NDN-based IoT Robotics

Loïc Dauphin, Emmanuel Baccelli, Cedric Adjih, Hauke Petersen

Demo: NDN-based IoT Robotics

Loïc Dauphin
Inria, Université Paris-Saclay
loic.dauphin@inria.fr

Emmanuel Baccelli
Inria, Université Paris-Saclay
emmanuel.baccelli@inria.fr

Cédric Adjih
Inria, Université Paris-Saclay
cedric.adjih@inria.fr

Hauke Petersen
Freie Universität Berlin
hauke.petersen@fu-berlin.de

ABSTRACT

In this paper, we demonstrate how NDN can be used as network primitive on low-cost robots with the Robot Operating System (ROS).

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability;

KEYWORDS

ACM proceedings

ACM Reference format:
Loïc Dauphin, Emmanuel Baccelli, Cédric Adjih, and Hauke Petersen. 2017. Demo: NDN-based IoT Robotics. In *Proceedings of ACM Conference on Information-Centric Networking, , September 2017 (ICN'17),* 3 pages.
DOI:

1 CONTEXT

In the near-future, humans will interact with swarms of low-cost, interconnected robots. Such robots will hence integrate the Internet of Things, and coin the term *IoT robotics*.

The simplest robots can be modeled as shown in Fig. 1, i.e. a control loop coupling a sensor, an actuator and a controller module. A sensor module retrieves, transforms and makes sensor data available for other modules. An actuator module receives commands from a controller and translates it for the actuator hardware. A controller module implements a behaviour by taking into account user commands and sensor data, and sends commands to the actuator. A controller may receive commands from another controller (in which case the former behaves partly like an actuator).

Generally, each module consists of both hardware and software. Robot software is thus typically distributed over several computing units communicating with one another : microcontrollers executing real-time tasks, embedded computers managing the autonomous behaviours, remote computers enabling operators to monitor or control the robot.

In this context, using ROS (Robot Operating System [4]) is currently the dominant approach to implement distributed robotic software modules communicating with one another. ROS nodes can publish or subscribe to topics, which are named and typed data streams sent over the network. In this paper, we present preliminary work exploring the potential of using NDN as network primitive for ROS2 nodes (the newest version of ROS [6]).

ICN'17,
2017. .
DOI:

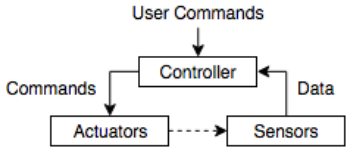
2 NDN-BASED IOT NETWORKING FOR ROS

As an improvement over ROS, ROS2 has an abstract middleware interface (RMW, ROS MiddleWare). Thus different network modules can be implemented in ROS2. The current one relies on publish/subscribe based on DDS [7] running on top of TCP/IP and multicast – a combination not ideally suited for low-end IoT hardware. For instance, the reference ROS2 implementation so far requires 700kB of Flash memory and 60kB of RAM for a very simple user application, including the OS and the networking module [7]. Such requirements do not fit low-end IoT hardware [3], which typically provide 100kB Flash and 20kB RAM in total, for the whole OS and application logic. Previous work [1] [5] has shown that NDN can provide an IoT network module with very small memory requirements: as small as 5kB of RAM and 16kB of Flash. Furthermore, the principle of naming data in NDN is similar to ROS's topics. Hence, our idea is to test NDN primitives to provide an alternate network module for ROS2 on low-end IoT robotics hardware.

Matching NDN primitives with ROS. NDN is based on a request/response pattern, while ROS requires its network module to provide publish/subscribe. This gap could be filled by prior work extending NDN with pub/sub functionality, such as COPSS [2]. Such extensions define a "Rendez-vous Point" (RP) which centralizes the data from publishers and synchronizes with subscribers. Due to the potentially high rate of message sending in robotics, the central RP leads to a bottle-neck and unacceptable delays. Incidentally, one of the main design goals of ROS2 (compared to ROS) is to avoid such centralized approaches. We hence designed and implemented a consumer-driven alternative mechanism as described below.

We assume that content fits in a single MTU, and that names (e.g. a command or a sensor reading) include a monotonically increasing sequence number, marking the order in which subsequent content was produced. We further assume consumers are preconfigured with the name of producers and corresponding FIB entries. These assumptions are reasonable for software modules of a simple robot communicating with one another. Nevertheless, ROS expects the producer pushes new data, thus dynamically tuning the rate of data retrieval, while with NDN the rate is consumer-driven. We thus had to devise a simple auto-adjustment mechanism for the rate at which Interests are sent by consumers. Note that here, contrary to traditional congestion control mechanisms, the consumer does not know if there actually is data to be retrieved. Hence a timeout cannot be simply interpreted as congestion/loss, but is more

Figure 1: Robot control architecture



likely to signal that newer data has not yet been produced, and the consumer thus needs to lower its Interest sending rate. To adapt its Interest rate, we thus use the following two-step process.

