Bachelor's thesis

EFFICIENCY AND UTILIZATION OF VECTOR PACKET PROCESSING IN HIGH-SPEED NETWORKS

Ondřej Slavík

Faculty of Information Technology
Department of Computer Systems
Supervisor: Ing. Jan Fesl, Ph.D.
April 16, 2025

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Efficiency and utilization of Vector Packet Processing in high-speed networks
Student:	Ondřej Slavík
Supervisor:	Ing. Jan Fesl, Ph.D.
Study program:	Informatics
Branch / specialization:	Computer Networks and Internet 2021
Department:	Department of Computer Systems
Validity:	until the end of summer semester 2025/2026

Instructions

Vector Packet Processing (VPP) je moderní softwarový framework, který umožňuje zpracování paketů ve vysokorychlostních sítích na úrovni uživatelského prostoru operačního systému. Významnou výhodou využití VPP by mělo být výrazné zvýšení propustnosti a snížení latence v rámci vysokorychlostní sítě. Zmíněné výhody VPP jsou primárně teoretické a zatím nebyly experimentálně dostatečně prokázány.

V rámci tvorby bakalářské práce postupujte dle níže uvedených kroků:
1) Nastudujte a popište detailně všechny principy, které VPP používá, jak je implementováno a jak lze VPP efektivně využívat.
2) Vytvořte testovací scénáře, které umožní srovnat efektivitu a cenu využití VPP oproti běžnému způsobu zpracování paketů na úrovni jádra operačního systému.
3) Po poradě s vedoucím práce realizujte infrastrukturu vhodnou pro reálné otestování VPP.
4) Na základě bodu 2) proveďte dostatečný počet měření (minimálně stovky) a srovnejte možný dosažitelný průtok, latenci a spotřebu el. energie s využitím resp. bez využití VPP.
5) Proveďte důkladný rozbor a diskuzi výsledků z předchozího kroku a explicitně uveďte nevýhody využití VPP, pokud nějaké budou.

Electronically approved by prof. Ing. Pavel Tvrdík, CSc. on 28 November 2024 in Prague.

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis: Slavík Ondřej. *Efficiency and utilization of Vector Packet Processing in high-speed networks.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2025.

I would like to express my sincere gratitude to my thesis supervisor, Ing. Jan Fesl, Ph.D., for his guidance, support, and valuable insights throughout the entire process of writing this thesis.
My thanks also go to the Silicon Hill club of the CTU Student Union for providing an inspiring environment and the technical resources that supported my work.
Finally, I would like to thank my family for their unwavering support, encouragement, and never-ending patience during my studies.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Section 2373(2) of Act No. 89/2012 Coll., the Civil Code, as amended, I hereby grant a non-exclusive authorization (licence) to utilize this thesis, including all computer programs that are part of it or attached to it and all documentation thereof (hereinafter collectively referred to as the "Work"), to any and all persons who wish to use the Work. Such persons are entitled to use the Work in any manner that does not diminish the value of the Work and for any purpose (including use for profit). This authorisation is unlimited in time, territory and quantity.

In Prague on April 16, 2025

Abstract

Fill in the abstract of this thesis in English. Lorem ipsum dolor sit amet. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Cras pede libero, dapibus nec, pretium sit amet, tempor quis. Sed vel lectus. Donec odio tempus molestie, porttitor ut, iaculis quis, sem. Suspendisse sagittis ultrices augue.

Keywords Vector Packet Processing, Network benchmark, Energy efficiency, Linux network stack, Data Plane Development Kit

Abstrakt

Fill in the abstract of this thesis in Czech. Lorem ipsum dolor sit amet. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Cras pede libero, dapibus nec, pretium sit amet, tempor quis. Sed vel lectus. Donec odio tempus molestie, porttitor ut, iaculis quis, sem. Suspendisse sagittis ultrices augue.

Klíčová slova enter, comma, separated, list, of, keywords, in, CZECH

Contents

Introduction 1

1 Theoretical part 3
- 1.1 "Vector Packet Processing (VPP) and Its Operating Principles 3
  - 1.1.1 Traditional network traffic processing . . . 3
  - 1.1.2 An Introduction to VPP . . . 4
  - 1.1.3 Techniques used in VPP . . . 5
  - 1.1.4 VPP Processing Graph and Graph nodes . . . 6
- 1.2 Implementation of Vector Packet Processing . . . 7
  - 1.2.1 DPDK and Its Role in VPP . . . 7
  - 1.2.2 VPP key architecture components . . . 8
    - 1.2.2.1 VPPINFRA . . . 8
    - 1.2.2.2 VNET . . . 9
    - 1.2.2.3 VLIB . . . 9
    - 1.2.2.4 Plugins . . . 9
  - 1.2.3 Configuration and Startup . . . 9
- 1.3 Utilization of Vector Packet Processing . . . 9
  - 1.3.1 Integration with the SDN/NFV Ecosystem . . . 10
  - 1.3.2 VPP as a Complete Router Solution . . . 10
- 1.4 Survey of Traffic Generation Tools . . . 10

