Rancher CNI Usage Recommendation

Christian Frank (#473088)

April 5, 2025

Data Science & Security Analytics
Prof. Dr. Alexander Lutz
FOM - Hochschule für Oekonomie & Management
SS 2024

This paper will perform a secondary analysis of Kubernetes CNI performance data, focusing on the CNIs available in the Rancher Kubernetes Engine. It aims to make a usage recommendation to support the selection of the CNI when installing a new cluster.

Contents

List of Figures

List of Tables

List of Abbreviations

APA	American Psychological Association
BGP	Border Gateway Protocol
CNCF	Cloud Native Computing Foundation
CNI	Container Network Interface
CPU	Central Processing Unit
CSV	Comma-Separated Values
eBPF	extended Berkeley Packet Filter
EDA	Exploratory Data Analysis
IP	RFC 791 Internet Protocol
K8s	Kubernetes
KPI	Key Performance Indicator
LAN	Local Area Network
MHA	My Hero Academia
MTU	Maximum Transmission Unit
NIC	Network Interface Card
NIST	National Institute of Standards and Technology
RKE	Rancher Kubernetes Engine
TCP	Transmission Control Protocol
TSV	Tab-Separated Values
UDP	User Datagram Protocol
VXLAN	Virtual Extensible LAN

1 Introduction

1.1 Original Article

In April 2024, Alexis Ducastel of InfraBuilder published their benchmark results for Kubernetes network plugins.[1] This article continues a series of published benchmarks they had published in the years before. I obtained permission from Alexis Ducastel to use the historical benchmark data to explore the evolution of Kubernetes networking over time.

1.2 Kubernetes

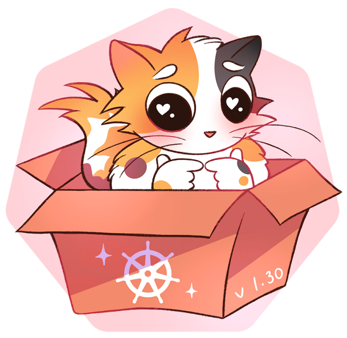
Kubernetes, or K8s, is an open-source system designed to automate deploying, scaling, and managing applications built using containers. Containers package software in a standardized unit that includes all dependencies the software needs to run, like code, libraries, and settings. This makes them portable and efficient.

Kubernetes helps manage these containers by grouping them logically. This makes it easier to track and manage complex applications with many containers. The original inspiration for Kubernetes came from Google's internal container orchestration system, Borg.[2]

In 2015, Kubernetes reached the 1.0 milestone, and in 2016, it was donated to the CNCF; the current release of Kubernetes is 1.30.

"For the people who built it, for the people who release it, and for the furries who keep all of our clusters online, we present to you Kubernetes v1.30: Uwubernetes, the cutest release to date."[3]

Figure 1: Kubernetes 1.30 Release Logo



[1]See *Ducastel, A. (2024)*: Benchmark results of Kubernetes network plugins. [6]
[2]See *Gemini (2024)*: What is Kubernetes. [11]
[3]*Dsouza, A. (2024)*: Kubernetes 1.30. [5]

1.3 Container Network Interfaces

CNI plugins are essential components of Kubernetes clusters and are responsible for managing network connectivity between pods. They provide the underlying infrastructure for pod communication within and outside the cluster. Kubernetes offers a variety of CNI plugins, each with distinct features and performance characteristics, allowing administrators to select the optimal solution based on specific cluster requirements.[4]

In this paper, we will focus on the four CNI plugins available for the SUSE Rancher Kubernetes Engine:[5]

- Flannel
- Calico
- Canal
- Cilium

1.3.1 Flannel

Flannel is the oldest CNI in the list and is a well-established overlay network. It provides a Layer 3 network fabric for Kubernetes clusters. The simple and flat nature of the overlay network allows for easy troubleshooting. Its most commonly used transport backend is VXLAN.[6]

1.3.2 Calico

Calico by Tigera uses IP routing and iptables for its data path and can create separate networks for various workloads. Calico is not a flat network, so it uses BGP to establish the routes between the nodes in a given Kubernetes cluster. Calico provides network policies and supports a variety of data planes.

1.3.3 Canal

Canal, also by Tigera, combines the Calico routing and policy engine with the Flannel transport. For RKE, Canal is the default CNI as it offers a VXLAN transport backend and network policies.

[4]See *Kubernetes (2024)*: Network Plugins. [3]
[5]See *SUSE (2024)*: Network Options. [20]
[6]See *Frank, C. (2020)*: Behind the scenes of Flannel. [7]

1.3.4 Cilium

Cilium by Isovalent and now Cisco is the newest entry in the list of available CNIs for RKE. Cilium uses a data plane based on eBPF and focuses on large networks and high network throughput.

1.4 Research Question

This paper will use data exploration techniques to guide which CNI to select for RKE2 based on current and historical performance data.[7] To deliver guidance, we will focus on bandwidth, CPU usage, and memory consumption.

1.5 Gender-neutral Pronouns

Our society is becoming more open, inclusive, and gender-fluid, and now I think it's time to think about using gender-neutral pronouns in scientific texts, too. Two well-known researchers, Abigail C. Saguy and Juliet A. Williams, both from UCLA, propose to use the singular they/them instead: "The universal singular they is inclusive of people who identify as male, female or nonbinary."[8] The aim is to support an inclusive approach in science through gender-neutral language.

We'll attempt to follow this suggestion in this paper, and I invite all our readers to do the same for future articles. Thank you!

If you're not sure about the definitions of gender and sex and how to use them, have a look at the definitions[9] by the American Psychological Association.

1.6 Climate Emergency

As Professor Rahmstorf puts it: "Without immediate, decisive climate protection measures, my children currently attending high school could already experience a 3-degree warmer Earth. No one can say exactly what this world would look like—it would be too far outside the entire experience of human history. But almost certainly, this earth would be full of horrors for the people who would have to experience it."[10]

[7]See *Tukey, J.W. (1977)*: Exploratory data analysis. [21]
[8]*Saguy, A. (2020)*: Why We Should All Use They/Them Pronouns. [18]
[9]See *APA (2021)*: Definitions Related to Sexual Orientation. [1]
[10]*Rahmstorf, A. (2024)*: Climate and Weather at 3 Degrees More. [17]

2 Data Sources and Research Methods

2.1 Original Data

For this paper's data analysis and visualization, we will use raw benchmark data from the original author's Github pages for the years 2024, 2021, and 2020 and perform a secondary data analysis.[11] We will use the previously collected measurements instead of gathering new data to answer our research question.

In their 2024 article, Alexis Ducastel posed the same question and recommended Cilium as the preferred CNI for RKE2. Their conclusion was based on a combination of functionality and performance.

2.2 Data Wrangling

To support the secondary analysis, we had to convert the original raw data into a standard format. The CSV format is widely supported in statistical analysis, so we chose this as our target format; most major programming languages (e.g., Python, R) and tools (e.g., Tableau, Excel) support CSV natively.

2.2.1 2020 and 2021 - Adding Header information

The original raw data for 2020 and 2021 are in tab-separated files without headers, split by CNI; they do not contain header information.

