Master thesis
Information Science

Radboud University

Quality of Service in a cloud native environment

Author:
Guy Versteeg
S4378857

First supervisor/assessor:
Dr. C.L.M. Kop
c.kop@cs.ru.nl

Second supervisor:
R. IJpelaar
remco.ijpelaar@
nl.thalesgroup.com

Second assessor:
Prof. dr. ir. A.P. de Vries
a.devries@cs.ru.nl

September 15, 2022

Abstract

More and more companies see the added value that computing in a cloud native environment can bring. As such Thales is looking for ways to incorporate cloud native computing in their products. Thales develops combat management systems for navy ships. Therefore, a high certainty of data delivery within the ship is necessary. In the current infrastructure used in these products, quality of service assurance is implemented. This study focuses on how quality of service can be safeguarded in a cloud native environment. We tried to apply tools that can be used for safeguarding quality of service in non-cloud native environments, in a cloud native environment. During this study we found that currently there exists no Container Network Interface (CNI) that allows us to make quality of service assurances within a cloud native environment.

Contents

Chapter 1

1 Introduction

Thales is an organization that has multiple business lines in the Netherlands for which they provide services. These business lines are cyber security, defense and transportation. Within this thesis we will put a focus on the defense business line. One of the products Thales offers in the naval domain is Thales' combat management system named Tacticos, which can connect to almost every sensor and/or weaponry. Another product line that Thales offers is all about sensory equipment, such as radars with various ranges of distance that can be installed and work together with Tacticos.

Thales is going to renew the infrastructure of their Combat Management System Tacticos and will move towards an infrastructure consisting of distributed microservices, governed with Kubernetes. The decision towards this new infrastructure is the amount of overhead on physical resources, among other things. On this infrastructure multiple levels of network traffic exist, ranging from mission critical data, such as radar tracks, to low priority data, such as back-ups.

1.1 Research Questions

In this study we focus on the question how quality of service can be safeguarded in a cloud infrastructure consisting of distributed microservices with Kubernetes. First, we look at the existing literature, and we answer what quality of service entails. Next, we look at ways of safeguarding quality of service in the current infrastructure. We then look at what's different in a cloud native environment and subsequently try to apply the existing tools in a cloud native environment. Our main problem is the following:

"How can quality of service be safeguarded within a cloud native environment?"

This research is split in three different parts, namely:

1. Literature study
2. Analysis of current technology
3. Experimentation

To answer the main problem we have defined the following sub-questions:

1. **Literature study**
   - 1.1. **How can quality of service be best described?**
     Quality of service is mentioned in many research papers and different perspectives on quality of service exist. By answering this question, we will show what views exist and which views are relevant for this research. After having taken all these views into account we will formalize a definition which will be used during the rest of the research.
   - 1.2. **How is quality of service safeguarded in a conventional network?**
     With this question we will investigate what tools currently exist to ensure quality of service in conventional network solutions. By answering this question, we also research if some of these applications can be mapped onto IT infrastructures that are built in cloud native environments.
   - 1.3. **What are the shortcomings or barriers of quality of service in a conventional network?**
     For us to present a decent solution to the research question or main problem, we must first note what exactly is not up to par within the current solutions that aim to provide quality of service.
2. **Analysis of cloud technology**
   - 2.1. **How is quality of service safeguarded in a cloud-native environment with microservices and software defined networking?**
     By answering this question we define a baseline of knowledge on how quality of service can be ensured in a cloud native environment.

This means that we compare information from this question with the answers to question 1.2 so that we know what the essential differences between the two environments are.

2.2. How can different Kubernetes pods be connected to enable quality of service?

A multitude of network tools exist for container orchestration software, such as Kubernetes. These network tools have different implementations and as such they also have multiple ways of being connected with each other. By answering this question, we will lay a foundation on how different tools can be implemented within a cloud native environment.

3. Experimentation

3.1. How do different tools that enable quality of service, differ from each other in a test environment?

At this stage of the research, we compare how the different network plugins that we found in question 2.2 perform in different settings and test environments. With the outcome of these tests, we will develop a model on how quality of service can be safeguarded.

1.2 Contributions

With this thesis we aim to provide the following contributions:

- An analysis of what is currently available for safeguarding quality of service.
- A clear distinction between how conventional IT infrastructures and cloud native infrastructures are built.
- An overview of how quality of service can be safeguarded in a cloud native environment and current barriers and limitations.
- Support for Thales in the development of their new naval combat management system.

1.3 Outline

As mentioned earlier, this study is split into four main parts. Apart from these three parts we discuss background information relevant to the study in chapter 2. The first part focuses on the literature study and can be found in chapter 3. In chapter 3 we form a definition of quality of service. Afterwards, we discuss how quality of service can be enabled in a conventional (non-cloud) environment and why this environment doesn't suffice anymore. In chapter 4 we investigate what technology currently exists in regards with quality of service and a cloud native environment.

Following the analysis of the current state of technology, we put these methods to the test in chapter 5. This entails testing multiple environments and tools in such a way that we can draw meaningful conclusions from this data.

In chapter 6 we discuss studies related to this subject. Finally, we present our recommendations for safeguarding quality of service in a cloud native environment in chapter 7 and reach a conclusion in chapter 8.

Chapter 2

2 Background

In this chapter we will give some background information on the research topic. This information will give some context on the terminology used in this research thesis.

2.1 Containers

Isolation of applications in conventional infrastructures is done by running each application on their own virtual machine(s). A virtual machine can be seen as a virtualized computer or server. This virtualization is done by a so-called hypervisor, which runs directly on the infrastructure. These virtual machines have a set amount of virtual CPU and RAM, but most of the time these resources aren't being used to the fullest by the application on the virtual machine. By running applications on their own virtual machine, we create a huge overhead on the available resources (Scheepers, 2014).

This overhead is due to two main reasons. First, we see that each virtual machine gets its own piece of hardware from the infrastructure to use. This means that in case the virtual machine does not fully use its capacity these resources cannot be utilized by another virtual machine. The second reason that tells us why overhead increases is that every virtual machine has its own complete operating system on which the application runs.

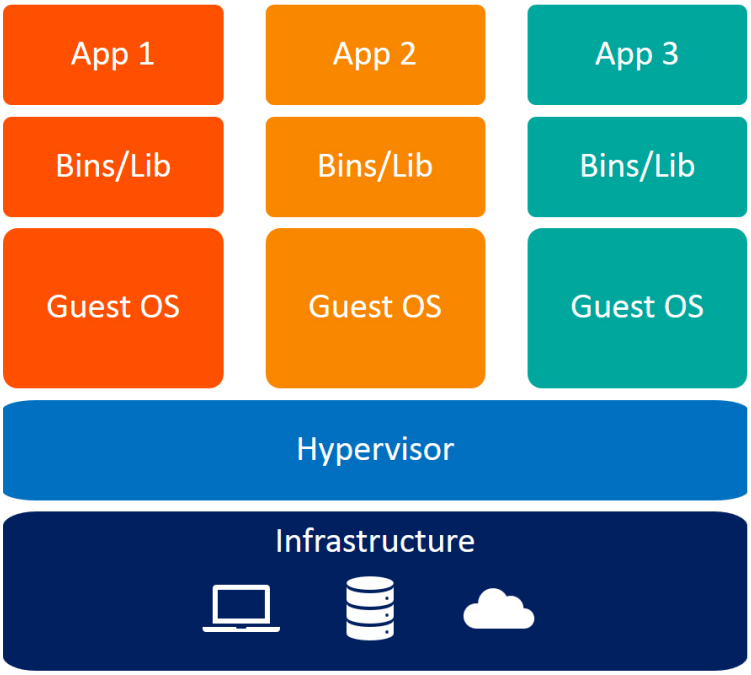

Figure 1 - Infrastructure with virtual machines (Raza & Kidd, 2020)

Containers are similar to the virtual machines being used already, but they are built upon a more lightweight concept. Containers use a lightweight version of the operating system and application specific resources. This reduces the number of resources needed for each virtualized application and thus reduces overhead. Pahl (2015) also notes that there is a difference between the intention of containers and virtual machines. Virtual machines mostly speak of the allocation of hardware, while containers are mostly a tool for delivering applications.

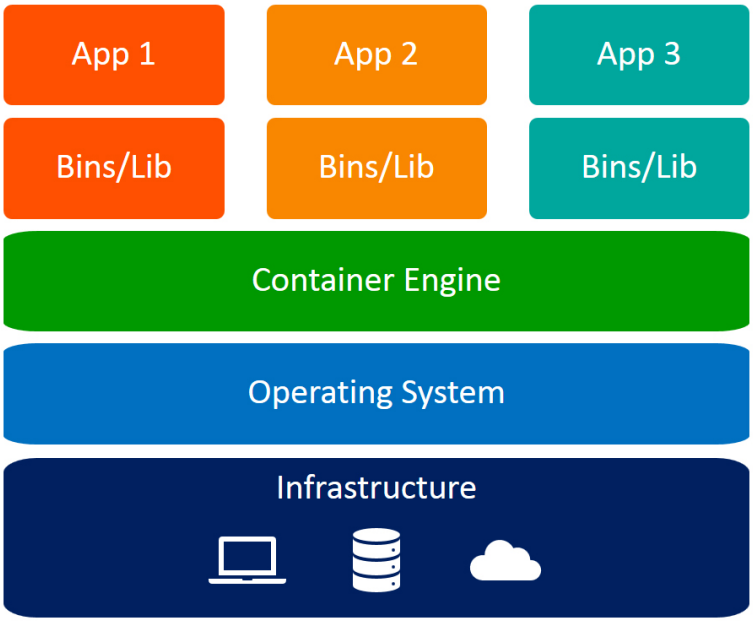

Figure 2- Infrastructure with containers (Raza & Kidd, 2020)

2.2 Kubernetes

Kubernetes is a so-called orchestrator and able to deploy applications that have been containerized. Since it has been introduced in 2014 it has become the standard API for building cloud native applications. Burns, Beda, & Hightower (2017) describe that there are four main reasons why containers and container APIs, such as Kubernetes, are so popular these days, namely:

1. Velocity
   In the current time updates to software are no longer distributed via CDs, but web based and at shorter intervals.
2. Scalability
   With the containerization of applications we are more able to scale applications on the aspect they really need to be scaled.

3. The abstraction of the IT infrastructure
   Kubernetes is an application-oriented API, which means that we no longer look at applications like virtual machines.
4. Efficiency
   Due to the fact that we no longer think of applications as virtual machines we can run applications from the same hardware as other applications

2.3 Thales domain

In modern warships we have the “combat information center” (CIC). this is a fortified part of any war vessel. The CIC functions as a tactical center and provides processed information of the near battlespace, or area of operations. The information being collected in the CIC is used by the commanding officer to make decisions. The information being shown could for instance be radar tracks, which registers each and every subject in the near surrounding. The software within the CIC, Tacticos, supports the operating crew by creating an automated situation awareness through the integration of multiple sensors, performing threat analyses and generating combat engagement plans.

This essentially means that in the CIC information from a lot of systems of the vessel is combined, such as radar tracks, status of missile launchers and the guns on deck. The radar tracks are part of the mission critical information, because they are also used for the guidance systems of the missiles and guns (and thus for the attack/defense mechanisms of the ship). All this information is gathered, processed and used for battle circumstances by the Thales combat management system, Tacticos.

The problem at hand is that there is so much data generated by all sorts of systems within the vessel that quality of service must be safeguarded for certain kinds of data, such as these radar tracks, while data like backups or video streams have lower priority and could be delayed for some time or in certain cases be completely dropped.

Another difficulty is that these streams of data are not constant. There are big differences between data streams when a ship finds itself in a combat

situation versus the situation when the ship is in a peace situation. In a combat situation the ship's radar will perform at its peak for an as accurate as possible situational awareness of the surrounding area. Furthermore, stabilization data, used to compensate the ship's weapon and radar system against the ship's movement, needs to be sent to for instance a gun. Meanwhile radar tracks also need to be sent to the gun for guidance to the target. You can probably imagine that in a situation like this network traffic is much more critical than when the ship is in some sort of "peace" mode, when updates to the system are applied.

