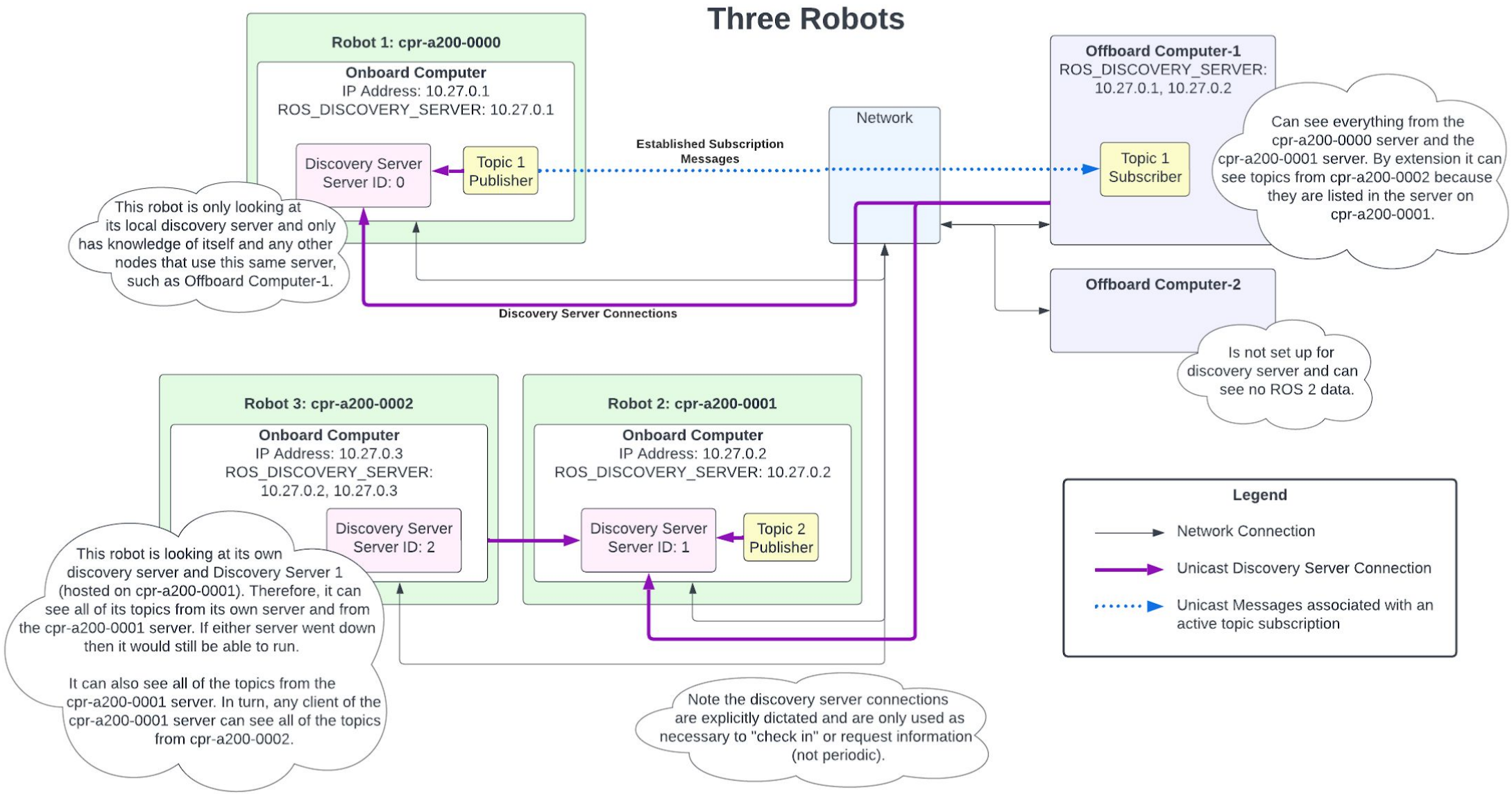
Two Robots - Fully Connected



Two Robots - Command Center

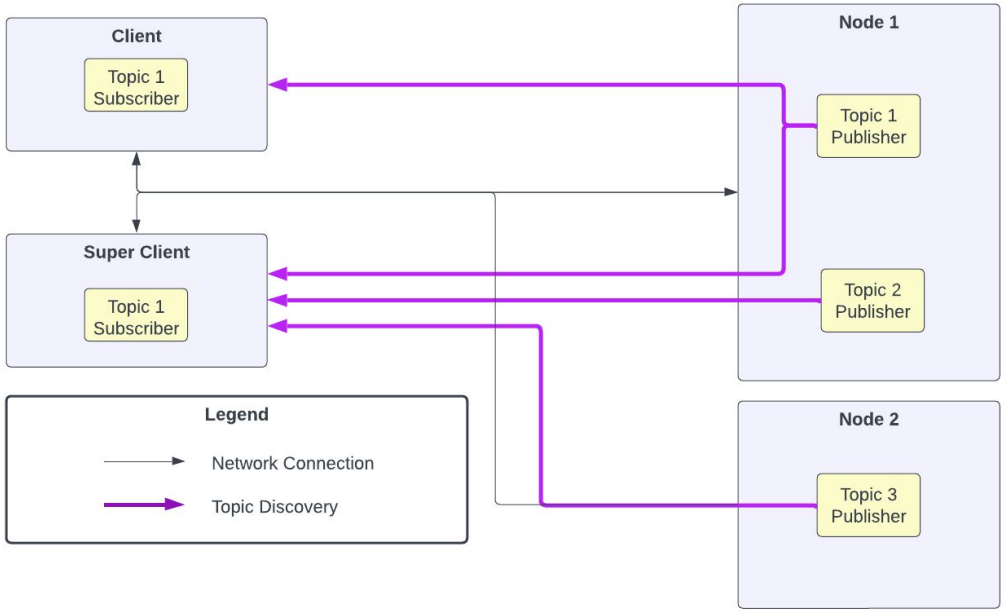


Discovery Server networks



Super Clients

- A default client will only discover nodes and topics that it publishes and subscribes to
- In some cases, it may be useful to discover all nodes and topics without subscribing to them (ros2cli)
- Super clients discover all nodes, topics, actions, and services on the same discovery server




```
export ROS_SUPER_CLIENT=1
```

XML profiles

- Discovery server settings can be configured in a FastDDS XML profile
- Multiple servers can be defined under `discoveryServersList`
- For complex robot networks it may be easier to maintain an XML profile than to manage environment variables
- Multiple participants can reuse the same profile (e.g. Remote laptops)

```
<discovery_config>
  <discoveryProtocol>SUPER_CLIENT</discoveryProtocol>
  <discoveryServersList>
    <RemoteServer prefix="44.53.00.5f.45.50.52.4f.53.49.4d.41">
      <metatrafficUnicastLocatorList>
        <locator>
          <udpv4>
            <address>127.0.0.1</address>
            <port>11811</port>
          </udpv4>
        </locator>
      </metatrafficUnicastLocatorList>
    </RemoteServer>
  </discoveryServersList>
</discovery_config>
```

```
export FASTRTPS_DEFAULT_PROFILES_FILE=my_discovery_server_profile.xml
```

When to use Discovery Server

- Robot is on a network that blocks multicasting (University, Corporate, etc.)
- If your network is being congested by ROS 2 discovery traffic or has limited bandwidth
- Limited processing power on the robot
- Many network participants, or excess nodes and topics
- The robot is being used by more experienced users

Ex. 4.1 - Setup a Server

Let’s begin by opening up a new terminal (Terminal 1), and starting up a server on the local host.

