Skema Pendanaan: UTAMA

LAPORAN PENELITIAN

CNCSA: NOVEL SEQUENTIAL ALGORITHM UNTUK PENINGKATAN EFISIENSI KONFIGURASI JARINGAN BERBASIS NETMIKO PADA INFRASTRUKTUR VIRTUAL

TIM PENELITI

Ketua : Hillman Akhyar Damanik., M. Kom (190016)

Anggota : Merry Anggraeni., M. Kom (180071)

FAKULTAS TEKNOLOGI INFORMASI

UNIVERSITAS BUDI LUHUR

FEBRUARI 2026

HALAMAN PENGESAHAN
LAPORAN PENELITIAN

Judul Penelitian	: CNCSA: Novel Sequential Algorithm untuk Peningkatan Efisiensi Konfigurasi Jaringan Berbasis Netmiko pada Infrastruktur Virtual
Bidang Penelitian	: Ilmu Komputer/Teknik Informatika
Ketua Peneliti	
a. Nama Lengkap	: Hillman Akhyar Damanik
b. NIP/NIDN/ID-SINTA	: 190016/ 0320038704/ 6694077
c. Jabatan Fungsional	: Lektor
d. Program Studi	: Teknik Informatika
e. Nomor HP	: 082299099294
f. Alamat e-mail	: hillman.akhyardamanik@budiluhur.ac.id
Anggota Peneliti (1)	
a. Nama Lengkap	: Merry Anggraeni
b. NIP/NIDN/ID-SINTA	: 180071/0330127502/6645512
Anggota Peneliti (2)	
a. Nama Lengkap	: -
b. NIP/NIDN/ID-SINTA	: -
Mahasiswa (1)	
a. Nama Lengkap	: Hanif Rahman Ibrahim
b. NIM	: 2311501403
Mahasiswa (2)	
a. Nama Lengkap	: Fickry Imamsyah
b. NIM	: 2311500967
Lama Penelitian	: 6 bulan
Biaya Penelitian	
a. Sumber Universitas Budi Luhur	: Rp. 12.000.000
b. Sumber lain (sebutkan jika ada)	: Rp. -

Jakarta, 09 Februari 2026

Mengetahui,
Dekan/Kepala Pusat Studi

(Dr. Achmad Solichin, S. Kom., M.T.I)
NIP 050023

Ketua Pelaksana

(Hillman Akhyar Damanik)
NIP 190016

Menyetujui,
Direktur Riset dan Pengabdian Kepada Masyarakat

(Dr. Ir. Prudensius Maring, M.A.)
NIP 190043

No. Registrasi : 0 0 4 0 1 LPJ 0 2 2 6
Tanggal : 0 9 0 2 2 6 Paraf :

RINGKASAN

Automasi konfigurasi jaringan menjadi kebutuhan penting dalam pengelolaan infrastruktur jaringan modern yang semakin kompleks. Penelitian ini mengusulkan Centralized Network *Configuration Sequential Algorithm* (CNCSA) untuk mengoptimasi proses konfigurasi *mass deployment* pada router MikroTik melalui pendekatan sistematis dalam optimasi parameter *library* Netmiko. Metode penelitian menggunakan eksperimental dengan pengujian terhadap 15 router MikroTik *Cloud Hosted Router* versi 7.16 yang masing-masing dikonfigurasi dengan 150 baris perintah *firewall*. CNCSA menerapkan optimasi pada parameter koneksi (*timeout* 8s, *session_timeout* 45s, *keepalive* 30s) dan parameter pengiriman *command* (*delay_factor* 0.5, *max_loops* 300, *strip_prompt* dan *strip_command* aktif). Pengukuran *timing* granular dilakukan untuk tiga komponen utama yaitu *Connection Time*, *Send Time*, dan *Logging Time* dengan formula Te = Tc + Ts + Tl. Hasil pengujian menunjukkan CNCSA mencapai peningkatan performa 40.22 persen dengan Total Execution Time 302.49 detik dibandingkan *Standard* Netmiko *Sequential Method* yang memerlukan 506.05 detik. *Speedup* ratio mencapai 1.67x dengan *time saved* 203.56 detik per eksekusi. Analisis efisiensi menunjukkan *Configuration Efficiency* 98.25 persen, *Connection Efficiency* 1.48 persen, *Logging Efficiency* 0.28 persen, dan *System Overhead* Ratio hanya 0.0112 persen. Success rate 100 persen memvalidasi bahwa optimasi tidak mengurangi keandalan. CNCSA terbukti *applicable* untuk *production* environment dengan manfaat praktis yang signifikan terutama untuk administrator yang melakukan konfigurasi secara regular.

Kata Kunci: CNCSA, NSM, Netmiko, Sequential, CHR MikroTik,

PRAKATA

Puji dan syukur kami curahkan ke hadirat Tuhan Yang Maha Esa yang telah mencurahkan semua karunia sehingga diberikan keyakinan dan kemudahan dalam menyusun dan menyiapkan laporan penelitian ini dengan judul CNCSA: Novel Sequential Algorithm untuk Peningkatan Efisiensi Konfigurasi Jaringan Berbasis Netmiko pada Infrastruktur Virtual.

