Technical Design Report

Software for Science - CERN Load Balancing

Team 6 Hexoxide
Eloy Maduro Clment
Geoffrey van Driessel
Corn Lukken
Bram Tukker

January 21, 2019
v1.1

Changelog

Version	Date	Description
0.2	07-11-2018	1. Added information on changing to Debian image. 2. Added information about investigating Metricbeat for information logging. 3. Added table of contents. 4. Added information on how legacy software was setup and used. 5. Added previously logged metrics and desired metrics.
0.3	01-12-2018	1. Remove outdated or incoherent information. 2. Add networking benchmarks.
0.4	07-12-2018	1. Add installation instructions to report. 2. Explain 94Mbit synthetic throughput. 3. Introduction to alternative technologies. 4. Incorporate research design into TDR. 5. Add experiment network-baseline details.
0.5	12-12-2018	1. Add operation diagram for network baseline experiment. 2. Finish documentation of network baseline experiment.
0.6	10-01-2019	1. Add round robin ICN experiment.
0.7	15-01-2019	1. Add source code documentation section. 2. Add results for round robin ICN.
1.0	16-01-2019	1. Determined mandatory changes. 2. Improved hardware & software setup introduction. 3. Added networking diagrams. 4. Described software and system components. 5. Generalized SSH description. 6. Described Metricbeat and its metrics. 7. Explained table underlining. 8. Describe the run operation for the round robin ICN experiment. 9. Discussion & results. 10. MELK-installation-guide reference added 11. MELK Sub Question. 12. Added ZooKeeper section.
1.0	19-01-2019	13. Added Bibtex for IEEE referencing. 14. Added CERN official Bibtex document output to reference TDR.
1.0	21-01-2019	15. Additional bookmark references for uncounted sections.

Contents

Introduction

This document reflects the discovered & analyzed technical components for the CERN Load Balancing project. Components are identified in previously available documents as well as discoveries of the project team itself.

Experiments and their setup are detailed so they can be repeated easily in the future, as well as instructions on how to install all required components. This work will introduce new software that can be used in future projects to continue experimenting and make new discoveries. This new software has evolved from previous work which will be briefly touched upon. It is important to note that there is a distinction between the two new systems we provide: the ICN and the Zookeeper implementation.

The rationale for the project will be explained in the introduction along with its common terminology, an overview of past and future events as well as information about previous work.

Alice O^2

CERN is the organization which has build worlds largest particle accelerator know as the Large Hadron Collider (LHC). The LHC, along with its various detectors, is currently receiving a scheduled upgrade during a phase called Long Shutdown 2 (LS2). One of these various detectors is known as A Large Ion Collider Experiment (Alice). This detector will also be receiving an upgrade during LS2 and as a result, the amount of data created by the detector will increase. However, the network behind the detector is currently not capable of processing the data at the increased rate. The Alice O^2 project describes the necessary changes to software, hardware and network in order to be able to process the increased data rate of the detector. In preparation of the upgrade, the Alice O^2 project was documented in a Technical Design Report (TDR) released by CERN in 2015 [**?**].

The CERN Load Balancing project focuses on the network load balancing described in the TDR for Alice O^2. The two primary types of computer nodes that will perform the network load balancing operations for the received detector data are known as the First Level Processors (FLPs), and the Event Processing Nodes (EPNs), however, throughout this document these will also be abbreviated in singular form being FLP and EPN. The FLPs will be directly interfacing with the Alice detector on a special device known as an Field Programmable Gate Array (FPGA) and based on triggers from these FPGAs, they will start transmitting data to the EPNs. In turn, the EPNs will receive this data and from the FLPs, however, to correctly process the data all FLP's need to transmit to a single EPN for each event. The load balancing should ensure the correct transmission of so called Sub-Time Frames(STFs), transmitted from the FLPs to the EPNs so they can be processed into Time Frames(TF). The overall Alice O^2 system and operation is described in greater detail but this is best read in the TDR [**?**] as the currently described information should suffice for further reading of this document.

Previous work

The first work was done in 2017 and the result showed the effectiveness of a simple round-robin algorithm [**?**]. The software implementation created to run the required experiments introduced a new third type of computing node named the Information Node. The Information Node was responsible for maintaining a list of available EPNs in the network using the ZooKeeper distributed key-value store [**?**].

Later on, the ZooKeeper configuration was optimized by determining the optimal ticktime while evaluating two new algorithms: Blacklist and Re-Initialization [**?**]. Upon evaluating the two algorithms, it became clear that the Blacklist algorithm was superior when compared to the Re-Initialization algorithm.

All work was based on evaluations and experiments that were run on X86 based processors but the work of Puls introduced experiments that could be run on ARM based processors, more notably Raspberry Pi clusters [**?**]. These clusters could now be used as a more available and lower cost platform to run experiments on.

Figure 1: Overview of past and future events with regards to the CERN Load Balancing project.

Original O2Balancer

Disclaimer: The original O2Balancer had never reached an operational stage between the duration of the Software for Science Load Balancing project 2018/2019 and although it provides details which will help others come a long way, it will take further steps to be able to use the original O2Balancer to run experiments.

At the start of this project, several experiments had previously been done on this subject. These previous experiments have led to the creation of existing software. It is important to identify how these software packages worked and reference them, so that they can be used in the future.

To make sure the exact version of the software can be retrieved, the git version commit hashes are listed. To checkout a specific commit hash from a repository, execute git checkout COMMIT-HASH.

Software	Purpose	Commit
O2Balancer [?]	Binaries used in experiments containing specific algorithms to be tested.	36eec89
BalancerScripts [?]	Setting up nodes with Ansible and orchestrating an experiment. Gathering and purging log files and creating plots and graphs from log data.	99f7177

Table 1: Original O2Balancer Software.

Newer software will likely still be distributed through git, but will be made available through the software-for-science Github organization. The repositories from previous experiments can be found on the organization project [?].

Installation

Documentation for the O2Balancer was based upon CentOS, as a result it might be difficult to get the software running for other types of Linux distributions although it has been successfully done before. The documentation on how to install & run the O2Balancer for the Raspbian Linux distribution has been lost but the installation instructions for CentOS are still available [?]. Attempts to retrieve the lost documentation by searching through Github, attempting to contact the author and even rescuing files of previously used SDCards were unsuccessful.

For other distributions with different package managers it should be fairly easy to replace the packages specified for the YUM package manager, however, installing all requirements for the O2Balancer can require a significant amount of system memory (between 8-16 gigabytes).

Verify software installation

Once the O2Balancer and its dependencies are installed, the installation can be verified by executing execute.sh in the build directory of O2Balancer, however, first ZooKeeper needs to be started.

```
sudo /usr/share/zookeeper/bin/zkServer.sh start
./execute.sh
```

Configuration arguments & yaml files

Each individual binary has command line arguments and a configuration file, some of the parameters configured through these two methods might override one or another.

Information Node

Parameter	Data type	Description
–info-config	string	Path to yaml configuration file.
–sample-size	int	Size of individual packets send between FLP & EPN in MB.
–ip	string	Ip address of ZooKeeper service

Table 2: Information Node parameters.

FLP

Parameter	Data type	Description
–flp-config	string	Path to yaml configuration file.
–restartFairRoot	boolean	Runs reinitialization algorithm if true
–ip	string	Ip address of ZooKeeper service

Table 3: FLP parameters.

EPN

Parameter	Data type	Description
–epn-config	string	Path to yaml configuration file.
–amount-flps	int	Amount of FLPs in this experiment the EPN will expect data from.
–flp-port	int	The network port the EPN will listen on for data from the FLP.
–ip	string	Ip address of ZooKeeper service

Table 4: EPN parameters.

The execute.sh script will open a total of six windows which should appear, each running a different type of node software. In total there should be three EPNs, twp FLPs & one Information Node.

1 Hardware & software setup

The Raspberry Pi clusters used in experiments were initially empty. So the Raspberry Pi clusters are configured with an image. To speed up the initial process, one image was made and reused for the other clusters. The required dependencies were also installed on the initial image. A Bash script was developed which helps with the installation of the dependencies. The remaining processes were automated with Ansible. All the scripts to speed up initial pi

cluster setup are stored in repository that are publicly accessible and listed within this document. Bash was chosen as an automation language because of its high availability on Linux distributions, Subsequently using Bash instead of Ansible allows to install dependencies on nodes which might not be accessible with Ansible such as isolated virtual machines.