Chapter 3

3 Literature on conventional networks

In this chapter we discuss the theoretical part of this study. Here we answer the research questions 1.1, 1.2 and 1.3. This means that after reading this chapter we have a definition of what quality of service is in the context of this thesis, how quality of service is safeguarded in a conventional network and why we move away from the conventional environment.

3.1 How can quality of service be best described?

In this section we will dive into the question how quality of service can be best described from different perspectives on the topic. Specifically, we consider these distinct viewpoints:

1. The technical side of quality of service, which includes metrics.
2. The organizational side of quality of service, where metrics are used as values for what is acceptable and what is not acceptable.
3. The end user perspective on quality of service. This part is covered in the organizational view on quality of service.

With this information we will formulate a definition of quality of service, that fits within the scope of this research. This definition is subsequently used for the rest of the study; which means that the testing and formalization will be based on this definition.

3.1.1 Technical view on Quality of Service

The default way for IP packages to travel the internet is that they have a best effort mode. This means that we try to deliver every package in a correct manner, but when congestion starts to happen, we do not ensure that every package reaches its destination (Ponnappan, Yang, Radhakrishna, & Braun, 2002).

Quality of service is all about how an application performs within a specific network. We do this through resource management. This quality of service can be accomplished by reserving bandwidth, prioritizing the different network packages, monitoring for change within the network and by scaling resources according to the needs of the application (Adis, 2003).

To measure the quality of our service we need to know how our network performs. This means that there should be some characteristics within our network and our application, that makes it possible for us to know if the user experience with regards to the application is satisfied with the network capacity and rules we have in place. There are several basic characteristics that are used to measure with what sort of quality a network package arrives at the receiver. The following characteristics are crucial when looking at quality of service (Szigeti, Hatting, Barton, & Briley, 2013):

- Delay (or latency): This is the amount of time it takes for a network package to reach the endpoint of the receiver from the endpoint of the sender.
- Jitter (or delay variation): This is the variation, or difference, in the end-to-end delay in arrival between sequential packets.
- Packet drops: This is a measure of the number of packets that are received by the receiving endpoint compared to the number of packages sent by the sending party, expressed as a percentage or in absolutes.
- Bandwidth: The amount of data that can be sent through a data connection.

By monitoring these metrics, we can get an overview of how the network environment is performing and see possible patterns appear. By recognizing the patterns of activity within the network we enable the possibility of scaling up or down in the resources allocated for the specific application and thus we get closer to safeguarding quality of service.

3.1.2 Organizational view on Quality of Service

When we talk about quality of service and network infrastructures, Service Level Agreements (SLAs) immediately come to mind. SLAs are a contract between a provider and a customer. For instance, between a cloud provider or a data center and a streaming service. The streaming service wants some sort of guarantee from the cloud provider that their infrastructure is on par with the needs of the streaming service. This form of guarantee is provided by making use of service level agreements. In this SLA the cloud provider and the streaming service define the outer bounds wherein the application will be able to show normal behavior. If the cloud provider does not meet the requirements set in the SLA, hefty fines can be applied due the fact that the streaming service quality lowers.

For us to know the normal behavior of an application, we need to know the standard or acceptable behavior of the application. These restrictions on behavior need to be formalized in so called Service Level Objectives (SLOs). In these SLOs topics such as availability, latency and capacity are considered (Hashman, 2019). An example of an SLO would be in the form of “There may be a maximum package drop of 2%”. An SLA would then consist of a multitude of these SLOs.

We know when things are going wrong by observing the behavior of the application (with metrics) and comparing these values with the values that are set earlier in the SLO. In other words, we can say that the end user or management expectations are declared in the SLO and with this we are able to commit to safeguarding quality of service.

Szigeti, Hatting, Barton, & Briley (2013) also state that it is critical to first develop these business or organizational targets the application should adhere to, before even talking about the technical implementation of quality

of service. This entails that service level requirements must be set for the application. With these requirements we can look for tooling that matches the requirements or start designing policies that enable quality of service.

As mentioned in the introduction of this section, we also have the user view on quality of service to discuss. There isn't much literature on this topic, apart from taking surveys and monitoring customer satisfaction. When we go back to our example of the cloud provider and the streaming service we can say that the streaming service does not want any form of video buffering. A requirement like this would be mentioned in the SLA and this would be translated into one or more SLO targets. Therefore, we can say that if the SLA and SLOs are set correct the end user will have a satisfying experience.

3.1.3 Defining Quality of Service

Now that we have seen the different views on quality of service we will formalize a definition of quality of service, suited for this study. This definition of quality of service will be the foundation of the rest of the research. Summarized we see that quality of service entails two main views: the organizational and the technical view. The core of the technical view is that we monitor and control the flow of network packages based on the metrics we see in our monitoring. We compare these metrics against the values we get from the SLOs, which in turn are derived from the SLAs. When we combine this information we get to the following definition:

Quality of service entails giving the end user a satisfiable experience, which is in line with the set organizational objectives.

This can be achieved via constant monitoring of the environment, resource management, scaling applications up or down and controlling the flow of network packages.

3.2 How is quality of service safeguarded in a conventional network?

As mentioned in section 3.1.1 quality of service is accomplished with management tools that reserve bandwidth, prioritize usage, monitor change and scale resources according to the current usage and needs.

There is a multitude of tools that can be implemented to safeguard quality of service in a conventional network. Naturally we would like some monitoring tools, but in general there are three groups of tools that can improve network quality (Szigeti, Hatting, Barton, & Briley, 2013):

- Classification and marking tools
- Policing, shaping and markdown tools
- Congestion management or scheduling tools

In this section we investigate how these network management tools can be implemented for a conventional network. When we speak of a conventional network, we talk about a network infrastructure that is not cloud native. To add some more detail; we speak of conventional networks in the case that switches and other networking devices are physical hardware devices. Now that we move towards a cloud native environment, we see that these physical hardware devices are being replaced by a virtual equivalent of these devices. It must be noted that these virtual devices run on physical devices which are connected via a physical infrastructure.

The outline of this section is as follows: first we describe each kind of tool used for safeguarding quality of service in a generic manner. After this brief overview we give an in-depth explanation of how such a tool can be implemented.

3.2.1 Monitoring

The first thing we must do to enable quality of service is using monitoring tools. How else would we know what the situation in our environment is like? This monitoring can be done on multiple locations within our environment such as monitoring the CPU usage of our application servers, but for this research question we limit the research to network monitoring. This monitoring is done on the service level objectives that are derived from the service level agreement that is agreed upon between provider and end user.

As mentioned earlier, these service level objectives are measurable characteristics within the network, such as delay, jitter, the percentage of packages dropped and bandwidth. Lee, Kim, Hong and Lee (2002) divide these characteristics into the following quality of service parameters:

1. Availability
2. Delivery
3. Latency
4. Bandwidth
5. Mean Time Between Failure (MTFB)
6. Mean Time to Restore Service (MTRS)

In this study we won't take MTFB and MTRS into account as they have no link with the network connection and are therefore out of scope.

The quality of service parameters can be split into network performance metrics (NPMs). These NPMs can be divided in the following four categories and their respective metrics (Lee, Kim, Hong, & Lee, 2002):

Availability	Loss (Delivery)	Delay (Latency)	Utilization (Bandwidth)
• Connectivity • Functionality	• One way loss • Round trip loss	• One way delay • Round trip delay • Delay variance (jitter)	• Capacity • Bandwidth • Throughput

Table 1 - Network Performance Metrics

First, we have the category *Availability*. There we have the NPMs *Connectivity* and *Functionality*. These metrics can be measured as in 'a device is reachable' or 'we receive functional data from application x', but apart from these statements we can't measure anything related to performance. It is connected/functional or not.

The second category is *Loss*. One way loss is the amount of network packages lost in a network stream from a sender A to a receiver B. Round trip loss measures the number of packages lost during a transmit. Apart from the number of packages lost this metric also shows if the package lost gets lost during sending or during reflection.

Delay entails three network performance metrics. The *One way delay* is the amount of time it takes for a message from sender A to be received by recipient B. *Round trip delay* also known as *Round trip time* is the amount of time that passes when a message is sent from A, received by B and that A receives an acknowledgement of successful transfer. The last metric from this category is *Jitter*. This displays the variance in the delays.

The last category is *Utilization*. The first metric is *Capacity*. Usually this is a given, for example, the cable between point A and point B can transmit data at a rate of 10 gigabit per second. *Bandwidth* is the rate at which a network stream may be sent. *Throughput* is the actual speed of the network stream.

The NPMs mentioned above are quite different. Some are measurable, but don't really have a performance scale, while other metrics are more of a given. Therefore, there exist multiple types of tooling to measure these NPMs.

There are three types of network monitoring tools. The first type is active monitoring. Active monitoring includes doing performance tests on the network, which enables us to monitor our connectivity, delay and loss metrics and bandwidth utilization. We can also apply passive monitoring. This enables us to measure the utilization metrics within our environment. Passive monitoring is mostly done on the forwarding devices within the network infrastructure. Traffic statistics, for instance, are automatically gathered by a switch and can be requested for monitoring goals. At last, we have the SNMP agents. SNMP-agents can be used to retrieve management data of the client. Through the collecting of this information, we can measure functionality and throughput metrics (Lee, Kim, Hong, & Lee, 2002).

These metrics on themselves don't improve our network, but they give us a better understanding of where in our environment congestion takes place and what other forms of potential hiccups might be present.

3.2.2 Classification and marking tools

Network flows of traffic can be analyzed to classify the network packets within the flow and consequently marked. In essence this means that we make a difference between the IP-packets we send and receive.

First the IP packages are marked differently. This marking is done in the header of the network packet. In the case of an IPv4 package this is called the DSCP field and the traffic class field in case of an IPv6 package.

The classification of IP packages is done based on the marking in the IP header fields. This in turn leads to IP packages belonging to different network classes. With these classes we can say that class X has higher priority on the network than class Y. The analysis of these network flows is done on the access switches of an environment. This has as effect that the analysis needs only to be done once within an environment (Xiao & Ni, 1999).

3.2.2.1 DSCP trust mode

We discussed using the DSCP field in a network package as a classification mechanism. By classifying these network packages, we can place them at the queue with the correct priority on a switch. Some applications can set the DSCP value by themselves, while others don't. This means that we cannot always believe that the priority of a network package is the same as the value the network package has in its DSCP field.

To make sure that the network packages receive the correct priority it is possible to overrule the value that is written in the DSCP field. Cisco switches have three different types of handling these DSCP values. The first mode is 'Basic Mode'. In this mode we trust that the DSCP value is correct and network traffic is handled as such. The next mode available on the Cisco switches is the 'Advanced Mode'. In this mode we make use of an Access Control List which helps us to give the right priorities to specific network traffic. The third option is to not take these different priorities into account and just make use of a best effort mentality (Cisco, n.d.).

As the reader might have guessed it is not possible to use more than one different mode at the same time on the same device. As we just mentioned the third mode has no different priorities and takes a best effort approach. In the advanced mode we generally trust no one and use the Access Control List for prioritizing and we have the Trust mode where we accept the value of the DSCP field.

Although we are not able to use more than one mode on one device, we do want to make use of different modes on different devices. On the access switches, or switches that are connected to the end users we don't want to trust the DSCP value of the network packages as this could lead to end users damaging the quality of service within our network by flooding the network with high priority packages. This means that on the access switches we want to use Advanced Mode if we want some level of quality of service at this point in the network. If we don't need quality of service for some part of the network, we can make use of the best effort class. On the core switches we want to trust every network package, as these switches are meant to forward data as fast as possible, without verifying what is in the package.

3.2.2.2 Access Control List

An Access Control List (ACL) allows us to apply rules to incoming and external traffic on a network switch. With an access control list, we are able to block certain types of traffic on the basis of IP address, protocol (TCP/UDP) and port number, but in the context of quality of service this is not used. An access control list doesn't allow for deep packet inspection telling us what type of traffic is being handled. However, based on the origin/destination, the protocol and the port number, we have a good understanding of what kind of traffic is being handled.

