A Comprehensive Benchmark of Flannel CNI in SDN/non-SDN Enabled Cloud-Native Environments

José Santos *, Bibin V. Ninan †, Bruno Volckaert *, Filip De Turck *, Mays Al-Naday †
* Ghent University - imec, IDLab, Department of Information Technology, Gent, Belgium
Email: {josepedro.pereiradossantos, bruno.volckaert, filip.deturck}@UGent.be
† University of Essex, Colchester, UK
Email: {mfhaln}@essex.ac.uk

***Abstract*—The emergence of cloud computing has driven advancements in software virtualization, particularly microservice containerization. This in turn led to the development of Container Network Interfaces (CNIs) such as Flannel to connect microservices over a network. Despite their objective to provide connectivity, CNIs have not been adequately benchmarked when containers are connected over an external network. This creates uncertainty about the operation reliability of CNIs in distributed edge-cloud ecosystems. Given the multitude of available CNIs and the complexity of comparing different ones, this paper focuses on the widely adopted CNI, Flannel. It proposes the design of novel benchmarks of Flannel across external networks, Software Defined Networking (SDN)-based and non-SDN, characterizing two of the key backend types of Flannel: User Datagram Protocol (UDP) and Virtual Extensible LAN (VXLAN). Unlike existing benchmarks, this study analysis the overhead introduced by the external network and the impact of network disruptions. The paper outlines the systematic approach to benchmarking a set of Key Performance Indicators (KPIs), including: speed, latency and throughput. A variety of network disruptions have been induced to analyse their impact on these KPIs, including: delay, packet loss, and packet corruption. The results show that VXLAN consistently outperforms UDP, offering superior bandwidth with efficient resource consumption, making it more suitable for production environments. In contrast, the UDP backend is suitable for real-time video streaming applications due to its higher data rate and lower jitter, though it requires higher resource utilization. Moreover, the results show less variation in KPIs over SDN, compared to non-SDN. The benchmark data are made publicly available in an open-source repository, enabling researchers to replicate the experiments, and potentially extend the study to other CNIs. This work contributes to the network management domain by providing an extensive benchmark study on container networking highlighting the main advantages and disadvantages of current technologies.**

***Index Terms*—Containers, Container Network Interfaces, Network Function Virtualization, Benchmark, Cloud-Native, Software-Defined Networking**

I. Introduction

Over recent years, application development and deployment has been profoundly transformed by advancements in virtualization and containerization technologies [1], [2]. Containers offer significant advantages over traditional Virtual Machines (VMs), such as lightweight deployment, ease of management, and robust isolation [3], [4]. Docker [5], and Containerd [6] have revolutionized containerization by leveraging the host's kernel and libraries, thus avoiding the overhead associated with running a full guest operating system, which is typical in VMs. This has led to the emergence of cloud-native applications, whereby software is decomposed into containerized microservices connected over the network to provide the overall application function [7], [8].

As the scale of microservices and the complexity of their mesh grows, there is a pressing need to develop networking solutions fit for container environments and can be orchestrated automatically at the granularity of microservices. Container Network Interfaces (CNIs) have been developed by the Kubernetes (K8s) community to meet this need [9]. CNIs enable networking across diverse systems and platforms, dynamically managing network resources and configurations as containers are created and destroyed [10]. Popular CNI plugins, such as Flannel [11], Calico [12], and Cilium [13] offer unique capabilities for container networking tailored to meet specific requirements [14], [15]. Despite the tailored design of various CNIs, their performance has not yet been adequately evaluated. Particularly when used to connect microservices over an external network, Software Defined Networking (SDN)-based or traditional (non-SDN), having a higher likelihood of disruptions. Although, recent works have studied the performance of container networks based on various factors, they lacked focus on integration with external networks [10], [15]–[20] (Further details in Sec. II). This leads to: *1)* uncertainty in the operational reliability of CNIs; *2)* lack of understanding of the network impact on end-to-end performance; and, *3)* limited ability to match a CNI with the suitable network backend for a specific use case.

This paper provides a novel, extensive, benchmark[1] of two backend types of one of the most widely adopted CNI, Flannel. Specifically, it analyzes the primary backends that enable Flannel overlay networks: User Datagram Protocol (UDP) and Virtual Extensible LAN (VXLAN). Through systematic and automated benchmarking of the two backends, this study provides a comprehensive performance evaluation of a set of Key Performance Indicators (KPI), including: speed, latency, and throughput. Other backends exist in Flannel [21], such as *host-gw* that create IP routes to subnets via remote machine IPs, and *WireGuard* which uses in-kernel WireGuard to en-

Manuscript received May 10, 2025

[1]All logs and graphs are openly available for review at: https://github.com/jpedro1992/container-networking/

capsulate and encrypt the packets. An extensive benchmark has been conducted in which the network status dynamically varies during the experiments in terms of delay, packet loss and packet corruption. Our experimental results indicate that VXLAN outperforms UDP in terms of overall performance, although UDP shows advantages in data transfer and lower jitter on average. Moreover, results show that variation in KPI values in SDN is considerably less than traditional non-SDN. The main contributions of this work can be summarized as follows:

- First, this paper introduces the applied benchmark methodology to assess the network performance of containers based on the Flannel CNI.
- Second, we detail the experiment procedure, the two testbeds, and the metrics used to assess the performance of the Flannel overlay network.
- Third, we comprehensively analyze the results obtained based on our extensive benchmark in which the network status dynamically varies during the experiments in terms of delay, packet loss and packet corruption.

The remainder of the paper is organized as follows: Sec. II discusses the literature on container networking. Sec. III presents the benchmark methodology while describing the applied components. Sec. IV describes the evaluation setup, followed by the results in Sec. V. Finally, Sec. VI concludes this paper.

II. Background & Related Work

Containers [22], [23] are a lightweight virtualization technology that allows applications to run in isolated environments. Docker, one of the most widely used containerization environments, provides a way to package applications along with their dependencies into a single container image [24]. This image can be deployed consistently across different environments, ensuring that applications behave similarly regardless of where they are executed. In addition, K8s is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications [25]. It provides a robust framework for running distributed systems resiliently, handling failover, scaling applications based on demand, and managing the lifecycle of containers. K8s abstracts the underlying infrastructure, allowing developers to focus on application logic while it handles networking, storage, and scheduling. K8s extensibility and vibrant ecosystem of plugins and tools further enhance its capability to support a wide range of use cases in modern cloud-native environments [26].

Container Network Interface (CNI) plugins [18] are responsible for configuring network interfaces in containers and assigning them IP addresses. Flannel is a popular CNI plugin that provides a simple overlay network using the VXLAN protocol [27]. Flannel creates a flat network where each host gets a subnet of IP addresses, which it assigns to its containers. This overlay network abstracts the complexity of the underlying network, allowing containers to communicate seamlessly across different hosts. Flannel is designed to work out-of-the-box with K8s, facilitating the deployment of containerized applications in distributed environments. Calico [12] is a widely used CNI plugin that provides scalable Layer 3 networking with Border Gateway Protocol (BGP)-based routing. It enables direct communication across different hosts without needing an overlay network, offering improved performance and simplified network troubleshooting. Similarly, Cilium [13] is another powerful CNI plugin built on eBPF technology, offering high-performance networking and advanced observability. Cilium extends beyond traditional network policies by supporting Layer 7 policies, enabling control over application-level traffic.

Kubernetes (K8s) Networking is abstracted through the CNI, a standard framework that allows different network solutions to manage pod IP allocation, routing, and inter-node communication. Flannel is one of the simplest CNIs, designed primarily for K8s clusters that do not require advanced network policies or deep observability. Flannel assigns each K8s node a unique subnet from a pre-configured IP address pool. This ensures that each pod gets a unique IP without requiring NAT, simplifying communication within the cluster. Flannel supports different backend mechanisms to handle pod-to-pod traffic:

- **Overlay Networking:** In VXLAN and UDP modes, Flannel encapsulates packets before transmission, allowing pods across different nodes to communicate seamlessly. The trade-off is additional encapsulation overhead and increased CPU utilization, which can affect performance at scale.
- **Underlay Networking:** In host-gw mode, Flannel avoids encapsulation by setting up direct routes between nodes. This reduces overhead but limits network flexibility, as all nodes must reside within the same Layer 2 domain.

These design choices position Flannel as a lightweight, easy-to-deploy solution, making it well-suited for edge deployments, small-scale clusters, and environments prioritizing simplicity over fine-grained control. However, it lacks the dynamic policy enforcement and network observability offered by CNIs like Cilium (eBPF) and Calico (BGP). In addition, Flannel relies on a key-value store (typically etcd) to manage network configuration data. While this architecture simplifies deployment, Flannel's dependency on etcd has been observed to introduce performance bottlenecks in large-scale clusters, particularly during network reconfigurations or under high churn. In contrast, Calico implements a pure Layer 3 networking model using BGP to distribute routing information efficiently across the cluster. This design enables high scalability and fine-grained network policy enforcement, making it suitable for complex, large-scale deployments. Calico can also operate in policy-only mode, enabling flexibility in hybrid networking setups. Cilium, on the other hand, is a modern CNI that leverages eBPF to insert networking logic directly into the Linux kernel. This provides low-latency packet processing, real-time observability, and security policy enforcement without the need for iptables or sidecars. Cilium's integration with observability tools (e.g., Hubble[2]) offers deep insight into packet flow and system behavior, positioning it as a leading solution in performance-sensitive and security-aware

[2]https://github.com/cilium/hubble

environments. While Calico and Cilium represent the state-of-the-art in CNI innovation, these CNIs also bring increased complexity and resource demands.

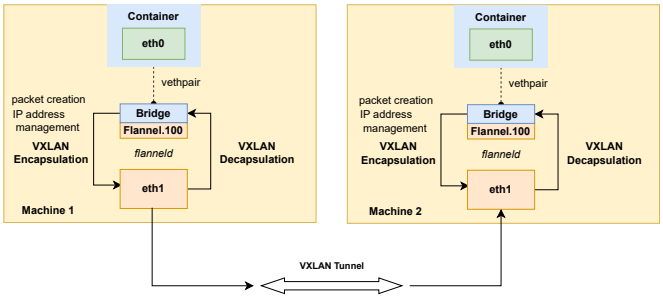
Software Defined Networking (SDN) [28], [29] and OpenFlow (OF) [30], [31] have significantly transformed networking. SDN, which uses software-based controllers to manage network traffic dynamically, often uses the widely adopted OpenFlow (OF) protocol to interact with the forwarding plane of network devices such as switches and routers. Open vSwitch (OVS) is an open-source implementation of a virtual switch supporting OF that provides advanced network functions, including traffic filtering, monitoring, and Quality of Service (QoS). It enables flexible and programmable network management, making it an essential component for building scalable and efficient SDN architectures.

Numerous studies have been conducted to benchmark the performance of container networks in the last few years [10], [15]–[20]. These studies evaluate various performance factors, such as network throughput, latency, and the impact of different CNI plugins. These prior works have provided insights into the trade-offs between ease of setup and network performance between enabling Docker's native networking or applying CNI plugins such as Flannel and Calico. For example, Kang et al. [10] further explored performance differences between K8s CNI plugins, identifying key trade-offs in complexity and efficiency. More recently, Koukis et al. [20] investigated the performance of CNIs in edge environments, emphasizing the growing relevance of CNIs under constrained network conditions. While these studies provided valuable benchmarks, they generally focused on isolated or intra-cluster network performance and did not systematically account for the role of external network dynamics, such as disruptions or variations in latency and packet loss. Moreover, few of these studies explicitly considered SDN-enabled infrastructures or compared their behavior with traditional non-SDN deployments under controlled degradation.