Raspberry Pi nodes are ARM based and this can pose limits on the available software, for instance Logstash will not compile or run on ARM based systems. It is important to take these limits into account and before starting to implement properly investigate software and its availability.

1.1 Hardware Specifications

According to the Alice technical design report [**?**], there will be two types of computing nodes in the Alice O^2 project after LS2. In total there will be about 2000 nodes in O^2, 250 so called FLP nodes and 1750 EPN nodes.

A set of 80 computing devices grouped in units of four has been provided by the Amsterdam University of Applied Sciences (AUAS) for this project. These 80 nodes will be used to simulate network scenarios as close as possible to the actual O^2 computing network. The computing devices at AUAS will be Raspberry Pis, specifically, model 3 B+.

In Table 5 a comparison of differences in hardware is detailed. It is clear that the computing nodes provided by AUAS are vastly inferior to the computing hardware that will become available for Alice O^2. These hardware differences must be taken greatly into account during the project and its various experiments.

Component	**FLP (250)**	**EPN (1750)**	**Raspberry Pi 3 B+ (80)**
CPU	? x86	32 Core x86	4 Core ARMv8
RAM	32GB	128GB	1GB
Ethernet	4 x 10Gbit	10Gbit	300Mbit

Table 5: Hardware specifications of computing nodes.

1.2 Network

The network during the experiments consisted of eight Raspberry Pi nodes each with two network interfaces. One of these interfaces had an internet connection through a bridged router, but was segregated from the actual internal network using Network Address Translation (NAT). The initial system design has every component installed on one Raspberry Pi but unfortunately some system components such as Logstash turned out to be incompatible with ARM based processors. A virtual machine running on an X86 based host was configured to resolve any ARM related compatibility issues, however, this host resided outside the NAT segregated network and additional steps using port forwarding and additional NAT were required to resolve them.

The following networking diagram shows an overview of the entire network using Cisco diagramming standards. In this diagram, the host at 192.168.3.2 is the virtual machine responsible for running X86 system components, while the host at 192.168.3.4 is the router responsible for segregating the Raspberry Pis from the rest of the internal network. Each of the firewalls represent the NAT that occurs on two places in the network which will require port forwarding and tunneling later.

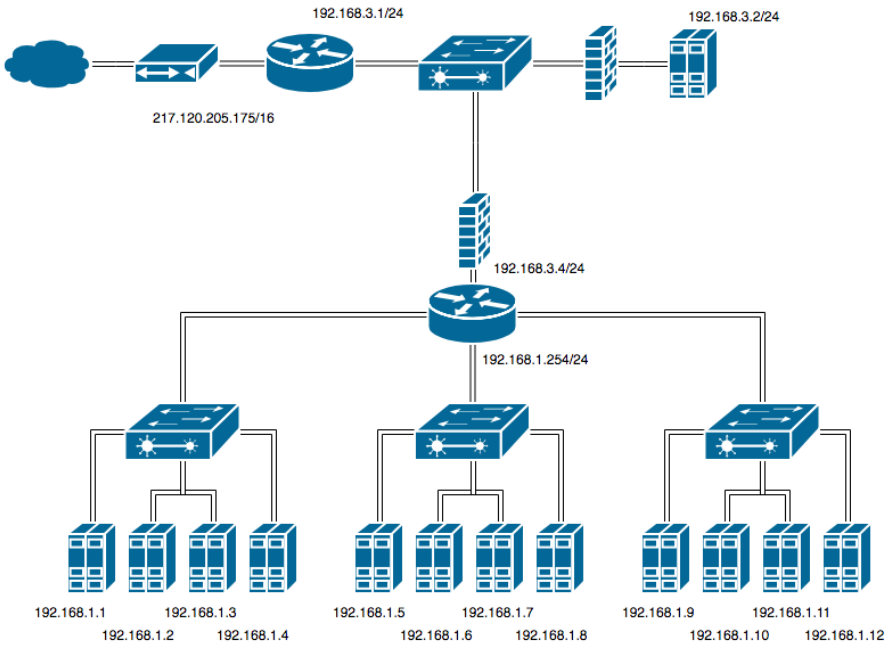


Figure 2: Networking diagram with auxiliary networking interfaces omitted. A diagram with the auxiliary interface can be found online [5].

1.3 New software introduction

An environment as similar as possible to the one used at CERN is desired. It has, however, proven unfeasible to use CentOS, due to the limitations the Raspberry pi version has in comparison to the CentOS x86_64 image. One of the main reasons is the inability to switch between gcc versions. Normally, a tool called SCL provides the switching for specific versions of many development tools. Furthermore, the version of gcc supplied with CentOS is incompatible with the version of boost that is specified in many of the previous experiments. To continue using CentOS, gcc would have to be built from source.

To resolve the issues with the CentOS image, an alternative had to be elected and based on previous experience, Manjaro 17.1 was chosen. Manjaro provides a full featured ARM image for with a large amount of available packages. Manjaro being an Arch based distribution is extremely permissive in adaptive configurations allowing it to closely reflect any other distribution.

Unfortunately, due to the rolling release model of Manjaro, this also led to a lot of issues including, but not limited to: broken packages, unavailable mirrors and missing kernel modules. As an alternative, Raspbian stretch was chosen because of its extensive support and large community, which should allow to more easily mitigate problems.

Based on the reports of the previous experiments, a table could be compiled which compares the version of libraries from the legacy software to the versions used in the new software. It should be noted that the information about the original O2Balancer was determined to be partially incorrect as it does not use FairMQ itself but instead relies on FairRoot.

Library/Tool/OS	**Version used**	**Version Original O2Balancer**
FairMQ	1.3.6	1.1.5
ZeroMQ	4.2.5	4.2.1
ZooKeeper	3.4.9	3.4.9
Cmake	3.12.4	3.11.0
Boost	1.68.0	1.66.0
Yaml-cpp	NONE	0.5.2
FairLogger	1.3.0	NONE
Compiler	gcc 6.3.0	gcc 6.3.0
OS	Raspbian stretch	Raspbian stretch

Table 6: The installed software dependencies on the Raspberry Pis.

1.4 New software & system components

The system consist of different nodes all of which are running Debian stretch as operating system and which are accessible using SSH. The Raspberry Pi based nodes have all dependencies installed using the raspberry-dependency repository and its install scripts. The X86 based virtual machine has different software components installed which will be described later for these components an installation guide is available in the documentation repository.

Among the set of nodes is one new node known as the Information Control Node (ICN). This node will replace the Information Node used in the original O2Balancer. The new ICN node will have additional functionality compared to the original Information Node which is the reason behind the new name. The other nodes in the new software such as the FLP and EPN will have the same functionality as in the original O2Balancer.

Repository	Purpose
raspberry-dependency [?]	Detailed instructions listing required Debian packages and automated install script to install further dependencies not installed through package manager.
BalancerScripts2 [?]	Small scripts that can be executed from the command line to orchestrate and/or configure software components that allow the experiment to run across different nodes.
O2-Balancer2 [?]	Core binaries which run the experiments by configuring and sending data on different channels across the network.
Documentation [?]	Essential documentation on the overall project as well as how to get started with the continuation of our project.

Table 7: Overview of essential repositories.

1.5 Installation

To ensure the reproducibility of experiments executed by the new software, a Bash script [?] has been developed which will install all the dependencies automatically. Using Bash instead of Ansible ensures the availability of the install script on isolated platforms. The installation instructions are based on Debian distributions and additional effort could be required to be able to use them on different distributions. Along this installation scripts additional documentation is available in the documentation repository which will allow to configure the X86 based system components.

1.6 SSH

Remote access to each of the individual nodes is done via SSH in the setup the first node is used a portal to access the other nodes in the network. Using a single node as portal limits the amount of ports that need to be forwarded to the outside network. Additionally, an SSH key from the portal node can be added to every node as known host so that SSH logins no longer require a username & password after logging into the portal.

