Robotics and Autonomous Systems 111 (2019) 73–87

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Pound: A multi-master ROS node for reducing delay and jitter in wireless multi-robot networks

Danilo Tardioli [a,*], Ramviyas Parasuraman [b], Petter Ögren [c]

[a] Centro Universitario de la Defensa, Zaragoza 50090, Spain
[b] University of Georgia, Athens, GA 30602, USA
[c] KTH Royal Institute of Technology, Stockholm 10044, Sweden

ARTICLE INFO

Article history:
Received 11 January 2018
Received in revised form 16 September 2018
Accepted 22 October 2018
Available online 31 October 2018

Keywords:
Multi-robot systems
Wireless multi-hop networks
Robot operating system
Jitter
Delay

ABSTRACT

The Robot Operating System (ROS) is a popular and widely used software framework for building robotics systems. With the growth of its popularity, it has started to be used in multi-robot systems as well. However, the TCP connections that the platform relies on for connecting the so-called ROS nodes presents several issues regarding limited-bandwidth, delays, and jitter, when used in wireless multi-hop networks. In this paper, we present a thorough analysis of the problem and propose a new ROS node called *Pound* to improve the wireless communication performance by reducing delay and jitter in data exchanges, especially in multi-hop networks. *Pound* allows the use of multiple ROS masters (roscores), features data compression, and importantly, introduces a priority scheme that allows favoring more important flows over less important ones. We compare *Pound* to the state-of-the-art solutions through extensive experiments and show that it performs equally well, or better in all the test cases, including a control-over-network example.

1. Introduction

The Robot Operating System (ROS),[1] introduced in 2007 is one of the most used open-source robotics middlewares and a flexible framework for writing robot software. ROS consists of a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior, across a wide variety of robotic platforms [1]. Similarly to other widely-accepted robotics middleware frameworks such as MIRO, ORCA, YARP, OROCOS, MRDS, etc., the main objective of ROS is to enable rapid research and development, in both academia and industry [2].

Recent years have witnessed significant interests in multi-robot systems in applications such as search and rescue, autonomous exploration, cooperative sensing, and adaptive communication infrastructure [3]. The successful execution of time-critical multi-robot missions is often dependent on meeting a set of requirements from a communication point of view. The timing of the data delivery, such as jitter (variation of delay) or excessive delays, can have a crucial impact on the performance of control loops and can compromise the mission.

Although many middleware frameworks are comparable to ROS regarding flexibility and features for single robot applications, some studies (e.g., [4]) suggest that ROS is better suited to multi-robot applications. Therefore, it is important to address the challenges specific to multi-robot networks, especially in meeting the wireless communication requirements in terms of bandwidth (throughput) and latency (delay and jitter).

ROS systems are based on so-called *nodes*, each in charge of performing a specific task and offering the results to the user or to other nodes, e.g., a Simultaneous Localization And Mapping (SLAM) node using the output from a LIDAR node publishing laser scan data. These nodes can have multiple inputs and outputs, and the exchange of information (flows) takes place through *topics* (named ports that hide simple network socket connections).

When the different nodes reside in the same computer, these connection are local (through the *loopback* interface), but when they reside on different computers/machines, they are usually connected either through a wired or a wireless network. Wired connections can handle extraordinary high amounts of data from ROS nodes, and therefore, neither the communication bandwidth nor the stability of the connection presents a problem, especially in modern systems equipped with Gigabit Ethernet cards. However, in the case of wireless connections, the reliability of the communication channel is a couple of orders of magnitude lower. Specifically, the available bandwidth is much smaller, and the number of retransmissions needed can increase jitter significantly, as seen in Fig. 1.

In multi-robot applications, the exchanged data have a periodic and perishable nature, such as the distributed construction of a map [5–7], cooperative localization [6], and audio and video

* Corresponding author.
E-mail addresses: dantard@unizar.es (D. Tardioli), ramviyas@uga.edu (R. Parasuraman), petter@kth.se (P. Ögren).

[1] http://wiki.ros.org.

https://doi.org/10.1016/j.robot.2018.10.009
0921-8890/© 2018 Elsevier B.V. All rights reserved.

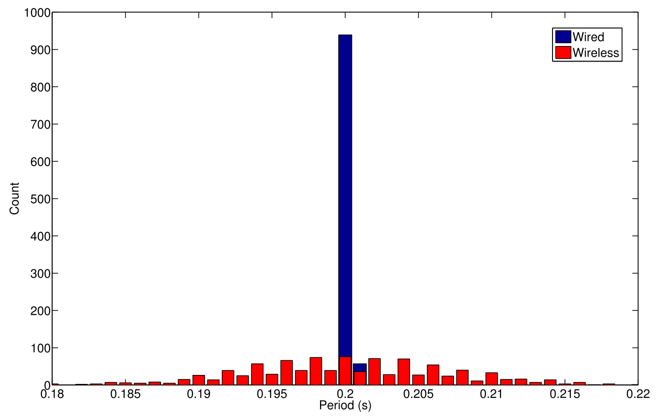


Fig. 1. Comparison between wired and wireless configuration. Period measured at the destination node in a system in which a flow with period of 200 ms is established over wireless and wired connections.

streaming over multi-hop networks [8]. In these networks, the data exchanged between robots must be transferred coherently within firm time windows to allow the task to be completed successfully.

Note that these primary data flows are not usually the only ones present in the network; they must share the available bandwidth with less important flows without being perturbed, which creates a need for prioritizing the different flows. Further, the situation gets worse when the communication channel is used to close a perception–actuation loop, possibly including a human (formation control [9], teleoperation, distributed control [10], etc.). In this case, not only is the communication bidirectional, which means that the prioritization of the traffic must be network-wide, but also the deadlines are strict rather than firm. Therefore, the late arrival of data can cause serious issues such as collisions, and jeopardize overall task completion. Furthermore, old data is generally not useful in robotic application needing (quasi) real-time behavior. This makes UDP preferable to TCP, especially because TCP only guarantees the delivery of the data but not the timing.

Therefore, in this paper, we propose an improved solution for multi-robot communication based on ROS through the development of a new UDP-based multi-master ROS node called *Pound*, (Priorities Over UDP for reducing Network Delay). *Pound* acts as a kind of proxy, combining the use of multiple ROS masters (cores) with a priority scheme, allowing the user to assign levels of importance to different topics with the objective of reducing the jitter and the delay for data that is considered more important. Furthermore, *Pound* includes compression using *Bzip2* to reduce the message size in the ROS topics and is available completely open source in Github.[2] The main contributions of this paper are presented below.

- A detailed analysis of the problem is presented, where the behavior of different connection methods provided by the ROS platform is discussed, in addition to related ROS multi-master solutions available in the literature.
- A new multi-master ROS solution, *Pound* is proposed, which goes beyond the state-of-the-art by providing a system of priority in a per-flow base favoring more important flows while avoiding network congestion and blockages.
- The theoretical advantages of *Pound* are verified through extensive experiments evaluated under single hop and two hop networks in a scenario with two flows of information of different importance concerning jitter, delay, and bandwidth. *Pound* is compared with the state-of-the-art approaches such as *Nimbro*[3] [11], *RT-WMP*[4] [12], and the standard ROS communication protocols.
- Finally, the effects of the above protocols on a real-time control loop are investigated.

The paper is organized as follows. We review the literature in Section 2. Section 3 provides an overview of ROS along with its challenges in wireless networks. In Section 4, we present the different solutions analyzed in this work and introduce *Pound* in Section 5. Then, we describe the experiments performed and present the results in Section 6. We discuss the results in Section 7 and conclude the paper in Section 8.

2. Related work

In this section, we review the related work on wireless communication in robotic applications. This problem has been addressed in both the robotics community and the communications community, with each of them having slightly different problem formulations and corresponding solutions. We begin the literature review from a communications point of view and then discuss the related works in the robotics community, including improvements working directly on ROS.

The problem with wireless networks is addressed mostly with modifications or adaptations of protocols. For instance, [13] presents the problems with the predominately used TCP/IP protocol in lossy wireless networks and suggest some possible proactive schemes such as TCP-Westwood and TCP-Jersey as replacements (or enhancements) to standard TCP to counter the issue of link instability and high Bit Error Rate (BER). However, such enhanced protocols add overheads which may have a significant negative impact in dynamic robotic networks. Deek et al. [14] advised the use of channel bonding in typical 802.11n WLAN to improve the data rates. Paasch et al. [15] proposed the Multipath TCP (MPTCP) protocol which uses all available interfaces in a terminal to increase the application throughput. Other solutions like [16] aim to provide support traffic scheduling, implementing multiple traffic classes with different priorities transmitted according to a fixed schedule. Although these improvements help solve some of the fundamental problems with the wireless network, most of the solutions do not solve the inherent latency and jitter problems in a multi-hop (multi-robot) network.

From a robotics perspective, it was noted in [17], that network parameters such as delay, jitter, and bandwidth (throughput) are the most common influencing metrics in Human–Robot Interaction (HRI) problems. These are important metrics because they affect the robot missions effectiveness and efficiency directly. For instance, [18,19] discuss how the delay between robot and the teleoperator can affect the task performance negatively. Faster completion time is crucial in tasks such as search and rescue during disasters because every second may be utilized in saving more lives in such situations. Moreover, [20] has demonstrated the importance of jitter as an indicator of the communication performances in a wireless networked robotic system. It also shows how variations in jitter compromise teleoperation performance and proposed a control algorithm to handle the variable communication delay in a WLAN for bilateral teleoperation. However, such solutions are application specific, and use tools from robotic control theory.

Birk et al. [21] focused on a wireless network for teleoperation application especially for search and rescue robots and proposed the Jacobs Intelligent Robotics Library (JIRlib) that features compression, prioritization, and serialization framework to exchange data between robots and control station over typical TCP/IP framework. Although an attractive robotic communication framework,

[2] https://github.com/dantard/unizar-pound-ros-pkg/.

[3] https://github.com/AIS-Bonn/nimbro_network.

[4] https://github.com/dantard/unizar-rt-wmp-ros-pkg/.

it is not integrated into ROS and therefore cannot be readily used. In some works, the problems of wireless communications are dealt with differently. Assuming poor connection quality with standard wireless networks, authors in [22,23] proposed enhanced visual and haptic feedback mechanisms to help the robot operator perceive directional wireless connectivity and thereby reduce the risk of communication failure. Wireless tethering and spatial diversity techniques are exploited in [24,25,6]. In [26], a Mobile Ad-hoc Network (MANET) using Unmanned Aerial Vehicles (UAV) is proposed for improved communication between robots. However, most of these schemes rely on a centralized approach, which may not be directly applicable to multi-hop, multi-robot networks.

Only a very limited amount of work can be found in the literature that discusses how ROS handles data flows and proposes solutions to improve the use of ROS in difficult network situations. For instance, [27] presents a reporting framework for providing network statistics, such as bandwidth and frequency, on every ROS node. They also further extend the framework using existing ROS tools (diagnostics tool, *rqt_graph*, etc.) and demonstrate their ideas in applications such as robot health monitoring. In [28], a similar approach is taken to provide advanced features to monitor and introspect a ROS system through a tool called ARNI.[5] Authors in [29] provided a nice comparison of various multi-agent and robotics middleware frameworks and emphasized the inability of ROS to provide real-time capabilities due to its dependency on TCP/IP. In [30], the authors propose a hybrid ROS architecture that combines a PC acting as a server (e.g., for visualization) and embedded systems (e.g., for real-time computations) to achieve a good balance between flexibility and real-time capability in heterogeneous multi-robot systems. In [31], the authors proposed a ROS tool called *dynamic_bandwidth_manager* (DBM),[6] which maximizes bandwidth usage in multi-robot systems by controlling the rate at which a ROS node publishes a topic. DBM is also designed to prioritize communication channels that transport important ROS topics, similar to the aim of *Pound*. However, it is worth noting that these network monitoring tools and ROS architectures do not provide new means of communication, but rather adapts the existing communication methods to the needs of the application.

