ROS 2 for Consumer Robotics

The iRobot use-case

ROSCON
November 1st, 2019

Soragna, Alberto
Oxoby, Juan
Dhiraj, Goel

ROS 2 for Consumer Robotics

Agenda

01 iRobot's use-case

02 ROS 2 Performance Framework

03 Performance Evaluation

04 New Intra-Process Manager

05 Scalability

Our Portfolio

ROS 2 to rule them all?

desktop

Linux-based
low-cost
embedded

µC

✓

??

✓

(µROS)

Is ROS 2 suitable for Linux-based low-cost embedded platforms?

iRobot

01

iRobot’s use-case

ROS 2 for Consumer Robotics

- Low-cost Linux-based embedded hardware
- Single process application
- ~1000 robots in the same network (dev stage)

Defining a target platform

The goal is to run our entire navigation system on a RPi2

Raspberry Pi 2 Model B specs:

- 900MHz quad-core ARM Cortex-A7 CPU
- 1GB RAM

Defining our performance requirements

Adding a ROS 2 layer to our existing system should add a small overhead:

- less than 20% CPU
- less than 20MB RAM
- have acceptable latency
- zero lost messages

02

ROS 2 Performance Framework

The iRobot ROS 2 Performance Framework

- Arbitrary user-defined ROS 2 topologies
- Simple to use but highly configurable
- Human-readable results
- Open-source

https://github.com/irobot-ros/ros2-performance

The iRobot ROS 2 Performance Framework

INPUT:

topology
file(s)

(run …)

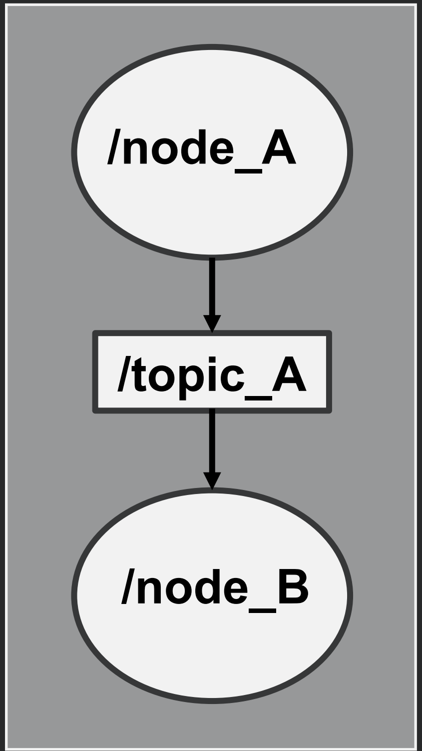
OUTPUT:

RAM
CPU%
latency
events

INPUT - Topology .json file

```
{
  "nodes": [
  {
      "node_name": ”node_A",
      "publishers": [
      {"topic_name": ”topic_A", "msg_type": ”1Kb", "period_ms": 10,
                     ”qos":”reliable"}
      ]
   },

   {
      "node_name": ”node_B",
      "subscribers": [
      {"topic_name": ”topic_A", "msg_type": ”1Kb", ”qos":”best-effort"}
      ]
   },
   ]
}
```



OUTPUT – Resources

time[ms]	cpu[%]	arena[KB]	in_use[KB]	mmap[KB]	rss[KB]	vsz[KB]
0	0	0	0	0	0	0
500	27	61080	60712	0	24948	305212
1000	51	167860	167539	0	47060	544012
1500	50	202844	202753	0	55496	660724
2000	43	202860	202769	0	55496	660724
2500	39	202876	202786	0	55496	660724
3000	36	202900	202801	0	55496	660724
3500	34	202928	202822	0	55496	660724
4000	33	202936	202837	0	55496	660724
4500	31	202948	202853	0	55496	660724
5000	31	202968	202868	0	55496	660724
5500	30	202992	202893	0	55496	660724
6000	29	203012	202909	0	55496	660724