The SSH configuration also uses SSH tunneling. This is done to allow access to a web based dashboard running on the X86 based virtual machine.

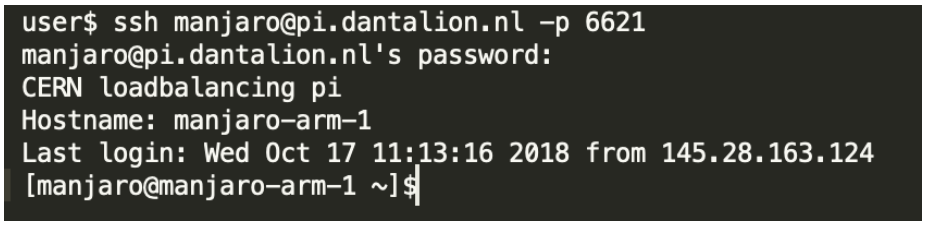


Figure 3: Example of login in using SSH on remote accessible raspberry pi.

1.7 Ansible

To run and configure tests on the pi clusters, some things will need to be configured. Ansible will be used to handle the configuration of each pi, defining which experiments to run and with which parameters to do so. Ansible allows to apply different parameters to different categories of hosts.

1.8 MELK

Metricbeat can be used to gather statistics of the nodes. This data can be sent to Logstash, which should have run on the ICN. Then, Logstash can output the data in two ways:

- Plain text file
- Elasticsearch

To easily visualize the data, Kibana can be used to retrieve the data from Elasticsearch. This service should also run on the ICN. By port forwarding the service on an external machine, Kibana can be viewed inside a web browser.

Due to incompatibilities with Logstash on ARM, both Logstash and Elastic-Search were installed on the X86 virtual machine. The overall experience with MELK was very poor. The exact problems, discoveries and conclusion will be discussed later.

Kibana is part of the ELK stack, which is used to properly interpret metrics from the hardware and network during experiments. ELK is short for Elasticsearch, Logstash and Kibana. These three pieces of software process incoming data from hardware components running Metricbeat.

1.8.1 Elasticsearch

Elasticsearch is a distributed, RESTful search and analytics engine capable of solving a growing number of use cases. As the core of the Elastic Stack, it centrally stores the data.

1.8.2 Logstash

Logstash is an open source, server-side data processing pipeline that ingests data from a multitude of sources simultaneously, transforms it, and then sends it to Elasticsearch.

1.8.3 Kibana

Kibana is able to visualize Elasticsearch data and navigate the Elastic Stack, and can do so using a wide range of visualizations.

1.8.4 Metricbeat

Collect metrics from systems and services. From CPU to memory, Redis to NGINX, and much more, Metricbeat is a lightweight way to send system and service statistics. However, in the setup for running O2Balancer experiments, the data gathered is limited to metrics such as CPU load (in percentages), memory usage (in MegaBytes), and network throughput (in KiloBytes per second).

1.8.5 Data

While Metricbeat offers a wide range of data to send, in the setup for running O2Balancer experiments, the data gathered is limited to metrics such as:

- CPU load (in percentages)
- Memory usage (in MegaBytes)
- Network throughput (in KiloBytes per second)
- Network latency (in milliseconds)

1.8.6 Visualizations

Kibana offers several types of visualizations: charts such as area, heat map, horizontal and vertical bar and pie. For the purpose of the experiments and their effects on the hardware and network, line charts proved to be most effective in visualising the relevant data, because these showed changes over time. These graphs are then displayed in a grid formation on a dashboard, which created an ordered overview of all measured statistics.

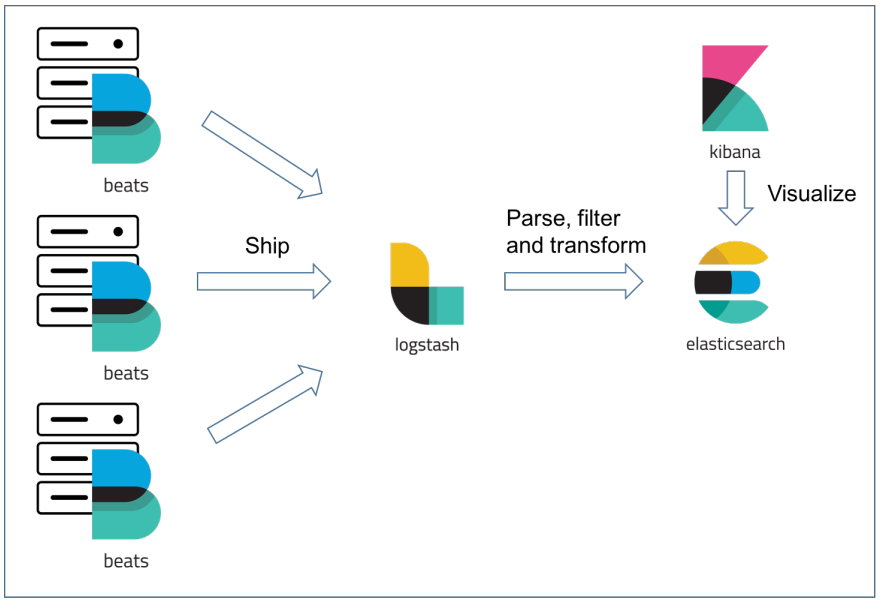


Figure 4: Overview of MELK stack.

1.8.7 Logging metrics

The decision to use Metricbeat was heavily based on its extensive selection of different metrics. The following overview of metrics shows what was intended to be used to evaluate different experiments.

#	Metric	Measure
1	Temperatures	degrees Celsius (°C)
2	Throughput	KiloBytes per second (KB/s)
3	Scaling governer / Core clock	MegaHertz (MHz)
4	Memory usage	MegaBytes (MB)
5	CPU usage	Percentage (%)
6	Swap	MegaBytes (MB)
7	Fail-over	Number of Nodes (NoN)

Table 8: Metric that will be used in the experiments.

1.9 Source code documentation

Beside the regular documentation, the source code itself can also be documented. This helps to understand what the code does. This is especially important for

the developers, since they might have to modify the logic or add additional features.

This project uses Doxygen to create the documentation. Doxygen is a library that can automatically generate documentation by scanning through the source file for specific comments.Furthermore, Doxygen can extract the file structure of the project. The output will be in the form of an HTML webpage or multiple LaTeX files.

1.9.1 Generate

In order to use Doxygen for this project, it should be installed on the machine beforehand. Its important to note that Doxygen will be executed through CMake. So the path to the library should be findable for CMake.

Normally, the documentation will be automatically generated when building the project. However, this can be disabled through the following command:

cmake –DENABLE_DOXY=OFF

1.9.2 Access

After Doxygen has been executed, the documentation can be found in the docs directory, located in the root directory of the project.

To access the HTML version, open index.html inside a web browser. The file is located in the html directory. Here follows a overview of the most important sections:

- **Main Page**: Contains the README.md file of the project.
- **Classes ⟩ Class List**: Shows a list of all the classes.
- **Classes ⟩ Class Members**: Shows a list of all the functions and variables of the classes.
- **Files ⟩ File List**: Overview of the project structure.

1.9.3 Modify

There are several ways for adding comments in the source code which will be detected by doxygen. Here follows the template comments used in this project.

- **Block**

```
/*
* [description]
* [more description]
*/
[entity]
```

- **Brief description (for classes)**

```
/// [description]
[entity]
```

- **One line**

```
/** [description] */
[entity]
```

- **After member**

```
[member] /**< [description] */
```

For the full documentation of Doxygen, please see: http://www.doxygen.nl/manual/docblocks.html

2 Synthetic benchmarks

To measure the maximum throughput that can be achieved on the network, the iPerf3 [4] tool was used to measure the throughput for 10 seconds. The measurement was repeated 50 times. The results show that the network throughput has a relatively low variance and is almost always consistently 94Mbit/s.

The 94Mbit of throughput was expected. Even though the Raspberry Pi has a 1Gbit interface, this is limited to 300Mbit due to sharing a USB 2.0 host controller. This limited throughput is due to the Raspberry Pis being connected via a 100Mbit switch.