Pada kesempatan ini, peneliti ingin menyampaikan rasa terima kasih yang tak terhingga kepada:

1. Bapak Dr. Achmad Solichin, S. Kom., M.T.I sebagai Dekan Fakultas Teknologi Informasi Universitas Budi Luhur.
2. Bapak Dr. Ir. Prudensius Maring, M.A sebagai Direktur Penelitian dan Pengabdian Masyarakat Universitas Budi Luhur.
3. Semua pihak yang telah membantu penelitian ini, mohon maaf apabila ada kesalahan yang terucap dan terbersit, mohon dibukakan pintu maaf yang selapang-lapangnya.

Akhirnya dengan segala kerendahan hati, kami berharap agar penelitian ini dapat memberikan manfaat bagi para Siswa, Dosen, Mahasiswa, lingkungan kampus dan sekitarnya. Saran dan kritik yang membangun sangat diharapkan guna perbaikan penelitian mendatang.

Jakarta, 09 Februari 2026

Hillman Akhyar Damanik

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1

PENDAHULUAN

1.1. Latar Belakang dan Rumusan Masalah

Pertumbuhan infrastruktur jaringan modern telah mengakibatkan peningkatan kompleksitas sistem di berbagai sektor. Organisasi kini bergantung pada ratusan hingga ribuan perangkat jaringan yang harus dikonfigurasi dan dikelola secara konsisten dan efisien (1) (2) (3) (4). Namun, proses konfigurasi yang masih dilakukan secara manual seringkali mengakibatkan waktu eksekusi yang lama dan sangat rentan terhadap kesalahan manusia, yang dapat mengganggu stabilitas dan operasi jaringan. Kondisi ini menunjukkan bahwa otomatisasi konfigurasi jaringan telah menjadi persyaratan penting dalam tata kelola infrastruktur jaringan modern (5) (6). Studi tentang otomatisasi jaringan menunjukkan bahwa kompleksitas infrastruktur yang tinggi, integrasi sistem yang terbatas, dan tantangan dalam menerapkan standar tetap menjadi hambatan utama dalam manajemen jaringan modern (7) (8) (9). Kondisi ini mendorong urgensi pengembangan solusi otomatisasi yang lebih efisien untuk mengurangi beban konfigurasi berulang, meminimalkan potensi risiko keamanan, dan mendukung proses pengambilan keputusan berdasarkan metrik kinerja yang terukur dan komprehensif (10) (11).

Penelitian sebelumnya tentang otomatisasi jaringan menunjukkan bahwa kompleksitas infrastruktur yang tinggi dan tantangan dalam penggunaan praktis pustaka Netmiko tetap menjadi masalah utama dalam manajemen jaringan modern. Kondisi ini menyoroti kebutuhan mendesak akan solusi otomatisasi yang lebih efisien untuk mengurangi beban kerja konfigurasi yang berulang dan untuk meningkatkan efektivitas metrik kinerja yang komprehensif dan terukur (12) (13) (14). Lebih lanjut, meskipun standar Netmiko mendukung konektivitas multivendor, namun belum dilengkapi dengan mekanisme sistematis untuk mengevaluasi waktu eksekusi setiap komponen secara terukur. Penggunaan parameter default seperti timeout, delay_factor, dan max_loops, yang dirancang secara konservatif, dalam praktiknya menyebabkan peningkatan durasi eksekusi yang signifikan dalam lingkungan infrastruktur jaringan (15) (16). Tinjauan penelitian sebelumnya tentang otomatisasi jaringan mengungkapkan kesenjangan yang signifikan, khususnya dalam mengukur metrik waktu dan efisiensi, yang biasanya dilakukan dalam lingkungan simulator jaringan. Penelitian oleh Elezi et al. (15) berfokus pada verifikasi konfigurasi router menggunakan jaringan emulator, tetapi tidak menyertakan metrik evaluasi kinerja kuantitatif. Wusu et al. (17) membahas perbandingan durasi proses pencadangan konfigurasi, tetapi tidak menguraikan analisis rinci komponen waktu untuk setiap perangkat router. Sementara itu, Imoukhuede et al. (18) membatasi pengukuran mereka pada waktu koneksi dan pengiriman konfigurasi, dan melakukan pengukuran tersebut dalam lingkungan emulator. Singh et al. (19) mengimplementasikan kerangka kerja Ansible tanpa menyajikan hasil pengukuran numerik yang dapat digunakan untuk mengevaluasi kinerja secara objektif. Studi yang dilakukan oleh Mazin et al. (20)

membandingkan pendekatan manual dan otomatis dengan membatasi evaluasi hanya pada pengukuran total waktu eksekusi.