In this paper, we go beyond the work described above and analyze the ROS framework and existing ROS-compatible network protocol extensions in the context of wireless networked mobile robots. Moreover, standardizing wireless communication for mobile robots is advocated in [32,33] as a key area of improvement. Authors in [33] suggested an architectural framework for mobile robot use cases and applications considering various layers in the OSI reference model. This paper provides one step in that direction, by identifying drawbacks in, and proposing solutions for, the most widely used ROS framework.

3. Problem analysis

In this section, we will first give an overview of ROS, and then describe the different challenges associated with running ROS over a multi-hop wireless network. Finally, we investigate the different solutions proposed so far to address the problems.

3.1. ROS overview

As described above, ROS is a flexible framework for writing robot software. It acts as a meta-operating system for robots as it provides hardware abstraction, low-level device control, inter-processes message-passing, and package management. It also provides tools and libraries for building, writing, and running code in a single or across multiple computers. A robotic system can be built as a set of so-called ROS nodes that, in turn, contain the processes in charge of producing and consuming information. ROS nodes are connected through ROS topics; labeled data streams used to pass information among them.

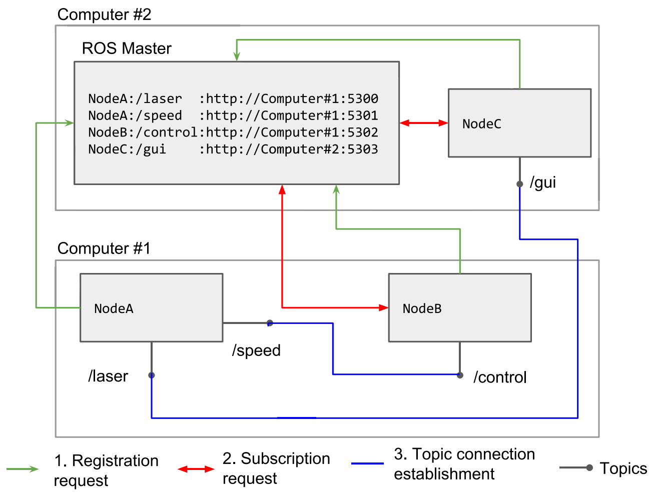

Computer #2
ROS Master
NodeA:/laser :http://Computer#1:5300
NodeA:/speed :http://Computer#1:5301
NodeB:/control:http://Computer#1:5302
NodeC:/gui :http://Computer#2:5303
NodeC
/gui
Computer #1
NodeA
NodeB
/speed
/laser
/control
1. Registration request
2. Subscription request
3. Topic connection establishment
Topics

Fig. 2. An overview of a ROS based robotics system. The ROS nodes, once launched, register themselves in the ROS master (which resides in one of the computers), specifying the address and port of their published topics (green lines). When a node wants to subscribe to other node's topic, it asks the master (*roscore*) which responds with the correspondent address and port (red lines). Finally, a direct topic connection is established between the publisher and the subscriber (blue lines).

Beside the publish/subscriber, ROS also supports the client/server paradigm through the ROS services. The exchange of information among nodes is carried out through simple network sockets using both connection-oriented and connectionless protocols called TCPROS and UDPROS (defined on top of the standard TCP and UDP protocols). Also, there exists the option of disabling the TCP's Nagle's algorithm using the so-called TCP_Nodelay option. All this allows having multiple local nodes and to distribute them in different computers. A special node called ROS master (or *roscore*) is in charge of establishing the required connections among the nodes. It acts similarly as a DNS: when a node is started, it registers itself and its topics (i.e., their address and ports) in the ROS master. When another node needs to subscribe to a specific topic, it asks the master –by providing the name – at what address and port that topic is located, and afterward establish a direct connection with the publisher node.

In Fig. 2, a generic overview of a typical ROS robotic system is presented in which multiple ROS nodes are running (in *Computer #1*) that publishes/subscribes to ROS topics from another computer (*Computer #2*) via the ROS master (core) situated on *Computer #2* as well. Among many references available in the literature for ROS, [1] provides a nice overview of how ROS works and how to use it in practical robotic applications.

ROS2: The second version of ROS (called ROS2) is under development by the ROS community. The main motivations for the ROS2 design are to meet real-time capability requirements and enhance multi-robot communication performance. ROS2 will use the Data Distribution Service (DDS) as its networking middleware because of several advantages such as the distributed nature and Quality of Service (QoS) based configurations. However, as ROS2 is still under initial development, it is still unclear when a switch from ROS1 to ROS2 will be possible. Thus, we focus this paper on problems, and corresponding solutions, of the current ROS framework.

Note that the below-mentioned problems, although applicable to ROS as it depend heavily on the TCP/IP standard, are generic to

5 http://wiki.ros.org/arni.
6 http://wiki.ros.org/dynamic_bandwidth_manager.

any wireless multi-agent systems. The inherent latencies caused by the ROS framework are negligible compared to the network latencies.

3.2. Reliability problems

ROS was originally designed for systems in which multiple computers residing in a single robot were connected through a LAN as a way to distribute the computational load in different units. However, given that the connection among ROS nodes is based on simple TCP connections, it is possible to distribute the nodes on computers connected differently, as long as the lower-level communication is based on the TCP/IP (or UDP/IP) stack. This includes, of course, wireless networks.

However, as is well known, wireless communications are not as stable as their wired counterparts. The error rate is much higher, at least two orders of magnitude in optimal conditions. This means that often frames must be retransmitted several times at the MAC level (e.g., the default retransmission count is fixed at 7 for the 801.11 protocol in common Linux distributions) and this provokes a spreading of the communication end-to-end delays and jitter.

To illustrate the differences regarding the performance between distributed wired and wireless systems, we performed an experiment in which a distributed ROS system was simulated in both configurations. Two computers were directly connected first through Ethernet interfaces (100 Mbps) and then using two 802.11g wireless cards at 54 Mbps. We generated a single flow with 64 KB bytes messages and a period of 200 ms between them.

Fig. 1 shows the distribution of the inter-arrival delay of the messages or, in other words, the period at the destination node. As expected, the figure shows a normal distribution around the original flow period of 200 ms. However, as can be seen, the wireless scenario presents a much higher degree of jitter showing a much wider normal distribution around the expected period. This impacts the precision of the system and makes it more difficult to implement a high-performance control loop if needed. Notice that the network was far from saturated in both cases, as the required bandwidth of $\approx$2.5 Mbps (plus TCP/IP overhead) was far below both the theoretical limit of the 802.11g protocol and, of course, the 100 Mbps of the Ethernet standard used. This trivial example demonstrates what we should expect from wireless links: higher jitter and thus less precise control of distributed systems.

3.3. Bandwidth limitations

Even as the bandwidths claimed by wireless standards are getting close to their wired counterparts, and sometimes even exceed them (the 802.11ac can theoretically reach a bandwidth of 3.67Gbps, for example), the performance, in reality, is often much worse. The claimed bandwidths are only reached when sender and receiver are close enough (e.g., the 802.11g standard guarantee a rate of 54 Mbps at a maximum distance of 37m indoor), the signal strength is high enough (generally this means they are in the line of sight of each other), and there is no interference, etc. Furthermore, the real bandwidths are sometimes far from the theoretical ones, even in optimal conditions. Finally, and probably most importantly, the available bandwidth is reduced due to the impossibility of spatial reuse, inevitable in a large variety of situations. It can occur both due to the use of a wireless access point (AP) or in multi-hop ad-hoc communication.

In the first case, all the network nodes are in the range of communication of the AP and all the communication passes through it: schematically, the source sends a frame to the AP that in turn forward it to the receiver. This means that only one node can transmit at a time otherwise the AP would lose the reception of the frames due to the collision. Moreover, as explained, the propagation of the

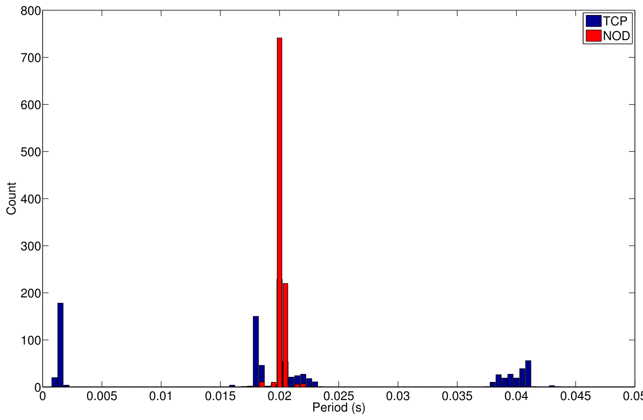


Fig. 3. Effect of the Nagle's algorithm.

cited frames involves the exchange of two frames; this reduces the bandwidth to the half, like in a two-hops communication.

In the second case, something similar happens: if to extend the range of a hypothetical robot-team its members form a chain configuration (consider a search and rescue scenario in a long tunnel, for example, [6,34]), the frames exchanged by the robots in the two ends would need to hop through all the other relay nodes. This would reduce the available bandwidth by a factor $1/(n-1)$ in the worst case, where n is the number of nodes (computers) involved. Notice that even if in this situation some kind of spatial reuse could be possible, it strictly depends on the specific configuration.

Bandwidth limitation is especially important in distributed ROS systems: if different ROS nodes subscribe to some other ROS node's topic, one network connection is established for each one of them. This could be especially dramatic if all connections take place over a wireless network, given that the bandwidth required is multiplied by the number of subscribers. Also, if the frequency a topic is published with is higher than that required, more bandwidth than the strictly necessary will be needed, jeopardizing global performance. To be fair, some of these limitations can be overcome using the so-called *topic_tools* (*relay* and *throttle* nodes can do the trick) but they still do not offer a general and specific solution for wireless networked systems. Another aspect that should be taken into account is the fact that, by default, ROS distributed systems rely on a centralized ROS core as explained above. The handshakes necessary to connect different nodes –also carried out establishing TCP connections – add extra overhead that has a significant impact on the required bandwidth.

3.4. Delays due to Nagle's algorithm

The connection between ROS topics is performed using a TCP connection by default. However, the Linux's implementation of the TCP protocol uses, again by default, the Nagle's algorithm to improve the efficiency of TCP/IP networks by reducing the number of packets sent over the network. However, this can provoke delays in the transmission of the data that are buffered at the sender side before being transmitted. This means for example that two ROS messages can be transmitted at the same time even if they are generated at different moments. Unfortunately, this behavior takes place both in local and in over-the-network communication and can be extremely harmful in case of tight control loops. Fig. 3 shows a situation in which a single flow with 1 KB messages and 20 ms period is exchanged between two ROS nodes using two different connection options. The graph reports the period measured at the receiver side; as can be seen TCP_Nodelay (NOD in Fig. 3) shows a narrow distribution around the source period while the

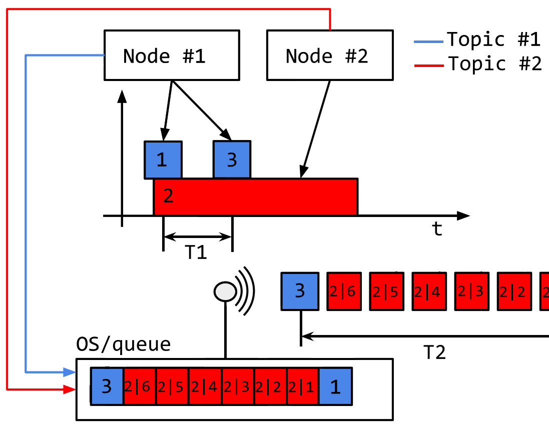


Fig. 4. Delay due to competing flows. The packet corresponding to message #3 is delayed by packets corresponding to message #2 (#2|1 through #2|6) in which it is split when entering the OS queue.

