Use Case 6: Kubernetes (K8s)

Return to User Experiences

Details

Author	Zhuangwei Kang
Title	DDS in Multi-platform Kubernetes
Organization	Vanderbilt University
Date	June 11 2021
Time	32 minutes
Presentation	https://www.brighttalk.com/webcast/12231/493865?utm_campaign=channel-feed&utm_source=brighttalk-portal&utm_medium=web
Document	http://public2.brighttalk.com/resource/core/345531/ddsf-student-presentation_756205.pptx

Abstract

Return to Top

The OMG Data Distribution Service (DDS) is an open, Real-time Publish-Subscribe (RTPS) Middleware standard, which has been widely adopted in many latency-sensitive and mission-critical Industrial Internet of Things (IIoT) applications. However, deploying and managing large-scale distributed DDS applications is tedious and laborious. As a successful Container orchestration platform for distributed applications in the cloud, Kubernetes (K8s) is a promising solution for DDS-based systems. However, the feasibility of running DDS applications in a k8s environment, and the overhead of different k8s virtualization network architectures on DDS application performance has not been systematically studied. To address this, we designed an automated benchmark framework, analyzed potential bottlenecks of several popular k8s virtual network plugins, and conducted a comprehensive set of experiments to demonstrate DDS performance variation under different QoS and k8s network configurations.

Overview

This topic is a comprehensive performance evaluation of different Kubernetes CNI plugins for Edge-Based and Containerized Publish/Subscribe applications. First, we have preliminary of Kubernetes and its Networking Model. Then I will use a real-world example to illustrate the feasibility & challenges of deploying containerized DDS application in a Kubernetes Cluster. We then did systematic experiments to learn the DDS performance in such kinds of deployments. Based on the observations we can get some conclusions and insights.

Quick Kubernetes Overview

To fully understand the Kubernetes-based service it's always good practice to have a brief overview of Kubernetes architecture. A Kubernetes Cluster is composed of a Control Plane/Master Node that leads to at least one worker node. A Node of a Kubernetes Cluster could be a virtual machine or physical machine. A Container is nothing but a process that isolates SIG group and namespace from other processes running on the host. However, Kubernetes builds another layer of encapsulation on top of

containers called a Pod, or a group of 1 or more containers. From the perspective of the networking namespace, each node in the Kubernetes Cluster has an individual seeder and any Pod running on the node has a globally unique IP. All containers in the Pod share the same network namespace.

Motivation for the Project

Benefits of Deploying DDS Applications In a Kubernetes Cluster

There are several benefits for deploying on Kubernetes such as:

- The container enables lightweight encapsulation of functional modules, such as DDS' publisher/subscriber capabilities
- Offers resource isolation thereby allowing distributed applications and services to be easily deployed and operated across heterogeneous platforms through sharing standardized container images.
- With Kubernetes, it is easy to deploy, update, scale, and self-heal distributed applications.

Kubernetes Networking Model

Since DDS is basically a networking application the Kubernetes network model is very much relevant to DDS' performance when running in containers. Kubernetes network model supports pluggable container interfaces, but these plugins have to satisfy the following requirements:

- All containers must communicate with all other containers without Network Address Translation(NAT)
- All nodes must communicate with all containers (and vice-versa) without NAT
- The IP that a container sees itself is the same IP that others see.

There are four types of Networking Schemas:

1. Container-to-Container Networking
2. Pod-to-Pod Networking
3. Pod-to-Service Networking
4. Internet-to-Service Networking

But we will only focus on Pod-to-Pod & Pod-to-Service Networking

Kubernetes Networking Approaches

In this project, we studied three virtual network plugins including Flannel, WeaveNet, & Kube-Router. The reason we selected them they are mainstream CNIs that support common multi-platform Edge-to-Cloud deployment.

- Host-Mode networking approach to have a baseline for understanding the overhead of virtual network plugins. When using the host-mode networking approach containers share the same network namespace as the host, so there is no network virtualization so applications can obtain

the best performance.

The problem is there may be many port conflicts or security issues because any container has a chance to manipulate the network stack of other containers running on the same host, or containers running on the same host may need the same port number.

