Simple Protocol for Heterogeneous Embedded Communication Networks

Martin Zenzes[1], Peter Kampmann[2], Moritz Schilling[3], and Tobias Stark[4]

[1]martin.zenzes@dfki.de
[2]peter.kampmann@dfki.de
[3]moritz.schilling@dfki.de
[4]tobias.stark@dfki.de

January 18, 2016

Abstract

In the field of robotic systems, the Internet of Things, or modern industrial applications, many systems consist not only of one central computer, but of a heterogeneous set of embedded electronics employing different types of microcontroller and Field Programmable Gate Array (FPGA). These electronics are located near sensors and actuators somewhere in the mechanical structure, and are connected with different communication technologies in order to exchange measurements and commands. We present Node-level Data Link Communication (NDLCom), a flexible and minimal protocol for communicating in such heterogeneous systems consisting of low end devices. The details of the protocol design together with application examples and timing measurements of two robotic systems are discussed in this paper.

1 Introduction

With the advancement of technologies like networked industrial plants, the Internet of Things, and highly integrated robotic systems, heterogeneous networks of embedded devices have to be designed to cope with increasingly complex tasks. Examples for these are the processing of multi-modal sensors distributed across the system or local high-frequency control loops.

Quick turnaround in development cycles increase the need for reusable electronics with stable communication-interfaces. The addressed issues do also arise in robotic systems which experience increased complexity due to their dexterity, strive for long-term autonomy and comprehensive range of sensors. Communication channels with reasonable bandwidth and elaborate transport layers are a key requirement to cope with the increasing need for exchanging data.

Until now, the choice among the non-proprietary communication networks for this task was between custom-built Universal Asynchronous Receiver Transmitter (UART) based communication based on string parsing and full standards defining many aspects of the communication, strongly differing in their bandwidth, hardware requirements, and protocol complexity. Solutions like SpaceWire [1], CAN (Controller Area Network) [2] or the MOST Bus [3] are backed by the need of professional industrial applications and are standardized to allow tight control over the quality and properties of the resulting network. High transmission rates, low error rates and hard realtime are opposed by a missing flexibility and high unit cost due to the extensive standardization. The specific cabling and additional ICs needed for signal transportation and modulation raise the requirements for PCB design and exclude very low end devices from communication.

Another possibility exists in adopting existing communication protocols to the needs of embedded applications in order to reduce the resource requirements, like nanoIP [5] and LightweightIP [6]. While they allow to use existing tools and experience, very small embedded devices with less than 1 kB of RAM are still out of reach.

To bridge this gap, a low-level communication protocol has been developed, that features low hardware requirements and is therefore able to connect a broad spectrum of devices, ranging from microcontrollers and FPGAs to POSIX computers. NDLCom defines a simple packet format to use the OSI Layers 2/3 (see Figure 1) for data exchange using point-to-point serial communication channels, and needs much fewer resources compared to traditional network technologies like IP. To allow frictionless and comfortable work with the binary streams of the protocol, a collection of

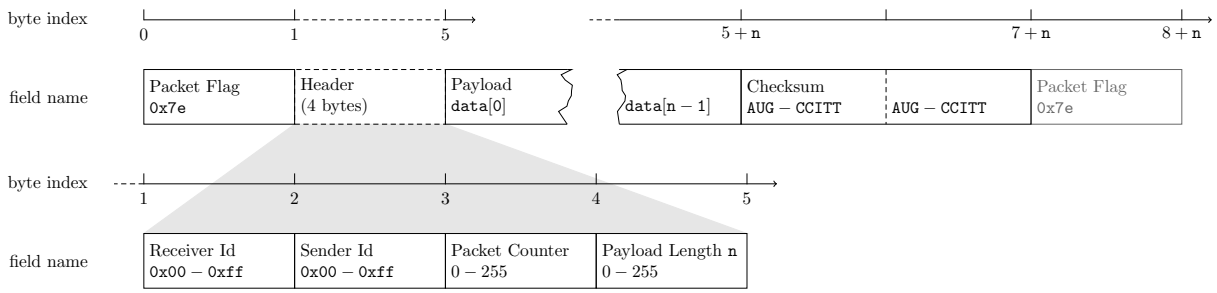


Figure 2: Visualization of the field layout in one NDLCom message. The protocol overhead for `n` bytes of payload is $4 + 2 + 2 = 8$ bytes, including the optional closing flag byte drawn in gray. The maximum size of the payload is limited to 255 B given by the range of the length field.