TCPROS (TCP in Fig. 3) shows three of them: around the real period, the double of the period and around 0 ms. As anticipated, this is because, sometimes, two consecutive messages are buffered on the sender side and then sent together. At the receiver side they are, instead, published as soon as received through the socket.

3.5. Delays due to competing flows

Another aspect, the most important one in our view, is the performance of the communication in the presence of competing flows. This issue does not only appear in wireless networks but is more marked in this configuration due to the more limited bandwidth (especially in multi-hop networks, as described above) and the lossy nature of this medium. When two or more flows have to share the same communication channel (either Ethernet or Wireless) the bandwidth must be shared as well, as is obvious. This should not be a problem if the flows do not saturate the network, but the transmission of one flow's messages can delay the transmission of the packets belonging to the other flow.

What happens in detail is that when a node (or any application) sends any information (messages) through a socket, this is split in different MTU-sized packets (the MTU depends on the specific physical layer) and added to the FIFO operating system transmission queue. Since packets are then sent following order by their position in the queue, any long messages introduce a noticeable delay in the transmission of the packets that are behind them in the queue. It can be observed from Fig. 4 that the flow generated by *Node #1*, despite having a period $T1$, will see the packet corresponding to the messages published with a period $T2 > T1$. This is because a message coming from *Node #2* is pushed in the OS queue after message *#1* but before message *#3*. That is, message *#3* must wait for the transmission of packets *#2|1* through *#2|6*, which is noticeably being delayed. Just to give an example, a message of size 64 KB in a 802.11, 6 Mbps network, with a MTU size of 1500 Bytes is split in $\lceil 65536/1500 \rceil = 44$ radio packets, and can introduce a delay of roughly $44 \cdot (1500 \cdot 8) \cdot 10^{-6}/6 = 88$ ms (or 10 ms in case of a 54 Mbps network).

3.6. Blackouts due to routing issues

Another aspect that should be treated although out of the scope of this paper, is the routing of the traffic. ROS itself does not offer any support for traffic routing that in case of a wireless network must be performed by lower-layer protocols like OLSR or BATMAN. However, these protocols have been proven very inefficient in case of node mobility provoking blackouts of several seconds, which is not favorable for robot control tasks [20].

3.7. Resilience

Resilience to intermittent network disruptions is a desirable feature in any robotic system, particularly in field robots. Re-establishing the connectivity after disconnection is a core part of network resilience. Although ROS (TCP connection) can handle the network disruption effectively, it is important to know how different protocols impact the efficiency of re-connection. In general, TCP connections are slower in getting reconnected, whereas UDP re-connections happen faster as they do not need a handshaking process. For instance, this is the same reason why *MOSH* [35], a remote terminal application, is preferable over the widely-used *SSH*, for its support and robustness in dealing with intermittent connectivity.

4. Solution candidates

We need solutions that address the above problems and at the same time, can also support multiple ROS masters so that each robot (or computer) can perform its operation individually. Thus, the robots can be tolerant of communication disruptions or intermittent failures.

There are a few existing ROS tools that provide multiple ROS master support which offer a way of detecting other cores on the same or neighboring networks simplifying the development of multi-master systems. For instance, *multimaster_fkie* [36] helps to bridge multiple robots (computers) running ROS master in each of them but without actually improving the way the traffic is exchanged among the nodes. On the contrary, *Nimbro*, *Pound* and *RT-WMP* propose a different, and in principle more efficient way of sending data from a source to a destination. Therefore, the following candidate solutions considered in this section are:

- Standard ROS
  - TCPROS (Referred as TCP in this paper, based on TCP)
  - UDPROS (Referred as UDP in this paper, based on UDP)
  - TCP_NODELAY (also referred as NOD is this paper; it corresponds to TCPROS without the Nagle's algorithm)
- *Nimbro* network
- *RT-WMP*

Unlike standard ROS, the last two solutions, together with *Pound* proposed here, rely on multiple *ros-masters* (one per each node involved in the multi-robot team) and act similarly to a communication proxy, sending over the network only the actual ROS messages exchanged among the nodes encapsulating them in other lower-level protocols, as detailed in the next sections.

There also exist some other ROS multi-master solutions (from individual researchers) such as *multimaster_experimental*,[7] *wifi_comm*,[8] and *multimaster-ros-pkg*.[9] However, they are deprecated in the latest ROS versions, and therefore, not discussed in this paper.

[7] http://wiki.ros.org/multimaster_experimental.
[8] http://wiki.ros.org/wifi_comm.
[9] https://github.com/jonfink/multimaster-ros-pkg.

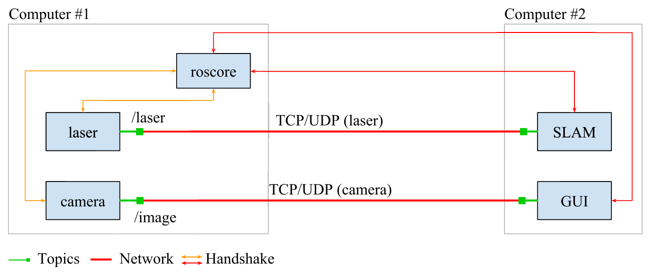


Fig. 5. A hypothetical 2-computers/2-topics system relying on standard ROS communication protocols. The LIDAR and camera ROS nodes publish data coming directly from the hardware while SLAM and GUI ROS nodes consume this information to create a map and to display the images respectively. One TCP or UDP connection is created for each topic after the handshaking managed by the only ROS master.

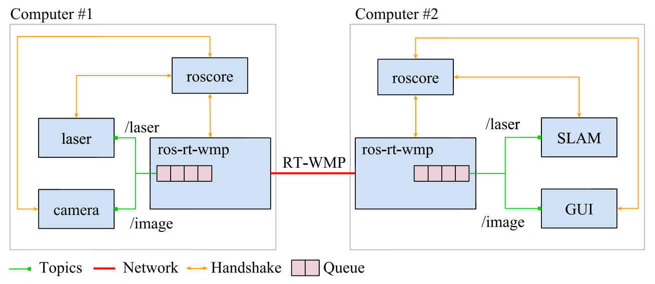


Fig. 7. A hypothetical 2-computers/2-topics system relying on *RT-WMP* nodes. Messages are split in MTU-sized packets (if necessary) and pushed in a common queue ordered accordingly to the priority of the topic that originated the messages and sent over the channel in this order using the *RT-WMP* protocol. Local connection are managed by the ROS masters present in each computer.

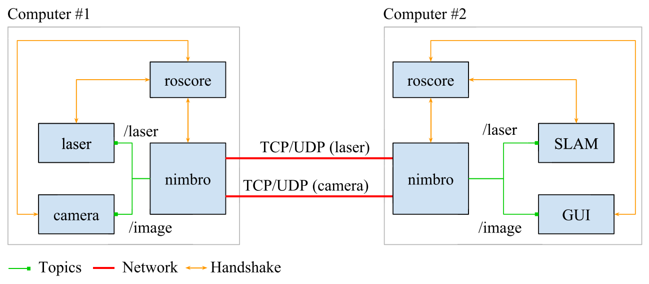


Fig. 6. A hypothetical 2-computers/2-topics system relying on *Nimbro* nodes. One TCP or UDP connection managed by the *Nimbro* nodes themselves is created for each topic. Local connection are managed by the ROS cores present in each computers.

4.1. *ROS standard communication protocols*

TCPROS is the standard transport layer for ROS Messages and Services. It uses standard TCP/IP sockets for transporting message data. Inbound connections are received via a TCP Server Socket with a header containing message data type and routing information.

The TCPROS method can be used with the *NoDelay* option. If this option is set, Nagle's algorithm is disabled. In this case, the jitter should be reduced at the expense of greater use of bandwidth. Finally, ROS allows the use of the *unreliable* option that allows connectionless (acknowledgment-less) between different ROS nodes. In this case, the protocol used is UDP, but as the name suggests, there are no guarantees of message delivery.

Fig. 5 shows the configuration that a hypothetical 2-computers, 2-topics (flows[10]) system would have when this method of connection is used. A single *roscore* manages the whole system and assists in the connection between the two nodes as described above. One TCP or UDP independent connection[11] is established for each flow between source and destination.

4.2. *Nimbro network*

This package, released by Bonn University is very flexible and offers several options. It has the possibility of *transporting* topics (with the publication frequency being modulated at the transmission side) and services using TCP and UDP protocols using an optional transparent BZip2 compression to speed up communication and an experimental Forward Error Correction technique (based on OpenFEC) for UDP unreliable transport. Additionally, it provides nodes and filters for transmitting the ROS log, the TF tree and H.264-compressed camera images. Finally, it includes an *rqt* plug-in for visualization and debugging of network issues.

Conceptually, the communication node subscribes to the publishing topic, serializes the messages, and send it to the receiver side where it is deserialized and published.

Nimbro allows easier configuration of the nodes and can be carried out by defining a simple configuration file which contains the settings such as the nodes to be transported, their maximum frequency, if compression should be used, the TCP/UDP port that the nodes should use, etc.

Again, Fig. 6 shows the configuration that a hypothetical 2-computers, 2-topics (flows) system would have when the above method of connection is used. This time two independent ROS masters are used: one for each computer; and the connection between the topics on different computers is performed by the *Nimbro* nodes themselves. However, like in the previous solution, an independent UDP/TCP connection is created for each topic to be transported.

In our set of experiments, we used the *udp_sender*, *udp_receiver*, *tcp_sender*, *tcp_receiver* nodes that, as their names indicate, use the UDP and TCP protocols respectively to transport the topics.

4.3. *ROS RT-WMP*

The *RT-WMP* ROS nodes use a similar technique as *Nimbro*, serializing the messages at the sender and recomposing them at the receiver side. However, they use the *RT-WMP* protocol [12] as medium access and transport layer. This means, among other things, that the messages larger that the MTU, is split by the protocol itself into different network packets and sent independently.

The *RT-WMP* is a token-based routing protocol that works with existing IEEE 802.11 networks. The protocol provides global static message priorities. Its target application is that of interconnecting a relatively small fixed-size group of mobile nodes, generally mobile robots (up to 32 units). It is based on a token-passing scheme and is designed to manage rapid topology changes through the exchange of a matrix containing the link quality among nodes. It works in three consecutive phases that repeat indefinitely: the priority arbitration phase (PAP), the authorization transmission phase (ATP), and the message transmission phase (MTP), together known as a loop.

More details can be found in [12].

Fig. 7 shows that the configuration in case of *RT-WMP* is, in the practice, similar to the *Nimbro* solution except that the packets/message are transported by the *RT-WMP* protocol instead of TCP/IP or UDP/IP.

[10] The terms *topic* and *flow* are used interchangeably in this paper.

[11] Although UDP is a connectionless protocol we use the term UDP connection to refer to a UDP flow between two ROS nodes.