Virtual Network plugins can be classified into two groups; Including Layer 2-(Data Link Layer???Needs Check-9:44) and Layer 3-Network Layer implementations.

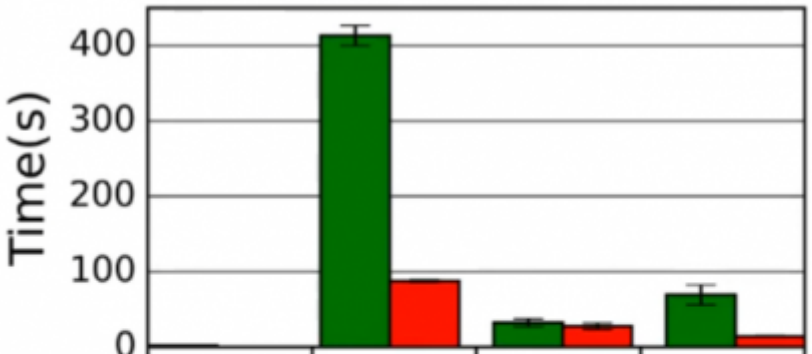
- Flannel supports both of them with VXLAN & Host-GateWay Backends respectively. VXLAN backend, a Layer-2 implementation, Flannel encapsulates packets with VXLAN header in Linux Kernel using VXLAN tunneling technique. Host-GateWay, a Layer-3 implementation, uses peer hosts as gateways, writing routing rules between source and target pods directly into the host's IPtables.
- WeaveNet utilizes VXLAN's tunneling technique as well but it enables this process with an open vSwitch maintained by a WeaveNet Daemon process.
- Kube-Router, another Layer-3 implementation, uses the iBGP to exchange routing and reachability information when performing inter-subnet communication, while BGP is used instead for cross-subnet networking.

Quantative Performance Evaluation

Multi-Platform Kubernetes Deployment

Why does the hybrid architecture deployment?

- Edge-to-Cloud continuum is a common deployment schema for managing large-scale IoT apps.
- Kubernetes Master Node is much more resource-intensive than worker nodes due to extensive cluster-wide operations, such as managing cluster state and processing user requests received from its API Server
- Deploying Kubernetes Master Node on resource-constraint edge devices may lead to un-predictive response times especially in large-scale clusters.



	GetNode	Init	Join	Reset
ARM32	2.4	413.3	32.2	69.4
AMD64	0.2	87.8	27.6	14.2

Fig2. kubectl operation cost comparison

Figure 1: Due to the amount of processing required to managed the cluster state, as well as process user interaction and requests from the API servers. The server with more processing power was able to outperform the more constrained server.

Benchmark Framework

Next we will show our Benchmark Framework to illustrate the feasibility of running DDS container in a Kubernetes Cluster. In this framework there are several components;

- Firstly the Auto Deployer works on setting up the whole cluster and installing the same plugins on the worker and Master node as well as deploy the Perftest container on worker nodes.
- Perftest Pod: a containerized Perftest benchmark application compiled with Connext DDS
- Since some Kubernetes CNIs don't support Multicast so we deployed DDS Cloud discovery service as a Kubernetes deployment to enable service discovery between publisher and subscribers
- The Resource Metrics Monitor is Kubernetes built-in metric server for record resource utilization for node and pod.
- We also used DDS-Perftest Manager that generates performance test commands with a given experiment profile, then executes them on pods and monitors the experiment's progress & record performance metrics.