Berdasarkan latar belakang penelitian yang telah dijelaskan di atas, penelitian ini berfokus pada identifikasi beberapa isu kunci. Penelitian ini bertujuan untuk menganalisis parameter dalam standar Netmiko yang memiliki pengaruh paling signifikan terhadap waktu eksekusi, serta menentukan nilai parameter koneksi dan pengiriman perintah yang paling optimal tanpa mengurangi keandalan proses konfigurasi. Selain itu, penelitian ini merancang metode untuk mengukur metrik waktu eksekusi dengan memecahnya menjadi beberapa komponen yang membentuk total waktu eksekusi. Penelitian ini juga mengevaluasi besarnya peningkatan kinerja yang dihasilkan dari pengoptimalan metrik waktu dan efisiensi dibandingkan dengan pendekatan standar Netmiko berdasarkan pengukuran waktu yang terukur. Lebih lanjut, penelitian ini merumuskan model matematika untuk menganalisis hubungan antara komponen waktu dan menghitung metrik waktu dan efisiensi yang dioptimalkan sebagai dasar untuk analisis komparatif. Terakhir, penelitian ini menguji efektivitas komparatif otomatisasi konfigurasi jaringan berbasis eksekusi sekuensial dan konkuren dengan mempertimbangkan metrik efisiensi, tingkat keberhasilan, dan tingkat kesalahan. Pendekatan pemecahan masalah dalam penelitian ini dilakukan melalui beberapa fase terstruktur. Fase awal berfokus pada identifikasi dan klasifikasi parameter standar Netmiko, khususnya yang terkait dengan pengaturan koneksi. Selanjutnya, kondisi dasar ditetapkan dengan mengimplementasikan Netmiko dengan parameter default-nya untuk berfungsi sebagai referensi objektif untuk perbandingan. Fase berikutnya melibatkan implementasi Algoritma Konfigurasi Jaringan Terpusat Berurutan (CNCSA), sebuah algoritma otomatisasi untuk konfigurasi router yang menguraikan waktu eksekusi menjadi beberapa komponen kunci, yaitu Total Waktu Eksekusi (T_e), Total Waktu Koneksi (T_c), Total Waktu Pengiriman (T_s), dan Total Waktu Pencatatan (T_l). Pendekatan ini memberikan visibilitas yang lebih detail terhadap kinerja router dan memfasilitasi optimasi yang lebih efektif. Selain itu, CNCSA diintegrasikan ke dalam platform berbasis web menggunakan Django untuk mengembangkan sistem manajemen terpusat dengan kemampuan pemantauan dan pencatatan waktu nyata untuk setiap router. Tahap selanjutnya merumuskan model matematika untuk menganalisis hubungan antar komponen waktu dan untuk menghitung metrik terkait waktu dan efisiensi. Terakhir, pengujian eksperimental dilakukan pada 15 Cloud Hosted Router (CHR) dengan beban kerja 150 baris perintah IP Firewall Mangle, IP Firewall NAT, dan IP Firewall Filter, menerapkan metrik terintegrasi di bawah eksekusi berurutan dan melakukan analisis komparatif efisiensi, tingkat keberhasilan, dan tingkat kesalahan.

1.2. Pendekatan Pemecahan Masalah

Sebagai upaya penyelesaian permasalahan yang telah dipetakan diatas, penelitian ini mengadopsi metode yang terperinci dan berbasis pengukuran kuantitatif terhadap komponen waktu eksekusi pada perangkat router. Pertama, dilakukan pengembangan CNCSA, yaitu algoritma otomasi jaringan pada perangkat router dengan pembagian komponen waktu eksekusi menjadi empat

bagian, yaitu *Total Execution Time* (Te), Total *Connection Time* (Tc), *Total Send Time* (Ts), dan *Total Logging Time* (Tl). Pendekatan ini memberikan visibilitas granular terhadap performa perangkat router dan mempermudah proses optimasi. Kedua, algoritma CNCSA diintegrasikan ke dalam platform berbasis web menggunakan Django, untuk mengembangkan antarmuka manajemen terpusat dengan kemampuan monitoring dan pencatatan data secara real-time dari distribusi masing-masing perangkat router. Ketiga, dilakukan pengujian kinerja (*performance benchmarking*) di lingkungan virtualisasi Proxmox VE, sebagai dasar pembandingan dengan metode Netmiko Standard Method (NSM) untuk mengukur peningkatan efisiensi dan stabilitas algoritma yang dikembangkan.

1.3. State of the Art dan Kebaruan

Aspek kebaruan (*state of the art*) dari penelitian ini terletak pada pengembangan segmentasi granular *time component analysis* dalam konteks otomasi konfigurasi router CHR berbasis web, yang belum banyak dibahas dalam penelitian terdahulu. Pertama, diterapkan pendekatan *granular time component analysis* dengan membagi waktu eksekusi konfigurasi router ke dalam tiga komponen terukur, yaitu *Total Connection Time* (Tc), *Total Send Time* (Ts), dan *Total Logging Time* (Tl). Ketiga parameter tersebut selanjutnya akan digunakan untuk menghitung Total Execution Time (Te) sebagai indikator performa utama dari algoritma CNCSA. Segementasi ini memungkinkan proses optimasi yang lebih terarah serta identifikasi hambatan secara presisi. Kedua, dikembangkan algoritma sekuensial yang terintegrasi dengan sistem manajemen berbasis web menggunakan kerangka kerja django. Ketiga, pengujian dilakukan pada lingkungan virtualisasi berbasis Proxmox VE, yang menghasilkan *baseline* untuk perbandingan CNCSA dan NSM.

