Sistemasi: Jurnal Sistem Informasi
Volume 15, Nomor 2, 2026: 574-582
ISSN:2302-8149
e-ISSN:2540-9719

Analisis Kinerja Jaringan *Service Chaining* NFV berbasis *Docker* pada Lingkungan *Single-Host*

Performance Analysis of Docker-based NFV Service Chaining Networks in a Single-Host Environment

[1]Rayhan Ziqrul Fitroh*, [2]Ichwan Nul Ichsan
[1,2]Program Studi Sistem Telekomunikasi, Universitas Pendidikan Indonesia
[1,2]Jl. Veteran No. 8, Purwakarta, 41115, Indonesia
*e-mail: ryhanfit@upi.edu

(***received*:** 30 November 2025, ***revised*:** 20 January 2026, ***accepted*:** 25 January 2026)

Abstrak

Network Function Virtualization (NFV) dan *Service Function Chaining* (SFC) memungkinkan fungsi jaringan dijalankan sebagai *Virtual Network Function* (VNF) di atas server umum yang fleksibel. Namun, penambahan beberapa VNF dalam satu rantai layanan berpotensi menurunkan kinerja jalur data, terutama pada lingkungan berbasis kontainer. Penelitian ini menganalisis kinerja SFC berbasis kontainer pada lingkungan Docker *single-host* melalui tiga skenario, yaitu koneksi langsung klien–server tanpa VNF (*baseline*), penambahan satu VNF L3 berupa *firewall* iptables, serta penambahan kombinasi VNF L3 *firewall* dan VNF L4 *load balancer* berbasis HAProxy. Evaluasi dilakukan dengan mengukur *throughput* TCP menggunakan iperf3, latensi *end-to-end* menggunakan ping, serta penggunaan CPU setiap kontainer menggunakan docker stats. Hasil menunjukkan bahwa penambahan *firewall* L3 menurunkan *throughput* sekitar 33% dan hampir menggandakan latensi dibandingkan *baseline*, sedangkan penambahan *load balancer* L4 menyebabkan penurunan *throughput* hingga sekitar 92%. Pengamatan penggunaan CPU memperlihatkan bahwa *firewall kernel-space* hampir tidak menambah beban di *user-space*, sementara VNF L4 menjadi sumber utama saturasi CPU. Temuan ini mengindikasikan bahwa pada SFC berbasis kontainer di lingkungan Docker *single-host*, batas kinerja utama lebih banyak ditentukan oleh VNF L4 di *user-space* dibandingkan jalur *forwarding* L3 di kernel, sehingga perlu dipertimbangkan secara khusus dalam perancangan *service chaining* pada *node edge* dengan sumber daya terbatas.

Kata kunci: docker, kinerja jaringan, *network function virtualization*, *service function chaining*, *single-host*

Abstract

Network Function Virtualization (NFV) and Service Function Chaining (SFC) enable network functions to be deployed as Virtual Network Functions (VNFs) on flexible commodity servers. However, chaining multiple VNFs within a service chain may degrade data-plane performance, particularly in container-based environments. This study analyzes the performance of container-based SFC in a single-host Docker environment under three scenarios: (1) a direct client–server connection without VNFs (baseline), (2) the addition of a single Layer 3 (L3) VNF in the form of an iptables firewall, and (3) the integration of an L3 firewall VNF combined with a Layer 4 (L4) load balancer VNF based on HAProxy. Performance evaluation was conducted by measuring TCP throughput using iperf3, *end-to-end latency using* ping, *and CPU utilization of each container using* docker stats. *The results indicate that adding the L3 firewall reduces throughput by approximately 33% and nearly doubles latency compared to the baseline. Meanwhile, incorporating the L4 load balancer causes throughput degradation of up to 92%. CPU utilization analysis shows that the kernel-space firewall introduces minimal additional overhead in user space, whereas the L4 VNF becomes the primary source of CPU saturation. These findings suggest that, in container-based SFC deployments on a single-host Docker environment, performance bottlenecks are primarily driven by user-space L4 VNFs rather than kernel-based L3 forwarding. Therefore, L4 VNFs require special consideration when designing service chaining architectures for resource-constrained edge nodes.*

 ISSN:2302-8149
e-ISSN:2540-9719

Keywords: *docker, network function virtualization, network performance, service function chaining, single-host*