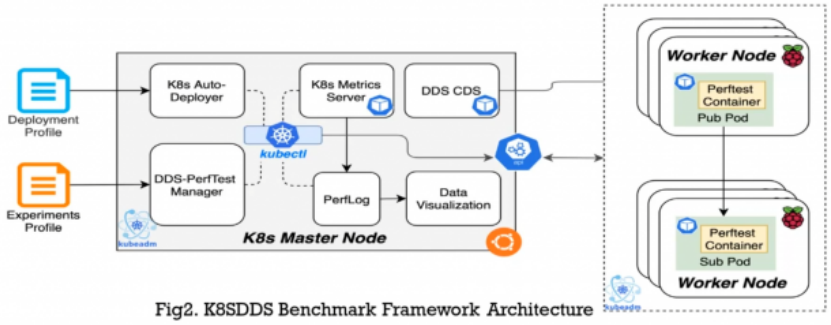
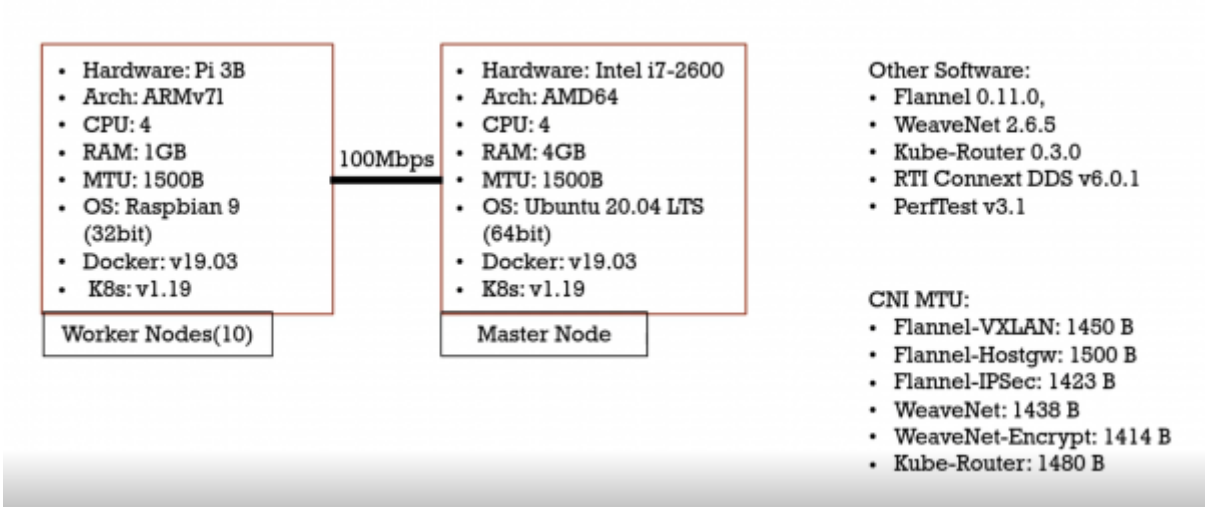


Fig2. K8SDDS Benchmark Framework Architecture

```
nohup kubectl exec -t ./perftest_cpp -pub
    ↪ -latencyTest -dataLen 1024 -
    ↪ executionTime 120 -domain 1 -
    ↪ transport UDPv4 -nic eth0 -cpu -
    ↪ noPrint -- perftest-pub0 > logs/
    ↪ perftest-pub0.log 2>&1 &
nohup kubectl exec -t ./perftest_cpp -sub
    ↪ -latencyTest -dataLen 1024 -
    ↪ executionTime 120 -domain 1 -
    ↪ transport UDPv4 -nic eth0 -cpu -
    ↪ noPrint -- perftest-sub0 > logs/
    ↪ perftest-sub0.log 2>&1 &
```

Experiment Environment



Measurement Approach

1. Throughput
   - 1-sub Scenario: Publisher sends data samples at a configurable rate, and a subscriber calculates throughput by counting the volume of received bytes and samples
   - N-sub Scenario: Average throughput overall subscribers
2. Latency
   - 1-sub Scenario: The publisher sends a “ping” to the subscriber and enters into a waiting state. The subscriber receives the ping and responds with a “pong” and the Publisher will then determine the one way latency from the measured roundtrip time.
   - N-sub Scenario: $$L_{avg} = \frac{1}{K} \sum_{i=1}^{K} L_{i \,\%\, N}$$

Where:

- $L_{i \,\%\, N}$ is the one-way latency of the $(i \,\%\, N)_{th}$ subscriber.
- K is the total number of samples emitted by a publisher during the test.