1.4. Peta Jalan (Road Map) Penelitian

Sebagai bagian dari peta jalan penelitian lima tahun ke depan, penelitian saat ini merupakan fase awal dari rangkaian pengembangan sistem otomasi jaringan berbasis algoritma adaptif. Pada tahun pertama, penelitian difokuskan pada tahap pengembangan dan validasi algoritma CNCSA sebagai fondasi utama sistem otomasi pada perangkat router. Pengembangan meliputi analisis kebutuhan, pengembangan struktur algoritma berbasis waktu eksekusi terukur (Te, Tc, Ts, Tl), serta integrasi awal dengan framework django untuk memastikan kestabilan dan pengukuran proses konfigurasi. Pada tahun kedua, penelitian difokuskan pada pengembangan sistem manajemen jaringan berbasis multivendor yang mencakup perangkat juniper dan cisco. Tahun ketiga, fokus diarahkan pada integrasi teknik pembelajaran mesin untuk mendeteksi pola eksekusi konfigurasi dan melakukan optimasi prediktif terhadap parameter pada perangkat router. Tahun keempat difokuskan pada tahap implementasi sistem di lingkungan nyata (production network) dengan melibatkan infrastruktur berskala menengah hingga besar. Tahap ini juga mencakup integrasi dengan sistem keamanan berbasis Intrusion Detection System (IDS) menggunakan Wazuh atau Suricata, untuk menghadirkan model otomasi jaringan yang tidak hanya efisien tetapi juga adaptif terhadap potensi

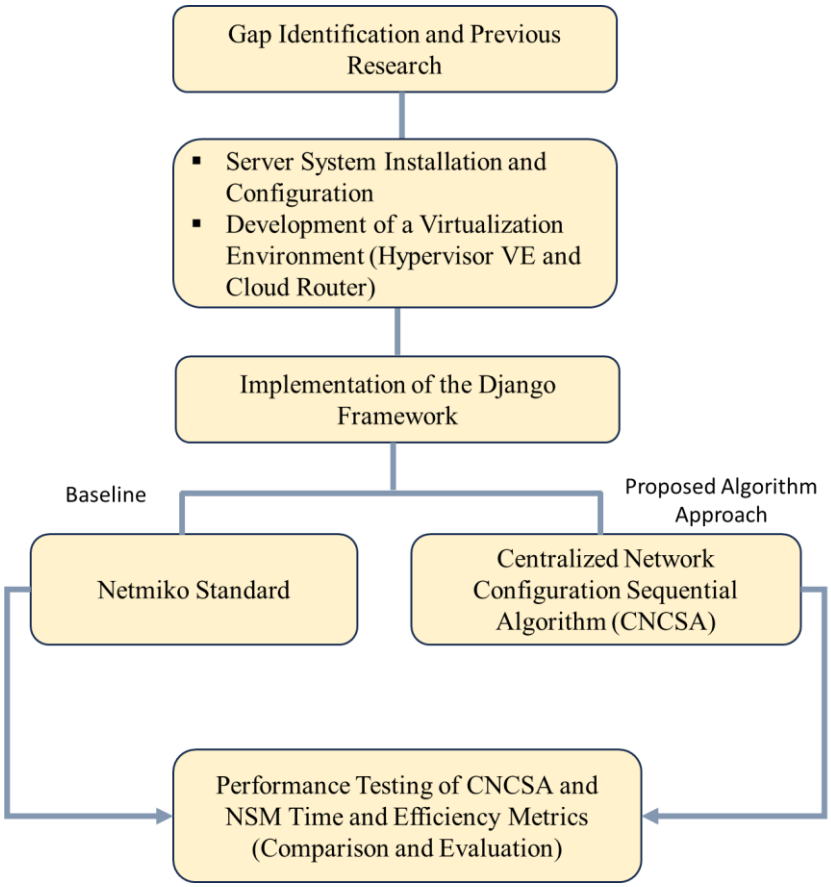
ancaman keamanan. Tahun kelima merupakan tahap finalisasi dan pengembangan lanjutan menuju sistem otomasi jaringan cerdas berbasis Software-Defined Networking (SDN). Pada fase ini, CNCSA akan dikembangkan menjadi sistem manajemen konfigurasi otonom yang mampu melakukan orkestrasi jaringan secara otomatis dengan pengendalian tersentralisasi.