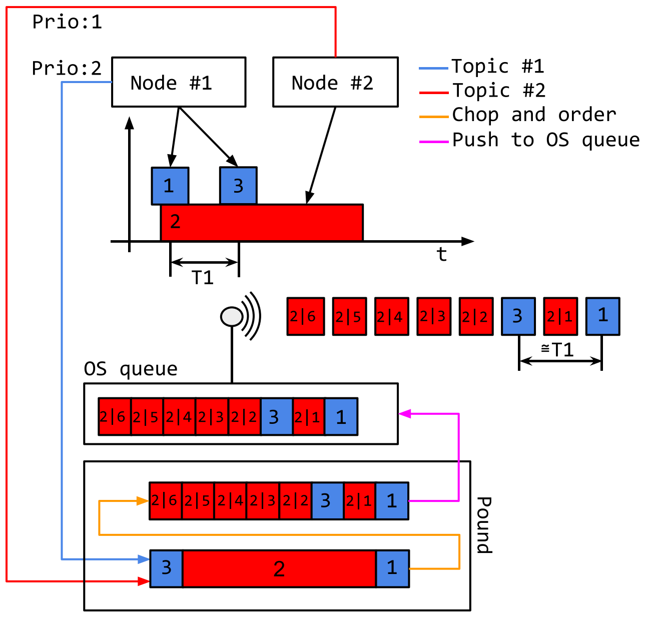


Fig. 8. *Pound* solution for competing flows with different level of importance. The messages enter the *Pound* node, are split in MTU-sized packets and these are sorted according to the priority of the topic that originated the messages. Then are pushed in the OS queue and later sent in the same order.

5. Proposed solution: *Pound*

In practical robot applications, each ROS flow (topic) has different latency and bandwidth requirements. Usually, these two requirements conflict with each other [37]. Recall from Section 3.5 that when two different type of flows compete for each other for the same wireless channel, the latency or jitter requirements of both the flows may not be met. For instance, a laser topic could need to be exchanged with less delay compared to an image topic. Hence, priorities in various flows become vital in meeting the latency requirements of important flows and thereby maintain stability in the whole system.

We found that none of the candidate solutions (or even any existing ROS solutions) consider priorities for different flows except the *RT-WMP* that, however, suffers from limitations due to the high overhead as will be shown. This means that all of them are affected by the problems originated by having multiple competing flows as explained in Section 3.5. To alleviate this problem, we propose a ROS node called *Pound* that implements priorities that can be modified online (assigned dynamically), can use compression and can modulate the frequency of the source topics. It offers fewer options to configure compared to *Nimbro*. Nevertheless, we believe this makes the *Pound* easy to configure and use. Also, it has the possibility of transporting the *tf* topic excluding the information that is not relevant to the destination node.

5.1. Rationale

In order to implement the priority scheme proposed, the *Pound* uses an intermediate transmission queue as shown in Fig. 8. Consider two different flow. Given that the messages will be transmitted in the order they enter the operating system (OS) transmission queue, the problem is especially serious if the lower-priority flows generate large messages as they can notably delay the higher-priority messages that enter the queue after them.

To alleviate this problem, we propose the introduction of an intermediate priority queue in which the messages are chopped into smaller parts (fitting in an MTU) which priority is that of the original message. In this way, the message the (parts of the) messages with higher priority will be pushed in the OS transmission queue before the others reducing the jitter of the most important flows.

The communication is performed over a single UDP flow: the *Pound* ROS node subscribes to the necessary topics, serializes the messages, split them in fragments that fit in a single radio packet and put them in a queue that is always maintained ordered according to the priority of the topic that generated each message. Then, it sends the packets following an order matching the priority of the source topics. On the other side, the messages are recomposed and published immediately.

We decide to use UDP as the transport protocol as it supports real-time constraints than TCP [38]. Although TCP offers reliable data delivery, it introduces a set of drawbacks when the timing of the data is equally or more important than the reliability itself.

On the one hand, the ACK messages, the retransmission of the lost data the window size dynamic adjustment and control packets in general use a considerable amount of bandwidth and can delay data packets. This situation is frequent in lossy Wireless networks. Also, as shown above, the Nagle's algorithm can introduce jitter especially if the data packet is small enough so that multiple of them fit in a single MTU. On the other hand, the fact that TCP protocol maintains (among other things) a window size, slows the re-establishment of the connection after a MAC layer disconnection (also frequent in wireless networks) which can again delay considerably important data packets and jeopardize their timing. Given that the data managed in a robotic system usually have a periodic nature (odometry, LiDAR readings, video streams, etc.) and have the common characteristics that are published with a high rate, the system can tolerate the sporadic loss of data.

UDP does not provide reliability but simplifies the delivery scheme avoiding the use of protocol management frames and allows better management of the packets delivery timing thus that many of the protocols that manage delay-sensitive data are built on top of the UDP itself. A distinguished example is RTP, which normally uses UDP to favor timeliness over reliability.

Internally, the topics' callback functions fragment the messages and push them in the transmission queue indicating the priority associated with the topic itself; a transmission and reception threads are in charge of popping the most priority message from the transmission queue and sending it and listening for packets, respectively. At the reception side, the messages are then reconstructed (when they are composed by multiple packets) and published. If some of the fragments are not received, the corresponding message is discarded. This means that there are not retransmissions at the transport layer which simplifies the scheme and avoids congestion. Notice that this behavior is, in our opinion, the adequate in a real-time system given that late information is usually useless.

As explained before, this solution is especially effective when dealing with topics that publish large messages given that small-sized higher priority messages do not need to wait for the transmission of larger messages having the opportunity to maintain their source period.

When transmitting, the *Pound* node takes into account the time needed by the network to propagate the messages to avoid the congestion of the network. For example, if the network rate is fixed at 6 Mbps, after sending a 1 KB packet, the transmission thread will be put to sleep during $1024 \cdot 8 \cdot 10^{-6}/6 = 1.3$ ms that corresponds roughly to the time needed to send the packet itself, before allowing it to pop the next packet from the *Pound* transmission queue. Of course, this value is not precise (also, the packet can be delayed before being transmitted if the channel is busy) but helps not to saturate the OS transmission queue and to reduce UDP discards. According to our experiments, this is particularly important when a large message is pushed into the

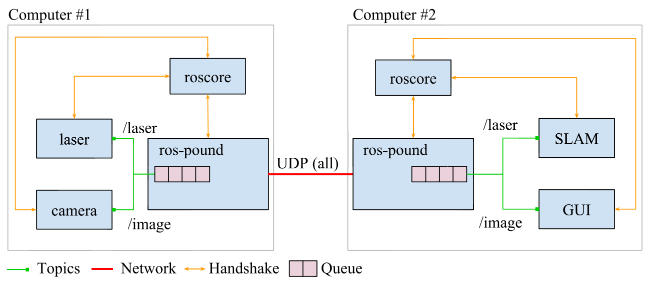


Fig. 9. A hypothetical 2-computers/2-topics system relying on *Pound* nodes. A single UDP connection is created between source and destination. Messages are split in MTU-sized packets (if necessary) and pushed in a common queue ordered accordingly to the priority of the topic that originated the messages and sent over the channel in this order. Local connections are managed by the ROS masters present in each computer.

queue because not doing so would imply a burst of consecutive calls to the socket *send()* function that can cause the overflow of the OS transmission queue and, in turn, the discard of (some of the) packets making the reconstruction of the messages impossible at the receiver side.

The *Pound* queue length can be limited to reduce network delay: when the queue is overflowing (which means that the bandwidth is not enough to allocate all the flows at the current rate), the node discards some of the messages starting from those coming from the fewer priority topics.

As shown in Fig. 9, this solution uses multiple roscores (masters) similar to *Nimbro*, one master for each computer involved in the system. However, the main differences concerning *Nimbro* are that: a single UDP connection is used between the source and the destination, and the messages are sent according to the priority of the topic which published them.

5.2. Global ordering vs. local ordering

The scheme proposed implements a node-wide priority ordering of the messages in contrast with network-wide priority ordering offered, for example, by RT-WMP. However, a global priority ordering or server-based scheduling is very expensive to achieve and (unless offline planning can be made in advance) a considerable amount of bandwidth is necessary. This becomes clear when the RT-WMP overhead is analyzed. What Pound does is *alleviating* this problem establishing a message priority ordering at node level without adding any overhead to the network. This improves notably the performance of different types of real-world distributed robotics systems like those used in teleoperated interventions or explorations, for example, where the most of the data travels from the robot to the base station as shown in the next section.

6. Experimental evaluation

As explained in the introduction, the connection of ROS nodes over a wireless network can introduce serious delays and especially jitter if different flows are competing for the same communication channel. Consider a search and rescue team of robots that are sent inside a tunnel to look for survivors as described in [39]. In this application, the network is composed by a set of robots that end up in a chain configuration and where the leader robot must send –possibly over a multi-hop path – sensor readings and camera images to the base station outside the tunnel while the joystick commands travel in the opposite direction.

In the most basic setup, we have at least two competing flows (video feedback and LIDAR readings) in one direction and one flow (robot control commands) in the opposite direction competing for the available bandwidth. All these flows, apart from using a different amount of bandwidth (the video feedback consumes more bandwidth than the other two), have different QoS requirements. If joystick commands arrive later than expected, for example, this can provoke a collision of the robot; similarly if the laser feedback suffers from a considerable delay, the human operator can send wrong commands with a similar outcome or with the undesirable consequence of system instability.

Additionally, excessive jitter in the propagation of commands/ feedback can make it difficult to close the control loop properly. However, in this example system, not all the flows have same importance: it could be possible to reduce the frame rate of the camera images without losing usefulness, while this is not possible with the other two flows, at least while the robot is moving.

This section is divided into three parts. Section 6.1 presents the test scenario inspired by the situation described above and has the goal of evaluating the local priority ordering, as defined in Section 5.2, of two competing flows. The experiments in Sections 6.2 and 6.3 have the goal of showing the general characteristics of the different methods in terms of bandwidth and resilience. Finally, the performance of the different methods in a control loop is analyzed in Section 6.4. Additionally in Section 6.5 a preliminary analysis of the influence of having more flows (in both directions) and the presence of additional hops is discussed.

Hardware and software setup: All the experiments have been carried out using 802.11 a/b/g/n network Wi-Fi cards configured to work in ad-hoc mode in a completely free channel in the 5 GHz band. The data rate was fixed at 6 Mbps to avoid automatic rate changes that could distort the measurements. This choice does not imply a loss of generality, given that (1) similar behaviors can be noticed with higher network rates and higher required bandwidths, and (2) in large real-world environments robotics system often have to work with the lowest available rate. Similarly, we also decided not to use frame compression (available in *Nimbro*, *Pound* and *RT-WMP*), noting that (1) sometimes compression does not reduce significantly the size of the messages (e.g., JPG images), (2) compressed frames can be as large as 64 KB or more, and importantly, (3) we want a fair comparison against standard ROS transport protocols. Each computer (network node) is a computer running Ubuntu 14.04 OS and uses ROS Indigo version.

6.1. Performance in presence of competing flows

As anticipated, in this experiment we considered the two feedback flows —*laser* and *image*— to test the systems with one high-importance low-bandwidth flow and one high-bandwidth lower-importance flow sent simultaneously. The *laser* flow has been configured to be composed in each run by 1000 messages of 1 KB each with a 20 ms period (50 Hz), while the *image* flow consists of 500 messages of 64 KB each with a 150 ms period (6.67 Hz). The flows were generated using a dedicated ROS node called *ros-profiler*.[12] capable of publishing and subscribing to multiple flows with different message sizes and frequency. With the methods that allowed it, we also tested different transport options for the two flows. Specifically, we tested a combination of UDP and NOD in the standard ROS protocol and UDP and TCP in the *Nimbro* network, which we refer as *Nimbro**, for the *laser* and *image* flows respectively.

To summarize, the methods analyzed in this paper and their identifiers are:

- *TCP*: Standard TCPROS protocol
- *NOD*: Standard TCPROS protocol with No_Delay option (Nagle's algorithm is off)

12 https://github.com/dantard/unizar-profiling-ros-pkg/.