1 Pendahuluan

Dengan kemajuan teknologi yang cepat, kebutuhan akan jaringan yang adaptif dan efektif semakin bertambah. Untuk memenuhi kebutuhan ini, arsitektur jaringan masa kini beralih ke metode baru yang lebih efisien. Dimana arsitektur yang sedang banyak digunakan saat ini adalah *Network Function Virtualization* (NFV). NFV dapat memisahkan fungsi jaringan pada perangkat keras dan Menjalankannya pada server standar, sehingga memberikan pengelolaan yang fleksibel dan dinamis [1]. Arsitektur NFV biasanya menggabungkan beberapa *Virtual Network Function* (VNF) untuk membentuk alur pemrosesan paket kompleks yang dikenal sebagai *Service Function Chaining* (SFC) [1]. Dalam kerangka NFV yang diusulkan ETSI, infrastruktur virtualisasi, VNF, dan lapisan *management and orchestration* (MANO) bekerja bersama untuk mengelola siklus hidup VNF dan memetakan SFC ke sumber daya fisik yang tersedia [2].

Dalam konteks kebutuhan layanan, muncul berbagai aplikasi dengan kebutuhan latensi *end to end* yang sangat rendah, seperti *Multi-Access Edge Computing* (MEC) dan *cloud gaming*. Untuk memenuhi kebutuhan tersebut, beragam penelitian mengajukan optimasi SFC melalui pemilihan jalur dan penempatan VNF yang lebih cerdas, baik di lingkungan MEC maupun IoT, serta pada komunikasi dengan batas latensi yang ketat [3-6]. Di saat yang sama, penggunaan kontainer sebagai platform virtualisasi fungsi jaringan semakin meluas karena dianggap lebih ringan dibandingkan *virtual machine* dan lebih sesuai untuk skenario *edge* maupun perangkat dengan sumber daya terbatas [7-10].

Meskipun berbagai studi telah membahas arsitektur NFV, optimasi SFC, pemilihan platform virtualisasi, dan orkestrasi kontainer pada beberapa *host* , karakteristik kinerja rantai layanan berbasis VNF kontainer pada satu *host* masih belum banyak dikaji secara rinci. Secara khusus, masih diperlukan pemahaman yang lebih jelas mengenai bagaimana penambahan fungsi jaringan pada *layer 3* (L3) dan *layer 4* (L4) memengaruhi *throughput*, latensi, dan pola penggunaan CPU, serta di mana *bottleneck* pemrosesan paket cenderung muncul (*kernel-space* atau *user-space*).

Penelitian ini bertujuan untuk menganalisis kinerja SFC berbasis kontainer pada lingkungan Docker *single-host* melalui tiga scenario, yaitu koneksi langsung tanpa VNF, penambahan satu VNF L3 berupa *firewall* iptables, dan penambahan kombinasi VNF L3 *firewall* dengan VNF L4 *load balancer* berbasis HAProxy. Analisis dilakukan dengan mengukur *throughput* TCP, latensi *end-to-end*, dan penggunaan CPU setiap kontainer, sehingga diperoleh gambaran empiris mengenai dampak penambahan VNF L3 dan L4 terhadap jalur data serta pergeseran *bottleneck* dalam satu *host*.

2 Tinjauan Literatur

Tinjauan literatur pada penelitian ini mencakup tiga aspek utama, yaitu arsitektur NFV dan *service function chaining* (SFC), platform virtualisasi untuk implementasi fungsi jaringan, serta studi kinerja *containerization* dan komunikasi *intra-host* yang relevan dengan pengujian pada satu *host*.

NFV memungkinkan pemisahan fungsi jaringan dari perangkat keras khusus dan menjalankannya sebagai *virtual network function* (VNF) yang berjalan di atas infrastruktur komputasi umum [1]. Lapisan *management and orchestration* (MANO) yang terdiri atas VIM, VNFM, dan NFVO berperan mengelola siklus hidup VNF dan memetakan SFC ke sumber daya fisik [2]. Berbagai penelitian mengusulkan mekanisme pengelolaan SFC untuk memenuhi kebutuhan layanan dengan latensi rendah, antara lain melalui pendekatan *mean-field game* dan *reinforcement learning* pada lingkungan MEC [3] serta skema adaptif berbasis *deep reinforcement learning* untuk mempertahankan kualitas layanan lalu lintas *real-time* pada IoT [4]. Selain itu, SFC juga dikaji sebagai mekanisme untuk mendukung komunikasi *ultra-low latency* melalui penempatan VNF dan pemilihan jalur yang ketat terhadap batas latensi [5], maupun sebagai solusi orkestrasi lintas domain berbasis SRv6 pada infrastruktur multi-domain [6].