Experiment 1 Virtualization Overhead

The purpose of the first experiment is to understand the overhead of container virtualization. We setup the experiment with 1 publisher and 1 sub and the publication range is unlimited for the throughput test. We enabled the reliability QoS but disabled the micro batching QoS Should be noted that BareMetal means we ran the Perftest application on the Baremetal machine, But for host-mode, we deployed the Perftest container but the container used the host network namespace. So in this case we can observe the overhead of container virtualization. In our observations showed that the impact of container virtualization does not impact the real-time properties of DDS applications. From the pie charts, you can see the overhead of either throughput and latency is trivial.

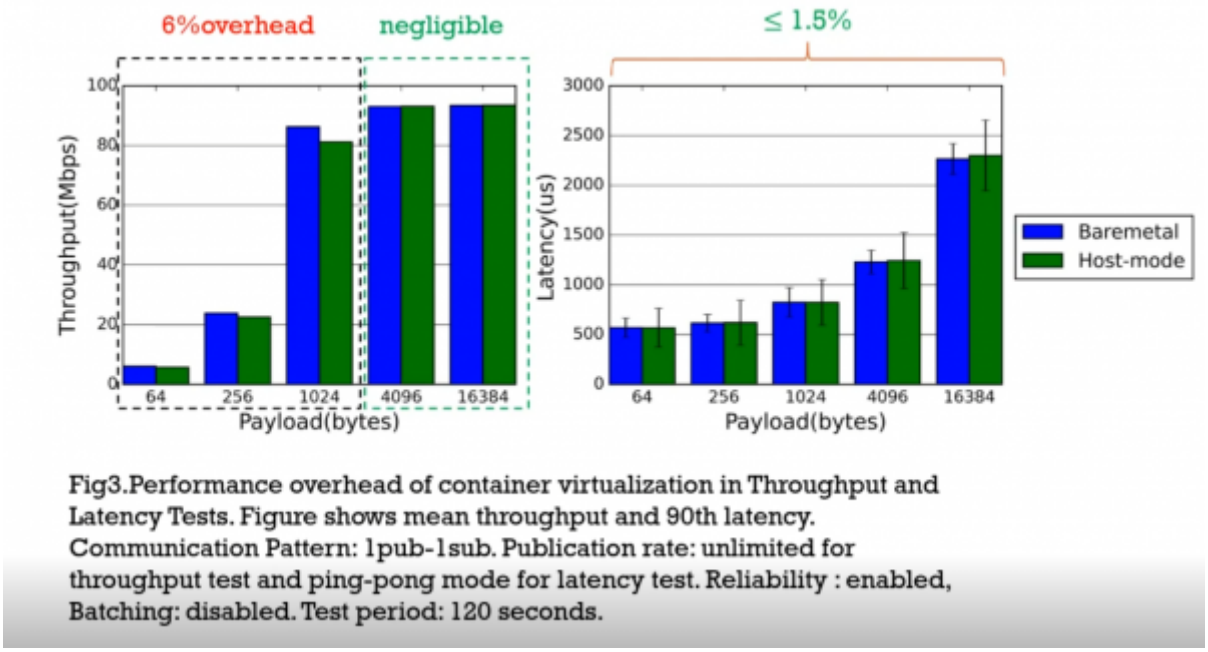


Fig3.Performance overhead of container virtualization in Throughput and Latency Tests. Figure shows mean throughput and 90th latency. Communication Pattern: 1pub-1sub. Publication rate: unlimited for throughput test and ping-pong mode for latency test. Reliability : enabled, Batching: disabled. Test period: 120 seconds.

Experiment 2 CNI Overhead

Our second experiment is to learn the performance overhead illustrated by Virtual network plugins, similarily we did 1pub-1sub experiments with similar QoS Settings.

