Technical Design Report

Software for Science - CERN Load Balancing

Team 6 Hexoxide
Eloy Maduro Clement
Geoffrey van Driessel
Corne Lukken
Bram Tukker

January 16, 2019
v1.0

Changelog

Version	Date	Description
0.2	07-11-2018	1. Added information on changing to debian image. 2. Added information about investigating metricbeats 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 Remove outdated or incoherent information. 2.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.

Table 1: Hardware specifications of computing nodes.

Contents

1 Introduction

This document reflects the discovered & analysed 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 there setup are detailed so they can be repeated easily in the future as well as instructions how to install all required components.

2 Legacy software

Disclaimer: The legacy software has never reached an operational stage in the duration of 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 legacy software to run experiments.

At the start of this project numerous previous experiments have been done on this subject. These previous experiments have led to the creation of existing software. It is important to identify how these softwares worked and references 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**	**Reference**	**Commit**
O2Balancer	Binaries used in experiments containing specific algorithms to be tested.	O2 [1]	36eec89
BalancerScripts	Setting up nodes with ansible and orchestrating an experiment. Gathering and purging log files and creating plots and graphs from log data.	Scripts [2]	99f7177

Table 2: TODO

Newer software will likely still be distributed through git, but will be made available through the software-for-science github organisation. The repositories from previous experiments can now be on the organisation project. See this link for the page: https://github.com/SoftwareForScience/O2-Balancer

3 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 before. The documentation on how to install & run the O2Balancer for the raspbian Linux distribution has been lost unfortunately. The installation instructions for CentOS can be found online on: https://github.com/SoftwareForScience/O2-Balancer/blob/master/Docs/Compilation.md

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).