- *UDP*: Standard UDPROS protocol
- *UDP/NOD*: Standard ROS protocol in which the *laser* flow was transported using ROSUDP protocol and the *image* flow using the ROSTCP protocol
- *Nimbro*: *nimbro_network* nodes in which both flows were transported using UDP protocol, *udp_sender* and *udp_receiver*
- *Nimbro**: *nimbro_network* nodes in which the *laser* flow was transported using UDP protocol and the *image* flow using TCP protocol, thus *udp_sender*, *udp_receiver* and *tcp_sender*, *tcp_receiver* respectively
- *Pound*: *pound* nodes in which all the flows were transported using a single UDP connection. The *laser* was assigned a higher priority than the *image* flow
- *RT-WMP*: *ros-rt-wmp* nodes in which all the flows were transported using the *RT-WMP* protocol. The *laser* was assigned a higher priority than the *image* flow

Two different set of experiments have been performed to consider a 1-*hop* and 2-*hops* configuration (i.e., 2-computer and 3-computer chain networks respectively). In the first case, the communication was peer-to-peer (Computer #1 $\longrightarrow$ Computer #2), while in the second case, we used a repeater configuring a standard IP forwarding and a fixed routing (Computer #1 $\longrightarrow$ Computer R $\longrightarrow$ Computer #2). In case of the *RT-WMP*, the routing was also fixed providing the nodes with a fake LQM corresponding to the desired topology.

The experiments had the goal of analyzing the jitter and delay of the message periods in both flows. The jitter was measured by calculating the inter-arrival delay at the receiver side or, in other words, the time span between the publication of a message and the subsequent one, which corresponds with the period at the destination node.

To measure the delay we used the following technique: the messages are time-stamped at the sender side before being transmitted. When the destination node receives the message, it sends back to the sender a frame containing such time-stamp through a dedicated Ethernet connection; at that moment the sender computes the elapsed time. The values obtained in this way are not absolute given that they are affected by the delay introduced by the backward communication. However, this delay is practically constant and usually below 0.25 ms at the same time that it affects all the measurements similarly, thus allowing a fair comparison. Also, this avoids the problem of synchronizing computers clock which is a tedious and sometimes imprecise task due to clocks drift.

Additionally, we recorded two parameters that relate to the efficiency of the network connection. They are (1) *Message Delivery Ratio (MDR)*: the ratio between messages successfully received and the total ROS messages sent over the network, and (2) utilized bandwidth (throughput) of the network.

6.1.1. *One hop experiment*

In this experiment, we used only two computers, one acting as the control station (source) and the other acting as the robot (receiver).

Jitter. Ideally, the recorded period values on the receiver should coincide with the message period. For instance, the *laser* flow has a period of 20 ms. Hence, the period at the receiver side should be the same. Fig. 10 shows the jitter period distribution for the different solutions analyzed for the laser flow. As expected, the *RT-WMP* and *Pound* obtain the sharpest distributions around the real period, followed by the NOD and the UDP/NOD. The TCP connection shows different peaks as in the experiment shown in Section 3.4 around the real period, the double of the period (due to Nagle's algorithm) while *Nimbro* presents a strange behavior with many messages published close to each other and other suffering from a jitter higher than 0.1 s. A similar thing happens with the UDP connection. The *Nimbro** behaves somehow better than the simple *Nimbro* but also shows different peaks. Notice that in this case the bandwidth required to allocate both flows is about 3.8 Mbps, below the theoretical limit of 802.11g when the rate is fixed at 6 Mbps.

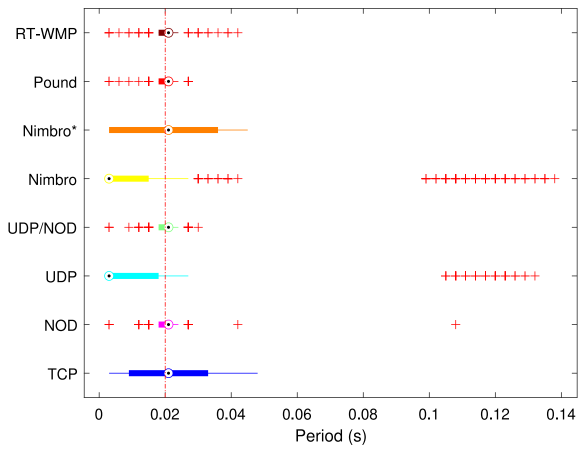


Fig. 10. A boxplot showing a compact representation of the *Jitter* measurements (represented as variation of the period at the destination node) of every protocol is shown for the *laser* topic in the 1-*hop* experiment. The vertical (red) line shows the ideal value (20 ms).

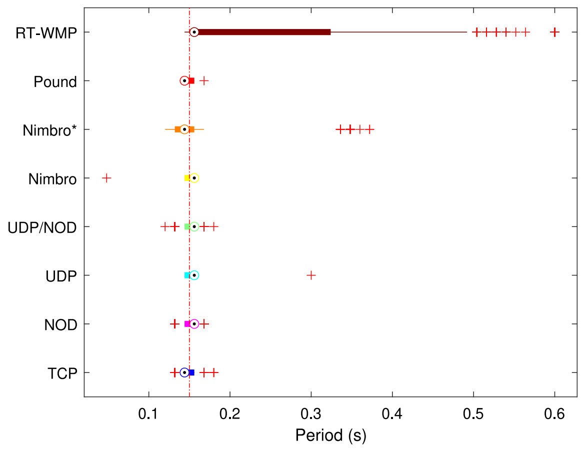


Fig. 11. A boxplot showing a compact representation of the *Jitter* measurements (represented as variation of the period at the destination node) of every protocol is shown for the *image* topic in the 1-*hop* experiment. The vertical (red) line shows the ideal value (150 ms).

Fig. 11 shows the jitter corresponding to the image flow. This time, all the solutions show a distribution around the expected period. This is because this flow's messages can only be delayed by just *one* message of the other flow. In this case, thus, *Pound*, *Nimbro*, and UDP offer very similar and narrow distributions while NOD and TCP have similar behavior, thanks to the fact that this time the messages of this flow are large enough not to need the Nagle's algorithm. Further, the priority assigned to the message in *RT-WMP* favors the *laser* flow as expected. However, *RT-WMP* comprises a 3-phase delivering algorithm, which has the highest overhead compared to other methods. The result is that *image* messages are delayed and sometimes discarded if the offered bandwidth in inadequate to allocate both flows provoking the second distribution over and around the double of the period.

Delay. Fig. 12 shows the corresponding delay distributions for the *laser* flow. UDP and *Nimbro* show a quite wide delay distribution (that is also reflected on the jitter at the receiver side, in fact) while NOD, *Pound* and *RT-WMP* show a very constant delay below 10 ms.

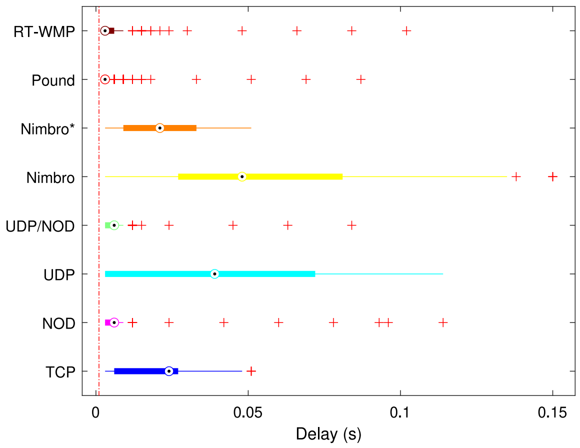


Fig. 12. A boxplot showing a compact representation of the *Delay* measurements of every protocol is shown for the *laser* topic in the 1-*hop* experiment. Ideally, the delay should be close to 0 ms.

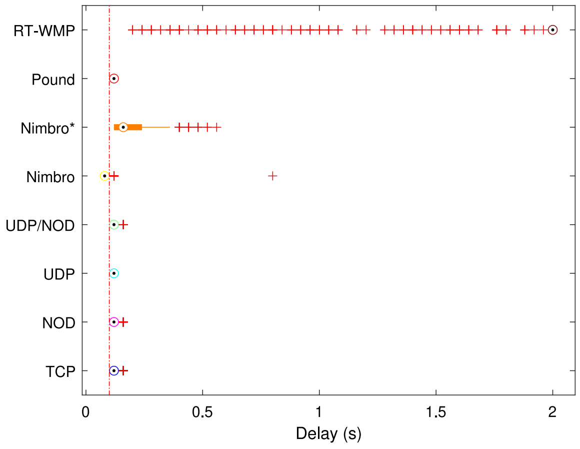


Fig. 13. A boxplot showing a compact representation of the *Delay* measurements for the *image* topic in the 1-*hop* experiment. Ideally, the delay should be close to 100 ms, because of the message size.

Also, UDP/NOD performs well, while *Nimbro** suffers from a delay that sometimes reaches 50 ms. TCP, again due to Nagle's algorithm, show two different distributions with the second being around 25 ms. Also, the graph reports a very small delay for an *RT-WMP* solution. This is because in this configuration (i.e., with only two nodes on the network) each message transmission involves the exchange of 3 frames at most (usually two frames will be enough).

Fig. 13 shows the corresponding delay distributions for the *image* flow. In this case, the most of the flows shows a delay between 100 ms and 130 ms except for *Nimbro** and *RT-WMP*. The former shows a quite spread distribution with a mean of about 200 ms because this flow, which uses the TCP transport, had to adapt to the bandwidth left free by the UDP flow. A similar situation happens with UDP/NOD but, surprisingly, this method shows a much better behavior. The *RT-WMP*, for its part, pays the limited bandwidth it can offer: the messages that are not discarded at the server side, are enqueued in its transmission queue and delivered with a delay that exceeds 4.5 s.

Tables 1 and 2 show the corresponding numerical values of the mean period at the destination J_d with its standard deviation $std(J_d)$, the mean delay D with its standard deviation $std(D)$, the message delivery ratio *MDR*, and the Bandwidth consumed by the flow *BW*. All the mean jitter values are around 20 ms but with great variability regarding standard deviation. Something similar

Table 1
Results for 1-*hop* network, *laser* topic.

	J_d (ms)	$std(J_d)$ (ms)	D (ms)	$std(D)$ (ms)	*MDR* (%)	*BW* (Mbps)
TCP	20.1	13.3	18.6	11.6	99.3	0.398
NOD	20.1	4.2	6.0	6.5	99.4	0.398
UDP	20.0	38.2	42.5	34.8	99.9	0.400
UDP/NOD	20.0	3.2	5.8	4.0	99.8	0.399
Nimbro	20.0	38.1	55.8	46.0	99.8	0.400
*Nimbro**	20.1	16.0	22.2	12.8	99.3	0.397
Pound	20.0	2.0	4.1	4.0	99.8	0.400
RT-WMP	20.1	2.8	4.7	5.0	99.6	0.399

Table 2
Results for 1-*hop* network, *image* topic.