Layer	Data Unit	Function	Examples
4. Transport	Segments	Transmission of data segments between points on a network, including segmentation and acknowledgement	TCP, UDP
3. Network	Packet, Datagram	Structuring and managing multi node network, including addressing and routing	AppleTalk, IPv4, IPv6 **NDLCom**
2. Data Link	Frame	Reliable transmission of data frames between two nodes connected by physical layer	PPP, IEEE 802.2, L2TP **HDLC**
1. Physical	Bit	Transmission and reception of raw bit stream over physical medium	DSL, USB

Figure 1: Visualization of the first four layers according to the OSI model [4], placing HDLC framing at the *Data Link* layer and the NDLCom protocol at the *Network* layer. As there is no segmentation of large messages in NDLCom, a Packet is synonymous to a Frame. The upper layer payload representation used by applications as well as the underlying low level transport are not covered in this paper.

command-line as well as graphical tooling was developed. These include real-time communication statistics, various mechanisms for transports, binary data logging, online data visualization, and automatic CSV export.

Details on the protocol implementation and design rationale are given in this paper together with experimental validation of the performance of the newly developed protocol. The computational resources needed by the current designs on different embedded platforms and languages are discussed to show the compactness and simplicity of the approach. Possible strategies for nodes forwarding messages between their ports are described and evaluated. The feasibility of the developed approach is demonstrated by successful applications inside robotic systems as they are developed at German Research Center for Artificial Intelligence (DFKI) in Bremen.

2 Design of NDLCom

The problem of communication in heterogeneous systems is commonly solved using different technologies, creating the need for flexible and adaptable infrastructure. Distributed devices (nodes) may need to communicate with any other participant to exchange information, including the central authority to allow the gathering of the global state of the system. During prototyping an additional requirement includes fast evolving designs and the need to reuse single components between generations of hardware.

To be able to use a diverse set of electronics with different properties and limited resources, the different layers of communication have to be flexible and interchangeable. Printed Circuit Board (PCB) area in embedded devices is sometimes limited, thus needing dedicated IC's for the electrical layer of the communication is not always practicable. A byte oriented transport mechanism, like the classical UART, is seen as the smallest common denominator to provide connectivity. This enables using heterogeneous electrical point-to-point connections without the need to alter older PCB designs. Each device is directly connected to at least one of its neighbours under the presumption that each connection is unique, i.e. that the resulting network can be represented by an undirected acyclic graph.

Most of the communication inside current robots is based on continuously updated streamed data, with update rates of up to hundreds of Hz. If a packet is lost due to transmission errors, it should be detectable but reliable transmission is not needed as the next packet of the stream will arrive shortly. Services which need the additional transport security can be implemented as additional layer according to the OSI model as shown in Figure 1, for example In-System-Programming. By carefully keeping the separation of different OSI layers the reusability and flexibility is maintained.

2.1 Message Layout

As shown in Figure 2, each NDLCom message consists of a start flag, a header, a payload, a checksum, and an optional end flag. The length of a header l_{header} includes entries for sender and receiver address, a packet counter, and the payload length.

Because the number of devices communicating in an internal network of a robot is known in advance (see Figure 4), 8 bit sender and receiver addresses are sufficient, limiting the overhead. One special receiver address is reserved to implement broadcast messages for easy and efficient access to every node in the whole system.

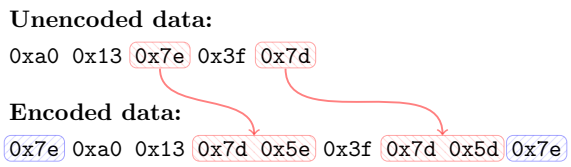


Figure 3: Encoding a set of bytes with HDLC-like framing and escaping to provide flow control for transmission on a byte oriented transport medium. The encoding is performed by replacing any reserved bytes in the block of data (⃠) and surrounding the result with a start/stop flag (⃠).

To be able to detect packages lost during transmission, an 8 bit packet counter is included in the header. The current counter value has to be incremented and remembered by a sender for each receiver, adding a maximum of 256 B to the memory requirements of the implementation.

With the length of the payload to follow as last byte the header comprises 4 B which can be easily handled, especially by 32 bit devices. When creating, forwarding, or routing messages inside the network dedicated memory to temporarily store a maximum sized message is needed in each component. Thus a small maximum payload size of 255 B allows using very small devices with limited capabilities.

As the underlying communication channel does not have to be reliable against transmission errors, a 16 bit Frame Check Sequence (FCS) is appended at the end of each packet. The FCS16-CCITT [7] using a `0x1021` polynomial is utilized to guard against data corruption. The length of a fully assembled message with a known payload size $l_{payload}$ is given by:

$$l_{assembled} = l_{header} + l_{payload} + l_{FCS} \tag{1}$$

For example, an unencoded payload of 20 B results in an assembled packet with a length $l_{assembled}$ of 26 B.