3.1 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
```

3.2 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.

InformationNode

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	?

Table 3: TODO

FLP

Parameter	Data type	Description
–flp-config	string	Path to yaml configuration file.
–restartFairRoot	boolean	?
–ip	string	?

Table 4: TODO

EPN

Parameter	Data type	Description
–epn-config	string	Path to yaml configuration file.
–amount-flps	int	Amount of FLP's 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	?

Table 5: TODO

The execute.sh script will open a total of 6 windows which should appear, each running a different type of node software. In total there should be 3 EPN's, 2 FLP's & 1 InformationNode.

4 Hardware Specifications

According to the A Large Ion Collider Experiment (ALICE) Technical Design Report (TDR) [1], there will be two types of computing nodes in the O2 project after Long Shutdown 2 (LS2). In total there will be about 2000 nodes in O2, 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 O2 computing network. The computing devices at AUAS will be Raspberry Pi's, specifically, model 3 B+.

In table 1 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 O2. 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 6: Hardware specifications of computing nodes.

5 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 mea-

surement was repeated 50 times. The results show that the network throughput has a relatively low variance and is almost always consistently 94Mbit/s.

This 94Mbit of throughput even though the Raspberry Pi has a 1Gbit interface limited to 300Mbit due to sharing an USB 2.0 host controller was expected. This limited throughput is due to the Raspberry Pi's being connected via a 100Mbit switch.

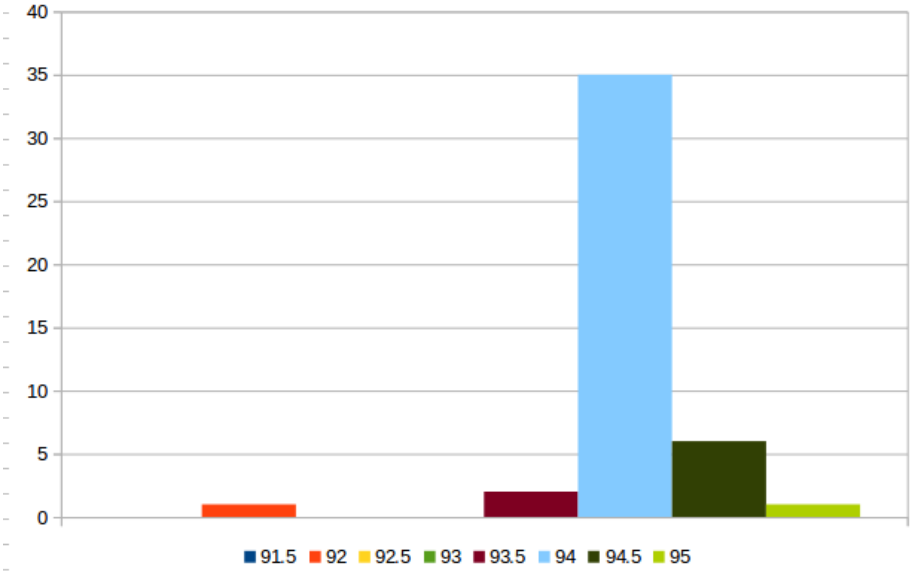


Figure 1: TODO

6 Hardware & software setup

The pi clusters used in experiments were initially empty. So the 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.

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.

6.1 Network

The network during the experiments consisted of 8 raspberry-pi nodes each with two network interfaces. One of these interface 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 turned out to be incompatible with ARM based processors. An virtual machine running on an X86 based host was configured to resolve any ARM related 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-pi's from the rest of the internal network. Each of the firewalls indicate the NAT that occurs on two places in the network and this will require port forwarding and tunneling later.

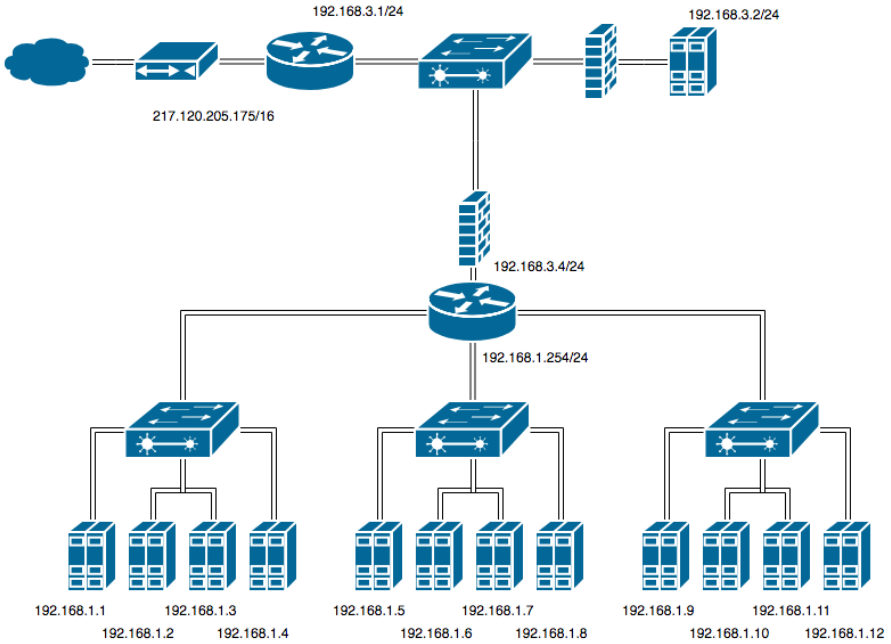


Figure 2: Networking diagram with auxiliary networking interfaces omitted. A diagram with the auxiliary interface can be found online [5].

6.2 New software introduction

An environment as similar as possible to the one used at CERN is desired, however, it has proven to be not feasible to use CentOS. This is 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 of gcc version, normally, a tool called scl provides the switching for specific versions of many development tools. The version of gcc supplied with CentOS is incompatible with the version of boost that is specified in many of the previous experiments. Furthermore the version of gcc is not capable of compiling c++2011 features which is required by CERN. To continue to use CentOS gcc would have to be built to be build from source.