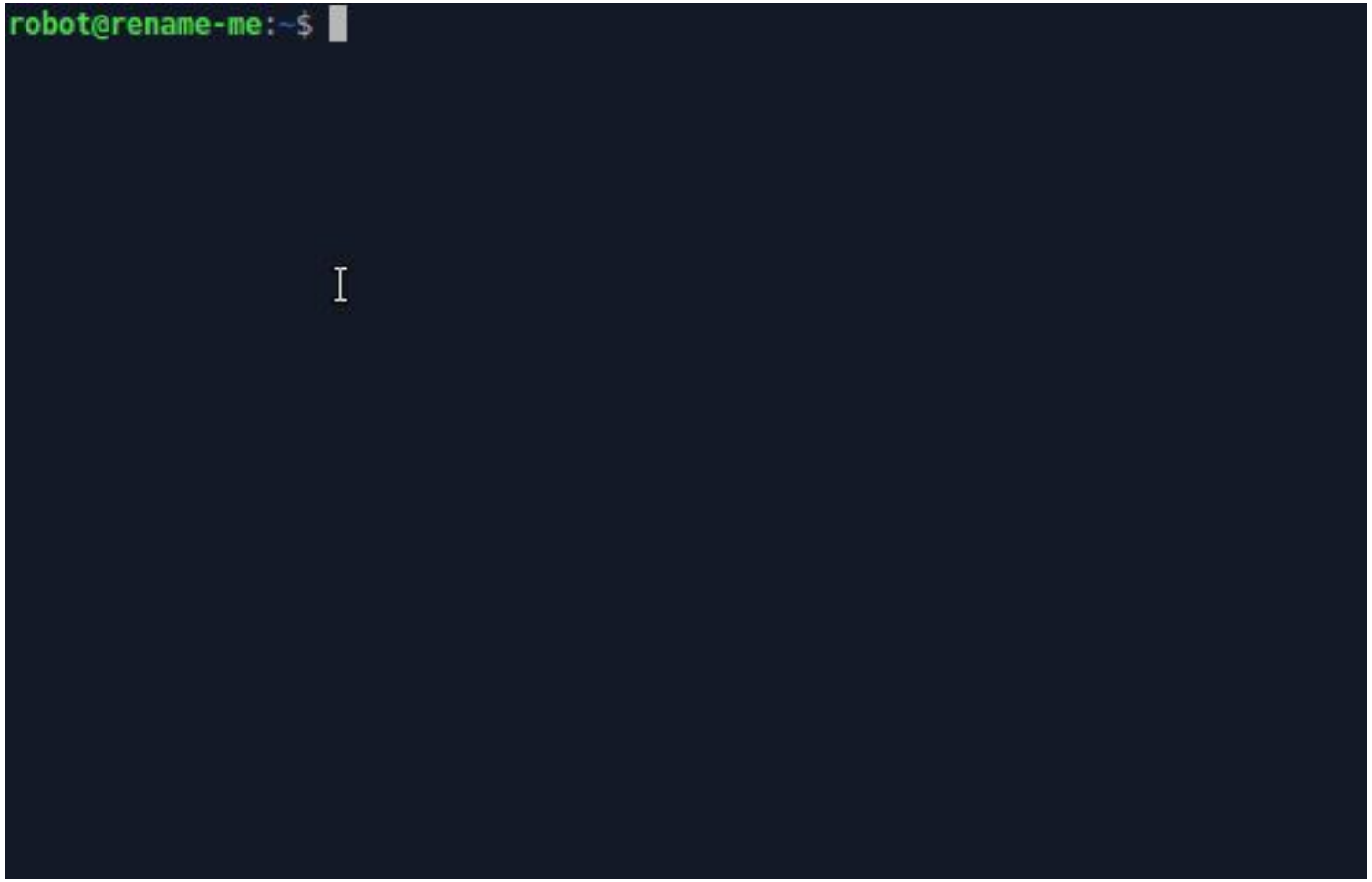
Local host server

```
fastdds discovery -i 0 -l 127.0.0.1 -p 11811
```

The flags correspond to the following:

- `-i 0`: assign ID 0 to the server.
- `-l 127.0.0.1`: tell the server to listen on the localhost.
- `-p 11811`: assign the server to use this port.