	J_d (ms)	$std(J_d)$ (ms)	D (ms)	$std(D)$ (ms)	*MDR* (%)	*BW* (Mbps)
TCP	150.1	7.7	132.3	5.9	99.9	3.418
NOD	150.1	7.7	132.7	5.6	99.9	3.418
UDP	150.4	7.2	106.3	1.9	99.7	3.410
UDP/NOD	150.1	7.4	129.1	5.2	99.9	3.417
Nimbro	149.9	5.3	100.3	31.5	100.0	3.421
*Nimbro**	150.2	44.7	194.9	86.2	99.9	3.416
Pound	150.2	2.9	112.0	1.9	99.9	3.417
RT-WMP	233.6	141.3	4635.9	2738.1	64.2	2.196

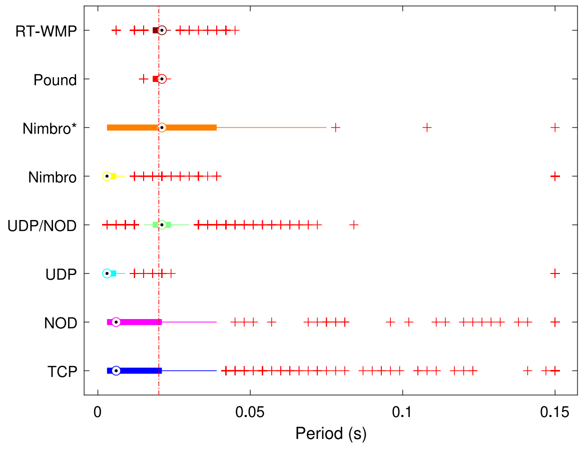


Fig. 14. A boxplot showing a compact representation of the *Jitter* measurements (represented as variation of the period at the destination node) of every protocol is shown for the *laser* topic in the 2-*hops* experiment. The vertical (red) line shows the ideal value (20 ms).

happens with the delay, with a mean that is in all the cases below 60 ms but having a standard deviation that varies of up two orders of magnitude. Nevertheless, all MDR are close to 100%, and all the methods consume, as expected, a similar amount of bandwidth, except for the *RT-WMP*, which discarded a noticeable amount of messages due to its inability of dealing with the required bandwidth.

6.1.2. Two hops experiment

This experiment is similar to the previous, except that we use an additional computer in between the source and receiver nodes to relay the information via a wireless network. The relay computer does the task of message forwarding between the source and receiver nodes. So, it is in principle, a two-hop network (i.e., a network with 3 computers statically configured to form a chain). In this case, the available bandwidth is virtually half compared to the previous experiment. For this reason, the two flows were configured with periods of 20 ms (*laser*) and 300 ms (*image*) respectively, which correspond to a load slightly below the

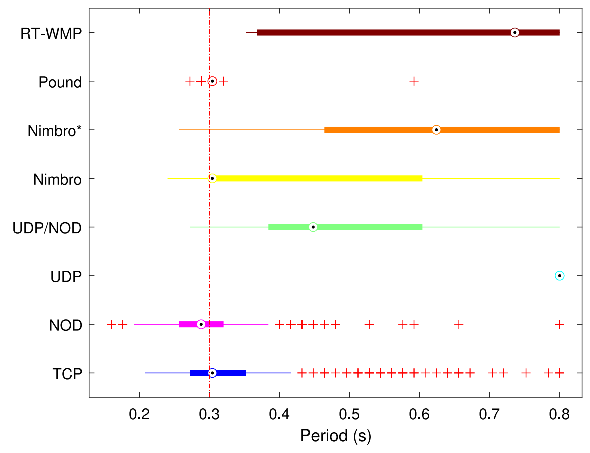


Fig. 15. A boxplot showing a compact representation of the *Jitter* measurements (represented as variation of the period at the destination node) of every protocol is shown for the *image* topic in the 2-*hops* experiment. The vertical (red) line shows the ideal value (300 ms). Note, several *RT-WMP* and *Nimbro* messages showed a period ≥ 0.6 s on an average.

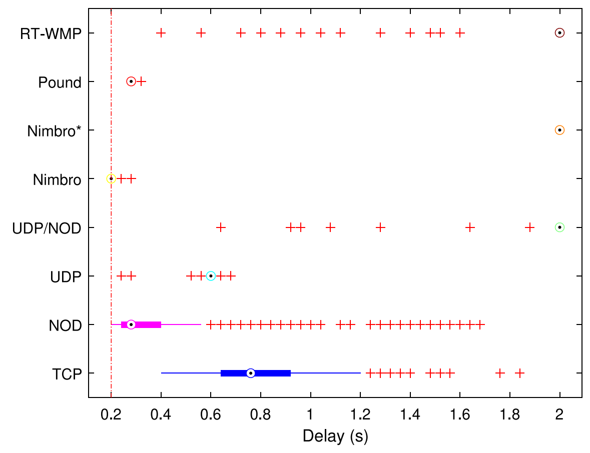


Fig. 17. A boxplot showing a compact representation of the *Delay* measurements of every protocol is shown for the *image* topic in the 2-*hops* experiment. Ideally, the delay should be close to 200 ms, because of the message size. Note the data is saturated to 2 s only for compact display purpose.

Table 3
Results for 2-*hops* network, *laser* topic.

	J_d (ms)	$std(J_d)$ (ms)	D (ms)	$std(D)$ (ms)	MDR (%)	BW (Mbps)
TCP	257.1	2870.8	3127.6	10243.4	7.8	0.031
NOD	56.0	787.3	1201.7	3869.0	35.7	0.143
UDP	20.1	61.7	128.5	74.6	99.5	0.398
UDP/NOD	21.5	11.7	13.0	10.4	92.9	0.372
Nimbro	21.6	60.0	113.1	66.8	92.4	0.370
*Nimbro**	21.4	489.5	41.9	345.0	93.6	0.375
Pound	20.1	1.7	6.8	1.4	99.4	0.398
RT-WMP	20.3	4.2	8.1	4.5	98.3	0.394

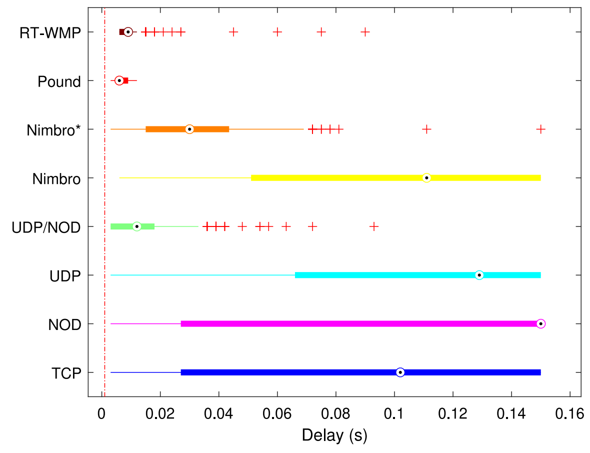


Fig. 16. A boxplot showing a compact representation of the *Delay* measurements of every protocol is shown for the *laser* topic in the 2-*hops* experiment. Ideally, the delay should be close to 0 ms.

Table 4
Results for 2-*hops* network, *image* topic.

	J_d (ms)	$std(J_d)$ (ms)	D (ms)	$std(D)$ (ms)	MDR (%)	BW (Mbps)
TCP	301.0	68.9	303.6	95.8	99.7	1.704
NOD	300.0	88.2	431.6	347.6	100.0	1.710
UDP	6908.4	9351.2	506.7	160.1	4.3	0.085
UDP/NOD	512.6	207.0	27218.3	7530.2	58.5	1.001
Nimbro	486.6	340.5	209.1	13.7	61.7	1.054
*Nimbro**	788.8	865.1	6618.9	869.1	38.0	0.650
Pound	300.7	13.6	294.2	2.8	99.8	1.706
RT-WMP	1108.2	1153.9	14479.1	4484.5	27.1	0.463

theoretical bandwidth offered by the wireless configuration used. Note the decrease in the message frequency of the *image* flow.

The introduction of a relay node significantly changes the situation. On the one hand, the probabilities of a transmission error and UDP (silent) discard grow considerably over a 2-hop path. On the other hand, TCP or NOD protocols can congest the network while retransmitting lost packets.

Jitter. The problems mentioned above introduced by a relay node are reflected in Fig. 14: all the distributions are wider than before but additionally all the IP-based methods except *Pound*, and somehow UDP/NOD combination (to a much lesser extent, though) are incapable of maintaining a sharp distribution around the period. This last combination (UDP/NOD) takes advantage of the fact that the TCP (NOD) protocol is designed to adapt itself to the available bandwidth left by the UDP flow (which is thus favored over the NOD flow). However, what surprises the most is that the MDR for TCP and NOD are below 10% and 40% respectively, despite being reliable protocols, showing a behavior in which the messages were delivered in bursts. The *RT-WMP*, meanwhile, shows two sharp distributions close to the expected period. This is because in a 2-*hops* networks the loops can vary more, and the delivery of a message can take the transmission of 6 to 9 frames.

Regarding the *image* flow, it can be seen in Fig. 15 that only the *Pound* shows a sharp distribution. *Nimbro*, NOD and TCP also show a fairly good performance. The methods with UDP, UDP/NOD, and *Nimbro** have been able to deliver only 4.91%, 58.5% and 38% of the frames in this experiment, respectively. The last two methods pay for the fact that the *image* flow uses TCP with a bandwidth limited by the UDP flow, which does not have any congestion detection method. *RT-WMP*, on the other hand, pays for its higher overhead and the fact of favoring the high priority flow being forced to discard about 73% of the image frames. This is reflected in the period on the destination node that presents a distribution at approximately 375 ms and spread values beyond 800 ms (not visible in the figure).

Delay. In terms of delay for the *laser* flow (see Fig. 16), *Pound* performs better (having messages delayed by 6.8 ms on average) together with *RT-WMP* with a delay below 10 ms. The measurement of the delay for TCP and NOD show a dual behavior with messages delivered within a few milliseconds and other delayed several seconds. The UDP/NOD also perform well with a mean

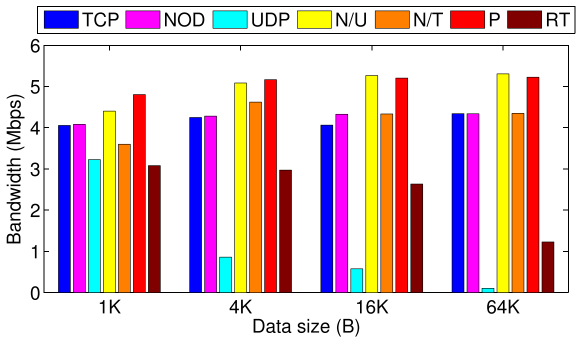


Fig. 18. Bandwidth offered by the different methods. *N/U* stands for *Nimbro/UDP*, *N/T* for *Nimbro/TCP*, *P* for *Pound* and *RT* for *RT-WMP*.

delay of 13 ms. Nimbro* behave similarly but with higher delay and standard deviation. UDP and Nimbro instead show a quite delay spread with a mean approx. 120 ms.

The situation is quite different in the *image* flow (Fig. 17), where *Nimbro* shows the best behavior. Both *Pound* and NOD show a mean delay below 300 ms, but *Pound* has a much lower deviation. UDP and TCP show an average performance, whereas UDP/NOD and *Nimbro** respectively show a significant delay of more than 6 s and 27 s (not visible in the figure). Further, *RT-WMP* has the highest delay because it has the highest overhead.

Tables 3 and 4 show corresponding numerical values, similar to the previous 1-*hop* experiment.

6.2. Bandwidth

Fig. 18 shows the user-bandwidth offered by the methods evaluated in a 1-*hop* network configured in the same way as the previous experiments. This has been computed considering saturated traffic at the sender side (of ≈6.5 Mbps) and for different sizes of the messages. This time a single flow was transported. The results show that the *Nimbro/UDP* and *Pound* methods provide the best performance with similar and constant bandwidths, above 5 Mbps for all the different message sizes. TCP, NOD, and *Nimbro/TCP* also offer good (and approximately constant) results close to 4 Mbps; the difference is due to the extra overhead that the TCP protocol has compared to UDP. Besides, UDP shows a performance that clearly decreases with the size of the message due to the growing amount of discarded packets that cause an almost vanishing bandwidth for messages of 64 KB.

