JURIKOM (Jurnal Riset Komputer), Vol. 13 No. 2, April 2026
e-ISSN 2715-7393 (Media Online), p-ISSN 2407-389X (Media Cetak)
DOI 10.30865/jurikom.v13i2.9618
Hal 619-631
https://ejurnal.stmik-budidarma.ac.id/index.php/jurikom

Evaluasi Efektivitas Firewall Pre-filtering berbasis eBPF/XDP menggunakan Random Forest untuk Deteksi Anomali Trafik pada Docker Swarm

Ahmad Rohliyanto[*], Ema Utami

Fakultas Ilmu Komputer, S2 PJJ Informatika, Universitas Amikom Yogyakarta, Yogyakarta, Indonesia
Email: [1*]ahmadrohliyanto@students.amikom.ac.id, [2]ema.u@amikom.ac.id
Email Penulis Korespondensi: ahmadrohliyanto@students.amikom.ac.id
Submitted **25-02-2026;** Accepted **16-04-2026;** Published **30-04-2026**

Abstrak

Overlay network pada platform orkestrasi kontainer seperti Docker Swarm rentan terhadap serangan DDoS volumetrik, sementara solusi firewall konvensional menimbulkan overhead tinggi saat memproses volume serangan besar. Penelitian ini mengimplementasikan dan mengevaluasi firewall pre-filtering berbasis eBPF/XDP yang mengintegrasikan aturan deteksi dari model Random Forest untuk mendeteksi anomali trafik pada overlay network Docker Swarm. Berbeda dari penelitian sebelumnya yang menggunakan Decision Tree tunggal atau memproses klasifikasi di user-space, penelitian ini mengekstraksi aturan Random Forest menjadi threshold per-source-IP yang dieksekusi langsung di kernel melalui XDP dan disimpan dalam eBPF config_map sehingga dapat diperbarui pada runtime tanpa kompilasi ulang. Model dilatih menggunakan dataset CIC-DDoS-2019 (174.221 record, 65 fitur), menghasilkan akurasi 99,88%, detection rate 99,90%, false positive rate 0,14%, dan ROC-AUC 0,9999. Evaluasi terhadap tujuh skenario pengujian dengan 10 iterasi menunjukkan bahwa firewall XDP secara konsisten men-drop lebih dari 99,9% paket serangan dengan median response time 0,69 ms, setara dengan kondisi baseline. Overhead CPU tetap rendah (0,92–1,18%) dan throughput stabil (~920 Mbps). Perbedaan antar-skenario signifikan secara statistik ($p < 0,05$) namun dengan effect size sangat kecil ($d < 0,25$). Perbandingan dengan iptables, baik global rate limiting maupun per-IP hashlimit, menunjukkan bahwa ketiga pendekatan (XDP, iptables global, dan iptables per-IP) efektif dalam mitigasi DDoS dengan median response time yang setara.

Kata Kunci: Firewall Pre-filtering; eBPF/XDP; Random Forest; Deteksi Anomali Trafik; Docker Swarm

Abstract

Overlay networks in container orchestration platforms such as Docker Swarm are vulnerable to volumetric DDoS attacks, while conventional firewall solutions impose high overhead when processing large attack volumes. This paper presents the implementation and evaluation of an eBPF/XDP-based pre-filtering firewall that integrates detection rules derived from a Random Forest model to identify traffic anomalies in Docker Swarm overlay networks. Unlike previous studies that employ a single Decision Tree or process classification in user-space, this research extracts Random Forest rules into per-source-IP thresholds executed directly in the kernel via XDP and stored in an eBPF config_map to enable runtime updates without recompilation. The model was trained on the CIC-DDoS-2019 dataset (174,221 records, 65 features), achieving 99.88% accuracy, 99.90% detection rate, 0.14% false positive rate, and ROC-AUC of 0.9999. Experimental evaluation across seven testing scenarios with 10 iterations demonstrates that the XDP firewall drops over 99.9% of attack packets with a median response time of 0.69 ms, comparable to baseline conditions. CPU overhead remains low (0.92–1.18%) and throughput is maintained at approximately 920 Mbps. Differences between scenarios are statistically significant ($p < 0.05$) but with negligible practical effect ($d < 0.25$). Comparative analysis with iptables, both global rate limiting and per-IP hashlimit, indicates that all three approaches (XDP, global iptables, and per-IP iptables) effectively mitigate DDoS with comparable median response times.

Keywords: Firewall Pre-filtering; eBPF/XDP; Random Forest; Deteksi Anomali Trafik; Docker Swarm

1. PENDAHULUAN

Adopsi teknologi kontainerisasi pada lingkungan produksi mengalami pertumbuhan yang pesat dalam beberapa tahun terakhir [1]. Platform orkestrasi kontainer seperti *Docker Swarm* [2] menyediakan kemudahan *deployment* dan skalabilitas melalui *overlay network berbasis* VXLAN [3] yang menghubungkan kontainer antar-node. Namun, arsitektur *overlay network* ini menghadirkan tantangan keamanan tersendiri, terutama terhadap serangan volumetrik *Distributed Denial of Service* (DDoS) yang terus meningkat.

Solusi *firewall* konvensional seperti *iptables* memproses paket setelah melewati sebagian besar *network stack kernel*, termasuk alokasi *socket buffer* (sk_buff), yang menimbulkan *overhead* komputasi cukup besar saat volume serangan tinggi. *Extended Berkeley Packet Filter* (eBPF) melalui *eXpress Data Path* (XDP) menawarkan pendekatan alternatif dengan memproses paket langsung di *level driver* NIC, sebelum alokasi sk_*buff* [4], [5]. Scholz dkk. [6] menunjukkan bahwa XDP mampu memproses paket pada kecepatan 10 Mpps *per core*, jauh melampaui kemampuan *iptables*. Vieira dkk. [4] menyediakan panduan menyeluruh yang menjadi acuan bagi implementasi eBPF/XDP di berbagai domain.

Di sisi lain, pendekatan deteksi berbasis aturan statis memiliki keterbatasan dalam menghadapi pola serangan yang terus berkembang. Berbagai pendekatan *machine learning* telah diusulkan untuk deteksi anomali jaringan, mulai dari model berbasis *autoencoder* [7] hingga *framework* deteksi berbasis *supervised learning* [8]. Di antara berbagai algoritma tersebut, *Random Forest* [9] dipilih dalam penelitian ini karena kompatibilitasnya dengan eBPF, yakni aturan klasifikasi berupa pohon keputusan [10] (*if-else*) dapat diekstraksi dan dieksekusi secara efisien di *kernel space* tanpa memerlukan operasi *floating-point* yang kompleks. Hal ini berbeda dengan pendekatan *deep learning* seperti *Decision Tree* [11]

JURIKOM (Jurnal Riset Komputer), Vol. 13 No. 2, April 2026
e-ISSN 2715-7393 (Media Online), p-ISSN 2407-389X (Media Cetak)
DOI 10.30865/jurikom.v13i2.9618
Hal 619-631
https://ejurnal.stmik-budidarma.ac.id/index.php/jurikom

maupun BiLSTM [12]. Selain itu, *threshold* deteksi dapat disimpan dalam eBPF map yang memungkinkan pembaruan pada *runtime* tanpa perlu melakukan kompilasi ulang pada program XDP.

Beberapa penelitian terdahulu telah mengeksplorasi penerapan eBPF/XDP dan *machine learning* untuk keamanan jaringan dan kontainer. Scholz dkk. [6] menguji kinerja eBPF untuk pemfilteran paket pada jaringan 10 GbE dengan *throughput* 10 Mpps, namun berfokus pada *benchmark* kinerja tanpa integrasi ML. Bachl dkk. [11] mengimplementasikan IDS berbasis *flow* menggunakan Decision Tree di eBPF dengan akurasi 99%, namun hanya menggunakan Decision Tree dan menyebutkan *Random Forest* sebagai *future work*. Anand dkk. [13] membandingkan empat algoritma ML untuk IDS berbasis eBPF dengan *Random Forest* mencapai akurasi 99,44%, namun terbatas pada level *socket filter* (bukan XDP) tanpa *environment* kontainer. Farasat dkk. [12] mengembangkan *framework* SmartX yang menggabungkan BiLSTM dengan eBPF/XDP pada *Kubernetes* dengan akurasi 99,3%, namun klasifikasi dilakukan di *user-space*. Chen dkk. [14] mengimplementasikan deteksi DDoS menggunakan XGBoost dengan eBPF/XDP pada lingkungan *cloud data center*, yang menunjukkan tren terkini integrasi ML dengan XDP untuk mitigasi serangan. Nam dkk. [15] merancang Bastionp untuk *security function chaining* antar kontainer menggunakan XDP/eBPF dengan fokus pada isolasi trafik. Lin dkk. [16] membangun ONCache untuk optimasi performa *overlay network* tanpa membahas aspek keamanan. He dkk. [17] menganalisis kerentanan eBPF sebagai vektor serangan lintas-kontainer, dan Lee dkk. [18] mengembangkan metode *packet tracing* berbasis eBPF untuk pengukuran latensi. Studi *review* terbaru [19] mengonfirmasi bahwa penerapan eBPF/XDP untuk mitigasi DDoS di lingkungan kontainer masih didominasi oleh platform *Kubernetes*, sementara *Docker Swarm* dengan arsitektur *overlay* VXLAN belum banyak diteliti.