Ex. 4.1 - Setup a Server



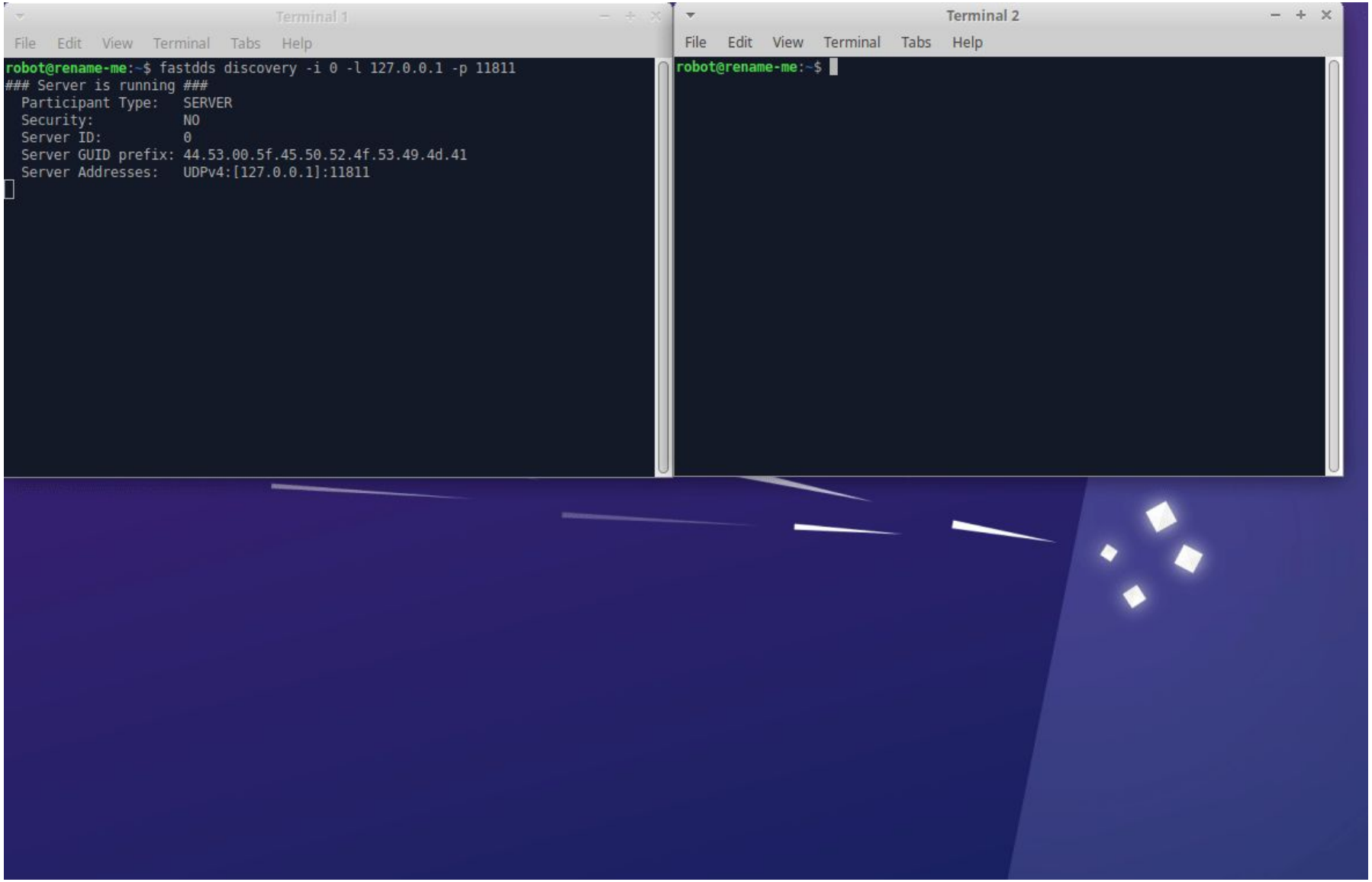
Ex. 4.1 - Setup a Server

In another terminal (Terminal 2), we will set the connection to the server and drive the *Turtlesim* once again.

Setup the server
`export ROS_DISCOVERY_SERVER="127.0.0.1:11811"`
Start the turtlesim
`ros2 run turtlesim turtlesim_node --ros-args -r __ns:=/unique_namespace`

The *Turtlesim* will only be discoverable through the discovery server on the localhost.

Ex. 4.1 - Setup a Server

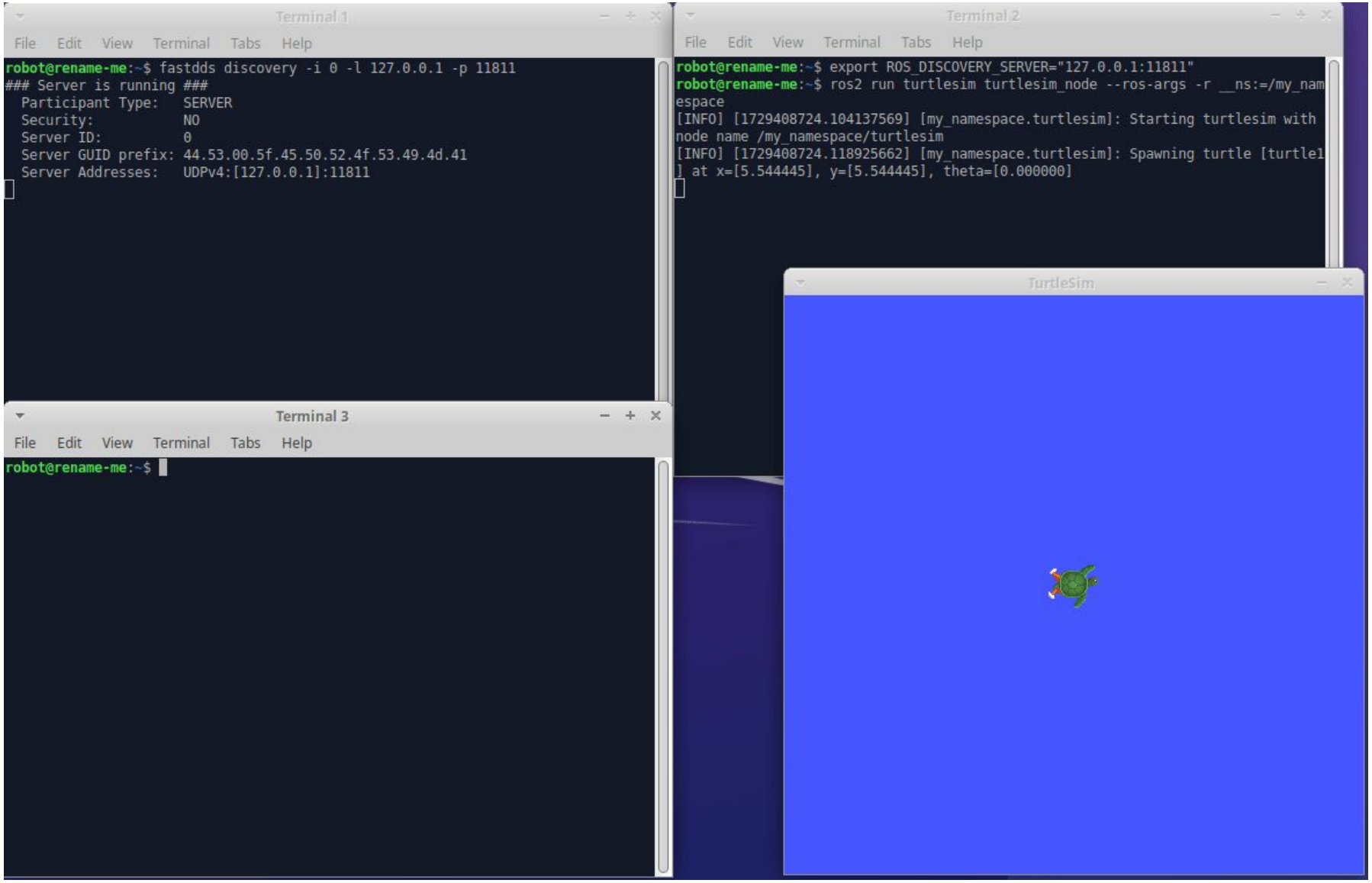


Ex. 4.1 - Setup a Server

In another terminal (Terminal 3), we will set the discovery server variable, check that we can discover the node, and then move the turtle.

Setup the server
`export ROS_DISCOVERY_SERVER="127.0.0.1:11811"`
Set the terminal as a super client
`export ROS_SUPER_CLIENT=1`
Restart the ros2cli daemon
`ros2 daemon stop; ros2 daemon start`
Check that the node is visible
`ros2 node list`
Start the teleoperation
`ros2 run turtlesim turtle_teleop_key --ros-args -r __ns:=/unique_namespace`

Ex. 4.1 - Setup a Server



Ex. 4.2 - Multiple Servers

Now, let's try to connect to another server. For this, we will add a second server for somebody else to connect to. Open a new terminal (Terminal 4), and start a server with ID 1.

Wi-Fi server

```
fastdds discovery -i 1 -l <YOUR_WIFI_IP> -p 11812
```

Find a partner. Exchange IP addresses. They will connect to your second server and you will connect to theirs.