BAB 2

METODE PENELITIAN

2.1. Metode Penelitian

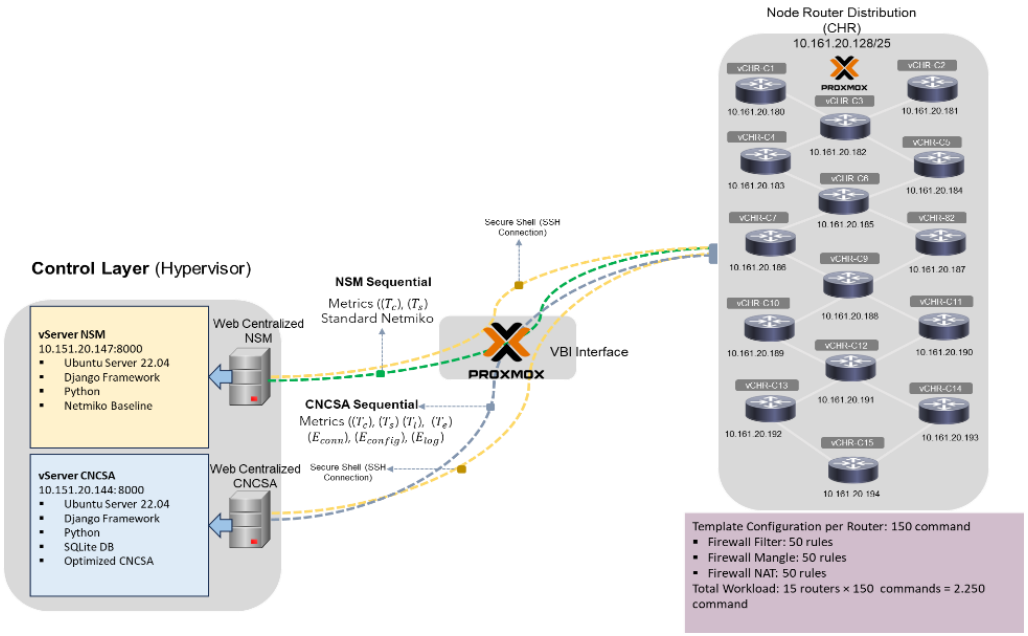
Pendekatan penelitian ini menggunakan metode kuantitatif eksperimen dan berfokus pada pengujian kinerja algoritma otomasi konfigurasi pada perangkat cloud router bernama *Centralized Network Configuration Sequential Algorithm* (CNCSA). Algoritma ini dirancang untuk mengatasi permasalahan pada proses konfigurasi massal router secara terpusat dan terukur melalui sistem berbasis web. Gambar 2.1 merupakan diagram alur penelitian untuk langkah-langkah utama dalam pengembangan dan evaluasi algoritma CNCSA sebagai solusi otomasi konfigurasi jaringan berbasis web. Setiap tahap disusun secara berurutan agar proses penelitian berjalan terarah dan hasil yang diperoleh dapat diukur secara objektif.



Gambar 2.1. Tahapan Penelitian Pengembangan Algoritma CNCSA

Tahap awal dari penyusunan laporan dilakukan sebagai dasar awal sebelum proses perencanaan dan pengembangan sistem dijalankan. Fokus utama adalah

menentukan lokasi penelitian yang bertempat di AJN Solusindo, berdasarkan kesesuaian infrastruktur dan kebutuhan uji coba sistem otomasi terpusat. Selanjutnya tahap perancangan sistem dan algoritma CNCSA, di mana struktur logika dikembangkan menggunakan bahasa pemrograman python yang terintegrasi dengan framework Django. Django dipilih karena kemampuannya dalam mengelola struktur aplikasi secara modular, mendukung penyimpanan data berbasis ORM serta dapat mengimplementasikan sistem pemantauan dan pencatatan log secara waktu nyata. Tahap berikutnya adalah pembangunan lingkungan uji (*testbed*) berbasis pada virtualisasi Proxmox *Virtual Environment* (PVE). Platform ini menyediakan infrastruktur bagi server aplikasi django dan 15 unit Cloud Hosted Router (CHR) yang dijalankan sebagai mesin virtual independen. Masing-masing router menerima 150 baris konfigurasi, meliputi 50 aturan *IP Firewall Mangle*, 50 *IP Firewall NAT*, dan 50 *IP Firewall Filter*. Konfigurasi yang dikirimkan ini dimaksudkan agar setiap perangkat memperoleh beban konfigurasi yang setara sehingga perbandingan hasil uji dapat dilakukan secara objektif antara CNCSA dengan NSM. Konektivitas dan komunikasi antara server django dan perangkat CHR dijalankan melalui protokol secure shell menggunakan library netmiko, dan bertugas menjalin koneksi, mengirim perintah konfigurasi, dan mencatat hasil eksekusi seperti pada gambar 2.2.



Gambar 2.2. Arsitektur dan Topologi Sistem CNCSA

Arsitektur sistem mengadopsi konsep manajemen jaringan berbasis web terpusat, di mana seluruh konfigurasi dan proses waktu koneksi, waktu pengiriman, logging dan total eksekusi router dikendalikan melalui antarmuka web yang dapat diakses oleh administrator jaringan. Ketika pengguna mengirimkan perintah konfigurasi melalui antarmuka web, django akan mengeksekusi logika di sisi backend dan mengatur proses pengiriman konfigurasi secara sekuensial ke setiap perangkat router CHR. Setiap tahapan dievaluasi berdasarkan tiga komponen waktu

utama yaitu 1) Waktu Koneksi (Tc) – durasi yang dibutuhkan untuk menjalin koneksi SSH ke perangkat Router. 2) Waktu Pengiriman (Ts) – durasi eksekusi seluruh baris konfigurasi. 3) Waktu *Logging* (Tl) – waktu yang digunakan untuk mencatat hasil konfigurasi ke basis data. 4) Ketiga parameter tersebut digunakan untuk menghitung *Total Execution Time* (Te) sebagai indikator performa utama dari pengembangan algoritma CNCSA. Kemudian sebagai *benchmarking* penelitian, akan memvalidasi efektivitas pendekatan yang dikembangkan, dengan Netmiko Standard Method (NSM). Metode ini menjalankan konfigurasi dengan cara tradisional tanpa optimasi waktu dan tanpa pembagian komponen eksekusi yang terstruktur ke masing-masing perangkat router. Dengan adanya pembanding, analisis perbandingan dapat dilakukan secara kuantitatif untuk menilai efisiensi masing-masing komponen kategori yang dihasilkan oleh CNCSA terhadap pendekatan standard.