Dari sisi platform, sejumlah penelitian membandingkan *virtual machine* dan kontainer sebagai landasan implementasi NFV dan menunjukkan bahwa kontainer cenderung memiliki *overhead* lebih rendah dan lebih sesuai untuk skenario *edge* serta perangkat dengan sumber daya terbatas [7-10]. Pendekatan berbasis kontainer digunakan untuk memvirtualisasi *node* LoRaWAN dan layanan

 ISSN:2302-8149
e-ISSN:2540-9719

jaringan lain, sehingga mempermudah pengelolaan dan evaluasi performa pada perangkat kecil maupun lingkungan terdistribusi [7], [8]. Studi lain mengevaluasi *containerization* dalam tumpukan *edge–cloud* untuk aplikasi industri dan menyoroti *trade-off* antara fleksibilitas orkestrasi, pemanfaatan sumber daya, dan kinerja *data plane* [11]. Di lingkungan *edge computing*, kinerja berbagai alat orkestrasi kontainer juga dibandingkan untuk menilai kemampuannya dalam menangani dinamika beban dan skala layanan [12].

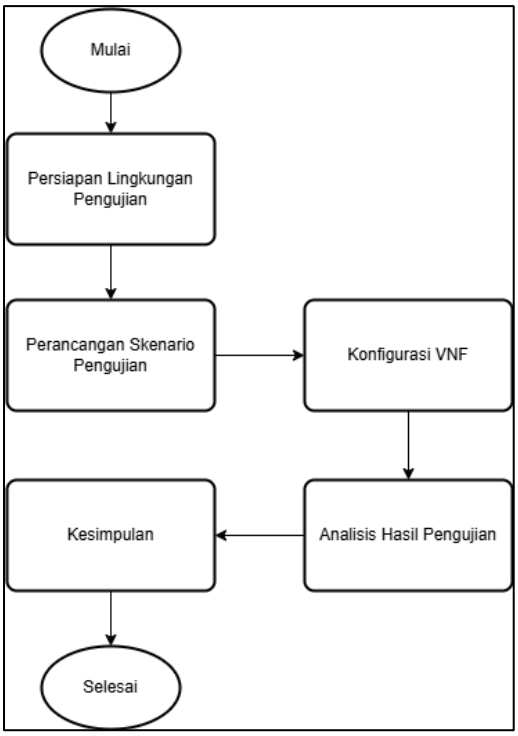
Pada tingkat orkestrasi NFV, beberapa solusi mengintegrasikan MANO dengan *container orchestrator* untuk mengoptimalkan penempatan dan alokasi sumber daya. HADES, misalnya, menggabungkan NFV dengan Kubernetes untuk penempatan VNF yang hemat energi dan adaptif pada infrastruktur heterogen [13]. Di sisi lain, *traffic-aware horizontal pod autoscaler* dikembangkan untuk menyesuaikan jumlah *pod* layanan dengan variasi lalu lintas pada infrastruktur *edge* berbasis Kubernetes [14]. Pendekatan-pendekatan ini berfokus pada bagaimana VNF ditempatkan dan diskalakan pada beberapa *host*, bukan pada karakteristik jalur data rinci di dalam satu *host*.

Selain aspek penempatan dan orkestrasi, komunikasi *intra-host* antar kontainer juga dapat menjadi faktor pembatas kinerja. Su et al. mengusulkan mekanisme komunikasi *intra-host* dengan *hardware offloading* untuk menurunkan *overhead* pemrosesan dan meningkatkan *throughput* antar-kontainer [15]. Hasil tersebut menunjukkan bahwa meskipun hanya melibatkan satu *host*, jalur data antar-kontainer tetap berpotensi menjadi *bottleneck* yang perlu dioptimalkan.

Secara umum, penelitian-penelitian terdahulu telah banyak membahas arsitektur NFV, optimasi SFC, pemilihan platform virtualisasi, dan orkestrasi kontainer pada skala *multi-host* [1]–[15]. Namun, karakterisasi kinerja rantai layanan berbasis VNF kontainer pada satu *host*, khususnya kombinasi fungsi L3 dan L4 serta pergeseran *bottleneck* antara *kernel-space* dan *user-space*, masih relatif jarang dikaji secara empiris. Celah ini menjadi fokus utama penelitian yang dilakukan.

3 Metode Penelitian

Penelitian ini menggunakan metode eksperimental kuantitatif untuk mengukur parameter kinerja jaringan (*throughput* dan latensi) serta penggunaan sumber daya komputasi (*CPU utilization*) dalam lingkungan terkontrol. Alur penelitian disajikan pada Gambar 1.



Gambar 1 Alur penelitian

 ISSN:2302-8149
e-ISSN:2540-9719

3.1 Lingkungan Pengujian

Seluruh eksperimen dilakukan di dalam satu *Virtual Machine* (VM) Ubuntu 22.04 LTS dengan spesifikasi 2 Core CPU dan 4 GB RAM yang berjalan di atas VMware Workstation. Perangkat lunak utama yang digunakan adalah Docker Engine versi 29.0.0 untuk menjalankan kontainer dan Docker Compose versi 2.40.3 untuk mendefinisikan dan merakit arsitektur layanan.

