1. SecInt: Results

In this section, the results from the investigation are presented, including the outcomes of the Proof-of-Concept (PoC), interviews with developers behind Multus-service and results from functional and performance evaluations with Meridio.

1.1 Results from the Exploration of Multus-service

The results of investigation with Multus-service will be presented in this section.

1.1.1 Interview with D through Slack

During the interview, D provided critical insights into the deprecation of Multus-service highlighting several factors contributing to this decision described in Table 1:

Subject	Description
Popularity and API Concerns	D noted that the services API in Kubernetes has been unpopular among some core developers, particularly within the SIG-Network group. Efforts were made to modify Multus-service to accommodate more use cases yet resistance from maintainers due to reluctance to alter the established services API persisted.
Project Viability and Future Direction	The conversation revealed that maintainers did not see the explored use cases as valid or compelling enough to continue supporting the prototype. Instead, there is a shift towards developing solutions that leverage Gateway API, aiming to fulfill similar network functionalities without altering core Kubernetes APIs.

Table 1: Summary of D's insights regarding deprecation of Multus-service.

1.1.2 Interview with T through Slack

T provided insights into the reasons behind the project's deactivation and the current status of Kubernetes networking solutions. T highlighted several critical points during our conversation described in Table 2:

Subject	Description
Deprecation of Multus-Service	T confirmed that the Multus-service was considered a prototype and not supported for production use mainly because it did not proceed to full development due to lack of community feedback and clear use-case definitions.
Technical Limitations and Community Feedback	The service API in Kubernetes, as T noted, is unpopular among core developers within the SIG-Network group, which influenced the decision to not pursue further development of the Multus-service.
Future of Kubernetes Services on Secondary Networks	According to T, while the current focus has shifted towards using the Gateway API to handle similar functionalities, the idea of supporting services on secondary networks through Multus-like tools remains underexplored.

Table 2: Summary of T's insights regarding deprecation of Multus-service.

1.1.3 Panel interview with T during Kubernetes Network Plumbing Work Group Meeting

The panel interview with T during Kubernetes Network Plumbing Work Group Meeting revealed several significant insights into the decision to archive the Multus-service repository:

- **Insufficient Use-Cases and Feedback**: It was noted that the repository suffered from a lack of adequate use-cases and feedback, particularly from vendors and users, which are essential for driving development tailored to actual market and technological needs. This gap made it challenging to discuss and justify the continuation of the service for secondary networks.
- **Shift Toward Gateway API**: The Kubernetes community is moving toward adopting the Gateway API over the service API used by Multus, as advocated by Tim Hockin at KubeCon 2023 NA. This shift is intended to better align with the current and future architecture of Kubernetes.
- **Ongoing Community Discussions**: The conversation highlighted that although the repository was archived, the discussion about secondary network services is not ending. Instead, it is transitioning to the Multi-network Working Group, which is expected to address the design and API requirements for such services comprehensively.

1.1.4 Self-healing for Service on Secondary Network

During the performance test, one instance of the "example-target-application-a" pod was intentionally terminated while the service was under load. Kubernetes promptly re-deployed the pod as part of its self-healing mechanism. Following the re-deployment, Meridio successfully identified the new pod and updated the endpointslices to include it. Observations from the Locust console indicated that there was no significant increase in response time. The newly deployed pod quickly began receiving traffic, and its functionality was verified by the active TCP traffic observed through "tcpdump." This indicates that the Meridio service proxy can effectively reconcile changes in the application environment.

1.2 Performance implication of service on secondary network

In this section, the result from performance tests will be presented.

1.2.1 Latency test

During the latency test for *service-i*, I observed the latency (in milliseconds) plotted against the number of TCP connections. The graph 1 illustrates the results from the latency test for *service-i* and highlights key statistics: a maximum latency of 2083 ms, a minimum latency of 21 ms, a latency range of 2062 ms, an average latency of 52.96 ms, and a jitter of 50.44 ms.