To resolve the issues with the CernOS image an alternative had to be elected. Based on previous experience Manjaro 17.1 was chosen, Manjaro provides a full features ARM image for which a tremendous amount of packages are available. 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 even this eventually led to great trouble, which included: Broken packages, unavailable mirrors and missing kernel modules. As an alternative Raspbian stretch was chosen because of its tremendous 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 legacy software was determined to be partially incorrect as it does not use FairMQ itself but instead relies on FairRoot.

Library/Tool	**Version used**	**Version Legacy Software**
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	1.66.0	1.66.0
Yaml-cpp	NONE	0.5.2
FairLogger	1.3.0	NONE
Compiler	gcc 6.3.0	gcc 6.3.0

Table 7: The installed software dependencies on the pi's.

6.3 New software & system components

The system consist of different nodes all of which are running Debian 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.

Repository	Purpose
raspberry-dependency [6]	Detailed instructions listing required Debian packages and automated install script to install further dependencies not installed through package manager.
BalancerScripts2 [7]	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 [8]	Core binaries which run the experiments by configuring and sending data on different channels across the network.
Documentation [9]	Essential documentation on the overall project as well as how to get started with the continuation of our project.

Table 8: TODO

6.4 Installation

To ensure the reproducibility of experiments executed by the new software, a bash script [6] has been developed which will install all the dependencies automatically. 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.

6.5 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 needs to be forwarded to the outside network. Additionally an SSH key from the portal node can be added to every node 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 an web based dashboard running on the X86 based virtual machine.

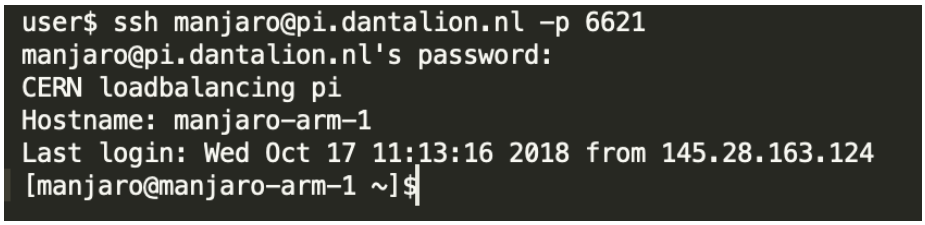


Figure 3: Example of login in using SSH on remote accessible raspberry pi.

6.6 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.

6.7 MELK

Metricbeat can be used to gather statistics of the nodes. This data can be send 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.

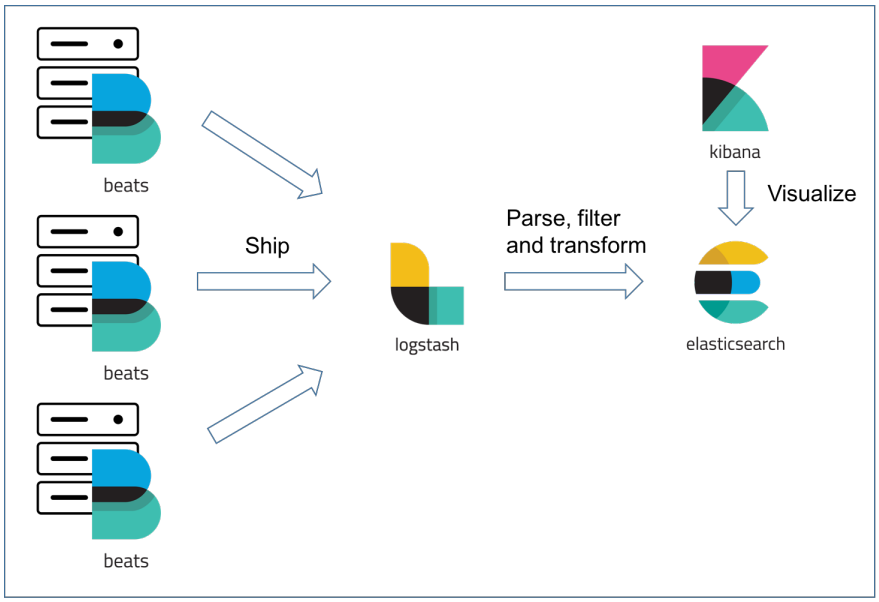


Figure 4: Overview of MELK stack.

6.7.1 Logging metrics

The decision to use Metricbeat was heavily based on it's 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 9: Metric that will be used in the experiments.

7 Source Code Documentation

Beside the regular documentation, the source code itself can also be documented. This helps understanding what the code does. This 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 a HTML webpage or multiple LaTeX files.

7.1 Generate

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

7.2 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 location in the 'html' directory. Here follows a overview of the most important sections:

- **Main Page**: Contains the REAME.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.

7.3 Modify