From the spreadsheets with overall aggregated results that are available on the author's Github, we added the following headers to the TSV Files and subsequently converted the data files to CSV format:

- smem - "Server Memory (MB)"
- scpu - "Server CPU (%)"
- cmem - "Client Memory (MB)"
- ccpu - "Client CPU (%)"
- tcpbw - "TCP Pod to Pod Bandwidth (Mbit/s)"
- tcpsm - "Server Memory (MB)"
- tcpsc - "Server CPU (%)"

[11]See *Jillier, W. (2021)*: A Guide To Secondary Data Analysis. [12]

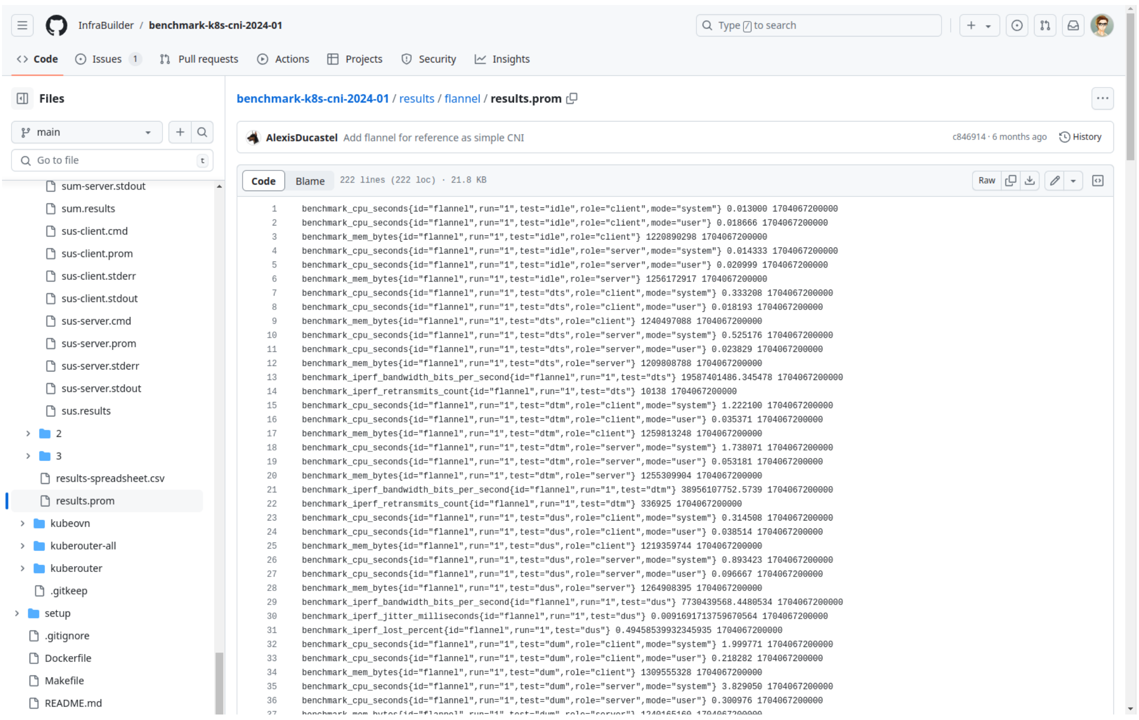
- tcpcm - "Client Memory (MB)"
- tcpcc - "Client CPU (%)"
- udpbw - "UDP Pod to Pod Bandwidth (Mbit/s)"
- udpsm - "Server Memory (MB)"
- udpsc - "Server CPU (%)"
- udpcm - "Client Memory (MB)"
- udpcc - "Client CPU (%)"
- tcpebw - "TCP Pod to Service Bandwidth (Mbit/s)"
- tcpesm - "Server Memory (MB)"
- tcpesc - "Server CPU (%)"
- tcpecm - "Client Memory (MB)"
- tcpecc - "Client CPU (%)"
- udpebw - "UDP Pod to Service Bandwidth (Mbit/s)"
- udpesm - "Server Memory (MB)"
- udpesc - "Server CPU (%)"
- udpecm - "Client Memory (MB)"
- udpecc - "Client CPU (%)"

We also removed the discovered MTU size from the measurements, as it was a constant value across observations and was not present in the 2024 data.

2.2.2 2024 - Data Transformation

The original raw data for 2024 is formatted as Prometheus gauges, and the data sets are also split by CNI.

Figure 2: 2024 Flannel Results (GitHub)



benchmark-k8s-cni-2024-01 / results / flannel / results.prom

AlexisDucastel Add flannel for reference as simple CNI

```
benchmark_cpu_seconds{id="flannel",run="1",test="idle",role="client",mode="system"} 0.013000 1704067200000
benchmark_cpu_seconds{id="flannel",run="1",test="idle",role="client",mode="user"} 0.018666 1704067200000
benchmark_mem_bytes{id="flannel",run="1",test="idle",role="client"} 1220890298 1704067200000
benchmark_cpu_seconds{id="flannel",run="1",test="idle",role="server",mode="system"} 0.014333 1704067200000
benchmark_cpu_seconds{id="flannel",run="1",test="idle",role="server",mode="user"} 0.020999 1704067200000
benchmark_mem_bytes{id="flannel",run="1",test="idle",role="server"} 1256172917 1704067200000
benchmark_cpu_seconds{id="flannel",run="1",test="dts",role="client",mode="system"} 0.333208 1704067200000
benchmark_cpu_seconds{id="flannel",run="1",test="dts",role="client",mode="user"} 0.018193 1704067200000
benchmark_mem_bytes{id="flannel",run="1",test="dts",role="client"} 1240497088 1704067200000
benchmark_cpu_seconds{id="flannel",run="1",test="dts",role="server",mode="system"} 0.525176 1704067200000
benchmark_cpu_seconds{id="flannel",run="1",test="dts",role="server",mode="user"} 0.023829 1704067200000
benchmark_mem_bytes{id="flannel",run="1",test="dts",role="server"} 1209808788 1704067200000
benchmark_iperf_bandwidth_bits_per_second{id="flannel",run="1",test="dts"} 19587401486.345478 1704067200000
benchmark_iperf_retransmits_count{id="flannel",run="1",test="dts"} 10138 1704067200000
benchmark_cpu_seconds{id="flannel",run="1",test="dtm",role="client",mode="system"} 1.222100 1704067200000
benchmark_cpu_seconds{id="flannel",run="1",test="dtm",role="client",mode="user"} 0.035371 1704067200000
benchmark_mem_bytes{id="flannel",run="1",test="dtm",role="client"} 1259813248 1704067200000
benchmark_cpu_seconds{id="flannel",run="1",test="dtm",role="server",mode="system"} 1.738071 1704067200000
benchmark_cpu_seconds{id="flannel",run="1",test="dtm",role="server",mode="user"} 0.053181 1704067200000
benchmark_mem_bytes{id="flannel",run="1",test="dtm",role="server"} 1255309904 1704067200000
benchmark_iperf_bandwidth_bits_per_second{id="flannel",run="1",test="dtm"} 38956107752.5739 1704067200000
benchmark_iperf_retransmits_count{id="flannel",run="1",test="dtm"} 336925 1704067200000
benchmark_cpu_seconds{id="flannel",run="1",test="dus",role="client",mode="system"} 0.314508 1704067200000
benchmark_cpu_seconds{id="flannel",run="1",test="dus",role="client",mode="user"} 0.038514 1704067200000
benchmark_mem_bytes{id="flannel",run="1",test="dus",role="client"} 1219359744 1704067200000
benchmark_cpu_seconds{id="flannel",run="1",test="dus",role="server",mode="system"} 0.893423 1704067200000
benchmark_cpu_seconds{id="flannel",run="1",test="dus",role="server",mode="user"} 0.096667 1704067200000
benchmark_mem_bytes{id="flannel",run="1",test="dus",role="server"} 1264908395 1704067200000
benchmark_iperf_bandwidth_bits_per_second{id="flannel",run="1",test="dus"} 7730439568.4480534 1704067200000
benchmark_iperf_jitter_milliseconds{id="flannel",run="1",test="dus"} 0.009169171375967056 4 1704067200000
benchmark_iperf_lost_percent{id="flannel",run="1",test="dus"} 0.49458539932345935 1704067200000
benchmark_cpu_seconds{id="flannel",run="1",test="dum",role="client",mode="system"} 1.999771 1704067200000
benchmark_cpu_seconds{id="flannel",run="1",test="dum",role="client",mode="user"} 0.218282 1704067200000
benchmark_mem_bytes{id="flannel",run="1",test="dum",role="client"} 1309555328 1704067200000
benchmark_cpu_seconds{id="flannel",run="1",test="dum",role="server",mode="system"} 3.829050 1704067200000
benchmark_cpu_seconds{id="flannel",run="1",test="dum",role="server",mode="user"} 0.300976 1704067200000
```