2 Pratical part 13
- 2.1 Building Infrastructure for Measurement . . . 13
- 2.2 Metodology . . . 14
- 2.3 Test Scenarios & Results . . . 14
  - 2.3.1 Bidirectional UDP 1 Gbit/s of 64-bytes packets . . . 15
  - 2.3.2 Bidirectional UDP 1 Gbit/s of 870-bytes packets . . . 16
- 2.4 Presentation and Analysis of Results . . . 17

3 Conclusion 18

A Nějaká příloha 19

Obsah příloh 22

List of Figures

1.1 Picture showing the VPP Processing Graph [4] 7

2.1 Picture showing hardware setup 13

List of Tables

2.1 Hardware details for DUT (Device Under Test) 14
2.2 Hardware details for Tester (Measurement Device) 14
2.3 Result of Bidirectional UDP 1 Gbit/s of 870-bytes packets test 15
2.4 Result of Bidirectional UDP 1 Gbit/s of 870-bytes packets test 16

List of code listings

List of abbreviations

DFA	Deterministic Finite Automaton
FA	Finite Automaton
LPS	Labelled Prüfer Sequence
NFA	Nondeterministic Finite Automaton
NPS	Numbered Prüfer Sequence
XML	Extensible Markup Language
XPath	XML Path Language
XSLT	eXtensible Stylesheet Language Transformations
W3C	World Wide Web Consortium

Introduction

Modern high-performance network devices are usually proprietary systems that combine custom hardware, specialized operating systems, and tightly coupled software. While these solutions offer high throughput and reliability, they are typically expensive, inflexible, and slower to evolve due to their closed design and development model. Vector Packet Processing (VPP) is a high-performance network stack that operates at layers 2 to 4 of the ISO/OSI model. It was originally developed by Cisco Systems, Inc. (which is a world leader in networking) and open-sourced in 2016 under the Fast Data Project (FD.io), that is part of the Linux Foundation. VPP brings the ability to perform efficient, high-speed packet processing on common off-the-shelf (COTS) hardware, across a wide range of platforms and operating systems. Its open and flexible architecture opens the door to a new class of network applications that can be deployed and scaled more easily than traditional hardware appliances. In this way, VPP could represent a shift in the traditionally conservative networking world, echoing the "Mainframe to PC" revolution, where general-purpose systems replaced proprietary platforms, enabling broader innovation and accessibility.

Since VPP was open-sourced only recently, it has not yet been widely adopted by the market, and there are only a limited number of academic studies on the subject. As a result, this area remains underexplored. This thesis aims to contribute to this field by evaluating VPP's[1] performance, with a particular focus on its electricity consumption. The findings could provide valuable insights for the industry and guide future research, especially in light of the increasing importance of energy efficiency, as highlighted in recent forecasts by ČEPS a.s. regarding the future of energy resources in the Czech Republic.

With the development of AI and the growing demand for high-resolution streaming services, it is highly likely that the demand for internet bandwidth

[1]The abbreviation VPP is also commonly used in academic literature to refer to a Virtual Power Plant.

will continue to rise. This will result in an increased need for network equipment capable of processing larger volumes of data more efficiently. Therefore, it is crucial to explore technologies like VPP that are capable to handle this growing demand and to explore their energy efficiency.

This thesis is divided into two parts: Theoretical and Practical. The Theoretical part presents the traditional approach to networking and packet processing, as well as an overview of how VPP is designed and the principles on which it operates. Additionally, it introduces the testing scenarios used. The Practical part describes the testing infrastructure, presents the results of various measurements, and provides an analysis of the findings.

Chapter 1

Theoretical part

1.1 "Vector Packet Processing (VPP) and Its Operating Principles

This section describes the fundamental principles behind the Vector Packet Processing (VPP) technology, which aims to enable efficient and high-performance network packet processing. VPP is built on modern programming and architectural principles that allow maximum utilization of contemporary hardware, particularly in parallel processing and memory access optimization.

The section begins with a brief description of traditional network traffic processing methods used by operating systems and their limitations in terms of performance and scalability. Following that, the architecture of VPP is explored in detail, explaining how packets are processed in vectors, the use of a node graph, and the various techniques that contribute to its high efficiency—such as I/O and compute batching, zero-copy methods, and lock-free multithreading. The purpose of this section is to provide a theoretical foundation for understanding how VPP operates.

1.1.1 Traditional network traffic processing

A *network packet* is a basic unit of data transmitted over a network. It consists of a *header*, which includes control information such as source and destination IP addresses, and a *payload*, which carries the actual user data. Packets are routed independently through the network and reassembled at the destination. This structure allows efficient and reliable communication, even over complex or unreliable network paths.

Currently, packet processing works as follows: a packet arrives at the network card, which then issues a system call (syscall) to the operating system

for packet processing. The microprocessor must save the currently executing instruction, perform a context switch, locate the appropriate service routine in the interrupt vector table, and handle the packet processing. Once completed, it must restore the saved instruction, perform another context switch, and return to processing the interrupted program.