Now that we know what kind of traffic passes through our ACL on the switch, we can label these kinds of traffic by editing their traffic class or DSCP value in the IP-header. Furthermore, we can classify different types of traffic and map this class to a specific queue on the switch. With this queueing we can ensure that the network classes get the right priority on the switch (Is5com, n.d.).

This has a positive impact on the delay of higher priority network packages, while network packages with no or low priority will experience a negative impact on the delay of network packages. The same can be said on the topic of package drops. All in all, we can say that these Access Control Lists are used as classification and marking tools.

3.2.3 Policing, shaping and markdown tools

We can create different classes of network traffic. Each network class gets its own portion of the network resources allotted. It can still happen that network packages need to be dropped, delayed or re-marked to avoid congestion when traffic exceeds the available resources.

Xiao & Ni, (1999) explain that an SLA describes what the requirements of a specific application are. It is the responsibility of the user/customer to make sure to not go out of the bounds that are set in the SLA. The user can prevent the network traffic going out of these bounds by making use of shaping. This entails that the outgoing (egress) routers on the user side will shape the network traffic such that peak traffic will stay within these bounds. On the provider side policing tools are used to verify that traffic towards the user

(ingress) adheres to the bounds set in the SLA. Szigeti et al. (2013) mention that if during peak traffic, traffic exceeds bounds, shaper tools try to flatten this peak by buffering and delaying traffic.

3.2.3.1 Shaping

An example of such a shaper is a token bucket algorithm. With a token bucket algorithm we can send network packages at high-speed bursts, for a very short while, to stay within the bandwidth limits. After a time limit, we are able to send network packages again at a high burst speed (Medhi & Ramasamy, 2018). This will help providing a satisfiable connection on for instance, a VoIP channel, for as long as our rate of sending data isn't taking more time than an acceptable delay of speech in the call. This is an example of a shaping mechanism that affects outgoing network traffic. Due to the constant time intervals between the sending of data we gain a non-variable amount of jitter. This in turn leads to a higher quality connection.

3.2.3.2 Policing

Policing tools are there to verify that the packages being sent adhere to these set requirements. These policing tools are used at the providing side of the SLA. If network traffic exceeds the bound that are set in the SLA these policing tools will be used to drop packages.

Cisco has developed two policing algorithms. Committed Access Rate and Class-Based policing (Heggi, Abd El-Kader, Eissa, & Baraka, 2009). Both policing algorithms use a Token Bucket algorithm. In case ingress traffic exceeds the limits of inbound traffic the Committed Access Rate algorithm drops network traffic to stay within the limit. Class-Based policing drops network traffic while also taking the priority of a network package into account by looking at the DSCP value of a network package.

3.2.4 Congestion management or scheduling tools

Networks are built with switches and routers. These devices forward information to the next switch, router or end device. If the input stream of such a device exceeds the output rate of the device congestion will arise (Bernet, 2000).

Enabling quality of service with congestion management entails that network streams must be grouped and put into specified queues at the forwarding devices. This buffering only happens when congestion starts to appear but placing the network packets in queues (of different importance) always happens. Congestion management is built upon two main ideas (Szigeti et al, 2013):

1. Queuing: When congestion starts to appear at the forwarding device network packages will be ordered in output queues each with their own class of importance. Some types of network traffic do not fare well with being queued. These kinds of traffic should make use of Admission Control techniques and should wait to be sent, until we know that there is enough room for these packages on the network.
2. Scheduling: Because of the different classifications of network packages with regards to quality of service, there must be some sort of scheduling management. Bernet (2000) describes the scheduling as traffic handling mechanisms, which are subject to policies that are set by network administrators. This means that the scheduling is dependent on the service level objectives of the organization.

The buffering is done in the memory of a forwarding device, such as a router or a switch. Since this memory is a limited resource on the forwarding device, it is impossible to have an infinite length of packages waiting in queue. This ultimately means that when the buffer size is full, that packages need to be dropped. There exist a multitude of algorithms or tools to keep these buffers from overflowing. We present an example of a scheduling and queueing algorithm in the next two sections.

3.2.4.1 Load Balancing

One of the easiest ways to create a higher quality experience is by monitoring the load of the environment. When we know where in our infrastructure load is getting high or congestion starts to appear, only then we know where we can scale our environment and decrease the load. This scaling can be done by adding, for instance, an extra application server or another switch to the environment (NGINX, n.d.). By decreasing the load, we lower the amount of

congestion, where load was high. This means that load balancing is a congestion management tool.

Congestion has a negative effect on the percentage of dropped packages, because when congestion starts to get critically high, packages most certainly will be dropped, if allowed. By balancing the load, we lower the amount of congestion of a specific object, which has a positive effect on the percentage of dropped packages.

A load balancer needs to redirect a network package to the correct endpoint. In other words, a load balancer needs to schedule network traffic. An example of such a scheduling algorithm is Round Robin. Round Robin is one of the simplest load balancing algorithms. Say you have K amount of endpoints. Round Robin algorithm will schedule package N at endpoint N. The next package N+1 is sent to endpoint N+1, until endpoint K is reached and the cycle starts again (Hidayat, Azzery, & Mahardiko, 2020).

3.2.4.2 Package dropping

Although we can make use of load balancers to lower the load, it is not always possible to scale in resources. This means that in certain circumstances traffic needs to be dropped to prevent the buffers of forwarding devices to overflow.

An example of such an algorithm that drops packages on a forwarding device is the RED algorithm. The problem with the RED algorithm concerning quality of service, is that the package being dropped is random. This could lead to the dropping of packages with critical information. To make sure that packages with higher priority don't get dropped we can also make use of the WRED protocol. This is in essence the same as the RED protocol, only we are able to add weight to a package (Kesh, Nerur, & Ramanujan, 2002). This weighing of the network package is done based on the traffic class field or DSCP field. (Szigeti et al, 2013).

3.2.5 Summary

In this section we talked about how quality of service can be safeguarded within a conventional environment. The first step we take is to monitor our environment and keep track of our Network Performance Metrics (NPMs).

These NPMs are shown in Table 1. Now that we know where potential bottlenecks in our environment take place, we are more capable of safeguarding quality of service. The gathering of these NPMs can be done via three types of monitoring:

1. Active monitoring through performance tests
2. Passive monitoring on forwarding devices
3. Using SNMP-agents

The second way that we can safeguard quality of service is by influencing how network traffic is transmitted within our environment. The tooling that we can use to influence network traffic can be divided into three categories:

1. Classification and Marking tools
2. Policing, Shaping and Markdown tools
3. Congestion Management or Scheduling tools

Within Classification and Markdown tools we have DSCP and ACL. DSCP can be used to add the priority of a package into the IP header. With an ACL we can read to what type of class a package belongs and put it on the according queue on a forwarding device.

Policing, Shaping and Markdown tools most often make use of a Token Bucket algorithm that allows us to shape egress traffic (by making use of a buffer) in a way that our sending throughput stays within our set limit. Token Bucket algorithms can also be used in a policing function. This entails that the algorithm drops traffic that exceeds our set limit.

We can manage congestion by making use of a load balancer. This load balancer needs a scheduling algorithm to determine to where network traffic should be sent. An example of such an algorithm is Round Robin. Queuers on the other hand determine how congestion at a queue should be handled. Queuers can drop packages with algorithms such as RED and WRED.

In conclusion, we can safeguard quality of service in a conventional environment by monitoring our environment and making use of one or more of the tools mentioned above.

3.3 What are the shortcomings or barriers of quality of service in a conventional network?

In this section we answer the question on what the shortcomings of a conventional infrastructure are. By answering this question, we also give some explanation why the usage of cloud computing keeps increasing. Section 3.3.1 is added to give some extra context to the situation but will not help us to answer the research question above.

3.3.1 Resource management

In conventional network infrastructures we see that every application runs on one or more application servers. This application server is built with a set of hardware requirements. For instance, the machine must be equipped with 32GB of RAM. In most cases an application has peak traffic at specific intervals and thus the application server is equipped for this peak traffic. At other times the application server doesn't nearly reach the usage of this peak traffic. In a conventional IT infrastructure, it isn't possible to dedicate some of the RAM the application server has to other application servers in need of resources, since this RAM is physically present in the application server.

This means that resources can be better utilized when the problem of physically allotted resources is taken away. By moving to a cloud native infrastructure, we no longer use physical application servers. Instead, we move towards an environment where we have a pool of resources and virtual machines or containers get a portion of the resources available in the pool. This in turn reduces the amount of overhead while deploying applications (Truong-Huu, Koslovski, Anhalt, & Montagnat, 2011).

3.3.2 Scalability

Another topic where the conventional infrastructure comes short is on the topic of scalability. As mentioned in the chapter on resource management, we scale by adding another physical server, or any other object for that matter. Due to this changing being a physical activity it will always take at least the time it takes to add the physical object to the environment. It can also happen that there is no physical room to add another physical device into the environment and scaling up on one part means scaling down on another.

This is where going to a cloud native environment seriously gets interesting. When moving towards a cloud native environment we use more and more virtual machines and containerized applications in our environment. By making use of applications that are 'containerized' we also lose an overhead on resources, due to not completely running a new virtual machine (including a complete operating system) in this virtual pool of resources (Xie, Yuan, Zhou, & Cheng, 2018). These containers can be automatically deployed based on how an application is performing. When an application has its peak usage new containers can be started to decrease the load on the application and when usage of the application normalizes, containers can be stopped and resources can be given back to the pool of resources (Kubernetes, 2021).

3.3.3 Physical nature of a network

IT networks in a conventional infrastructure are mostly placed in a non-dynamic environment. If packages need to be sent out of the own environment, we need to place a router and we keep it placed to maintain the ability of sending these packages. These hardware devices can be configured automatically, but need human intervention, to function properly. By moving towards a cloud native environment, we start to make use of virtualized hardware as mentioned above. This virtualization process also applies to the network hardware. Virtual routers and switches are available for these cloud native environments. This means that also these kinds of virtual hardware can be created with software (Clayman, Maini, Gallis, Manzalini, & Mazzocca, 2014). The ability of software being able to add virtual network hardware into the environment makes the environment more fluid than the conventional situation. Apart from the software being able to add functionality, we also need to do less configuration manually.

3.3.4 Summary

In this section we investigated the quality of service shortcomings in the conventional environment are and thus indirectly why an organization would move to a cloud native environment.

In short it comes down to that a cloud native environment reduces the amount of overhead on hardware. By reducing the overhead, we have more room for scaling of our applications. This can be done by horizontally scaling (adding more servers) or vertical scaling (adding more resources to one object, for instance a server). In a conventional environment a lot of the configuration of servers and forwarding devices needs to be done manually. This process can be automated with the help of software in a cloud native environment, which in turn leads to a more fluid environment than a conventional environment.

Chapter 4

4 Analysis of cloud technology

In this chapter we explore the possible ways of safeguarding quality of service in a cloud native environment. This includes looking at what currently exists and how this cloud native environment could be connected.

4.1 How is quality of service safeguarded in a cloud native environment with microservices?

As we discussed in earlier sections, quality of service is safeguarded through constant monitoring of the environment, resource management and scaling applications up and down. This will not change when making use of a cloud native environment, but how we can apply these tactics does change. In this chapter we investigate the characteristics that are already there in a cloud native environment and how we can adapt these to ensure quality of service. Again, resource management does not help us to answer the research question, but it is such an essential part of the subject that we have added it to the thesis.

4.1.1 Resource Management

As we move from a conventional infrastructure with physical hardware to an infrastructure with virtualized hardware, the way we manage these resources also changes. In a cloud native environment we make use of pools of physical hardware. Virtualized hardware can request resources from these pools.

To make sure that every application has access to the resources it needs to perform we have to make use of resource scheduling. There is a multitude of algorithms that can take on the job of this scheduling, but we won't go in depth of this theory, due to it not helping us answer the research question. In short, we need algorithms within the cloud native environment that schedule which task is performed on which part of the hardware pool. The better the scheduler is at allocating resources for the process, the less overhead we have on our physical resources. An example of a best effort algorithm is the Round Robin Algorithm, which is a CPU scheduler. With Round Robin every process gets an even amount of processing power. In this way a queue of processes is being handled and new processes are added to the end of the queue (Pradhan, Behera, & Ray, 2016). Other algorithms exist, which also support Quality of Service by giving weight to the processes in the queue, such as Weighted Round Robin.