2.2 Framing and Escaping

The NDLCom protocol uses High-level Data Link Control (HDLC)-like framing to segment the byte stream provided by the underlying transport medium into individual datagrams [7][8]. The start of a datagram is exclusively denoted by the special byte `0x7e`, which is not allowed inside the datagram and has to be removed via escaping. This allows for simple stateless segmenting of an incoming byte stream into datagrams: Always assume a new packet after an observed `0x7e`. To be able to transmit this reserved flag byte as part of a message it is replaced with the sequence `0x7d 0x5e` while the escape byte `0x7d` itself is replaced by `0x7d 0x5d`. Both steps are visualized in Figure 3 and have to be reversed on the receiving side in oder to decode the datagram into the original message.

As two different bytes out of 256 possible values need to be escaped by inserting an additional byte, the average length of an escaped message depends on the distribution of bytes in the assembled message. Including the start/stop flag and given an uniformly distribution of bytes the length can be expressed as:

$$l_{encoded} = 2 + (1 + 2/256) \cdot l_{assembled} \tag{2}$$

The closing flag byte is not strictly needed by the protocol itself, but eases processing during forwarding of messages. In the worst case, the length of an escaped message can double.

The overhead of assembling a message by adding the header to a payload of 20 B is 30 %. Encoding this assembled message by adding the start/stop flags and escaping any special bytes, an additional overhead of 8 % is introduced, assuming uniformly distribution of bytes. Thus the total overhead from payload to escaped message is in this example at least 41 %.

2.3 Forwarding

Due to using individual point-to-point connections to create a network each node has not only to handle messages designated for it, but also to retransmit messages which are addressed to other nodes. For this purpose two different forwarding mechanism are implemented, *Cut Through* and *Store & Forward*, configurable per device at build time.

When a device is performing *Cut Through* forwarding, the received data can be transmitted on the outgoing interface as soon as the receiver address is known. As shown in Figure 2 this is represented by the second byte of a packet, thus the forwarding latency $t_{forward}$ introduced on each hop is at least $3 \cdot t_{byte}$. The mean memory consumption is reduced on all devices along the chain, as less transient bytes have to be stored while a packet is decoded. The downside is that broken packages are not filtered out during forwarding and can increase bus congestion until they are consumed by the receiver or dropped at the end of a chain. Additional byte-level timeout logic is needed to prevent blocking the outgoing interface while a receiver is waiting for the remainder of a broken message. If a passing message is to be used to update the routing table with the observed originating address, the full FCS has to be calculated and checked to prevent invalid entries.

The second type of message handling is *Store & Forward*, where each message is completely received by every node along the path. Only after verification of the FCS the message is processed further. In this case, the forwarding latency for each hop is twice the full transmission time for a packet, $2 \cdot t_{trans}$. *Store & Forward* increases the computational requirement and message latency because a packet has to be decoded, stored and encoded again on every hop along the path. This allows each node to detect and filter invalid messages on a passing stream and prevents unnecessary blocking of faster interfaces by messages coming from slower ones.

Both forwarding mechanisms are implemented, while *Cut Through* limits latency and *Store & Forward* is preferable when crossing bandwidth domains, see also Section 3.3.

2.4 Routing

When a node has more than one interface, it has to decide on which interface a message has to be sent. Therefore each of these nodes has to maintain a routing table containing the mapping of each device address to the last interface at which a message with this address as a sender has been observed. Combined with the special broadcast address, this yields the steps shown in Algorithm 1. As the network is not allowed to contain cyclic connections, selection of the outgoing route is always unique, contrary to more complex protocols like TCP/IP.

```
routingTable[all] ← UNKNOWN
for all message from origin do
  if recvId is ownId or recvId is BROADCAST then
    process internally
  else
    routingTable[sendId] ← origin
  end if
  dest ← routingTable[recvId]
  if recvId is not ownId then
    if dest is UNKNOWN or recvId is BROADCAST
    then
      forward to all except origin
    else
      forward to dest
    end if
  end if
end for
```

Algorithm 1: Processing of messages after successfully decoding on a receiving interface (*origin*). A device has to handle messages directed at its own or at the broadcast address internally. All messages with a receiver addresses not its own (or broadcast) have to be retransmitted, either on an interface specified by the routing table or on all other interfaces to make sure the destination is reached.

Note that after initial system boot, the routing tables of all nodes are uninitialized. This can lead to excessive message flooding because of unknown destinations, but converges to normal operation once all relevant connections have been used to fill the routing table.