3.2 Skenario Pengujian

Tiga skenario dirancang untuk mengisolasi dampak penambahan VNF terhadap kinerja jaringan. skenario 1 berfungsi sebagai tolak ukur kinerja maksimal, sedangkan skenario 2 dan 3 berfungsi sebagai perbandingan ketika VNF ditambahkan pada jaringan. Skenario dan alur disajikan pada Tabel 1.

Tabel 1 Rancangan skenario *service chaining*

Skenario	Alur
1. *Baseline*	Koneksi langsung antara *Client* dan *Backend* dalam satu jaringan *bridge* tanpa VNF.
2. 1-VNF (*Firewall*)	Lalu lintas dari *Client* dipaksa melewati VNF *Firewall* sebelum mencapai *Backend*.
3. 2-VNF (*Firewall* + *Load Balancer*)	Lalu lintas melewati rantai layanan lengkap, *Client* melewati VNF *Firewall*, kemudian melewati VNF *Load Balancer* dan sampai ke *Backend*

3.3 Konfigurasi VNF

Membuat 2 konfigurasi VNF, yaitu VNF *Firewall* dan VNF *Load Balancer*. VNF *Firewall* Menggunakan iptables untuk fungsi *Network Address Translation* (NAT), fungsi ini beroperasi di *kernel-space*. VNF *Load Balancer* Menggunakan HAProxy (mode TCP L4) yang beroperasi di *user-space*.

3.4 Pengujian dan Pengumpulan Data

Setelah skenario dan konfigurasi dibuat, dilakukan pengujian skenario untuk mengumpulkan data dilakukan dengan menggunakan alat standar yang disajikan pada Tabel 2. Parameter evaluasi yang digunakan adalah *throughput* TCP (Gbits/s) yang diperoleh dari ringkasan akhir iperf3 sebagai rata-rata laju transfer selama durasi pengujian, latensi *end-to-end* (ms) yang direpresentasikan oleh nilai RTT rata-rata dari ping dan penggunaan CPU kontainer (%) yang diamati menggunakan docker stats selama iperf3 berjalan. Untuk meningkatkan reliabilitas, setiap skenario dijalankan sebanyak 3 kali selama 30 detik dan nilai yang dilaporkan merupakan rata-rata hasil pengujian.

Parameter evaluasi dihitung untuk mengkuantifikasi dampak VNF terhadap kinerja jaringan. Persentase penurunan *throughput* (D_{th}) dan kenaikan latensi (K_{lat}) dihitung menggunakan Persamaan (1) dan (2):

$$D_{th} = \left(\frac{T_{base} - T_{vnf}}{T_{base}}\right) \times 100\% \quad (1)$$

$$K_{lat} = \left(\frac{L_{base} - L_{vnf}}{L_{base}}\right) \times 100\% \quad (2)$$

Dimana T_{base} dan L_{base} adalah nilai *throughput* dan latensi pada skenario *baseline*, sedangkan L_{base} dan L_{vnf} adalah nilai pada skenario penambahan VNF.

Tabel 2 Parameter dan instrumen pengujian

Parameter	Satuan Ukur	Alat Ukur	Fungsi
Throughput	Gbits/s	Iperf3	Mengukur seberapa banyak data yang bisa dikirim melalui jaringan dalam satu waktu.
Latensi	ms	Ping (ICMP)	Mengukur *delay* yang dibutuhkan paket data untuk pergi ke tujuan dan kembali lagi (RTT).

 ISSN:2302-8149
e-ISSN:2540-9719

Utilisasi CPU	%	Docker Stats	Mengukur seberapa besar sumber daya (CPU/RAM) yang sedang digunakan oleh kontainer.

3.5 Analisis Hasil

Pada tahap ini, data dari *throughput* dan latensi pada skenario 2 dan 3 dibandingkan terhadap skenario 1 untuk mengetahui penurunan dari *overhead* kinerja jaringan. Analisis *bottleneck* CPU dilakukan dengan membandingkan utilisasi CPU VNF L3 (Skenario 2) dan VNF L4 (Skenario 3) untuk mengidentifikasi pergeseran beban pemrosesan dari *kernel-space* ke *user-space*.