There are several ways to put comment in the source code. 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 on this topic, please see: http://www.doxygen.nl/manual/docblocks.html

8 Research design

The research design describes the approach of the experiments.

8.1 Problem Statement

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 across other similar problems.

8.1.1 ALICE

ALICE, which stands for A Large Ion Collider Experiment, is one of many detectors mounted on the Large Hadron Collider (LHC) at CERN. ALICE is used to study states of matter in extreme energy densities. Specifically, a state of matter known as quark-gluon plasma. This state is achieved by shooting two lead ions against each other inside the LHC. This results in temperatures over 100,000 times the center of the sun.

8.1.2 Problem

ALICE will be upgraded their systems during the Long Shutdown 2 (LS2). This upgrade will significantly increase the data gathered. It is estimated that the collection of data will increase by a factor of 100, which roughly translates to 1.1 TB/s of continuous throughput.

This upgrade also comes with an upgrade of the computing infrastructure capable of handling the increase in data collection; i.e. increasing and improving the amount of First Level Processors (FLPs) and Event Processing Nodes (EPNs), as well as the networking infrastructure. The computing upgrade will need to be accompanied by software to efficiently handle load balancing.

8.2 Research Framework

The research framework below shows the resources (existing research, et cetera) used to construct the conceptual model with which the objects of research (Sub-TimeFrames, Fair MQ, et cetera) were put to the test. Results of those test were finally gathered and used to make recommendations.

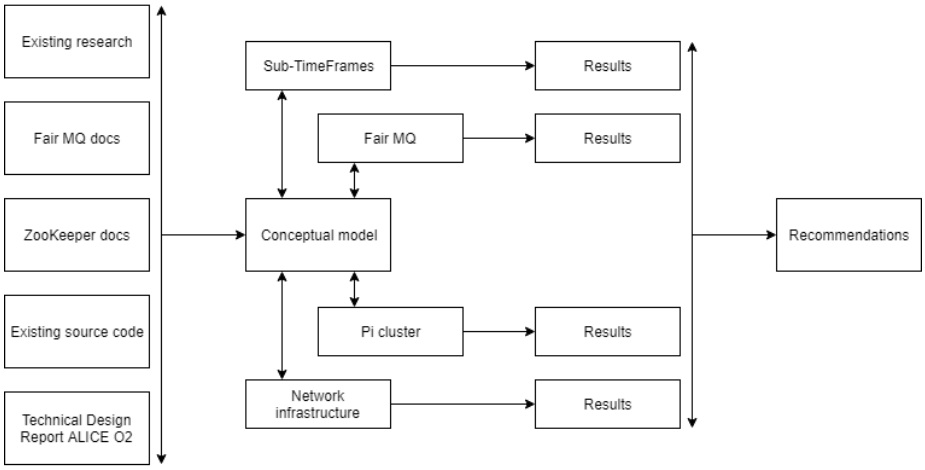


Figure 5: A Schematic representation of the research (focussed on the actual experiments).

8.3 Main Research Question

Based on the problem statement and the research framework, a main research question can be established. The focus is on trying new approaches for the load balancer, with the goal of improving the overall performance and usability. A comparison will be made between these approaches to give an overview of the advantages and disadvantages. That's why the main research question is defined as follows: *"How do different load-balancing algorithms compare in transferring a continuous data stream across parallel many-to-one streams?"*

8.4 Sub Research Question

In order to answer the main research question, a few sub question are established to divide the whole research into smaller parts. This will be handy, especially for the different algorithms that will be used for each approach. These are the sub questions:

- 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?

9 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 10: Underlining of the synthetic benchmark equivalent allows for quickly comparing results.

9.1 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 FLP's 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.

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

The experiment consisted of 9 different configurations were both the message size and the number of FLP's in the network were varied. Each configuration was run for 30 times so additional statistical analysis could be performed if required.