As the 2024 data format vastly differs, we converted the Prometheus gauges for easier handling. The 2024 data also has many more data points, so we mapped the gauges to the most appropriate fields and adjusted the scale where necessary (e.g., Bytes to Megabytes).

- smem - "benchmark_mem_bytes - server - idle"
- scpu - "benchmark_cpu_seconds - server - idle"
- cmem - "benchmark_mem_bytes - client - idle"
- ccpu - "benchmark_cpu_seconds - client - idle"
- tcpbw - "benchmark_iperf_bandwidth_bits_per_second - dts"
- tcpsm - "benchmark_mem_bytes - server - dts"
- tcpsc - "benchmark_cpu_seconds - server - dts"
- tcpcm - "benchmark_mem_bytes - client - dts"
- tcpcc - "benchmark_cpu_seconds - client - dts"
- udpbw - "benchmark_iperf_bandwidth_bits_per_second - dus"

- udpsm - "benchmark_mem_bytes - server - dus"
- udpsc - "benchmark_cpu_seconds - server - dus"
- udpcm - "benchmark_mem_bytes - client - dus"
- udpcc - "benchmark_mem_bytes - client - dus"
- tcpebw - "benchmark_iperf_bandwidth_bits_per_second - sts"
- tcpesm - "benchmark_mem_bytes - server - sts"
- tcpesc - "benchmark_cpu_seconds - server - sts"
- tcpecm - "benchmark_mem_bytes - client - sts"
- tcpecc - "benchmark_cpu_seconds - client - sts"
- udpebw - "benchmark_iperf_bandwidth_bits_per_second - sus"
- udpesm - "benchmark_mem_bytes - server - sus"
- udpesc - "benchmark_cpu_seconds - server - sus"
- udpecm - "benchmark_mem_bytes - client - sus"
- udpecc - "benchmark_mem_bytes - client - sus"

2.3 What's Not in the Data

The benchmark data was initially recorded with the author's k8s-bench-suite. Their test suite uses iPerf3 for measurements and provides a convenient shell interface for data collection. iPerf3 is a tool for active measurements of the maximum achievable bandwidth on IP networks and is most suitable for measuring CNI bandwidth.

The data itself does not contain information on the infrastructure setup and the software versions used, so we record the information here:

- The 2020 setup was three Supermicro bare-metal servers connected through a Supermicro 10Gbit switch, running Ubuntu 18.04 LTS and Kubernetes 1.19.0.

Table 1: 2020 CNI Versions

Flannel	Calico	Canal	Cilium
v0.12.0	v3.16.1	v3.16.1	v1.8.2

- The 2021 setup was three Supermicro bare-metal servers connected through a Supermicro 10Gbit switch, running Ubuntu 20.04 LTS and Kubernetes 1.21.0.

Table 2: 2021 CNI Versions

Flannel	Calico	Canal	Cilium
v0.15.1	v3.19.2	v3.19.2	v1.11.2

- For 2024, the test infrastructure consisted of three Supermicro bare-metal servers connected through a Supermicro 40Gbit switch, running Ubuntu 22.04 LTS and Kubernetes 1.26.12.[12]

Table 3: 2024 CNI Versions

Flannel	Calico	Canal	Cilium
v0.24.3	v3.27.2	v3.27.2	v1.15.2

2.4 Data Exploration Method

We will analyze the data sets using the Exploratory Data Analysis (EDA) method.

John Tukey has promoted EDA since 1977 to encourage data scientists to explore data and formulate hypotheses that could lead to new data collection and experiments.[13]

EDA uses statistical methods to analyze and visualize the data, understand key characteristics, and identify patterns and relationships.

We will use summary statistics, such as mode, mean, and median, to understand the data and augment this with data visualization, e.g., histograms. We have already cleaned up the data and converted the data sets to a common format. EDA should help us to make informed decisions about how to proceed with our analysis and draw meaningful conclusions from the data.[14]

We will not use the data to predict performance gains in future releases of the CNIs or through improved network speeds; instead, we will focus on our research question and attempt to arrive at a recommendation for CNI usage.

2.5 Tools

We performed most of the initial data wrangling with Bash and vi, and Microsoft Excel to support operations on columns and the final weighted ranking.

[12]See *Ducastel, A. (2024)*: Benchmark results of Kubernetes network plugins. [6]
[13]See *Tukey, J.W. (1977)*: Exploratory data analysis. [21]
[14]See *Gemini (2024)*: Exploratory Data Analysis. [9]

For the subsequent data exploration, we will mainly use R and RStudio.

```
version
```

```
platform       x86_64-w64-mingw32
arch           x86_64
os             mingw32
crt            ucrt
system         x86_64, mingw32
status
major          4
minor          2.3
year           2023
month          03
day            15
svn rev        83980
language       R
version.string R version 4.2.3 (2023-03-15 ucrt)
nickname       Shortstop Beagle
```

3 Data Exploration

3.1 Data Layout

The data consists of twelve datasets overall, one per year (3) and one per CNI (4).

After our data wrangling, all datasets now contain the following five blocks of metrics in a single row per observation:

- Idle server memory and CPU consumption
- TCP Pod-to-Pod bandwidth, memory, and CPU consumption
- UDP Pod-to-Pod bandwidth, memory, and CPU consumption
- TCP Pod-to-Server bandwidth, memory, and CPU consumption
- UDP Pod-to-Server bandwidth, memory, and CPU consumption

In our data, as defined by the original measurement setup, Pod-to-Pod traffic refers to traffic between Pods in the same cluster, and Pod-to-Server traffic refers to traffic from a Pod to an outside application.

The individual metric names are derived from each dataset's first row and are consistent across all sets.