4.1.2 Monitoring

As mentioned in the introduction quality of service is ensured via monitoring, resource management and scaling. In this section we look into how a cloud native environment can be monitored and what kind of tooling currently exists with this purpose.

In section 3.2.1 we described the three categories of monitoring that we can use for network monitoring: active monitoring, passive monitoring and making use of SNMP agents to test the functionality of our environment. When moving towards a cloud native environment the metrics that enable us to safeguard quality of service do not change (Cignoli, 2016). This means that the four basic metrics we described in section 3.1.1 are still relevant, although Cignoli has added a fifth metric, quality of service. Cignoli actually means an implementation of the DSCP tag. In line with the rest of the study we see this as a way of enabling quality of service and therefore not as a new metric.

The way in which a cloud native environment is monitored however, does Change. In the conventional environment we monitor forwarding devices, such as switches, but as we make use of a cloud native environment this physical monitoring extends to the monitoring of virtual devices. All in all, the way in which we monitor quality of service metrics does not change that

much when moving to a cloud native environment combined with virtual networks. However, we need to monitor more 'entities' as we get virtual and physical devices in the same environment.

4.1.3 Scalability

As mentioned in section 3.3.2 on scalability it takes at least the time to perform a physical action, to scale up or down in a conventional environment. We see that in a cloud native environment all physical components of the conventional infrastructure get their virtual counterparts. This means that there is no physical action required to enable the scaling of an application, as we do not make use of physical objects in the cloud native environment.

We are able to scale on different levels within a cloud native environment. First, we have the horizontal scaling where we add new virtual components to lessen the load on the existing component. This can, for instance, be done by firing up another application server during peak usage of the application. The addition of another application server makes it so that we have another server that can handle the requests. This allows us in case of starting congestion at server 1, to start server 2 which helps to reduce the congestion and thus helps us safeguard quality of service.

Another possibility is to vertically scale a virtual object. An example of this is by adding more RAM to the object, but most operating systems do require a reboot for these changes to be effective (Patibandla, Kurra, & Mundukur, 2012). As mentioned above we don't have to perform a physical action to actually add these resources to the object that needs to scale. This allows us to define rules on when the virtual object needs to be scaled. By making use of this automated scaling, we can in turn reduce the amount of overhead on resources. Vertical scaling allows us to increase the resources and performance of a virtual object. This in turn leads to a virtual object that can process more requests and therefore helps us safeguarding quality of service.

4.1.4 Kubernetes

With the new software platform that is currently in development by Thales, Thales is moving toward a cloud native environment based on Kubernetes. Therefore, we lay a foundation on how the tooling that we've discussed

earlier can be applied in an environment with Kubernetes. The information in this section is a summary of the Kubernetes documentation that is relevant to quality of service. Apart from monitoring, resource management and scaling Kubernetes does not enable us to take quality of service measurements on the networking level. For us to make use of such tooling we need external developed container network interfaces. We will elaborate on these plugins in section 4.2.

4.1.4.1 Classification of pods

Kubernetes has three types of classifications at the ready for the different pods within Kubernetes. Pods are the smallest deployable objects within Kubernetes, but they can host a multitude of containers running on them. The classifications for these pods are:

1. Guaranteed
   Pods that need consistent and good performance should get the *Guaranteed* classification. These pods have top priority and will not be killed unless they exceed their limits.
2. Burstable
   Burstable pods have a minimal resource guarantee, but if more resources are available pods may use these resources. Under pressure these pods might be killed when no best effort pod exists.
3. Best effort
   Best effort pods are treated with the lowest priority and are the first to be killed when under pressure. These pods can however access resources when available.

4.1.4.2 Resource management

For these classifications to function correctly, resource management must be considered when creating the pods. Kubernetes makes use of two different types of resources: CPU and RAM. Both resources represent a virtual instance of the physical resources. This enables us to request 0.1CPU for a specific container. Each pod must make clear the requested amount of RAM and CPU. If these specifications are not set when creating the pods, the default class for these pods is *Best effort*.

If the request and limit are exactly the same, Kubernetes will assign the class of *Guaranteed* to the pod. In case the limit is higher than the request amount, the pod will be given the class *Burstable*. When no limit and request size is specified, the pod will be classified as *Best effort*.

Before the creation of the pod the Kubernetes scheduler will first check if the intended node to place the pod on meets the resource requirements of the pod before placing the pod.

4.1.4.3 Monitoring

The resource status of a pod is by default visible in the pod status. The metrics that are being shown automatically are only the usage of RAM and CPU. This does not restrain us from making use of third-party tools that enable us to view all available metric data, such as Prometheus. It should be noted that these metrics only report the resource status of the node and don't say anything about the functionality of the application.

4.1.4.4 Scalability

Kubernetes has implemented their so-called Horizontal Pod Autoscaler (HPA). This enables us to automatically scale our pods depending on current resource utilization. These resources can be the amount of RAM or CPU used, but we are also able to scale on custom metrics. The HPA compares the resource utilization with the limits that are set on creation of the pod. By default, the HPA does this every fifteen seconds, but this interval can be changed.

4.2 How can different pods be connected to enable quality of service?

Kubernetes itself has functionality which enables network communication between different nodes and pods. Apart from this basic functionality so called Container Network Interfaces (CNIs) exist. These allow us to have a more fine-grained control over the network traffic within an environment. In Table 2 and Table 3 we show the different CNIs that exist for Kubernetes and at the same time have functionalities that enable quality of service. These CNIs are divided into three different categories:

1. Encapsulation: These CNIs make use of an overlay network, which acts as an abstraction layer for services within the Kubernetes cluster to communicate with each other. Due to the encapsulation and decapsulation of network packages, performance decreases.
2. Non-encapsulation: These CNIs do not rely on an overlay network to enable communication within the Kubernetes cluster and therefore should have a higher throughput than CNIs that make use of encapsulation.
3. Virtual switches: These CNIs mimic the functionalities that are offered by a physical switch. This means that CNIs that make use of these techniques, in theory should offer the most customizations that can be related to quality of service.

In the category of non-encapsulating CNIs we see that Calico is the only one which by default uses this network infrastructure and has basic functionality for quality of service.

Name	Description	Category
Cilium	Supports eBPF Supports Kubernetes network policy controls Supports service load balancing Supports connectivity of multiple Kubernetes clusters Supports cluster wide policies	Encapsulation
Flannel	Enables basic network connectivity within the cluster Does not have support for network policies	Encapsulation
Kube-OVN	Supports network policies with ACLs Supports multi-cluster networking Supports configuration options for quality of service, such as ingress/egress traffic rate	Encapsulation

Table 2 – Encapsulation Container Network Plugins

With the encapsulation network infrastructure, we have more CNIs to choose from. Flannel is very basic, but Cilium and Kube-OVN have some quality of service tools available.

Name	Description	Category
Antrea	Supports OVS, which has better performance than iptables Supports Kubernetes network policy controls Supports tiering of policies, rule priorities and cluster-level policies Uses Open vSwitch	Virtual switch
Contiv-VPP	Supports traffic policies	Virtual switch
Open vSwitch	Supports traffic queuing and shaping	Virtual switch

Table 3 – Virtual Switch Container Network Plugins

In the Virtual Switch category, we see that most CNIs offer basic quality of service mechanisms and Antrea delivers some mechanisms that integrate with a Kubernetes environment.

Chapter 5

5 Experiments

In this chapter we experiment with different Kubernetes environments to research their capability on enabling quality of service. This chapter covers research question 3.1, “How do different tools that enable quality of service differ from each other in a test environment?”. The results of this chapter will help us gain insight on how CNIs can be used to safeguard quality of service.

As mentioned in section 3.2 there are three types of tooling that can be used to safeguard quality of service in a conventional network. These categories and some of their practical implementations are shown in Table 4. During this study we try to use these types of tooling to safeguard quality of service in a cloud native environment with Kubernetes.

Classification and marking tools	Policing, shaping and markdown tools	Congestion Management
• Add priority to network stream • Move network streams to correct class queue	• Police ingress traffic • Shape egress traffic	• Load Balancer • Drop packets with for example WRED algorithm

Table 4 - Quality of Service implementations

At the moment of writing these tools are not yet developed for the CNI, or still very basic in their functionality. A classification via a DSCP tag is something that we can easily add to our network packages and by adding this tag we can prioritize network traffic. An ACL, if supported by the CNI, is very basic and only has a firewalling function between Kubernetes objects.

On the subject of policing, shaping and markdown tools we can currently use the bandwidth plugin, which is supported by most CNIs. With the bandwidth plugin we can limit the rate at which ingress (incoming) and egress (outgoing) traffic is sent to and from pods. This traffic is handled by a token bucket algorithm to shape it according to the rate limit. Cilium has its own implementation of a bandwidth limiter called the Bandwidth Manager. This Bandwidth Manager only supports egress rate limiting (shaping) and doesn't support the policing of ingress traffic. We can only functionally test these plugins and if these work as intended, they can be used as a tool for quality of service. Apart from these functionality tests we cannot do a performance test on these tools.

A load balancer can't be used as a test for quality of service. With a load balancer we would scale the application and thus increase resources. A load balancer will help us decrease congestion, but this would not help us reaching a higher quality of service level when resources cannot be increased. Ultimately, this also means that we cannot test quality of service support for a CNI with a load balancer.

Functionality such as dropping packages in case queues are starting to overflow is not possible with the current technology. We had hoped that CNIs would have implemented a congestion management technique such as AC/DC TCP as mentioned by He, et al., 2016, but no CNI has this implemented as of yet.

We had hoped that CNIs based on virtual switch technology would provide us with a programmable environment, just like regular virtual switches. Unfortunately, this is not (yet) the case.

In section 4.1.4 we discussed three quality of service classes that exist within Kubernetes; *Guaranteed*, *Burst* and *Best Effort.* We can't use these quality of service classes within our tests, because these quality of service classes only have impact on CPU and memory metrics. They do not affect network metrics and have no relation with a CNI.

All in all, there is lot of functionality that we can't test at the moment. However, we can still experiment with adding priority to a network stream via a DSCP tag. On the topic of the policing and shaping of network traffic there are also some experiments that can be done. These will follow in section 5.1.

5.1 Approach

In this section we look at how we approach the experiments. As there are quite a few CNIs to choose from and only limited time, we focus in this research on one CNI each from the different categories we described in section 4.2. Therefore, we perform these experiments with Calico (Non-encapsulation), Cilium (Encapsulation) and Antrea (Virtual Switch) as these CNIs promote that they support more quality of service features than other CNIs. There are two different environments that we want to test:

1. Intra node traffic
2. Inter node traffic

With the intra node traffic or internal node tests we can test how the three CNIs compare to each other when traffic stays within the machine. With inter node traffic or external node tests we can see how performance changes when network traffic must leave the virtual environment and needs to use physical connections.

We can test quality of service implementations from two categories in these setups. First, we want to verify if giving priority to network traffic via a DSCP tag is possible. Afterwards, we verify if shaping egress traffic and limiting ingress traffic is possible. Shaping egress traffic and limiting ingress traffic is tested with the bandwidth plugin that can be added to the CNI. In section 3.2.1 we discussed three different types of monitoring. As we are actively measuring how our Network Performance Metrics change under certain conditions, we are making use of active monitoring tools.

In our priority via DSCP test we perform a baseline test, where we don't use quality of service settings. We test the Network Performance Metrics we described in section 3.2.1. We use the following three tools to test these NPMs:

Iperf3: One way loss, Bandwidth, Throughput and Jitter
Owamp: One way delay
Twamp: Round trip delay and Round trip loss

With the baseline test we can verify the performance of the CNI when we add congestion on the network. Afterwards, we add congestion by adding a second network stream. In our third test we give priority to one network stream via a DSCP tag, to verify how the CNI handles these types of traffic.