OUTPUT – Events

Time[ms]	Caller	Code	Description
90	SYSTEM	0	[discovery] PDP completed
151	SYSTEM	0	[discovery] EDP completed
156	amazon->lyon	1	msg 0 late. 4300us > 2000us
156	danube->mandalay	1	msg 0 late. 4082us > 2000us
250	danube->ponce	1	msg 10 late. 2303us > 2000us
250	danube->geneva	1	msg 10 late. 2617us > 2000us
338	arkansas->arequipa	2	msg 0 too late. 184382us > 50000us
338	arkansas->arequipa	2	msg 1 too late. 136936us > 50000us
339	arkansas->arequipa	1	msg 2 late. 37064us > 5000us
2314	parana->geneva	1	msg 216 late. 2097us > 2000us
2314	parana->osaka	1	msg 216 late. 2131us > 2000us
3614	parana->osaka	1	msg 346 late. 2044us > 2000us
3615	parana->geneva	1	msg 346 late. 2682us > 2000us
3644	nile->hamburg	1	msg 349 late. 2234us > 2000us
3645	tigris->hamburg	1	msg 349 late. 2081us > 2000us

OUTPUT – Latency

node	topic	size[b]	received[#]	late[#]	lost[#]	mean[us]	sd[us]	min[us]	max[us]
lyon	amazon	36	12001	11	0	602	145	345	4300
hamburg	danube	8	12001	15	0	796	233	362	5722
hamburg	ganges	16	12001	10	0	557	119	302	4729
hamburg	nile	16	12001	18	0	658	206	300	5258
hamburg	tigris	16	12000	17	0	736	225	310	5994
osaka	parana	12	12001	32	0	636	236	346	4343
mandalay	danube	8	12001	16	0	791	189	418	6991
mandalay	salween	48	12001	1	0	663	297	391	6911
ponce	danube	8	12001	15	0	882	203	437	7270

received[#]	mean[us]	late[#]	late[%]	too_late[#]	too_late[%]	lost[#]	lost[%]
126496	744	204	0.161	2	0.0016	0	0

OUTPUT – Latency

Message classification by their latency

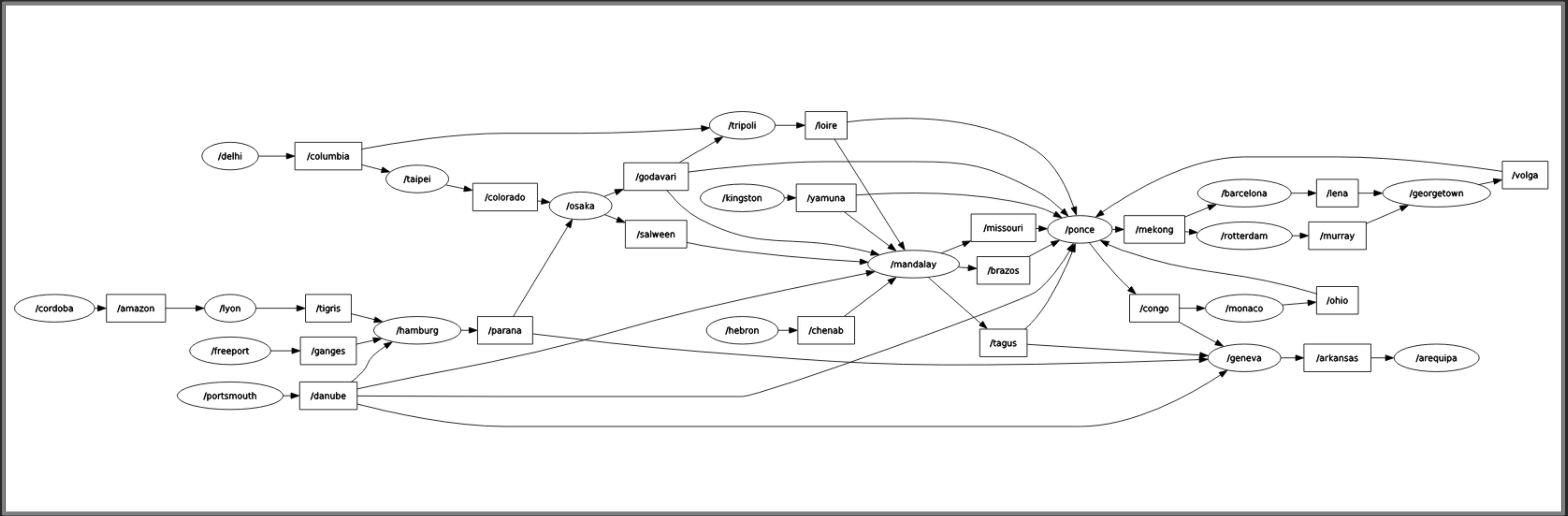
```
+                               +                              +
|                               |                              |
|                               |                              |
|                               |                              |
|                               |                              |
|                               |                              |
|                               |                              |
|                               |                              |
|                               |                              |
|                               |                              |
+--------------------------------------------------------------+
<-------> <---------------------> <--------------------------------------->
 on_time         late                          too_late

<------------------------------->
             period
```