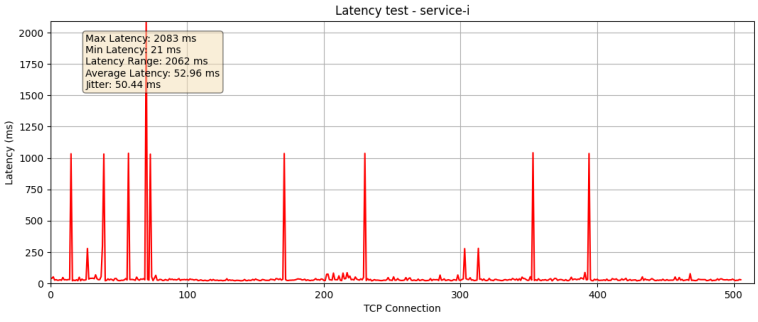

Figure 1: Graph for results from latency test on *service-i*.

During the latency test for *service-primary*, I observed the latency (in milliseconds) plotted against the number of TCP connections. The graph 2 illustrates the results from the latency test for *service-primary* and highlights key statistics: a maximum latency of 2055 ms, a minimum latency of 22 ms, a latency range of 2033 ms, an average latency of 50.19 ms, and a jitter of 43.19 ms. Similar to *service-i*, there were occasional spikes in latency, but the majority of the connections maintained relatively low and consistent latency values.

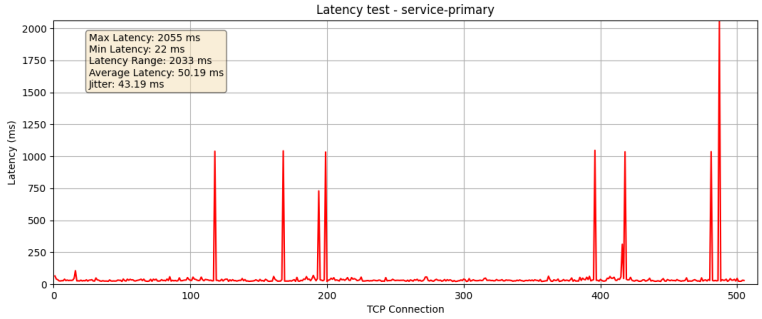

Figure 2: Graph for results from latency test on *service-primary*.

1.2.2 Throughput test

Table 3 shows the performance tests conducted on the *service-i* connection over a period of approximately 60 seconds. These tests revealed varying throughput rates and connect times across five distinct trials. The total data transfer ranged from 101 MBytes to 138 MBytes, showing variability in network conditions or server performance. Bandwidth similarly fluctuated, with the lowest average of 13.6 Mbits/sec observed in the first test and the highest average of 18.6 Mbits/sec in the fourth test.

Test	Total Transfer (MB)	Avg BW (Mbits/sec)	Min Conn Time (ms)	Max Conn Time (ms)	Standard Deviation Conn Time (ms)
1	101	13.6	15.624	15.624	0.000
2	130	17.5	6.137	6.137	0.000
3	110	14.9	100.167	100.167	0.000
4	138	18.6	15.539	15.539	0.000
5	102	13.7	15.612	1087.796	452.072

Table 3: Throughput test results for *service-i*.

Notably, connect times exhibited significant differences; particularly in the fifth test, the maximum connect time spiked dramatically compared to other tests, accompanied by a high standard deviation indicating substantial variability in connection establishment during this trial.

Test	Total Transfer (MB)	Avg BW (Mbits/sec)	Min Conn Time (ms)	Max Conn Time (ms)	Standard Deviation Conn Time (ms)
1	114	15.3	15.799	28.923	5.326
2	109	14.7	31.395	31.395	0.000
3	128	17.3	15.718	31.252	8.187
4	130	17.6	31.253	31.253	0.000
5	134	18.2	31.381	31.381	0.000

Table 4: Throughput test results for *service-primary*.

Table 4 shows the performance tests conducted on the *service-primary* connection over a period of 60 seconds. These tests revealed varying throughput rates and connect times across five distinct trials. The total data transfer ranged from 109 MBytes to 134 MBytes, indicating some variability in the connection stability or network conditions. The bandwidth similarly varied, with the lowest average of 14.7 Mbits/sec observed in the second test and the highest average of 18.2 Mbits/sec in the fifth test.

Connect times showed minimal variation, except in the second and third tests where the maximum connect time was notably higher than other tests.

The total average bandwidth is 16.62 Mbps for *service-primary* and 15.66 Mbps for *service-i*. The following graph (Figure 3) illustrates the average bandwidth for *service-i* (Red) and *service-primary* (Blue) during the throughput test.