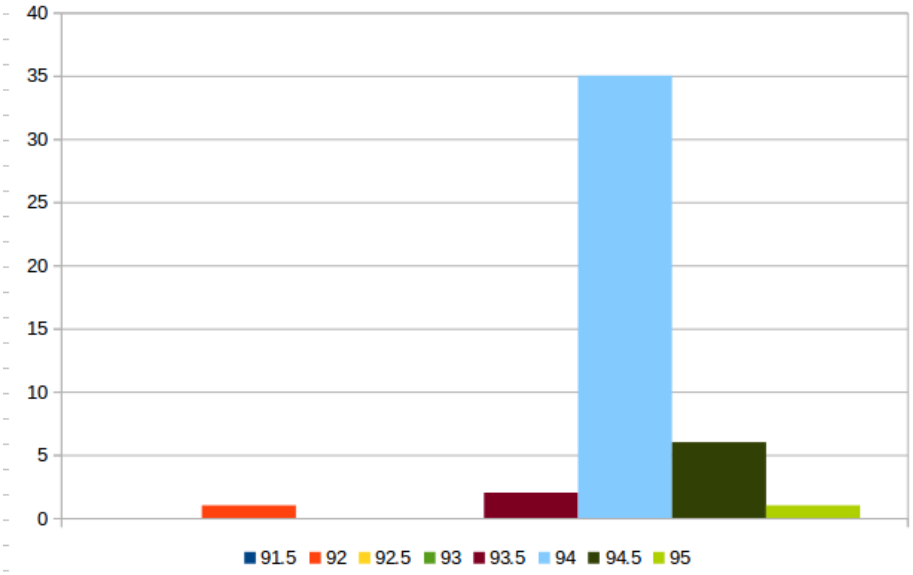


Figure 5: Distribution of synthetic benchmark throughput.

3 Research design

Before diving into the research, this paragraph describes the general problem of the research. The problem is a specific problem of CERN, but the research itself should be applicable to other similar problems.

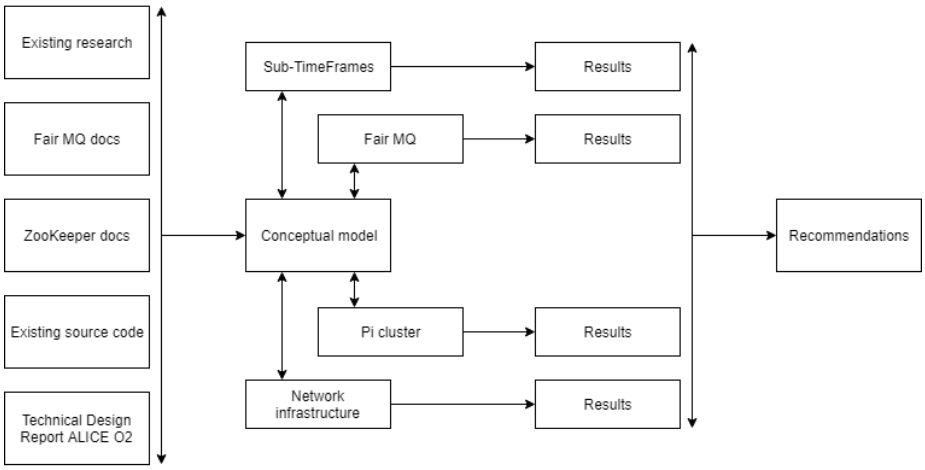


Figure 6: A Schematic representation of the research (focussed on the actual experiments).

3.1 Main Research Question

The focus of the project is to deliver a scalable, performent and autonoom load balancing system. And also to conduct experiments to gather information about the reliability of the system. At the initial state of the project we defined the main research question as following: *How do different load-balancing algorithms compare in transferring a continuous data stream across parallel many-to-one streams?*

3.2 Sub Research Question

In order to answer the main research question, a few subquestion are established to divide the whole project into smaller parts. These are the subquestions:

- Which algorithms have been developed already and what are the results?

- What are potential alternatives on which experiments can be run?
- Which metrics are available to the validate the quality of the experiments?
- How can the metrics be visualised using Kibana?

4 Experiments

Experiments are documented in 4 parts starting with a general introduction / description. After that the configuration is detailed and the necessary information is provided to revalidate the experiment. An operational overview and description is given and finally the results are measured. The experiments will only detail preliminary results and a conclusive total overview of results and how they evaluate together can be found in the results chapter.

The necessary information to revalidate an experiment can be found by using the references in the configuration section.

In the tables describing the experiments the specific results which have the same throughput as the maximum determined in the synthetic benchmark will be underlined. The following short table is an example of that underlining.

Example for underlining synthetic benchmark equivalent throughput

Rate (Hertz)	**Message size (Bytes)**	**Percentage loss (%)**
200	58000	0.00
<u>200</u>	<u>58750</u>	<u>0.49</u>

Table 9: Underlining of the synthetic benchmark equivalent allows for quickly comparing results.

4.1 Network throughput baseline

This experiments tests network throughput using the FairMQ application layers after having measured the maximum throughput that could be achieved on the specific hardware using the synthetic benchmark. The absolute maximum throughput that can be achieved is 94Mbit as demonstrated with iperf3.

The 94 Mbit throughput can be translated into a 11.75 Mbyte throughput. With the message rate of 200Hz it can be determined that the network should be able to sustain a message size of 58750 bytes. The number of FLPs and the message size was varied during this experiment.

The experiment should run for 30 seconds and is validated with tight timing constraints, the total experiment time should be no less than 29.7 seconds and no more than 30.3. Failure of a run to be within these timing constraints means it will not be used as part of the results.

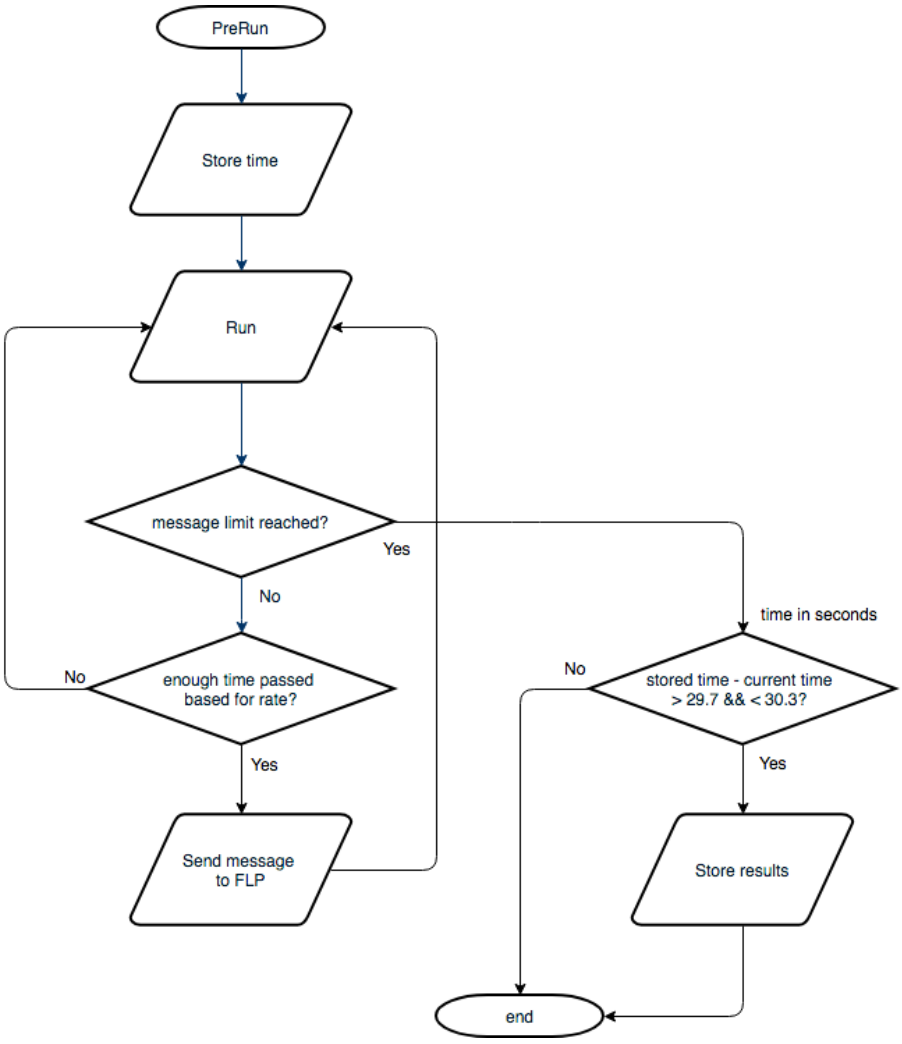


Figure 7: Flow diagram of run step during experiments

4.1.1 Configuration

Rate	200Hz	**Node**	
Messages	6000	**Type**	**Ip**
Runtime	30 seconds	ICN	192.168.1.4
Message size	Variable	FLP 1	192.168.1.3
Source commit	cbaebb9 [10]	EPN	192.168.1.2
Documentation	cdbb643 [11]	FLP 2	192.168.1.5
Arguments	appendix	FLP 3	192.168.1.6

Table 10: The specific configuration lists essential parameters as well as the specific commit referencing to the source code.