Terdapat tiga kesenjangan utama pada penelitian terdahulu. Pertama, penelitian keamanan kontainer berbasis eBPF lebih banyak berfokus pada *Kubernetes* [12], [14], [15], [19], sedangkan *Docker Swarm* dengan arsitektur *overlay network* VXLAN yang berbeda masih kurang mendapat perhatian. Kedua, implementasi ML di eBPF masih terbatas pada *Decision Tree* [11] Bachl dkk. menyebutkan bahwa pengujian *Random Forest* di eBPF merupakan *future work* yang belum dilakukan, sementara pendekatan yang lebih kompleks seperti BiLSTM [12] memerlukan pemrosesan di *user-space* yang menambah *overhead*. Ketiga, sebagian besar penelitian menggunakan aturan statis tanpa mekanisme pembaruan parameter deteksi pada *runtime*.

Berdasarkan kesenjangan tersebut, penelitian ini mengajukan pertanyaan "sejauh mana efektivitas *firewall* pre-filtering berbasis eBPF/XDP yang mengintegrasikan aturan deteksi *Random Forest* dalam mendeteksi anomali trafik pada *Docker Swarm overlay network*, dan bagaimana perbandingannya dengan iptables?".

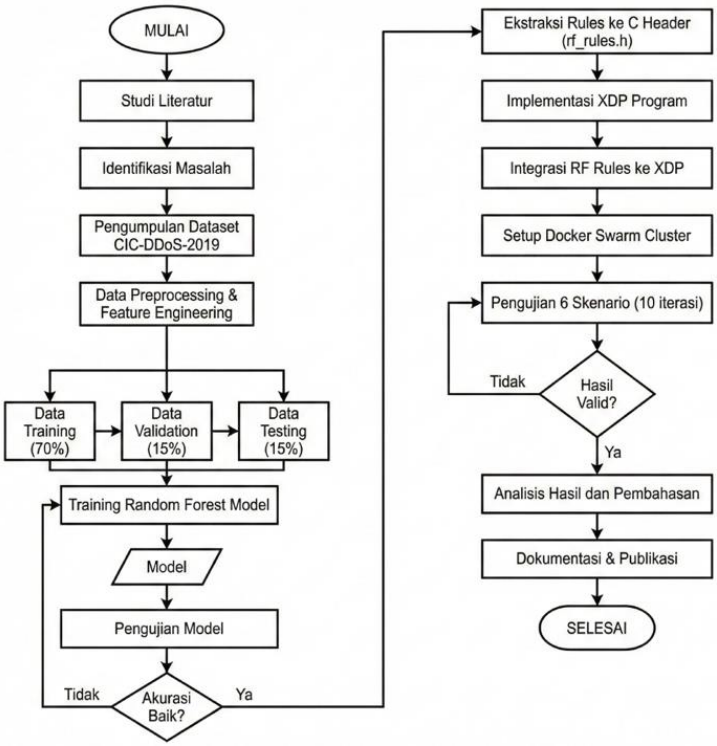
Penelitian ini mengimplementasikan *firewall* pre-filtering [20] berbasis eBPF/XDP yang mengintegrasikan aturan deteksi dari model *Random Forest* untuk mendeteksi anomali trafik dan memblokir serangan DDoS pada *overlay network Docker Swarm*. Kontribusi utama penelitian ini meliputi (1) implementasi dan evaluasi *firewall* eBPF/XDP secara spesifik pada *platform Docker Swarm* dengan *overlay network* VXLAN, (2) integrasi aturan *Random Forest* yang dilatih di *user-space* namun dieksekusi di *kernel-space* sebagai *threshold rule*, menjawab *future work* yang diidentifikasi oleh Bachl dkk. [11], (3) mekanisme konfigurasi dinamis melalui eBPF config_map yang memungkinkan pembaruan parameter deteksi pada *runtime* tanpa kompilasi ulang, serta (4) analisis perbandingan antara XDP dan *iptables* (*global* dan per-IP) dalam mitigasi DDoS.

Hasil pengujian menunjukkan bahwa *firewall* XDP mampu men-*drop* lebih dari 99,9% paket serangan pada seluruh skenario dengan dampak minimal terhadap layanan.

2. METODOLOGI PENELITIAN

Penelitian ini menggunakan pendekatan eksperimental kuantitatif. Metodologi penelitian mengikuti alur yang ditunjukkan pada Gambar 1, yang terdiri dari lima tahapan utama: (1) persiapan dan *preprocessing dataset* CIC-DDoS-2019, (2) *training* model *Random Forest* dan ekstraksi *threshold*, (3) implementasi *program* XDP, (4) *setup testbed Docker Swarm*, dan (5) pengujian tujuh skenario.

JURIKOM (Jurnal Riset Komputer), Vol. 13 No. 2, April 2026
e-ISSN 2715-7393 (Media Online), p-ISSN 2407-389X (Media Cetak)
DOI 10.30865/jurikom.v13i2.9618
Hal 619-631
https://ejurnal.stmik-budidarma.ac.id/index.php/jurikom

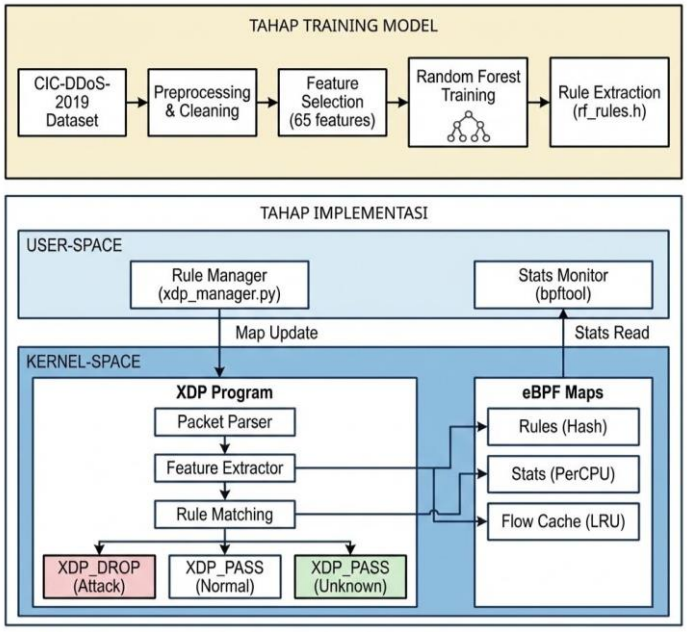


Gambar 1. Alur Penelitian

Gambar 1 menunjukkan bahwa proses dimulai dari tahap *offline* (*preprocessing* dan *training*) yang menghasilkan *threshold* deteksi, kemudian dilanjutkan ke tahap *online* (implementasi XDP dan pengujian) untuk mengevaluasi efektivitas sistem.

2.1 Arsitektur Sistem

Arsitektur sistem dirancang dengan memisahkan dua komponen utama sebagaimana ditunjukkan pada Gambar 2, yaitu komponen *user-space* untuk training model dan ekstraksi aturan, dan komponen *kernel-space* berupa program XDP yang mengeksekusi aturan tersebut pada setiap paket yang masuk melalui *interface* jaringan.



Gambar 2. Arsitektur Integrasi *Random Forest* dengan XDP

Pada Gambar 2 terlihat bahwa model *Random Forest* dilatih secara *offline* menggunakan dataset CIC-DDoS-2019 di *user-space*. Aturan klasifikasi yang dihasilkan kemudian diekstraksi menjadi *threshold* sederhana dan dimuat ke dalam eBPF config_map, sehingga program XDP di *kernel-space* dapat menggunakan *threshold* tersebut untuk mengevaluasi setiap paket secara *real-time*.

JURIKOM (Jurnal Riset Komputer), Vol. 13 No. 2, April 2026
e-ISSN 2715-7393 (Media Online), p-ISSN 2407-389X (Media Cetak)
DOI 10.30865/jurikom.v13i2.9618
Hal 619-631
https://ejurnal.stmik-budidarma.ac.id/index.php/jurikom

2.2 Persiapan Dataset

Dataset CIC-DDoS-2019 [21] digunakan sebagai data *training*. Dari 431.371 baris dan 80 kolom, dilakukan *filtering* terhadap *label* yang relevan dengan skenario pengujian seperti *Benign* (97.831 baris), *Syn* (49.373), UDP (18.090), dan UDP-lag (8.927), menghasilkan *subset* 174.221 baris. Komposisi kelas pada *subset* ini adalah 56,2% *Benign* dan 43,8% *Attack*, yang tergolong cukup seimbang sehingga tidak memerlukan teknik *resampling*. Setelah penghapusan kolom konstan (12 kolom), kolom *index*, dan kolom *label*, tersisa 65 fitur numerik. Data dibagi dengan rasio 70:15:15 menghasilkan *training set* (121.954 baris), *validation set* (26.133 baris), dan *test set* (26.134 baris).

2.3 Training Model *Random Forest*

Model *Random Forest* [9] dikonfigurasi dengan 100 *decision trees*, kedalaman maksimal 10 *level*, dan *random state* 42. *Training* dilaksanakan menggunakan *scikit-learn* 1.4 pada *Ubuntu Server* 24.04 LTS. Pendekatan *ensemble Random Forest* dipilih karena kemampuannya menghasilkan aturan klasifikasi berbasis pohon keputusan yang dapat diekstraksi menjadi format *if-else* sederhana, kompatibel dengan batasan eBPF *verifier* yang melarang *loop* tak terbatas dan operasi *floating-point* [4], [11]. Evaluasi dilakukan berdasarkan *confusion matrix* yang menghasilkan metrik akurasi, *precision*, *recall*, F1-*score*, *detection rate*, dan *false positive rate*. Aturan klasifikasi diekstraksi menjadi kumpulan *threshold* sederhana dalam format C *header* (rf_rules.h) yang dikompilasi bersama *program* XDP.