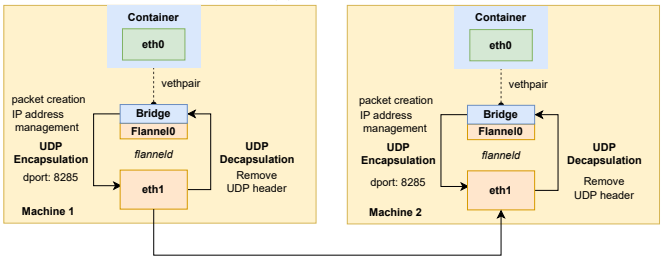
Our contribution addresses this specific gap by analyzing the performance of Flannel over external networks under both SDN (i.e., using OVS with OpenFlow rules) and non-SDN (i.e., using native Linux bridges) setups, while introducing artificial impairments during the experiments in terms of delay, packet loss, and packet corruption to emulate real-world network dynamics. We focus on benchmarking a single CNI—as opposed to multiples—given a wide range of network conditions, in order to provide a comprehensive assessment of the network impact and the CNI behaviour when microservices are distributed and connected over an external network. We aim to refer to this analysis in future work that compares multiple CNIs. We selected Flannel as the CNI plugin for benchmarking due to its simplicity, ease of deployment, and extensive adoption in K8s environments. Flannel serves as a widely used baseline for performance evaluations due to its straightforward overlay networking approach, making it an ideal candidate for investigating fundamental performance characteristics under various network conditions. Prior works [10], [15], [20] have also used Flannel as a representative baseline when comparing different CNIs. Additionally, our study focused on analyzing the impact of different network

TABLE I: A comparison between Flannel's backends.

Backend	VXLAN	UDP
Device name	flannel.100	flannel0
UDP Port	8472	7890/8285
Data Transfer	Unicast	Multicast
Networking	through physical network	through tunnel device



(a) VXLAN.



(b) UDP.

Fig. 1: The Flannel CNI plugin's internal operation.

backends, which Flannel natively supports. This allowed us to isolate the effects of these backends without introducing additional complexities, such as policy enforcement, BGP-based routing, or security rules found in CNIs like Calico. The aim of this study is to establish a clear performance baseline before expanding the scope to more feature-rich CNIs. Nevertheless, we acknowledge that comparing Flannel with other CNIs, such as Calico, Cilium, or Weave, would provide additional insights into different networking approaches, and we plan to explore this in future work by extending our benchmarking framework to include multiple CNIs and analyzing their respective trade-offs. Further details about the benchmark methodology of this study are presented in the next section.

III. Benchmark Methodology

This section presents the end-to-end system applied to assess the network performance of containers based on the Flannel CNI. Performance metrics used to evaluate Flannel are detailed in Sec. IV.

A. System Overview

Flannel, developed by CoreOS, is a widely used CNI plugin designed to provide simple and efficient networking for K8s clusters. It facilitates a large internal network and subnets on each host using the K8s Application Programming Interface (API) or etcd for network configuration. Unlike more advanced CNIs such as Calico (which leverages BGP for routing) or Cilium (which utilizes eBPF for enhanced security and observability), Flannel focuses on lightweight

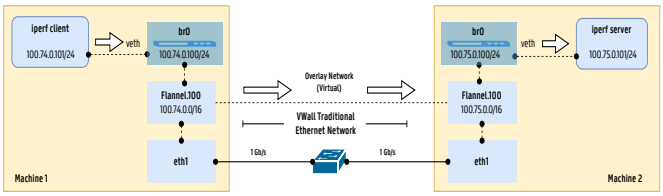
and straightforward network overlay solutions. It supports multiple backend mechanisms, including VXLAN and UDP, which determine how packets are encapsulated and forwarded across nodes. It creates a layer 3 IPv4 network across nodes in a cluster, assigning IP addresses and maintaining a routing table via etcd, and encapsulating Transmission Control Protocol (TCP) data in network packets. Each host runs a *flanneld* agent, providing subnet leases. Comparative details of VXLAN and UDP backends are summarized in Table I, the two main adopted backends. Fig 1 outlines the architecture of Flannel, emphasizing the role of the *flanneld* daemon in managing IP address assignment and maintaining the overlay network. It also details the packet processing flow, including how packets are encapsulated using either VXLAN (Fig. 1a) or UDP (Fig. 1b), depending on the selected backend.

The control plane is responsible for handling the configuration and state of the network. It executes tasks such as address management and configuration distribution. If there is an issue in the control plane, there will be a delay in contacting pods in the newly created network. Protocols such as BGP and OSPF operate within this plane. Other functions include traffic segmentation, data flow prioritization, and traffic isolation across different parts of the network. The control plane ensures that each node knows the correct IP address range, subnet, and how to route traffic between them.

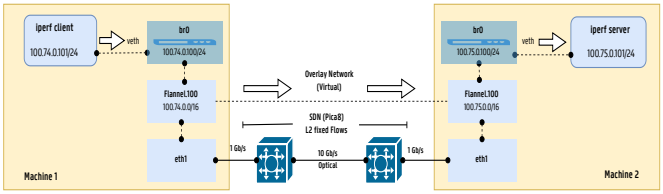
The data plane is responsible for handling data transmission between the nodes and containers, relying on the underlying physical network to establish virtual networks and paths. Key configurations in the data plane include the backend and Maximum Transmission Unit (MTU) settings. The backend defines different network settings and routing mechanisms used to transmit packets.

VXLAN [32] is a network overlay technology that enables Layer 2 connectivity over a Layer 3 network. It uses UDP-based encapsulation, where each Ethernet frame is wrapped inside a VXLAN header and transmitted over an underlying IP network. VXLAN introduces a 24-bit segment ID (VNI), allowing up to 16 million isolated virtual networks. It uses the in-kernel VXLAN feature to encapsulate packets. The default UDP port for sending encapsulated packets is 8472 in Linux and 4789 in Windows. VXLAN creates a virtual interface linked to the physical network device (e.g., eth0) for tunneling. The *flanneld* daemon provides the node's Address Resolution Protocol (ARP) table and Forwarding Database Bridge (FDB), allowing the virtual device to know which path to choose for forwarding traffic. Encapsulation occurs within the kernel, avoiding overhead from data transfers between user space and the kernel. While VXLAN provides better scalability than traditional VLANs, it also introduces additional encapsulation overhead, which may impact performance in certain environments.

UDP [33] is primarily used for debugging or when other kernel options are unavailable. UDP forms a TUN device to encapsulate IP fragments in UDP packets, creating an overlay network. The *flanneld* daemon creates the TUN device, enabling packet sending and receiving for user space programs. In UDP mode, *flanneld* encapsulates packets and sends them over the physical interface, simultaneously unwrapping received packets.



(a) Virtual Wall (VWall) [34].



(b) Essex setup.

Fig. 2: Overview of the Flannel implementation in both testbeds. Testbed details are discussed in Sec. IV-B.

This study focuses on Flannel's UDP and VXLAN backends, which represent the most commonly used configurations for enabling cross-node communication in K8s clusters through overlay networking. Although Flannel also supports host-gw and WireGuard backends, they were excluded from our benchmark for specific reasons: *1)* the host-gw backend, while offering high performance, requires direct Layer 2 adjacency between nodes and is typically unsuitable for virtualized cloud environments or multi-hop networks; *2)* WireGuard, on the other hand, remains an experimental feature within Flannel, with limited documentation and adoption at the time of our experiments. Our selection ensures consistency across testbeds and reflects the practical deployment scenarios encountered in resource-constrained and edge-oriented clusters.

B. Flannel Implementation

Fig. 2 illustrates the flannel implementation used in this study, in which Flannel has been set up across two machines to facilitate container connectivity. Both hosts have been connected using an Ethernet cable, which served as the data plane for the cluster. After establishing connectivity, etcd [35] has been installed on both machines to act as a key-value store for system details and managing subnet allocations across both machines. Each node obtains a unique subnet range from etcd, which is then used for inter-container communication through the overlay. While this design simplifies IP address management, it introduces a centralized coordination point that may become a bottleneck under high churn or large-scale operations. Specifically, etcd's strong consistency model (via the Raft consensus protocol) ensures correctness, but can introduce write latency and consistency delays, especially when managing frequent lease updates or handling node joins/leaves in larger clusters. These effects are minimal in our setup, where etcd experiences limited contention. However, in multi-tenant or geographically distributed environments, etcd performance can degrade, potentially delaying Flannel's responsiveness in allocating or updating routes. Recent studies [36], [37] have shown that etcd can experience increased write

TABLE II: The different networking configurations applied with traffic control (tc).

Test Name	Type	Description
Baseline	-	No tc rule added.
Delay	*low*	10 ms 1ms 25%
	medium	100 ms 10ms 25%
	high	500 ms 50ms 25%
Loss	*low*	loss 5%
	medium	loss 20%
	high	loss 60%
Corruption	*low*	corrupt 5%
	medium	corrupt 20%
	high	corrupt 40%
Delay-Loss	low	10 ms 1ms 25% - loss 5%
	medium	100 ms 10ms 25% - loss 5%
	high	500 ms 50ms 25% - loss 5%
Delay-Corruption	*low*	10 ms 1ms 25% - corrupt 5%
	medium	100 ms 10ms 25% - corrupt 5%
	high	500 ms 50ms 25% - corrupt 5%

latency under concurrent access, which may affect pod time to readiness and the responsiveness to network reconfiguration. Alternatives such as using local caches, reducing etcd API calls, or adopting CNIs that minimize centralized coordination (e.g., Calico without etcd) may offer more scalability in large-scale deployments.

Once etcd is running on both systems, both nodes form a cluster, allowing each other to detect network changes and disruptions in the other, and display their status as healthy. Next, Flannel specifications, including network details, subnet ranges, and backend types, have been configured in a file. The *flanneld* process has been executed on both nodes, mapping the respective IP addresses. Two virtual bridges have been created on both nodes using OVS, along with a virtual interface pair to connect containers to the virtual switch. Both *ovs-vsctl* and *ovs-ofctl* utilities are used to create and manage the connection between both nodes. Also, in the Essex testbed, we installed default flow rules on the SDN switches with *ovs-ofctl add-flow* rules to ensure proper Layer 2 forwarding. For both backends (UDP and VXLAN), the MTU is set to 1450 bytes because Flannel subtracts 50 bytes for encapsulation overhead from the default 1500 bytes MTU. Docker needs to be restarted at this stage so that the *docker0* bridge receives a subnet from the Flannel network. An IP address from the subnet is then manually assigned to the *br0* interface after the restart. To complete the setup, Ubuntu containers have been deployed on both systems, and a virtual adapter pair is created to connect each container to the default bridge. IP addresses were assigned to both containers within the bridge range. Lastly, connectivity is verified by running a ping command between containers, followed by more advanced performance tests further detailed in the next section.

IV. Evaluation Setup

This section details the experiment procedure, the two testbeds, and the metrics used to assess the performance of the Flannel overlay network.

A. Experiment Procedure

The performance evaluation is based on a simplified setup involving two containers connected via a Flannel overlay network to ensure a controlled benchmarking environment. This approach allowed us to systematically isolate and analyze the effects of different Flannel backends on protocol performance and resource consumption. The aim is to ensure that the observed differences could be directly attributed to the networking stack and infrastructure characteristics, rather than to side effects introduced by large-scale container deployments. Nonetheless, we acknowledge the importance of scalability, and as part of our future work, we intend to extend the current evaluation to larger-scale scenarios involving multiple nodes and a significantly higher number of containers.