Ex. 4.2 - Multiple Servers

Rockwell Automation | CLEARPATH ROBOTICS

Terminal 1

File Edit View Terminal Tabs Help

```
robot@rename-me:~$ fastdds discovery -i 0 -l 127.0.0.1 -p 11811
### Server is running ###
  Participant Type:   SERVER
  Security:           NO
  Server ID:          0
  Server GUID prefix: 44.53.00.5f.45.50.52.4f.53.49.4d.41
  Server Addresses:   UDPv4:[127.0.0.1]:11811
```

Terminal 2

File Edit View Terminal Tabs Help

```
robot@rename-me:~$ export ROS_DISCOVERY_SERVER="127.0.0.1:11811"
robot@rename-me:~$ ros2 run turtlesim turtlesim_node --ros-args -r __ns:=/my_nam
espace
[INFO] [1729408724.104137569] [my_namespace.turtlesim]: Starting turtlesim with
node name /my_namespace/turtlesim
[INFO] [1729408724.118925662] [my_namespace.turtlesim]: Spawning turtle [turtle1
] at x=[5.544445], y=[5.544445], theta=[0.000000]
```

Terminal 3

File Edit View Terminal Tabs Help

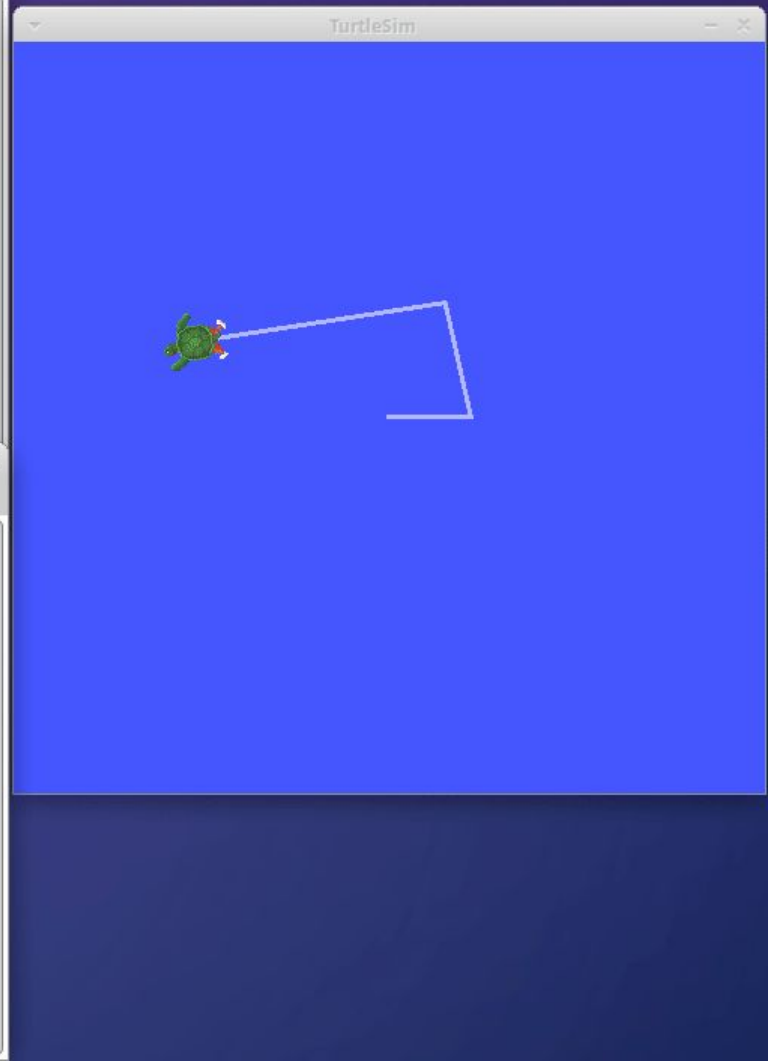
```
robot@rename-me:~$ export ROS_DISCOVERY_SERVER="127.0.0.1:11811"
robot@rename-me:~$ export ROS_SUPER_CLIENT=1
robot@rename-me:~$ ros2 daemon stop; ros2 daemon start
The daemon has been stopped
The daemon has been started
robot@rename-me:~$ ros2 node list
/my_namespace/turtlesim
robot@rename-me:~$ ros2 run turtlesim turtle_teleop_key --ros-args -r __ns:=/my_
namespace
Reading from keyboard
---------------------------
Use arrow keys to move the turtle.
Use G|B|V|C|D|E|R|T keys to rotate to absolute orientations. 'F' to cancel a rot
ation.
'Q' to quit.
```

Terminal 4

File Edit View Terminal Tabs Help

```
robot@rename-me:~$
```

TurtleSim



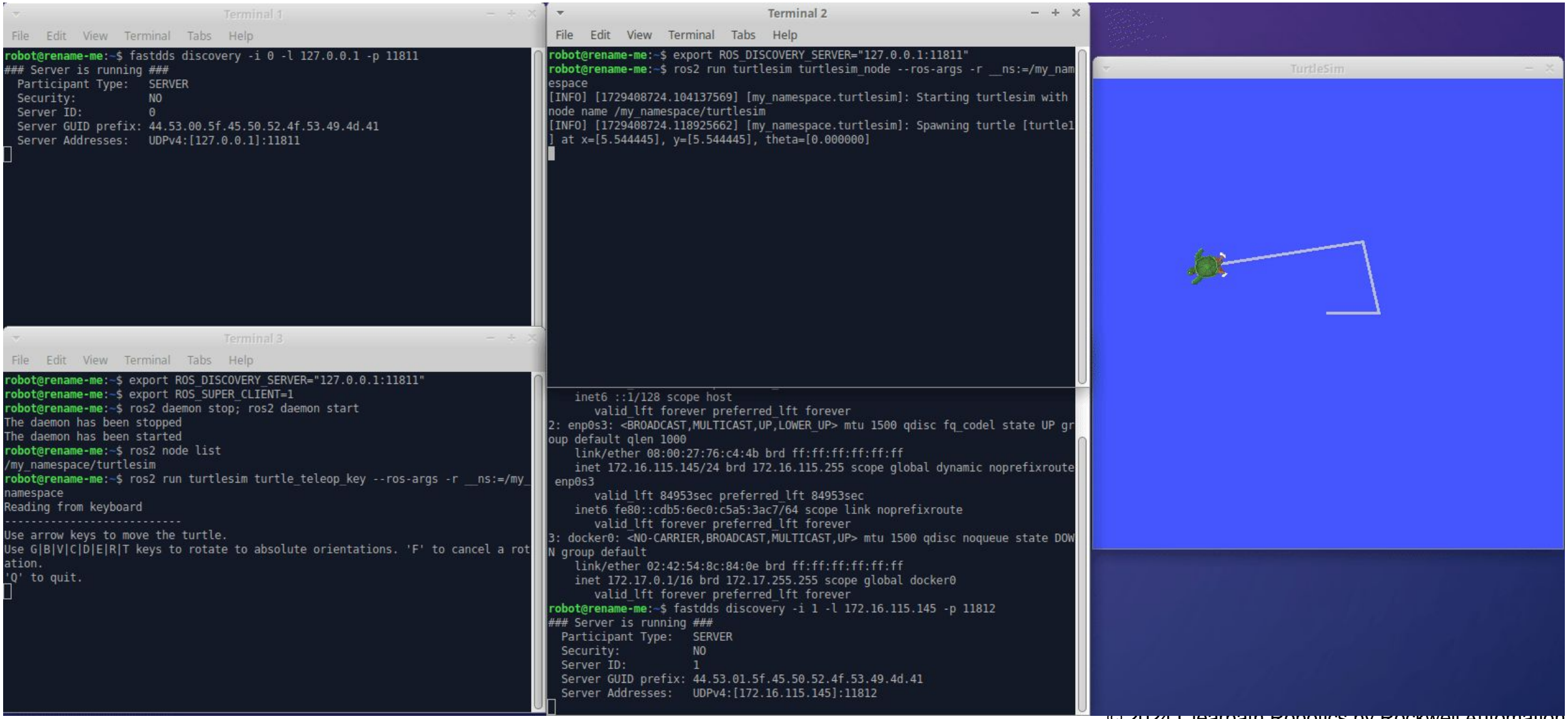
Ex. 4.2 - Multiple Servers

Back in the Terminal 2, we need to reset the discovery server variable to add both servers.