The experiment consisted of nine different configurations were both the message size and the number of FLPs in the network were varied. Each configuration was run for 30 iterations so additional statistical analysis could be performed if required.

Configuration overview

#	Message size (Bytes)	Rate (Hertz)	Number of FLPs
1	58000	200	1
2	58750	200	1
3	59500	200	1
4	29000	200	2
5	29375	200	2
6	29750	200	2
7	19333	200	3
8	19583	200	3
9	19833	200	3

Table 11: All nine configurations during the network baseline experiment.

4.1.2 Operation

The operation of the experiment involved a two step procedure, were in the first step the binaries are configured and in the second step the actual experiment is run. During this experiment the ICN its available channels are hard coded into the binary and could only be changed at compile time.

The three different core Binaries are started at the same time and the EPN & FLP will wait for incoming messages while the ICN starts counting down a ten second wait period. After the grace period the ICN will send the EPN channel configuration to all FLPs. After the message has been send the ICN will wait an additional two seconds for the FLPs to configure the newly received channels.

When the ICN has waited for the FLPs to be configured the experiment will run.

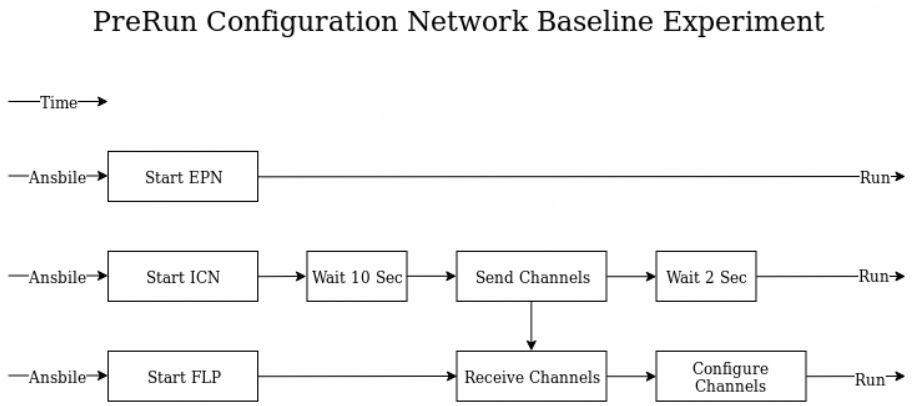


Figure 8: PreRun configuration stage depicts different operations across each type of node in time.

The network is shown for each of the possible configurations during the experiment which indicates the physical location of each node in the network. All connections between devices are 100Mbit.

Network setup

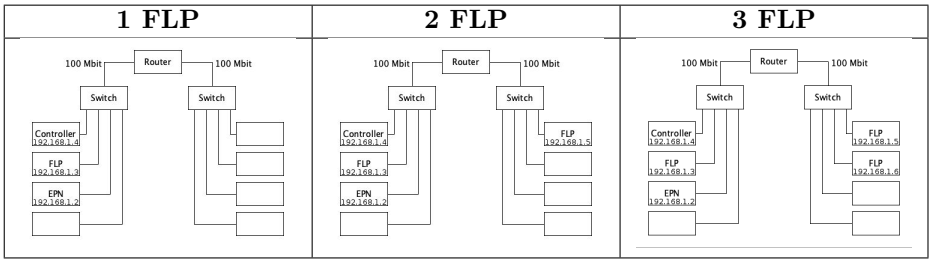


Table 12: Network diagrams for each number of FLP configurations.

4.1.3 Results

During the synthetic benchmarks the maximum network bandwidth was determined to be 94Mbit. In accordance to this synthetic throughput a maximum message size of 58750 bytes and one FLP is expected to have close to zero or zero loss. In a similar fashion a message size of 29375 bytes with two FLPs was expected to have similar loss.

During the experiment it became clear that having multiple FLPs further limits the amount of available bandwidth available for throughput between FLPs and EPNs. Attempting to sustain a throughput of 93 Mbits with more than one FLP results in significant loss of messages, furthermore, increasing the message size beyond the equivalent of 93 Mbit throughput also results in significant loss of messages. 93 Mbit is close but not equivalent of 94 Mbit this lower throughput is likely due to the network traffic from ICN to FLP & from EPN to ICN because these network connections arent present in the synthetic benchmark.

The results shows that the number of FLPs has an impact on the maximum throughput that can be achieved and that this should be taken into account in further experiments.

Mean message loss per configuration - 6000 messages at an rate of 200Hz (30 seconds)

Number of FLPs	**Message size (per FLP)**	**Mean loss (total)**	**Mean loss (%)**
1	58000	2.333333333	0.03
1	58750	42.13333333	0.70
1	59500	119.7333333	1.99
2	29000	148.3666667	2.47
2	29375	121.4333333	2.02
2	29750	120.8666667	2.01
3	19333	196.1	3.26
3	19583	198.6333333	3.31
3	19833	885.4333333	14.75

Table 13: Message loss during the network baseline with 200Hz event rate.

4.2 Round Robin ICN

This experiment tests the achievable throughput with minimal packet loss using the same algorithm as performed with previous version of the software. Each of the available EPNs is selected in turn by incrementing an iterator and returning it to the start if the last element has been accessed.

Eight nodes are configured during this experiment and the node ratio of 1:6 with regards to the FLP and EPNs is achieved. Both 50Hz and 200Hz event rates will be tested to completely cover the event operating range of the network after LS2.

Each configuration is run for ten iterations to achieve a normal distribution should advanced statistical analysis be required later. Additionally each configuration gets one iteration where the total operation time is set to three minutes instead of 30 seconds so that any variation caused by iteration time can be detected. This will also help determining if an iteration time of 30 seconds is sufficient as this has been used as iteration time in past experiments as well as

these experiments. The rationale is that the short iterations could hide potential network bottlenecks that are mitigated using network buffers, however, network buffers can only mitigate such a network bottleneck for a short duration so, as a result the longer iteration time should discover these problems.

```
/**
 * Determines the next appropriate channel the FLP should use to send data to the EPN
 * @return the desired channel
 */
uint64_t InformationControlNode::determineChannel()
{
    uint64_t index = 0;

    // currentChannel initialization
    if(currentChannel == nullptr)
    {
        currentChannel = channels.at(0);
    }

    if(currentChannel == channels.back())
    {
        // LOG(trace) << "current channel was last entry";
        currentChannel = channels.front();
    }
    else
    {
        // LOG(trace) << "Advancing channel";
        auto it = std::find(channels.begin(), channels.end(), currentChannel);
        std::advance (it, 1);
        index = std::distance(channels.begin(), it);
        currentChannel = channels.at(index);
    }

    LOG(trace) << "Chosen channel:" << currentChannel->index;
    return currentChannel->index;
}
```

Figure 9: Channel determination for round robin.