Iperf3 [38] has been chosen to assess the network performance due to its reliability, reproducibility, and wide adoption in network performance benchmarking [39], [40]. It has been used to test each backend under various conditions, increasing the bandwidth from 500 Mbits/s to 1500Mbits/s in four steps, with each test running for 5 minutes over 10 iterations. The Iperf3 client (in the first container) sent packets to the Iperf3 server (in the second container), and performance has been measured and saved in several logs. While we acknowledge that other packet-level tracing tools such as *tcpdump* or *bpftrace* could offer fine-grained insights, these were beyond the scope of our evaluation. As part of future work, we intend to extend our methodology with eBPF-based observability to better characterize performance variations in the container network stack under varying workloads and deployment contexts.

Prometheus [41], a widely popular monitoring platform, has also been used to monitor resource consumption (mainly CPU and memory usage) of both Flannel's backends. Do note that we limited the scope of monitoring the resource consumption of the overall CPU and memory usage of the client node to maintain experimental consistency and to reduce variability introduced by potential asynchronous behavior or system noise on the server node. This aims to ensure a controlled and comparable benchmarking environment across all tested scenarios. Nevertheless, we acknowledge that server-side processing may introduce additional overhead, particularly under asymmetric or high-traffic loads.

Also, traffic control (tc) has been applied to configure the kernel packet scheduler in the Flannel's interface (of the client), simulating multiple packet delays, loss, and corruption. Table II shows the different networking configurations applied during the experiments. The delay configurations introduced using tc are designed to emulate realistic edge-cloud latency scenarios. Specifically:

- **10 ms:** reflects the round-trip time commonly observed in 5G edge networks or regional data centers, where ultra-low-latency connectivity is supported through local infrastructure.
- **100 ms:** simulates more typical cloud-to-edge communication over 4G or Internet backbone connections with moderate distance and routing complexity.
- **500 ms:** represents degraded or extreme latency conditions, which may occur in highly congested networks, long-haul international traffic, or under failure-recovery paths.

Furthermore, each delay setting includes variation ($\pm$ 1 – 50ms) and correlation (25%) to mimic jitter and temporal dependencies found in real-world networks. For example, the

(a) Virtual Wall (VWall) [34].

(b) Network Convergence Lab (NCL) [42].

Fig. 3: Overview of each testbed used in the evaluation.

delay experiment with type *low* sets a network delay of 10ms on the Flannel's interface with a uniform random variation of ±1ms and a 25% correlation, simulating realistic network conditions where delays are not entirely random but have some predictability. The use of these configurations allows us to evaluate the CNIs under a spectrum of conditions, ranging from optimal to severely impaired environments. The combined setups (e.g., *delay-loss*, *delay-corruption*) further extend this realism, approximating multi-factor impairments in cloud-native systems. This approach ensures a comprehensive understanding of Flannel's performance under various conditions, providing valuable insights into its efficiency and reliability.

Do note that Docker restarts have been used to reset container state and interface configurations between benchmarking runs, ensuring clean start conditions. This approach, although less scalable than production-grade orchestration, aligns with established benchmarking practices in the networking community when fine-tuning testbed conditions is critical to avoid interference from transient system states. In future work, we plan to extend our experiments to fully K8s-native deployments and explore how dynamic orchestration affects CNI performance under real-world scaling scenarios.

B. Testbeds

Two distinct testbeds have been deployed across different architectures to validate the results through various scenarios. The first testbed (Fig. 3a) is deployed in the imec Virtual Wall (VWall) infrastructure at IDLab, Belgium [34]. The connection between the two machines utilizes a 1Gb link via a switch, which is the standard setup for machine connections in the VWall testbed (Fig. 2a). The second scenario, named as the Essex testbed (Fig. 3b), is implemented at the University of Essex, UK. The connection between the two machines in this testbed is established with an SDN OVS switch known as Pica8[3] (Fig. 2b). The aim is to compare the performance of a software-based OVS, implemented in VWall, with a hardware-based OVS, deployed in Essex. Table III details the hardware configurations of each testbed while listing the applied software versions.

TABLE III: The hardware and software configurations of each testbed.

Virtual Wall (VWall)		
Node	**CPU**	**Memory**
Worker 1 Worker 2	Intel(R) Xeon(R) CPU 8-core E5-2650 v2 @ 2.60GHz	48 GB
Software	**Version**	
Operating System	Ubuntu 20.04.4 LTS	
Docker version	24.0.5	
TCP version	TCP Cubic	
Essex		
Node	**CPU**	**Memory**
Worker 1 Worker 2	AMD EPYC 7281 16-Core Processor @ 2.7GHz	32 GB
Software	**Version**	
Operating System	Ubuntu 22.04.4 LTS	
Docker version	24.0.5	
TCP version	TCP Cubic	

C. Performance Metrics

The network performance has been evaluated on the basis of the following metrics:

- **Bandwidth (Mbit/s)**: Indicates the maximum data rate the network can support between both containers, reflecting the network's speed.
- **Throughput (Mbit/s)**: Measures the actual amount of data transferred from source to destination, helping to assess the network's reliability.
- **Lost Datagrams (%)**: Evaluates the amount of data lost during transmission.
- **Jitter (ms)**: Shows the variation in packet arrival times at the destination, crucial for real-time and streaming applications such as audio, and video streaming.
- **Number of Retries**: Counts the number of attempts to resend lost packets, mainly relevant for the TCP protocol.
- **CPU Usage (%)**: Monitors the CPU consumption of the node during the experiment.
- **Memory Usage (%)**: Tracks the memory usage of the node during the experiment.

V. Results

A. Resource consumption

Table IV shows the resource consumption of VXLAN and UDP backends across different protocols (UDP and TCP) under varying bandwidths for both testbeds (Essex and VWall),

[3]https://www.pica8.com/

TABLE IV: The resource consumption of VXLAN and UDP backends over different protocols (UDP and TCP).

Backend	Protocol	Bandwidth (in Mbit/s)	CPU Usage (in%)	Memory Usage (in%)
UDP	UDP	Idle	V: 0.1 - E: 0.1	V: 2.4 - E: 4.0
		500.0	V: 3.8 - E: 3.2	V: 2.4 - E: 4.0
		800.0	V: 4.5 - E: 5.7	V: 2.4 - E: 4.0
		1000.0	V: 4.9 - E: 5.0	V: 2.4 - E: 4.1
		1500.0	V: 5.0 - E: 5.1	V: 2.4 - E: 4.1
	TCP	Idle	V: 0.1 - E: 0.1	V: 2.4 - E: 4.0
		500.0	V: 3.8 - E: 6.7	V: 2.4 - E: 4.0
		800.0	V: 3.9 - E: 3.6	V: 2.4 - E: 4.0
		1000.0	V: 4.1 - E: 3.5	V: 2.4 - E: 4.1
		1500.0	V: 4.2 - E: 3.6	V: 2.4 - E: 4.1
VXLAN	UDP	Idle	V: 0.1 - E: 0.1	V: 2.4 - E: 4.0
		500.0	V: 2.9 - E: 3.1	V: 2.4 - E: 4.0
		800.0	V: 3.0 - E: 3.2	V: 2.4 - E: 4.0
		1000.0	V: 3.1 - E: 3.2	V: 2.4 - E: 4.0
		1500.0	V: 3.2 - E: 3.2	V: 2.4 - E: 4.0
	TCP	Idle	V: 0.1 - E: 0.1	V: 2.4 - E: 4.0
		500.0	V: 0.3 - E: 3.2	V: 2.4 - E: 4.0
		800.0	V: 1.7 - E: 3.2	V: 2.4 - E: 4.0
		1000.0	V: 0.8 - E: 0.4	V: 2.4 - E: 4.0
		1500.0	V: 0.6 - E: 0.5	V: 2.4 - E: 4.0

Testbeds: V = VWall, E = Essex.

focusing on assessing the CPU and memory usage of the node hosting the Iperf client.

CPU usage generally scales with increased bandwidth for both backends. However, notable differences emerge between the two testbeds. For instance, under the UDP protocol, the UDP backend in VWall shows a gradual increase in CPU usage, peaking at 5.0% at 1500 Mbit/s, while Essex shows a similar trend but with slightly higher CPU consumption at certain bandwidths. In contrast, for the TCP protocol, Essex demonstrates higher CPU usage for the UDP backend, especially at lower bandwidths (e.g., 500 Mbit/s with 6.7% CPU usage), while VWall's CPU usage remains comparatively stable. For the VXLAN backend, CPU usage is consistently lower across all bandwidths. Under the UDP protocol, CPU usage remains between 3.1% and 3.2% at Essex, and under the TCP protocol, it significantly drops in VWall, with a notable decrease at 1000 Mbit/s (0.8%) and 1500 Mbit/s (0.6%). This suggests that VXLAN, particularly under TCP, is more CPU-efficient in the VWall environment.

Memory usage remains largely consistent across all conditions and environments, with no significant variations. Both VXLAN and UDP backends exhibit stable memory consumption, maintaining 2.4% usage in VWall and 4.0% in Essex across all bandwidth levels. This indicates that memory usage is not a significant differentiator between the two backends or the different protocols under consideration.

The differences in resource consumption between the two testbeds could be due to the architectural characteristics and packet processing workflows of each backend. Interestingly, VXLAN generally shows lower CPU utilization compared to the UDP backend, particularly on the VWall testbed. This may seem counterintuitive, as VXLAN encapsulation typically introduces additional overhead. However, VXLAN encapsulation is optimized for hardware offloading, reducing the CPU burden on the host system. In contrast, UDP-based tunneling relies on software processing, which can introduce additional overhead, particularly in software-based switching environments. Thus, the VWall setup—relying more heavily on software-based switching—appears to benefit from VXLAN's more structured and offload-friendly packet encapsulation model when operating under TCP traffic. For example, under TCP at 1500 Mbps, VXLAN consumes only 0.6% CPU on VWall, compared to 3.2% for UDP. In contrast, Essex consistently shows slightly higher CPU usage than VWall across most scenarios due to its more aggressive hardware acceleration and tight integration with SDN-based control. While this optimization boosts forwarding performance and control granularity, it also introduces overhead associated with processing rules and flow entries, especially under the UDP backend. This explains the minor CPU increase observed in Essex when UDP is used (e.g., 5.7% at 800 Mbps compared to 4.5% on VWall). Additionally, memory usage remains stable across all configurations, suggesting that CPU is the primary bottleneck affected by backend protocol and infrastructure characteristics.

In summary, the resource consumption results indicate that the VXLAN backend is generally more CPU-efficient than the UDP backend, particularly under TCP in the VWall environment. The slight differences in CPU usage between VWall and Essex highlight the importance of considering the deployment environment when selecting a backend, as resource consumption can vary depending on the specific setup. Overall, these findings indicate that while both VXLAN and UDP backends are viable for use in different networking scenarios, VXLAN may offer advantages in terms of CPU efficiency, particularly when deployed in environments similar to VWall and using TCP protocols. This efficiency could translate to lower operational costs and better performance in resource-constrained environments.

B. Performance Evaluation over TCP and UDP

This section presents the Flannel performance across different backends using UDP and TCP protocols as shown in Table V and Table VI. The experiments have been conducted on both testbeds, focusing on key performance metrics such as the receiver's bandwidth (in Mbit/s), the jitter (in ms), the number of retries, and the percentage of lost datagrams.