4 Hasil dan Pembahasan

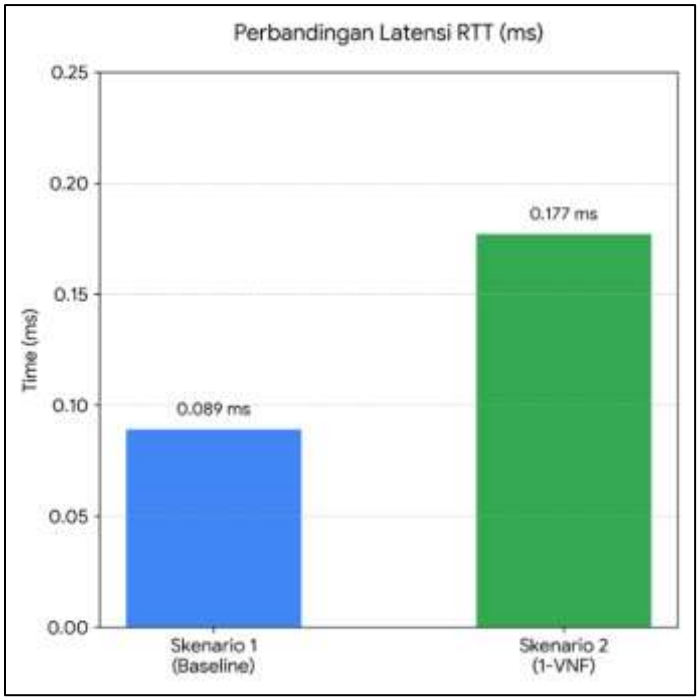
Bagian ini memaparkan data hasil pengujian dari ketiga skenario dan menganalisis fenomena jaringan yang terjadi berdasarkan data yang diperoleh.

4.1 Hasil Pengujian Latensi

Pengujian latensi dilakukan untuk mengukur *delay* tambahan yang diberikan dari setiap penambahan VNF dalam *service chaining*. Tabel 3 dan Gambar 2 merangkum hasil pengukuran rata-rata. Pengukuran latensi *end-to-end* menggunakan ICMP (Ping) tidak dapat diterapkan (N/A) dikarenakan karakteristik VNF *Load Balancer* (HAProxy) yang bekerja sebagai *reverse proxy* pada Layer 4, yang secara efektif memutus jalur ICMP langsung antara klien dan *backend*.

Tabel 3 Hasil pengukuran latensi rata-rata

Skenario	Latensi Rata-rata (ms)	Kenaikan (%)
Baseline	0.089	-
1-VNF	0.177	98.8
2-VNF	N/A	N/A



Gambar 2 Perbandingan latensi rata-rata

Seperti terlihat pada Gambar 2, penambahan satu hop berupa VNF *Firewall* menyebabkan peningkatan latensi hampir dua kali lipat (dari 0.089 ms menjadi 0.177 ms). Hal ini menunjukkan bahwa meskipun pemrosesan di kernel cepat, penambahan antrian jaringan virtual tetap memberikan *overhead* latensi yang nyata.

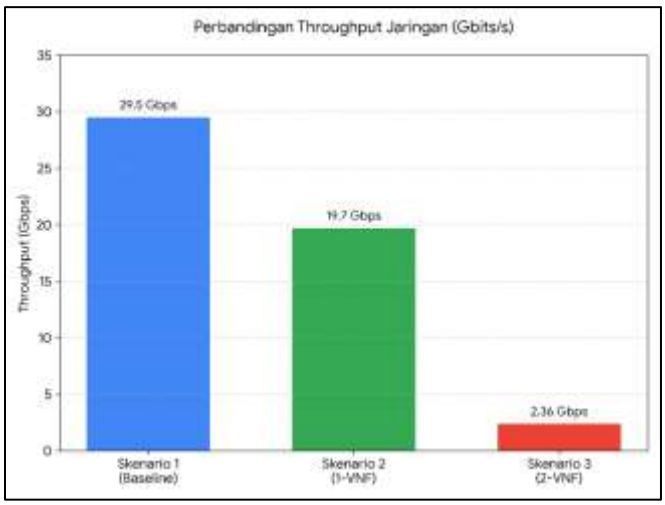
 ISSN:2302-8149
e-ISSN:2540-9719

4.2 Hasil Pengujian Throughput

Pengujian *throughput* memberikan gambaran paling jelas mengenai degradasi kinerja kapasitas jaringan. Data hasil pengukuran disajikan pada Tabel 4 dan divisualisasikan pada Gambar 3.

Tabel 4 Hasil pengujian *throughput*

Skenario	Throughput (Gbits/s)	Penurunan (%)
Baseline	29.5	-
1-VNF	19.7	33.2
2-VNF	2.36	92



Gambar 3 Degradasi *throughput* jaringan pada tiga skenario

Hasil pada Gambar 3 menunjukkan penurunan performa yang drastis. Pada Skenario 2, *throughput* turun menjadi 19.7 Gbps akibat *overhead routing* dan NAT. Namun, penurunan paling ekstrem terjadi pada Skenario 3, di mana *throughput* anjlok hingga 2.36 Gbps. Ini menandakan adanya *bottleneck* yang parah pada komponen tambahan di skenario ketiga, yaitu *Load Balancer*.