2.5 Resource Usage

The code for performing sending, routing and receiving of messages is written in C without dynamic memory allocation to allow reuse on all platforms supported by a compiler. For an ARM STM32F1 target the code size of the implementation is 3.2 kB, while the usage of static RAM accounts to 1 kB, excluding byte level buffering performed by the interrupt routines. The memory usage can be significantly reduced if the maximum value of device addresses and the maximum amount of payload is limited at compile time, allowing to fit into devices with less than a kB of RAM. A VHDL implementation synthesized for a Spartan6 XC6SLX45 featuring 2 ports is using 1300 Slice Registers, 1900 LUTs, and 10 BRAMs, including logic to realize UART-like interfaces.

	System	Data		Nodes per	
		quantity	rate	chain	system
Charlie	● Actuator	↑↓ 5.2 KiB/s	100 Hz	4-6	20
	◇ ForceTorque	↓ 1.5 KiB/s	20 Hz	1	5
	◇ SensorArray	↓ 2.3 KiB/s	10 Hz	1	2
Gripper	● BaseBoard	↑↓ 0.1 KiB/s	1 Hz	–	1
	● SensorFusion	↓ 0.1 KiB/s	1 Hz	1	3
	● CamControl	↓ 7.0 KiB/s	30 Hz	2	6
	◇ ForceTorque	↓ 0.5 KiB/s	13 Hz	1	3
	◇ PiezoSensor	↓ 2.0 KiB/s	40 Hz	2	6
SherpaTT	● Actuator	↑↓ 6 KiB/s	200 Hz	5-6	26
	◇ ForceTorque	↓ 4.5 KiB/s	100 Hz	1	5
	◇ FuseControl	↑↓ 0.1 KiB/s	1 Hz	5	5
	◇ Module	↓ 1 KiB/s	10 Hz	2	2

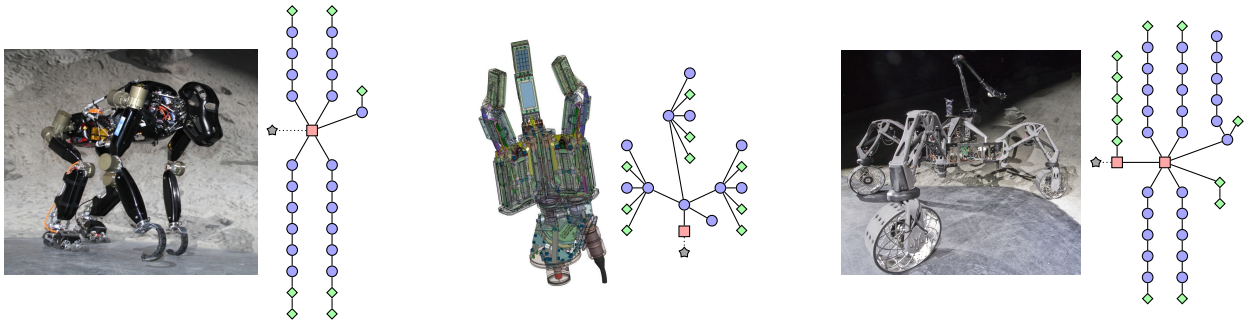
Table 1: Data- and updaterate for streams of devices in the three different systems described, see also Figure 4. Streams denoted by a ↑↓ are birectional, where setpoints are sent to the device and measurements are transmitted back. A high update rate as used for Actuators helps to to mitigate problems due to spurious packet losses.

3 Application

NDLCom is used since 2011 in a number of robotic systems in DFKI, three of which are depicted in Figure 4. Different byte level transports are used by the various systems, ranging from simple TTL UART, to RS422, a custom high-speed Low Voltage Differential Signal (LVDS), or even tunneling packets through UDP. Plattforms which are used for nodes understanding NDLCom range from Atmega88 to STM32F4, and from Spartan3/6 to Zynq.

3.1 Charlie

The four-legged hominid robotic system Charlie was designed with regard to a low system weight, robustness, and agility [9]. The sizes of limbs in the Charlie system is proportional to the lengths of body segments found in a chimpanzee. Altogether 36 active and six passive Degrees of Freedom (DoF) allow the robot to perform a variety of different movements. In order to improve the locomotion and mobility characteristics, biologically inspired structural components like an active artificial spine and sensing feet (to improve the perception of the environment) are applied to the robotic system. These structures enable the robot to demonstrate exceptional movement patterns, e.g. to change is posture from a quadrupedal to a bipedal pose. A local control loop realized in the two rear legs uses force vectors measured by local microcontrollers to influence



(a) Walking robot *Charlie* [9] is using one device chain to control each of its four legs, the two front legs pointing upwards while the two rear legs point downwards. One additional short chain is used to control the spine. The central Embedded System connects all devices and talks via Wifi to the remote Control PC.