Dynamic Interest Timeout. First, a consumer sends a bootstrap Interest for a well-known name to the producer, to retrieve the current sequence number for this content. Then, the consumer sends Interests for subsequent content with sequence numbers with a rate adapted as follows. At any time, only we keep at most one pending Interest per content, and as soon as a PIT entry is removed/consumed, the subsequent Interest (resp. the Interest retransmit) is generated, with its associated PIT entry. Let t be the timeout for PIT entries and Interest retransmit. If a PIT entry times out, t is doubled (for the Interest retransmit). If the content is received before $t/4$ then t is halved for the next Interest to this content. If the content is received before timeout but after $t/4$, then t is left unchanged for the next Interest to this content. Initially, t has the value t_{max}, the master controller refresh rate, which can be assumed to be preconfigured. In practice, t is kept between t_{max} and t_{min}, another preconfigured value tuning reactivity.

3 IMPLEMENTATION & PRELIMINARY TESTS

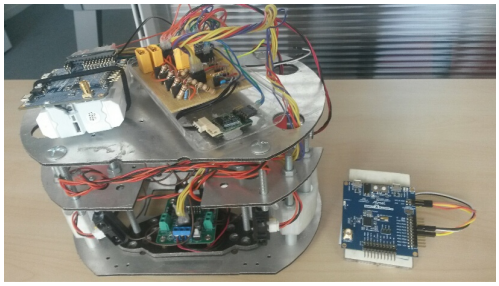
We implemented a proof-of-concept network module for ROS2 based on NDN using NDN-RIOT [5]. We then tested ROS2 enhanced with our implementation on a simple robot using low-end IoT communications modules: Atmel SAMR21 boards, based on low-power ARM Cortex M0+ microcontrollers and IEEE 802.15.4 communication.

Implementation details: We take advantage of the multithreading and IPC capabilities of RIOT. The NDN thread is a basic NDN router node, which considers the user application and the networking stack as 2 different faces. The user application uses callbacks to handle the arrival of an interest message (for the publisher) and to handle the arrival of a data message or a timeout associated with a previously sent interest (for the subscriber). Note that the implementation is MAC-layer independent, e.g. can be debugged on top of an ethernet emulator (with RIOT native).

Demo setup: We built a simple 2-wheeled robot wirelessly remote-controlled by a gyroscope. The robot's angular speed is controlled by the angle of the 1 dimension gyroscope. One SAMR21 board controls the robot's motors while another SAMR21 board extracts raw gyro data, and computes the angle. Note that in this configuration, the actuator module and the controller module (see Fig. 1) run on the same board.

The consumer and producer use the topic name `/gyro/angle`, and content is a series of fixed point numbers representing the gyro angle in radians. The subscriber, which only knows the topic

Figure 2: Wheeled robot and remote control



name, first sends the bootstrap interest for a well-known name `/gyro/angle/sync` to get the latest data published associated with its sequence number. By convention, the content of such a name is never cached by potential intermediate nodes (only the producer knows for sure what the latest sequence number is). Once the current sequence number is known, the consumer sends subsequent interests for the names `/gyro/angle/<sequence number>`, at a rate driven by DIT. If synchronization is lost at some point down the line (detected via too many timeouts in a row) the consumer reverts to the bootstrap phase of DIT and starts over. Note that, using this approach in cases where several nodes are consumers for the same content, the content could be cached at potential intermediate nodes to decrease traffic near the producer.

In our demo, we will show the robot moving, controlled remotely by the gyro and NDN with DIT over wireless. In terms of user experience, we achieve an RTT lower than 20ms, far less than the maximum 100ms typically required for the control to be fluid.

4 CONCLUSION & NEXT STEPS

In this demo we demonstrate how NDN can be used as network primitive for ROS on low-cost robots in the Internet of Things. Our next steps are to test NDN on robots embarking more sensors and actuators – thus more consumers, and more producers which could be discovered dynamically. We will also test this approach in cases where some modules interact through Internet (higher RTT) where it might be beneficial to send several interests in advance. In effect, complementary ROS features such as nodes discovery and services would be implemented and could be tested on low-end IoT robots.

REFERENCES

[1] E. Baccelli et al. 2014. Information centric networking in the IoT: Experiments with NDN in the wild. *ACM ICN* (2014).
[2] Haitao Wang et al. 2017. COPSS-lite: Lightweight ICN Based Pub/Sub for IoT Environments. *arXiv:1706.03695* (2017).
[3] L. Dauphin et al. 2017. Low-Cost Robots in the Internet of Things: Hardware, Software & Communication Aspects. *ACM EWSN Next Mote Workshop* (2017).
[4] M. Quigley et al. 2009. ROS: an open-source Robot Operating System. *Proceedings of ICRA Workshop on Open Source Software* (2009).
[5] W. Shang et al. 2016. The design and implementation of the NDN protocol stack for RIOT-OS. *IEEE Globecom Workshops* (2016).
[6] Brian Gerkey. 2015. Why ROS 2.0? (2015). http://design.ros2.org/articles/why_ros2.html
[7] Victor Mayoral. 2014. ros2 embedded nuttx. (2014). https://github.com/ros2/ros2_embedded_nuttx