3.2 2020

3.2.1 Flannel

The first data set we will look at is the 2020 data set for Flannel, which we will read using R's built-in read.csv function.

```r
main <- "."
cni <- read.csv(here(main,"data","2020-flannel.csv"))
```

In the next step, we will have a look at the structure of our data and visually inspect the measurements.

```r
str(cni)
```

```
'data.frame':   3095 obs. of  24 variables:
 $ smem  : num  593 596 590 591 588 ...
 $ scpu  : num  0.682 0.679 0.662 0.642 0.728 ...
 $ cmem  : num  588 595 591 587 592 ...
 $ ccpu  : num  0.572 0.601 0.54 0.517 0.579 ...
 $ tcpbw : num  9675 9757 9693 9689 9773 ...
 $ tcpsm : num  590 589 593 592 593 ...
 $ tcpsc : num  5.16 5.16 5.18 5.2 5.12 ...
 $ tcpcm : num  593 591 591 593 593 ...
 $ tcpcc : num  5.17 5.1 5.02 5.04 5.23 ...
 $ udpbw : num  9845 9842 9842 9841 9840 ...
 $ udpsm : num  591 590 592 594 589 ...
 $ udpsc : num  5.49 5.47 5.44 5.42 5.5 ...
 $ udpcm : num  608 611 608 613 614 ...
 $ udpcc : num  12.9 12.9 12.9 12.9 13 ...
 $ tcpebw: num  9831 9828 9827 9826 9838 ...
 $ tcpesm: num  589 596 591 590 589 ...
 $ tcpesc: num  4.92 5.07 5.08 5.03 5.14 ...
 $ tcpecm: num  595 601 595 595 593 ...
 $ tcpecc: num  5.47 5.55 5.46 5.52 5.29 ...
 $ udpebw: num  9823 9833 9818 9818 9833 ...
 $ udpesm: num  594 596 594 593 597 ...
 $ udpesc: num  5.25 5.3 5.34 5.41 5.23 ...
 $ udpecm: num  601 601 603 604 600 ...
 $ udpecc: num  12.6 12.7 12.7 12.7 12.6 ...
```

The various values for memory consumption (MB), CPU usage (Percentage), and bandwidth (MBit/s) look fine at first glance and in line with our expectations.

For further analysis, we will inspect the values a bit more closely, starting with memory consumption. First, we'll look at the values for an idle server and get:

```
median(cni$smem)
```

```
[1] 592.8159
```

```
mean(cni$smem)
```

```
[1] 593.1625
```

```
min(cni$smem)
```

```
[1] 585
```

```
max(cni$smem)
```

```
[1] 606
```

Comparing the idle memory consumption with the other measurements for memory consumption (TCP and UDP, and Pod-to-Pod and Pod-to-Server), we get a fairly consistent picture:

```r
median(cni$tcpsm)
```

```
[1] 591.2724
```

```r
median(cni$udpsm)
```

```
[1] 591.5915
```

```r
median(cni$tcpesm)
```

```
[1] 590.1462
```

```r
median(cni$udpesm)
```

```
[1] 594.3655
```

Next, we will check CPU consumption, starting again with idle values:

```r
median(cni$scpu)
```

```
[1] 0.6546849
```

```r
mean(cni$scpu)
```

```
[1] 0.6695833
```

```r
min(cni$scpu)
```

```
[1] 0.6
```

```r
max(cni$scpu)
```

```
[1] 0.82
```

When we validate this with the other measurements for CPU consumption, we get again fairly consistent values across the measurements, but with a clear differentiation to the idle values as we would expect:

```r
median(cni$tcpsc)
```

```
[1] 5.162688
```

```r
median(cni$udpsc)
```

```
[1] 5.483832
```

```
median(cni$tcpesc)
```

```
[1] 5.055657
```

```
median(cni$udpesc)
```

```
[1] 5.30168
```

Idle CPU consumption does not seem to be a good overall performance indicator.

As the Median is more robust against outliers and works better with data that might be skewed, we'll concentrate on the median values going forward.[15]

The final data point we want to look at is bandwidth, the core performance metric for a network:[16]

```
median(cni$tcpbw)
```

```
[1] 9705.809
```

```
median(cni$udpbw)
```

```
[1] 9843.559
```

```
median(cni$tcpebw)
```

```
[1] 9828.35
```

```
median(cni$udpebw)
```

```
[1] 9822.046
```

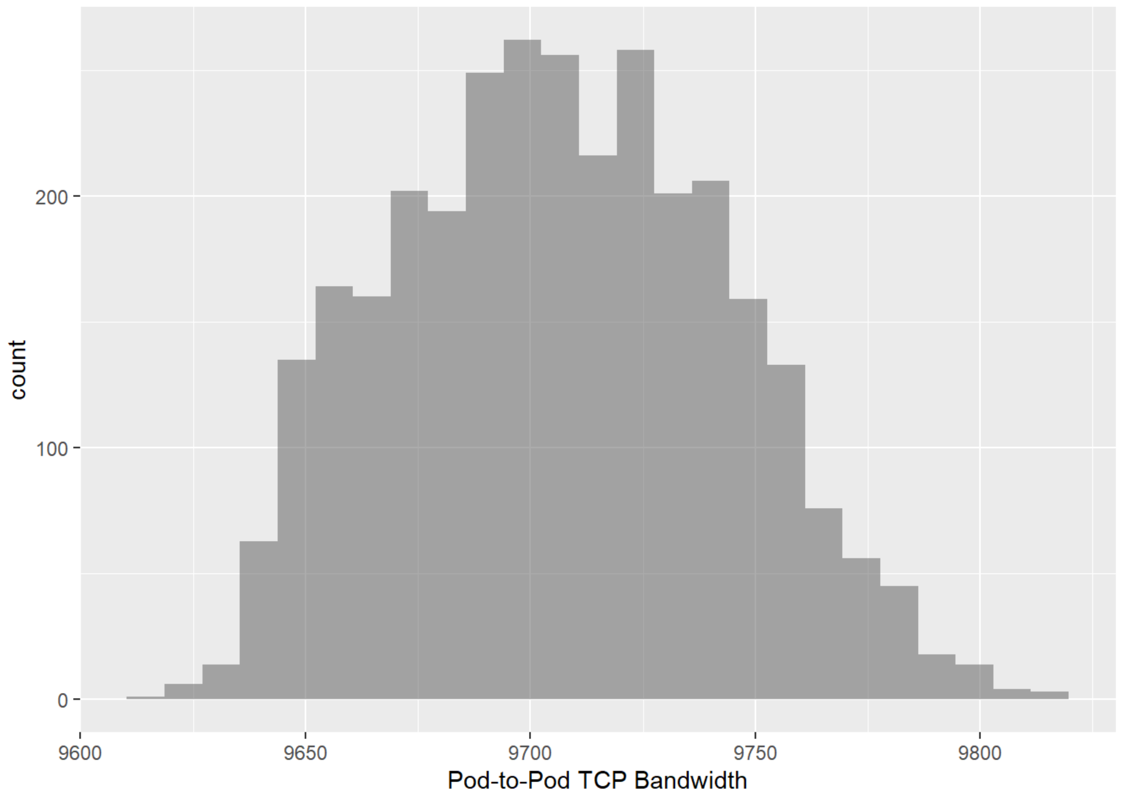
With the majority of the internet traffic being TCP, we will focus on the measurements for TCP for analysis and comparison and leave the UDP bandwidth values aside.[17]

We visualize both the TCP Pod-to-Pod and Pod-to-Server traffic bandwidth using histograms:

[15]See *Orn, A. (2023)*: Means and Medians: When To Use Which. [15]
[16]See *Solarwinds (2024)*: What are Network Performance Metrics? [19]
[17]See *Quian, L. (2012)*: A flow-based performance analysis of TCP and TCP applications. [16]

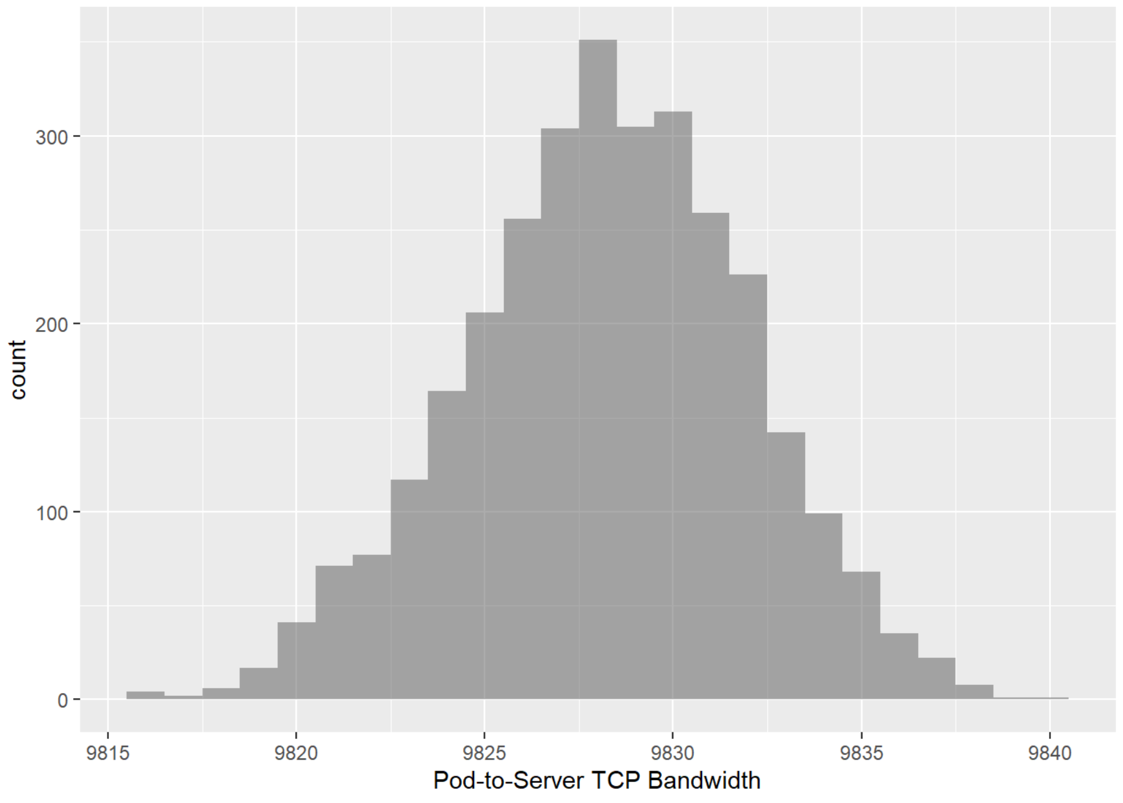
```
gf_histogram(~tcpbw, data = cni)
```

Figure 3: Flannel Pod-to-Pod Bandwidth



```
gf_histogram(~tcpebw, data = cni)
```

Figure 4: Flannel Pod-to-Server Bandwidth



The data distribution supports our choice to use the median values as key indicators for TCP bandwidth.

Following the results of a scientific coin toss,[18] we will use the TCP Pod-to Server values for CPU usage and memory consumption for further analysis.

From the 2020 Flannel dataset, we will thus use the following values:

```
median(cni$tcpesm)
```

```
[1] 590.1462
```

```
median(cni$tcpesc)
```

```
[1] 5.055657
```

[18]See *Kim, S.E. (2024)*: Scientists Destroy Illusion That Coin Toss Flips Are 50–50. [14]

```
median(cni$tcpbw)
```

```
[1] 9705.809
```

```
median(cni$tcpebw)
```

```
[1] 9828.35
```

3.2.2 Calico

We will now read the 2020 data set for Calico.

```
main <- "."
cni <- read.csv(here(main,"data","2020-calico.csv"))
```

In the next step, we will examine the structure of our data and visually inspect the measurements.

```
str(cni)
```

```
'data.frame':   3091 obs. of  24 variables:
 $ smem  : num  667 665 666 669 665 ...
 $ scpu  : num  1.64 1.64 1.62 1.64 1.63 ...
 $ cmem  : num  666 666 666 653 666 ...
 $ ccpu  : num  1.27 1.27 1.31 1.3 1.29 ...
 $ tcpbw : num  8879 8885 8884 8882 8884 ...
 $ tcpsm : num  660 664 662 664 660 ...
 ...
```

Then, we'll extract the key metrics for further analysis:

```
median(cni$tcpesm)
```

```
[1] 659.3846
```

```
median(cni$tcpesc)
```

```
[1] 6.44727
```

```
median(cni$tcpbw)
```

```
[1] 8882.457
```

```
median(cni$tcpebw)
```

```
[1] 8675.868
```

3.2.3 Canal

We will now read the 2020 data set for Canal.

```
main <- "."
cni <- read.csv(here(main,"data","2020-canal.csv"))
```

In the next step, we will examine the structure of our data and visually inspect the measurements.

```
str(cni)
```

```
'data.frame':   3076 obs. of  24 variables:
 $ smem  : num  662 659 663 663 663 ...
 $ scpu  : num  1.43 1.56 1.49 1.47 1.43 ...
 $ cmem  : num  663 663 660 666 659 ...
 $ ccpu  : num  1.97 2.01 1.98 1.99 1.96 ...
 $ tcpbw : num  8490 8586 8635 8647 8648 ...
 $ tcpsm : num  659 656 655 657 657 ...
 ...
```

Then, we'll extract the key metrics for further analysis:

```
median(cni$tcpesm)
```

```
[1] 655.4456
```

```
median(cni$tcpesc)
```

```
[1] 6.697056
```

```
median(cni$tcpbw)
```

```
[1] 8634.413
```

```
median(cni$tcpebw)
```

```
[1] 8576.709
```

3.2.4 Cilium

We will now read the 2020 data set for Cilium.

```
main <- "."
cni <- read.csv(here(main,"data","2020-cilium.csv"))
```

In the next step, we will examine the structure of our data and visually inspect the measurements.

```
str(cni)
```

```
'data.frame':   3082 obs. of  24 variables:
 $ smem  : num  859 867 863 857 863 ...
 $ scpu  : num  3.95 3.68 2.63 3.32 3.53 ...
 $ cmem  : num  867 872 871 868 864 ...
 $ ccpu  : num  1.78 1.74 1.69 1.73 1.76 ...
 $ tcpbw : num  9445 9547 9467 9449 9466 ...
 $ tcpsm : num  861 867 864 864 864 ...
 ...
```

Then, we'll extract the key metrics for further analysis:

```
median(cni$tcpesm)
```

```
[1] 866.5502
```

```
median(cni$tcpesc)
```

```
[1] 13.2222
```

```
median(cni$tcpbw)
```

```
[1] 9475.213
```

```
median(cni$tcpebw)
```

```
[1] 9673.348
```