Gambar 2.2 menjelaskan secara umum untuk komunikasi antara virtuliasisasi dan perangkat cloud router melalui Virtual Bridge *Interface* (VBI) yang menghubungkan dua segmen jaringan berbeda. Server Django yang memiliki alamat IP 10.151.20.144/25 terhubung melalui proxmox ke gateway jaringan dengan IP 10.151.20.230/25, koneksi untuk subnet router CHR dengan subnet 10.161.20.128/25. Dari struktur jaringan yang dimodelkan bertujuan untuk mempermudah pengujian terfokus terhadap performa pengiriman konfigurasi berbasis waktu dan efisiensi. Kemudian pada web CNCSA akan mencatat waktu yang dibutuhkan pada setiap tahap eksekusi, meliputi Total Waktu Koneksi (Tc), Total Waktu Pengiriman (Ts), *Total Logging Time* (Tl), dan Total Waktu Eksekusi (Te) untuk masing-masing perangkat cloud router. Dengan arsitektur ini, setiap proses konfigurasi cloud router dapat dimonitor dan dianalisis secara kuantitatif untuk mengevaluasi efektivitas algoritma CNCSA.

Luaran tahap ini adalah sistem otomasi konfigurasi jaringan yang mampu melakukan pengiriman perintah dan pencatatan waktu secara terstruktur. Setelah implementasi sistem, dilakukan pengujian kinerja dan pengumpulan data eksperimen. Pada tahap ini, CNCSA dievaluasi berdasarkan empat parameter waktu utama, yaitu Connection Time (Tc), Send Time (Ts), Logging Time (Tl), dan Total Execution Time (Te). Sebagai pembanding, digunakan Netmiko *Standard Method* (NSM) yang menjalankan konfigurasi jaringan secara sekuensial dengan parameter default tanpa optimasi. Penggunaan metode pembanding ini bertujuan untuk memvalidasi efektivitas optimasi yang diterapkan pada CNCSA secara kuantitatif dan objektif. Tahap akhir penelitian adalah analisis dan evaluasi hasil pengujian. Data waktu eksekusi yang diperoleh dianalisis untuk membandingkan efisiensi CNCSA terhadap metode standar NSM, baik dari sisi waktu total, rata-rata waktu per perangkat, maupun tingkat keberhasilan konfigurasi. Indikator capaian pada tahap ini adalah diperolehnya kesimpulan berbasis data mengenai peningkatan performa dan efisiensi konfigurasi jaringan yang dihasilkan oleh algoritma CNCSA. Hasil analisis ini menjadi dasar dalam mengevaluasi kelayakan CNCSA sebagai solusi otomasi konfigurasi jaringan terpusat berbasis web.

BAB 3

HASIL PELAKSANAAN PENELITIAN

3.1. Desain Penelitian dan Pendekatan Metode

3.1.1. Model Penelitian

Penelitian ini menggunakan pendekatan kuantitatif dengan metode eksperimen perbandingan. Tujuan utama penelitian adalah untuk mengoptimalkan proses konfigurasi jaringan massal menggunakan library Python Netmiko melalui pengembangan algoritma CNCSA (Centralized Network Configuration Sequential Algorithm). Penelitian ini membandingkan dua metode konfigurasi:

1) CNCSA (*Optimized*): Metode yang dikembangkan dengan berbagai optimasi parameter
2) Standard *Library* Netmiko (*Baseline*): Metode standar tanpa optimasi sebagai pembanding

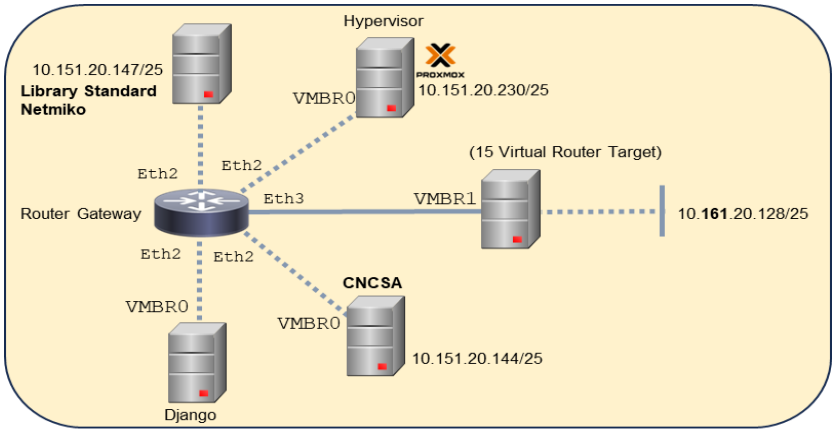
Pendekatan eksperimen ini dipilih karena memungkinkan pengukuran yang terukur dan dapat diulang untuk mengevaluasi efektivitas optimasi yang dilakukan. Seluruh data dikumpulkan dalam bentuk angka (waktu dalam detik) sehingga dapat dibandingkan secara objektif. Karakteristik penelitian ini:

1) Terukur: Semua variabel (waktu koneksi, waktu konfigurasi, waktu logging) diukur dalam satuan detik.
2) Dapat diulang: Prosedur eksperimen dapat dilakukan kembali dengan hasil yang konsisten
3) Objektif: Pengukuran menggunakan timer otomatis dari Python, bukan penilaian subjektif
4) Dapat diverifikasi: Hasil dapat dicek ulang melalui formula matematis

Penelitian ini bukan menguji hipotesis, melainkan mengusulkan dan mengevaluasi algoritma optimasi untuk meningkatkan kecepatan konfigurasi jaringan.