Untuk mengukur kinerja klasifikasi model, digunakan *confusion matrix* yang membandingkan prediksi model terhadap label aktual. Tabel 1 memperlihatkan struktur *confusion matrix* yang digunakan untuk mengklasifikasikan hasil prediksi model ke dalam empat kategori yaitu *True Positive* (TP), *True Negative* (TN), *False Positive* (FP), dan *False Negative* (FN).

Tabel 1. Struktur *Confusion Matrix*

	Prediksi Attack	**Prediksi Normal**
Aktual Attack	True Positive (TP)	False Negative (FN)
Aktual Normal	False Positive (FP)	True Negative (TN)

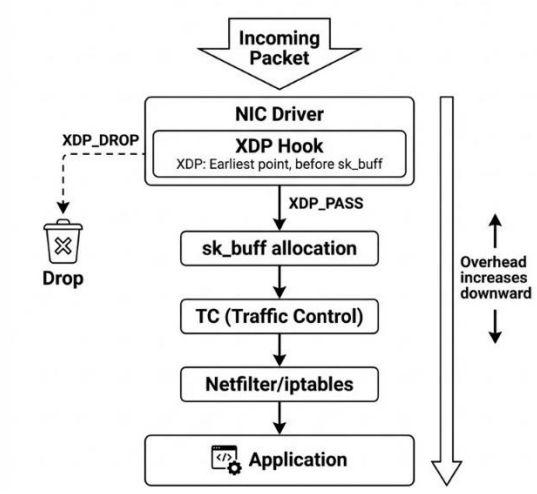
Berdasarkan nilai TP, TN, FP, dan FN pada Tabel 1, selanjutnya dihitung enam metrik evaluasi yang merepresentasikan berbagai aspek kinerja klasifikasi, sebagaimana dirumuskan pada Tabel 2.

Tabel 2. Metrik Evaluasi

Metrik	**Rumus**
Accuracy	False (TP + TN) / (TP + TN + FP + FN)
Precision	TP / (TP + FP)
Recall	TP / (TP + FN)
F1-Score	2 × (Precision × Recall) / (Precision + Recall)
Detection Rate	TP / (TP + FN)
False Positive Rate	FP / (FP + TN)

2.4 Implementasi *Program* XDP

Program XDP (xdp_firewall.c) diimplementasikan dalam bahasa C dan dikompilasi dengan *Clang* sebagai target eBPF. Untuk memahami posisi XDP dalam arsitektur jaringan *Linux*, Gambar 3 menunjukkan lokasi *hook point* XDP dalam *network stack*. *Hook point* XDP terletak langsung di NIC *driver*, sebelum alokasi sk_buff, sehingga pemrosesan paket memiliki *overhead* yang sangat rendah.



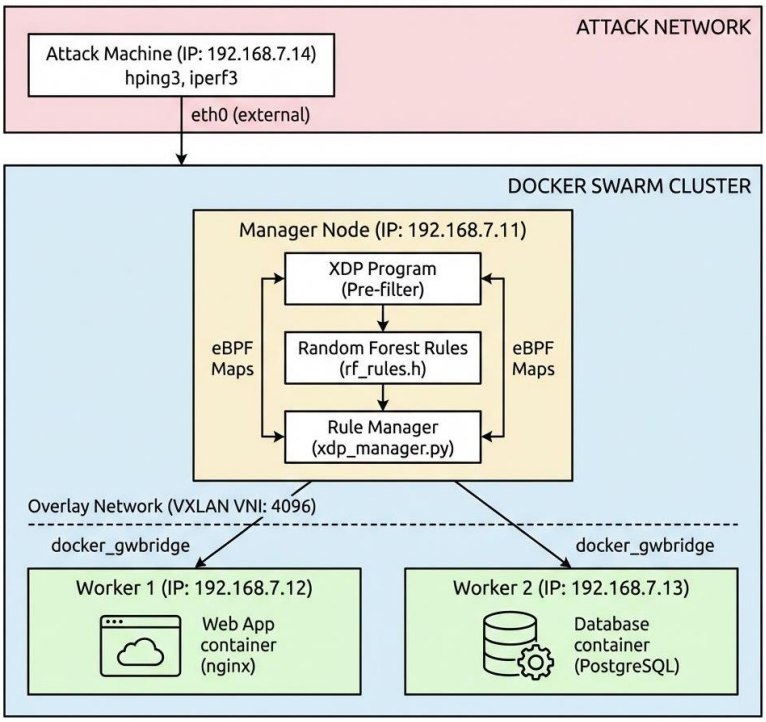
Gambar 3. Posisi XDP dalam *Linux Network Stack*

JURIKOM (Jurnal Riset Komputer), Vol. 13 No. 2, April 2026
e-ISSN 2715-7393 (Media Online), p-ISSN 2407-389X (Media Cetak)
DOI 10.30865/jurikom.v13i2.9618
Hal 619-631
https://ejurnal.stmik-budidarma.ac.id/index.php/jurikom

Threshold deteksi diturunkan dari model *Random Forest* melalui dua tahap. Pertama, analisis *feature importance* mengidentifikasi fitur-fitur yang paling berpengaruh seperti *Bwd Packets*/s (*rank* 1, *importance* 0,1172), *Flow Packets*/s (*rank* 4, 0,0493), dan Total *Fwd Packets* (*rank* 24, 0,0221). Pendekatan serupa digunakan oleh Anand dkk. [13] dan Chen dkk. [14] dalam mengekstraksi aturan ML untuk diterapkan pada eBPF. Kedua, metrik per-*flow* dari dataset CIC-DDoS-2019 dikonversi menjadi *threshold* per-*source*-IP yang dapat diterapkan oleh XDP, menggunakan formula yaitu *threshold* = benign_P95_per_flow × concurrent_flow_factor(5) × safety_margin(2,0). Dengan pendekatan ini, dihasilkan tiga aturan deteksi (1) deteksi SYN *flood* dengan *threshold* 50 paket SYN per *flow* (ditetapkan secara praktis karena SYN *Flag Count* memiliki *rank* 65 dari 65 fitur, RF mendeteksi serangan SYN melalui *pola rate*, bukan *flag counting*), (2) deteksi UDP *flood* dengan *threshold* 350 paket UDP per *flow* (*benign* P95 = 36 × 5 × 2,0 = 360, dibulatkan ke 350), dan (3) deteksi anomali *packet rate* melebihi 3.000 paket per detik (*benign* P75 *Flow Packets*/s = 328 × 5 × 2,0 = 3.281, *capped* ke 3.000). *Threshold* ini disimpan dalam eBPF *map* bertipe BPF_MAP_TYPE_ARRAY (*config_map*), sehingga dapat diperbarui pada *runtime* melalui tool *user-space (xdp_manager.py)* tanpa perlu mengompilasi ulang program XDP. Setiap *flow* dilacak menggunakan LRU *hash map* dengan kapasitas 65.536 entri, 5-*tuple key* (*source* IP, *destination* IP, *source port*, *destination port, protocol*), dan *window timeout* 10 detik.

2.5 Konfigurasi *Testbed*

Testbed dibangun menggunakan dua mesin fisik sebagaimana ditunjukkan pada Gambar 4. *Server Lenovo ThinkSystem* ST50 V2 (*Intel Xeon* E-2324G, 16 GB RAM ECC) menjalankan *Proxmox* VE 9.0.11 sebagai *hypervisor* untuk tiga *virtual machine* sebagai *Docker Swarm cluster*, dan PC *desktop* (*Intel Core* i5-11400F, 16 GB RAM) sebagai *attack machine*. Setiap VM menjalankan *Ubuntu Server* 24.04 LTS (kernel 6.8). *Program* XDP berjalan dalam *mode generic* pada *interface virtio-net* (NIC *Virtual*). Kedua mesin terhubung melalui jaringan *Gigabit Ethernet* (~941 Mbps *via iperf3*).



Gambar 4. Topologi *testbed Docker Swarm*

Gambar 4 menunjukkan bahwa *node manager* berfungsi sebagai titik masuk trafik sekaligus lokasi program XDP, sedangkan dua *worker node* menjalankan layanan *web* dan *database*. Adapun untuk spesifikasi lengkap konfigurasi *testbed* diperlihatkan pada Tabel 3.

Tabel 3. Topologi *testbed Docker Swarm*

Node	Peran	IP	RAM	vCPU
vm-manager	*Swarm Manager*, XDP, RF	192.168.7.11	6 GB	4
vm-worker-1	*Worker: Web App* (nginx)	192.168.7.12	5 GB	2
vm-worker-2	*Worker: Database* (PostgreSQL)	192.168.7.13	5 GB	2
PC *Attacker*	*Attack Machine* (*hping3, iperf3*)	192.168.7.14	16 GB	6C/12T

Sebagaimana ditunjukkan pada Tabel 3, *node manager* dialokasikan dengan sumber daya terbesar (6 GB RAM, 4 vCPU) karena menjalankan *program* XDP dan model RF selain fungsi orkestrasi *Swarm*. PC *Attacker* menggunakan spesifikasi yang lebih tinggi untuk menghasilkan *volume* serangan yang memadai.

JURIKOM (Jurnal Riset Komputer), Vol. 13 No. 2, April 2026
e-ISSN 2715-7393 (Media Online), p-ISSN 2407-389X (Media Cetak)
DOI 10.30865/jurikom.v13i2.9618
Hal 619-631
https://ejurnal.stmik-budidarma.ac.id/index.php/jurikom

2.6 Skenario Pengujian

Untuk mengevaluasi efektivitas *firewall* secara menyeluruh, dirancang tujuh skenario pengujian yang mencakup kondisi *baseline*, tiga jenis serangan DDoS dengan XDP, perbandingan dengan *iptables*, dan analisis dampak layanan. Rincian setiap skenario ditunjukkan pada Tabel 4. Setiap skenario dijalankan sebanyak 10 iterasi dengan durasi 60 detik per iterasi. Jumlah iterasi dipilih berdasarkan standar minimum uji statistik parametrik (t-test) yang mensyaratkan n ≥ 10 sampel [22]. Metrik yang dikumpulkan meliputi *response time* (*curl* setiap 1 detik ke port 80), CPU *usage* (*mpstat* 1 detik *interval*), *throughput* (*iperf3* selama 60 detik), dan *packet statistics* (*bpftool map dump* untuk XDP, *iptables* -L -v -n untuk *iptables*).