The test setup for the intra node traffic can be seen in Figure 3. In this setup we have four pods on one node. First, we want to verify what the capacity is within this virtual network. We can measure this maximum capacity within the virtual network through an iperf3 test from pod A to pod C. This is measured by using the following command on pod A "iperf3 -c *ip-address-pod-c* -u -b 0". This measurement is done ten times per CNI, and the highest value is our capacity. We measure a capacity of 2174Mbit/s for Calico, 3920Mbit/s for Cilium and 3541Mbit/s for Antrea.

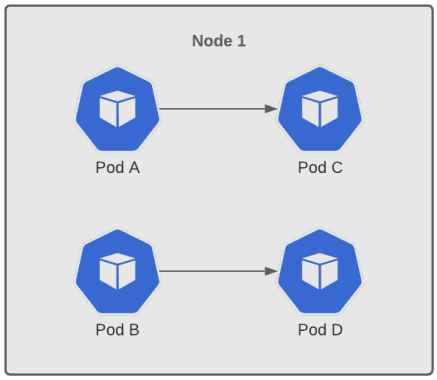

Figure 3 – Test setup Intra node traffic

The second test scenario is for network streams that leave the Kubernetes node and need to connect to a Kubernetes node placed in the same cluster. This setup is shown in Figure 4. In this test setup we have two machines connected to the internet, while also having a direct connection to each other. This allows us to download the necessary software on the machines and to test without using intermediate forwarding devices. This means that all Kubernetes related traffic is sent via the direct cable connection between the two nodes.

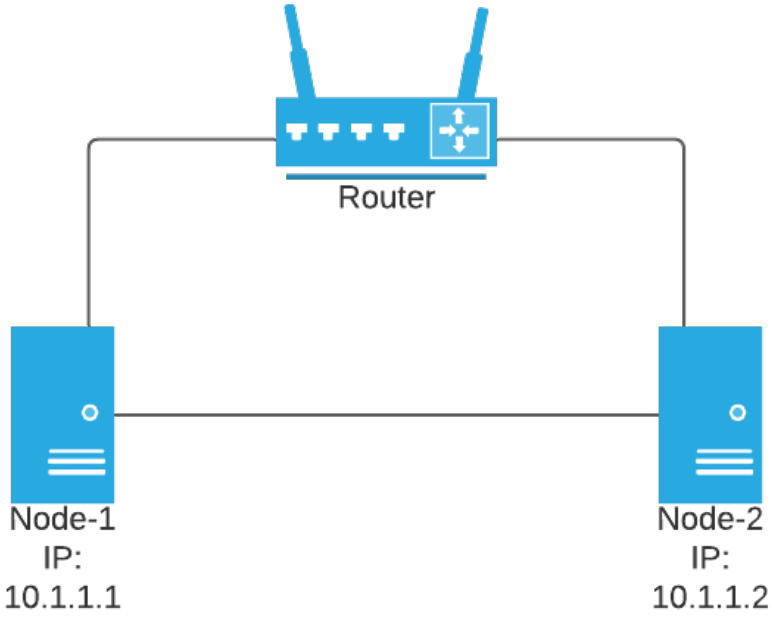

Figure 4 – Overview setup inter node traffic

Now that we are testing with two physical machine the first step is to make sure the clocks of the two nodes are synchronized. On Node-1 we install a NTP server. This allows us to synchronize the clocks between Node-1 and Node-2.

The cluster view of the external node traffic is shown in Figure 5. Traffic in this setup goes from virtual (within the pod) to physical (when it leaves the node). This means that the capacity now changes to the maximum throughput our physical cable supports. In our test setup we use cat5e-cables, which should give us a capacity of 1000Mbit/sec. By performing an iperf3 test from a pod on Node-1 directly to a pod on Node-2 we get a maximum throughput of the CNI. We measure a maximum bandwidth of 950Mbit/s for Calico, 916Mbit/s for Cilium and 912Mbit/s for Antrea.

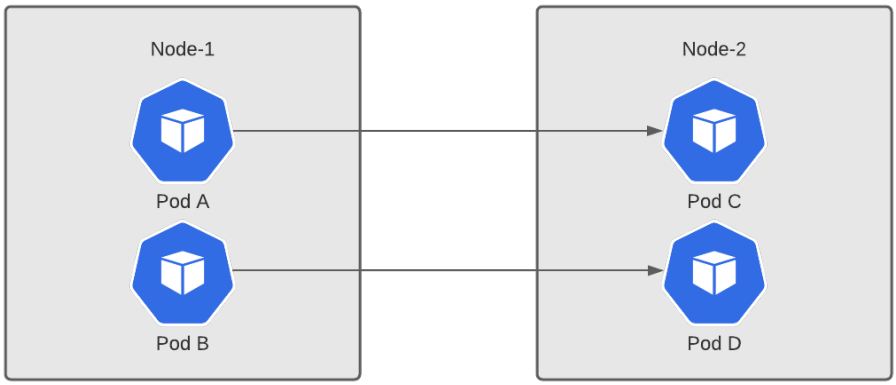

Figure 5 – Inter node traffic cluster view

5.2 Baseline

Now that we know the capacity of our inter and intra node networks we can perform the baseline test. By doing these baseline tests we can set a foundation for the congestion tests, but we can also compare the CNI (categories) with each other. As mentioned in section 5.1 we have three testing tools that we use for the baseline, congestion and congestion with prioritization (via a DSCP tag) tests:

1. Iperf3
2. Owamp
3. Twamp

During these tests we expect a package drop rate of 0% or close to 0% as we stay within the maximum capacity of our environments.

5.2.1 Iperf3

In Table 5 we show the results of our intra node iperf3 baseline tests. The Bandwidth, Jitter Sender and Lost Sender are omitted from this table to increase readability. In our intra node tests the bandwidth is the same as the

capacity of the environment. Jitter Sender and Lost Sender both had a value of zero. Apart from Capacity the columns in this table are averages.

The first thing that catches the eye is the capacity and throughput of Cilium. Antrea also scores high at capacity and throughput, while Calico keeps behind. When we look at the jitter and package drop, we see that Calico scores best. This indicates that Calico has the most consistent connection.

	Capacity	**Throughput Sender**	**Throughput Receiver**	**Jitter Receiver in ms**	**Lost Receiver**
Calico	2174 Mbit/s	1592,5 Mbit/s	1588,9 Mbit/s	0,0011	0,233%
Cilium	3920 Mbit/s	2539,7 Mbit/s	2517,5 Mbit/s	0,0022	0,871%
Antrea	3541 Mbit/s	2368,1 Mbit/s	2350,3 Mbit/s	0,0032	0,761%

Table 5 – Intra node iperf3 results

In Table 6 we see the results of the inter node iperf3 baseline tests. Some columns are omitted to improve readability of this table. This concerns the columns Capacity, Jitter Sender and Lost Sender. The Jitter Sender and Lost Sender were 0 in every test. The capacity is the link speed of the cable, which is 1Gbit/s. A reference to the complete data set can be found in Appendix A.

	Bandwidth in Mbit/s	Throughput Sender in Mbit/s	Throughput Receiver in Mbit/s	Jitter Receiver in ms	Lost Receiver in %
Calico	950	950	947,4	0,019	0,247
Cilium	916	894,5	893,6	0,013	0,103
Antrea	912	898,3	896,7	0,014	0,165

Table 6 - Inter node iperf3 results

5.2.2 One-way active measurement protocol

The results of our intra node owamp tests are shown in Figure 6. The results of this test are very close together. We can say that Calico has the most consistent connection, and that Antrea is the fastest.

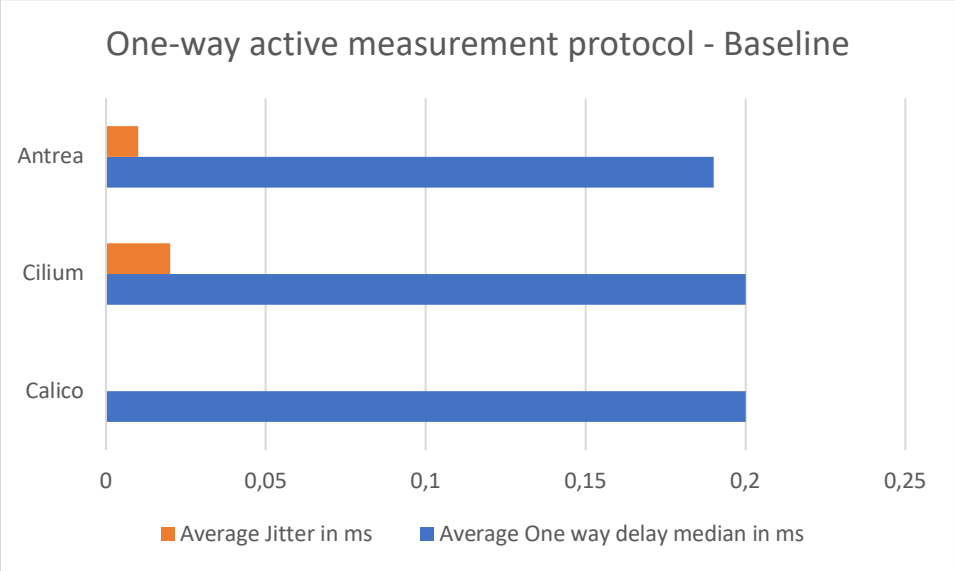

Figure 6 – Intra node owamp

In Figure 7 we see the results of inter node owamp tests. None of the CNIs encountered package loss during this test, so these results are omitted from

the figure. Here we see that Antrea has the least amount of jitter, which means that this is the most stable connection. Cilium has the lowest average one way delay median and is thus the fastest CNI in this test.

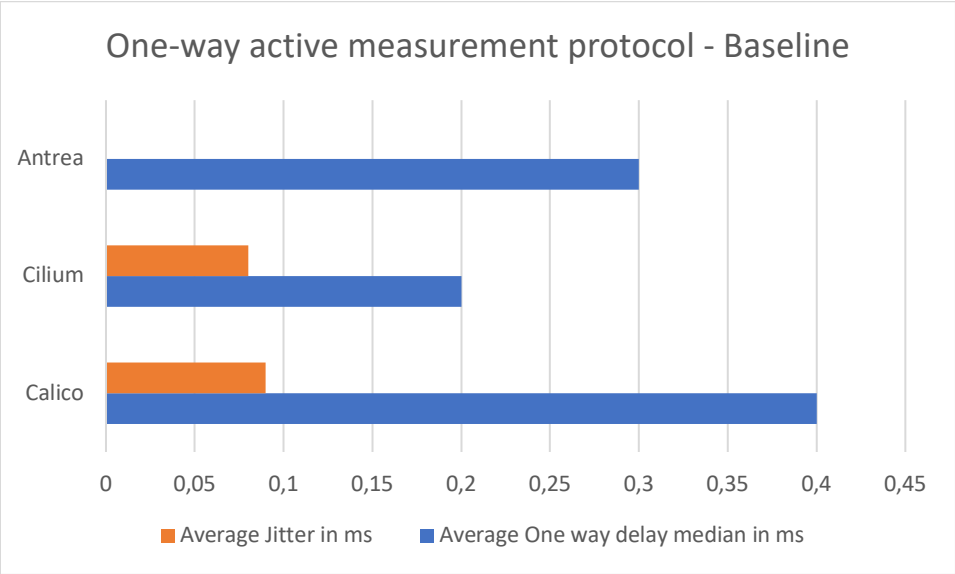

Figure 7 - Inter node owamp

5.2.3 Two-way active measurement protocol

Unfortunately, we couldn't do the twamp test for Antrea, due to twamp not supporting the NAT infrastructure of Antrea[1]. The results of Calico and Cilium are shown in Table 7. For this test it also holds that there were no dropped packages and to increase the readability we omitted this column from the results. There is no difference in performance of the NPMs between Calico and Cilium.

[1] https://github.com/perfsonar/owamp/issues/49

CNI	Round trip loss	Round trip delay in ms
Calico	0	0,3
Cilium	0	0,3

Table 7- Intra node twamp results

In Table 7 we show the inter node twamp results. The results are again very close, but Cilium provides a faster connection.