Setup the servers
`export ROS_DISCOVERY_SERVER="127.0.0.1:11811;<YOUR_WIFI_IP>:11812;"`
Restart the turtlesim
`ros2 run turtlesim turtlesim_node --ros-args -r __ns:=/unique_namespace`

Ex. 4.2 - Multiple Servers

Rockwell Automation | CLEARPATH ROBOTICS

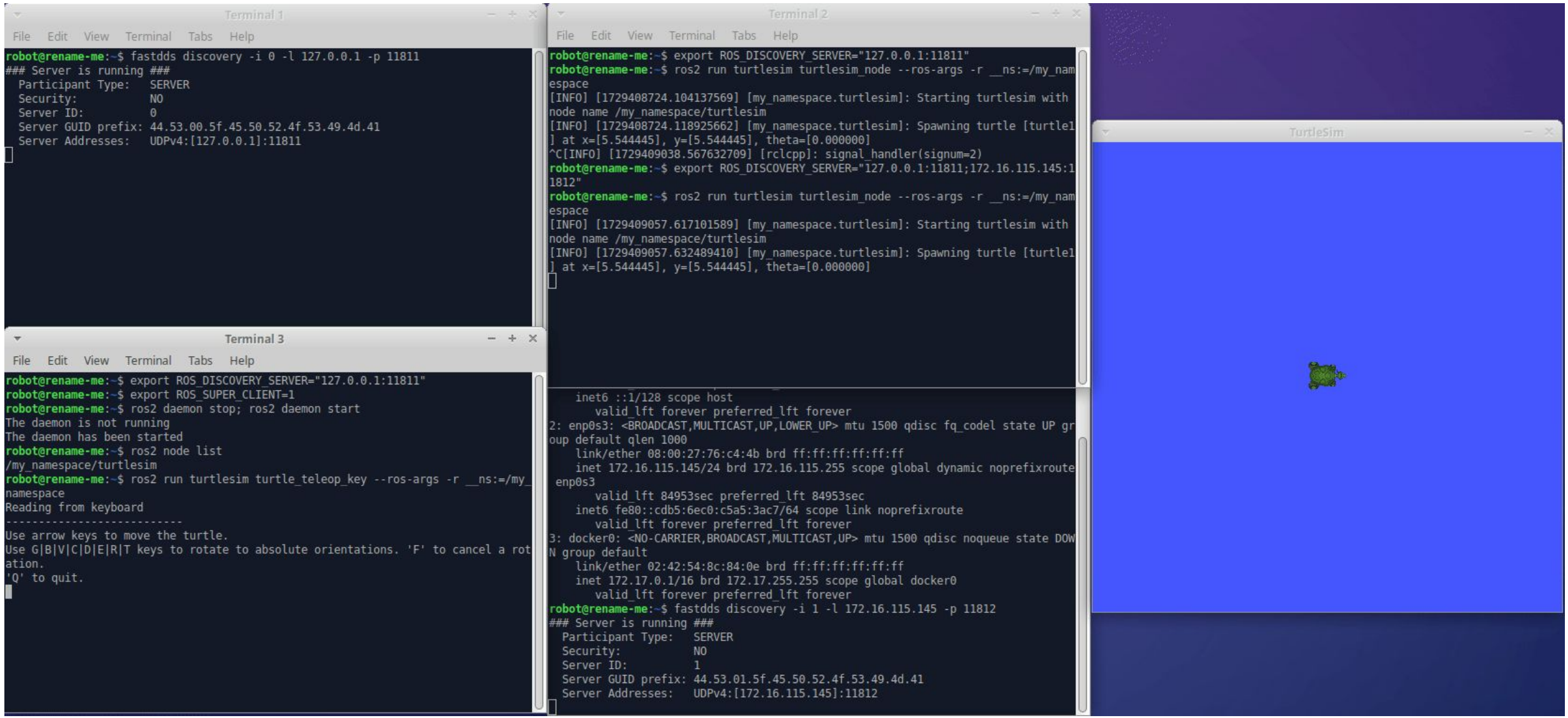


Ex. 4.2 - Multiple Servers

Back in the Terminal 3, we will set our first server and our partner's second server. We should be able to command our partner's turtle.

Setup the server
`export ROS_DISCOVERY_SERVER="127.0.0.1:11811;<PARTNER_IP>:11812;"`
Set the terminal as a super client
`export ROS_SUPER_CLIENT=1`
Restart the ros2cli daemon
`ros2 daemon stop; ros2 daemon start`
Check that both `turtlesim` nodes are discoverable
`ros2 node list`
Start the teleoperation
`ros2 run turtlesim turtle_teleop_key --ros-args -r __ns:=/<partners_namespace>`

Ex. 4.2 - Multiple Servers



Ex. 4.2 - Multiple Servers

Now, return to Terminal 1 and stop the first server (the one with `-i 0`). Then, go to the Terminal 3, and stop teleoperation.

You should still be able to drive your partner's robot, but you should be unable to drive yours. Check if you can still see your *turtlesim* on the node list (using the Terminal 3):

Check the node list, is your node discoverable? Is your partners?
`ros2 node list`
Can you drive your robot or your partners?
`ros2 run turtlesim turtle_teleop_key --ros-args -r __ns:=/<namespace>`

Ex. 4.3 Stop the local server

Between you and your partner, pick the person that will continue running the server; the other person will stop theirs. The server should still be running in Terminal 4 (the one with -i 1).

To the person that stopped the server: return to Terminal 2, set only your partners discovery server, and restart the *turtlesim*.

Setup only partner's server (the semicolons are important).
`export ROS_DISCOVERY_SERVER=";<PARTNER_IP>:11812;"`
Restart the turtlesim
`ros2 run turtlesim turtlesim_node --ros-args -r __ns:=/unique_namespace`

Ex. 4.4: One server is enough

To the person that didn’t stop their server: return to Terminal 3, set your own server, and try to drive your own robot:

Setup the server.
`export ROS_DISCOVERY_SERVER=“;<YOUR_WIFI_IP>:11812;”`
Restart the ros2cli daemon
`ros2 daemon stop; ros2 daemon start`
Check that the node is visible
`ros2 node list`
Start the teleoperation
`ros2 run turtlesim turtle_teleop_key --ros-args -r __ns:=/<your_namespace>`

Ex. 4.4: One server is enough

Both of you should be able to discover both simulators and command both turtles.

Before we continue to the next step, stop the daemon and close all terminals.

Ex. 5 - Turtlebots

We will assign you to a group and a Turtlebot. Each robot is numbered. That number specifies the IP and the ID of the discovery server running on the robot.