Tabel 4. Skenario Pengujian

Skenario	Deksripsi	Serangan	Firewall
S1	*Baseline* Normal	Tidak ada	Tidak ada
S2	SYN *Flood* + XDP	SYN *flood* (*hping3 --flood*)	XDP
S3	UDP *Flood* + XDP	UDP *flood* (*hping3 --udp --flood*)	XDP
S4	*Mixed Traffic* + XDP	SYN 30s + UDP 30s	XDP
S5	SYN *Flood* + *iptables*	SYN *flood* (*hping3 --flood*)	*iptables mangle PREROUTING (rate limit global)*
S6	*Service Impact* + XDP	SYN *flood* (*hping3 --flood*)	XDP (fokus dampak layanan)
S7	SYN *Flood* + *iptables (*per-*IP)*	SYN *flood* (*hping3 --flood*)	*iptables hashlimit* (*rate limit* per-*source*-IP)

Berdasarkan Tabel 4, skenario S1 menjadi acuan *baseline* untuk mengukur kondisi normal tanpa serangan, sedangkan S2–S4 mengevaluasi efektivitas XDP terhadap tiga jenis serangan. Skenario S5 dan S7 menyediakan perbandingan dengan *iptables* menggunakan dua mekanisme yang berbeda, dan S6 mengukur dampak serangan terhadap layanan.

Perbandingan *iptables* pada S5 menggunakan aturan di tabel *mangle chain* PREROUTING dengan *rate limiting global*, yaitu titik pemrosesan paling awal dalam *stack netfilter* yang setara dengan posisi XDP pada NIC *driver*. Skenario S7 menambahkan perbandingan dengan *iptables hashlimit* yang menerapkan *rate limiting* per-*source*-IP, setara dengan mekanisme per-*flow tracking* pada XDP. Pemilihan *tabel mangle* PREROUTING diperlukan karena *Docker Swarm* menggunakan DNAT pada *tabel nat* untuk meneruskan *traffic port* yang di*publish* ke kontainer melalui *chain* FORWARD, sehingga aturan pada *chain* INPUT tidak efektif untuk memfilter serangan. Seluruh *threshold* pada *iptables* (SYN 50/s, UDP 350/s, PPS 3.000/s) disamakan dengan *threshold* XDP untuk menjamin perbandingan yang adil.

3. HASIL DAN PEMBAHASAN

3.1 Evaluasi Model *Random Forest*

Proses *training Random Forest* dilakukan pada 121.954 baris data *training* dengan 65 fitur numerik. Algoritma membentuk 100 *decision tree* secara paralel, dimana setiap *tree* dilatih pada subset acak dari data dan fitur. Pada tahap prediksi, setiap *tree* menghasilkan klasifikasi independen (*Attack* atau *Benign*), dan prediksi akhir ditentukan melalui *voting* mayoritas dari seluruh 100 *tree*. Model yang telah dilatih kemudian diuji pada *test set* yang terdiri dari 26.134 baris data yang tidak pernah dilihat selama *training*.

Evaluasi model pada *test set* menghasilkan *confusion matrix* dengan *True Positive* (TP) = 11.447, *True Negative* (TN) = 14.655, *False Positive* (FP) = 20, dan *False Negative* (FN) = 12. Dari nilai tersebut, akurasi dihitung sebagai (11.447 + 14.655) / 26.134 = 99,88%, *detection rate* sebagai 11.447 / (11.447 + 12) = 99,90%, dan *false positive rate* sebagai 20 / (20 + 14.655) = 0,14%. Rangkuman lengkap seluruh metrik evaluasi disajikan pada Tabel 5.

Tabel 5. Hasil Evaluasi Model *Random Forest*

Metrik	*Validation Set*	*Test Set*
Accuracy	99,86%	99,88%
Precision	0,9983	0,9983
Recall	0,9986	0,9990
F1-Score	0,9984	0,9986
Detection Rate	-	99,90%
False Positive Rate	-	0,14%
ROC-AUC	-	0,9999

Tabel 5 menunjukkan bahwa performa model konsisten antara validation set dan test set, yang mengindikasikan tidak terjadi overfitting. Analisis *feature importance* menunjukkan bahwa *Bwd Packets*/s (0,1172), *Fwd* IAT *Mean* (0,0563), dan *Flow* IAT Std (0,0503) merupakan tiga fitur paling berpengaruh. Evaluasi ROC-AUC pada *test set* menghasilkan nilai 0,9999, yang mengindikasikan kemampuan klasifikasi model yang sangat tinggi. Hasil ini sejalan

JURIKOM (Jurnal Riset Komputer), Vol. 13 No. 2, April 2026
e-ISSN 2715-7393 (Media Online), p-ISSN 2407-389X (Media Cetak)
DOI 10.30865/jurikom.v13i2.9618
Hal 619-631
https://ejurnal.stmik-budidarma.ac.id/index.php/jurikom

dengan temuan Anand dkk. [13] yang mencatat akurasi *Random Forest* sebesar 99,44% dan *Decision Tree* 99,38%, serta penelitian yang dilakukan Bachl dkk. [11] yang mencapai akurasi 99% menggunakan *Decision Tree* di eBPF, yang mendukung keunggulan *Random Forest*.

3.2 Implementasi XDP *Firewall*

Bagian utama dari implementasi *program* XDP untuk pembaruan *threshold* pada *runtime* dan deteksi SYN *Flood* ditunjukkan pada kode program berikut. Kode ini mendefinisikan struktur konfigurasi (fw_config) yang menyimpan empat parameter *threshold* deteksi dalam eBPF map bertipe *array*. Ketika *program* XDP menerima paket, ia membaca *threshold* dari *config_map* menggunakan *bpf_map_lookup_elem*. Jika pembacaan gagal atau nilai *threshold* belum diinisialisasi, program menggunakan nilai *default* yang dikompilasi bersama *program*. Mekanisme ini memungkinkan operator mengubah parameter deteksi saat program sedang berjalan tanpa perlu melakukan kompilasi ulang.

```
/* Runtime configuration - thresholds updatable via eBPF map */
struct fw_config {
    __u32 syn_threshold;     /* default: 50 */
    __u32 udp_threshold;     /* default: 350 */
    __u32 pps_high;          /* default: 3000 */
    __u32 flow_timeout_sec;  /* default: 10 */
};

struct {
    __uint(type, BPF_MAP_TYPE_ARRAY);
    __uint(max_entries, 1);
    __type(key, __u32);
    __type(value, struct fw_config);
} config_map SEC(".maps");

/* Di dalam xdp_firewall(): */
struct fw_config *cfg = bpf_map_lookup_elem(&config_map, &zero);
__u32 syn_th = (cfg && cfg->syn_threshold > 0)
             ? cfg->syn_threshold : DEFAULT_SYN_THRESHOLD;

/* Rule 1: TCP SYN Flood Detection */
if ((flags & TCP_SYN) && !(flags & TCP_ACK)) {
    flow->syn_count++;
    if (flow->syn_count > syn_th) {
        flow->blocked = 1;
        return XDP_DROP;
    }
}
```

Pada kode di atas, deteksi SYN *flood* bekerja dengan menghitung jumlah paket SYN yang diterima per *flow*. Setiap paket TCP yang memiliki *flag* SYN tanpa ACK (menandakan koneksi baru) akan menambah *counter syn_count*. Ketika *counter* melebihi *threshold* yang dibaca dari *config_map* (default: 50), flow ditandai sebagai *blocked* dan seluruh paket selanjutnya dari *flow* tersebut langsung di-*drop* melalui aksi XDP_DROP`.

Selain deteksi SYN *flood*, *program* XDP juga mengimplementasikan deteksi anomali *packet rate* sebagaimana ditunjukkan pada kode *program* berikut. Kode ini menghitung *packet rate* (paket per detik) berdasarkan jumlah paket dan durasi *flow*, kemudian membandingkannya dengan *threshold* dari *config_map*.

```
/* Rule 3: Packet Rate Anomaly */
__u64 duration_ns = now - flow->first_seen;
if (duration_ns > 1000000 && flow->pkt_count > 10) {
    __u64 pps = ((__u64)flow->pkt_count * 1000000000ULL)/duration_ns;
    if (pps > pps_hi) { /* threshold dari config_map */
        flow->blocked = 1;
        __sync_fetch_and_add(&s->dropped_packets, 1);
        __sync_fetch_and_add(&s->rate_limit_drops, 1);
        return XDP_DROP;
    }
}
```

Pada kode di atas, perhitungan *packet rate* hanya dilakukan setelah *flow* berjalan minimal 1 milidetik (duration_ns > 1000000) dan menerima lebih dari 10 paket, untuk menghindari *false positive* pada flow baru. *Packet rate* dihitung dengan membagi jumlah paket dengan durasi *flow* dalam satuan *nanosecond*, menghasilkan nilai *packets per second* (PPS). Jika PPS melebihi *threshold* (default: 3.000 pps), *flow* ditandai *blocked*, *counter* statistik diperbarui secara atomik menggunakan __sync_fetch_and_add, dan paket di-*drop*.