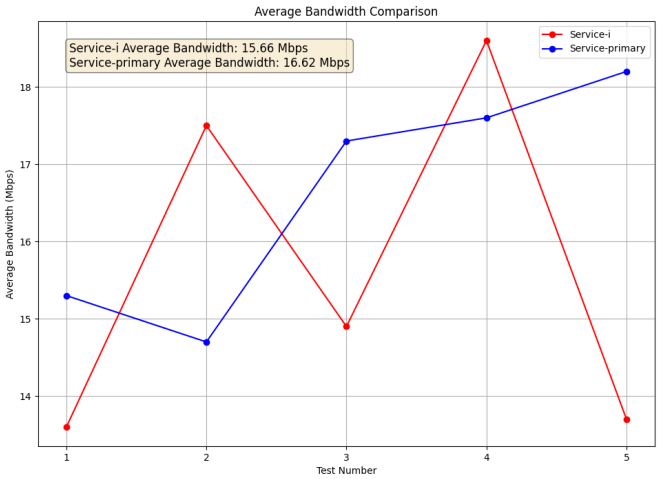

Figure 3: Average bandwidth comparison between *service-i* (Red) and *service-primary* (Blue).

1.3 Load test

Figure 4 displays three primary performance metrics for *service-i* during a load test spanning from 17:41:00 to 17:42:53. The top panel tracks Total Requests per Second (RPS) in green, which initially peaks at approximately 2400 around 17:42:05, dips to around 1700 shortly after, and then climbs back to a steady peak near 2500 for the remainder of the test. The red line representing failures per second shows no data and is not relevant for this test. In the middle panel, Response Times are illustrated with the Average Response Time in yellow, consistently low and stable, and the 95th percentile in purple, which exhibits slight fluctuations before ending in a sharp spike. The bottom panel follows the Number of Users in blue, which progressively increases to reach 8000 by the test's conclusion. Each panel shares a synchronized time axis from 17:41:00 to 17:42:53 and includes a grid to facilitate easy comparison of data points across the metrics.

Figure 5 displays three primary performance metrics for *service-primary* during a load test spanning from 17:37:25 to 17:39:13. The top panel tracks Total Requests per Second (RPS) in green, which shows a continuous increase and then stabilizes towards the end of the test period. The red line representing failures per second remains at zero throughout the test, indicating no failures. In the middle panel, Response Times are illustrated with the Average Response Time in yellow, consistently low and stable, and the 95th percentile in purple, which exhibits minor fluctuations before ending in a significant spike. The bottom panel follows the Number of Users in blue, which progressively increases to reach approximately 8000 by the test's conclusion. Each panel shares a synchronized time axis from 17:37:25 to 17:39:13 and includes a grid to facilitate easy comparison of data points across the metrics.

1.4 Footprint

This section presents the results from the footprint test for *service-i* and *service-primary*.

1.4.1 Load on the services

Figure 6 illustrates the load being generated to *service-i*. The peak for total connections was 19439 users. The test started from 17:55:38 and ended at 17:58:16.

Figure 7 illustrates the load being generated to *service-primary*. The peak for total connections was 20000 users. The test started at 17:43:34 and ended at 17:46:00.

Figure 4: Load test results from Locust console for *service-i*.

1.4.2 CPU usage

Figure 8 shows the CPU usage for *service-i* and *service-primary*. Different colors of lines represent different pods (see Figure 9, 10). The monitored pods are described in section 7.4.4, which are the same pods for the entire footprint test. There are two spikes shown in the graph: the first one corresponds to the time when load was generated for *service-primary* from 17:43:34 to 17:46:00, and the second spike corresponds to the time when load was generated for *service-i* from 17:55:38 to 17:58:16.

The peak of CPU usage for *service-i* was at 17:57:55, as illustrated in Figure 9 which shows the CPU usage for a list of pods during the peak, sorted by CPU usage.

The peak of CPU usage for *service-primary* was at 17:45:49, as illustrated in Figure 10 which shows the CPU usage for a list of pods during the peak, sorted by CPU usage.

1.4.3 RAM usage

Figure 11 shows the RAM usage for *service-i* and *service-primary*. Different colors of lines represent different pods (see Figure 13 12).

The peak RAM usage during the test for *service-i* was at 17:58:17. The list of RAM usage for monitored pods during 17:58:17, sorted by RAM usage, is shown in Figure 12.