(b) In the *Gripper* [10], each finger is independently controlled and connected to a central FPGA, which also connected an independently preprocessing FPGA. Every finger contains different processing FPGA and microcontroller. An Embedded Computer provides Wifi connectivity for the FPGA to the remote Control PC.

(c) The *SherpaTT* [11] system employs four symmetric legs plus one arm, pointing upwards on the right of the schematic above. All are connected to a central Embedded Computer, which additionally connects two microcontroller for interface managment. A second internal computer, branching to the right, controls one supplemental chain of microcontroller extending upwards and is providing Wifi access.

Figure 4: Node structures in several robotic systems using NDLCom. The heterogeneous architectures used are depicted as FPGA (○), embedded Linux (□), microcontroller (◇) and control PC (⬠). Pysical connections based on UART/LVDS are drawn as solid lines while remote Wifi connections to controlling computers are drawn dotted.

the set point of the FPGA based actuator modules.

For the byte level communication transport an UART-like transmission with 921 600 Baud is used between all nodes.

3.2 Gripper

The Gripper system [10] of the SeeGrip project was designed for the application in deep-sea environments in depths of up to 6000 m. It comprises a variety of tactile sensing modules of different modality in order to mimic the touch sensing capabilities of humans in robotic systems. In total 570 exteroceptive sensing modules and 25 proprioceptive sensing modules are integrated into the sensing system. The distributed hardware architecture within the Gripper, shown in Figure 4b, is used to locally pre-process sensory data to minimize the communication effort between the processing elements and to allow forming local control loops.

For the embedded communication the same UART-like transport running at 921 600 Baud as in the Charlie system is used. This system was not available for the measurements performed in Section 4.2.

3.3 SherpaTT

SherpaTT is a hybrid driving and walking rover system [11] and part of a multi-robot system that aims at establishing a logistics chain for sample acquisition in the lunar polar regions [12]. The rover is using an active suspension system consisting of four legs each with a drivable and steerable wheel at its end. The legs of the suspension have three active DoF each for positioning the wheel within the respective work space. Together with the wheel steering and the wheel drive, each leg has 5 DoF, resulting in 20 DoF in the overall suspension system. For active adaption to the ground, a force-torque sensor at each wheel mounting point and an Inertial Measurement Unit (IMU) mounted in the central body are used as feedback. Vision, laser scanners or alike are not used for ground adaption but part of higher-level functionality. Apart from the suspension system, a 6 DoF manipulator is mounted on top of the rover for handling payloads with a standard interface, such as sampling modules or battery packs that can be exchanged with other systems in the multi-robot team.

The SherpaTT system uses a combination of a high speed Manchester Encoded LVDS transmission with 320 MBaud and the UART-like communication with 921 600 Baud as it is used in the Charlie and Gripper systems. While all actuators in the four legs use the fast transport, the actuators located in the arm following the *ArmBase* device and all of the microcontroller based measurement devices are connected via the slow transport, see also Figure 4c. Because of the crossing of bandwidth domains, a *Store & Forward* based routing algorithm is used to transfer messages between the two bandwidth domains.

4 Evaluation

When NDLCom is to be used to create distributed low level control loops or handle transient emergency messages, it is important to know some of the time and reliability properties of the combined communication infrastructure. These determine the maximum control frequency of a distributed control loop or properties for data processing chains consisting of two or more NDLCom devices sharing information. Measuring and monitoring the realtime properties of individual nodes and comparing them to expected values also helps to find bugs and missing features in the implementation.

The latency & jitter between between a sender and receiver introduced by the message transport is examined in more detail. Latency is understood to be the mean time it takes until a message is successfully delivered from a sender across the network to a receiver. It is dominated by the processing inside intermediate nodes forwarding messages to their destination. Jitter is defined as the deviation in latency of a message with respect to the ideal latency over a number of randomly selected samples [13].

4.1 Methodology

Two of the systems introduced in Figure 4 where available to perform measurements: SherpaTT and Charlie. These where examined with the whole system moving in normal operating conditions, all devices sending their normal data streams and motors performing their tasks.

To assess the capability of the communication layer an NDLCom message with a *Ping* payload is used, where any receiving node will send a reply to the sender of the message as fast as currently possible. The time passed between sending the message and receiving the reply from the network is called the Round Trip Time (RTT). In order to account for every device of a particularly system, a list of all active device addresses is assembled. A *Ping* message is transmitted to a randomly selected address of this list to trigger a response packet. If such a response is observed within a large enough wait time the elapsed time is noted in a log file, otherwise a package loss is noted. After a small random time offset, the next request is sent.