3.3 2021

3.3.1 Flannel

We will now read the 2021 data set for Flannel.

```
main <- "."
cni <- read.csv(here(main,"data","2021-flannel.csv"))
```

In the next step, we will examine the structure of our data and visually inspect the measurements.

```
str(cni)
```

```
'data.frame':   3112 obs. of  24 variables:
 $ smem  : num  591 596 599 593 599 ...
 $ scpu  : num  0.67 0.665 0.699 0.654 0.699 ...
 $ cmem  : num  596 592 591 589 580 ...
 $ ccpu  : num  0.539 0.502 0.546 0.554 0.55 ...
 $ tcpbw : num  9657 9737 9680 9755 9689 ...
 ...
```

Then, we'll extract the key metrics for further analysis:

```
median(cni$tcpesm)
```

```
[1] 590.5901
```

```
median(cni$tcpesc)
```

```
[1] 5.015945
```

```
median(cni$tcpbw)
```

```
[1] 9695.797
```

```
median(cni$tcpebw)
```

```
[1] 9825.204
```

The difference between the measurements in the 2020 and 2021 data sets is not that big; the data was taken only a couple of months apart, and the CNI versions were relatively close to each other.

3.3.2 Calico

We will now read the 2021 data set for Calico.

```
main <- "."
cni <- read.csv(here(main,"data","2021-calico.csv"))
```

In the next step, we will examine the structure of our data and visually inspect the measurements.

```
str(cni)
```

```
'data.frame':   3103 obs. of  24 variables:
 $ smem  : num  663 669 663 665 667 ...
 $ scpu  : num  1.56 1.56 1.64 1.6 1.61 ...
 $ cmem  : num  666 664 665 669 667 ...
 $ ccpu  : num  1.25 1.24 1.25 1.27 1.24 ...
 $ tcpbw : num  8868 8875 8873 8867 8880 ...
 $ tcpsm : num  664 665 663 659 663 ...
 ...
```

Then, we'll extract the key metrics for further analysis:

```
median(cni$tcpesm)
```

```
[1] 661.9909
```

```
median(cni$tcpesc)
```

```
[1] 6.366529
```

```
median(cni$tcpbw)
```

```
[1] 8876.478
```

```
median(cni$tcpebw)
```

```
[1] 8763.889
```

3.3.3 Canal

We will now read the 2021 data set for Canal.

```
main <- "."
cni <- read.csv(here(main,"data","2021-canal.csv"))
```

In the next step, we will examine the structure of our data and visually inspect the measurements.

```
str(cni)
```

```
'data.frame':   3115 obs. of  24 variables:
 $ smem  : num  657 661 660 663 656 ...
 $ scpu  : num  1.49 1.51 1.47 1.51 1.51 ...
 $ cmem  : num  653 658 666 656 651 ...
 $ ccpu  : num  1.99 1.86 1.94 2.01 2.01 ...
 $ tcpbw : num  8638 8636 8627 8639 8605 ...
 $ tcpsm : num  652 653 659 659 656 ...
 ...
```

Then, we'll extract the key metrics for further analysis:

```r
median(cni$tcpesm)
```

```
[1] 659.3958
```

```r
median(cni$tcpesc)
```

```
[1] 6.66032
```

```r
median(cni$tcpbw)
```

```
[1] 8612.275
```

```r
median(cni$tcpebw)
```

```
[1] 8579.366
```

3.3.4 Cilium

We will now read the 2021 data set for Cilium.

```r
main <- "."
cni <- read.csv(here(main,"data","2021-cilium.csv"))
```

In the next step, we will examine the structure of our data and visually inspect the measurements.

```r
str(cni)
```

```
'data.frame':   3105 obs. of  24 variables:
 $ smem  : num  871 868 868 867 868 ...
 $ scpu  : num  3.25 2.85 3.31 3.3 3.67 ...
 $ cmem  : num  864 869 866 880 879 ...
 $ ccpu  : num  1.69 1.7 1.66 1.65 1.72 ...
 $ tcpbw : num  9450 9458 9440 9532 9450 ...
 $ tcpsm : num  865 863 863 866 868 ...
 ...
```

Then, we'll extract again the key metrics for further analysis:

```
median(cni$tcpesm)
```

```
[1] 867.1192
```

```
median(cni$tcpesc)
```

```
[1] 12.77646
```

```
median(cni$tcpbw)
```

```
[1] 9444.988
```

```
median(cni$tcpebw)
```

```
[1] 9679.113
```

3.4 2024

3.4.1 Flannel

Moving on to 2024, we will now read the 2024 data set for Flannel.

```
main <- "."
cni <- read.csv(here(main,"data","2024-flannel.csv"))
```

In the next step, we will examine the structure of our data and visually inspect the measurements.

```
str(cni)
```

```
'data.frame':   3104 obs. of  24 variables:
 $ smem  : num  1204 1210 1209 1208 1209 ...
 $ scpu  : num  0.0149 0.0145 0.0146 0.0154 0.0153 ...
 $ cmem  : num  1187 1199 1201 1222 1219 ...
 $ ccpu  : num  0.0125 0.0121 0.0122 0.0122 0.0121 ...
 $ tcpbw : num  18936 19076 19206 19056 19097 ...
 ...
```

Then, we'll extract the key metrics for further analysis:

```
median(cni$tcpesm)
```

```
[1] 1175.85
```

```
median(cni$tcpesc)
```

```
[1] 5.052353
```

```
median(cni$tcpbw)
```

```
[1] 19166.04
```

```
median(cni$tcpebw)
```

```
[1] 19942.3
```

We can see a significant increase in bandwidth compared to previous years. In the years between the 2021 and 2024 measurements, we saw significant improvements in Kubernetes and Linux kernel development, and the 2024 infrastructure setup for the measurements had 40GBit interface cards and switches.

3.4.2 Calico

We will now read the 2024 data set for Calico.

```
main <- "."
cni <- read.csv(here(main,"data","2024-calico.csv"))
```

In the next step, we will examine the structure of our data and visually inspect the measurements.

```
str(cni)
```

```
'data.frame':   3098 obs. of  24 variables:
 $ smem  : num  1435 1413 1414 1402 1419 ...
 $ scpu  : num  0.0213 0.02 0.0212 0.0213 0.0206 ...
 $ cmem  : num  1377 1370 1361 1369 1364 ...
 $ ccpu  : num  0.0259 0.025 0.0243 0.0265 0.0252 ...
 $ tcpbw : num  18520 18603 18600 18549 18524 ...
 $ tcpsm : num  1388 1390 1397 1394 1399 ...
 ...
```

Then, we'll extract the key metrics for further analysis:

```
median(cni$tcpesm)
```

```
[1] 1420.843
```

```
median(cni$tcpesc)
```

```
[1] 5.446998
```

```
median(cni$tcpbw)
```

```
[1] 18571.56
```

```
median(cni$tcpebw)
```

```
[1] 19263.95
```

3.4.3 Canal

We will now read the 2024 data set for Canal.

```
main <- "."
cni <- read.csv(here(main,"data","2024-canal.csv"))
```