3.3 Pengujian

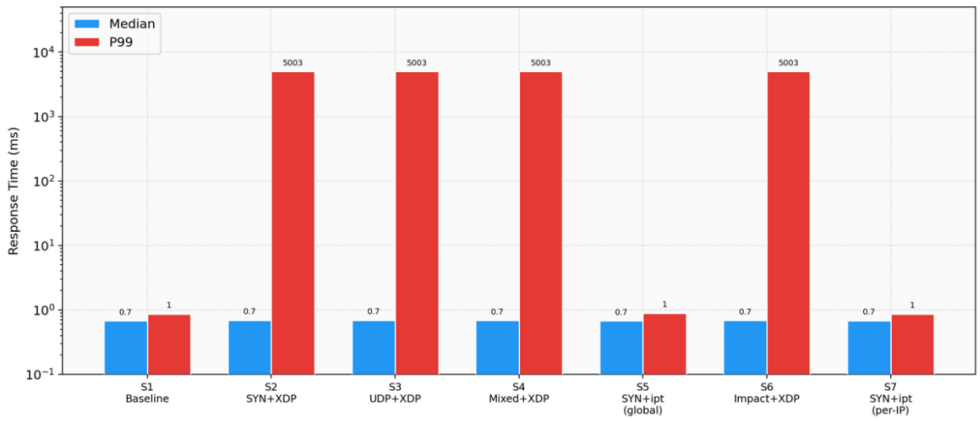
Setelah program XDP berhasil di-*attach* pada *interface* jaringan *node manager*, dilakukan pengujian pada tujuh skenario yang telah dirancang. Tabel 6 menyajikan ringkasan hasil pengujian di seluruh skenario. XDP *firewall* mencatat *drop rate* lebih dari 99,9% pada setiap skenario yang diujikan. Pada skenario SYN *flood* (S2), dari total 240.896.618 paket yang diterima, sebanyak 240.755.926 paket berhasil di-drop oleh *program* XDP. Pola serupa teramati pada UDP *flood* (S3) dengan 242.377.921 paket di-*drop* dan *mixed traffic* (S4) dengan 250.849.277 paket di-*drop*.

JURIKOM (Jurnal Riset Komputer), Vol. 13 No. 2, April 2026
e-ISSN 2715-7393 (Media Online), p-ISSN 2407-389X (Media Cetak)
DOI 10.30865/jurikom.v13i2.9618
Hal 619-631
https://ejurnal.stmik-budidarma.ac.id/index.php/jurikom

Tabel 6. Hasil Pengujian Seluruh Skenario (10 Iterasi × 60 Detik)

Metrik	S1 *Baseline*	S2 SYN + XDP	S3 UDP + XDP	S4 *Mixed* + UDP	S5 SYN + ipt (*global*)	S6 *Impact* + XDP	S7 SYN + ipt (per-IP)
Response Mean	0,75 ms	193,4 ms	174,0 ms	150,9 ms	0,68 ms	247,8 ms	0,69 ms
Response Median	0,68 ms	0,69 ms	0,69 ms	0,68 ms	0,68 ms	0,69 ms	0,68 ms
Response P99	0,87 ms	5.003 ms	5.003 ms	5.003 ms	0,89 ms	5.003 ms	0,86 ms
CPU Mean	1,14%	1,03%	0,92%	1,18%	1,42%	0,19%	0,91%
Throughput	920 Mbps	927 Mbps	31 Mbps	43 Mbps	919 Mbps	36 Mbps	932 Mbps
Packets Dropped	-	241 M	242 M	251 M	-	242 M	-
XDP Drop Rate	-	99,94%	99,94%	99,97%	-	99,98%	-

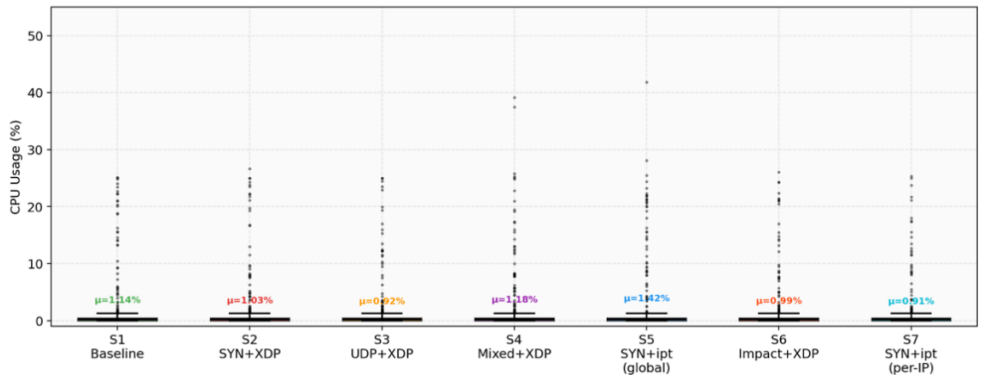
Dari Tabel 6 dapat diamati beberapa temuan penting. Pertama, nilai *mean response time* pada skenario S2–S4 dan S6 terlihat tinggi karena distribusi yang tidak simetris akibat lonjakan *timeout* saat deteksi *flow* awal, sedangkan *median* tetap stabil pada 0,68–0,69 ms di seluruh skenario yang menunjukkan bahwa mayoritas *request* tetap dilayani dengan cepat. Kedua, *drop rate* sebesar 99,9% pada setiap skenario XDP menunjukkan bahwa aturan deteksi yang diturunkan dari model *Random Forest* mampu mengidentifikasi dan memblokir *traffic* serangan *DDoS* secara efektif. Ketiga, *throughput* pada skenario UDP *flood* (S3) dan *mixed traffic* (S4) turun ke 31–43 Mbps karena serangan UDP menggunakan *bandwidth* yang lebih besar dibanding SYN *flood* yang hanya mengirim paket kecil (*header only*). Distribusi *response time* per skenario ditunjukkan pada Gambar 5.



Gambar 5. Visualisasi perbandingan *response time* per skenario *(median vs P99)*

Gambar 5 menunjukkan kontras yang jelas antara *median response time* yang konsisten rendah di seluruh skenario dengan P99 yang melonjak pada skenario XDP (S2–S4, S6), divisualisasikan dalam skala logaritmik untuk mengakomodasi perbedaan *magnitude*.

Untuk mengevaluasi dampak firewall terhadap beban komputasi, dilakukan pengukuran CPU menggunakan *mpstat* selama 10 iterasi per skenario menunjukkan bahwa rata-rata penggunaan CPU tidak berbeda jauh antara kondisi *baseline* (tanpa serangan) sebesar 1,14% dengan kondisi saat serangan berlangsung dan XDP aktif (1,03% pada SYN *flood*, 0,92% pada UDP *flood*, 1,18% pada *mixed traffic*). Distribusi CPU usage ditunjukkan pada Gambar 6.

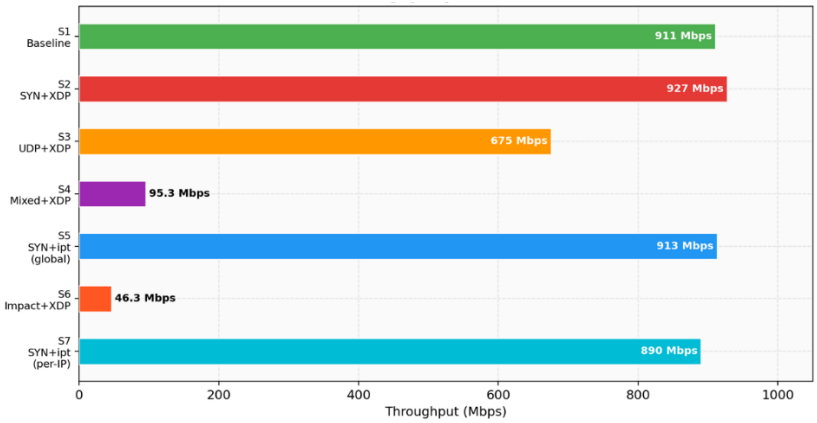


Gambar 6. Visualisasi Distribusi CPU *usage* per skenario (*box plot*)

Gambar 6 menunjukkan bahwa variasi CPU *usage* antar-skenario sangat kecil, yang mengindikasikan bahwa pemrosesan XDP di level *driver* NIC tidak menambah beban signifikan pada CPU meskipun *volume* serangan mencapai sekitar 24 juta paket per menit.

JURIKOM (Jurnal Riset Komputer), Vol. 13 No. 2, April 2026
e-ISSN 2715-7393 (Media Online), p-ISSN 2407-389X (Media Cetak)
DOI 10.30865/jurikom.v13i2.9618
Hal 619-631
https://ejurnal.stmik-budidarma.ac.id/index.php/jurikom

Untuk melengkapi analisis dampak *firewall*, *throughput* jaringan diukur menggunakan *iperf3*. *Throughput* pada skenario SYN *flood* (S2, S5, S7) stabil pada kisaran 919–932 Mbps sebagaimana divisualisasikan pada Gambar 7, mendekati kapasitas teoritis *Gigabit Ethernet*. Temuan ini mengindikasikan bahwa *firewall* tidak menyebabkan penurunan kapasitas *bandwidth* untuk serangan SYN *flood.*



Gambar 7. Visualisasi *throughput* per skenario

Pada Gambar 7 terlihat bahwa *throughput* hanya turun signifikan pada skenario UDP *flood* (S3) dan *mixed traffic* (S4), bukan karena *firewall* membatasi *bandwidth*, melainkan karena serangan UDP itu sendiri yang mengonsumsi kapasitas jaringan.