CNI	Average Packages Lost (Round Trip Loss)	Average Round Trip median delay in ms
Calico	0	0,60
Cilium	0	0,54

Figure 8 - Inter node twamp

5.2.4 Problems with intra node testing

During the experiments with intra node testing we encountered some strange behavior with our test software. With the TCP-protocol we can generate 60Gbit/s of throughput with ease with Iperf3 when using parallel streams. The usage of multiple network streams simultaneously is to verify the maximum throughput within our environment. When using the UDP protocol we can get a maximum throughput of around 7Gbit/s. We can increase the number of parallel streams with UDP traffic as well, but unexplainable package drop starts to appear quickly. We tried contacting Iperf3 on their GitHub page[2], but we did not get an answer that helped us solve the problem.

The problem is that we need the UDP protocol to set a capacity limit in our intra node environment so that every test runs under the same condition.

[2] https://github.com/esnet/iperf/issues/1263

This can be done at the capacities that we mentioned in section 5.1, but we know that this is nowhere near the actual capacity of the environment.

This means that we can perform baseline tests, but we cannot use this environment to observe meaningful quality of service differences between CNIs. Using TCP for congestion and congestion with prioritization testing in the intra node testing is no good alternative as well. The reason for this is that we are not able set the capacity at which a test will run. This means that we are not in control of the test environment and therefore we cannot conclude much from results that are gathered this way. Therefore, the congestion and congestion with prioritization tests are only done in the inter node environment.

Another problem that we ran into is that we could not test all network performance metrics for Antrea as mentioned in section 5.2.3. This has to do with how Antrea is designed and we cannot work around this design.

5.3 Congestion

In our congestion test we will add a network stream parallel to the original stream so that we can investigate how a CNI tries to deal with congestion on itself. We create the congestion by running an iperf3 instance next to our regular test tool. This iperf3 instance will try to use the complete capacity in our environment. Due to the problems with our intra node tests this test will only be done in a inter node environment.

5.3.1 Iperf3

In Table 8 we see our iperf3 results of the congestion test. The capacity of the environment is still the same and this also holds for the bandwidth of our CNIs. The capacity is 1Gbit/s and the maximum bandwidth is 950Mbit/s for Calico, 916Mbit/s for Cilium and 912Mbit/s for Antrea.

The first thing that catches the eye in this table is that with Calico the throughput at the receiver end seems to be halved when having two network streams trying to send at maximum bandwidth. The next thing that stands

out is that both Calico streams have a package drop of 50%. We can say that Calico by default makes use of a best effort way of delivering packages, because all metrics seem to be (close to) equally divided.

When we look at Cilium and Antrea, we see that both network streams get around half of the maximum bandwidth we could reach with the CNI. From this we can conclude that both Cilium and Antrea, by default, make use of their integrated load balancer to distribute network traffic evenly. Apart from this we also see that Cilium has the least number of packages dropped and that Cilium is the most stable CNI (least amount of jitter) in general in this test setup.

	Throughput Sender in Mbit/s	**Throughput Receiver in Mbit/s**	**Jitter Receiver in ms**	**Lost Receiver in %**
Calico Stream 1	950	473,5	0,025	50
Calico Stream 2	950	473,5	0,037	50
Cilium Stream 1	460,3	459,8	0,023	0,018
Cilium Stream 2	460,3	460	0,028	0,062
Antrea Stream 1	456,2	455,7	0,047	0,060
Antrea Stream 2	456,2	455,8	0,025	0,072

Table 8 - Congestion iperf3

5.3.2 One-way active measurement protocol

We see the results of the owamp test in Figure 9. Again, no CNI experienced package loss, so this metric is omitted from the figure. Antrea has the most consistent connection, while Cilium has the quickest connection. Calico has the least consistent connection of these three CNIs and is also the slowest in this test.

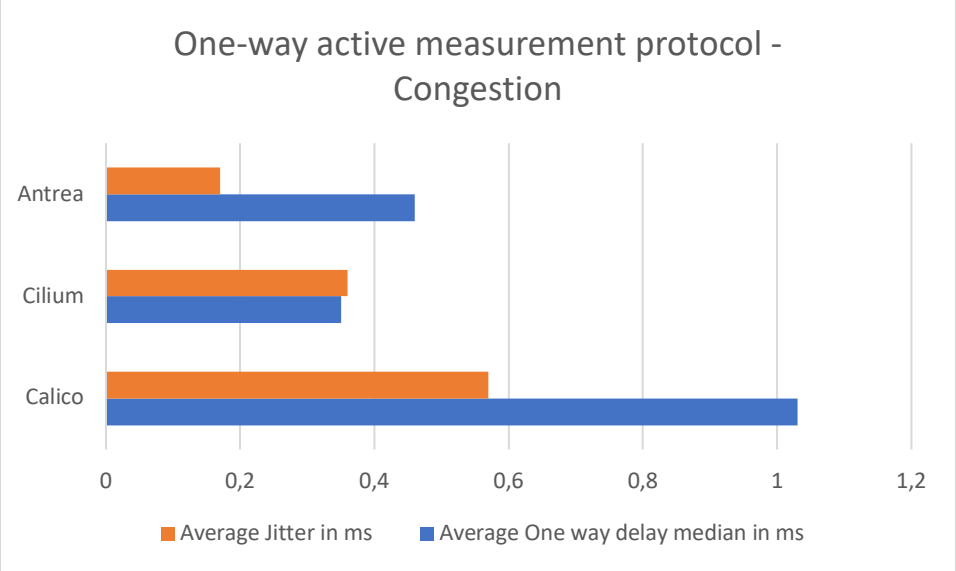

Figure 9 – Owamp and congestion

5.3.3 Two-way active measurement protocol

From the information in Table 9 we can conclude that Cilium is quicker than Calico. This is surprising as Cilium encapsulates packages before sending them.

CNI	Average Packages Lost (Round Trip Loss)	Average Round Trip median delay in ms
Calico	0	1
Cilium	0	0,62

Table 9 – Twamp and congestion

5.3.4 Summary

Both Calico network streams send at the maximum bandwidth and exceeding the capacity of this environment in our iperf3 test. This led to a package drop of 50 percent on both streams. Cilium and Antrea however, made use of their integrated load balancer. This slowed traffic at the sender, but it also led to almost no package loss compared to Calico. Cilium is the best at keeping the package drop percentage as low as possible, whilst also having the most stable connection when we look at jitter. In our owamp test we see that Antrea provides the most consistent connection, whilst Cilium is the quickest CNI, and Calico performs the worst. The twamp test could not be done for Antrea and Cilium is quicker than Calico in this test.

5.4 Congestion and prioritization

Now that we know the performance of each CNI while network traffic is congested, we want to find out if we can prioritize network traffic by adding a DSCP tag. The environment is the same as during the baseline and the congestion test. So, the capacity is still 1Gbit/s and the maximum bandwidth is 950Mbit/s for Calico, 916Mbit/s for Cilium and 912Mbit/s for Antrea.

5.4.1 Iperf3

In this test we verify the impact of adding a DSCP tag to one network stream, whilst having a parallel network stream using the same network cable to transfer data. When we look at the data we've collected in Table 10, we see that these metrics are quite like the metrics we got in our congestion without

prioritization tests. This means that adding a DSCP did not have any effect in this test.

Calico still uses the best effort approach and sends everything, which results in a fifty percent package drop on both network streams. Antrea and Cilium still make use of their integrated load balancer, but Cilium seems to do a better overall job than Antrea.

CNI	Throughput Sender in Mbit/s	Throughput Receiver in Mbit/s	Jitter Receiver in ms	Lost Receiver in %
Calico Stream 1	950	473,4	0,030	50
Calico Stream 2	950	474,4	0,033	50
Cilium Stream 1	460,7	460,5	0,023	0,039
Cilium Stream 2	460,8	460,7	0,027	0,035
Antrea Stream 1	456,2	456,2	0,026	0,034
Antrea Stream 2	456,2	456,1	0,028	0,045

Table 10 - Iperf3 congestion and prioritization

5.4.2 One-way active measurement protocol

We also added the DSCP tag to the owamp test while the environment was congested. Antrea still has the most consistent connection and this time also the fastest way of delivering packages. It seems that Cilium has some struggles with the DSCP tag during this test and scores the same as Calico.

When we compare these results with the congestion test, we see that Antrea performs the same. The one-way delay of Cilium performs worse and Calico performs slightly better than in the congestion test.

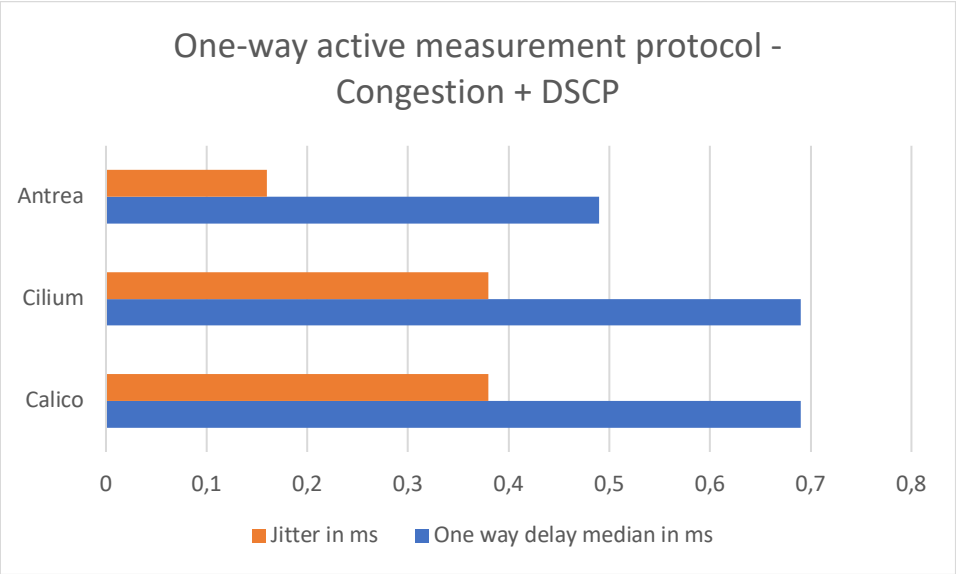

Figure 10 – Owamp and congestion with DSCP

5.4.3 Two-way active measurement protocol

In our twamp test with DSCP we see that Calico scores the same as without a DSCP tag. Just as in our owamp test Cilium performs worse when adding a DSCP tag. Cilium still has a lower average round trip delay, which means that Cilium is faster than Calico in this test. These results are similar to the results of the congestion test.

CNI	Average Packages Lost (Round Trip Loss)	Average Round Trip median delay in ms
Calico	0	1
Cilium	0	0,7

Figure 11- Twamp and congestion with DSCP

5.4.4 Summary

Now that we have the results of our third test setup we can compare this with the results of our congestion test. The iperf3 results of our congestion test can be seen in Table 8. We see that the results of the congestion and the results of our congestion with prioritization tests are roughly the same. This means that the DSCP tag did not result into better performance in the iperf3 test. In the one-way active measurement test we note a slightly increase in performance for Calico with a DSCP tag. Cilium performs slightly worse with the DSCP tag and Antrea performs roughly the same. In our two way active measurement test we see that Calico has the same performance and that Cilium performs slightly worse. From this we can conclude that we cannot use the DSCP tag to make a prioritized stream perform significantly better.

5.5 Policing and shaping of traffic

As mentioned earlier we can use the bandwidth plugin to limit the throughput or rate of our network traffic. This means that we've set the limit X and traffic above limit X is not allowed for the pod that this plugin applies to. We tested the functionality of this plugin for the three CNIs. We only have test results for egress bandwidth limiting (shaping). We don't have results for ingress bandwidth limiting (policing) as we encountered the following error 'control socket has closed unexpectedly' during the tests with Calico and Antrea, while Cilium does not support policing as a design choice.

In Figure 12 we see the bandwidth plugin at work within the Calico environment. We've set the bandwidth limit at 1Mbit/s. It seems that it takes

a bit for the bandwidth plugin to start working. Generally speaking, the bandwidth plugin functions correctly, but sometimes the limit was exceeded in time interval 1-2. After three seconds, throughput always stayed within bounds.