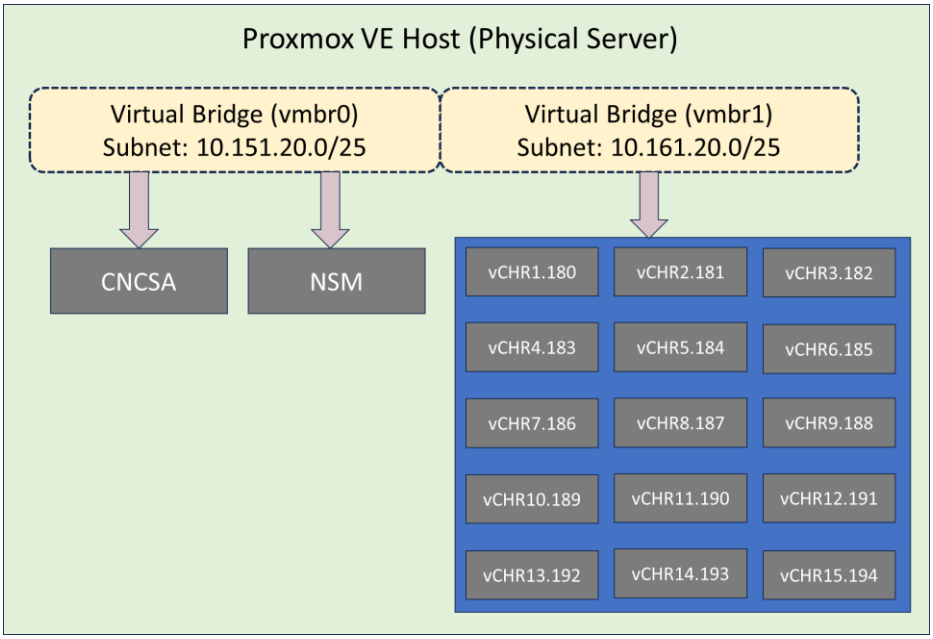
Arsitektur dan Topologi Sistem

Gambar 3.1 merupakan arsitektur sistem dan topologi yang dibangun dalam penelitian ini mengadopsi pendekatan virtualisasi dengan hypervisor proxmox terpusat dengan manajemen konfigurasi berbasis web. Seluruh infrastruktur pengujian berjalan di atas platform Proxmox *Virtual Environment* (PVE) versi 6.3-2 yang berperan sebagai hypervisor untuk mengelola perangkat virtual. Arsitektur ini terdiri dari tiga komponen utama yaitu server aplikasi CNCSA sebagai *controller* yang menjalankan algoritma optimasi konfigurasi perangakat router, server aplikasi Standard Netmiko yang digunakan sebagai baseline perbandingan, dan 15 unit MikroTik *Cloud Hosted Router* (CHR) sebagai target konfigurasi yang disimulasikan secara virtual.



Gambar 3.1. Arsitektur Sistem dan Topologi

Struktur komunikasi arsitektur untuk penelitian ini bersifat client-server dengan pola *centralized* control seperti divisualisasikan pada gambar 3.2. Penulis jaringan mengakses antarmuka web CNCSA melalui browser dengan mengakses alamat http://10.151.20.144:8000. Setelah login, penulis memilih perangkat target dari daftar 15 CHR yang tersedia dan menginputkan perintah konfigurasi yang akan dikirim. Request dari browser diterima oleh django framework yang menjalankan logika pada modul *views.py*, khususnya fungsi *configure*(). Setelah request divalidasi, Django memanggil *library* netmiko untuk membuka koneksi SSH ke setiap perangkat CHR secara *sequential* (satu per satu).



Gambar 3.2. Arsitektur Sistem dan Topologi Centralized Control

Proses ini mencakup tiga fase utama yang diukur waktunya: fase koneksi SSH (Tc) untuk *establish connection*, fase pengiriman command (Ts) untuk mengirim dan eksekusi 150 baris konfigurasi firewall, dan fase *logging* (Tl) untuk

menyimpan hasil ke database SQLite. Setiap fase diukur dengan presisi menggunakan timer python `time.time()` untuk menghasilkan data performa yang akurat hingga level *microsecond*.