4.2.1 Configuration

Rate	50Hz / 200Hz	**Node**	
Messages	1500 / 6000	**Type**	**Ip**
Runtime	30 seconds	ICN	192.168.1.1
Message size	Variable	FLP 1	192.168.1.2
Source commit	5914f78 [10]	EPN 1 / FLP 2	192.168.1.3
Documentation	347173e [11]	EPN 2	192.168.1.4
Arguments	appendix	EPN 3	192.168.1.5
		EPN 4	192.168.1.6
		EPN 5	192.168.1.7
		EPN 6	192.168.1.8

Table 14: The specific configuration lists essential parameters as well as the specific commit referencing to the source code.

A total of fourteen different configurations were tested during this experiment. The message size, event rate and the amount of FLPs in the network were varied. The differences between these configurations should demonstrate the effects on the effective bandwidth of the network with an event rate of 200Hz.

Configuration overview

#	Message size (Bytes)	Rate (Hertz)	Number of FLPs
1	232000	50	1
2	235000	50	1
3	238000	50	1
4	58000	200	1
5	58750	200	1
6	59500	200	1
7	65000	200	1
8	116000	50	2
9	117500	50	2
10	119000	50	2
11	29000	200	2
12	29375	200	2
13	29750	200	2
14	32500	200	2

Table 15: All 14 configurations during the round robin ICN experiment.

In the results section the configuration numbers will be reused as a method to uniquely identify configurations as a result it should easier to cross-reference and get a good overview of how configurations compare.

4.2.2 Operation

The operation is similar to the network baseline experiment but additional automatic configuration has been added to simplify the process. The overall operation still consist of a two step procedure but the EPNs have additional features that allow them to report their listening channel to the ICN. This allows to ICN to maintain a list of EPNs that are available which can then be iterated over in a round robin fashion.

During the PreRun step the EPN will wait ten seconds after being started and will then proceed by sending its channel data to the ICN. This initial delay allows ICN to be started without very tight timing constraints as delays between nodes could differ in a network. The ICN is configured to be dormant until it receives the first channel. After receiving the first channel it will allow for additional channels to be added during a ten second period. After this period it will forward all the channels to the FLPs in the network and it will wait two seconds before continuing. During this two second period the FLPs will have time to configure their channels. With the FLP their channels configured the PreRun step is finished and the run step is entered.

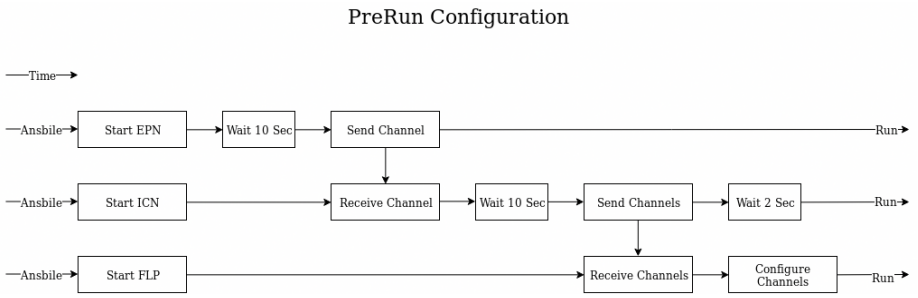


Figure 10: PreRun configuration stage depicts different operations across each type of node in time.

During the run step the ICN will monitor the rate of transmission. Whenever the FLP should send the next message to the currently selected EPN the ICN will send an message to the FLP. This message will contain an heartbeat ID as well as the selected EPN that should receive the message. The ICN will then increment the list of EPNs as a means to select the next one for the following message and the FLPs will send the configured amount of bytes to the EPN. When the EPN has successfully received the messages from the FLPs it will send an acknowledgement to the ICN. Should an EPN receive to messages with different heartbeat IDs it will discard the messages and trigger an out-of-order error as a result it will not send an acknowledgement to the ICN for that heartbeat.

This run step procedure allows to discriminate between messages that were lost due to high bandwidth & network buffers or due to timing errors. The expectation is that the increased event rate of 200Hz will increase the amount of out-of-order / timing errors with an equivalent total bandwidth.

To more correctly measure the effectiveness of the round robin algorithm at 50 and 200Hz all network communication between ICN to FLP and EPN to ICN will be transmitted on the auxiliary networking interface.

Network setup

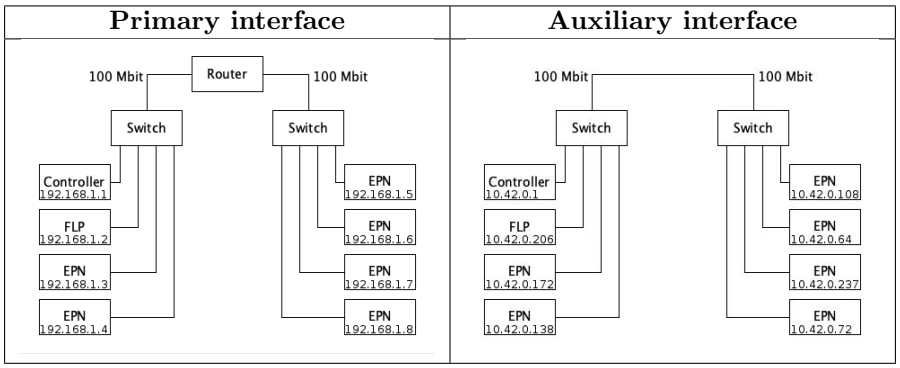


Table 16: Network interfaces that are available to nodes.

4.2.3 Results

Losses across different configurations were there is one FLP in the network are extremely similar to the network throughput baseline experiment. This is likely due to the inability to cause out-of-order errors as there is only a single FLP in the network.

When introducing the second FLP into the network losses start to occur even at lower throughput than the synthetic benchmark.

Mean message loss - 1500 messages sent at an rate of 50hz (30 seconds)

#	Number of FLPs	Message size (per FLP)	Mean loss (total)	Mean loss (%)
1	1	232000	0	0.00
2	1	235000	0	0.00
3	1	238000	0	0.00
8	2	116000	7.44	0.50
9	2	117500	5.11	0.34
10	2	119000	11.11	0.74

Table 17: Comparing loss of configurations with an 50Hz event rate.

The losses with two FLPs are higher with an event rate of 200Hz all of the losses were due to out-of-order errors directly showing that the tighter timing constraints increase the amount of errors that occur at the same bandwidth.

Mean message loss - 6000 messages sent at an rate of 200hz (30 seconds)

#	Number of FLPs	Message size (per FLP)	Mean loss (total)	Mean loss (%)
4	1	58000	0	0.00
5	1	58750	0	0.00
6	1	59650	0	0.00
7	1	65000	441.89	7.35
11	2	29000	30.56	0.51
12	2	29375	52.67	0.87
13	2	29750	53.34	0.87
14	2	32500	85.56	1.42

Table 18: Comparing loss of configurations with an 200Hz event rate.

4.3 Comparing long & short runs

Similar configurations should result in similar loss even if the duration is varied, on the condition that the duration of the test is sufficient to correctly measure the desired characteristics. The results from comparing short & long runs, however, show that there is a measurable difference. The results highlighted in red have a higher long run percentage loss than the short run equivalent, while the results highlighted in green have a lower long run percentage loss over the short run.

Comparing losses of 30 second (short) & 3 minute (long) runsw

#	Mean loss short-run (%)	Mean loss long-run (%)
1	0.00	0.00
2	0.00	0.00
3	0.00	0.27
4	0.00	0.00
5	0.00	0.00
6	0.00	1.23
7	7.35	9.63
8	0.50	0.00
9	0.34	0.00
10	0.74	0.01
11	0.51	0.06
12	0.88	0.01
13	0.89	0.06
14	1.39	0.21

Table 19: Effects of different iteration times on the percentage of loss during experiments.

In all the green highlighted configurations there were two FLPs in the network moreover all the losses were due to so called out-of-order errors. An out-of-order errors occurs when an EPN receives two messages from the FLPs but each one with a different heartbeat, as a result the EPN invalidates the data and discards it so it can continue receiving new messages. Out-of-order errors are more likely to occur with tighter timing constraints or with fewer EPNs in the network. The lower percentage loss on long runs might indicate that most of the out-of-order events occur during the beginning of the experiment and eventually settle out or at least become less likely to occur.