3.4 Pembahasan

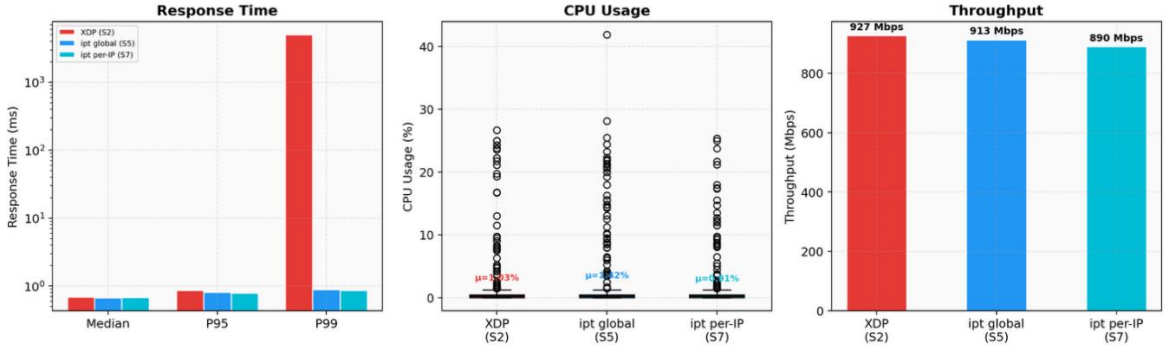
Bagian ini membahas temuan utama dari hasil pengujian, meliputi perbandingan pendekatan *firewall*, analisis statistik, mekanisme pembaruan *threshold* pada *runtime*, serta keterbatasan pengujian yang akan dibahas pada masing-masing sub bab.

3.4.1 Perbandingan XDP vs *iptables*

Untuk mengevaluasi posisi pemrosesan paket, dilakukan perbandingan langsung antara XDP (S2), *iptables global rate limiting* (S5), dan *iptables* per-IP *hashlimit* (S7) menggunakan skenario SYN *flood* yang sama. Tabel 7 menyajikan perbandingan detail dari enam aspek utama, dan Gambar 8 memvisualisasikan perbandingan ketiga pendekatan pada tiga metrik kunci.

Tabel 7. Perbandingan XDP vs *iptables* pada SYN *Flood*

Aspek	XDP (S2)	*Iptables global* (S5)	*Iptables* per-IP (S7)
Processing Layer	NIC *driver* (sebelum *sk_buff*)	*Netfilter mangle* PREROUTING	*Netfilter mangle* PREROUTING
Packets Dropped	241 juta (99,94%)	Rate limited	Rate limited
Response Median	0,69 ms	0,68 ms	0,68 ms
Response P99	5.003 ms	0,89 ms	0,86 ms
CPU *Mean*	1,03%	1,42%	0,91%
Throughput	927 Mbps	919 Mbps	932 Mbps
Mekanisme	Per-*flow threshold + block*	*Rate limiting global* (50 SYN/s)	*Rate limiting* per-*source*-IP (*hashlimit* 50/s)

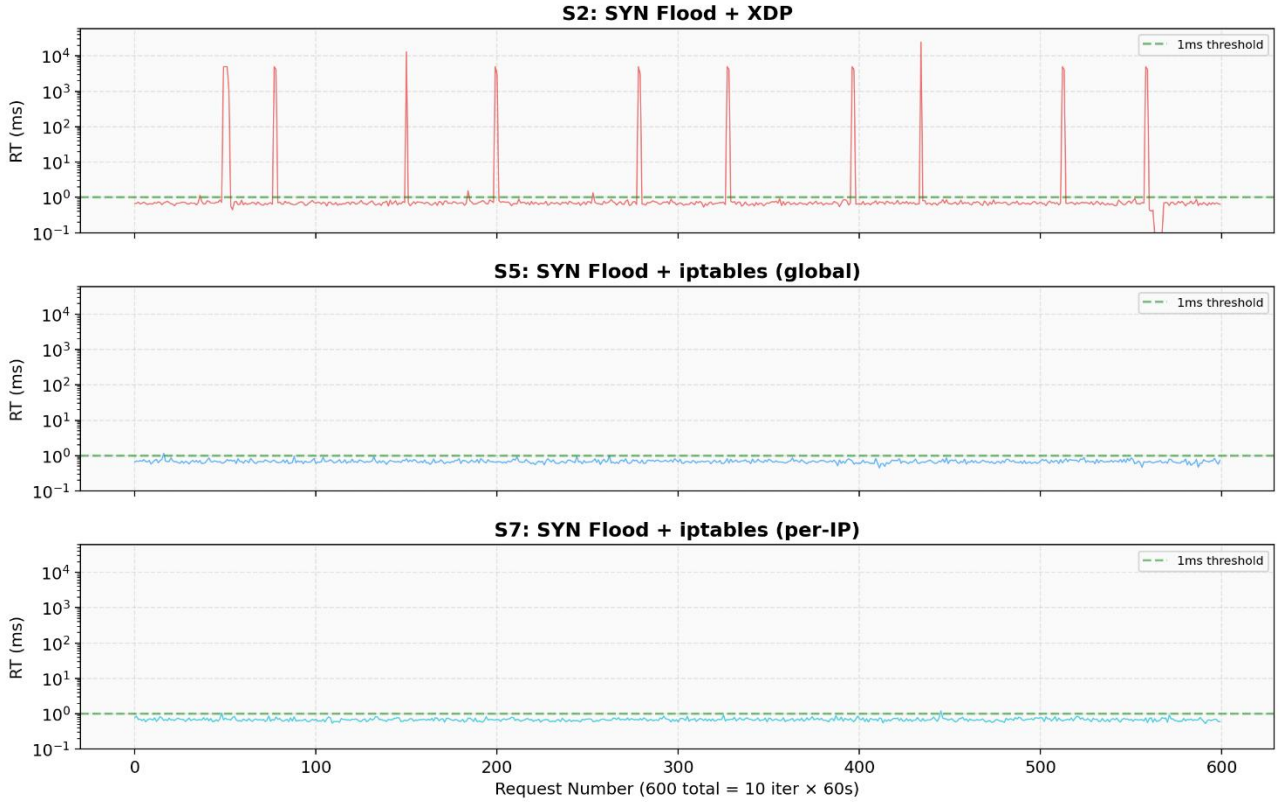


Gambar 8. Visualisasi Perbandingan XDP vs *iptables* (SYN Flood) pada *response time*, CPU dan *throughput*

JURIKOM (Jurnal Riset Komputer), Vol. 13 No. 2, April 2026
e-ISSN 2715-7393 (Media Online), p-ISSN 2407-389X (Media Cetak)
DOI 10.30865/jurikom.v13i2.9618
Hal 619-631
https://ejurnal.stmik-budidarma.ac.id/index.php/jurikom

Dari Tabel 7 dan Gambar 8, ketiga solusi berhasil mempertahankan *median response time* yang setara (~0,68–0,69 ms). Perbedaan utama terletak pada P99 *response time*, XDP mengalami lonjakan sementara (P99: 5.003 ms) saat *flow* baru pertama kali terdeteksi, sedangkan kedua konfigurasi *iptables* mempertahankan *response time* yang stabil (P99: 0,86–0,89 ms) sepanjang pengujian. Perbedaan ini disebabkan mekanisme kerja yang berbeda, dimana XDP menggunakan per-*flow threshold* yang memerlukan akumulasi paket sebelum memblokir suatu *flow*, sementara *iptables* menggunakan *rate limiting* yang langsung membatasi SYN melebihi 50 paket/detik.

Keunggulan pendekatan per-*flow* yang diterapkan XDP terletak pada kemampuannya mengevaluasi setiap *flow* secara independen, sehingga *flow* normal tidak ikut terdampak saat *flow* serangan diblokir. Mekanisme *rate limiting global* pada iptables S5 bekerja secara keseluruhan dan dapat memblokir koneksi yang sah ketika terjadi lonjakan trafik normal secara bersamaan. Sementara itu, *iptables hashlimit* pada S7 lebih mirip dengan XDP karena menerapkan limit per-source-IP, namun tetap menggunakan pendekatan *rate limiting* bukan evaluasi per-*flow*. Gambar 9 menampilkan timeline response time yang menunjukkan pola lonjakan pada masing-masing pendekatan.



Gambar 9. Visualisasi *timeline response time* S2 (XDP) vs S5 (*iptables global*) vs S7 (*iptables* per-IP)

Pada Gambar 9 terlihat bahwa lonjakan *response time* pada XDP (S2) terjadi secara periodik setiap ~10 detik (sesuai f*low timeout*), sementara kedua konfigurasi *iptables* (S5, S7) menunjukkan *response time* yang relatif konstan sepanjang pengujian.

Pengujian pada skenario dampak layanan (S6), *median response time* layanan *web* tercatat 0,69 ms selama serangan SYN *flood* berlangsung dengan XDP aktif, tidak berbeda dari kondisi *baseline*. Lonjakan *response time* terjadi setiap 10 detik (P99: 5.003 ms), bertepatan dengan habisnya *window timeout* pada *flow table* yang menyebabkan paket dari *attacker* kembali dievaluasi sebagai *flow* baru. Secara lebih rinci, mekanisme lonjakan ini terjadi dalam tiga tahap yaitu (1) saat *flow timeout* habis, *entry flow* dihapus dari LRU *hash map*, sehingga paket berikutnya dari *attacker* dianggap sebagai *flow* baru, (2) selama akumulasi menuju *threshold* (50 paket SYN), paket-paket tersebut melewati XDP dan masuk ke kernel *network stack*, mengisi TCP SYN *backlog*, (3) setelah *counter flow* mencapai *threshold*, flag *flow* > *blocked* diset dan seluruh paket selanjutnya langsung di-*drop* melalui XDP_DROP. Durasi lonjakan bergantung pada kecepatan *attacker* mencapai *threshold.* Pada pengujian ini, *hping*3 --*flood* menghasilkan jutaan paket per menit sehingga *threshold* 50 SYN tercapai dalam waktu kurang dari 1 detik.