The ideal transmission time t_{trans} of a packet is the time it takes to send it from one device to the next and it is limited by the raw bandwidth of communication layer.

$$t_{byte} = bandwidth^{-1} \tag{3}$$

The time it takes to transmit one byte on an `8N1`-UART with 921 600 Baud, as implemented in the Charlie system, is for example 10.85 µs. The transmission duration t_{trans} of a complete escaped message depends on the actual number of bytes of this message and the time it takes for each byte to be encoded:

$$t_{trans} = t_{byte} \cdot l_{encoded} \tag{4}$$

With an payload of 20 B the transmission duration for the fully encoded packet becomes 306.0 µs. In practice the current load on the communication layer will increase the mean transmission time as a message might have to be buffered until the outgoing interface of the sending device is no longer blocked.

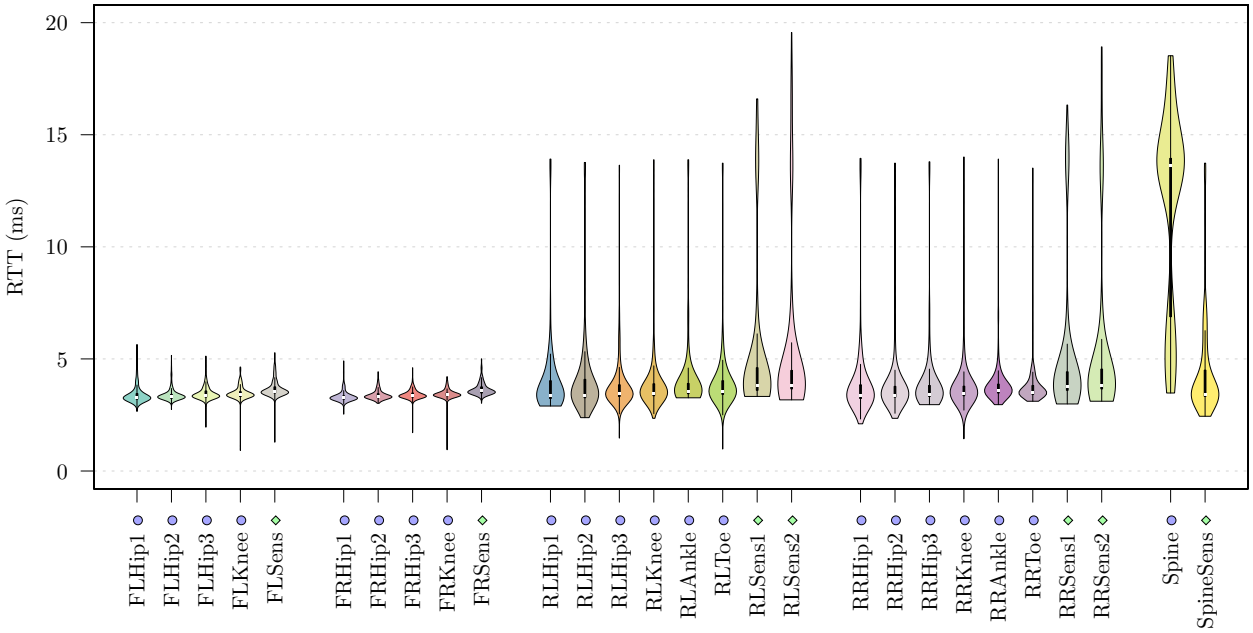
Response time, or RTT, is the delay between finishing the transmission of a packet by the originating device until it finishes decoding the received response. The minimal response time for the *Ping* reply on an idle communication bus is limited by twice the forwarding delay $t_{forward}$ (see Section 2.3) per device along the path, the transmission delay t_{trans} of the responding device and the transmission delay of the sending device. When all devices possess the same raw bandwidth this results in:

$$t_{response} = 2 \cdot n \cdot t_{forward} + 2 \cdot t_{trans} \tag{5}$$

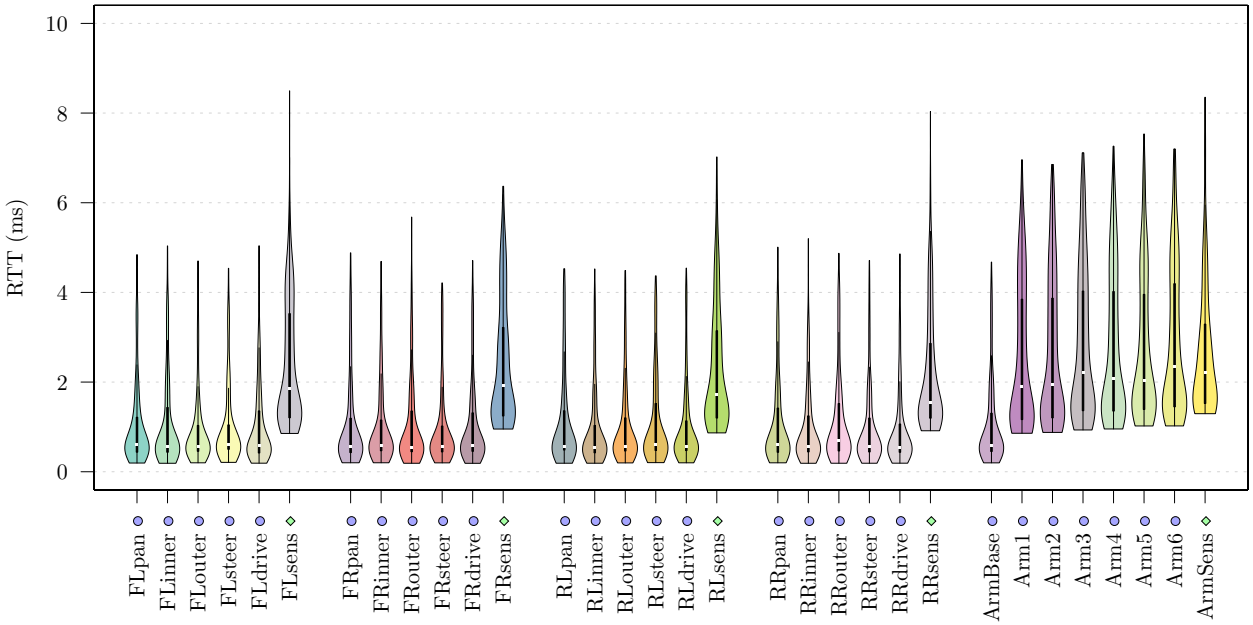
When for example a *Ping* message with 20 B of payload has to perform $n = 4$ hops to reach its destination via *Cut Through*, the resulting ideal response time would be expected to be at least 872.5 µs. When all of the forwarding devices along the chain are performing *Store & Forward*, each message has to be fully decoded before it can be send again. In this case, the response time increases to be at least 3060 µs. In a real system, the response time can additionally vary substantially if participating devices and channels are not idle as is show in Figure 5.

4.2 Measured Response Time

The distribution of observed RTT for each device, grouped by their respective low-level transport chain, in both the Charlie and SherpaTT systems is shown in Figure 5. The FPGA used in Charlie and SherpaTT are performing the fast *Cut Through* forwarding when they can. It can be seen that the RTT is not primarily influenced by the hop count, e.g. distance, in the network to the queried device. Transmission of very big packets as by the *Sens2* boards in the two rear legs of the Charlie system can cause very large increases in the jitter as they block traffic on the bus for a long time. All observed response times are fast enough to perform update rates in the range of hundreds of Hz, the observed jitter is most of the time uniformly distributed. Crossing a bandwidth domain, where packages have to be buffered, adds a clearly notable step of around 1.3 ms to the measured latency due to storing the message. This happens in the SherpaTT system, where all microcontroller and the FPGA beyond the ArmBase device use a UART connection with only 921 600 Baud, contrary to the 320 MBaud in the rest of the system. Additionally, the lower available total bandwidth influencing the forwarding time can be observed by the marginally steeper mean RTT slope in the devices of the SherpaTT arm.



(a) Results for the Charlie system. The high maximum response time on the devices of the two Rear legs are caused by a large payload of 250 B sent from the very last devices in the chain, see Table 1. The FPGA of the *Spine* device uses a softcore without any free processing capabilities for answering messages, so a packet can only be handled after finishing the long running main-loop resulting in a changed distribution. Forwarding is done in the FPGA. Due to the limited available bandwidth because of the high traffic the mean response time of all devices is relatively high for all devices.



(b) Results for the SherpaTT system. The slower response characteristic of the sensing microcontroller at the end of each chain are clearly seen. The bandwidth crossing done there causes a large step in minimum RTT, caused by the *Store & Forward* performed. Using the slower low level transport and FPGA type beyond the *ArmBase* also influences the response behaviour. The high available bandwidth on the rest of the devices, e.g. low bus congestion, produces faster mean response times.