This system for operating peripherals was designed under the assumption that the peripherals would not request interrupts continuously, which is not the case with network devices that need to process large volumes of data split into small parts. This method requires the microprocessor to execute a significant number of instructions not directly related to packet processing. Chase et al. [1] discovered [1] that if MTU is 1500 bytes, then interrupt handling accounts for 20% - 25% of receiver packet-processing overhead. Another disadvantage of tradidtional packet processing is the inefficient handling of cache memory; the processing of the packets one by one in response to interrupts leads to frequent cache misses in both cache & inctruction caches.[2][2]

1.1.2 An Introduction to VPP

Vector Packet Processing (VPP) is a multi-platform network stack that operates at layers 2-4 of the ISO/OSI model and is developed by the FD.io project. It consists of a set of forwarding vertices arranged in an oriented graph and auxiliary software and provides out-of-the-box switch/router functionality. Unlike traditional network stacks, which run in the kernel, VPP operates in user space.

In a traditional approach, packets are processed one by one. In contrast, VPP reads the largest available number of packets called vector from the network interface card (NIC) and processes the entire vector through a VPP node-graph one node at a time. Each node in this graph handles a specific part of the packet processing. This approach reduces cache misses and spreads fixed overhead costs across multiple packets, lowering the average processing cost per packet. Additionally, it allows VPP to take advantage of multiple cores, enabling parallel processing, which significantly improves overall performance.

Vector Packet Processing (VPP) runs on common off-the-shelf hardware (COTS), ensuring its broad compatibility and flexibility for deployment. It supports various architectures such as x86, ARM, and Power, and can be deployed on both standard servers and embedded devices. The design of VPP is agnostic to hardware, kernel, and deployment platform, meaning it can operate across a wide range of systems, including bare metal servers, virtual machines (VMs), and containers. This approach allows VPP to be deployed on widely available infrastructure without the need for specialized hardware.[3]

[1] kap. 3.3 obr. 6
[2] kap. 4.2

1.1.3 Techniques used in VPP

DOPAST !!! Low-level code optimization technique

According to Linguaglossa et al.[4] VPP uses theese kernel-bypass techniques:

- **Lock-Free Multi-Threading (LFMT)** is a programming technique that leverages modern multi-core CPUs to increase system performance. In network applications, parallelism is achieved by running multiple threads in the same time. Ideally, the more threads are used, the better the system performance but only up to a saturation point beyond which additional threads bring no gainns. However, to reach this ideal performance, traditional synchronization mechanisms such as mutexes and semaphores must be avoided, as they introduce delays due to thread contention. Instead, lock-free architectures have to be used, allowing threads to operate independently without blocking each other. In the context of VPP this approach is enabled by hardware features like multi-queue NICs, which allow each thread to handle a distinct subset of traffic, ensuring efficient and parallel processing.
- **I/O batching (IOB)** is a key technique used in VPP. Instead of raising an interrupt for every incoming packet, the network interface card (NIC) collects multiple packets into a buffer and triggers an interrupt only when the buffer is full. This reduces the overhead caused by frequent context switching and interrupt handling. VPP typically uses poll-mode drivers, which collect packets in batches without relying on interrupts. Moreover, the batching technique is applied system-wide in VPP. This approach maximizes CPU efficiency, improves cache usage, and delivers stable, high-throughput performance even under heavy load.
- **Compute batching (CB)** is a technique that extends I/O batching to the processing phase itself. Instead of processing one packet at a time, network functions are designed to operate on entire batches of packets. This approach minimizes overhead from function calls (such as context switches and stack setup) and improves instruction cache efficiency. When a batch of packets enters a processing function, only the first packet might cause an instruction cache miss, while the rest benefit from already-warmed cache. Additionally it is possible to take advatage of instruction-level parallelism.
- **Receive-Side Scaling (RSS)** is a hardware-based technique used by modern NICs to distribute incoming packets across multiple RX queues. This enables parallel packet processing by allowing each queue to be handled by a separate thread, improving scalability and throughput. Packet assignment is typically done using a hash function over packet header fields (e.g., the 5-tuple).

- **Zero-Copy (Z-C)** is a technique used to eliminate unnecessary memory copying during packet processing. Instead of copying incoming packets from the network interface card (NIC) to a separate buffer via system calls, the NIC writes packets directly into a pre-allocated memory region that is shared with the user-space application via Direct Memory Access (DMA). This allows the application to access packet data without invoking system calls or duplicating memory, significantly reducing CPU overhead.
- **Cache Coherence and Locality (CC&L)** are critical factors in the performance of modern software-based packet processing systems. In current COTS architectures, memory access has become a major bottleneck, which is mitigated by a multi-level cache hierarchy. Minimizing cache misses and maintaining data locality during packet processing is essential for achieving high performance and low latency.
- **Low-Level Parallelism (LLP)** refers to the ability to exploit the internal micro-architecture of modern CPUs, including multi-stage pipelines, arithmetic-logical units (ALUs), and branch predictors that help maintain pipeline efficiency. Well-optimized code can keep these pipelines full and execute multiple instructions per clock cycle, increasing overall throughput. Performance can be further improved by giving hints to the compiler – such as indicating the likely outcome of conditional branches – to reduce pipeline stalls. Vectorized packet processing and specific coding practices can take full advantage of these hardware features and VPP was specifically designed to taky advantage of LLP.