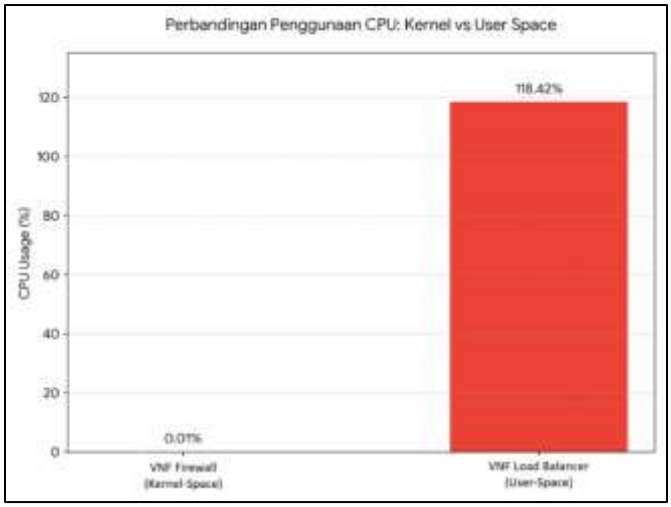
4.3 Analisis Penggunaan Sumber Daya

Untuk memahami penyebab jatuhnya *throughput* pada Skenario 3, dilakukan analisis penggunaan CPU saat beban puncak. Data penggunaan CPU dirangkum dalam Tabel 5 dan Gambar 4.

Tabel 5 Penggunaan CPU

Tipe VNF	Proses Utama	Penggunaan CPU (%)
Firewall	Kernel	~0.01
Load Balancer	User (HAProxy)	~118.42

 ISSN:2302-8149
e-ISSN:2540-9719



Gambar 4 Perbandingan penggunaan cpu pada *kernel-space* dan *user-space*

Data pada Gambar 4 mengungkap fenomena menarik. VNF *Firewall* hampir tidak membebani CPU kontainer (0.01%) meskipun menangani Lalu lintas 19.7 Gbps. Hal ini karena fungsi NAT ditangani langsung oleh kernel *host*, memvalidasi temuan Khalilnasl et al. [7] tentang efisiensi modul kernel.

Sebaliknya, VNF *Load Balancer* mengalami saturasi CPU hingga 118.42% (menggunakan lebih dari 1 core penuh) hanya untuk menangani lalu lintas 2.36 Gbps. HAProxy bekerja di *user-space*, yang mengharuskan setiap paket data disalin dari kernel ke memori aplikasi, diproses, lalu disalin kembali ke kernel. Su et al. [15] menjelaskan bahwa operasi penyalinan memori dan pemrosesan tumpukan TCP/IP di *user-space* adalah penyebab utama tingginya *overhead* ini.

4.4 Pembahasan

Seperti yang ditunjukkan oleh hasil pengukuran, penambahan VNF ke dalam *service chaining* NFV di lingkungan Docker *single-host* secara signifikan memengaruhi kinerja dan latensi *end to end*. Dalam skenario 1 tanpa VNF, *throughput* mencapai sekitar 29,5 Gbps dan latensi rata-rata di bawah 0,1 ms. Hasil ini mencerminkan kapasitas jalur data ketika lalu lintas hanya melewati soket aplikasi dan tumpukan jaringan inti, tanpa fungsi jaringan tambahan.

Pada skenario 2, penambahan VNF tipe *firewall* menyebabkan kinerja *throughput* turun sekitar sepertiga dan rata-rata latensi meningkat hampir dua kali lipat. Penurunan ini disebabkan karena setiap paket harus melewati tambahan satu *hop* dan diproses oleh mekanisme NAT di jalur *forwarding*. Namun, *throughput* tetap berada di puluhan Gbits/s dan beban tambahan pemrosesan L3 di kernel satu VNF tetap dapat diterima untuk VNF. Dalam praktiknya, kecuali jika persyaratan lalu lintas data sangat ketat, penambahan lapisan keamanan *firewall* L3 tetap efektif.