To begin, we will set the discovery server and make sure we can see all nodes and topics on the robot. Create a new terminal and set the following:

Set robot server in discovery server variable
`export ROS_DISCOVERY_SERVER=”192.168.0.<20 + Robot Number>:11811”`
Set the terminal as super client
`export ROS_SUPER_CLIENT=1`

Ex. 5 - Turtlebots

Check that you can see the nodes and topics on the robot:

Restart the daemon
`ros2 daemon stop; ros2 daemon start`
Node list
`ros2 node list`
Topic list
`ros2 topic list`

Ex. 5 - Turtlebots

Undock the Turtlebot using the undock action:

List actions
`ros2 action list`
Undock
`ros2 action send_goal /<tb_ns>/undock irobot_create_msgs/action/Undock {}`

Note: the namespace will change depending on the robot number.

```
sudo apt install ros-humble-irobot-create-msgs
```

Ex. 5 - Turtlebots

Drive the robot around using the *teleop_twist_keyboard* package.

Run node
`ros2 run teleop_twist_keyboard teleop_twist_keyboard --ros-args -r __ns:=/<tb_ns>`

Make sure to set the namespace to match the robot number.

Ex. 5.2 - Bandwidth

Stop the teleop keyboard node. In the same terminal, we will try to visualize the camera data.

Try to check the default the camera topic:

Topic Hz
`ros2 topic hz /<tb_ns>/oakd/rgb/preview/image_raw`

If you’re lucky you might get some packets through. It is likely that no data is reaching your computer.

Ex. 5.2 - Bandwidth

We will instead try encoding the image topic.

Install ffmpeg image transport
`sudo apt-get install ros-humble-ffmpeg-image-transport`
Run the republisher
`ros2 run image_transport republish ffmpeg in/ffmpeg:=/<tb_ns>/oakd/rgb/preview/image_raw/ffmpeg raw out:=/<unique_namespace>/oakd/decoded/image`

Ex. 5.2 - Bandwidth

Now, open a new terminal, set your environment variables, and then try testing the ffmpeg topic.

Set robot server in discovery server variable
`export ROS_DISCOVERY_SERVER=”;;192.168.0.2<robot_number>:11811”`
Set the terminal as super client
`export ROS_SUPER_CLIENT=1`
Now, you should get the image date as a standard image topic
`ros2 topic hz /<unique_namespace>/oakd/decoded/image`
Open RViz to view the image
`rviz2`

RMW Zenoh

Troubleshooting

Troubleshooting Flowcharts

These flowcharts walk through the troubleshooting process for connecting to computers with ROS 2 communication and cover many common issues. It is divided into:

1. Network Integrity: Achieving ping

2. ROS 2 Comms: Achieving publish / subscribe

Troubleshooting Tools

ROS 2 Command Line Interface

- `ros2 node list` - check that the nodes are visible and running
- `ros2 topic list` - check that the topics are visible (have either a subscriber or a publisher)
- `ros2 topic info --verbose` - see the identity of the publishers and subscribers
- `ros2 topic hz` - check the frequency of messages received
- `ros2 topic bw` - check the bandwidth being used for a topic
- `ros2 topic delay` - calculated message delay using timestamps

Troubleshooting Tools

nmap

- Useful for detecting what is on the network and checking ports

bmon

- Useful for monitoring network bandwidth live in the terminal

Troubleshooting Tools

tshark

- Useful for capturing networking logs on the robot using command line to be analyzed visually in wireshark offboard

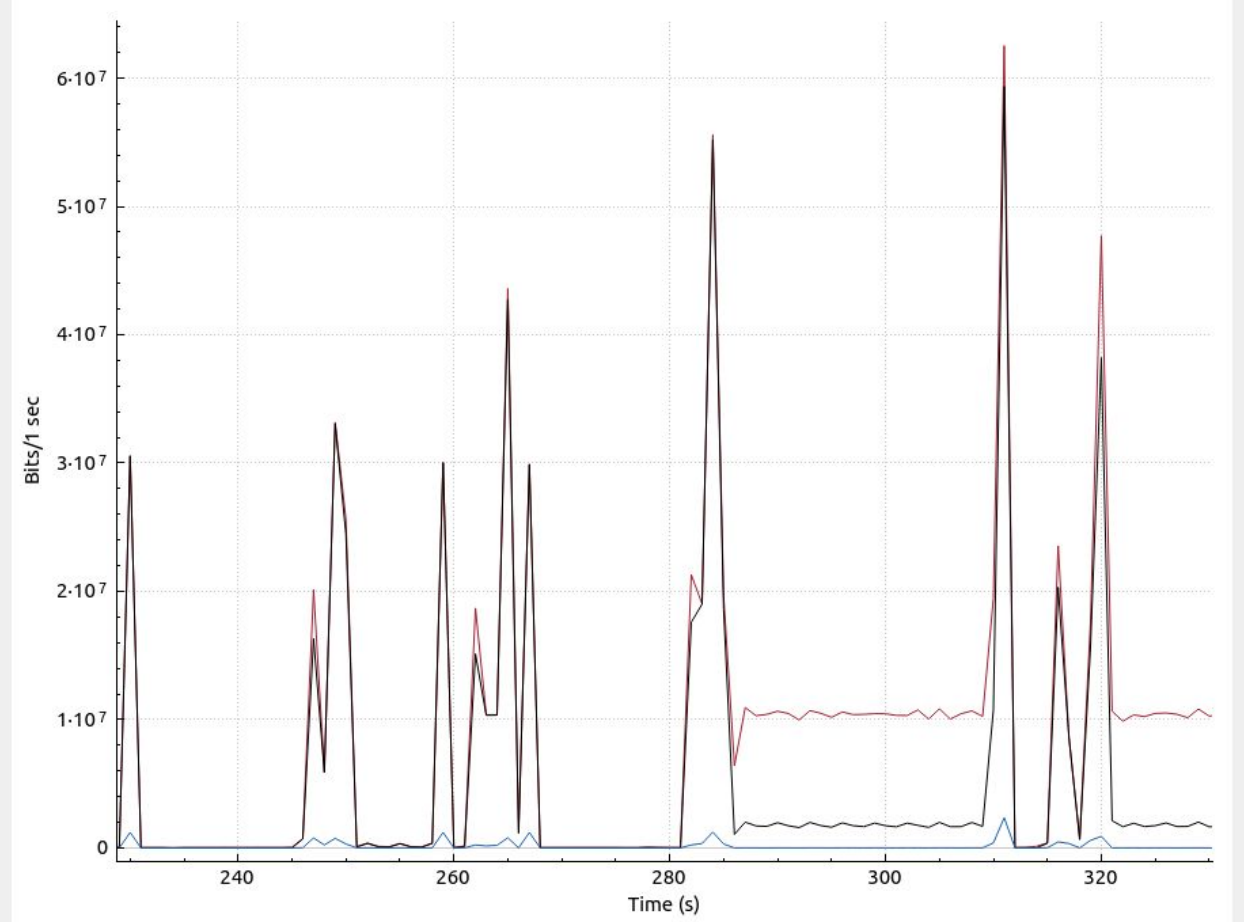
wireshark

- Useful for monitoring or analyzing ports in use, packets sent, bandwidth used and investigating individual messages
- If the capture includes the discovery process then wireshark will try to parse the DDS information and display topic names alongside the packets