In all the red highlighted configurations the total size of accumulated messages multiplied by the rate of events exceeded the determined maximum throughput as determined in the synthetic benchmark. The results show that in the short run exceeded these network limits does not result in detected losses, however, in the long runs this does result in losses. This is likely due to network buffers which are temporarily capable of resolving the exceeded limits.

5 O2Balancer2 Zookeeper version

The second system we provide is the round-robin whitelist loadbalancer, using Zookeeper [12] for node registration and distribution. This new implementation offers a robust system: the order of which FLP/EPN/ICN starts-up or shuts-down has no impact on the system. With a failover of any of the three nodes the system will continue working: after the node comes back online, it will be part of the system again.

It is imported to note that the ICN is much different in this system than in the O2Balancer2 ICN version. The ICN in this implementation **only** sends heartbeats to the FLPs.

The whitelist algorithm has a different approach than the previous version [13]. The previous version overrides a fairmq class to allow for removing a channel from the channel list contained in MQProgOptions. The new version solves this by maintaining a list of the online nodes so it is known which channels not to use. This allows for significant faster reinitializing of the node, since the channel doesnt have to be broken down, instead the channels stay unused until the node comes back online in which case the node will be added to the list of online nodes.

Multiple small improvements have been made, such as: no overriding of fairmq classes(so that is compatible to further versions), FLP only retrieves zk values of new EPNs (instead of all EPNs with each watcher trigger), uses watchers, can detect restarted nodes, relieved network load by using direct channels from FLP to ICN(instead of using zookeeper), and finally a simpler file structure.

The software can be found on the zookeepertje [12] branch from the O2Balancer2 repo. Unfortunately there are no experiments conducted with this zookeeper implementation due to time constraints.

5.1 ZooKeeper throughput experiment

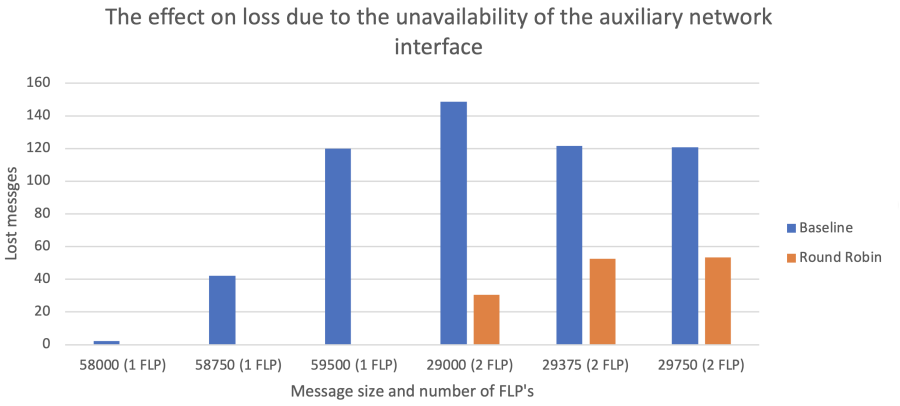
The round-robin whitelist implementation using Zookeeper has the following throughput:

- Using packet size: 232000 bytes
- 2 flps - 5 epns - 60 tfps - measured input: 18 packets (3,67MB per epn = 18,8 MB/s total)
- 1 flps - 5 epns - 60 tfps - measured input: 12,5 packets (2,16 MB per epn = 10,8 MB/s total)

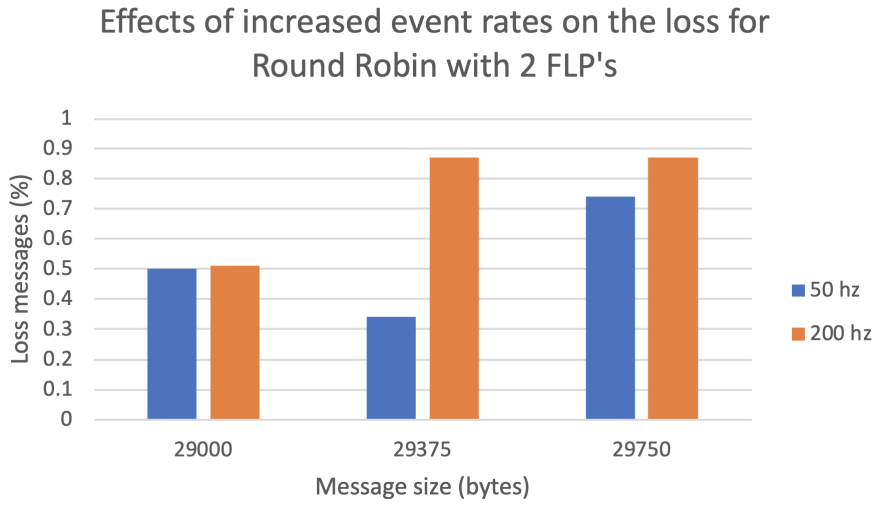
6 Results

In this section the results from different experiments are compared as well as data from the synthetic benchmark. The information will be used to draw conclusion and recommend further work.

During the synthetic benchmark the network was able to sustain a throughput of 94Mbit while during the network baseline only 93Mbit could be achieved with an single FLP, however the network baseline only used the primary networking interface for all data and ran at an event rate of 200Hz. The additional results from the round robin ICN experiment show that the 94Mbit throughput can be sustained at both 50 & 200Hz when using the auxiliary networking interface for communication between ICN to FLP and EPN to ICN.

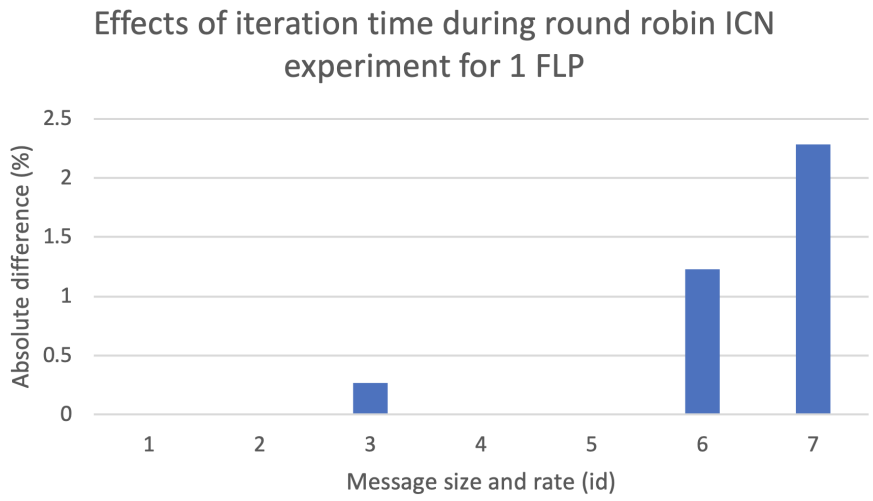


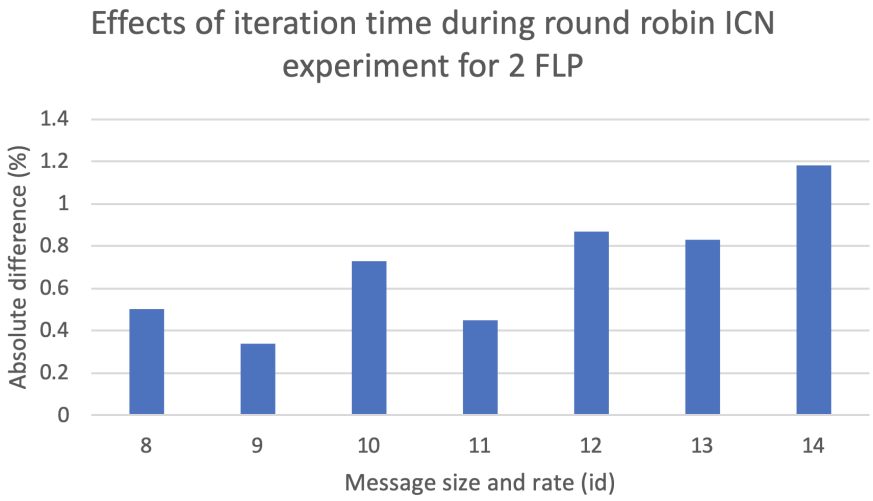
This sustained throughput of 94Mbit was only possible with a single FLP in the network while with two FLPs there were still losses at 93Mbit throughput, although these losses were around 0.5% for both 50 & 200Hz event rates. The experiments show significant data that the 200Hz event rate could be sustained by an round robin algorithm with only a small reduction in network throughput.