Perubahan yang jauh lebih besar muncul ketika VNF L4 *load balancer* berbasis HAProxy ditambahkan ke dalam rantai. *Throughput* turun hingga sekitar 2,36 Gbit/s, atau hanya sekitar 8% dari nilai *baseline*. Pada skenario ini, setiap aliran TCP tidak hanya diteruskan, tetapi juga diakhiri dan dibentuk ulang di *user-space* oleh proses HAProxy. Transisi berulang antara *user-space* dan *kernel-space*, pengelolaan soket ganda, dan logika pemilihan *backend* menimbulkan beban CPU yang tinggi pada kontainer *load balancer*. Kondisi ini menunjukkan bahwa kapasitas pemrosesan VNF L4 menjadi faktor pembatas utama, meskipun jalur L3 di kernel masih mampu menangani lalu lintas yang lebih besar.

Pengamatan terhadap penggunaan CPU pada setiap kontainer memperjelas pola tersebut. Pada skenario dengan satu *firewall* L3, kontainer *firewall* hampir tidak menunjukkan peningkatan CPU yang berarti, sementara kontainer *client* dan *backend* mendominasi pemakaian CPU selama pengujian. Hal ini konsisten dengan cara kerja iptables yang beroperasi di *kernel-space*, yaitu setelah aturan dimuat, pemrosesan paket berlangsung di jalur kernel tanpa banyak aktivitas tambahan di proses *user-space*. Sebaliknya, pada skenario dengan penambahan *load balancer* L4, kontainer HAProxy menjadi komponen dengan penggunaan CPU tertinggi dan mendekati saturasi, sedangkan kontainer lain relatif

 ISSN:2302-8149
e-ISSN:2540-9719

lebih rendah. Perbedaan ini mengindikasikan adanya pergeseran *bottleneck* dari jalur forwarding L3 di *kernel-space* menuju fungsi L4 di *user-space*.

Temuan ini sejalan dengan kecenderungan umum dalam penelitian NFV dan kontainerisasi, di mana layanan yang banyak memproses lalu lintas di *user-space* cenderung menjadi titik jenuh ketika beban meningkat, sementara fungsi yang lebih dekat dengan jalur kernel dapat menangani lalu lintas lebih besar dengan *overhead* yang lebih rendah [7], [11], [15]. Perbedaannya, penelitian ini menyoroti fenomena tersebut secara spesifik pada SFC NFV *single-host* dengan kombinasi *firewall* L3 dan *load balancer* L4, sehingga pola *bottleneck* dapat diamati dengan lebih jelas. Hasil ini melengkapi studi-studi sebelumnya yang lebih banyak berfokus pada perbandingan platform (VM vs kontainer) atau orkestrasi *multi-host*, dengan memberikan gambaran rinci tentang bagaimana satu *host* saja dapat mencapai batasnya ketika *service chaining* disusun dari fungsi L4 yang berat.

Dari sudut pandang perancangan sistem, pembahasan di atas menunjukkan bahwa komposisi rantai layanan menjadi faktor yang tidak kalah penting dibandingkan pemilihan platform virtualisasi. Rantai yang didominasi oleh fungsi L4 *user-space* berisiko mengalami penurunan *throughput* yang tajam meskipun infrastruktur fisik masih cukup kuat, sedangkan penambahan beberapa fungsi L3 *kernel-space* cenderung memberikan kompromi kinerja yang lebih moderat. Oleh karena itu, ketika SFC NFV berbasis docker *single-host* diimplementasikan pada *node edge* dengan sumber daya terbatas, penyusunan rantai layanan perlu mempertimbangkan keseimbangan antara kebutuhan fungsional, beban pemrosesan di *user-space*, dan target kinerja yang ingin dicapai.

5 Kesimpulan

Penelitian ini memberikan gambaran konkret tentang perilaku SFC NFV pada skenario *single-host*, di mana penambahan VNF L3 dan VNF L4 memiliki dampak yang berbeda terhadap kinerja sistem. VNF L3 berbasis *firewall* masih menurunkan batas kinerja yang bisa ditoleransi, sedangkan VNF L4 seperti *load balancer* mendorong sistem mencapai batas kapasitas dan menggeser *bottleneck* dari jalur forwarding di *kernel-space* ke proses di *user-space*, sehingga fungsi L4 menjadi faktor utama seberapa jauh rantai layanan dapat diperpanjang. Secara praktis, temuan ini menunjukkan bahwa rantai layanan yang melibatkan VNF L4 berat di *user-space* perlu dirancang dengan hati-hati karena berisiko menurunkan *throughput* secara signifikan, sedangkan penambahan fungsi L3 seperti *firewall* iptables masih relatif dapat diterima untuk memperkuat kebijakan keamanan selama kompromi kinerja tetap dalam batas yang diinginkan.

Referensi

[1] H. U. Adoga and D. P. Pezaros, "*Network Function Virtualization and Service Function Chaining Frameworks: A Comprehensive Review of Requirements, Objectives, Implementations, and Open Research Challenges*," *Future Internet*, Vol. 14, No. 2, Feb. 2022, DOI: 10.3390/fi14020059.