9.1.1 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 it's available channels are hardcoded 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 10 second wait period. After the grace period the ICN will send the EPN channel configuration to all FLP's. After the message has been send the ICN will wait an additional 2 seconds for the FLP's to configure the newly received channels. When the ICN has waited for the FLP's to be configured the experiment will run.

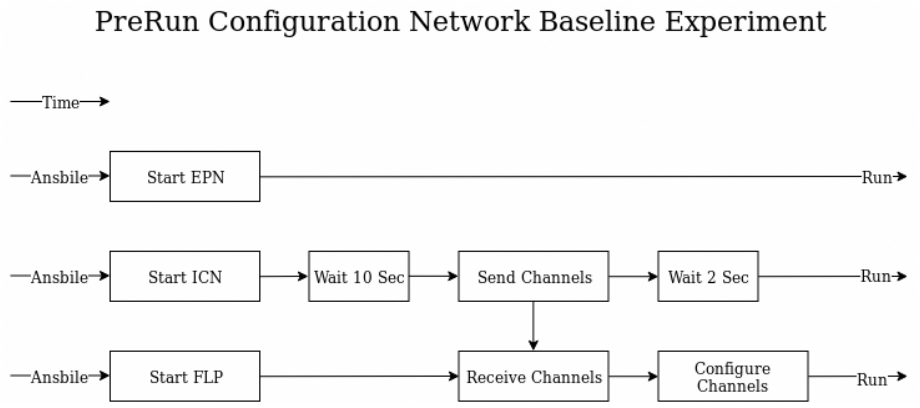


Figure 6: PreRun configuration stage depicts different operations across each type of node in time.

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

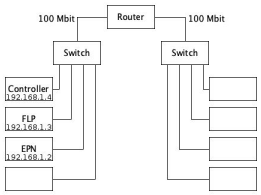
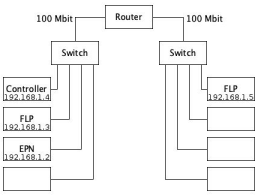
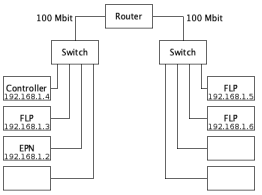
1 FLP	2 FLP	3 FLP
		

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

9.1.2 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 1 FLP is expected to have close to zero or zero loss. In a similar fashion a message size of 29375 bytes with 2 FLP's was expected to have similar loss.

During the experiment it became clear that having multiple FLP's further limits the amount of available bandwidth available for throughput between FLP's and EPN's. 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 aren't present in the synthetic benchmark.

The results shows that the number of FLP's 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 sent

Number of FLP's	**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.

9.2 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 EPN's is selected in turn by incrementing an iterator and returning it to the start if the last element has been accessed.

8 Node's are configured during this experiment and the AliceO2 ratio of 1:6 with regards to the FLP and EPN's 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 ran for 10 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 3 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 7: Channel determination for round robin.

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 14 different configurations were tested during this experiment. The message size, event rate and the amount of FLP's 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 FLP's
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.

9.2.1 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 EPN's have additional features that allow them to report their listening channel to the ICN. This allows to ICN to maintain a list of EPN's 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 FLP's in the network and it will wait two seconds before continuing. During this two second period the FLP's will have time to configure their channels. With the FLP their channels configured the PreRun step is finished and the run step is entered.

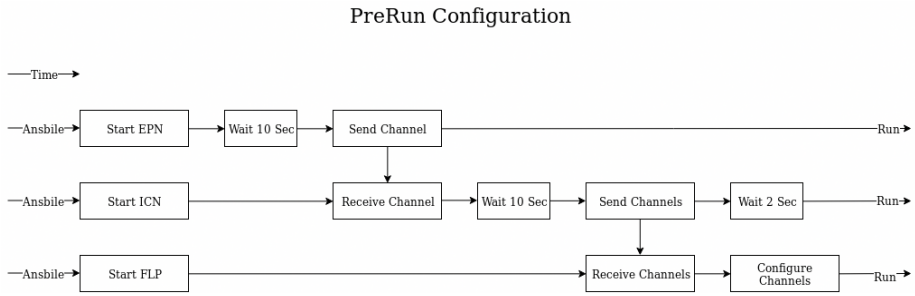


Figure 8: PreRun configuration stage depicts different operations across each type of node in time.

Acronyms

ALICE A Large Ion Collider Experiment. 6

AUAS Amsterdam University of Applied Sciences. 6

LS2 Long Shutdown 2. 6

SSH Secure Shell. 10

TDR Technical Design Report. 6

References

[1] O2-Balancer

[2] Balancer Scripts

[3] ALICE Technical Design Report

[4] iPerf

[5] Networking diagram with auxiliary interface

[6] Raspberry Dependency

[7] Balancer Scripts 2

[8] O2-Balancer 2

[9] Documentation

[10] O2-Balancer 2 Commits

[11] Documentation cdbb643

10 Appendix

10.1 Commands to replicate network throughput baseline