Comparing Concepts of Service Blocking Queues in Hardware-in-the-Loop Systems

Tobias Konheiser
Test System Development
ZF Mobility Solutions GmbH
Ingolstadt, Germany
tobias.konheiser@zf.com

Christoph Funda
Test System Development
ZF Mobility Solutions GmbH
Ingolstadt, Germany
christoph.funda@zf.com

Reinhard German
Department Computer Science 7
Friedrich-Alexander-Universität Erlangen-Nürnberg
Erlangen, Germany
reinhard.german@fau.de

Kai-Steffen Hielscher
Department Computer Science 7
Friedrich-Alexander-Universität Erlangen-Nürnberg
Erlangen, Germany
kai-steffen.hielscher@fau.de

***Abstract*—ZF is developing an autonomous driving system, which requires extensive testing of the developed devices and software on hardware-in-the-loop (HIL) systems. Therefore, a robust and high-performing HIL system is essential. The purpose of a HIL system is to replay recorded data to the device-under-test. Recordings are loaded, processed and streamed to the device-under-test with real-time requirements. This streaming chain includes processing nodes and queues. This requires careful management of queue configurations. An overflow in the queue will result in packet loss, while an underflow may violate the real-time constraint.**

This study aims to develop and evaluate concepts for service blocking queues. These concepts block or pause the incoming service to a queue when necessary to avoid queue overflows and associated data loss. However, an out-of-the-box solution is not available and different approaches affect the behaviour and performance of the system. Therefore, the developed concepts are evaluated against each other and against the existing system based on selected performance parameters in specific scenarios. The scenarios cover a wide range of situations, reflecting standard input data with varying numbers of parallel streams and bottleneck scenarios forcing queue overflows or blockages.

The developed service blocking queue concepts eliminate data loss in all scenarios, but introduce overhead, resulting in reduced system performance. However, the service blocking queue concept using a modified token-bucket approach proved to be the best solution, as the elimination of data loss justifies the additional overhead. This concept is proposed for implementation and deployment on the HIL system.

***Index Terms*—robot operating system, performance evaluation, queueing, hardware-in-the-loop**

List of Acronyms

CAN	Controller Area Network
CPU	Central Processing Unit
DUT	Device Under Test
FIFO	First-In-First-Out
HIL	Hardware-in-the-Loop
KPI	Key Performance Indicator
LV	LabVIEW
NC	Network Calculus
RAM	Random Access Memory
ROS	Robot Operating System
TCP	Transmission Control Protocol

Supported by ZF Friedrichshafen AG.

I. Introduction

Complex computer systems need validation, especially in safety-critical areas of application such as autonomous driving. In order to validate the developed control devices, Hardware-in-the-Loop (HIL) systems are commonly used. Therefore, an appropriate HIL system must be developed, which streams recorded data to a Device Under Test (DUT). Due to real time requirements, the HIL system needs to provide high data throughput and reliability. Once a solid foundation for a HIL system is developed, more edge cases need to be analysed to improve the system. Data loss in the HIL streaming chain is a worst-case scenario in the validation process for electronic devices and must be eliminated in the HIL streaming chain. Not only can data loss require the repetition of a test run, it can also cause inaccurate or wrong results. The HIL system under development consists of three main components, where data loss may occur. The first part is a streaming chain implemented in the Robot Operating System (ROS). The second part is implemented in LabVIEW (LV). Both components are connected by a Transmission Control Protocol (TCP) connection specified in [1].

The TCP algorithm includes a data loss prevention mechanism, and such a mechanism can easily be realised in LV. Therefore, the ROS streaming chain is the only part where data loss can occur in the system under study, where no solution is available.

In this work, we propose, implement and evaluate the performance of different data loss prevention mechanisms in ROS. The service-blocking-queue concepts address the issue of data loss. This work explains the relevant fundamentals for the service blocking queue concepts, followed by a detailed analysis of different concept approaches. The work concludes

with an evaluation of the results using important Key Performance Indicators (KPIs).