In the next step, we will examine the structure of our data and visually inspect the measurements.

```
str(cni)
```

```
'data.frame':   3088 obs. of  24 variables:
 $ smem  : num  1243 1269 1269 1269 1245 ...
 $ scpu  : num  0.0281 0.0281 0.0281 0.0281 0.0281 ...
 $ cmem  : num  1235 1229 1228 1238 1238 ...
 $ ccpu  : num  0.0233 0.0251 0.0228 0.0284 0.0276 ...
 $ tcpbw : num  16841 16843 16840 16845 16847 ...
 $ tcpsm : num  1260 1279 1257 1242 1233 ...
 ...
```

Then, we'll extract the key metrics for further analysis:

```
median(cni$tcpesm)
```

```
[1] 1285.42
```

```
median(cni$tcpesc)
```

```
[1] 5.647729
```

```
median(cni$tcpbw)
```

```
[1] 16842.78
```

```r
median(cni$tcpebw)
```

```
[1] 16328.69
```

3.4.4 Cilium

We will now read the final 2024 data set, Cilium.

```r
main <- "."
cni <- read.csv(here(main,"data","2024-cilium.csv"))
```

In the next step, we will examine the structure of our data and visually inspect the measurements.

```r
str(cni)
```

```
'data.frame':   3101 obs. of  24 variables:
 $ smem  : num  1612 1613 1607 1605 1604 ...
 $ scpu  : num  0.0111 0.0109 0.0111 0.0126 0.0121 ...
 $ cmem  : num  1596 1592 1603 1597 1594 ...
 $ ccpu  : num  0.0118 0.0122 0.0125 0.0123 0.0122 ...
 $ tcpbw : num  20929 20932 20761 20525 20675 ...
 $ tcpsm : num  1538 1538 1538 1538 1534 ...
 ...
```

Then, we'll extract one last time the key metrics for further analysis:

```r
median(cni$tcpesm)
```

```
[1] 1521.506
```

```r
median(cni$tcpesc)
```

```
[1] 6.152466
```

```r
median(cni$tcpbw)
```

```
[1] 20709.53
```

```r
median(cni$tcpebw)
```

```
[1] 21758.27
```

We have now processed all datasets and can proceed with the analysis.

4 Data Analysis

4.1 Performance Considerations

In the previous chapters, we've already identified our key performance metrics: CPU consumption, memory usage, and TCP bandwidth.

In the first step of the analysis, we will tabulate the performance metrics and, from the results, define a ranking for each CNI within each metric and each year.

As a final step of the analysis, we will weigh the rankings to arrive at an overall ranking for the CNI performance and guidance on which CNI to use.

4.2 Server CPU Usage

The first metric that we want to look at is server CPU usage.

Table 4: Median Server CPU Usage

Year	Flannel	Calico	Canal	Cilium
2020	5.05	6.44	6.69	13.22
2021	5.01	6.36	6.66	12.77
2024	5.05	5.44	5.64	6.15

We get a reasonably uniform distribution across the three measurements. In 2024, we have a much more powerful set of hardware for our servers with more modern CPUs, and we can see that Cilium benefits the most from the increase in CPU speed.

Overall, we get the following ranking of the CNIs:

Table 5: Server CPU Usage Ranking

Year	1st	2nd	3rd	4th
2020	Flannel	Calico	Canal	Cilium
2021	Flannel	Calico	Canal	Cilium
2024	Flannel	Calico	Canal	Cilium

4.3 Server Memory Consumption

The second metric we want to analyze is server memory consumption.

Table 6: Median Server Memory Consumption

Year	Flannel	Calico	Canal	Cilium
2020	590	659	655	866
2021	590	661	659	867
2024	1175	1420	1285	1521

We again get a reasonably uniform distribution, albeit with Canal in 2nd place instead of Calico. The 2024 data shows that all CNIs use the more powerful hardware and consume more memory across the board.

This leads us to the following ranking of the CNIs:

Table 7: Server Memory Consumption Ranking

Year	1st	2nd	3rd	4th
2020	Flannel	Canal	Calico	Cilium
2021	Flannel	Canal	Calico	Cilium
2024	Flannel	Canal	Calico	Cilium

4.4 Pod-to-Pod Bandwidth

The third metric we'll analyze will be the first bandwidth metric, TCP Pod-to-Pod traffic.

Table 8: Median Pod-to-Pod Bandwidth

Year	Flannel	Calico	Canal	Cilium
2020	9705	8882	8634	9475
2021	9695	8876	8612	9444
2024	19166	18571	16842	20709

The bandwidth distribution is slightly more varied, with Cilium gaining the most over the years. The 2024 data shows that all CNIs use the faster NIC speed and the updated kernel, and an overall significant increase in bandwidth can be observed.

We get the following ranking of the CNIs, with Cilium overtaking Flannel and moving to first place:

Table 9: Pod-to-Pod Bandwidth Ranking

Year	1st	2nd	3rd	4th
2020	Flannel	Cilium	Calico	Canal
2021	Flannel	Cilium	Calico	Canal
2024	Cilium	Flannel	Calico	Canal

4.5 Pod-to-Server Bandwidth

The fourth and final metric will be the second bandwidth metric, TCP Pod-to-Server traffic.

Table 10: Median Pod-to-Server Bandwidth

Year	Flannel	Calico	Canal	Cilium
2020	9828	8675	8576	9673
2021	9825	8763	8579	9679
2024	19942	19263	16328	21758

The data again shows that in 2024, all CNIs use the faster NIC speed and show an overall increase in bandwidth.

We get the same ranking for the CNIs, with Cilium overtaking Flannel and again occupying first place in TCP Pod-to-Server communication:

Table 11: Pod-to-Server Bandwidth Ranking

Year	1st	2nd	3rd	4th
2020	Flannel	Cilium	Calico	Canal
2021	Flannel	Cilium	Calico	Canal
2024	Cilium	Flannel	Calico	Canal

4.6 Weighted Ranking

Now that we have the individual performance rankings for our key indicators, we must identify the best overall choice. To do this, we'll use a weighted ranking of the performance indicators to arrive at a final result.

The weighted ranking works by assigning a rank or score to individual items based on multiple criteria, where each criterion is given a specific weight or importance level.[19]

To judge overall network performance, we'll assign the following weights to the rankings:

- CPU usage: 20%
- Memory consumption: 30%
- Pod-to-Pod Bandwidth: 25%
- Pod-to-Server Bandwidth: 25%

To calculate the weighted rankings, we'll use the reverse values for the individual rankings of the four CNIs, i.e., 4 for first place and 1 for last place, from the tables above.

We'll use Microsoft Excel to tabulate the results.[20]

4.7 Results

In 2020 and 2021, the results were pretty clear:

Table 12: Table generated by Excel2LaTeX from sheet 2020

CNI	CPU	Memory	P2P	P2E	Avg	Rank
Weight	0.2	0.3	0.25	0.25		
Flannel	4	4	4	4	4	1
Calico	3	2	2	2	2.2	2
Canal	2	3	1	1	1.8	4
Cilium	1	1	3	3	2	3

Table 13: Table generated by Excel2LaTeX from sheet 2021

CNI	CPU	Memory	P2P	P2E	Avg	Rank
Weight	0.2	0.3	0.25	0.25		
Flannel	4	4	4	4	4	1
Calico	3	2	2	2	2.2	2
Canal	2	3	1	1	1.8	4
Cilium	1	1	3	3	2	3

[19]See *Gemini (2024)*: Weighted Ranking. [10]
[20]See *Bobbitt, Z. (2023)*: How to Calculate Weighted Ranking in Excel. [2]

Flannel comes out on top, delivering the highest bandwidth with the lowest CPU and memory usage; Canal comes in second, and Cilium third. It is interesting to note that Canal takes last place, even though it is, in essence, a combination of Calico and Flannel. However, it seems to be weighed down by higher CPU and memory usage.