The peak RAM usage during the test for *service-primary* was at 17:45:52. The list of RAM usage for monitored pods during 17:45:52, sorted by RAM usage, is shown in Figure 13.

1.4.4 Rate of transmitted packets

Figure 14 shows the rate of transmitted packets during the footprint tests for *service-i* and *service-primary*. The rate of transmitted packets is measured in kilo packets per second (kpps). The two spikes represent the increased amount of transmitted packets during the footprint test. Different colors of lines represent different pods (see Figure 15, 16).

The first peak around 17:45:55 occurred when *service-primary* was being loaded. The list of rates of transmitted packets of monitored pods can be seen in Figure 15, sorted from highest kpps to lower.

Figure 5: Load test results from Locust console for *service-primary*.

The second peak around 17:57:35 occurred when *service-i* was being loaded. The list of rates of transmitted packets of monitored pods can be seen in Figure 16, sorted from highest kpps to lower.

1.4.5 Rate of received packets

Figure 17 shows the rate of received packets during the footprint tests for *service-i* and *service-primary*. The rate of received packets is measured in kilo packets per second (kpps). The two spikes represent the increased amount of received packets during the footprint test. Different colors of lines represent different pods (see Figure 19, 18).

The first peak around 17:45:32 occurred when *service-primary* was being loaded. The list of rates of received packets of monitored pods can be seen in Figure 18, sorted from highest kpps to lower.

The second peak around 17:57:45 occurred when *service-i* was being loaded. The list of rates of received packets of monitored pods can be seen in Figure 19, sorted from highest kpps to lower.

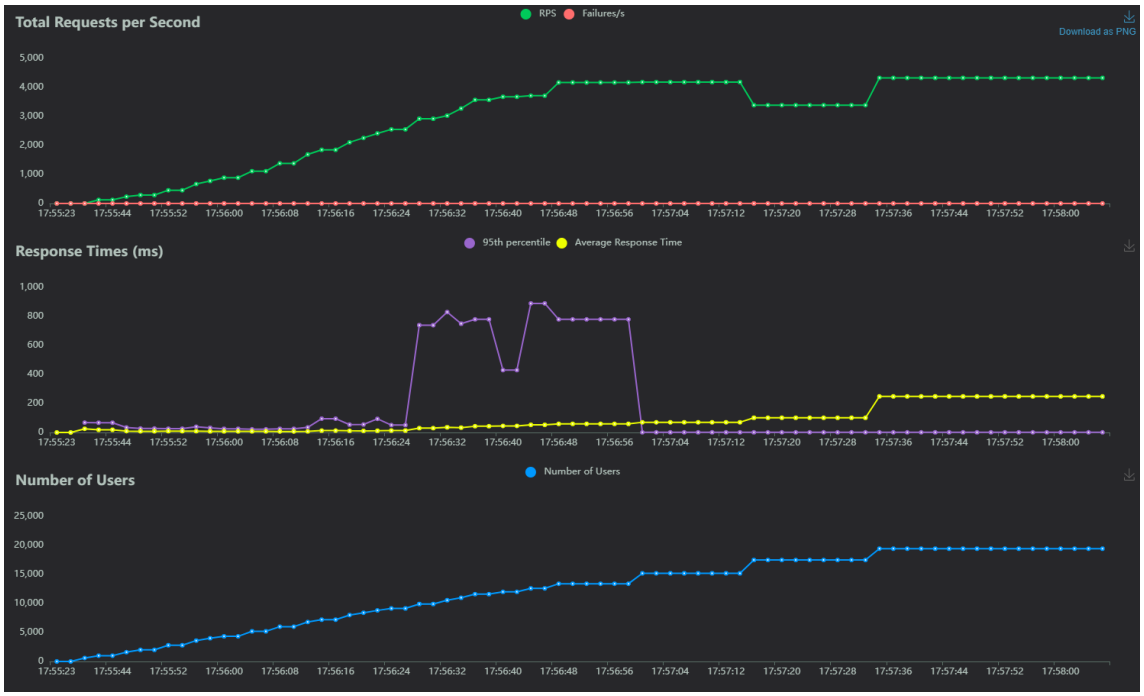

Figure 6: Load on *service-i* during the footprint test.