Special requirements need to be fulfilled by those concepts. The main goal is to eliminate data loss in every scenario. Simultaneously, the performance of the HIL system must be ensured, and the designed concepts must not be prone to errors during implementation and deployment.

II. Methods

A. HIL System

"Hardware-in-the-loop system is a non-intrusive test approach, containing physical controller connected in open- or closed-loop with virtual or semi-virtual subsystems, providing faithful physical replicas of the real world and evaluating the System under test in either black/grey/white box manner." is a well suited definition for a HIL system proposed by [5].

The present study focuses on the HIL structure illustrated in Figure 1.

The streaming chain consists of different processing nodes, queues, and the connections between them. The PCs run Linux with ROS or LV respectively.

The overall HIL system has hard real time requirements for the timing precision of the outgoing packets. Through the usage of a playback buffer, the processes that stream the data to the buffer have only soft real time requirements.

Buffer overflows have already been a concern in previous work. As described in [2], buffer underflows and overflows are undesired behaviour. Therefore, a monitoring concept to detect system performance is implemented on the HIL system. This algorithm based on Network Calculus (NC) is described in [3]. Data loss due to queue overflows is an undesired and hard to analyse behaviour.

Derived from this problem, the following research questions can be formulated:

- Which communication concepts can be used to eliminate data loss in the ROS streaming chain?
- How does a data loss prevention concept influence the system performance?

B. ROS

The Robot Operating System is a framework used to send messages between processing nodes. The documentation can be found in [6]. It is commonly used to program robots and other computer systems. A project is structured in workspaces and packages. There, different nodes or nodelets are implemented. ROS nodes are the basic building blocks that can communicate with other nodes and provide services.

A ROS node can be implemented in C++. A basic node consists of a cpp file, which contains a main method. This method initialises the node and performs the implemented functionality. As different nodes run in different processes, the nodes have separate memory spaces. This is a hurdle, as data that is sent from one node to another needs to be copied.

Nodelets offer a solution to this issue. The key distinction between nodes and nodelets is that multiple nodelets operate within the same process, and therefore share the same memory space. This can be used to share pointers to data instead of copying the data.

To communicate, ROS provides two mechanisms. ROS topics are a publish-subscribe based concept, where two First-In-First-Out (FIFO) queues buffer the packets, but drop the oldest message, once a queue overflow occurs. A ROS service is a request-response style communication, where a client sends a request to the server and waits for its response.

To send messages, the underlying TCPROS protocol is used. It is based on the same error prevention mechanisms as TCP.

C. Concepts

This work aims to evaluate different concepts based on data loss and performance metrics. The developed concepts are based on different ideas.

1) Concept 1: Nodes with Topic: The first concept models the current setup. This reference implementation consists of two ROS nodes, one producer that creates messages and one consumer that processes these messages. The producer sends the packets to the consumer via a ROS topic.

This concept is used to estimate the changes introduced by the other concepts to the HIL system.

2) Concept 2: Nodelets with Topic: Concept 2 introduces the nodelet concept. As shared memory is required for concept 4, this concept is a modified version of concept 1, where the producer and consumer are implemented as ROS nodelets.

3) Concept 3: Topic and Token: This concept adds a token counter to concept 2. This counter is used to keep track of the number of elements between the producer and consumer. The producer decreases the counter for each sent message. If the counter reaches zero, no new messages can be sent. Once a message is received by the consumer, it replies with a token message. When the producer receives the token message, the counter is increased by one and a paused streaming can be restarted. The starting number of the token counter must reflect the minimum queue size in the section, to eliminate any data loss. Both communication paths, the message sending and the token responses, use separate ROS topics.