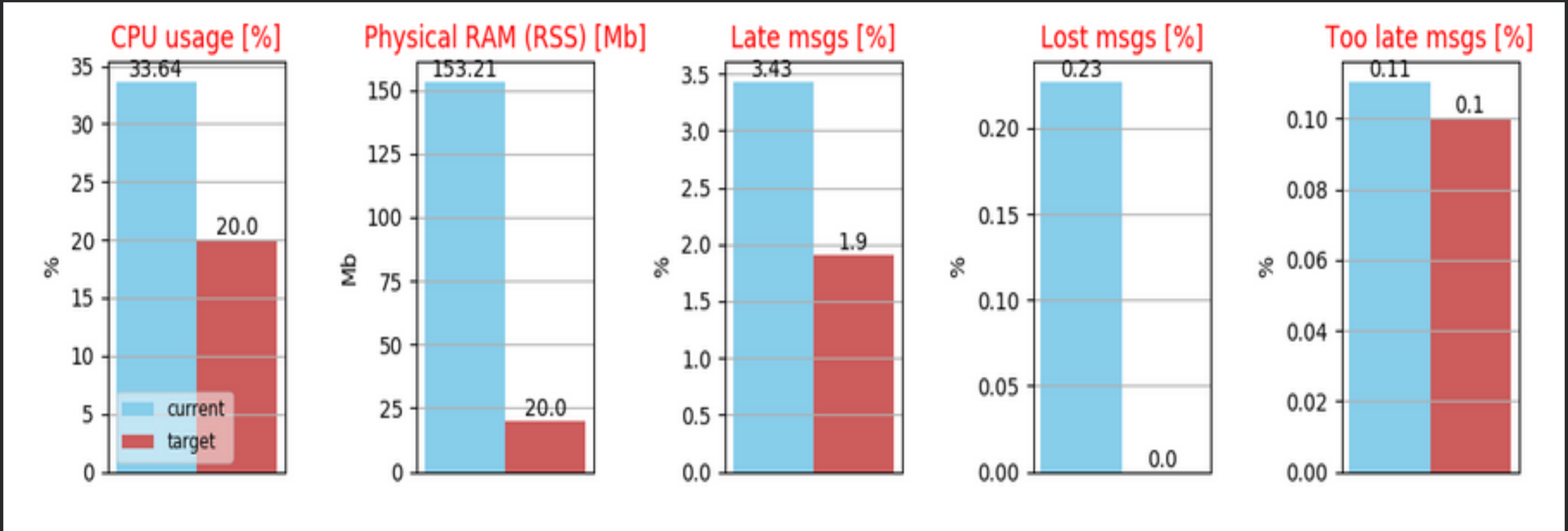
03

ROS 2 Performance Evaluation

ROS 2 Performance – Benchmark Topology

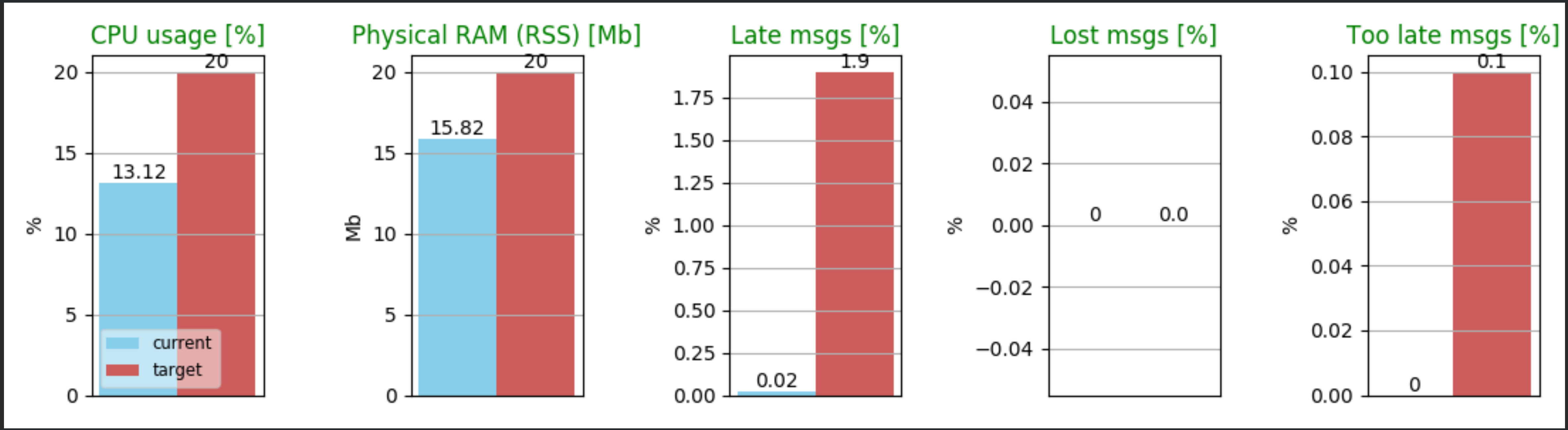


ROS 2 Performance – Fast-RTPS



Remember that this system is just doing message passing (no extra computation)

ROS 2 Performance – CycloneDDS



Reducing the ROS 2 RAM usage

The DDS specification requires that nodes create proxy objects for **EACH** entity they discover.

What if we know in advance that some entities don't need to know about some others?

BLACK-LISTING??

Considering our system, the RAM usage decreased to <u>45% of the original</u>!