- Flannel-VXLAN and WeaveNet present notable throughput degradation overall payload lengths due to the overhead of L2 encapsulation.
- WeaveNet's throughput is comparably lower than Flannel-VXLAN, suggesting Open vSwitch VXLAN encapsulation introduces more overhead.
- Flannel-Hostgw and Kube-Router obtain similar throughput because they are both Layer-3
- But Flannel-Hostgw performs better than kube-router when the payload length is grater than 1KB due to the difference in MTU(1500B vs 1480B)

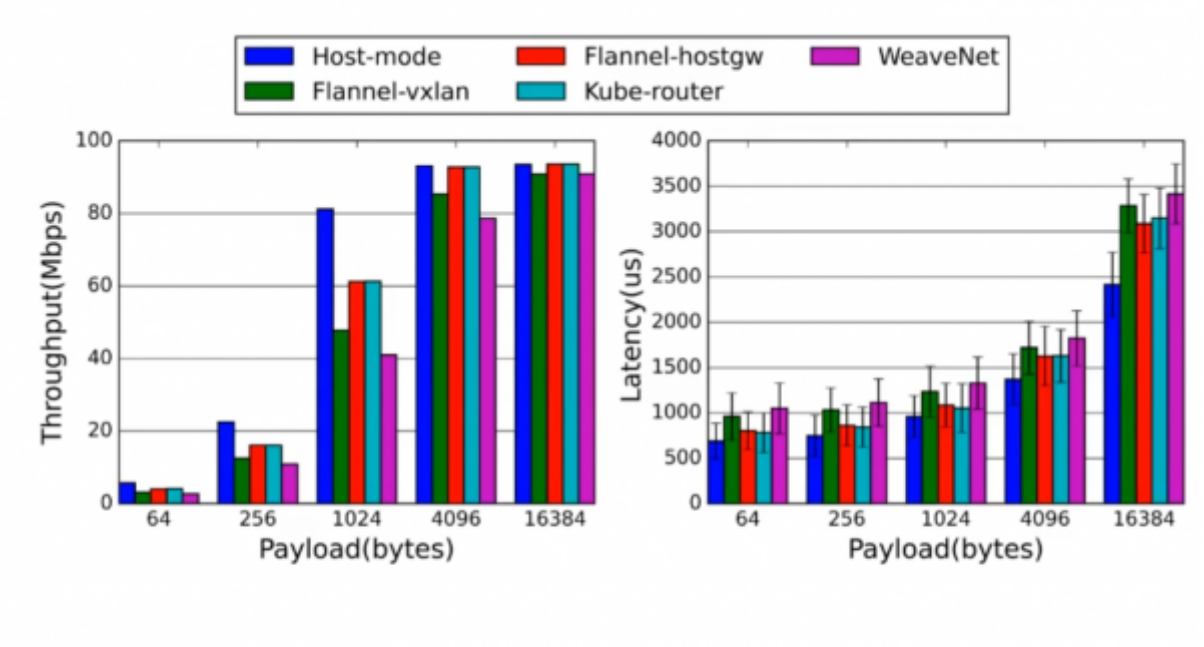


Fig4. Performance comparison of different K8s CNI plugins in Throughput and Latency Tests. Figure shows mean throughput and 90th latency. Communication Pattern: 1pub- 1sub. Publication rate: unlimited for throughput test and ping- pong mode for latency test. Reliability: enabled, Batching: disabled. Test period: 120 seconds.

$\Delta T(\%)$ Payload(B) / CNI	64	256	1024	4096	16384
Flannel-VXLAN	45.6	44.4	41.3	8.4	2.9
Flannel-Hostgw	29.8	28.4	24.6	0.3	-0.1
Kube-Router	28.1	28.4	24.5	0.3	-0.1
WeaveNet	52.6	51.6	49.5	15.5	2.8

$\Delta L(\%)$ Payload(B) / CNI	64	256	1024	4096	16384
Flannel-VXLAN	39.3	37.8	28.7	25.6	35.9
Flannel-Hostgw	16.5	15.2	13.1	18.8	27.7
Kube-Router	13.5	12.6	9.8	19.1	30.3
WeaveNet	52.2	48.7	38.6	33.1	41.4

Table1. overhead of each CNI relative to Host-mode in percentage.

Experiment 3 Performance With BestEffort QoS

If the strict reliability is not required by the DDS application, maybe you already enable BestEffort QoS to obtain lower latency and a tolerable number of packets drop.

- After the experiment we notice that Layer-2 CNIs(Flannel-VXLAN, WeaveNet) have higher latency than Layer-3 ones(Kube-Router, Flannel-Hostgw)
- The latency slightly reduced in the BestEffort mode when running the latency test.