Troubleshooting Tools

wireshark

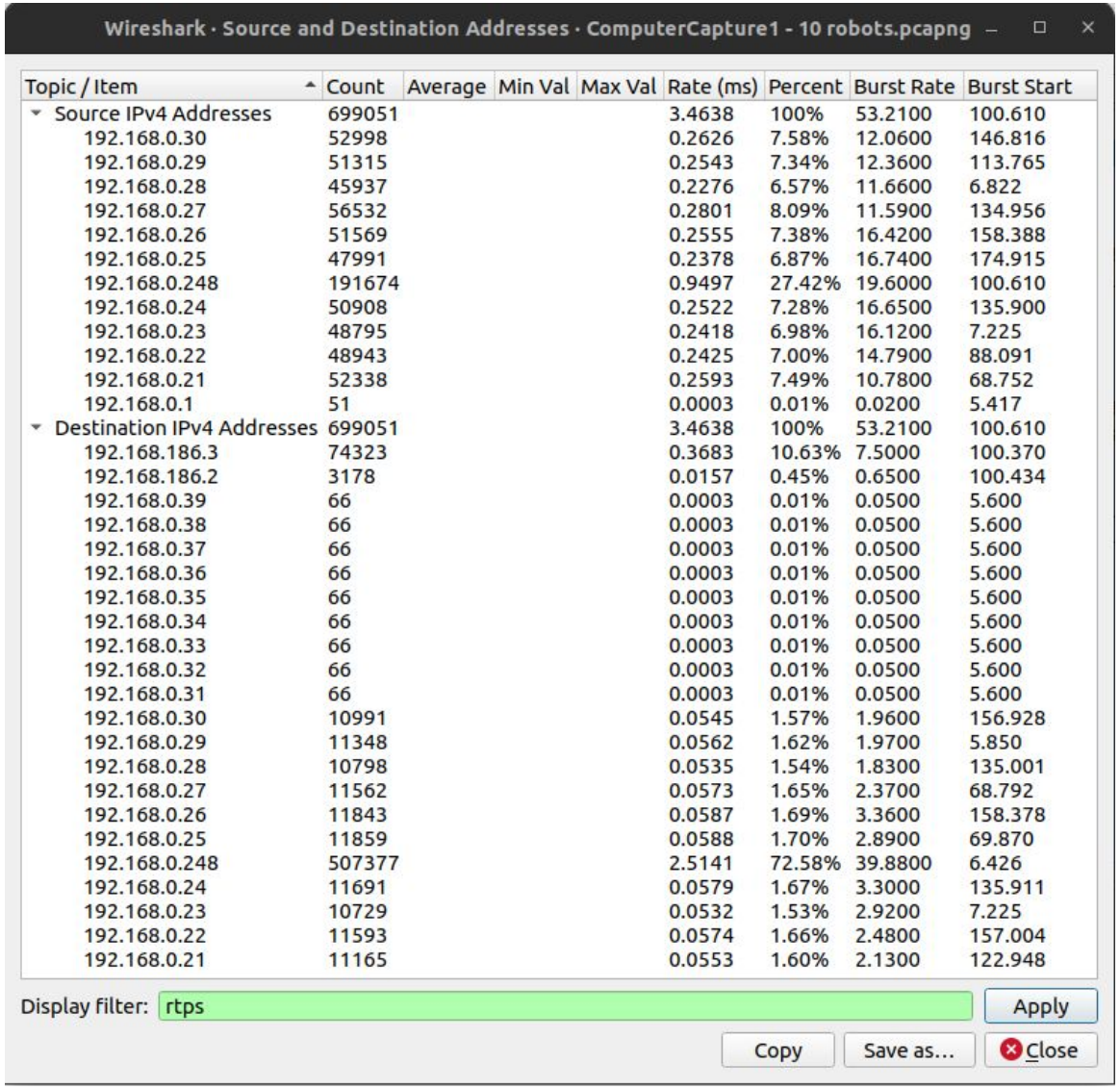
- Statistics -> I/O Graphs:
  - Useful for tracking number of packets sent and bandwidth usage over time with optional filters



Troubleshooting Tools

wireshark

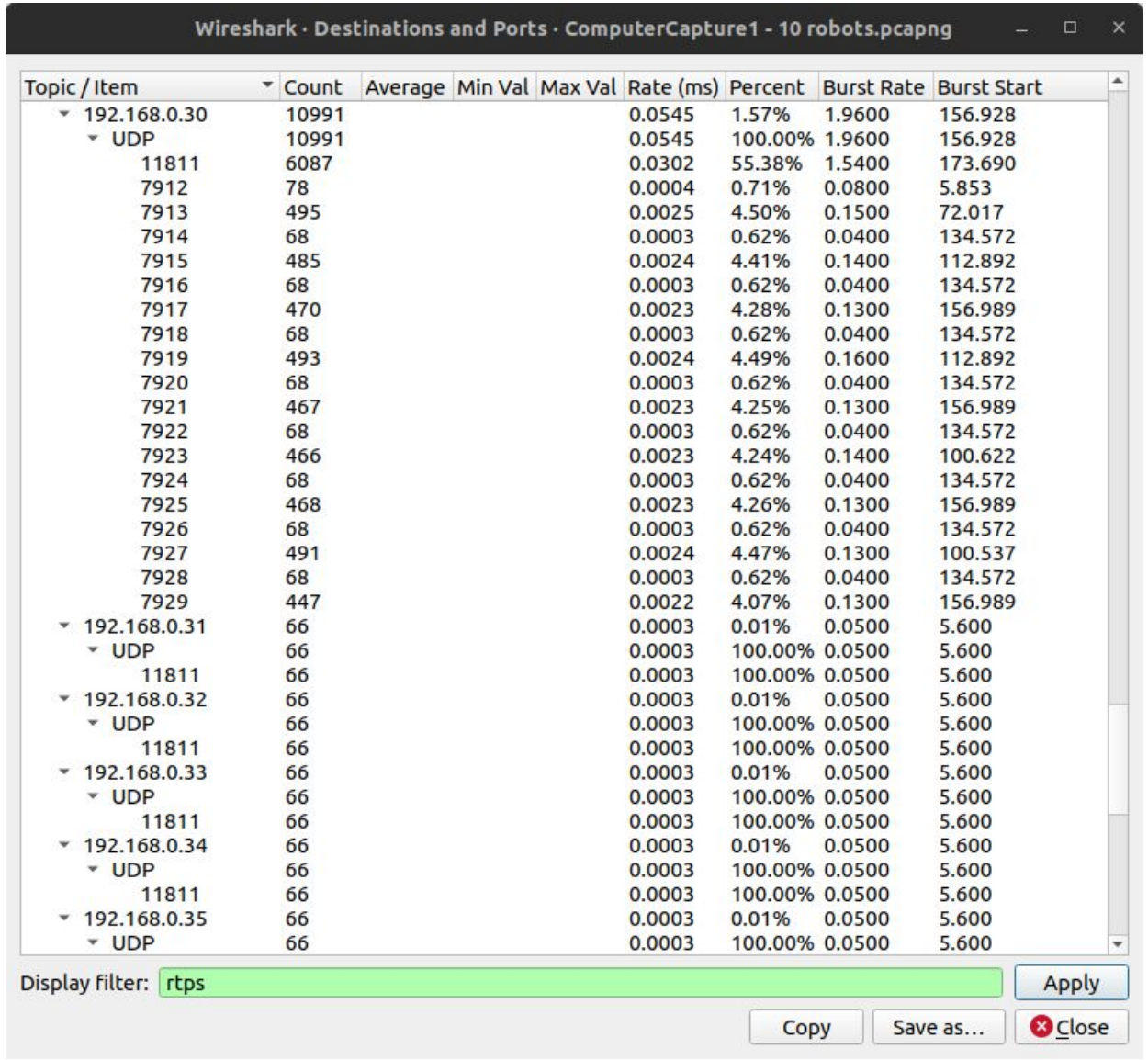
- Statistics -> IPv4 Statistics -> Source and Destination Addresses:
  - See which devices have sent / received how many packets