1.1.4 VPP Processing Graph and Graph nodes

At the core of VPP (Vector Packet Processing) lies the Packet Processing Graph, a directed graph composed of relatively small, modular & loosely coupled nodes. Each node is designed to perform a specific task and there are 3 types of them: *process*, *input* & *internal*. Process nodes do not participate in the packet forwarding graph; instead, they handle timers, events, and other background tasks within the VPP runtime. Input nodes are used for input of data and internal nodes are used for vector processing. Internal nodes also serves as output nodes. When a vector of packets is prepared by input node, it is then pushed through the internal nodes. During processing, the vector may be split if the batch contains packets of different protocols or types, as they may need to follow different paths through the graph When the original vector is completely processed, the process repeats. Illustration of this Processing Graph is shown in fig. 1.1.

Thanks to VPP's modular design, the processing graph is highly customizable and extensible. New nodes – referred to as plugins – can be easily added to implement specific functionality or repleace existing ones. Plugins are shared libraries that are loaded during startup of VPP, and they are not dependent

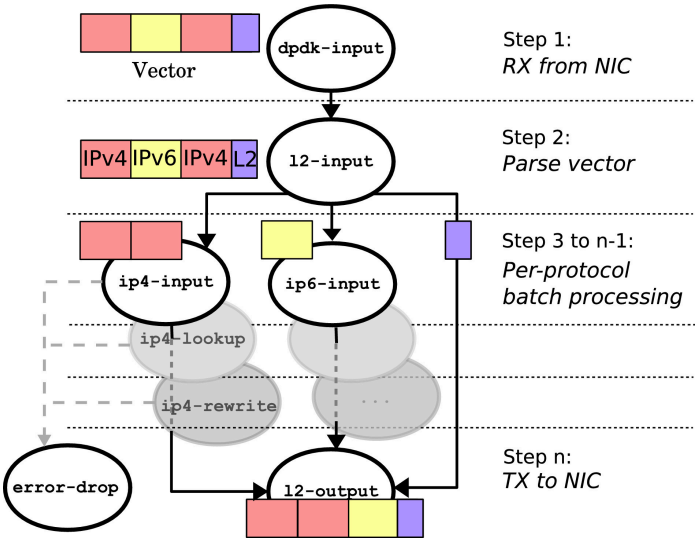

Figure 1.1 Picture showing the VPP Processing Graph [4]

on the VPP source code, allowing them to be developed independently. Moreover, existing nodes can be rewired to modify the packet processing logic when necessary.[4, 5, 6]

1.2 Implementation of Vector Packet Processing

1.2.1 DPDK and Its Role in VPP

přesunout

The Data Plane Development Kit (DPDK) is an open-source collection of libraries and drivers designed to support high-speed packet processing in user space. It was initially developed by Intel in 2010 and is now maintained as a Linux Foundation project. DPDK provides a set of APIs and components that allow applications to bypass the kernel network stack and directly access network interface cards (NICs) through poll-mode drivers, significantly reducing the overhead associated with traditional packet handling mechanisms.[7]

The DPDK completely bypasses the kernel, communicating directly with the NIC. DPDK avoids the use of the kernel's system calls, instead handling its own I/O synchronization and memory management. DPDK employs a Poll Mode Driver (PMD) that uses busy-polling to retrieve, process, and deliver network packets to user-space applications without relying on interrupts. While this approach enhances performance by reducing latency, it also results in high CPU utilization, with the CPU usage on each core remaining close to 100% regardless of the network load.[8]

DPDK is used in VPP for interfacing with hardware. It is implemented as a plugin called *dpdk-plugin*.[4, 5]

1.2.2 VPP key architecture components

VPP's dataplane is implemented by four main architectural layers: VPPINFRA, VNET, VLIB, and Plugins. Each layer provides distinct functionalities that support efficient networking operations, from low-level data structure management to high-level network function optimizations. The following sections describe these layers in detail: [3]

- **VPPINFRA** – layer providing foundational libraries for performing tasks with memory, vectors, rings, lookups in hash tables & timers.
- **VNET** – layer that deals with networking on layers 2 - 4 and is responsible for Control plane.
- **VLIB** – layer that provides library for vector processing, implements CLI and handles application management functions.
- **Plugins** – layer which is a set of plugins that allow for adding network functions and optimizations tailored to specific needs.

1.2.2.1 VPPINFRA

VPPINFRA is a collection of library services designed to offer high-performance capabilities for various tasks. It includes features such as dynamic arrays, hashes, bitmaps, high-precision real-time clock support, event logging, and data structure serialization. The following functionalities are implemented:

- **Vectors** – dynamically resized arrays with *headers* defined by user. They serve as a core building block for other data structures (e.g., hash tables, pools) and allow efficient memory reuse via safe length resetting.
- **Bitmaps** – dynamic bitmaps based on VPPINFRA vectors.
- **Pools** – structures used to quickly allocate & free fixed-size data structures.
- **Hashes** – structures thats provide fast key-value lookups, commonly mapping keys to indices in vectors or pools. Bihash is used in the data plane for fixed-size keys and is thread-safe, while the simpler hash is used in the control plane for exact string matching.
- **Timekeeping** – service providing high-precision, low-cost timing based on CPU ticks. Since CPU ticks are not perfectly accurate, the system continuously adjusts its estimate of "ticks per second" by comparing with the kernel's time. This results in precise and smooth time measurements without the need for expensive system calls.