UDP Protocol For the UDP backend, in the VWall testbed, the performance was optimal at 500 Mbit/s and 800 Mbit/s, with the received bandwidths matching the sender, and low jitter values of 0.0035 ms and 0.0041 ms, respectively. Lost datagrams have also been minimal, recorded at 0.00008% and 0.0026%. At 1000 Mbit/s, the received bandwidth has been slightly higher at 937.8 Mbit/s, with jitter increasing to 0.0063 ms and lost datagrams to 6.24%. At 1500 Mbit/s, the received bandwidth remained constant at 937.8 Mbit/s, jitter was at 0.0065 ms, and lost datagrams increased to 34.15%. For the Essex testbed, the results at 500 Mbit/s and 800 Mbit/s showed perfect bandwidth utilization with received values matching the sender and low jitter. However, lost datagrams increased slightly from 0.0007% to 0.013%. At 1000 Mbit/s, the received bandwidth was 932.5 Mbit/s, with jitter at 0.0069 ms and lost datagrams at 6.75%. The highest bandwidth setting of 1500 Mbit/s resulted in a received bandwidth of 933.1 Mbit/s, the

TABLE V: The performance of Flannel over different backends for the UDP protocol across the different testbeds.

Backend	Protocol	Band. Sender (in Mbit/s)	Band. Receiver (in Mbit/s)	Jitter (in ms)	Lost Datagrams (in %)
UDP	UDP	500.0	VWall: 500.0 - Essex: 500.0	VWall: 0.0035 - Essex: 0.0077	VWall: 0.00008 - Essex: 0.0007
		800.0	VWall: 800.0 - Essex: 800.0	VWall: 0.0041 - Essex: 0.0071	VWall: 0.0026 - Essex: 0.013
		1000.0	VWall: 937.8 - Essex: 932.5	VWall: 0.0063 - Essex: 0.0069	VWall: 6.24 - Essex: 6.75
		1500.0	VWall: 937.8 - Essex: 933.1	VWall: 0.0065 - Essex: **0.011**	VWall: **34.15** - Essex: **37.78**
VXLAN	UDP	500.0	VWall: 500.0 - Essex: 500.0	VWall: 0.013 - Essex: 0.012	VWall: 0.0001 - Essex: 0.00027
		800.0	VWall: 800.0 - Essex: 800.0	VWall: 0.014 - Essex: 0.013	VWall: 0.00016 - Essex: 0.0011
		1000.0	VWall: 923.0 - Essex: 923.0	VWall: 0.016 - Essex: 0.016	VWall: 7.64 - Essex: 7.64
		1500.0	VWall: 923.0 - Essex: 923.0	VWall: 0.018 - Essex: 0.019	VWall: 18.44 - Essex: 19.2

TABLE VI: The performance of Flannel over different backends for the TCP protocol across the different testbeds.

Backend	Protocol	Band. Sender (in Mbit/s)	Band. Receiver (in Mbit/s)	Number of Retries	Lost Datagrams (in %)
UDP	TCP	500.0	VWall: 500.0 - Essex: 500.0	VWall: 266 - Essex: 0.2	VWall: 0 - Essex: 0
		800.0	VWall: 706.3 - Essex: 799.8	VWall: **80862** - Essex: 5684	VWall: 0 - Essex: 0
		1000.0	VWall: 696.1 - Essex: 901.6	VWall: **83402** - Essex: 9907	VWall: 0 - Essex: 0
		1500.0	VWall: 694.1 - Essex: 896.2	VWall: **84602** - Essex: 10604	VWall: 0 - Essex: 0
VXLAN	TCP	500.0	VWall: 500.0 - Essex: 500.0	VWall: **0.4** - Essex: 0	VWall: 0 - Essex: 0
		800.0	VWall: 800.0 - Essex: 800.0	VWall: **34** - Essex: 0	VWall: 0 - Essex: 0
		1000.0	VWall: 909.0 - Essex: 909.0	VWall: 8040 - Essex: 8017	VWall: 0 - Essex: 0
		1500.0	VWall: 909.0 - Essex: 909.0	VWall: 8003 - Essex: 8013	VWall: 0 - Essex: 0

highest jitter observed at 0.011 ms, and lost datagrams peaking at 37.78%. When using the VXLAN protocol, the VWall testbed also displayed optimal performance at 500 Mbit/s and 800 Mbit/s, with received values matching the sender, and low jitter values of 0.013 ms and 0.014 ms, respectively. At 1500 Mbit/s, the received bandwidth is on average 923 Mbit/s, with jitter increasing to 0.018 ms, and lost datagrams of 18.44%. On the Essex testbed, the VXLAN backend also showed perfect bandwidth utilization at 500 Mbit/s and 800 Mbit/s settings, with minimal jitter (0.012 ms and 0.013 ms), and low lost datagrams (0.00027% and 0.0011%). At the highest setting of 1500 Mbit/s, the received bandwidth is on average 923 Mbit/s, with jitter increasing to 0.019 ms, and lost datagrams of 19.2%.

TCP Protocol For the UDP backend, the VWall testbed exhibited optimal performance at 500 Mbit/s with 266 retries and no lost datagrams. At higher bandwidths (800 Mbit/s, 1000 Mbit/s, and 1500 Mbit/s), the received bandwidths have been reduced to 706.3 Mbit/s, 696.1 Mbit/s, and 694.1 Mbit/s, with extremely high retries (80862, 83402, and 84602), but no lost datagrams. The Essex testbed performed perfectly at 500 Mbit/s with only 0.2 retries and no lost datagrams. At 800 Mbit/s, 1000 Mbit/s, and 1500 Mbit/s settings, the received bandwidths have been high at 799.8 Mbit/s, 901.6 Mbit/s, and 896.2 Mbit/s, with retries increasing to 5684, 9907, and 10604, but no lost datagrams. Under the VXLAN backend, the VWall testbed also showed optimal performance at 500 Mbit/s with 0.4 retries and no lost datagrams. At higher bandwidths (800 Mbit/s, 1000 Mbit/s, and 1500 Mbit/s), the received bandwidths have been 800 Mbit/s and 909 Mbit/s, with increased retries (34, 8040, and 8003) and no lost datagrams. The Essex testbed also achieved perfect performance at 500 Mbit/s and 800 Mbit/s settings, with no retries or lost datagrams. At 1000 Mbit/s and 1500 Mbit/s settings, the received bandwidths have been 909 Mbit/s, with retries of 8017 and 8013, and no lost datagrams.

Overall, the evaluation results indicate that Flannel performs efficiently with both UDP and TCP protocols across different testbeds. Performance varies based on the sender's bandwidth and testbed configurations, with notable differences in jitter, the number of retries, and the percentage of lost datagrams at higher bandwidth settings.

C. Analysis of Flannel performance across various network configurations for Essex and VWall

This section highlights the performance of Flannel for the UDP and TCP protocols across different backends and network configurations as illustrated in Table VII and Table VIII. Several metrics have been analyzed for both testbeds: VWall and Essex. The network configurations examined included the scenarios previously shown in Table II.

UDP Protocol Under low *Delay* conditions, the Essex testbed exhibited a higher bandwidth (791.1 Mbit/s) compared to VWall (762.8 Mbit/s) when using the UDP backend, although VWall had a lower jitter (0.4 ms vs. 0.3 ms). The percentage of lost datagrams has also been significantly lower on VWall (1.8%) compared to Essex (9.6%). For the VXLAN backend, VWall outperformed Essex in terms of bandwidth (786.5 Mbit/s vs. 741.0 Mbit/s) at a slightly higher jitter (0.6 ms vs. 0.4 ms) with a similar percentage of datagrams lost. With medium *Delay*, both testbeds showed similar bandwidths around 113 Mbit/s for the UDP backend, with Essex experiencing higher jitter (3.9 ms) than VWall (3.3 ms). In high *Delay* conditions, both Essex and VWall had high datagram loss rates for both backends (97.1%). For the *Loss* scenario under low conditions, Essex provided higher bandwidth (788.7 Mbit/s) compared to VWall (776.6 Mbit/s) for the UDP backend, with negligible jitter in both cases. However, VWall experienced higher datagram loss (10.5%) than Essex (5.9%). Under high *Loss* conditions, VWall outperformed Essex in terms of bandwidth (356.0 Mbit/s vs. 316.6 Mbit/s) for the UDP backend, with both showing negligible jitter and high datagram loss (60%). For the VXLAN backend, VWall also achieved higher performance, with higher bandwidth and similar datagram loss rates. For the *Corruption* case under low conditions, Essex achieved slightly lower bandwidth

TABLE VII: The performance of Flannel over different backends for the UDP protocol under various network configurations.

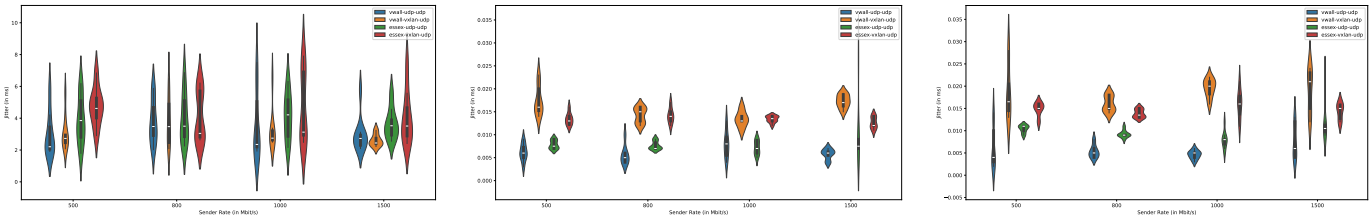
Backend	Test	Type	Band. Receiver (in Mbit/s)	Jitter (in ms)	Lost Datagrams (in %)
UDP	*Delay*	*Low*	V: 762.8 ± 35.3 - E: **791.1 ± 39.5**	V: 0.4 ± 0.03 - E: 0.3 ± 0.02	V: 1.8 ± 0.9 - E: **9.6 ± 2.9**
		Medium	V: 113.1 ± 0.1 - E: 113.0 ± 0.0	V: 3.3 ± 0.4 - E: 3.9 ± 0.4	V: 85.4 ± 1.2 - E: 86.2 ± 1.5
		High	V: 22.6 ± 0.01 - E: 22.7 ± 0.01	V: 13.8 ± 1.2 - E: **19.3 ± 2.3**	V: 97.1 ± 0.3 - E: 97.1 ± 0.3
VXLAN	*Delay*	*Low*	V: 786.5 ± 38.7 - E: 741.0 ± 46.3	V: **0.6 ± 0.04** - E: 0.4 ± 0.05	V: 2.5 ± 0.6 - E: 2.5 ± 0.9
		Medium	V: 111.9 ± 0.1 - E: 111.9 ± 0.1	V: 3.3 ± 0.3 - E: **4.4 ± 0.4**	V: 85.2 ± 1.3 - E: 86.5 ± 1.4
		High	V: 22.2 ± 0.01 - E: 22.3 ± 0.01	V: 14.5 ± 1.4 - E: 15.1 ± 0.9	V: 97.1 ± 0.3 - E: 97.3 ± 0.3
UDP	*Loss*	*Low*	V: 776.6 ± 43.4 - E: 788.7 ± 44.3	V: 0.01 ± 0.0 - E: 0.01 ± 0.0	V: **10.5 ± 2.2** - E: 5.9 ± 0.5
		Medium	V: 681.0 ± 50.3 - E: 693.1 ± 52.0	V: 0.01 ± 0.0 - E: 0.01 ± 0.0	V: 22.6 ± 1.1 - E: 23.9 ± 2.1
		High	V: **356.0 ± V: 31.6** - E: 316.6 ± 48.0	V: 0.02 ± 0.0 - E: 0.01 ± 0.0	V: 60.0 ± 0.0 - E: 60.0 ± 0.01
VXLAN	*Loss*	*Low*	V: 734.2 ± 47.6 - E: 736.8 ± 70.8	V: 0.01 ± 0.0 - E: 0.01 ± 0.0	V: 6.2 ± 0.5 - E: 5.9 ± 0.5
		Medium	V: 676.5 ± 51.4 - E: 678.7 ± 54.3	V: 0.02 ± 0.0 - E: 0.01 ± 0.0	V: 20.2 ± 0.1 - E: 21.2 ± 0.7
		High	V: **351.4 ± 43.6** - E: 303.6 ± 60.7	V: 0.03 ± 0.0 - E: 0.01 ± 0.0	V: 60.0 ± 0.0 - E: 60.0 ± 0.01
UDP	*Corruption*	*Low*	V: **745.0 ± 40.3** - E: 730.3 ± 43.5	V: 0.01 ± 0.0 - E: 0.01 ± 0.0	V: **11.3 ± 2.1** - E: **15.1 ± 3.7**
		Medium	V: 629.0 ± 40.9 - E: 658.5 ± 31.3	V: 0.01 ± 0.0 - E: 0.01 ± 0.0	V: 24.8 ± 1.7 - E: **29.4 ± 3.2**
		High	V: 461.4 ± 36.4 - E: 446.0 ± 41.4	V: 0.01 ± 0.0 - E: 0.01 ± 0.0	V: 43.1 ± 1.8 - E: 43.8 ± 2.9
VXLAN	*Corruption*	*Low*	V: 735.6 ± 41.4 - E: 719.2 ± 45.7	V: 0.02 ± 0.0 - E: 0.01 ± 0.0	V: 8.8 ± 1.3 - E: 7.7 ± 0.9
		Medium	V: 651.2 ± 40.5 - E: 630.7 ± 42.8	V: 0.02 ± 0.0 - E: 0.01 ± 0.0	V: 24.3 ± 1.3 - E: 24.3 ± 1.8
		High	V: 492.8 ± 32.3 - E: 507.1 ± 32.7	V: 0.03 ± 0.0 - E: 0.02 ± 0.0	V: 44.1 ± 1.1 - E: 45.1 ± 1.9