iRobot

04

New Intra-Process Manager

ROS 2 Intra-Process Communication

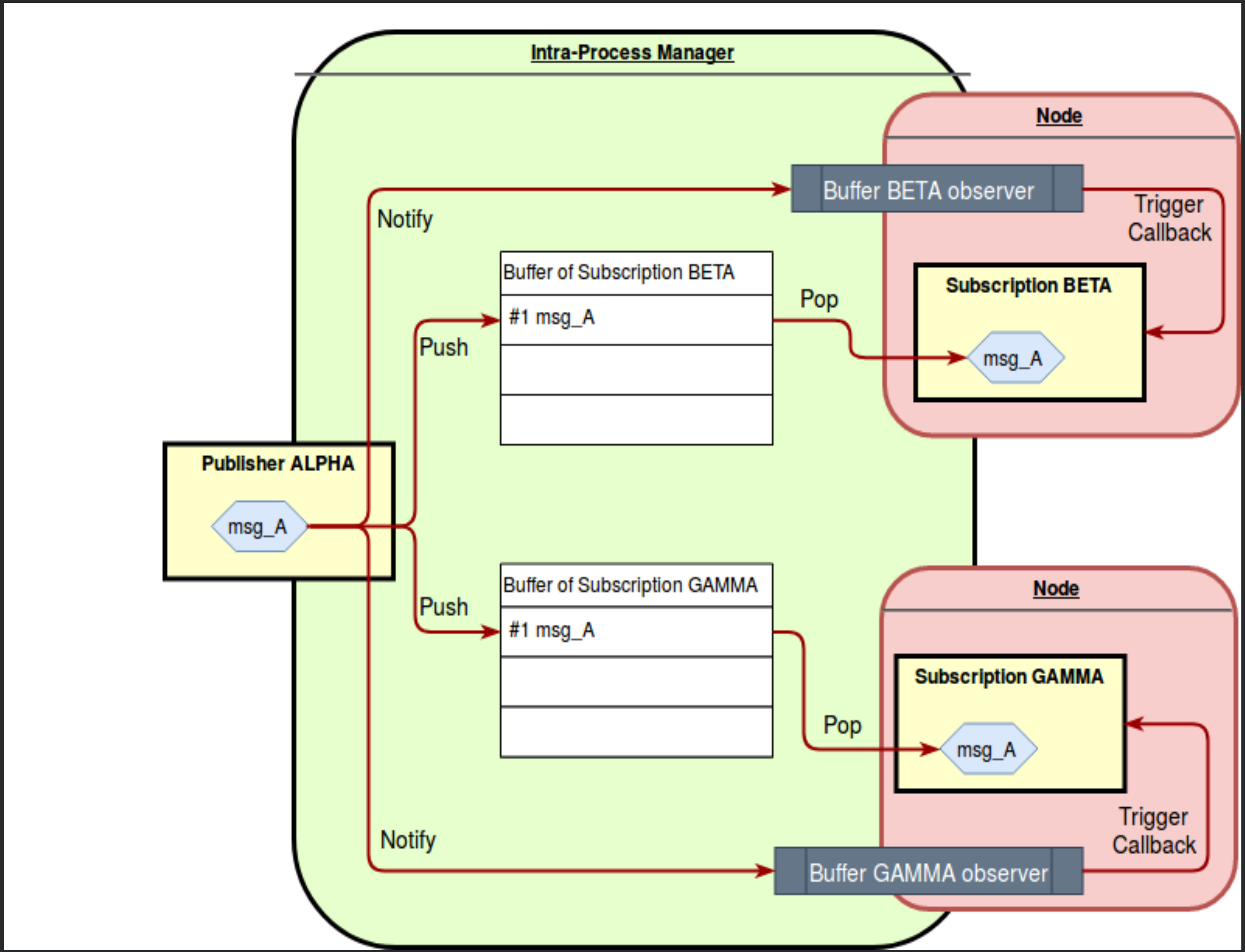
Dashing Intra-Process Manager

- Sends meta-messages through the RMW layer!
- Latency and CPU improvement only noticeable for big messages