In 2024, though, the faster hardware and the years of development and improvement of eBPF and Cilium are changing the picture. Cilium shows a lot of bandwidth gains and moves up to second place, past Calico and Canal.

Table 14: Table generated by Excel2LaTeX from sheet 2024

CNI	CPU	Memory	P2P	P2E	Avg	Rank
Weight	0.2	0.3	0.25	0.25		
Flannel	4	4	3	3	3.5	1
Calico	3	2	2	2	2.2	3
Canal	2	3	1	1	1.8	4
Cilium	1	1	4	4	2.5	2

Flannel takes first place in all measurements, making it an excellent choice for using it as the CNI in Kubernetes cluster deployments.

Looking at bandwidth alone, though, Cilium would have taken first place. Also, with all the additional features we haven't looked at, such as a built-in load balancer and the myriad of observability features, it does make a compelling use case for itself. However, it comes with significantly higher CPU usage and memory consumption compared to our overall winner, Flannel.

For real,[21] I see Flannel as the best option for a CNI unless you need to implement network policies, and I, for one, was thrilled when SUSE announced full support for Flannel in RKE2 earlier this year.[22]

From the data analysis, we can now clearly guide you to select either Flannel or Cilium as the CNI for your RKE cluster. Your choice will heavily depend on whether you need the additional features that Cilium offers or whether you can live with Flannel's flat overlay network and enjoy its simplicity.

4.8 Outlook

EBPF is currently the most talked-about feature in the Linux kernel, offering the most exciting prospects for network performance improvements. Cilium was developed on top

[21]See *Kelly, J. (2024)*: Gen-Z Slang Is Revolutionizing Work Jargon. [13]
[22]See *Frank, C. (2024)*: RKE2: Flannel? Flannel! [8]

of eBPF and will significantly benefit from future enhancements to it. It does not take much imagination to see a significant further increase in bandwidth in the future.

On the other hand, Flannel is relatively simple and has a low overall overhead, so all performance improvements in the Linux kernel for IP networking outside of eBPF will also benefit Flannel's performance and bandwidth. We can also most likely look forward to an increase in bandwidth in the future.

Both CNIs, Flannel, and Cilium, will likely remain the top choices to select as CNI for Kubernetes clusters for the next few years.

5 Summary

The data analysis showed marked improvements in Kubernetes networking performance over the last couple of years. We saw performance improvements across the board for all four CNIs included with SUSE's Rancher Kubernetes Engine.

The power is in the data—we were able to identify Flannel and Cilium as the two CNIs that will deliver the best performance and recommend them for future cluster configurations.

Cilium is the most promising new development in Kubernetes networking, and the venerable Flannel is holding up well, delivering similar performance with a smaller footprint and a smaller feature set.

We were merely able to scratch the surface with this secondary analysis, but I do hope that you will find at least some valuable insights and pointers to start with. A big shoutout to Alexis Ducastel of infraBuilder; none of this would have been possible without their excellent work.

To quote Toshinori Yagi: "Next, it's your turn."[23] - go and create your own cluster!

Happy Ranching!

[23] *Crunchyroll (2024)*: 40 My Hero Academia Quotes Worth Remembering. [4]

References

[1] APA. (2021) Definitions related to sexual orientation. [Access 2021-04-06]. [Online]. Available: https://www.apa.org/pi/lgbt/resources/sexuality-definitions.pdf

[2] Z. Bobbitt. (2023) How to calculate weighted ranking in excel. [Access 2024-08-07]. [Online]. Available: https://www.statology.org/excel-weighted-ranking/

[3] CNCF. (2024) Network plugins. [Access 2024-07-31]. [Online]. Available: https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/network-plugins/

[4] Crunchyroll. (2024) 40 my hero academia quotes worth remembering. [Access 2024-08-06]. [Online]. Available: https://www.crunchyroll.com/news/features/2024/8/4/my-hero-academia-quotes

[5] A. Dsouza and K. Martin. (2024) Kubernetes v1.30. [Access 2024-04-18]. [Online]. Available: https://kubernetes.io/blog/2024/04/17/kubernetes-v1-30-release/

[6] A. Ducastel. (2024) Benchmark results of kubernetes network plugins. [Access 2024-07-30]. [Online]. Available: https://itnext.io/benchmark-results-of-kubernetes-network-plugins-cni-over-40gbit-s-network-2024-156f085a5e4

[7] C. Frank. (2020) Behind the scenes of flannel. [Access 2024-07-30]. [Online]. Available: https://medium.com/@chfrank_cgn/behind-the-scenes-of-flannel-307aef347f20

[8] C. Frank. (2024) Rke2: Flannel? flannel! [Access 2024-08-08]. [Online]. Available: https://medium.com/@chfrank_cgn/rke2-flannel-flannel-6564dd1ae49e

[9] Gemini. (2024) Exploratory data analysis. [Access 2024-08-05]. [Online]. Available: https://gemini.google.com

[10] Gemini. (2024) Weighted ranking. [Access 2024-08-07]. [Online]. Available: https://gemini.google.com

[11] Gemini. (2024) What is kubernetes. [Access 2024-04-18]. [Online]. Available: https://gemini.google.com

[12] W. Hillier. (2021) A guide to secondary data analysis. [Access 2024-08-04]. [Online]. Available: https://careerfoundry.com/en/blog/data-analytics/secondary-data-analysis/

[13] J. Kelly. (2024) Gen-z slang is revolutionizing work jargon. [Access 2024-08-04]. [Online]. Available: https://www.forbes.com/sites/jackkelly/2024/07/31/gen-z-slang-at-work/

[14] S. E. Kim, "Heads or tails?" *Scientific American Magazine*, vol. 330, no. 1, p. 12, Jan. 2024.

[15] A. Orn. (2023) Means and medians: When to use which. [Access 2024-08-05]. [Online]. Available: https://research-collective.com/means-and-medians-when-to-use-which/

[16] L. Qian and B. Carpenter, "A flow-based performance analysis of tcp and tcp applications," in *A flow-based performance analysis of TCP and TCP applications*, 12 2012, pp. 41–45.

[17] S. Rahmstorf, *Climate and Weather at 3 Degrees More*. Cham: Springer Nature Switzerland, 2024, pp. 3–17.

[18] A. Saguy and J. Williams. (2020) Why we should all use they/them pronouns. [Access 2020-05-20]. [Online]. Available: https://blogs.scientificamerican.com/voices/why-we-should-all-use-they-them-pronouns/

[19] Solarwinds. (2024) What are network performance metrics? [Access 2024-08-04]. [Online]. Available: https://www.solarwinds.com/resources/it-glossary/network-metrics

[20] SUSE. (2024) Network options. [Access 2024-07-31]. [Online]. Available: https://docs.rke2.io/networking/basic_network_options

[21] J. W. Tukey, *Exploratory Data Analysis*. Reading, Mass.: Addison-Wesley Pub. Co., 1977.