Finally the variation of iteration time from 30 seconds to 3 minutes has shown that this can have an effect of the loss that will be measured during the

experiment. The total losses measured for both 30 second and 3 minute iterations are relatively low (always below 10% and regularly below 3%), however, the difference between the two iteration times can be as much as 650%. This is an substantial difference that shows that other iteration times than 30 seconds should be considered in upcoming experiments.





6.1 Metricbeat

During the round robin ICN experiments metricbeat was used on the raspberry pi nodes to record metric data, unfortunately the results from this data proved unusable due to unreliability as well as the metricbeat process requiring a lot of system resources. The metricbeat process occasionally caused 65% CPU utilization, additionally many graphs tended to be, making it hard to correlate them. Many of the nodes stopped correctly reporting metrics after a period of time which required the restarting of the metricbeat process to be resolved.

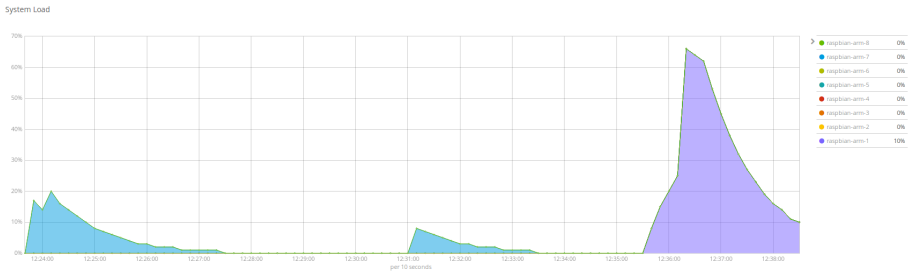


Figure 11: Graph depicting the occasional 65% CPU utilization which was determined to be metricbeat using tools as Htop.

7 Conclusion

200Hz event rates should be manageable with only a small reduction in throughput when using the round robin algorithm, additional experiments with more nodes are required.

Metricbeat could be useful but it requires a lot of effort to be correctly configured in combination with Logstash, Elasticsearch & Kibana. However, it might be unfeasible to run this service on ARM based nodes (such as Raspberry PIs) and additional work is required to determine if it is feasible.

Future experiments should take multiple iteration times into account as this parameter can affect results. Perhaps previous experiments should be repeated with different iteration times.

8 Discussion

Additional experiments with more nodes are required to better determine the effects of the round robin algorithm at 200Hz event rate. During this experiments, additional metrics would be of great value, but the overall feasibility of metricbeats and the entire MELK stack needs to be properly evaluated, especially for the ARM platform. In addition to the increased amount of nodes a larger

variety of iteration times could also benefit the results. An important aspect of the zookeeper system that could be tested in a further study is whether all the flps send all the subtime frames of a heartbeat to the same epn, and if they keep doing that with a prolonged run-time (1 hour+). Also a reliability comparison between the two systems used (Zookeeper vs ICN) would be interesting.

9 Appendix

9.1 Commands to replicate network throughput baseline

XXXXXX = Message size
YYY = Number of FLPs in the network

9.1.1 Commands for experiment with single network card

```
./icn/icn --severity trace --verbosity high --id 1 --
    iterations 6000 --rate 200 --channel-config name=
    broadcast,type=pub,method=bind,rateLogging=0,address=
    tcp://*:5005 name=feedback,type=pull,method=bind,
    rateLogging=0,address=tcp://*:5000

./flp/flp --severity trace --verbosity high --id 1 --
    bytes-per-message XXXXXX --channel-config name=
    broadcast,type=sub,method=connect,rateLogging=1,
    address=tcp://192.168.1.4:5005

./epn/epn --severity trace --verbosity high --id 1 --num-
    flp YYY --channel-config name=1,type=pull,method=bind,
    address=tcp://*:5555,rateLogging=1 name=feedback,type=
    push,method=connect,address=tcp://192.168.1.4:5000
```

9.1.2 Commands for experiment with dual network cards

```
./icn/icn --severity trace --verbosity high --id 1 --
    iterations 6000 --rate 200 --channel-config name=
    broadcast,type=pub,method=bind,rateLogging=0,address=
    tcp://*:5005 name=feedback,type=pull,method=bind,
    rateLogging=0,address=tcp://*:5000

./flp/flp --severity trace --verbosity high --id 1 --
    bytes-per-message XXXXXX --channel-config name=
    broadcast,type=sub,method=connect,rateLogging=1,
    address=tcp://10.42.0.138:5005
```

```
./epn/epn --severity trace --verbosity high --id 1 --num-
    flp YYY --channel-config name=1,type=pull,method=bind,
    address=tcp://*:5555,rateLogging=1 name=feedback,type=
    push,method=connect,address=tcp://10.42.0.138:5000
```

9.2 Commands to replicate round robin

XXXXXX = Message size
YYY = Rate of the experiment
ZZZ = Number of iterations
WWW = EPN number

9.2.1 Build configuration

```
cmake -DCMAKE_BUILD_TYPE=Release -DENABLE_DOXY=false ..
make
make test
```

9.2.2 1 FLP

```
./icn/icn --control static --severity trace --verbosity
    high --id 1 --iterations ZZZ --rate YYY --channel-
    config name=broadcast,type=pub,method=bind,rateLogging
    =1,address=tcp://*:5005 name=feedback,type=pull,method
    =bind,rateLogging=1,address=tcp://*:5000
```

```
./flp/flp --severity trace --verbosity high --id 1 --
    bytes-per-message XXXXXX --channel-config name=
    broadcast,type=sub,method=connect,rateLogging=1,
    address=tcp://10.42.0.1:5005
```

```
./epn/epn --control static --severity trace --verbosity
    high --id WWW --primary-interface eth0 --num-flp 1 --
    channel-config name=WWW,type=pull,method=bind,address=
    tcp://*:5555,rateLogging=1 name=feedback,type=push,
    method=connect,address=tcp://10.42.0.1:5000
```

9.2.3 2 FLP

```
./icn/icn --control static --severity trace --verbosity
    high --id 1 --iterations ZZZ --rate YYY --channel-
    config name=broadcast,type=pub,method=bind,rateLogging
    =1,address=tcp://*:5005 name=feedback,type=pull,method
    =bind,rateLogging=1,address=tcp://*:5000
```

```
./flp/flp --severity trace --verbosity high --id 1 --
    bytes-per-message XXXXXX --channel-config name=
    broadcast,type=sub,method=connect,rateLogging=1,
    address=tcp://10.42.0.1:5005
```

```
./epn/epn --control static --severity trace --verbosity
    high --id WWW --primary-interface eth0 --num-flp 2 --
    channel-config name=WWW,type=pull,method=bind,address=
    tcp://*:5555,rateLogging=1 name=feedback,type=push,
    method=connect,address=tcp://10.42.0.1:5000
```

9.2.4 Commands for experiment with round-robin whitelist Zookeeper

git checkout zookeepertje

```
node 1:
sudo /usr/share/zookeeper/bin/zkServer.sh start
./icn --severity trace --verbosity high --id 1 --
    iterations 6000 --rate 200 --channel-config name=
    broadcast,type=pub,method=bind,rateLogging=0,address=
    tcp://*:5005

node 2 and 5:
./flp --severity trace --verbosity high --id 1 --channel-
    config name=broadcast,type=sub,method=connect,
    rateLogging=1,address=tcp://<ip address of icn>:5005

node 3,4,6,7 and 8
./epn --severity trace --verbosity high --id 1 --channel-
    config name=1,type=pull,method=bind,address=tcp://<own
     ip address>:5555,rateLogging=1
```