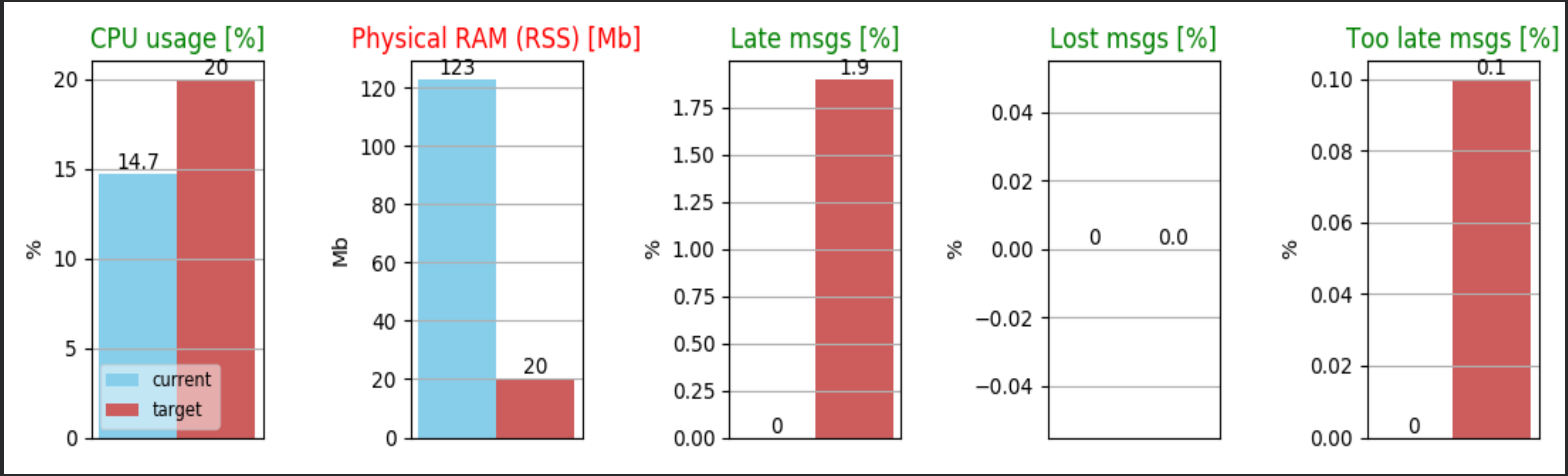
New Intra-Process Manager (now on master!)

- The communication is performed exclusively inside the RCLCPP layer
- Reduces CPU usage and latency for any message size

New Intra-Process Manager



ROS 2 Performance – Fast-RTPS + new IPC



45% CPU% decrease!

05

Scaling ROS 2

Problem Statement

~1000 robots in the same network

~35 topics from each robot

~5 different subnetworks across the globe

We want to be able to individually access any robot's messages from a remote debugging machine

Addressing Scalability

Proposed solution 1:
Use default DDS configuration and a different namespace for each robot

Problems:
ALL robots discover each other!
Slow/unreliable discovery, network saturation, multicast requires forwarding between subnets

Addressing Scalability

Proposed solution 2:
Solution 1 + ROS_DOMAIN_ID

Problems:
Limited numbers of domain IDs. What about assignment?
Still considerable network usage and multicast limitations

Addressing Scalability

Proposed solution 3:

Disable multicast discovery.
When a remote machine wants to connect to a robot, add the robot’s IP to the initial peers list (unicast)

Problems:

The user needs direct access to the DDS configuration. This can be solved by new APIs.