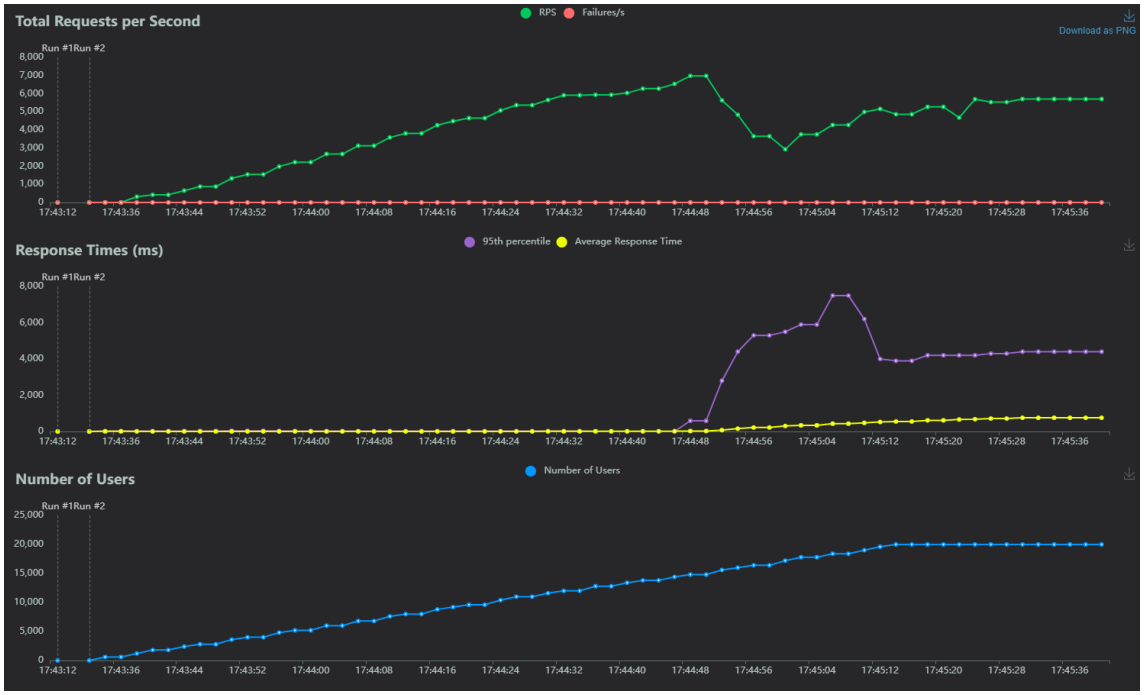

Figure 7: Load on *service-primary* during the footprint test.

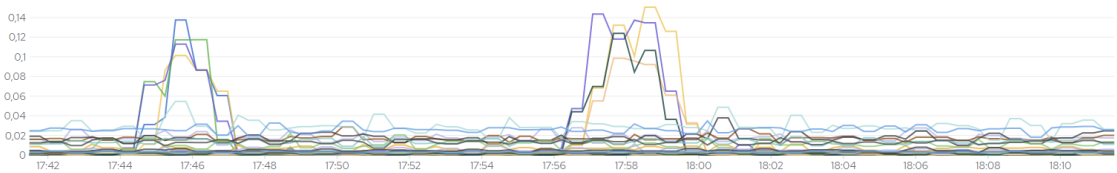

Figure 8: CPU usage for *service-i* (right spike) and *service-primary* (left spike).

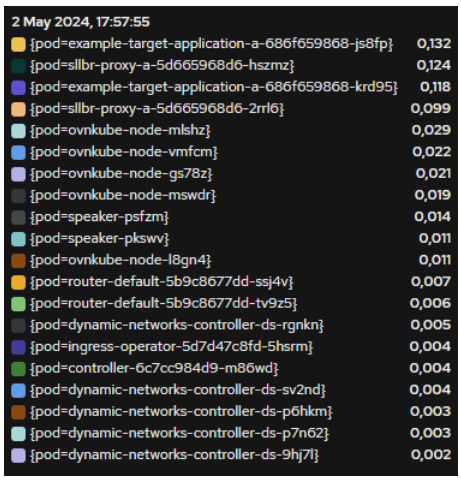

Figure 9: CPU usage of monitored pods at 17:57:55 for *service-i*.