Testbeds: V = VWall, E = Essex.

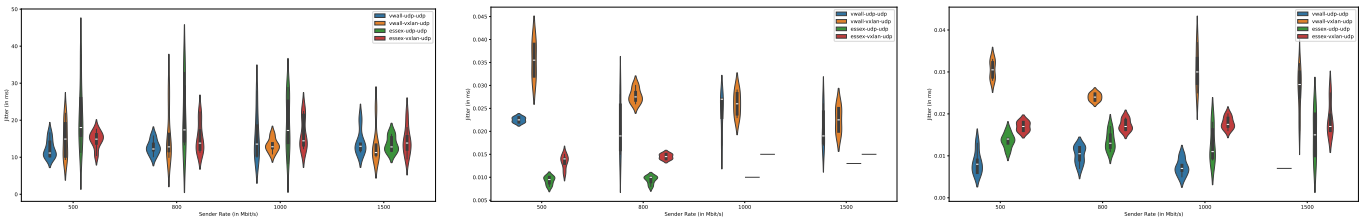
TABLE VIII: The performance of Flannel over different backends for the TCP protocol under various network configurations.

Backend	Test	Type	Band. Receiver (in Mbit/s)	Number of Retries	Lost Datagrams (in %)
UDP	*Delay*	*Low*	V: 308.7 ± 1.9 - E: 504.0 ± 7.5	V: **1728.6 ± 91.6** - E: 1125.0 ± 62.6	V: 0.0 - E: 0.0
		Medium	V: 130.3 ± 1.9 - E: 212.5 ± 1.8	V: **1653.2 ± 102.6** - E: 549.5 ± 122.4	V: 0.0 - E: 0.0
		High	V: 31.5 ± 0.2 - E: 43.4 ± 0.4	V: 1.8 ± 0.7 - E: 0.9 ± 0.6	V: 0.0 - E: 0.0
VXLAN	*Delay*	*Low*	V: **777.7 ± 53.8** - E: **779.1 ± 54.1**	V: 691.8 ± 204.8 - E: 764.5 ± 238.0	V: 0.0 - E: 0.0
		Medium	V: 197.7 ± 0.5 - E: 223.0 ± 0.4	V: 1.2 ± 0.7- E: 1.0 ± 0.6	V: 0.0 - E: 0.0
		High	V: 31.6 ± 0.2 - E: 43.9 ± 0.4	V: 0.9 ± 0.6 - E: 0.9 ± 0.6	V: 0.0 - E: 0.0
UDP	*Loss*	*Low*	V: 95.5 ± 1.1 - E: **252.5 ± 2.9**	V: 132.6K ± 1547.8 - E: 350.7K ± 4004.7	V: 0.0 - E: 0.0
		Medium	V: 0.7 ± 0.02 - E: 0.7 ± 0.01	V: 4643.0 ± 96.4 - E: 4679.4 ± 73.5	V: 0.0 - E: 0.0
		High	V: 0.08 ± 0.1 - E: 0.01 ± 0.0	V: 282.6 ± 73.4 - E: 241.2 ± 48.7	V: 0.0 - E: 0.0
VXLAN	*Loss*	*Low*	V: 155.3 ± 1.7 - E: **259.4 ± 2.9**	V: 219.3K ± 2517.01 - E: 366.5K ± 4037.7	V: 0.0 - E: 0.0
		Medium	V: 0.7 ± 0.02 - E: 0.7 ± 0.05	V: 4925.7 ± 113.7 - E: 4591.9 ± 302.6	V: 0.0 - E: 0.0
		High	V: 0.03 ± 0.04 - E: 0.01 ± 0.0	V: 212.5 ± 44.8 - E: 201.7 ± 73.2	V: 0.0 - E: 0.0
UDP	*Corruption*	*Low*	V: 121.4 ± 2.4 - E: 414.9 ± 3.8	V: 59487.8 ± 734.6 - E: 141.9K ± 1200.9	V: 0.0 - E: 0.0
		Medium	V: 0.7 ± 0.01 - E: 0.7 ± 0.02	V: 4694.6 ± 79.3 - E: 4592.0 ± 132.5	V: 0.0 - E: 0.0
		High	V: 0.08 ± 0.0 - E: 0.07 ± 0.01	V: 1444.0 ± 69.8 - E: 1312.0 ± 130.0	V: 0.0 - E: 0.0
VXLAN	*Corruption*	*Low*	V: 335.7 ± 16.3 - E: **619.8 ± 27.5**	V: 104.5K ± 4994.2 - E: 167.6K ± 6451.1	V: 0.0 - E: 0.0
		Medium	V: **32.1 ± 63.2** - E: 1.1 ± 0.1	V: 5808.7 ± 117.11 - E: 5649.5 ± 284.5	V: 0.0 - E: 0.0
		High	V: 0.14 ± 0.05 - E: 0.10 ± 0.01	V: 1701.9 ± 164.3 - E: 1528.9 ± 203.1	V: 0.0 - E: 0.0

Testbeds: V = VWall, E = Essex.



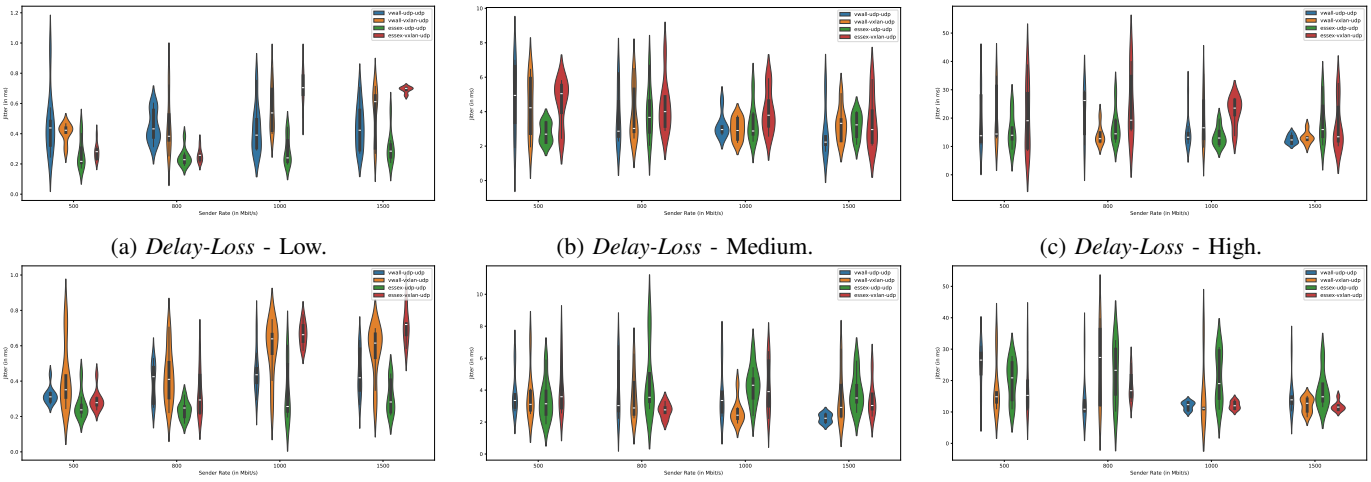
(a) *Delay* - Medium. (b) *Loss* - Medium. (c) *Corruption* - Medium.



(d) *Delay* - High. (e) *Loss* - High. (f) *Corruption* - High.

Fig. 4: The Jitter variations obtained for the *Delay*, *Loss*, and *Corruption* configurations over the UDP protocol.

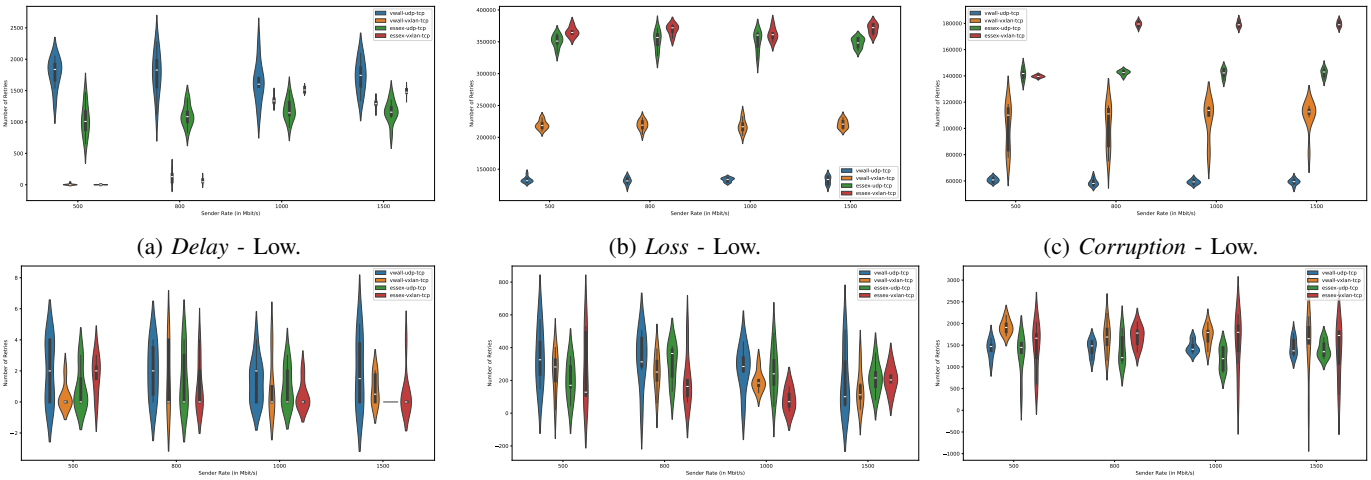
(730.3 Mbit/s) compared to VWall (745.0 Mbit/s) for the UDP backend, and achieved higher datagram loss (15.1%) compared to VWall (11.3%). In high *Corruption* conditions, bandwidth decreased for both testbeds, with Essex achieving 446.0 Mbit/s compared to VWall's 461.4 Mbit/s for the UDP backend. Datagram loss rates were high but comparable across



(a) *Delay-Loss* - Low. (b) *Delay-Loss* - Medium. (c) *Delay-Loss* - High.