4) Concept 4: Service with 2 Queues: The last concept utilises two local queues and the shared memory of nodelets. This enables the producer and consumer to exchange pointers through a ROS service. The producer fills its queue and the consumer pops elements from the other queue. As soon as the producer queue is full, the producer requests a queue pointer exchange service from the consumer with the pointer to the full queue. Once the consumer queue is empty, it replies with the pointer to the emptied queue.

Through unrestricted access to the local queues, the nodelets can avoid queue overflows and data loss.

D. Scenarios

To test the created concepts, different scenarios help to identify differences.

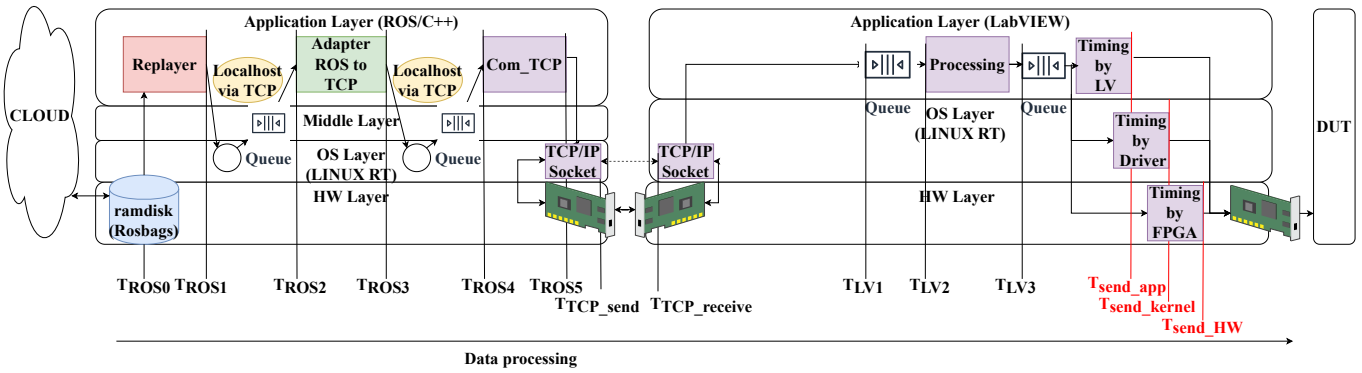

Fig. 1. Detailed Conceptual Model and Software Instrumentation [4]

1) Data Types: The HIL system works with a lot of data types that need to be replayed. This can include Controller Area Network (CAN) messages, Radar packets and other sensor streams. To test the concepts, representative CAN and Radar data was used. Some describing KPIs of the input data is listed in Table I.

TABLE I
KPIs OF CAN AND RADAR DATA

KPI	CAN	Radar
Message Size	32 B	1538 B
Min Cycle Time	2453 ns	2707 ns
Mean Cycle Time	2940961 ns	1703144 ns
Max Cycle Time	29295807 ns	163720481 ns
Min Processing Time	943 ns	1628 ns
Mean Processing Time	5416 ns	8136 ns
Max Processing Time	39170 ns	108098 ns

2) Multiple Streams: Another key ability of a HIL system is the capability to stream multiple parallel streams. Therefore, the scalability of the concepts is important. For testing this, the concepts are tested with 1, 2, 4, 8 and 10 parallel streams of CAN and Radar data. To achieve this, the created nodes and nodelets are duplicated, and they only influence each other through the shared resources.

E. KPIs

To evaluate the performance of an implementation, various KPIs are needed. In this work, the following measurements are used:

- Central Processing Unit (CPU) Utilisation
- Random Access Memory (RAM) Utilisation
- End-to-End Delay
- Jitter
- Data Loss

The CPU and RAM utilisation is measured during the test run of a concept in a defined scenario. The CPU utilisation reflects a single core, running only the components under study. The RAM utilisation is measured for the related processes. The test runs are performed, and the measurements are logged on a development system with the specifications listed in Table II.