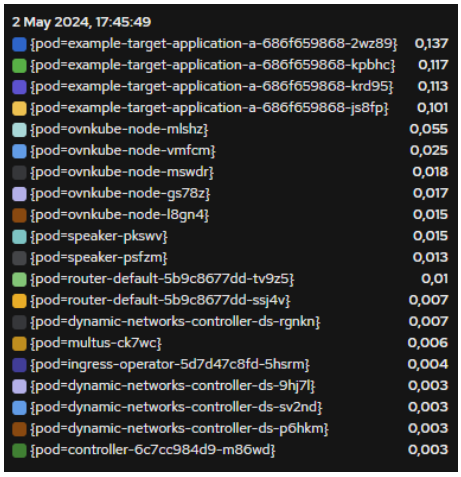

Figure 10: CPU usage of monitored pods at 17:45:49 for *service-primary*.

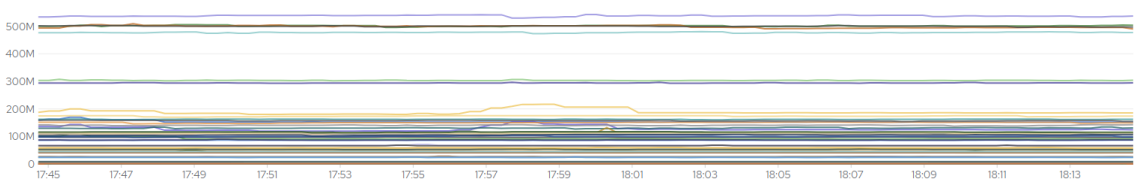

Figure 11: RAM usage during the footprint test for *service-i* (17:55:38-17:58:16) and *service-primary* (17:43:34-17:46:00).

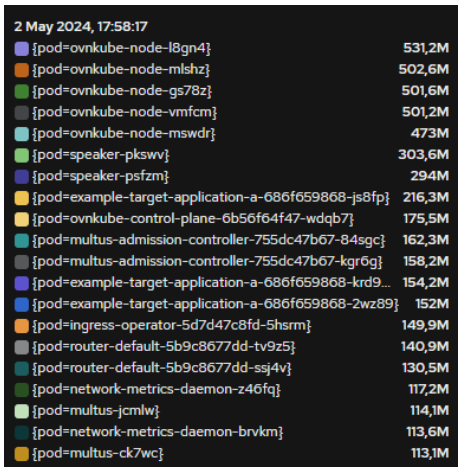

Figure 12: RAM usage of monitored pods at 17:58:17 for *service-i*.

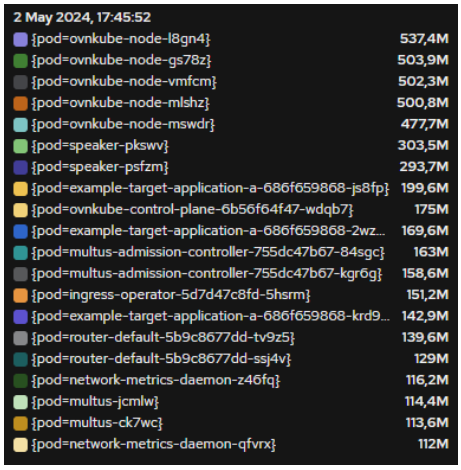

Figure 13: RAM usage of monitored pods at 17:45:52 for *service-primary*.

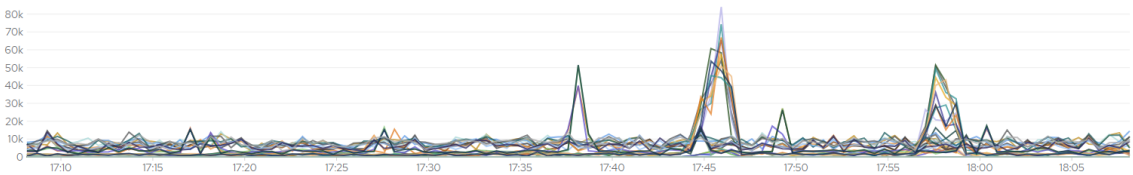

Figure 14: Rate of transmitted packets during footprint tests for *service-i* and *service-primary*.