In Figure 13 we show the bandwidth plugin functioning with traffic that stays within the node. This graph closely represents the graph that represents the inter node test scenario. Indeed, the results don't vary much between the inter- and intra-node test cases. From this we can conclude that Calico supports this feature equally within the virtual network as well as two nodes being connected with a single physical cable.

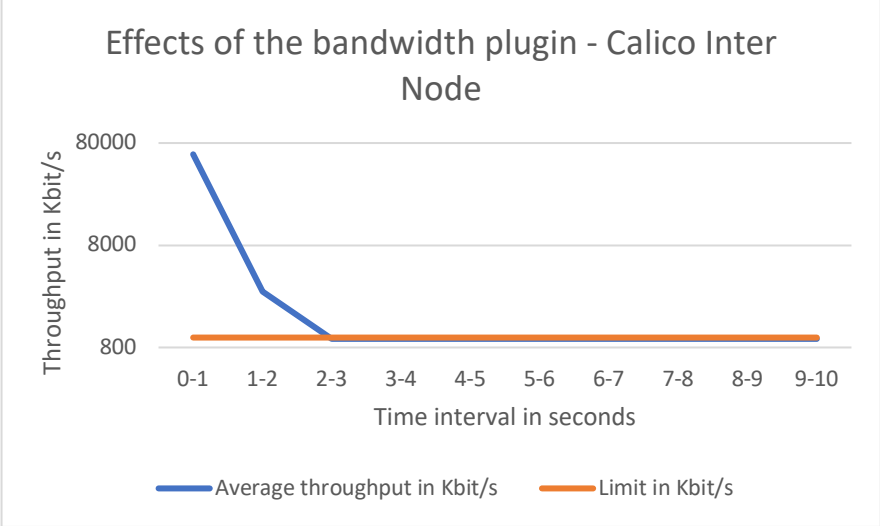

Figure 12 – Calico Bandwidth plugin inter node

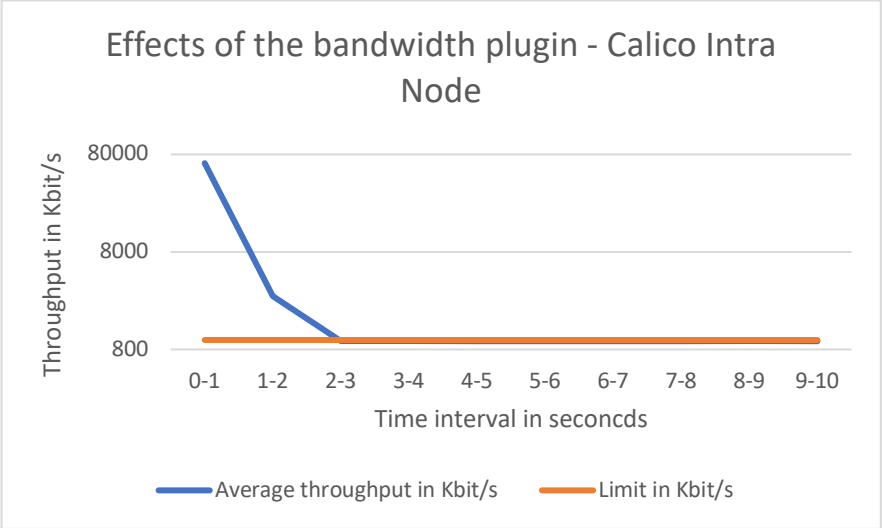

Figure 13- Calico bandwidth plugin intra node

In the case of Cilium, the documentation already says that Cilium only supports their Bandwidth Manager for inter node communications. This entails that limiting bandwidth within the node is not possible. Both cases are tested, and the Bandwidth Manager indeed does not kick in when testing intra node communications. The results of the inter node communications are shown in Figure 14. We see that the Bandwidth Manager of Cilium limits the throughput better than the Bandwidth plugin used by Calico. The maximum reached throughput is only 1,487Mbit/s with a bandwidth of 200Mbit/s, while Calico has an average throughput of 65Mbit/s in the first second of limiting.

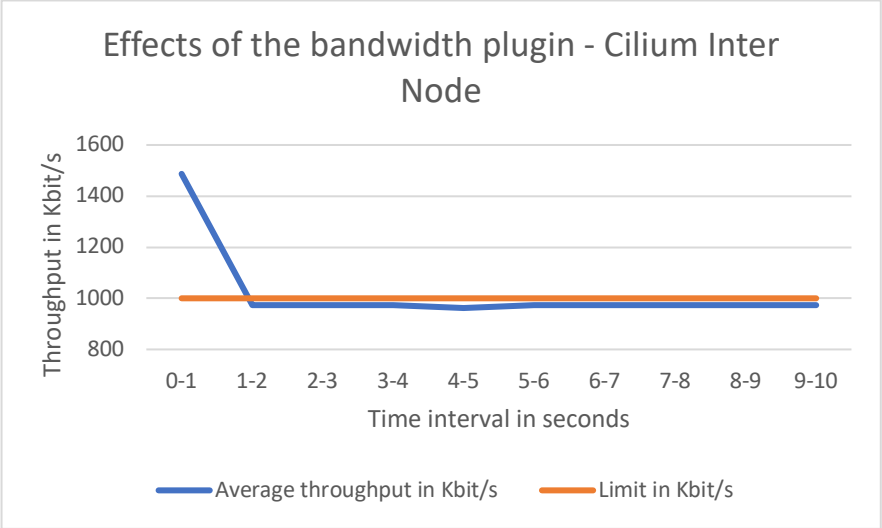

Figure 14 – Cilium bandwidth plugin

Figure 15 shows the performance of Antrea and the bandwidth plugin in a inter node test. The results look a lot like those of Calico. The bandwidth plugin does function, but it takes up to two seconds to completely stay within limits. In Figure 16 we see that the results of the intra node test are much like the inter node testing.

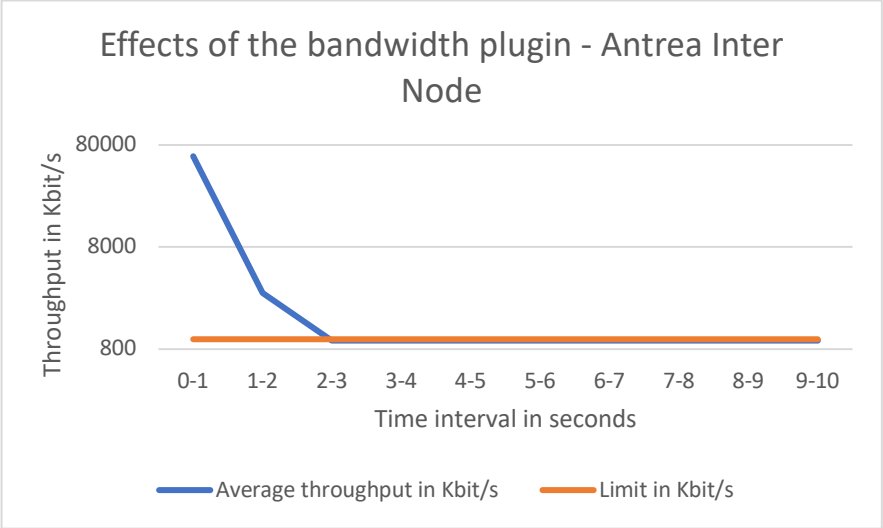

Figure 15 – Antrea Bandwidth plugin Inter Node

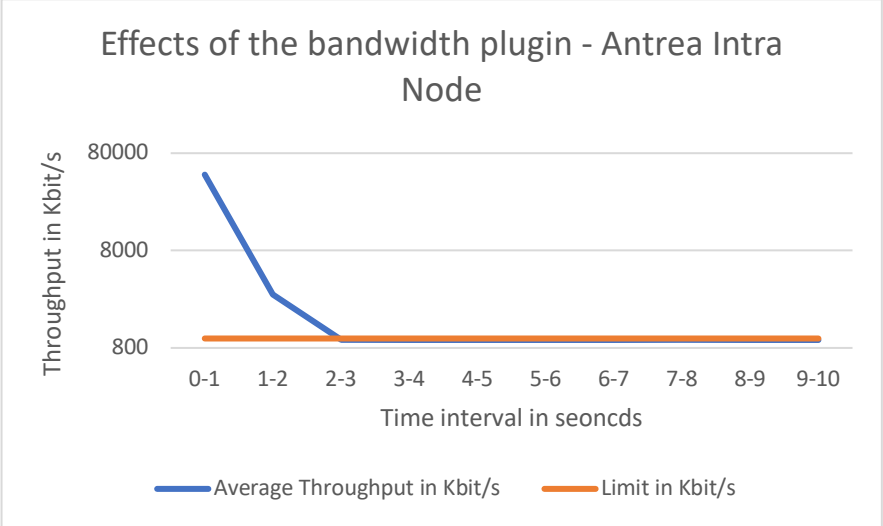

Figure 16 – Antrea Bandwidth plugin Intra Node

We can conclude that Cilium is the best at limiting the throughput of traffic. If this is an important use case within an organization then Cilium seems like a the best CNI to implement. We do need to consider however that Cilium does

not (yet) support intra node limiting of throughput and that Cilium only limits egress traffic. Policing on ingress traffic won't be implemented due to a design choice. In case intra node traffic needs to be limited, Antrea seems to limit throughput better than Calico in the first second. After the first two seconds they are both able to limit the throughput.

5.6 Summary

In this chapter we tried to safeguard quality of service within a cloud native environment. We've used two different test environments: intra- and inter node. We used three different types of CNIs for these tests; Calico as a non-encapsulating CNI, Cilium which uses encapsulation and Antrea which is a virtual switch CNI. In these environments we've done two different tests. In our first test we made use of the DSCP tag to prioritize network traffic during congestion. In our second test we evaluated shaping and policing capabilities of each CNI. Other ways of safeguarding quality of service, that can be used in a conventional network are not (yet) available. This means that safeguarding quality of service in a cloud native environment is still very primitive.

In our first test we see that adding priority to a network stream via a DSCP tag does not work. This holds for both the intra- and inter node environment. In the intra node environment we couldn't do all experiments. The testing software could not generate enough traffic to reach the capacity of this environment. Therefore, experiments that are based on congestion don't produce reliable outcomes. In our baseline test we see that Calico has the most consistent connection and the lowest percentage of dropped packages. Cilium is the fastest CNI in this environment.

In the inter node test we see that Calico has the highest bandwidth and doesn't make use of an internal load balancing mechanism to keep traffic in check. This means that more traffic can be sent than the receiving end can handle. This causes package loss. Cilium and Antrea, however, have a mechanism that does not send more traffic than can be handled. The throughput halves for both the CNIs and other NPMs are similar. In the end none of the CNIs support priority network streams via DSCP.

During our policing and shaping experiments we also encountered some problems. Cilium does not support intra node shaping, which means that Cilium cannot always be used for traffic shaping. Calico and Antrea make use of the bandwidth plugin to shape network traffic and perform similar. In the inter node environment however, Cilium is faster at shaping network traffic to within the set limits. All in all, traffic shaping does work in certain situations. In these situations traffic shaping can be used to safeguard quality of service, because it allows us to limit bandwidth usage on each pod. This way we can reserve bandwidth for our traffic with a higher priority.

Policing is not yet fully supported at the time of testing, while Cilium doesn't support policing as a design choice. Performance tests with iperf3 on Calico and Antrea did not go well. When traffic is sent at a throughput close to the policing rate, policing won't happen at all. When we raise the volume of traffic iperf3 sends traffic at high speed, which gives us more indications that the bandwidth plugin doesn't intervene. Tests subsequently time out with an error. All in all, we can conclude that policing is not yet a viable option to safeguard quality of service.

Chapter 6

6 Related work

In this chapter we discuss earlier studies that are relevant to this one. The number of studies into this subject is relatively low, due to the field being upcoming. It must also be noted that there has been some research into quality of service and Kubernetes or any other cloud native environment, but most often it does not really touch the subject of this study. Most studies investigate the automatic scaling of pods, or horizontal scaling, like we mentioned in this study, but have no real interest in quality of service with Container Network Interfaces.