TABLE II
HARDWARE SPECIFICATION OF DEVELOPMENT SYSTEM

KPI	Value
CPU Model	Intel(R) Core(TM) i7-6820HQ
CPU Core(s)	4
Thread(s) per core	2
CPU max MHz	3600
RAM size in GB	16
RAM Type	DDR4
RAM Speed in MT/s	2133

To calculate the following KPIs, three timestamps for each packet are logged. The first timestamp is logged, when a packet is sent by the producer. The other two timestamps relate to the input and output time at the consumer. With these timestamps, the end-to-end delay of a packet can be calculated. It reflects the time, a packet needs from leaving the producer until it was processed by the consumer.

The jitter describes the maximum deviation in both directions from the output of the consumer node to the input trace provided in the scenario.

Having timestamp logs for each packet, enables us to calculate the data loss, as we know how many messages were processed at a logging point.

III. RESULTS AND DISCUSSION

We introduced the implemented concepts and different scenarios in the previous section. Through a larger message size and smaller message cycle times in the radar data, the differences of the concepts are better emphasized in these results. The results of scenarios using CAN data are comparable. Therefore, we limit the presentation of results to the scenarios using radar data.

A. Results of Concept 2

The results of concept 2 are used as a baseline to compare the different concepts, as concept 2 allows a well-founded comparison to concept 1 regarding the differences between

nodes and nodelets and to concept 3 and 4 regarding the additional service-blocking-queue mechanisms.

TABLE III
RESULTS OF CONCEPT 2 WITH RADAR DATA

Stream Count	1	2	4	8	10
mean CPU Util	7.6%	10.5%	16.4%	26.3%	29.9%
mean RAM Util	1.0%	1.2%	1.6%	2.4%	2.9%
mean Delay	0.3ms	0.5ms	0.8ms	2.0ms	3.1ms
mean Jitter	166ms	168ms	174ms	177ms	177ms
mean Data Loss	0.2%	0.2%	0.3%	4.0%	15.6%

Table III lists the absolute values for the relevant KPIs. The CPU utilisation scales well with the number of streams and lies in a moderate range. The RAM utilisation is low, even with a small amount of available memory. The end-to-end delay increases with the number of parallel streams, as the different streams influence each other. The jitter of the message output remains nearly constant throughout the scaling. The amount of lost packets due to queue overflows increases significantly, but because of small queue sizes, this behaviour is expected.

B. Results of Concept 1

To compare the concepts, the following tables list the difference of the measured KPIs compared to the baseline of concept 2. A positive value relates to an increased value and a negative value to a reduction.

TABLE IV
DIFFERENCE OF CONCEPT 1 TO BASELINE WITH RADAR DATA

Stream Count	1	2	4	8	10
mean CPU Util	-16%	39%	44%	43%	47%
mean RAM Util	-40%	-50%	-62%	-75%	-79%
mean Delay	3126%	481%	393%	270%	245%
mean Jitter	26%	25%	16%	29%	31%
mean Data Loss	13566%	4262%	3529%	840%	225%

The results in table IV show some significant trends. The CPU utilisation increases significantly, whereas the RAM utilisation is reduced even more. This relates to the separation of different ROS nodes in different processes. As nodelets share the same process, less overhead is produced, resulting in a reduced CPU utilisation but an increased RAM utilisation. The increased overhead is also reflected in the mean end-to-end delay. It is extremely high, and this results in an increased jitter and data loss.

C. Results of Concept 3

This is the first concept that introduces a data loss prevention mechanism. This results in an elimination of data loss, as seen in Table V.

TABLE V
DIFFERENCE OF CONCEPT 3 TO BASELINE WITH RADAR DATA

Stream Count	1	2	4	8	10
mean CPU Util	22%	17%	7%	9%	24%
mean RAM Util	0%	0%	0%	0%	0%
mean Delay	33%	24%	22%	23%	9%
mean Jitter	3%	4%	3%	12%	12%
mean Data Loss	-100%	-100%	-100%	-100%	-100%

As additional messages need to be sent, some overhead is introduced. This increases the CPU utilisation slightly. The delay and jitter increase, because new waiting times are necessary to not lose any packets. This overhead is expected and seems justified, as no packets are lost any more.