Finally, *RT-WMP* provides a decreasing bandwidth: large messages needed the exchange of a higher number of packets and given that for each message the protocol executes a 3-phase consensus algorithm, the overhead grows considerably with the size of the message to be sent limiting the available user bandwidth. It is worth remarking, however, that a higher raw bandwidth does not necessarily mean a better result if it is not provided guaranteeing a limited amount of jitter and delay, especially in a system with tight control loops.

6.3. Resilience

As anticipated earlier, another important aspect to take into account in robotics systems is the resilience against unexpected situations that can take place, for example, in search and rescue scenarios. To check one of the many aspects involved in these situations, we performed a simple experiment in which the intermittent operation of a network node is simulated. We set up, in the usual 1-*hop* and 2-*hops* chain configuration, a scenario in which two flows, both with 20 ms period and 1 KB payload, were transmitted

Table 5
Reconnection time after node failure (s).

	1-*hop*		2-*hops*	
	Flow 1	Flow 2	Flow 1	Flow 2
TCP	1.73	1.93	2.19	2.28
NOD	1.96	1.96	2.46	2.32
UDP	1.29	1.28	1.47	1.46
UDP/NOD	1.28	1.92	1.46	2.32
Nimbro	1.31	1.30	1.46	1.46
*Nimbro**	1.32	1.68	1.64	2.99
Pound	1.31	1.30	1.53	1.52
RT-WMP	0.42	0.54	0.50	0.49

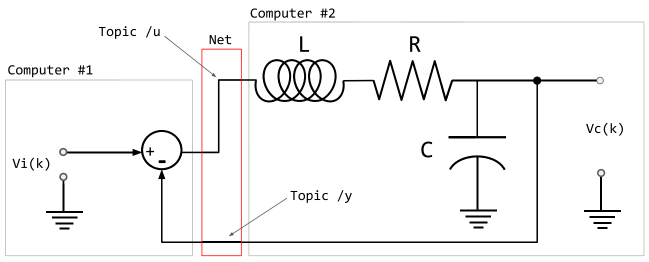


Fig. 19. RLC control system used in the experiment.

between the head and the tail of the chain. Then we simulated the malfunctioning of the sender node (in the 1-*hop* experiment) or the relay node (in the 2-*hops* experiment) bringing down the interface for exactly 5 s and then bringing it up again. The goal was to measure the time the different methods take to reestablish the flows of data after this issue. Notice that this includes the time needed by the wireless cards to rejoin the ad-hoc network. The experiment is thus not absolute but must be interpreted as a basic comparison among the different methods.

The results are shown in Table 5. From the data, it can be concluded that, as expected, the methods based on UDP are about half a second faster than that of TCP. This is because TCP has to renegotiate the connection and that there is a difference of around 300 ms between the 1-*hop* and 2-*hops* systems. Further, the *RT-WMP*, which does not use the IP protocol but raw 802.11 frames, shows a much faster behavior performing the switch in about 0.5 s.

6.4. Network in the closed control loop

The last test investigated the capabilities of the methods to deal with a real control loop in presence of another *perturbing* flow. We set up a system like the one shown in Fig. 19 implementing the (corresponding discrete-time) system itself in a computer and the (corresponding discrete-time) controller (in this case just a P controller with $K = 1$) in another computer. The two parts were connected through two ROS topics (namely u for the control signal and y for the feedback) transported from a computer to another with some of the methods being analyzed in this paper. The values were fixed as $C = 0.1$, $L = 0.1$ and $R = 1$ to obtain an under-damped system with a transient response of about 2 s and an overshoot of about 40%. The period was fixed in $T = 20$ ms and $Vi(k)$ =1 V. The u and y flows (about 4 Kbps each) were perturbed by another flow from *Computer #2* to *Computer #1* having period of 200 ms and message-size of 64 KB (2.5 Mbps, approx). In case of *Pound* and *RT-WMP*, u and y were given higher priorities than the perturbing flow.

The results of $Vc(k)$ over the time are shown in Fig. 20. Comparing them with the locally executed simulation, the *Pound* and especially the *RT-WMP* are the only methods capable of supporting control performance close to the local case while the other methods fail more or less noticeably due to jitter and delay that

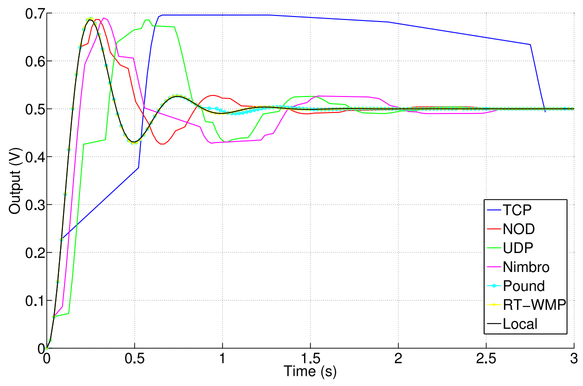


Fig. 20. Output of the simple RLC system in a feedback control loop.

the messages –especially those of the *y* flow that must compete with the perturbing flow – suffer in their network trip. It worth noting that the (this-time-small) differences between *RT-WMP* and *Pound* stem from the fact that *RT-WMP* provides a system of *global* priorities whereas *Pound* only establish a priority between flows that originate in the same computer. This means that in case of *Pound* the *u* flow messages must compete with the perturbing flow's messages to access the medium, while with *RT-WMP* their access is always given higher priority thanks to the protocol's MAC scheme. Notice that when the experiment was conducted without perturbing flow, all the methods offered the same correct results, making the output indistinguishable from the locally executed.

6.5. More flows and more hops

We also performed preliminary experiments with multiple flows in different directions and with multiple network nodes in a chain configuration that is multiple hops. The results confirm, as foreseeable, those in the configuration proposed in this paper.

On the one hand, additional hops cause a noticeable reduction of the available bandwidth. However, even when the required bandwidth is below the limit, the decreased end-to-end packet delivery ratio, the poor performance of TCP protocols in lossy networks and the congestion due to the lack of mechanisms for managing time-sensitive data or burst transmissions, make the delay and jitter increase noticeably at the same time the MDR decreases rapidly. On the other hand, the presence of multiple flows in the same direction worsens the situation not only due to the increased required bandwidth but also because the flows interfere with each other as shown by the experiments. In both scenarios, the *Pound* offers solutions to mitigate this effects by introducing priorities and taking into account the time needed to send the packets over the network.

The introduction of flows in the opposite direction (a forward flow from the base station to a teleoperated robot in the scenario proposed in this paper, for example) indirectly perturbs all the others since it competes with them to access the medium –thus reduces the available bandwidth – and can delay the delivery of more important or more priority messages. However, since the access to the medium is resolved on a per-packet basis, the influence is usually assumable. Even so, the use of a network-wide priority mechanism, as *RT-WMP* does, improves the global behavior as shown in the experiment in Section 6.4 at the expense of additional overhead and thus results in reduced throughput.

7. Discussion

Through extensive experiments, we demonstrated that the standard ROS behavior, which uses the TCP protocol by default to connect topics of different nodes, is not completely adequate for wireless networks. The other options, like NOD and UDP, offered better results, which however degrade when multiple flows compete to gain access to the medium. Mixed solutions, that is those in which UDP is used in conjunction with TCP flows, had shown a better behavior regarding jitter and delay for UDP flows but degrade the latter flow that adapts to the bandwidth left by the former flow. This scheme could be taken into account for the scenarios considered in this paper, but similar issues to those shown with single-type transport would reappear if a system needs three or more flows in the same direction. Also, the experiments showed that the results sometimes depend on how the TCP (and NOD) flows adapt to the network and how they coexist with others.

Moreover, the fact of having a single ROS master that acts as topics' name resolution system, increase the overhead, reducing the available bandwidth of the network, besides limiting its flexibility. The solutions based on multiple ROS masters such as *Nimbro*, *Pound* and *RT-WMP* mitigate these problems, but the results show that the *Nimbro* produced similar results as the standard ROS solutions regarding jitter, both for single-type (UDP/UDP) and mixed-type (UDP/TCP) solutions. This is because one connection is created for each flow with the same problems regarding coexistence and perturbation that they inflict on each other. This suggests the use of a unique connection mechanism for transporting all the different flows, which would improve the results. The results also highlight the need for a system of priorities for topics with different requirements and importance. This is the solution that both *Pound* and *RT-WMP* propose –the former in a per-node basis and the latter in global terms – showing better results, especially the jitter of the higher priority flow.

The improvement is even more clear in 2-*hops* network experiments, where standard solutions resulted in high delay and jitter, due to the congestion that the network suffers. Especially, TCP-based methods, which has been designed to work in non-lossy networks, show huge message loss rates that make their use impractical in real-time systems in which a control loop is involved, both alone and in conjunction with UDP flows. This shows that using ROS over a multi-hop wireless link is likely to offer unsatisfactory results. Also, *Nimbro* that performed reasonably well in 1-*hop* network, suffered from UDP silent discards due to the congestion of the network either at the sender side or in the relay node. According to our experiments, this is due to packets (in which large messages are split into) being queued in the Linux network layer in bursts, regardless of whether the queue has enough free space or not. As explained earlier, to avoid this, *Pound* introduced a delay after any socket *send()* call, that in principle left sufficient time for the packet to be sent over the network.

In summary, the proposed solution, *Pound*, outperformed all the other methods including *RT-WMP*, which paid a higher overhead and was incapable of maintaining a proper rate for the *image* flow. This is also confirmed by the bandwidth experiment results.

Besides, the control loop experiment confirmed the results regarding jitter and delay, showing that only the *Pound* and *RT-WMP* methods were capable of properly closing the control loop and delivered the desired response rate. The other methods, which complied with the control requisites when tested without the perturbing flow, failed more or less noticeably in the experimented scenario.

Finally, it should be noted that Pound is fully based on the UDP protocol, which does not offer a guarantee of packet delivery. This could be a drawback in scenarios where the delivery of the packets must be guaranteed. However, our experiments have shown that most of the UDP silent discards took place when the transmission queue is full. Therefore, proper management of the timing can help mitigate this issue and achieve negligible packet loss rates. Pound takes a step in this direction by introducing a short delay after each transmission to de-congest the transmission queue. This resulted in performance improvement and drastically reduced the packet losses. Also, we believe that it is often preferable to the behavior offered by UDP over TCP, especially in distributed multi-robot networks.

8. Conclusions

In this paper, we presented a comparison of different solutions for using the Robotics Operating System (ROS) platform in wireless distributed systems. Standard ROS communication protocols are not optimized for wireless communication where aspects like reduced bandwidth (especially in multi-hop networks) and increased delay and jitter must be taken into account.

First, an analysis of the limitations of ROS in this context was presented, showing that the default choice for connecting topics can suffer from irregular delays and jitters. Then, an analysis of the performance in the presence of multiple flows (ROS topics) with different sizes and periods was performed showing that flows can interfere with each others timing making the implementation of tight control loops difficult. Further, we discussed various candidate multi-master solutions for ROS, and we proposed a new multi-master solution (called *Pound*) that exploits flow priorities to meet the bandwidth and delay requirements of the system. We compared all the solutions including the standard ROS supported protocols with and without relay nodes in static wireless networks.

The results show that flow prioritization is necessary if the reduction of jitter is required, mainly in multi-hop networks. The paper aims to be a reference for those implementing wireless multi-robot and distributed systems, especially involving tight control loops.

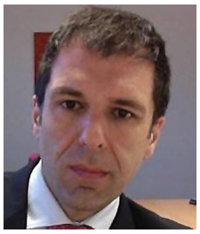
Acknowledgments

This work has been funded by the projects ESTER, Spain (CUD 2017-18) and ROBOCHALLENGE, Spain (DPI2016-76676-R) and programa Ibercaja-CAI de estancias de investigación.