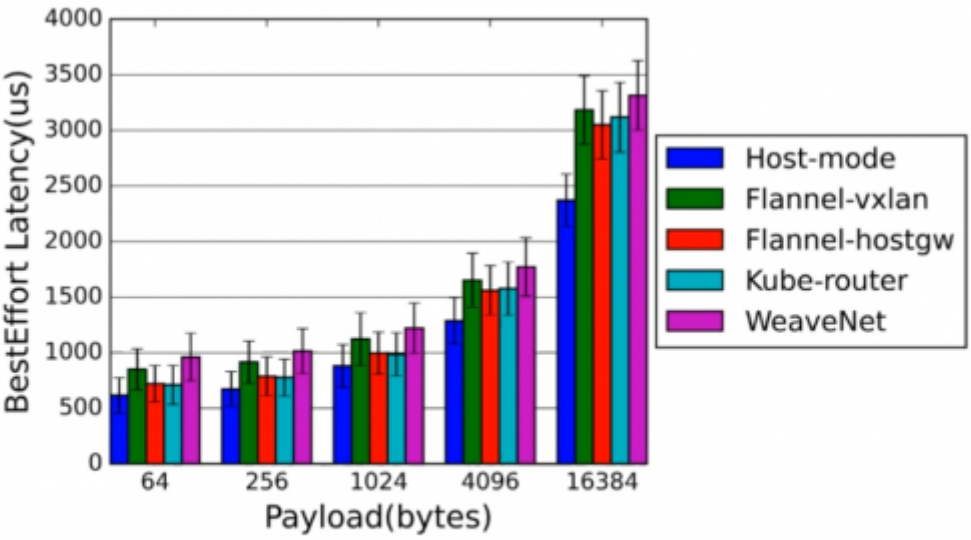


Fig5. 90th Latency with BestEffort QoS. Communication Pattern: 1pub-1sub. Transport: UDP. Batching: disabled. Test period: 120 seconds.

Experiment 4 Performance With Multicast

As you multicast is always important to QoS settings in DDS application multi-subscriber Scenario because its a promising way to deliver high throughput and low latency. However, WeaveNet is a Unix CNI that emulates multicast. Based on the previous experiment we observed that Open vSwitch based VXLAN encapsulation used by WeaveNet introduces more overhead than other CNIs. So we want to know whether the multicast feature can make up the overhead cost of VXLAN encapsulation, then making WeaveNet a more promising Virtual network plugin multi subscriber DDS deployment. To this end we ran experiments with 1pub-to-4sub, 1pub-to-8sub, & 1pub-to-12sub, we then configure the payload length to 1KB, as well as enabled the reliability QoS and disabled the micro batching QoS.

- The most left pie chart shows the performance of WeaveNet-Multicast lags behind all other unicast solutions when there are four subscribers, then exceeds WeaveNet-Unicast and Flannel-VXLAN in the 1pub-to-8subs use case, eventually becoming the optimal CNI as the number of subscribers increases to 12.
- For Weavenet-Multicast, the process of wrapping samples to individual subscribers takes place in the Weave container, note this container is not the Perftest container where our benchmark applications are running it's another container maintained by the WeaveNet Daemon process. Since the wrapping occurs in the Weave Container we can see an increase in CPU utilization on the host.
- WeaveNet-Multicast effectively saves host memory because packets are copied only once in the multi-subscriber use cases.
- The average latency over all subscribers for WeaveNet-Multicast is greater than others which means the delay in the introduction by emulating multicast is trivial.