(d) *Delay-Corruption* - low. (e) *Delay-Corruption* - Medium. (f) *Delay-Corruption* - High.

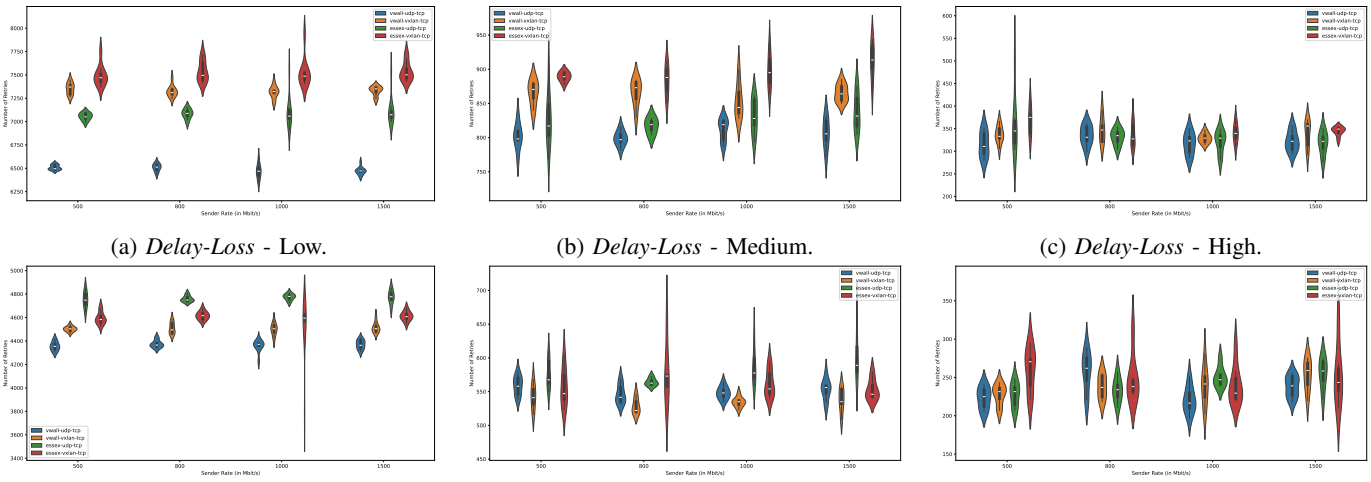
Fig. 5: The Jitter variations for the *Delay-Loss* and *Delay-Corruption* configuration over the UDP protocol.



(a) *Delay* - Low. (b) *Loss* - Low. (c) *Corruption* - Low.

(d) *Delay* - High. (e) *Loss* - High. (f) *Corruption* - High.

Fig. 6: The number of retries obtained for the *Delay*, *Loss*, and *Corruption* configurations over the TCP protocol.



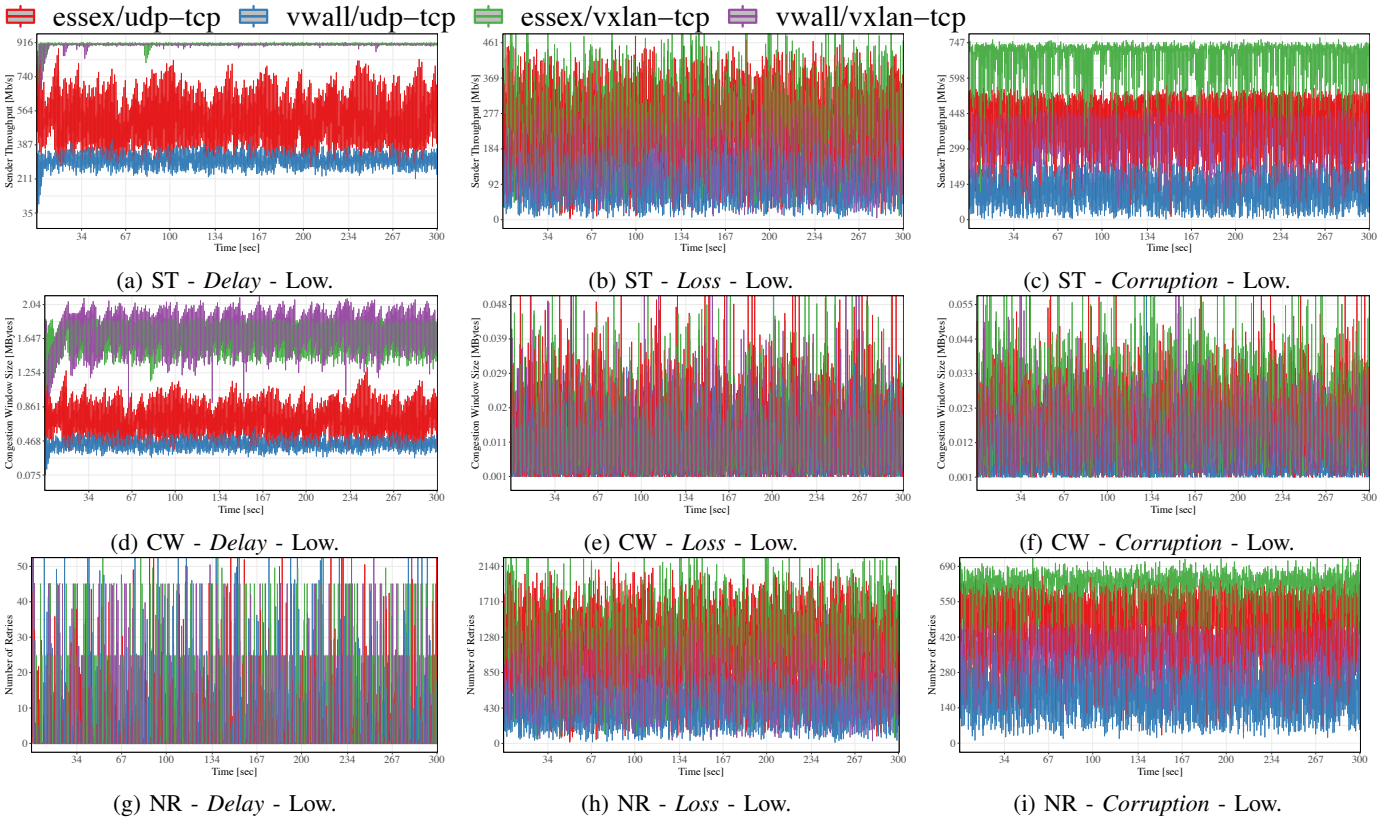
(a) *Delay-Loss* - Low. (b) *Delay-Loss* - Medium. (c) *Delay-Loss* - High.

(d) *Delay-Corruption* - low. (e) *Delay-Corruption* - Medium. (f) *Delay-Corruption* - High.

Fig. 7: The number of retries obtained for the *Delay-Loss* and *Delay-Corruption* configuration over the TCP protocol.

both testbeds. The VXLAN backend showed slightly higher performance on VWall with higher bandwidth and similar datagram loss rates.

Fig. 4 and Fig. 5 highlight the variations in jitter observed

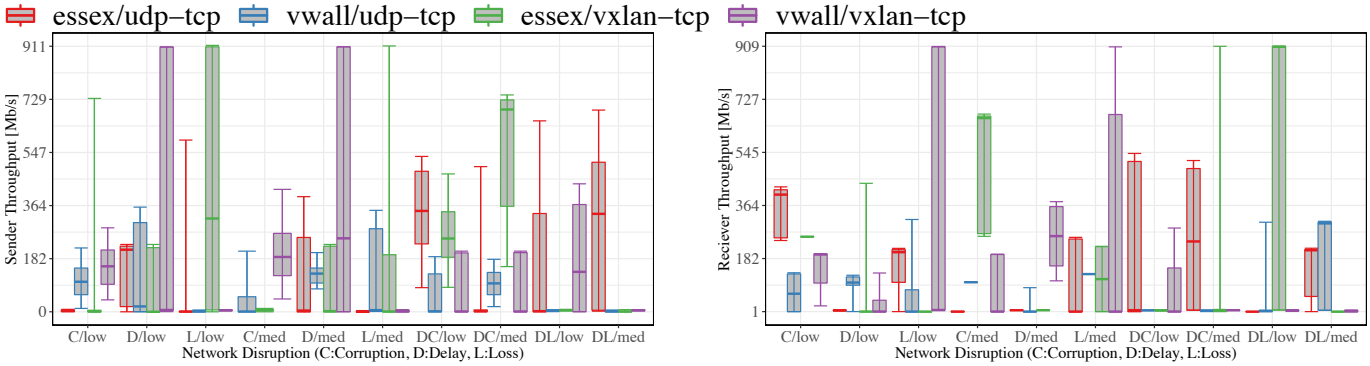


(a) ST - *Delay* - Low. (b) ST - *Loss* - Low. (c) ST - *Corruption* - Low.

(d) CW - *Delay* - Low. (e) CW - *Loss* - Low. (f) CW - *Corruption* - Low.

(g) NR - *Delay* - Low. (h) NR - *Loss* - Low. (i) NR - *Corruption* - Low.

Fig. 8: Sender Throughput (ST), congestion window (CW) and Number of Retries (NR) per second, when the bandwidth is fixed to 1500 Mb/s for the TCP protocol.



(a) Sender side. (b) Receiver side.

Fig. 9: Sender vs. Receiver throughput, when the bandwidth is fixed to 1500 Mb/s for the TCP protocol, given low to medium network disruptions (D: Delay, L: Loss, C: Corruption).

TABLE IX: Summary Table based on the Flannel performance.