Troubleshooting Tools

wireshark

- Statistics -> IPv4 Statistics -> Destination and Ports:
  - See which ports are being used for each device and how many packets were sent to each port



Troubleshooting Tools

iperf3

- Tests and displays the bandwidth available across a network
  - Can specifically test TCP or UDP with different packet sizes etc
- Useful to determine maximum throughput of the network
- Useful to test communication at a particular port

Case 1

I have camera data that I want to publish for remote teleop operation. How can I calculate a reasonable publish frequency for my network?

- What data do I need to know?

Case 1

I have camera data that I want to publish for remote teleop operation. How can I calculate a reasonable publish frequency for my network?

- What data do I need to know?
  a. How big is each packet?
     - Using `ros2 topic echo --no-arr` I know the array is 1 MB long
     - 1 MegaByte = 8 Megabits
  b. What is the throughput on my network?
     - Using iperf3 I have 80 Mbps

Case 1

I have camera data that I want to publish for remote teleop operation. How can I calculate a reasonable publish frequency for my network?

- 80 Mbps = 10 frames per second

But is that reasonable?

Case 1

I have camera data that I want to publish for remote teleop operation. How can I calculate a reasonable publish frequency for my network?

- 80 Mbps = 10 frames per second

But is that reasonable?

No, the image should be compressed better before transmission to achieve a greater frame rate with a much smaller percentage of the bandwidth.

Case 1

I have camera data that I want to publish for remote teleop operation. How can I calculate a reasonable publish frequency for my network?

- After better compression:
  a. How big is each packet?
     - Using `ros2 topic echo --no-arr` I know the array is 0.1 MB long on average
     - 0.1 MegaBytes = 0.8 Megabits
  b. What is my throughput on my network?
     - Using iperf I have 80 Mbps

Case 2

A ROS 2 node has been started but the topics do not show up in `ros2 topic list`.

What could cause this? (Select all that apply)

A. The Discovery Server is not running or is not accessible
B. The terminal does not have the correct ROS environment variables
C. The node crashed
D. The ROS 2 Daemon was already running before the ROS environment variables were updated
E. `ros2 topic list` was run only once

Case 3

When SSH'd into the robot computer the topics show up in `ros2 topic list` but they are not visible on the offboard computer even after running it twice.

What could cause this? (Select all that apply)

A. Mismatching ROS Domain IDs
B. Mismatching hostnames
C. Not set as super client
D. Mismatching discovery configurations
E. ROS 2 Daemon isn't running

Case 3

When SSH’d into the robot computer the topics show up in `ros2 topic list` but they are not visible on the offboard computer even after running it twice.

What are the troubleshooting steps to determine the cause?

Case 4

Two computers are connected over ethernet and they can see each other's ROS 2 topics. They are then connected to the same WiFi network and the ethernet cable is removed. They can no longer see each other's ROS 2 topics.

What could cause this? (Select all that apply)

A. Configured for simple discovery but multicast is blocked by the router
B. Client isolation is enabled on the WiFi network
C. The nodes have not been stopped and restarted
D. The discovery server addressing is incorrect over WiFi
E. Non-standard ports are blocked by the router

Case 5

With Fast-DDS Discovery Server configured, the offboard computer can list the topics from the robot but not echo them.

What could cause this? (Select all that apply)

A. The computer hosting the discovery server is accessible but the robot computer is not
B. Topic has no publisher (only subscribers)
C. Incorrect Discovery Server IP
D. Didn’t wait long enough for the connection to be established
E. Incorrect ROS Domain ID

Case 5

With Fast-DDS Discovery Server configured, the offboard computer can list the topics from the robot but not echo them.

What are the troubleshooting steps?

Case 6

The camera video feed from the robot is being displayed on the offboard computer but the frame rate is much lower than expected. What could cause this?

A. The camera is not properly configured to the expected rate
B. The robot computer is unable to process the video at the requested rate
C. Network bandwidth is insufficient
D. Communication buffers are undersized for the data
E. The offboard computer is unable to process the video at the requested rate

Case 7

True or False:

To work with two robots on different ROS Domain IDs, different terminals can be started on a single computer with the corresponding ROS environment variables for each robot, and all command line interface (CLI) tools can be used on each without conflict.

Advanced Issue 1

Fast DDS Global Unique Identifier (GID or GUID) Host ID conflict:

- Host ID generated from the list of IP addresses of the device when the ROS nodes start
- GUID Host ID conflicts prevent successful communication
- Fix by ensuring all IP addresses are allocated prior to starting ROS nodes

Advanced Issue 2

Fast DDS Locator List does not update while running:

- When a node or server launches, it creates its locator list (a list of IP addresses where it is listening / can be accessed)
- When IP addresses change or are added on the device, the node locator lists do not update. If ROS is to be accessed over the new IP address, the nodes / servers on that device must be restarted.
- Symptom: ROS entities are discoverable onboard and not offboard (or on one device but not another)

Advanced Issue 3

Undersized Buffers or Incompatible Buffer Settings:

- IPv4 fragmentation buffer size / timeout duration can be a problem when transmitting large data and/or using a lossy network
- Symptom: Often some initial messages arrive and then most if not all other messages are lost
- See https://docs.ros.org/en/jazzy/How-To-Guides/DDS-tuning.html

Advanced Issue 4

Nodes available at multiple addresses cause duplicated traffic

- If a subscribing node has multiple IP addresses in its locator list, the messages from all topics it subscribes to will be sent in duplicate to each of those IP addresses

Advanced Issue 5

/rosout and /parameter_events topics

- These two topics are generally published by every node and /parameter_events is subscribed to by almost every node
- More ROS contexts (generally 1 per process) mean more discovery traffic, composable nodes can be used to optimize this

Advanced Issue 6

ros_discovery_info topic

- Invisible topic with reliable QoS that must be functional for full ROS functionality (only visible through packet capture)
- It rarely causes issues but can be a problem if your IPv4 receive buffer is excessively small and many subscriptions are created
- Symptom is very high bandwidth when no data is being requested
- See https://design.ros2.org/articles/Node_to_Participant_mapping.html

Thank you for attending!

Rockwell Automation | CLEARPATH ROBOTICS