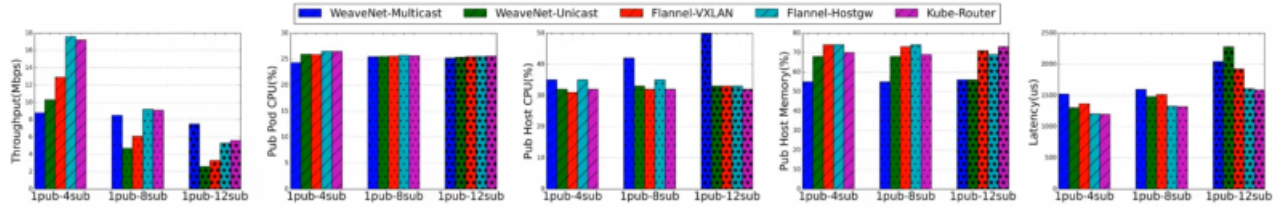


Fig6. Performance comparison of multicast/unicast-enabled CNIs in aspects of throughput, pub pod CPU usage, pub host CPU usage, pub host memory usage, and 90th ping-pong latency. The left four plots are produced by the same throughput tests. QoS settings: Reliable QoS: enabled, Batching QoS: disabled, Transport: UDP. Payload length: 1KB. Test period: 120 seconds.

Experiment 5 Security Support

Security Specs

The last experiment is to explore the performance of DDS application communication when security is a concern. Users can secure their packets using the DDS security plugin or Kubernetes CNIs. DDS security supports five protection levels, where the data integrity and confidentiality is are supported in RTPS message, sub-message, and payload levels. The source authenticity is supported in the RTPS message and sub-message level. In terms of Kubernetes CNI, Flannel and WeaveNet support the network-level data security based on IPsec ESP.

- Flannel-IPSec operates in the ESP tunnel mode the encapsulates and encrypts whole IP packet.
- WeaveNet-Encrpy: Encapsulates packets using Open vSwitch VXLAN and encrypts UDP datagrams in ESP transport mode. So there is more overhead in tunnel mode than transport mode.

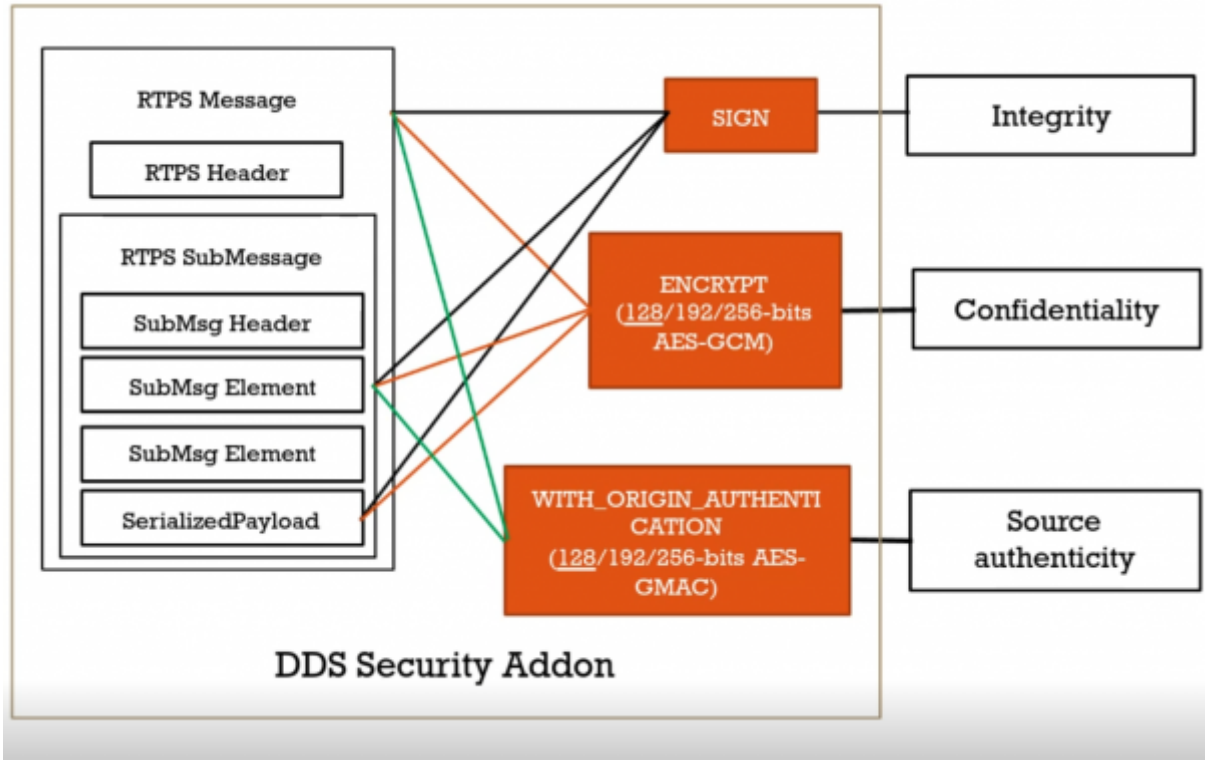


Figure 8: This illustrates the DDS Security Plugin

Security Performance Evaluation

If we enable all protection levels in the DDS security plugin then the combination with the Kube-