Flannel			Network Condition									
Testbed	Backend	Protocol	Default	Delay			Loss			Corruption		
			Normal	Low	Medium	High	Low	Medium	High	Low	Medium	High
Essex	UDP	UDP	✓	✓	✗	✗	✓	✓	✗	✓	✓	✗
VWall			✓	✓	✗	✗	✓	✓	✗	✓	✓	✗
Essex	VXLAN		✓	✓	✗	✗	✓	✓	✗	✓	✓	✗
VWall			✓	✓	✗	✗	✓	✓	✗	✓	✓	✗
Essex	UDP	TCP	✓	✓	✓	✗	✓	✗	✗	✓	✗	✗
VWall			✓	✓	✓	✗	✓	✗	✗	✓	✗	✗
Essex	VXLAN		✓	✓	✓	✗	✓	✗	✗	✓	✗	✗
VWall			✓	✓	✓	✗	✓	✗	✗	✓	✓	✗

Green Color indicates the best performance.
Yellow Color represents a moderate performance.
Red Color denotes poor performance.

across different network conditions for both testbeds. The VWall testbed with the VXLAN backend consistently exhibits higher jitter values, indicating potential instability under certain conditions. Conversely, the UDP backend generally shows lower jitter values across all sender rates, demonstrating more stable performance, particularly under medium conditions. Similarly, the Essex testbed with the UDP backend also displays relatively low jitter values, making it a reliable choice for stable network performance. In contrast, the Essex testbed with the VXLAN backend shows higher jitter values, though not as pronounced as those observed in the VWall testbed. This suggests that while VXLAN encapsulation affects performance, the impact may vary depending on the underlying network setup. As the sender rate increases, jitter values tend to rise, especially in both testbeds using the VXLAN backend. This indicates that higher traffic loads can exacerbate the variability in packet delivery times, particularly when using VXLAN encapsulation.

TCP Protocol Under low *Delay* conditions for the UDP backend, Essex significantly outperformed VWall with a higher bandwidth of 504.0 Mbit/s compared to 308.7 Mbit/s. The number of retries has also been higher for VWall (1728.6) compared to Essex (1125.0). For the VXLAN backend, both testbeds showed similar performance with bandwidths around 779 Mbit/s, although Essex had slightly higher retries. In medium *Delay* conditions, Essex again outperformed VWall in terms of bandwidth (212.5 Mbit/s vs. 130.3 Mbit/s). The number of retries was significantly higher for VWall (1653.2) compared to Essex (549.5). The VXLAN backend showed Essex achieving higher bandwidth (223.0 Mbit/s) with fewer retries compared to VWall. For the *Loss* scenario under low conditions, Essex had a significantly higher bandwidth (252.5 Mbit/s) compared to VWall (95.5 Mbit/s) for the UDP backend, with Essex also experiencing a higher number of retries (350.7K) compared to VWall (132.6K). In medium *Loss* conditions, both testbeds exhibited minimal bandwidth with a similar number of retries around 4600-4700, with Essex slightly outperforming VWall in bandwidth for the VXLAN backend. Under high *Loss* conditions, bandwidth has been negligible for both testbeds with a similar number of retries. In scenarios with low *Corruption*, Essex achieved higher bandwidth (414.9 Mbit/s) compared to VWall (121.4 Mbit/s), with Essex experiencing a higher number of retries (141.9K) compared to VWall (59487.8) for the UDP backend. For medium and high *Corruption* conditions, both testbeds exhibited minimal bandwidth.

The high number of TCP retries observed in both testbeds is a direct consequence of the injected network impairments, with no changes made to the default TCP retransmission thresholds. However, the reported packet loss percentage remains at 0% because packets are recovered through these retransmissions. In addition, in our experiments, Cubic was used as the default congestion control algorithm in Iperf3, which is known for its aggressive probing for bandwidth and multiplicative decrease upon loss detection. This behavior can lead to increased retransmissions, especially in Essex, where higher bandwidth enables Cubic to push the network to its limits more frequently, triggering more losses and subsequent retransmissions. Alternative congestion control algorithms, such as Bottleneck Bandwidth and Round-trip time (BBR) [4], take a fundamentally different approach. Instead of relying on loss-based congestion control, BBRv3, for example, estimates the available bottleneck bandwidth and minimizes retransmissions by maintaining a steady rate, which could potentially reduce retry occurrences in high-bandwidth environments such as Essex.

Fig. 6 and Fig. 7 highlight the variations in the number of retries across different network conditions for both testbeds. The Essex testbed performs poorly in low *Loss* and low *Corruption* scenarios, requiring more retries. However, under low *Delay*, Essex with the VXLAN backend performs well, while the VWall testbed with UDP shows the highest number of retries. In *Delay-Loss* and *Delay-Corruption* conditions, Essex generally experiences more retries. Fig 8 shows the throughput, congestion window, and the number of retries per second of the sender under different network conditions for both testbeds. The VWall testbed with UDP backend shows the lowest throughput (100-300 Mb/s), while Essex consistently outperforms VWall. Essex with UDP maintains the largest congestion window, indicating better transmission efficiency. Both testbeds show similar sensitivity to *Loss* and *Corruption*, though Essex still leads in overall performance. The number of retries fluctuates widely across configurations, with no single setup consistently outperforming others.

Fig. 9 compares sender and receiver throughput under different network disruptions with a fixed bandwidth of 1500 Mb/s for TCP. The Essex testbed for the VXLAN backend generally achieves higher throughput across most conditions, particularly in low *Corruption* and low *Loss* conditions. In contrast, VWall with the UDP backend performs poorly across all scenarios, with minimal throughput. VWall with VXLAN, and Essex with UDP show more varied performance, with VWall occasionally reaching high throughput but lacking consistency under different disruptions. Similar patterns occur on the receiver side. Essex with VXLAN remains the best performer, while VWall with VXLAN shows sporadic high throughput but with greater variability. VWall with UDP continues to underperform. Overall, Essex with VXLAN proves the most reliable for both sender and receiver throughput under different network disruptions.

Use case suitability Furthermore, these results indicate that Essex achieves higher bandwidth but experiences more retries under *Corruption*, which can be attributed to its hardware-accelerated OVS implementation. The optimized packet forwarding in Essex prioritizes throughput, but in high *Loss* environments, TCP retransmissions increase as a mechanism to recover lost packets. This behavior is particularly relevant for different application types:

- **Real-Time Applications:** These applications prioritize low-latency delivery over absolute reliability, meaning that higher bandwidth in Essex may be beneficial despite the increased retries. Real-time protocols such as RTP over UDP often tolerate some packet loss but require sustained high throughput to ensure smooth playback.

[4]https://github.com/google/bbr

- **Data Transfer applications:** In contrast, file transfer applications (e.g., FTP, HTTP-based downloads) demand high reliability, and excessive retries could degrade performance. In this context, VWall, despite slightly lower bandwidth, may provide a more stable performance with fewer retransmissions, reducing overhead for reliable transfers.

Discussion on performance differences To complement the performance results, we also analyzed which underlying causes could be behind the observed differences between Flannel's backends (UDP and VXLAN) across both testbeds (Essex and VWall). Our findings indicate that several architectural and system-level factors significantly contribute to the observed performance variability:

- **VXLAN encapsulation:** The VXLAN backend incurs additional encapsulation overhead, adding approximately 50 bytes per packet. This reduces the effective MTU and can lead to increased fragmentation, especially under high-throughput or high loss conditions. Thus, VXLAN may show reduced throughput and higher retransmissions compared to UDP, particularly in scenarios with packet loss or limited bandwidth. On the other hand, the UDP backend, while simpler and more lightweight, lacks the scalability benefits of VXLAN, which can affect predictability and efficiency in large-scale deployments.
- **VXLAN offloading:** While VXLAN offloading reduces CPU usage, Essex's handling of VXLAN traffic might prioritize throughput over latency consistency, leading to jitter fluctuations in certain scenarios.
- **Architectural differences:** Differences between the two testbeds play a substantial role. The Essex testbed features hardware-accelerated packet processing, allowing for lower CPU usage and reduced latency, particularly under high-bandwidth conditions. In contrast, VWall relies on software-based OVS, introducing additional processing overhead and increased jitter. These distinctions are reflected in our measurements, with Essex consistently demonstrating better performance under similar conditions, especially for UDP traffic. Also, the EPYC processors in Essex utilize aggressive dynamic frequency scaling, which can sometimes introduce slight inconsistencies in packet timing compared to the more predictable performance of the older Intel E5 processors in VWall.
- **TCP retransmissions:** We observed that TCP retries increase significantly in high-loss or high-delay environments, which is expected given TCP's built-in retransmission and congestion control mechanisms. Do note that default kernel-level TCP parameters were not modified, and the high number of retries stems from the injected impairments, not from protocol misconfigurations. Despite these retries, the reported packet loss remains at 0% due to successful retransmissions, a behavior confirmed by comparing baseline and impaired scenarios.
- **Flannel intrinsic mechanisms:** Our experiments also suggest that Flannel's performance is sensitive to kernel-level queuing disciplines and scheduling policies. While not the primary focus of this study, such factors can influence latency and throughput in SDN/non-SDN environments, and we identify them as promising directions for future work.

D. Summary

The performance evaluation of Flannel using both UDP and TCP protocols under various network configurations across Essex and VWall testbeds reveals several insights, illustrated in Table IX. For the UDP protocol, Essex generally demonstrated higher bandwidth and lower jitter compared to VWall under low and medium *Delay* conditions, though VWall had slightly higher performance under high *Delay* conditions. Both testbeds showed significant packet loss in high *Delay* and medium to high *Loss* conditions, with Essex often experiencing slightly higher packet loss percentages. For the TCP protocol, Essex consistently outperformed VWall in terms of bandwidth across all network conditions, although the number of retries was often higher, particularly in low *Delay* and low *Corruption* scenarios. The VXLAN backend typically exhibited more stable and higher bandwidth performance for both testbeds, with Essex again showing marginally better results. This reflects the advantage of hardware-based systems in managing TCP traffic, which often requires more robust handling of retries and congestion. However, Essex struggles with certain network conditions requiring frequent retries.

Overall, Essex tends to provide better performance for both UDP and TCP protocols across most network conditions, particularly attaining higher bandwidth and lower packet loss. However, the choice of the backend (UDP vs. VXLAN) and the specific network conditions (e.g., *Delay*, *Loss*, *Corruption*) can significantly influence these performance metrics. While Essex generally demonstrates higher throughput, our results reveal important nuances that highlight VWall's strengths in certain performance metrics, particularly jitter. In several scenarios, VWall exhibits lower jitter compared to Essex, which we attribute to the deterministic nature of its software-based OVS and reduced parallelization overhead. In contrast, Essex's hardware-accelerated OVS is designed to optimize for throughput and flow scalability, which may introduce transient bursts and variability in packet inter-arrival times due to aggressive batching and offloading. Furthermore, the SDN control plane in Essex allows for dynamic flow rule installation and granular traffic engineering. These capabilities enable more efficient path selection and flow isolation, reducing contention and boosting overall throughput. However, the same mechanisms may introduce microbursts or latency spikes during rapid rule updates or load-balancing events, impacting jitter performance. This highlights the trade-offs inherent in SDN-based acceleration: while beneficial for maximizing bandwidth and overall throughput, these also introduce variability in packet timing, particularly under bursty or adaptive traffic patterns. These findings also demonstrate the importance of selecting appropriate testbeds and configurations to optimize network performance for specific applications and scenarios. Furthermore, the benchmark results and logs are publicly available in an open-source repository, enabling researchers not only to replicate the experiments in their

environments but also to extend the study by applying the benchmark to other CNIs. By leveraging our methodology, researchers can conduct comparative analyses in different networking solutions, contributing to a deeper understanding of CNI performance under various conditions and improving the generalizability of the results in diverse cloud-native environments.