Figure 5: Measured RTT for the SherpaTT and Charlie system using 10 000 *Ping* messages sent randomly from the embedded PC. Visualized as Violin Plot to show the distribution of observed RTT for every device in the system. Devices are plotted in the order they appear on their chain and an additional gap between each chain is inserted. The naming scheme for device names in the legs of both systems is using a *FL* for *Front Left* for example, while *RR* represents *Rear Right*.

5 Outlook

Being able to transmit messages in a standardized manner from all devices connected to an NDLCom network allows to synchronize the local clocks of every device across the system using an algorithm like Simple Network Time Protocol [14, #14]. Such synchronized clocks allow for advanced sensor fusing algorithms and stream alignment of jittered data packets prior to actual processing.

One possible application of synchronized clocks would be to correlate timing information from distributed embedded microphones located inside the system using algorithms based on the Time Difference of Arrival (TDOA) applied to detected peaks. With the speed of sound of $330\,\mathrm{m\,s^{-1}}$ and a global accuracy of distributed clocks better than $50\,\mu\mathrm{s}$ the error in distance when triangulating should be below 17 mm, well below the typical dimension of a robotic system.

Automatic runtime assessment of link-quality and node availability is possible for example by sending broadcast beacon messages of actively querying each device from a central location. It might be possible to deduce the current bus load by monitoring the mean response time of devices behind the network. Automatic shape detection of the network tree can be done by collecting and analyzing the routing tables of all devices.

Swapping and reusing electronics as well as code between systems after an initial implementation phase speeds up the development process and reduces the learning curve for newly participating engineers. In model based robot design, it becomes easier to generate code for involved platforms as the communication infrastructure is shared. To facilitate reuse in other environments it is planned to release the C and VHDL implementation under an Open Source license.

References

[1] SM Parkes and Philippe Armbruster. SpaceWire: a spacecraft onboard network for real-time communications. In *Real Time Conference, 2005. 14th IEEE-NPSS*, pages 6–10. IEEE, 2005.

[2] ISO 11898-1:2015. Road vehicles – Controller area network (CAN) – Part 1. [Online; accessed 7-January-2016].

[3] Stefan Poferl, Markus Becht, and Piet De Pauw. 150 Mbit/s MOST, the Next Generation automotive infotainment system. In *2010 12th International Conference on Transparent Optical Networks*, 2010.

[4] Wikipedia. OSI model — Wikipedia, The Free Encyclopedia, 2016. [Online; accessed 14-January-2016].

[5] Zach Shelby, Petri Mähönen, Janne Riihijärvi, Ossi Raivio, and Pertti Huuskonen. NanoIP: The Zen of Embedded Networking. In *Communications, 2003. ICC'03. IEEE International Conference on*, volume 2, pages 1218–1222. IEEE, 2003.

[6] Adam Dunkels. Design and Implementation of the lwIP TCP/IP Stack. *Swedish Institute of Computer Science*, 2:77, 2001.

[7] ISO 13239:2002. Telecommunications and information exchange between systems – High-level data link control (HDLC) procedures. [Online; accessed 12-January-2016].

[8] Keith Hogie, Ed Criscuolo, and Ron Parise. Using Standard Internet Protocols and Applications in Space. *Comput. Netw. ISDN Syst.*, 47(5):603–650, April 2005.

[9] D. Kuehn, M. Schilling, T. Stark, M. Zenzes, and F. Kirchner. System Design and Field Testing of the Hominid Robot Charlie. *submitted to Journal of Field Robotics*, July 2015.

[10] Peter Kampmann and Frank Kirchner. Towards a fine-manipulation system with tactile feedback for deep-sea environments. *Robotics and Autonomous Systems*, 67:115–121, 2015.

[11] Florian Cordes, Christian Oekermann, Ajish Babu, Daniel Kuehn, Tobias Stark, and Frank Kirchner. An Active Suspension System for a Planetary Rover. In *Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS 2014), June 17-19, Montreal, Canada*. o.A., 6 2014.

[12] Roland U Sonsalla, Florian Cordes, Leif Christensen, Steffen Planthaber, Jan Albiez, Ingo Scholz, and Frank Kirchner. Towards a heterogeneous modular robotic team in a logistics chain for extended extraterrestial exploration. In *Proceedings of the 12th International Symposium on Artificial Intelligence, Robotics and Automation in Space-i-Sairas*, 2014.

[13] JEDEC Standard 65B, September 2003.

[14] D. Mills, J. Martin, J. Burbank, and W. Kasch. Network Time Protocol Version 4: Protocol and Algorithms Specification. RFC 5905 (Proposed Standard), June 2010.