References

[1] F. Furrer, M. Burri, M. Achtelik, R. Siegwart, Robot Operating System (ROS): The Complete Reference (Volume 1), Cham: Springer International Publishing, 2016, pp. 595–625.
[2] R. Bischoff, U. Huggenberger, E. Prassler, KUKA youBot - a mobile manipulator for research and education, in: 2011 IEEE International Conference on Robotics and Automation, ICRA, 2011, http://dx.doi.org/10.1109/ICRA.2011.5980575.
[3] J. Gregory, J. Fink, E. Stump, J. Twigg, J. Rogers, D. Baran, N. Fung, S. Young, Application of multi-robot systems to disaster-relief scenarios with limited communication, in: Field and Service Robotics, Springer, 2016, pp. 639–653.
[4] S.-G. Chitic, J. Ponge, O. Simonin, Are middlewares ready for multi-robots systems? in: International Conference on Simulation, Modeling, and Programming for Autonomous Robots, Springer, 2014, pp. 279–290.
[5] N. Mahdoui, E. Natalizio, V. Fremont, Multi-uavs network communication study for distributed visual simultaneous localization and mapping, in: 2016 International Conference on Computing, Networking and Communications (ICNC), 2016, pp. 1–5, http://dx.doi.org/10.1109/ICCNC.2016.7440564.
[6] D. Tardioli, D. Sicignano, L. Riazuelo, A. Romeo, J.L. Villarroel, L. Montano, Robot teams for intervention in confined and structured environments, J. Field Robot. 33 (6) (2016) 765–801.
[7] D. Tardioli, A proof-of-concept application of multi-hop robot teleoperation with online map building, in: Industrial Embedded Systems (SIES), 2014 9th IEEE International Symposium on, IEEE, 2014, pp. 210–217.
[8] D. Sicignano, D. Tardioli, S. Cabrero, J.L. Villarroel, Real-time wireless multi-hop protocol in underground voice communication, Ad Hoc Netw. (2011).
[9] P.U. Lima, A. Ahmad, A. Dias, A.G. Conceição, A.P. Moreira, E. Silva, L. Almeida, L. Oliveira, T.P. Nascimento, Formation control driven by cooperative object tracking, Robot. Auton. Syst. 63 (2015) 68–79.
[10] A. Gasparri, L. Sabattini, G. Ulivi, Bounded control law for global connectivity maintenance in cooperative multirobot systems, IEEE Trans. Robot. 33 (3) (2017) 700–717.
[11] M. Schwarz, M. Beul, D. Droeschel, S. Schüller, A.S. Periyasamy, C. Lenz, M. Schreiber, S. Behnke, Supervised autonomy for exploration and mobile manipulation in rough terrain with a centaur-like robot, Front. Robot. AI 3 (2016) 57, http://dx.doi.org/10.3389/frobt.2016.00057.
[12] D. Tardioli, D. Sicignano, J.L. Villarroel, A wireless multi-hop protocol for real-time applications, Comput. Commun. 55 (2015) 4–21.
[13] Y. Tian, K. Xu, N. Ansari, TCP in wireless environments: problems and solutions, IEEE Commun. Mag. 43 (3) (2005) S27–S32.
[14] L. Deek, E. Garcia-Villegas, E. Belding, S.J. Lee, K. Almeroth, Intelligent channel bonding in 802.11 n WLANs, IEEE Trans. Mobile Comput. 13 (6) (2014) 1242–1255.
[15] C. Paasch, O. Bonaventure, Multipath tcp, Commun. ACM 57 (4) (2014) 51–57.
[16] G. Patti, G. Alderisi, L.L. Bello, SchedWiFi: An innovative approach to support scheduled traffic in ad-hoc industrial IEEE 802.11 networks, in: IEEE International Conference on Emerging Technologies & Factory Automation (ETFA), 20th ed., 2015, Luxembourg.
[17] A. Steinfeld, T. Fong, D. Kaber, M. Lewis, J. Scholtz, A. Schultz, M. Goodrich, Common metrics for human-robot interaction, in: Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction, ACM, 2006, pp. 33–40.
[18] A. Owen-Hill, F. Suárez-Ruiz, M. Ferre, R. Aracil, in: A.M. Armada, A. Sanfeliu, M. Ferre (Eds.), ROBOT2013: First Iberian Robotics Conference: Advances in Robotics, Vol. 2, Springer International Publishing, Cham, 2014, pp. 555–568.
[19] J. Storms, K. Chen, D. Tilbury, A shared control method for obstacle avoidance with mobile robots and its interaction with communication delay, Int. J. Robot. Res. 36 (5–7) (2017) 820–839, http://dx.doi.org/10.1177/0278364917693690.
[20] P. Haller, L. Márton, Z. Szántó, T. Vajda, in: L. Busoniu, L. Tamás (Eds.), Handling Uncertainty and Networked Structure in Robot Control, Springer International Publishing, Cham, 2015, pp. 291–311.
[21] A. Birk, S. Schwertfege, K. Pathak, A networking framework for teleoperation in safety, security, and rescue robotics, Wireless Commun. IEEE 16 (1) (2009) 6–13.
[22] A. Owen-Hill, R. Parasuraman, M. Ferre, Haptic teleoperation of mobile robots for augmentation of operator perception in environments with low-wireless signal, in: Safety, Security, and Rescue Robotics (SSRR), 2013 IEEE International Symposium on, IEEE, 2013.
[23] S. Caccamo, R. Parasuraman, F. Baberg, P. Ogren, Extending a ugv teleoperation flc interface with wireless network connectivity information, in: Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on, IEEE, 2015, pp. 4305–4312.
[24] R. Parasuraman, T. Fabry, K. Kershaw, M. Ferre, Spatial sampling methods for improved communication for wireless relay robots, in: Connected Vehicles and Expo (ICCVE), 2013 International Conference on, IEEE, 2013, pp. 874–880.
[25] B.-C. Min, R. Parasuraman, S. Lee, J.-W. Jung, E.T. Matson, A directional antenna based leader–follower relay system for end-to-end robot communications, Robot. Auton. Syst. 101 (2018) 57–73, http://dx.doi.org/10.1016/j.robot.2017.11.013.
[26] W.H. Robinson, A.P. Lauf, Aerial MANETs: Developing a resilient and efficient platform for search and rescue applications, J. Commun. 8 (4) (2013) 216–224.
[27] D. Forouher, J. Hartmann, E. Maehle, Data flow analysis in ROS, in: ISR/Robotik 2014; 41st International Symposium on Robotics; Proceedings of, 2014, pp. 1–6.
[28] A. Bihlmaier, M. Hadlich, H. Wörn, Advanced ROS network introspection (ARNI), in: A. Koubaa (Ed.), Robot Operating System (ROS): The Complete Reference (Volume 1), Springer International Publishing, Cham, 2016, pp. 651–670.
[29] P.I. nigo Blasco, F.D. del Rio, M.C. Romero-Ternero, D.C.M. niz, S. Vicente-Diaz, Robotics software frameworks for multi-agent robotic systems development, Robot. Auton. Syst. 60 (6) (2012) 803–821, http://dx.doi.org/10.1016/j.robot.2012.02.004.
[30] C. Hu, C. Hu, D. He, Q. Gu, A new ros-based hybrid architecture for heterogeneous multi-robot systems, in: Control and Decision Conference (CCDC), 2015 27th Chinese, IEEE, 2015, pp. 4721–4726.
[31] R.E. Julio, G.S. Bastos, A ROS package for dynamic bandwidth management in multi-robot systems, in: Robot Operating System (ROS), Springer, 2017, pp. 309–341.
[32] A. Knoll, R. Prasad, Wireless robotics: A highly promising case for standardization, Wirel. Pers. Commun. 64 (3) (2012) 611–617.
[33] H. Schiøler, T.S. Toftegaard, Wireless communication in mobile robotics a case for standardization, Wirel. Pers. Commun. 64 (3) (2012) 583–596.
[34] K. Loupos, A.D. Doulamis, C. Stentoumis, E. Protopapadakis, K. Makantasis, N.D. Doulamis, A. Amditis, P. Chrobocinski, J. Victores, R. Montero, et al., Autonomous robotic system for tunnel structural inspection and assessment, Int. J. Intell. Robot. Appl. 2 (1) (2018) 43–66.
[35] K. Winstein, H. Balakrishnan, Mosh: An interactive remote shell for mobile clients, in: USENIX Annual Technical Conference, Boston, MA, 2012.
[36] M. Garzón, J. Valente, J.J. Roldán, D. Garzón-Ramos, J. de León, A. Barrientos, J. del Cerro, Using ROS in multi-robot systems: Experiences and lessons learned from real-world field tests, in: A. Koubaa (Ed.), Robot Operating System (ROS): The Complete Reference (Volume 2), Springer International Publishing, Cham, 2017, pp. 449–483, http://dx.doi.org/10.1007/978-3-319-54927-9_14.

[37] R. Parasuraman, K. Kershaw, M. Ferre, Experimental investigation of radio signal propagation in scientific facilities for telerobotic applications, Int. J. Adv. Robot. Syst. 10 (10) (2013) 364.
[38] P. Ghosh, A. Gasparri, J. Jin, B. Krishnamachari, Robotic wireless sensor networks, 2018, arXiv preprint arXiv:1705.05415.
[39] C. Rizzo, D. Tardioli, D. Sicignano, L. Riazuelo, J. Villarroel, L. Montano, Signal-based deployment planning for robot teams in tunnel-like fading environments, Int. J. Robot. Res. 32 (12) (2013) 1381–1397.



Danilo Tardioli is an Associate Professor at Centro Universitario de la Defensa, Zaragoza, Spain. He received the computer science engineer degree from the University of Bologna, Italy, in 2004 and the Ph.D. from the University of Zaragoza in 2010. His research activity is mainly focused on wireless real-time and quality-of-service communication in mobile ad-hoc networks in confined and challenging environments. He is a member of the Robotics, Perception, and Real-Time group of the University of Zaragoza. Also, he belongs to the steering committee of the IEEE RAS committee in multi-robot systems and is the secretary of the Spanish Society for Research and Development in Robotics. In the last few years, he has been visiting different institutions through Europe such as ETH Zurich, LAAS-CNRS Toulouse and KTH Stockholm.

Ramviyas Parasuraman is an Assistant Professor in the Department of Computer Science at the University of Georgia, Athens, USA. His research mainly focuses on control and communication aspects of networked robotic systems, intelligent teleoperation, and human–robot interfaces. Previously, he has held postdoctoral researcher positions at Purdue University, USA and KTH Royal Institute of Technology, Sweden. He worked at the European Organization for Nuclear Research (CERN) and obtained his Ph.D. from Universidad Politécnica de Madrid, Spain, in 2014. He graduated with Masters from Indian Institute of Technology Delhi, India in 2010, and was a recipient of the prestigious Marie-Curie ESR fellowship in 2011–2014.

Petter Ögren was born in Stockholm, Sweden, in 1974. He received the M.S. degree in engineering physics and the Ph.D. degree in applied mathematics from the Royal Institute of Technology (KTH), Stockholm, Sweden, in 1998 and 2003, respectively. In the fall of 2001, he visited the Mechanical Engineering Department, Princeton University, Princeton, NJ. From 2003 to 2012 he worked as a senior scientist and deputy research director in Autonomous Systems at the Swedish Defence Research Agency (FOI). He is currently an Associate Professor at the Robotics, Perception and Learning Lab (RPL) at KTH. His research interests include robot control system architectures, multi-agent coordination, teleoperation, navigation, and obstacle avoidance.