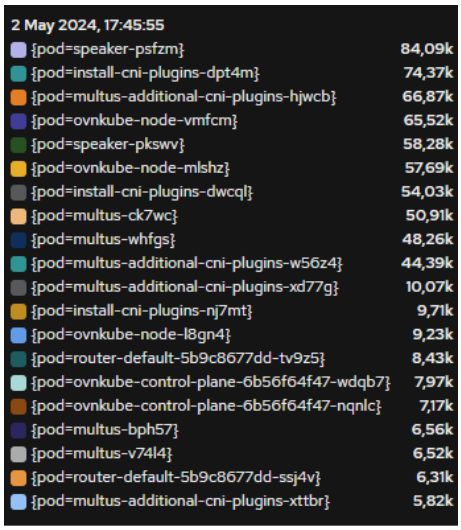

Figure 15: Rate of transmitted packets for monitored pods during 17:45:55 for *service-primary*.

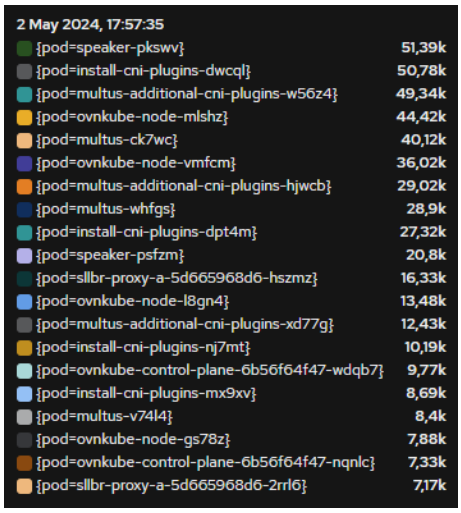

Figure 16: Rate of transmitted packets for monitored pods during 17:57:35 for *service-i*.

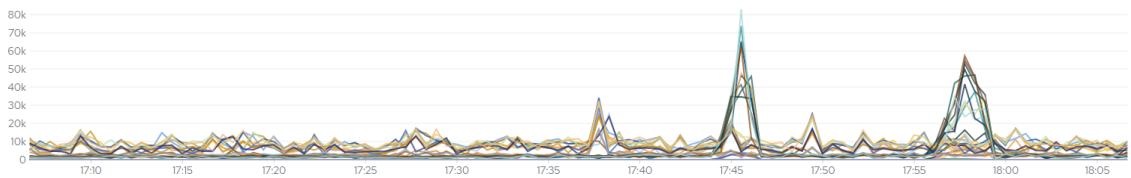

Figure 17: Rate of received packets during footprint tests for *service-i* and *service-primary*.

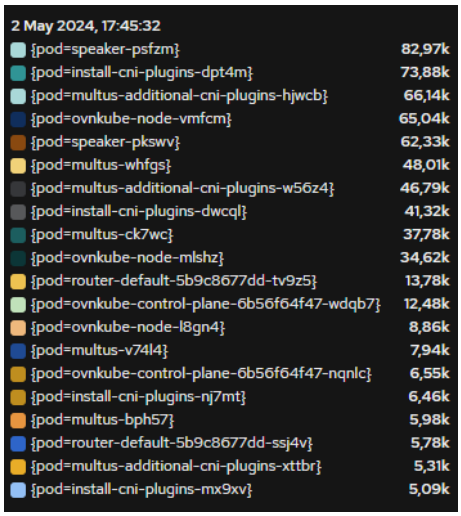

Figure 18: Rate of received packets for monitored pods during 17:45:32 for *service-primary*.

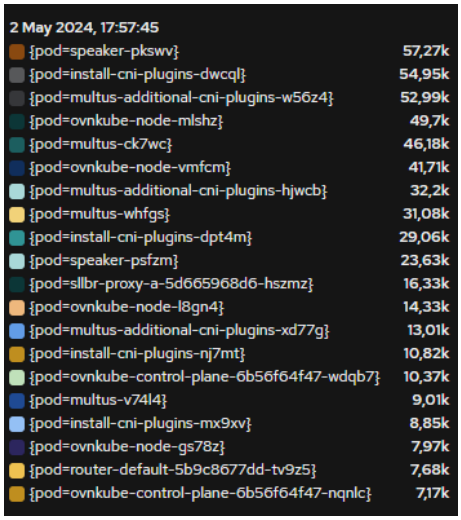

Figure 19: Rate of received packets for monitored pods during 17:57:45 for *service-i*.