Efektivitas kombinasi *Random Forest* dan XDP dalam mendeteksi anomali trafik dapat dijelaskan dari tiga aspek. Pertama, aturan klasifikasi *Random Forest* yang berupa pohon keputusan *if-else* secara alami kompatibel dengan batasan eBPF *verifier* yang melarang *loop* tak terbatas dan membatasi jumlah instruksi, sehingga *threshold* yang diekstraksi dapat dieksekusi secara efisien di *kernel* tanpa *overhead* interpretasi model. Kedua, pemrosesan XDP yang terjadi di level *driver* NIC sebelum *kernel* mengalokasikan *sk_buff* yang memungkinkan paket serangan di-*drop* pada tahap paling awal *network stack*, sehingga CPU tidak terbebani oleh pemrosesan *protocol stack* untuk paket yang akan dibuang. Ketiga, mekanisme per-*flow tracking* memungkinkan evaluasi independen terhadap setiap koneksi, sehingga trafik normal tetap dilayani meskipun *flow* serangan sedang diblokir.

JURIKOM (Jurnal Riset Komputer), Vol. 13 No. 2, April 2026
e-ISSN 2715-7393 (Media Online), p-ISSN 2407-389X (Media Cetak)
DOI 10.30865/jurikom.v13i2.9618
Hal 619-631
https://ejurnal.stmik-budidarma.ac.id/index.php/jurikom

Untuk memposisikan hasil penelitian ini dalam konteks literatur, Tabel 8 menyajikan perbandingan kuantitatif dengan penelitian sebelumnya yang mengintegrasikan *machine learning* dengan eBPF.

Tabel 8. Perbandingan dengan Penelitian Sebelumnya

Penelitian	Metode ML	*Platform eBPF*	Akurasi	*Environment*	*Pembaruan Runtime*
Bachl dkk. [11]	Decision Tree	Socket	99,0%	Standalone	Tidak
Anand dkk. [13]	Random Forest	Socket filter	99,44%	Standalone	Tidak
Farasat dkk. [12]	BiLSTM	XDP + user-space	99,3%	Kubernetes	Tidak
Chen dkk. [14]	XGBoost	XDP	-	Cloud DC	Tidak
Penelitian ini	Random Forest	XDP (Kernel)	99,88%	Docker Swarm	Ya

Tabel 8 menunjukkan bahwa penelitian ini mencapai akurasi tertinggi (99,88%) di antara pendekatan yang dibandingkan, sekaligus menjadi satu-satunya yang menerapkan mekanisme pembaruan *threshold* pada *runtime* melalui eBPF *config_map*. Selain itu, penelitian ini secara spesifik mengevaluasi pada *environment Docker Swarm overlay network* yang belum banyak diteliti dibandingkan *Kubernetes*.

Dari perspektif penerapan praktis, sistem yang diusulkan memiliki beberapa keunggulan operasional. Pertama, program XDP dapat di-*deploy* pada *node manager Docker Swarm* tanpa modifikasi pada aplikasi atau konfigurasi kontainer yang sudah berjalan. Kedua, *overhead* CPU yang rendah (<1,2%) memungkinkan penerapan di lingkungan produksi tanpa perlu menyediakan sumber daya komputasi tambahan. Ketiga, *threshold* deteksi yang disimpan dalam *config_map* memungkinkan operator menyesuaikan parameter sesuai karakteristik trafik aktual pada *runtime* tanpa kompilasi ulang maupun *downtime*.

3.4.2 Uji Statistik

Untuk memvalidasi apakah perbedaan hasil antar-skenario bersifat signifikan secara statistik, dilakukan uji normalitas dan uji perbandingan. Karena distribusi *response time* tidak normal berdasarkan uji *Shapiro-Wilk* sebagaimana ditunjukkan pada Tabel 9, maka digunakan uji *Mann-Whitney U* untuk membandingkan antar-skenario, disertai pengukuran *Cohen's* d untuk menilai dampak praktisnya dengan hasil perbandingan yang ditampilkan pada Tabel 10.

Tabel 9. Hasil Uji Normalitas *Shapiro-Wilk*

Skenario	n	W	p-value	Normal (α=0,05)
S1 (*Baseline*)	600	0,0308	$1{,}85\times10^{-47}$	Tidak
S2 (SYN+XDP)	600	0,1233	$6{,}08\times10^{-46}$	Tidak
S3 (UDP+XDP	600	0,1528	$1{,}98\times10^{-45}$	Tidak
S4 (*Mixed*+XDP)	600	0,1335	$9{,}12\times10^{-46}$	Tidak
S5 (*iptables global*)	600	0,9484	$1{,}26\times10^{-13}$	Tidak
S6 (*Service Impact*)	600	0,0766	$1{,}01\times10^{-46}$	Tidak
S7 (*iptables* per-IP)	600	0,9247	$9{,}35\times10^{-17}$	Tidak

Seluruh skenario pada Tabel 9 menunjukkan distribusi yang tidak normal ($p << 0{,}05$) karena data *response time* bersifat *right-skewed* yaitu mayoritas nilai berada di sekitar *median* (~0,68 ms) namun terdapat lonjakan (*outlier*) hingga 5.003 ms. Oleh karena itu, dipilih uji non-parametrik Mann-Whitney U.

Tabel 10. Hasil Uji *Mann-Whitney U*

Perbandingan	U	*p-value*	Signifikan	*Cohen's d*	*Effect Size*
Baseline vs SYN+XDP	159.682	$7{,}12\times10^{-4}$	Ya (***)	-0,1996	Sangat Kecil
Baseline vs UDP+XDP	160.639	$1{,}26\times10^{-3}$	Ya (**)	-0,2307	Kecil
Baseline vs Mixed+XDP	166.568	$2{,}52\times10^{-2}$	Ya (*)	-0,2091	Kecil
XDP vs iptables (global)	200.917	$4{,}93\times10^{-4}$	Ya (***)	0,1997	Sangat Kecil
XDP vs iptables (per-IP)	197.748	$3{,}11\times10^{-3}$	Ya (**)	0,1997	Sangat Kecil
iptables global vs per-IP	176.034	$5{,}09\times10^{-1}$	Tidak (ns)	-0,0193	Sangat Kecil
Baseline vs Service Impact	158.754	$4{,}01\times10^{-4}$	Ya (***)	-0,1526	Sangat Kecil

Keterangan: *p<0,05; **p<0,01; ***p<0,001; ns = tidak signifikan. Effect size: sangat kecil (|d|<0,2), kecil (0,2≤|d|<0,5), sedang (0,5≤|d|<0,8), besar (|d|≥0,8).

Dari Tabel 10 terlihat bahwa hampir seluruh perbandingan menunjukkan perbedaan yang signifikan ($p < 0{,}05$), namun nilai *Cohen's d* yang berkisar antara 0,02 hingga 0,23 menunjukkan bahwa perbedaan tersebut tergolong sangat kecil hingga kecil. Meskipun uji statistik mendeteksi adanya perbedaan, terutama yang disebabkan oleh lonjakan sesaat pada P99, dampaknya terhadap kualitas layanan secara keseluruhan sangat minimal karena *median response time* tetap setara di seluruh skenario. Satu-satunya perbandingan yang tidak signifikan adalah antara *iptables global* (S5) dan *iptables* per-IP (S7) dengan $p = 0{,}509$, yang menunjukkan bahwa kedua konfigurasi *iptables* memiliki performa yang setara.

3.4.3 Konfigurasi Dinamis *Threshold*

JURIKOM (Jurnal Riset Komputer), Vol. 13 No. 2, April 2026
e-ISSN 2715-7393 (Media Online), p-ISSN 2407-389X (Media Cetak)
DOI 10.30865/jurikom.v13i2.9618
Hal 619-631
https://ejurnal.stmik-budidarma.ac.id/index.php/jurikom

Arsitektur *firewall* yang diimplementasikan menyimpan *threshold* deteksi (SYN, UDP, PPS, dan *timeout*) dalam eBPF *map* bertipe *BPF_MAP_TYPE_ARRAY* yang disebut *config_map*. Dengan pendekatan ini, parameter deteksi dapat diperbarui saat program XDP sedang berjalan melalui *tool user-space* tanpa perlu melakukan kompilasi ulang program atau me-*restart* layanan.

Pembaruan *threshold* dilakukan melalui *xdp_manager.py* yang berinteraksi dengan *config_map* menggunakan *bpftool*. Gambar 10 menunjukkan contoh output pembaruan *threshold* yang dilakukan saat *program* XDP sedang aktif memproses paket.

```
manager@vm-manager:~/thesis/xdp$ python3 xdp_manager.py threshold set 30 100 3000
Thresholds updated successfully:
  SYN Threshold:  30
  UDP Threshold:  100
  PPS High:       3000
  Flow Timeout:   10s
manager@vm-manager:~/thesis/xdp$ make threshold-show
python3 xdp_manager.py threshold show
Raw config data:
[{
        "key": 0,
        "value": {
            "syn_threshold": 30,
            "udp_threshold": 100,
            "pps_high": 3000,
            "flow_timeout_sec": 10
        }
    }
]
manager@vm-manager:~/thesis/xdp$ python3 xdp_manager.py threshold init
Initializing config_map with default thresholds...
Thresholds updated successfully:
  SYN Threshold:  50
  UDP Threshold:  200
  PPS High:       5000
  Flow Timeout:   10s
manager@vm-manager:~/thesis/xdp$
```

Gambar 10. Output pembaruan *threshold* pada *runtime* melalui eBPF config_map

Pada Gambar 10 terlihat bahwa *threshold* deteksi berhasil diubah saat program XDP tetap berjalan, dan nilai baru tersebut langsung diterapkan pada paket yang akan di proses berikutnya. Mekanisme ini memberikan fleksibilitas operasional yang tidak tersedia pada pendekatan aturan statis.