[3] https://my-vpp-docs.readthedocs.io/en/vpp-config/gettingstarted/developers/swarch/softwarearchitecture.html

- **Timer wheel** – system for efficiently managing timers or timeouts. It allows the user to define parameters like the number of wheels, slots per ring, and timers per object, optimizing time-based operations in systems requiring high-performance event management.

1.2.2.2 VNET

odmítám dělat teď

1.2.2.3 VLIB

Zítra je taky den

1.2.2.4 Plugins

Plugins are used to modify or create new features into the VPP. Developers can create plugins through a straightforward process, involving the generation of necessary files and integration into the system. After building, the new plugin can be loaded and tested within the VPP environment.

VLIB supports a simple mechanism for loading and using plugins. VLIB client applications specify a directory where the plug-ins are located and can apply a filter to narrow down the search. Once the plug-ins are loaded, VLIB ensures they are correctly registered and ready for use.

1.2.3 Configuration and Startup

1.3 Utilization of Vector Packet Processing

VPP supports a comprehensive set of Layer 2 to Layer 4 network features. At Layer 2, it provides Ethernet bridging, MAC learning, VLAN tagging (including dot1q and QinQ), and support for L2 cross-connects and policers.

At Layer 3, VPP implements both IPv4 and IPv6 routing with ECMP support, NAT44/NAT64, and ACL-based filtering. It also supports tunneling mechanisms such as GTP-U, IP-in-IP, and VXLAN. Segment routing (SRv6), LISP, and punt redirect mechanisms are included as well.

At the transport layer (L4), basic UDP and TCP stack functionality is available, enabling packet forwarding and processing for a wide range of use cases within virtualized networking environments.

Additionally, supported features include PPPoE, the WireGuard VPN protocol, GRE tunneling, DHCP client and proxy functionality, and L2TPv3.[9]

According to the authors, VPP can be for example effectively utilized as a virtual switch, virtual router, gateway or used as a basis for a firewall, IDS and load balancer.[3] It already includes enough features to be deployed in production environments.

1.3.1 Integration with the SDN/NFV Ecosystem

To meet the requirements of modern virtualized and cloud-native networking environments, Vector Packet Processing (VPP) was architected with a clear separation between the data plane and control plane. This design choice enables its integration into SDN and NFV frameworks, where packet forwarding logic can operate independently from centralized control mechanisms. VPP's modularity and userspace implementation allow it to function efficiently within dynamic, multi-tenant infrastructure, while remaining compatible with orchestration systems and control-plane protocols commonly used in such deployments

VPP is fully compatible with both Virtual Network Functions (VNFs) and Cloud-Native Network Functions (CNFs). Its modular architecture allows deployment in environments utilizing service function chaining, Kubernetes-based orchestration, or OpenStack-based infrastructures. Because of its userspace design and performance-optimized data plane, VPP can serve as the fast packet processing backend for SDN-controlled systems and NFV orchestrators.[10]

1.3.2 VPP as a Complete Router Solution

Vector Packet Processing (VPP) is implemented solely as a data-plane, meaning it is not a complete routing solution on its own. VPP is dedicated to efficiently forwarding packets between interfaces based on routing rules and access control filters, but it does not include a native control-plane or support for dynamic routing protocols such as BGP or OSPF.

However, as demonstrated by the authors of the VBSR (VPP-Bird Software Router) project [11], it is possible to integrate VPP with additional components such as the Linux Control Plane (Linux-CP) plugin and the BIRD routing daemon. Bird acting as a control-plane enables dynamic routing using protocols like BGP and the Linux-CP is responsible for communication between VPP and BIRD This integrated system creates a nearly feature-complete router solution, comparable in functionality to commercial routers.

It is important to note, however, that firewall functionality is still limited and was left by authors of VBSR as a future work.[11] While VPP supports basic packet filtering through ACLs, it lacks advanced stateful firewall features[9]. These would need to be handled externally.

1.4 Survey of Traffic Generation Tools

In order to evaluate the performance of network devices and data-plane frameworks such as VPP, synthetic traffic must be generated in a controlled and reproducible manner. Selecting appropriate traffic generation tools is therefore essential for conducting accurate benchmarking and stress-testing. Although numerous traffic generation tools exist [12], this section focuses on a subset

commonly used for high-performance benchmarking and synthetic traffic generation in research and practice, namely iPerf3, D-ITG, TRex, Pktgen-DPDK & Genesids.