Router and DDS security plugin is comparable to Flannel-IPSec and WeaveNet-Encrypt Our experiment result shows that Kube-Router with DDS security(SignWithOrigAuthRTPSEncryptSM) performs better than Flannel-IPSec when sending messages that are smaller than 1KB but earns lower throughput and higher latency for larger ones. The fine-grained data protection functions that are unique to the DDS security plugin enables applications to plug tailored security options to avoid performance overhead induced by unnecessary security operations. Although the ESP transport mode of WeaveNet-encrypts incurs lower encapsulation overhead than the tunnel mode in Flannel, the impact of VXLAN encapsulation offsets the overall performance. For Multi subscriber use cases, DDS security produces better throughput performance than security provided by Network level security.

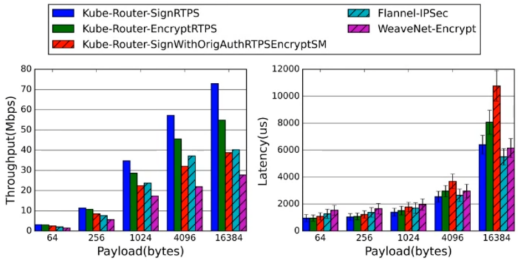


Fig7. Performance comparison of security implementations on application-layer(DDS Security plugin) and network- layer(Flannel-IPSec, WeaveNet-Encrypt). Communication pattern: **1pub-1sub**. Reliability: enabled, Batching: disabled. Test period: 120 seconds.

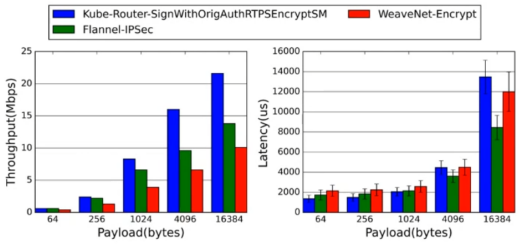


Fig8. Performance comparison of security implementations on application-layer(DDS Security plugin) and network- layer(Flannel-IPSec, WeaveNet-Encrypt) in multi-subscriber use case. Communication pattern: **1pub-4sub**. Reliability: enabled, Batching: disabled. Test period: 120 seconds.

Conclusion

- DDS Containers earn better performance when using Kube-Router or Flannel-hostgw CNI in reliable unicast communication.
- Enabling DDS BestEffort QoS is a good practice to reduce latency and L3 CNI shows better performance improvement than Flannel-hostgw and Kube-router.
- For WeaveNet, the overhead induced by WeaveNet VXLAN encapsulation may offset its multicast's benefit when operating in small-scale clusters. But it promises better scalability in larger-scale multi-subscriber use cases.
- The DDS Security extension offers more flexible and fine-grained protection than IPSec approaches supported by Flannel & WeaveNet.
- In the 1pub/1sub communication, DDS security outperforms Flannel-IPSec and WeaveNet-Encrypt when the payload is smaller than MTU, while incurs overhead when sending large samples.
- In multi-subscriber use cases, DDS security is the best practice for achieving high performance and scalability.
- Flannel-IPSec performs better then WeaveNet-Encrypt in Unicast communication.

What's Next

- We plan to evaluate the performance of containerized DDS applications with dynamic data flows as well as develop intelligent configuration tuning frameworks for DDS and Kubernetes CNI knobs under particular QoS.
- The cross-cluster communication of containerized DDS applications has not been systematically studied.
- Lastly, this paper that is linked above, will guide our future research to depict a blueprint to achieving high-performance and automated deployment of containerized DDS applications on Kubernetes platforms.