Open-Source contribution To promote reproducibility and further research, we have made all benchmark logs, raw measurement data, and benchmarking scripts publicly available. This enables researchers and practitioners to validate our results, extend our analysis, and apply the dataset to new use cases. Potential applications of our dataset might include:

- **Modeling Flannel Performance:** The detailed logs provide a foundation for building a performance model for Flannel under varying network conditions, helping researchers predict behavior in different cloud and edge environments.
- **Training ML-Based Optimizers:** The dataset can serve as a valuable resource for ML models that aim to optimize container networking parameters dynamically, such as tuning VXLAN configurations or adjusting congestion control strategies.
- **Comparative Studies on CNIs:** By sharing our benchmarking scripts, we encourage the community to apply our methodology to other CNIs (e.g., Calico, Cilium) and contribute to a broader understanding of container networking performance across different architectures.
- **Adaptive Network Management Strategies:** The dataset can inform adaptive orchestration mechanisms (e.g., Reinforcement Learning strategies [43], [44]), where network settings (e.g., UDP vs. VXLAN) are dynamically adjusted based on workload characteristics and real-time performance data.
- **Anomaly Detection & Predictive Maintenance:** Applying AI models to detect deviations in network behavior, allowing proactive performance tuning and mitigation of failures.

VI. Conclusions

This paper provides a comprehensive benchmark analysis of two primary backends used in Flannel overlay networks: UDP and VXLAN. Results show that TCP on top of the VXLAN backend consistently outperforms the other configurations. This configuration offers superior performance in terms of bandwidth and retry counts while maintaining optimal resource consumption, making it the preferred choice for production environments. For TCP at higher bandwidths, the received bandwidth ranged from 800 Mbit/s to 909 Mbit/s, with retries increasing between 8003 and 8040, but no datagrams were lost. In contrast, with the UDP backend at 1500 Mbit/s, both testbeds achieved an average received bandwidth of 923 Mbit/s. However, jitter increased to 0.018 ms - 0.019 ms, and datagram loss ranged from 18% to 19%. For scenarios where resource consumption is less critical, UDP over VXLAN demonstrates the highest data rates and throughput, presenting a viable alternative with significant performance. Conversely, UDP over UDP proves to be the optimal choice for real-time streaming or video applications due to its excellent data rate and minimal jitter, albeit at the expense of higher resource utilization. On the other hand, TCP over UDP backend is not recommended due to its poor performance as shown by the results. These findings highlight the importance of selecting the appropriate backend for specific use cases. This benchmark study contributes valuable insights to the network management community, particularly in the context of container networking. As cloud computing and containerized applications continue to evolve, understanding the performance implications of different CNIs is crucial for optimizing network performance. In future work, we plan to explore additional CNIs and backends, as well as conduct benchmarks in more complex and varied network environments, such as multi-zone K8s environments. Furthermore, building upon the observed performance differences between UDP and VXLAN backends, future work will explore the potential of UDP's low-jitter profile in real-time edge applications, such as Augmented Reality (AR) and video streaming, where maintaining consistent latency is vital for user experience. We also plan to investigate scaling VXLAN's bandwidth in a multi-node cluster, which could help improve the scalability and reliability of VXLAN as a viable option for large-scale edge-cloud deployments, potentially offering a more efficient solution compared to traditional approaches.

Acknowledgment

José Santos is funded by the Research Foundation Flanders (FWO), grant number 1299323N.

References

[1] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Towards low-latency service delivery in a continuum of virtual resources: State-of-the-art and research directions," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, pp. 2557–2589, 2021.

[2] O. Bentaleb, A. S. Belloum, A. Sebaa, and A. El-Maouhab, "Containerization technologies: Taxonomies, applications and challenges," *The Journal of Supercomputing*, vol. 78, no. 1, pp. 1144–1181, 2022.

[3] P. Sharma, L. Chaufournier, P. Shenoy, and Y. Tay, "Containers and virtual machines at scale: A comparative study," in *Proceedings of the 17th international middleware conference*, 2016, pp. 1–13.

[4] Q. Zhang, L. Liu, C. Pu, Q. Dou, L. Wu, and W. Zhou, "A comparative study of containers and virtual machines in big data environment," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 2018, pp. 178–185.

[5] J. N. Acharya and A. C. Suthar, "Docker container orchestration management: A review," in *International Conference on Intelligent Vision and Computing*. Springer, 2021, pp. 140–153.

[6] L. Espe, A. Jindal, V. Podolskiy, and M. Gerndt, "Performance evaluation of container runtimes." in *CLOSER*, 2020, pp. 273–281.

[7] J. Alonso, L. Orue-Echevarria, V. Casola, A. I. Torre, M. Huarte, E. Osaba, and J. L. Lobo, "Understanding the challenges and novel architectural models of multi-cloud native applications–a systematic literature review," *Journal of Cloud Computing*, vol. 12, no. 1, p. 6, 2023.

[8] S. Henning and W. Hasselbring, "A configurable method for benchmarking scalability of cloud-native applications," *Empirical Software Engineering*, vol. 27, no. 6, p. 143, 2022.

[9] A. Poniszewska-Marańda and E. Czechowska, "Kubernetes cluster for automating software production environment," *Sensors*, vol. 21, no. 5, p. 1910, 2021.

[10] Z. Kang, K. An, A. Gokhale, and P. Pazandak, "A comprehensive performance evaluation of different kubernetes cni plugins for edge-based and containerized publish/subscribe applications," in *2021 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2021, pp. 31–42.

[11] Flannel, "Flannel is a simple and easy way to configure a layer 3 network fabric designed for kubernetes." accessed on 2 September 2024. [Online]. Available: https://github.com/flannel-io/flannel.

[12] Calico, "Project calico is an open-source project with an active development and user community." accessed on 2 September 2024. [Online]. Available: https://www.tigera.io/project-calico/.

[13] Cilium, "ebpf-based networking, observability, security." accessed on 2 September 2024. [Online]. Available: https://cilium.io/.

[14] N. Kapočius, "Overview of kubernetes cni plugins performance," *Mokslas–Lietuvos ateitis/Science–Future of Lithuania*, vol. 12, 2020.

[15] S. Novianti and A. Basuki, "The performance analysis of container networking interface plugins in kubernetes," in *Proceedings of the 6th International Conference on Sustainable Information Engineering and Technology*, 2021, pp. 231–234.

[16] Y. Park, H. Yang, and Y. Kim, "Performance analysis of cni (container networking interface) based container network," in *2018 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 2018, pp. 248–250.

[17] S. Qi, S. G. Kulkarni, and K. Ramakrishnan, "Understanding container network interface plugins: design considerations and performance," in *2020 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN*. IEEE, 2020, pp. 1–6.

[18] ——, "Assessing container network interface plugins: Functionality, performance, and scalability," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 656–671, 2020.

[19] R. Kumar and M. C. Trivedi, "Networking analysis and performance comparison of kubernetes cni plugins," in *Advances in Computer, Communication and Computational Sciences: Proceedings of IC4S 2019*. Springer, 2021, pp. 99–109.

[20] G. Koukis, S. Skaperas, I. A. Kapetanidou, L. Mamatas, and V. Tsaoussidis, "Performance evaluation of kubernetes networking approaches across constraint edge environments," in *2024 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2024, pp. 1–6.

[21] Flannel, "Flannel backends." accessed on 2 September 2024. [Online]. Available: https://github.com/flannel-io/flannel/blob/master/Documentation/backends.md.

[22] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE cloud computing*, vol. 1, no. 3, pp. 81–84, 2014.

[23] C. Pahl, A. Brogi, J. Soldani, and P. Jamshidi, "Cloud container technologies: a state-of-the-art review," *IEEE Transactions on Cloud Computing*, vol. 7, no. 3, pp. 677–692, 2017.

[24] S. Singh and N. Singh, "Containers & docker: Emerging roles & future of cloud technology," in *2016 2nd international conference on applied and theoretical computing and communication technology (iCATccT)*. IEEE, 2016, pp. 804–807.

[25] M. Luksa, *Kubernetes in action*. Simon and Schuster, 2017.

[26] B. Burns, J. Beda, K. Hightower, and L. Evenson, *Kubernetes: up and running*. " O'Reilly Media, Inc.", 2022.

[27] R. Figueiredo and K. Subratie, "Edgevpn. io: Open-source virtual private network for seamless edge computing with kubernetes," in *2020 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2020, pp. 190–192.

[28] M. Paliwal, D. Shrimankar, and O. Tembhurne, "Controllers in sdn: A review report," *IEEE access*, vol. 6, pp. 36 256–36 270, 2018.

[29] K. Wang, Y. Wang, D. Zeng, and S. Guo, "An sdn-based architecture for next-generation wireless networks," *IEEE Wireless Communications*, vol. 24, no. 1, pp. 25–31, 2017.

[30] J. Tourrilhes, P. Sharma, S. Banerjee, and J. Pettit, "Sdn and openflow evolution: A standards perspective," *Computer*, vol. 47, no. 11, pp. 22–29, 2014.

[31] J. Miguel-Alonso, "A research review of openflow for datacenter networking," *IEEE Access*, vol. 11, pp. 770–786, 2022.

[32] Y. Zhang, T. Pan, Y. Zheng, E. Song, T. Huang, and Y. Liu, "Vxlan-based int: In-band network telemetry for overlay network monitoring," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2021, pp. 1–2.

[33] A. Faisal and M. Zulkernine, "A secure architecture for tcp/udp-based cloud communications," *International Journal of Information Security*, vol. 20, no. 2, pp. 161–179, 2021.

[34] Virtual Wall, "The virtual wall emulation environment." accessed on 2 September 2024. [Online]. Available: https://doc.ilabt.imec.be/ilabt/virtualwall/index.html.

[35] etcd, "A distributed, reliable key-value store for the most critical data of a distributed system." accessed on 2 September 2024. [Online]. Available: https://etcd.io/.

[36] T. Nagendra and R. Hemavathy, "Unlocking kubernetes networking efficiency: Exploring data processing units for offloading and enhancing container network interfaces," in *2023 4th IEEE Global Conference for Advancement in Technology (GCAT)*. IEEE, 2023, pp. 1–7.

[37] V. Dakić, J. Redžepagić, M. Bašić, and L. Žgrablić, "Performance and latency efficiency evaluation of kubernetes container network interfaces for built-in and custom tuned profiles," *Electronics*, vol. 13, no. 19, p. 3972, 2024.

[38] iperf, "The ultimate speed test tool for tcp, udp and sctp." accessed on 2 September 2024. [Online]. Available: https://iperf.fr/iperf-download.php.

[39] C. Dumitrache, G. Predusca, G. Gavriloaia, N. Angelescu, D. Circiumarescu, and D. C. Puchianu, "Comparative analysis of routing protocols using gns3, wireshark and iperf3," in *2022 14th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*. IEEE, 2022, pp. 1–6.

[40] D. Gedia and L. Perigo, "Performance evaluation of sdn-vnf in virtual machine and container," in *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2018, pp. 1–7.

[41] prometheus, "From metrics to insight." accessed on 2 September 2024. [Online]. Available: https://prometheus.io/.

[42] "Network Convergence Laboratory (NCL)," accessed on 9th September 2024. [Online]. Available: https://www.essex.ac.uk/departments/computer-science-and-electronic-engineering/research/communications-and-networks.

[43] J. Chen, J. Chen, and H. Zhang, "Drl-qor: Deep reinforcement learning-based qos/qoe-aware adaptive online orchestration in nfv-enabled networks," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1758–1774, 2021.

[44] J. Santos, M. Zaccarini, F. Poltronieri, M. Tortonesi, C. Stefanelli, N. Di Cicco, and F. De Turck, "Hephaestusforge: Optimal microservice deployment across the compute continuum via reinforcement learning," *Future Generation Computer Systems*, vol. 166, p. 107680, 2025.