- **iPerf3** – iPerf3 is a network testing tool used to measure TCP, UDP, and SCTP throughput between two endpoints. It allows detailed configuration of testing parameters such as buffer size, number of parallel streams, test duration, and jitter. iPerf3 can also measure jitter, providing insights into the variation in packet arrival times, which is useful for evaluating network stability. Its client-server architecture makes it a common tool for performance benchmarking of networks and devices.[13]
- **D-ITG** – Distributed Internet Traffic Generator is a network traffic generator designed to produce traffic flows that accurately emulate a wide range of real-world application behaviors. It supports multiple transport layer protocols, including TCP, UDP, DCCP, and SCTP. D-ITG allows users to define parameters such as packet size, inter-departure time, and number of flows, making it suitable for controlled experiments on delay, jitter, packet loss, and throughput. It can operate in both single-node and distributed modes, enabling flexible deployment for testing complex topologies and performance conditions. D-ITG also includes tools for logging and analyzing the generated traffic, facilitating detailed post-experiment evaluation.[14]
- **TRex** – TRex, developed by Cisco, is a high-performance, stateful and stateless traffic generator built on top of DPDK. It supports the generation of realistic Layer 4–7 traffic using pre-recorded PCAP files and emulates multiple concurrent users and flows. TRex is especially suited for benchmarking network function virtualization (NFV) platforms, routers, and firewalls in both laboratory and production-like environments.[15]
- **Pktgen-DPDK** – Pktgen-DPDK is a high-performance traffic generator tool developed as part of the Data Plane Development Kit (DPDK). Pktgen-DPDK supports various network protocols, including IPv4, IPv6, UDP, and TCP. The tool allows precise control over traffic parameters, such as packet rate, size, and timing. Pktgen-DPDK is used in network performance tests and can capture packet-level statistics to assess the performance of the devices under test.[16]

Among the reviewed tools, the author decided to utilize iPerf3 and TRex in the subsequent experimental evaluation. IPerf3 was selected due to its status as a de facto standard for basic throughput and jitter measurements, ease of use, and widespread adoption in academic and practical contexts. TRex was chosen for its modern architecture, support for high-speed stateful and stateless traffic generation, ability to simulate real-world traffic. In addition, TRex provides a Python-based API that enables scripting and automation of test scenarios, making it well-suited for integration into continuous testing

pipelines and reproducible experiments. Other tools, such as Pktgen-DPDK and D-ITG, were excluded due to their relatively complex usage (Pktgen-DPDK) or limited maintenance and outdated design (D-ITG).

Chapter 2

Pratical part

2.1 Building Infrastructure for Measurement

The testing infrastructure has been implemented as recomended in RFC 2544, which defines methods for evaluating network performance. It consists of a device under test (DUT), connected to a measurement device called *Tester*.[1]. In line with the more modern RFC 8219, which states that: "All tests described SHOULD be performed with bidirectional traffic" [17], the infrastructure is designed to operate with bidirectional traffic. This approach ensures more accurate performance measurement under real-world network conditions, as opposed to unidirectional traffic. The Device Under Test (DUT) and the measurement device are connected using 100Gbit capable cables, preventing any potential bottlenecks in the connection. The illustration of this hardware setup is shown in fig. 2.1

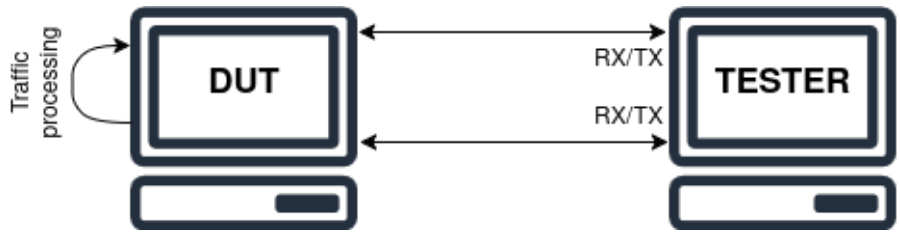

Figure 2.1 Picture showing hardware setup

The Device Under Test (DUT) is the network device being evaluated during testing. It is configured with a specific network stack and settings based on measurement scenario and serves as the focus of performance and behavior analysis in a controlled test environment. The DUT is responsible for processing network traffic and responding to the test conditions set by the measure-

[1]The hardware used in this testing setup was loaned free of charge for the purposes of this bachelor thesis by Silicon Hill club.

ment device. Additionally, the electrical power consumption of the DUT is monitored and measured during the tests to assess its energy efficiency under varying loads. The hardware of DUT is shown in table 2.1.

Hardware Component	DUT (Device Under Test)
CPU Model	2x Intel(R) Xeon(R) CPU E5-2660 v3
Frequency	2.60GHz
Cores	10 physical cores each (one thread per core)
Memory (RAM)	Size, type, speed
Network Interface Cards (NIC)	Mellanox ConnectX-6 Dx (Dual-port)

Table 2.1 Hardware details for DUT (Device Under Test)

The Tester (Measurement Device), on the other hand, is responsible in generating the network traffic and capturing the responses from the DUT. Its physical features are shown in table 2.2.

Hardware Component	Tester (Measurement Device)
CPU Model	2x Intel(R) Xeon(R) Gold 6136 CPU
Frequency	3.00GHz
Cores	12 physical cores each (two threads per core)
Memory (RAM)	Size, type, speed
Network Interface Cards (NIC)	2x Mellanox ConnectX-5

Table 2.2 Hardware details for Tester (Measurement Device)

The DUT is running Debian GNU/Linux 12 (Bookworm) x86_64 with Linux kernel version *6.1.0-32-amd64*, VPP v25.02-release, and DPDK version 24.11.1.

The tester is running ...

2.2 Metodology

The RFC 2544 recommends to test be at least 60 seconds in duration[18] and NAJÍT ZDROJ??? kolikrát opakovat Each test scenario executed using TRex was repeated 30 times, with each individual run lasting five minutes. The reported results represent the arithmetic mean of these 30 measurements. In cases where an anomalous spike or irregularity was observed in the results, the corresponding measurement was discarded and the test was repeated. All this steps shoudl ensure consistency and statistical reliability.