[2] K. Kaur, V. Mangat, and K. Kumar, "*A Review on Virtualized Infrastructure Managers with Management and Orchestration Features in NFV Architecture*," Nov. 09, 2022, *Elsevier B.V.* DOI: 10.1016/j.comnet.2022.109281.

[3] A. Abouaomar, S. Cherkaoui, Z. Mlika, and A. Kobbane, "*Service Function Chaining in MEC: A Mean-Field Game and Reinforcement Learning Approach*," *IEEE Syst J*, Vol. 16, No. 4, pp. 5357–5368, 2022, DOI: 10.1109/JSYST.2022.3171232.

[4] P. Tam, S. Math, and S. Kim, "*Priority-Aware Resource Management for Adaptive Service Function Chaining in Real-Time Intelligent IoT Services*," *Electronics (Switzerland)*, Vol. 11, No. 19, Oct. 2022, DOI: 10.3390/electronics11192976.

[5] M. M. Erbati, M. M. Tajiki, and G. Schiele, "*Service Function Chaining to Support Ultra-Low Latency Communication in NFV †*," *Electronics (Switzerland)*, Vol. 12, No. 18, Sep. 2023, DOI: 10.3390/electronics12183843.

[6] Y. Wu and J. Zhou, "*Dynamic Service Function Chaining Orchestration in a Multi-Domain: A Heuristic Approach based on SRv6*," *Sensors*, Vol. 21, No. 19, Oct. 2021, DOI: 10.3390/s21196563.

 ISSN:2302-8149
e-ISSN:2540-9719

[7] H. Khalilnasl, P. Ferrari, A. Flammini, and E. Sisinni, "*On the Use of Containers for LoRaWAN Node Virtualization: Practice and Performance Evaluation*," *Electronics (Switzerland)*, Vol. 14, No. 8, Apr. 2025, DOI: 10.3390/electronics14081568.

[8] C. Tipantuña, A. Yazán, and J. Carvajal-Rodriguez, "*Containers-based Network Services Deployment: A Practical Approach*," *Enfoque UTE*, Vol. 15, No. 1, pp. 36–44, Jan. 2024, DOI: 10.29019/enfoqueute.1005.

[9] H. Sturley, A. Fournier, A. Salcedo-Navarro, M. Garcia-Pineda, and J. Segura-Garcia, "*Virtualization vs. Containerization, a Comparative Approach for Application Deployment in the Computing Continuum Focused on the Edge*," *Future Internet*, Vol. 16, No. 11, Nov. 2024, DOI: 10.3390/fi16110427.

[10] S. Zahoor, I. Ahmad, A. U. Rehman, E. T. Eldin, N. A. Ghamry, and M. Shafiq, "*Performance Evaluation of Virtualization Methodologies to Facilitate NFV Deployment*," *Computers, Materials and Continua*, Vol. 75, No. 1, pp. 311–329, 2023, DOI: 10.32604/cmc.2023.035960.

[11] Y. Liu, D. Lan, Z. Pang, M. Karlsson, and S. Gong, "*Performance Evaluation of Containerization in Edge-Cloud Computing Stacks for Industrial Applications: A Client Perspective*," *IEEE Open Journal of the Industrial Electronics Society*, Vol. 2, pp. 153–168, 2021, DOI: 10.1109/OJIES.2021.3055901.

[12] I. Čilić, P. Krivić, I. Podnar Žarko, and M. Kušek, "*Performance Evaluation of Container Orchestration Tools in Edge Computing Environments*," *Sensors*, Vol. 23, No. 8, Apr. 2023, DOI: 10.3390/s23084008.

[13] A. Cañete, M. Amor, and L. Fuentes, "*HADES: An NFV Solution for Energy-Efficient Placement and Resource Allocation in Heterogeneous Infrastructures*," *Journal of Network and Computer Applications*, Vol. 221, Jan. 2024, DOI: 10.1016/j.jnca.2023.103764.

[14] L. H. Phuc, L. A. Phan, and T. Kim, "*Traffic-Aware Horizontal Pod Autoscaler in Kubernetes-based Edge Computing Infrastructure*," *IEEE Access*, Vol. 10, pp. 18966–18977, 2022, DOI: 10.1109/ACCESS.2022.3150867.

[15] Q. Su *et al.*, "*Low-Overhead Intra-Host Container Communication with Hardware Offloading*," *IEEE Transactions on Networking*, Vol. 33, No. 3, pp. 1070–1085, 2025, DOI: 10.1109/TON.2024.3520210.