3.4.4 Keterbatasan Pengujian

Perlu dicatat bahwa pengujian ini menggunakan XDP *mode generic* pada *interface virtio-net* di lingkungan virtualisasi (*Proxmox* VE). Pada *mode generic*, pemrosesan XDP terjadi setelah alokasi *sk_buff*, berbeda dengan mode *native* yang memproses sebelum *sk_buff* [5]. Konsekuensi dari pemilihan mode ini adalah *overhead* pemrosesan yang lebih tinggi dibandingkan mode *native* pada NIC fisik. Meskipun demikian, perbandingan relatif antara XDP dan *iptables* tetap *valid* karena keduanya diuji pada *environment* yang sama. *Throughput* yang terukur (919–932 Mbps pada skenario SYN *flood*) merepresentasikan kapasitas *environment* virtualisasi, bukan batas kinerja XDP *native*. Implementasi pada NIC fisik yang mendukung driver XDP *native* berpotensi menghasilkan kinerja yang lebih baik.

Berdasarkan keterbatasan tersebut dan temuan yang diperoleh, terdapat beberapa rekomendasi untuk penelitian selanjutnya. Pertama, implementasi XDP *mode native* pada NIC fisik yang mendukung *driver* XDP untuk mengukur kinerja aktual tanpa *overhead* virtualisasi. Kedua, pengembangan mekanisme pembaruan *threshold* secara otomatis berdasarkan analisis pola serangan secara *real-time*, misalnya dengan melatih ulang model *Random Forest* secara periodik dan memperbarui *config_map* berdasarkan *threshold* baru. Ketiga, pengujian pada skala *cluster* yang lebih besar dengan jumlah *node* dan variasi layanan yang lebih representatif terhadap lingkungan produksi. Keempat, evaluasi terhadap jenis serangan yang lebih beragam seperti *Slowloris*, DNS *amplification*, dan serangan *application-layer* untuk menguji keumuman pendekatan.

4. KESIMPULAN

Penelitian ini mengimplementasikan dan mengevaluasi *firewall* pre-*filtering* berbasis eBPF/XDP yang aturan deteksinya diturunkan dari model *Random Forest* untuk mendeteksi anomali trafik pada *Docker Swarm overlay network*. Model *Random Forest* yang dilatih menggunakan dataset CIC-DDoS-2019 mencapai akurasi 99,88%, *detection rate* 99,90%, *false positive rate* 0,14% dan ROC-AUC 0,9999. *Threshold* deteksi diturunkan dari analisis *feature importance* model dan disesuaikan untuk penerapan per-*source*-IP, menghasilkan SYN 50, UDP 350, dan PPS 3.000 paket per detik. Pada tujuh skenario pengujian, XDP berhasil men-*drop* lebih dari 99,9% paket serangan baik pada SYN *flood*, UDP *flood*, maupun *mixed traffic*, dengan penggunaan CPU berkisar 0,92%–1,18%, *throughput* yang stabil sekitar 920 Mbps dan *median response time* 0,68–0,69 ms di seluruh skenario, setara dengan kondisi tanpa serangan. Uji Mann-Whitney U menunjukkan bahwa perbedaan antar-skenario signifikan ($p < 0,05$), namun dengan *effect size* sangat kecil (Cohen's d

berkisar 0,02-0,23), sehingga dampak terhadap kualitas layanan sangat minimal. Dibandingkan *iptables*, baik *global rate limiting* maupun per-IP *hashlimit*, ketiga solusi memiliki efektivitas yang setara dalam mitigasi DDoS dengan *median response time* yang identik. Hasil penelitian ini menunjukkan bahwa pendekatan integrasi aturan *machine learning* ke dalam *kernel* melalui eBPF/XDP merupakan alternatif yang yang terbukti efektif untuk perlindungan infrastruktur kontainer terhadap anomali trafik, dengan keunggulan pada deteksi trafik hingga level per-*flow* dan fleksibilitas konfigurasi *runtime* melalui eBPF *config_map*.

REFERENCES

[1] I. M. Al Jawarneh et al., "Container Orchestration Engines: A Thorough Functional and Performance Comparison," in ICC 2019 - 2019 IEEE International Conference on Communications (ICC), IEEE, May 2019, pp. 1–6. doi: 10.1109/ICC.2019.8762053.

[2] Inc. Docker, "Docker Swarm Mode Overview," Docker Documentation. Accessed: Jan. 10, 2026. [Online]. Available: https://docs.docker.com/engine/swarm

[3] M. Mahalingam et al., "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks," Aug. 2014, RFC Editor. doi: 10.17487/RFC7348.

[4] M. Vieira, M. Castanho, R. Pacífico, E. Santos, E. Pinto, and L. Vieira, "Fast Packet Processing with eBPF and XDP: Concepts, Code, Challenges, and Applications," ACM Computing Surveys (CSUR), vol. 53, pp. 1–36, Feb. 2020, doi: 10.1145/3371038.

[5] T. Høiland-Jørgensen et al., "The eXpress data path: Fast programmable packet processing in the operating system kernel," in CoNEXT 2018 - Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies, Association for Computing Machinery, Inc, Dec. 2018, pp. 54–66. doi: 10.1145/3281411.3281443.

[6] D. Scholz, D. Raumer, P. Emmerich, A. Kurtz, K. Lesiak, and G. Carle, "Performance Implications of Packet Filtering with Linux eBPF," in 2018 30th International Teletraffic Congress (ITC 30), 2018, pp. 209–217. doi: 10.1109/ITC30.2018.00039.

[7] H. Yamada and R. Kawahara, "Evaluation of HTTP Request Anomaly Detection Model Using fastText and Convolutional Autoencoder," IEICE Communications Express, vol. 13, no. 7, pp. 240–243, May 2024, doi: 10.23919/comex.2024xbl0060.

[8] P. Schummer, A. del Rio, J. Serrano, D. Jimenez, G. Sánchez, and Á. Llorente, "Machine Learning-Based Network Anomaly Detection: Design, Implementation, and Evaluation," AI (Switzerland), vol. 5, no. 4, pp. 2967–2983, Dec. 2024, doi: 10.3390/ai5040143.

[9] L. Breiman, "Random Forests," Mach. Learn., vol. 45, no. 1, pp. 5–32, 2001, doi: 10.1023/A:1010933404324.

[10] by J. Ross Quinlan, M. Kaufmann Publishers, and S. L. Salzberg, "Programs for Machine Learning," 1994.

[11] M. Bachl, J. Fabini, and T. Zseby, "A flow-based IDS using Machine Learning in eBPF," Mar. 2022, [Online]. Available: http://arxiv.org/abs/2102.09980

[12] T. Farasat, J. Kim, and J. Posegga, "SmartX Intelligent Sec: A Security Framework Based on Machine Learning and eBPF/XDP," Oct. 2024, [Online]. Available: http://arxiv.org/abs/2410.20244

[13] N. ANAND, S. M. A, and P. K. Aakula, "High-performance Intrusion Detection Systemusing eBPF with Machine Learning algorithms," Jul. 06, 2023. doi: 10.21203/rs.3.rs-3140072/v1.

[14] Z. Chen, H. Kong, S. Ding, Q. Lv, and G. Wei, " Efficient DDoS Detection and Mitigation in Cloud Data Centers Using eBPF and XDP ," in 2024 IEEE 23rd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom) , Los Alamitos, CA, USA: IEEE Computer Society, Dec. 2024, pp. 1869–1874. doi: 10.1109/TrustCom63139.2024.00258.

[15] J. Nam, S. Lee, P. Porras, V. Yegneswaran, and S. Shin, "Secure Inter-Container Communications Using XDP/eBPF," IEEE/ACM Transactions on Networking, vol. 31, no. 2, pp. 934–947, Apr. 2023, doi: 10.1109/TNET.2022.3206781.

[16] S. Lin et al., "ONCache: A Cache-Based Low-Overhead Container Overlay Network," Jun. 2024, [Online]. Available: http://arxiv.org/abs/2305.05455

[17] Y. He et al., "Cross Container Attacks: The Bewildered eBPF on Clouds," in Proceedings of the 32nd USENIX Security Symposium, Anaheim, CA, USA, 2023, pp. 5971–5988.

[18] C. Lee, R. Yoshitani, and T. Hirotsu, "Enhancing Packet Tracing of Microservices in Container Overlay Networks using eBPF," in ACM International Conference Proceeding Series, Association for Computing Machinery, Dec. 2022, pp. 53–61. doi: 10.1145/3570748.3570756.

[19] M. Ţălu, "DDoS Mitigation in Kubernetes: A Review of Extended Berkeley Packet Filtering and eXpress Data Path Technologies," JUTI: Jurnal Ilmiah Teknologi Informasi, vol. 23, no. 2, pp. 60–73, 2025, doi: 10.12962/j24068535.v23i1.a1268.

[20] J. K. Lee, T. Hong, and G. Li, "Traffic and Overhead Analysis of Applied Pre-filtering ACL Firewall on HPC Service Network," Journal of Communications and Networks, vol. 23, no. 3, pp. 192–200, Jun. 2021, doi: 10.23919/JCN.2021.000011.

[21] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing Realistic Distributed Denial of Service (DDoS) Attack Dataset and Taxonomy," in 2019 International Carnahan Conference on Security Technology (ICCST), 2019, pp. 1–8. doi: 10.1109/CCST.2019.8888419.

[22] D. C. Montgomery, Design and Analysis of Experiments, 9th ed. John Wiley & Sons, 2017.