2.3 Test Scenarios & Results

2.3.1 Bidirectional UDP 1 Gbit/s of 64-bytes packets

In this scenario, the DUT is exposed to a bidirectional UDP traffic load of 1 Gbit/s combined (500+500 Mbit/s), consisting of 64-bytes packets, generated by TRex using the *udp_1pkt_src_ip_split.py* profile. This configuration ensures that each packet carries a unique source IP address, simulating multiple clients while maintaining a single destination per direction. The routing table of the DUT contains only two active forwarding entries, corresponding to the test routes, in addition to two administrative entries used for management. The aim of this test is to obse rve the behavior of the VPP forwarding plane under low traffic load of small packets and to evaluate its energy efficiency.

The chosen load of 1 Gbit/s is representative of a realistic aggregate traffic pattern that could be observed in a small or medium-sized enterprise network, especially when routed through a central gateway. The use of 64-byte packets represents a worst-case scenario in packet forwarding, as processing such small packets means to process a large amount non-payload data. These packets put increased stress on the processing path due to their higher packet-per-second rate for a given bandwidth, thereby providing a stringent test of the forwarding plane's efficiency

The DUT is configured with the Vector Packet Processing (VPP) stack, tested under three configurations using 1, 4, and 10 worker threads. The number of RX/TX queues is aligned with the number of active worker threads in each case to ensure balanced packet distribution and optimal performance. For each configuration, the same traffic pattern is replayed to measure how well the VPP-based router handles low traffic load under different degrees of parallelism.

As a baseline for comparison, the scenario is also executed using a standard Linux network stack, configured with similar routing and interface parameters. This enables a direct comparison between VPP and traditional kernel-based forwarding in terms of performance and power efficiency.

Configuration	Transmited packets	Transmited bytes	Watts used
VPP – 1 worker	17 578 124 988	1 124 999 999 232	848.08
VPP – 4 workers	17 578 124 986	1 124 999 999 104	951.03
VPP – 10 workers	17 578 124 988	1 124 999 999 232	1 193.56
Linux stack	17 578 124 978	1 124 999 998 592	1 257.25

Table 2.3 Result of Bidirectional UDP 1 Gbit/s of 870-bytes packets test

As the results in Table 2.3 show, the power consumption increases notably with the number of worker threads in the VPP stack. While all VPP configurations deliver identical packet and byte throughput, the most energy-efficient setup is this measurement is the single-worker variant, consuming roughly 25.4 kWh during the test. In contrast, the traditional Linux network stack

demonstrates the highest energy usage, despite handling the same volume of packets.

This discrepancy can likely be attributed to the cost of processing a high number of small packets in kernel space. Since the test uses fixed-size 64-byte packets, which are known to generate frequent system calls and context switches in Linux, the forwarding path becomes less efficient compared to VPP's user-space architecture, where such overheads are significantly reduced. The results highlight the energy cost of kernel-based packet forwarding in scenarios dominated by small-packet traffic.

2.3.2 Bidirectional UDP 1 Gbit/s of 870-bytes packets

In this scenario, the DUT is subjected to a bidirectional UDP traffic load of 1 Gbit/s (500+500 Mbit/s) composed of 870-byte packets, generated using a modified TRex *udp_1pkt_src_ip_split.py* profile. The packet size of 870 bytes is used becuase it was identified as the average size in real-world network traffic by Jurkiewicz et al. [19].

This test complements the previous one by simulating a more typical traffic pattern, as opposed to the stress scenario represented by minimal 64-byte packets. Larger packets result in a lower packet-per-second (PPS) rate for the same bandwidth, thus reducing per-packet processing overhead and more closely reflecting actual router workloads.

The DUT is again configured with the VPP stack and evaluated under three different worker thread configurations (1, 4, and 10), with queue allocation matching the thread count. The Linux kernel-based router is also included in the comparison under equivalent conditions.

This scenario provides insight into forwarding and energy efficiency under more realistic conditions, allowing for better interpretation of the DUT's performance in practical deployments.

Configuration	Transmited packets	Transmited bytes	Watts used
VPP – 1 worker	1 293 103 500	1 125 000 045 000	823.20
VPP – 4 workers	1 293 103 500	1 125 000 045 000	972.49
VPP – 10 workers	1 293 103 500	1 125 000 045 000	1 170.26
Linux stack	1 293 103 500	1 125 000 045 000	946.28

Table 2.4 Result of Bidirectional UDP 1 Gbit/s of 870-bytes packets test

As shown in table 2.4, the second test with 870-byte packets, VPP's power consumption remains relatively stable across different worker thread configurations, with a variation of only about ±5% compared to previous measurement. This indicates that VPP's performance and power consumption are consistent, regardless of the increased packet size. On the other hand, the Linux stack's

power consumption shows a positive reduction compared to the previous test, which is attributed to the lower number of packets being processed. Since larger packets reduce the packet-per-second (PPS) rate, the system overhead and processing cost in Linux are lower, leading to more efficient energy usage in this scenario.

2.4 Presentation and Analysis of Results

Chapter 3

Conclusion

Appendix A
Nějaká příloha

Sem přijde to, co nepatří do hlavní části.