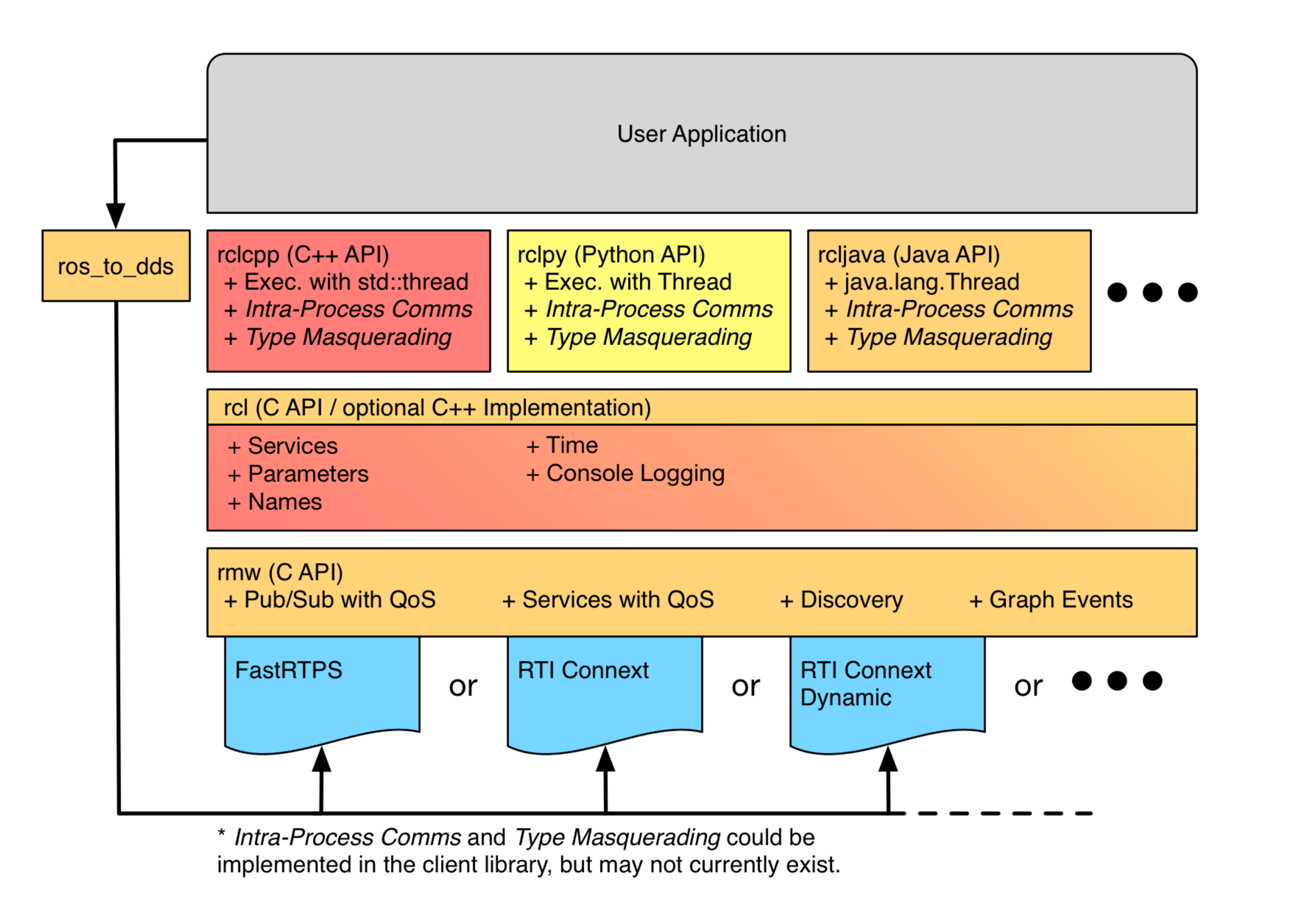
✓

Solution 3 with CycloneDDS

```xml
<CycloneDDS>
  <Domain>
    <Id>any</Id>
  </Domain>
  <DDSI2E>
    <General>
      <NetworkInterfaceAddress>auto</NetworkInterfaceAddress>
      <AllowMulticast>false</AllowMulticast>
      <EnableMulticastLoopback>true</EnableMulticastLoopback>
    </General>
    <Discovery>
      <ParticipantIndex>0</ParticipantIndex>
      <Peers>
        <Peer Address="${ROBOT_IP}:7410"/>
      </Peers>
    </Discovery>
  </DDSI2E>
</CycloneDDS>
```

```
ROBOT_IP=10.22.22.90 ros2 topic list
```

ros_to_dds ?



User Application
ros_to_dds
rclcpp (C++ API)
+ Exec. with std::thread
+ Intra-Process Comms
+ Type Masquerading
rclpy (Python API)
+ Exec. with Thread
+ Intra-Process Comms
+ Type Masquerading
rcljava (Java API)
+ java.lang.Thread
+ Intra-Process Comms
+ Type Masquerading
rcl (C API / optional C++ Implementation)
+ Services
+ Parameters
+ Names
+ Time
+ Console Logging
rmw (C API)
+ Pub/Sub with QoS
+ Services with QoS
+ Discovery
+ Graph Events
FastRTPS
or
RTI Connext
or
RTI Connext Dynamic
or
* Intra-Process Comms and Type Masquerading could be implemented in the client library, but may not currently exist.

irobot.com/careers