6.1 Benchmarking CNIs

A study on the subject of performance benchmarking with Kubernetes CNIs has been done by Liffredo in 2020. The focus of this study did not especially focus on quality of service, but raw performance tests are performed. Liffredo described Key Performance Indicators and includes Throughput, Latency and CPU usage. As you can see this differs from our study where we focus more on quality of service and include metrics such as jitter, which tell us something about the stability of the connection.

UDP throughput in the study done by Liffredo also encounters the same problems as we have run into. Liffredo also notices slow UDP throughput in an intra-node environment, but has not found an explanation for this behavior. The tool used for these kinds of test is iperf3. I conclude that there could be two situations:

1. UDP throughput is significantly worse in an intra-node environment compared to an inter-node environment.
2. There is a bug within a testing utility which is widely used for network performance testing.

We have the idea that it could be option 2 which we mentioned in section 5.2.4. In order to fix this behavior, contacted iperf3 via their GitHub page but have gotten no answer that helped us redeem the problem[3]. This means that we cannot be sure of what causes this behavior.

6.2 Quality of Service classes

Xu, Rajamani, & Felter performed some research on implementing the existing Kubernetes quality of service classes on the subject of networking. Kubernetes implements the three classes *Garanteed*, *Burst* and *Best Effort*. These are related to CPU and memory usage. Xu et al. created these classes to give priority to network flows.

In their proceedings Xu et al. state that their implementation works as expected and that they can safeguard quality of service in this manner. Xu et al. describe how they implemented this technology. They developed this technology in such a way that it is independent of the CNI. This implementation is placed directly in the Linux kernel of a pod. Furthermore, a working product has not been distributed. In our study we focus on how quality of service can be safeguarded with the help of CNIs. This means that this solution is at odds with our research, but even if we wanted to experiment with this technology, we couldn't due to it not being distributed. Therefore, the solution that is presented by Xu et al. is out of scope for this study.

[3] https://github.com/esnet/iperf/issues/1263

Chapter 7

7 Discussion

The research problem we answered during this study is *"How can quality of service be safeguarded in a cloud native environment"*. We tried to apply tools that can be used for quality of service in a non-cloud native environment, to a cloud native environment. There were three tools that we tested:

1. Giving network traffic priority via DSCP
2. Shaping
3. Policing

The results of our tests indicate that none of these tools functioned completely, or they didn't function at all in combination with the CNIs that we tested. This means that it is not possible with the current technology to safeguard quality of service within a cloud native environment and that Thales cannot use this technology in the current form for their naval products.

Limitations of this study include that only three CNIs are tested due to time constraints. Furthermore, we couldn't perform all tests for an intra node environment, due to UDP being faster than TCP in our tests. This indicates that the maximum network capacity couldn't be utilized in this test setup. This in turn makes congestion tests in such an environment at the least unreliable and therefore these tests have not been performed. We found yet another limitation in our testing software. The testing software couldn't run tests for every CNI, due to the infrastructure one of the CNIs relies on not being supported by the two-way active measurement protocol. We do however believe that this does not have a big influence on our results as other NPMs show no difference in the prioritization of network traffic, with or without a DSCP tag.

For CNIs to support quality of service we believe that a first step to safeguarding quality of service is the ability of prioritizing network traffic. CNI vendors could implement this by making use of the work of Xu et al. mentioned in section 6.2. The three tested CNIs seem to have partial functionality on the topic of the shaping of network traffic and no functionality for the policing of traffic. We believe that shaping has somewhat more priority than policing as policed traffic most often gets retransmitted and therefore makes the environment busier. If these features would be implemented by CNI vendors, we believe that this would enable us to safeguard quality of service in a cloud native environment.

Chapter 8

8 Conclusion

In this study we set out to answer the question "*How can quality of service be safeguarded within a cloud native environment."* We started by defining what quality of service precisely entails and came up with following definition:

"Quality of service entails giving the end user a satisfiable experience, which is in line with the set organizational objectives." This can be achieved via constant monitoring, controlling the flow of network packages and the scaling of applications.

We identified three main ways of monitoring (active, passive and using SNMP-agents) and defined nine Network Performance Metrics that we use for displaying the quality of a network stream.

After the literature research we see that there are three main categories of tools that can be used to safeguard quality of service in a non-cloud native environment. These categories are:

- Classification and marking tools
- Policing, shaping and markdown tools
- Congestion management or scheduling tools

We found out that the only quality of service tools we could test for the Container Network Interface (CNI) that connects the cloud native environment are classification of network traffic via a DSCP tag and making use of policing and shaping tools. This means that in this study we've only made use of active monitoring tools to gather performance metrics. Other tools such as an Access Control List are absent in current CNIs or are too basic to have quality of service capabilities. Of course, we can scale an application in a Kubernetes environment, but this does not directly affect the CNI and is out of scope for this study.

For our experiments we used Calico, Cilium and Antrea as our CNI. In our priority/classification experiment we see that we can in fact add a DSCP tag to a network stream and thereby indicate priority for this network stream. The CNIs however, did not treat network streams any different as can be seen in our iperf3, owamp and twamp results. In some cases the DSCP tag led to slightly worse results. This means that adding a DSCP tag to a network stream did in general not affect our Network Performance Metrics. During these tests we encountered that Cilium and Antrea have a built-in mechanism for load balancing, which could be of benefit for a cloud native environment, but this is not investigated further as we weren't able to influence this behavior.

Policing should be possible for Calico and Antrea via the bandwidth plugin. We couldn't retrieve useful data for this experiment. When throughput was near our policing throughput limit, traffic wouldn't be policed. In case our policing limit was set at 1Mbit/s and throughput set at 100Mbit/s the test failed to complete, because of time outs. This indicates that policing is not a viable quality of service option in the current state.

Shaping of network streams is functional, but needs a second or two, before the CNIs start shaping network traffic. Calico and Antrea use the bandwidth plugin for shaping functionality, while Cilium has its own implementation to enable traffic shaping. In our experiments we see that Antrea and Calico perform roughly the same, but that Cilium does a better job at shaping network traffic.

Now that we have this information, we can say that at the moment it isn't possible to safeguard quality of service in a cloud native environment. The reasons are that existing tools that can be used to enable quality of service are not (yet) implemented in the CNIs that existed during the duration of this study.

9 Bibliography

Szigeti, T., Hatting, C., Barton, R., & Briley, K. (2013). *End-to-End QoS Network Design.* Indianapolis: Cisco Press.

Adis, W. (2003). Quality of service middleware. *Industrial Management & Data Systems*, 47-51.

Hashman, E. (2019). Retrieved from Usenix.

Scheepers, T. (2014). *Virtualization and containerization of application infrastructure: A comparison.*

Pahl, C. (2015). Containerization and the PaaS Cloud. *IEEE Cloud Computing Volume 2*, 24-31.

Burns, B., Beda, J., & Hightower, K. (2017). *Kubernetes.* Sebastopol: O'Reilly Media.

Ponnappan, A., Yang, L., Radhakrishna, P., & Braun, P. (2002). Policy Based QoS Management System for the IntServ/DiffServ Based Internet. *Proceedings Third International Workshop on Policies for Distributed Systems and Networks*.

Xiao, X., & Ni, L. (1999). Internet QoS: a big picture. *IEEE Network*, 8-18.

Bernet, Y. (2000). The complementary roles of RSVP and differentiated services in the full-service QoS network. *IEEE Communications Magazine*, 154-162.

Kesh, S., Nerur, S., & Ramanujan, S. (2002). Quality of service - technology and implementation. *Information Management & Computer Security*, 85-91.

Patibandla, R., Kurra, S., & Mundukur, N. (2012). A Study on Scalability of Services and Privacy Issues in Cloud Computing. *International Conference on Distributed Computing and Internet Technology*, 212-230.

Pradhan, P., Behera, P. K., & Ray, B. (2016). Modified Round Robin Algorithm for Resource Allocation in Cloud Computing. *Procedia Computer Science*, 878-890.

Lee, H.-J., Kim, M.-S., Hong, J. W., & Lee, G.-H. (2002). QoS Parameters to Network Performance Metrics Mapping for SLA Monitoring. *KNOM Review*, 42-53.

Kubernetes. (2021, 9 8). *Configure Quality of Service for Pods*. Retrieved from Kubernetes: https://kubernetes.io/docs/tasks/configure-pod-container/quality-service-pod/

Cisco. (n.d.). *QoS Modes*. Retrieved from Cisco: https://www.cisco.com/assets/sol/sb/Switches_Emulators_v2_3_5_xx/help/350_550/index.html#page/tesla_350_550_olh/ts_quality_service_27_04.html

Kubernetes. (n.d.). *Managing Resources for Containers*. Retrieved from Kubernetes: https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#resource-types

Kubernetes. (2021, 09 09). *Horizontal Pod Autoscaler*. Retrieved from Kubernetes: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

Cignoli, C. (2016, 12 12). *5 Network Metrics for a Cloud World*. Retrieved from AppNeta: https://www.appneta.com/blog/5-network-metrics-for-a-cloud-world/

NGINX. (n.d.). *What is Load Balancing? How Load Balancers Work*. Retrieved from https://www.nginx.com/resources/glossary/load-balancing/

Is5com. (n.d.). *Mapping of Classes to Queue*. Retrieved from QoS Configuration Guide: https://is5com.com/documentation/QoS-Configuration/CM-QoS-iBiome_1.8.07-1-EN/iMX_Common/Configuration_Guides/DITA_Topics/QoS/3_9_Mapp_Class_to_Queue.html

Chen, T., Zhu, Y., Gao, X., & Kong, L. (2017). Improving resource utilization via virtual machine placement in data center networks. *Mobile Networks and Applications*, 1-12.

Truong-Huu, T., Koslovski, G., Anhalt, F., & Montagnat, J. (2011). Joint Elastic Cloud and Virtual Network Framework for Application Performance-cost Optimization. *Journal of Grid Computing*, 27-47.

Medhi, D., & Ramasamy, K. (2018). *Network Routing (Second Edition).* Morgan Kaufman.

Xie, X., Yuan, T., Zhou, X., & Cheng, X. (2018). Research on Trust Model in Container-Based Cloud Service. *Computers, Materials & Continua, 56*(2), 273-283.

Clayman, S., Maini, E., Gallis, A., Manzalini, A., & Mazzocca, N. (2014). The dynamic placement of virtual network functions. *IEEE Network Operations and Management Symposium* (pp. 1-9). Krakow: IEEE.

Heggi, T., Abd El-Kader, S., Eissa, H., & Baraka, H. (2009). A New Historical Based Policing Algorithm for IP Networks. *Ubiquitous Computing and Communication Journal*.

Hidayat, T., Azzery, Y., & Mahardiko, R. (2020). Hidayat, T., Azzery, Y., & Mahardiko, R. (2020). Load Balancing Network by using Round Robin Algorithm: A Systematic Literature Review. *Journal of Interconnection Networks*, 85-89.

Raza, M., & Kidd, C. (2020). *Virtual Machines (VMs) vs Containers: What's The Difference.*

Liffredo, D. (2020). *Analysis and Benchmarking of Kubernetes Networking.* Politecnico di Torino, Torino.

Xu, C., Rajamani, K., & Felter, W. (2018). NBWGuard: Realizing Network QoS for Kubernetes. *In Proceedings of the 19th International Middleware Conference Industry*, (pp. 32-38).

He, K., Rozner, E., Kanak, A., Gu, Y., Felter, W., Carter, J., & Akella, A. (2016). AC/DC TCP: Virtual Congestion Control Enforcement for Datacenter Networks. *ACM SIGCOMM* (pp. 244-257). New York: ACM Press.

Appendix

10 Appendix A

The tables containing test results are too big to fit on standard sized pages. Therefore, I have made these results publicly available on the following GitLab page: https://gitlab.com/thesis47/quality-of-service. This repository contains the inter and intra node tests for each CNI as well as an overview of all CNIs.