Bibliography

1. GALLATIN, Andrew J.; CHASE, Jeffrey S.; YOCUM, Kenneth G. Trapeze/IP: TCP/IP at Near-Gigabit Speeds. In: *Proceedings of the USENIX Annual Technical Conference*. 1999, pp. 109–120. Available also from: https://www.usenix.org/event/usenix99/full_papers/gallatin/gallatin.pdf.
2. COX, Alan L.; SCHAELICKE, Lambert; DAVIS, Al; MCKEE, Sally A. Profiling I/O Interrupts in Modern Architectures. *Proceedings of the Workshop on Performance Analysis and Its Impact on Design*. 2000. Available also from: https://users.cs.utah.edu/~ald/pubs/interrupts.pdf.
3. FD.IO. *What is VPP?* [https://wiki.fd.io/view/VPP/What_is_VPP%3F]. 2025. Accessed: 2025-04-07.
4. LINGUAGLOSSA, Leonardo; ROSSI, Dario; PONTARELLI, Salvatore; BARACH, Dave; MARJON, Damjan; PFISTER, Pierre. High-speed data plane and network functions virtualization by vectorizing packet processing. *Computer Networks*. 2019, vol. 149, pp. 187–199. ISSN 1389-1286. Available from DOI: https://doi.org/10.1016/j.comnet.2018.11.033.
5. BARACH, David; LINGUAGLOSSA, Leonardo; MARION, Damjan; PFISTER, Pierre; PONTARELLI, Salvatore; ROSSI, Dario. High-speed Software Data Plane via Vectorized Packet Processing. *IEEE Communication Magazine* [https://perso.telecom-paristech.fr/drossi/paper/rossi18commag.pdf]. 2018, vol. 56, no. 12, pp. 97–103. ISSN 0163-6804. Available from DOI: 10.1109/MCOM.2018.1800069.
6. FD.IO. *Extensible: VPP and its plugin architecture* [https://fd.io/docs/vpp/v2101/whatisvpp/extensible]. 2021. Accessed: 2025-04-10.
7. DPDK PROJECT. *About DPDK* [https://www.dpdk.org/about/]. 2025. Accessed: 2025-04-13.

8. FREITAS, Eduardo; DE OLIVEIRA FILHO, Assis T.; DO CARMO, Pedro R.X.; SADOK, Djamel; KELNER, Judith. A survey on accelerating technologies for fast network packet processing in Linux environments. *Computer Communications*. 2022, vol. 196, pp. 148–166. ISSN 0140-3664. Available from DOI: https://doi.org/10.1016/j.comcom.2022.10.003.
9. FD.IO PROJECT. *VPP Feature List – FD.io Documentation (Release 25.02)*. 2025. Available also from: https://s3-docs.fd.io/vpp/25.02/aboutvpp/featurelist.html. Accessed: 2025-04-15.
10. FD.IO PROJECT. *FD.io VPP: High Performance, Modular, and Production Quality Software Forwarder*. 2017-07. Tech. rep. FD.io. Available also from: https://fd.io/docs/whitepapers/FDioVPPwhitepaperJuly2017.pdf. Accessed: 2025-04-16.
11. SLAVIC, Goran; KRAJNOVIC, Nenad. Practical implementation of the vector packet processing software router. In: *2024 32nd Telecommunications Forum (TELFOR)*. 2024, pp. 1–4. Available from DOI: 10.1109/TELFOR63250.2024.10819057.
12. ADELEKE, Oluwamayowa Ade; BASTIN, Nicholas; GURKAN, Deniz. Network Traffic Generation: A Survey and Methodology. *ACM Comput. Surv.* 2022, vol. 55, no. 2. ISSN 0360-0300. Available from DOI: 10.1145/3488375.
13. TEAM, The iPerf. *iPerf - The Network Testing Tool*. 2025. Available also from: https://iperf.fr/. Accessed: 2025-04-16.
14. COMPUTER NETWORKING GROUP, Università degli Studi di Napoli Federico II. *D-ITG: Distributed Internet Traffic Generator Manual*. 2013. Available also from: https://traffic.comics.unina.it/software/ITG/manual/. Accessed: 2025-04-16.
15. CISCO SYSTEMS, Inc. *TRex - Traffic Generator*. 2025. Available also from: https://trex-tgn.cisco.com/. Accessed: 2025-04-16.
16. PROJECT, DPDK. *Pktgen-DPDK* [https://pktgen-dpdk.readthedocs.io/]. 2025. Accessed: 2025-04-16.
17. ALLAN, David; MARTINEZ, Jordi Palet. *Benchmarking Methodology for IPv6 Transition Technologies* [https://datatracker.ietf.org/doc/html/rfc8219]. 2017. RFC 8219.
18. *Benchmarking Methodology for Network Interconnect Devices* [RFC 2544]. RFC Editor, 1999. Request for Comments, no. 2544. Available from DOI: 10.17487/RFC2544.
19. JURKIEWICZ, Piotr; RZYM, Grzegorz; BORYŁO, Piotr. Flow length and size distributions in campus Internet traffic. *Computer Communications*. 2021, vol. 167, pp. 15–30. ISSN 0140-3664. Available from DOI: https://doi.org/10.1016/j.comcom.2020.12.016.

Obsah příloh