D. Results of Concept 4

To test different approaches, concept 4 uses local queues. The queues are filled completely before they are forwarded to the consumer node. This results in an increased end-to-end delay for most packets, while the queue is filled.

TABLE VI
DIFFERENCE OF CONCEPT 4 TO BASELINE WITH RADAR DATA

Stream Count	1	2	4	8	10
mean CPU Util	62%	69%	94%	156%	188%
mean RAM Util	0%	0%	0%	0%	-2%
mean Delay	2661%	1749%	1025%	559%	385%
mean Jitter	46%	44%	33%	19%	11%
mean Data Loss	-100%	-100%	-100%	-100%	-100%

The consumer nodelet processes the full queues as fast as possible, resulting in a bursty message output flow. This increases the difference to the input flow and therefore the jitter, as noted in Table VI. The memory usage is not influenced, but the used ROS service does not scale well, resulting in an unreasonable increase in CPU utilisation. Due to strict blocking of individual threads when calling a service, the CPU resource consumption is increased. A larger queue size can reduce this issue. The nodelets in concept 3 do not block and just idle when the service is blocked. Therefore, concept 4 is not as efficient and less suitable for the HIL use case as concept 3.

IV. CONCLUSION

In this work, we developed and compared different concepts for service blocking queues in ROS in the context of HIL systems.

We introduced the required background information and explained the developed concepts. The concepts, which include a reference implementation, a token bucket approach, and local queues, were analysed in different scenarios. Additionally, the scenarios are performed with a varying number of parallel streams.

During development and testing of the created concept, advantages and disadvantages were revealed. The reference implementation with ROS nodes worked with only slight differences to the nodelet approach.

The approach with tokens in concept 3 required the least change to the existing structure of the streaming system, but the handling of counters and messages requires caution during implementation to avoid errors and hard to diagnose problems. This concept would also allow the usage of ROS nodes.

Concept 4 utilises local queues and a blocking service call to exchange queue pointers. Here, ROS nodelets are necessary. This change in communication structure requires fundamental changes, and the concepts do not scale as well as concept 3.

Therefore, concept 3 is the best-suited approach for this use case.

As expected, all service-blocking-queue concepts introduce an overhead, but the overall performance is acceptable, especially because data loss was eliminated in all scenarios.

Future work is to add this concept to the HIL system streaming chain and to ensure the estimated performance is achieved. Other concepts can be added to this framework, based on the results and insights gained through concept 3 and 4.

REFERENCES

[1] Wesley Eddy. Transmission Control Protocol (TCP). RFC 9293, August 2022.

[2] Christoph Funda. Arrival- and service-curve estimation methods from measurements to analyze and design soft real-time streaming systems with network calculus. 12 2022.

[3] Christoph Funda, Pablo Marin Garcia, Reinhard German, and Kai-Steffen Hielscher. Online Algorithm for Arrival & Service Curve Estimation, 2023.

[4] Christoph Funda, Tobias Konheiser, Reinhard German, and Kai-Steffen Hielscher. How to Model and Predict the Scalability of a Hardware-In-The-Loop Test Bench for Data Re-Injection?, 2023.

[5] Christoph Funda, Tobias Konheiser, Thomas Herpel, Reinhard German, and Kai-Steffen Hielscher. An industrial case study for performance evaluation of hardware-in-the-loop simulators with a combination of network calculus and discrete-event simulation. In *2022 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, pages 1–7, 2022.

[6] Open Robotics. *ROS Documentation*. Available at https://wiki.ros.org/Documentation.

ACKNOWLEDGMENT

This research was supported by ZF AG and ZF Mobility Solutions GmbH (a company of ZF group).