3.1.2. Arsitektur Aplikasi Web Django

Pengembangan CNCSA dibangun menggunakan Django *Framework* versi 5.2.3 dengan bahasa pemrograman Python 3.12.4. Django dipilih karena menyediakan struktur *Model-View-Template* (MVT) yang memisahkan aturan kerja utama aplikasi, representasi data, dan tampilan antarmuka secara jelas, sehingga memudahkan pengembangan dan maintenance sistem. Arsitektur aplikasi terdiri dari dua direktori utama yang mengikuti best practice django *project structure*. Direktori pertama bernama *project_network_automation* yang berfungsi sebagai pusat (root) dari proyek Django dan menyimpan seluruh pengaturan utama aplikasi. Di dalam folder ini terdapat file `settings.py` yang digunakan untuk mengatur koneksi database menggunakan SQLite, daftar aplikasi yang dipakai (*installed apps*), *middleware*, serta pengaturan keamanan. File `urls.py` pada tingkat proyek bertugas mengatur alamat (*routing*) utama dan meneruskannya ke URL yang dimiliki oleh setiap aplikasi di dalam proyek. Selanjutnya, file `wsgi.py` dan `asgi.py` digunakan sebagai jalur masuk (*entry point*) ketika aplikasi akan dijalankan pada lingkungan produksi, meskipun pada penelitian ini aplikasi masih dijalankan menggunakan development server. Selain itu, terdapat file requirements.txt yang berisi daftar seluruh library Python yang diperlukan agar aplikasi dapat berjalan, seperti Django, Netmiko, dan pustaka pendukung lainnya.

Direktori kedua adalah *network_automation* yang berfungsi sebagai aplikasi utama dalam Django dan menjadi tempat seluruh proses kerja CNCSA dijalankan. Di dalam folder ini terdapat beberapa file penting yang menjadi bagian inti dari sistem. File *models.py* digunakan untuk mendefinisikan tiga model database utama dengan memanfaatkan Django ORM (*Object Relational Mapping*). Model D*evice* digunakan untuk menyimpan data perangkat jaringan, yang meliputi: ip_address sebagai alamat IP, hostname sebagai nama perangkat, *username* untuk akses SSH, *encrypted_password* untuk menyimpan kata sandi yang sudah dienkripsi, ssh_port dengan nilai default 22, *remote_name* sebagai nama *remote* CHR, *location* untuk menunjukkan lokasi fisik atau logis perangkat, serta vendor yang berisi pilihan jenis vendor dan saat ini hanya menggunakan MikroTik. Model Log berfungsi menyimpan hasil dari proses konfigurasi yang dijalankan. Field yang terdapat pada model Log digunakan untuk mencatat seluruh aktivitas konfigurasi perangkat. Field target menunjukkan perangkat tujuan, action menjelaskan jenis proses yang dijalankan, status menyatakan apakah proses berhasil atau gagal, time mencatat waktu pelaksanaan, message menyimpan hasil dari perintah yang dikirim, dan duration menunjukkan lama waktu proses tersebut berlangsung. File `views.py` merupakan bagian utama aplikasi yang berisi penerapan algoritma CNCSA. Di

dalamnya terdapat fungsi `configure (request)` yang menerima permintaan HTTP POST dari form konfigurasi, memeriksa apakah data yang dimasukkan sudah benar, lalu menjalankan algoritma CNCSA secara berurutan pada setiap perangkat yang dipilih. Pada fungsi ini juga dilakukan pengukuran waktu untuk setiap tahap secara tepat. Penghitungan waktu dimulai dari `conn_start = time.time()` sebelum proses koneksi SSH, kemudian nilai *Connection Time* diperoleh dari `tc_i = time.time() - conn_start`. Proses yang sama juga diterapkan untuk menghitung *Send Time* dan *Logging Time*. Setelah seluruh perangkat selesai diproses, fungsi akan menjumlahkan seluruh nilai waktu (T_c, T_s, dan T_l), kemudian menghitung T_e serta rasio efisiensi, lalu menyimpan semua data pengukuran tersebut ke dalam session Django agar dapat ditampilkan di halaman template. Sementara itu, file `urls.py` pada tingkat aplikasi berfungsi untuk mengatur jalur akses (*routing*) ke setiap fitur yang tersedia. Dalam penelitian ini, pola URL yang digunakan meliputi: `path (views.home)` untuk halaman utama, path `(devices/, views.devices)` untuk pengelolaan perangkat, `path (configure/, views.configure)` sebagai halaman utama konfigurasi yang menjalankan CNCSA, dan path `(log/, views.log)` untuk melihat riwayat hasil konfigurasi. Pola URL lainnya seperti `verify_config, backup, restore`, dan `send_config` tidak dibahas lebih lanjut karena tidak termasuk dalam ruang lingkup evaluasi performa CNCSA.

Direktori templates digunakan untuk menyimpan file HTML yang ditampilkan sebagai antarmuka web oleh Django. File `base.html` berisi tampilan dasar yang mencakup navigation bar, CSS dari Bootstrap, serta JavaScript, lalu digunakan sebagai kerangka utama oleh halaman lainnya. File `config.html` merupakan halaman utama untuk menjalankan konfigurasi menggunakan CNCSA. Di dalamnya tersedia pilihan perangkat dalam bentuk checkbox, bagian input perintah untuk setiap perangkat yang ditampilkan dalam bentuk accordion, serta area untuk menampilkan hasil pengukuran waktu (T_c, T_s, T_l, T_e, ε, dan rasio efisiensi) setelah proses selesai. File `devices.html` menampilkan daftar perangkat dalam bentuk tabel yang dilengkapi dengan fitur untuk menambah, mengubah, dan menghapus data perangkat. Sementara itu, file `log.html` digunakan untuk menampilkan riwayat hasil konfigurasi dalam bentuk tabel yang dilengkapi dengan fungsi pencarian dan penyaringan data.

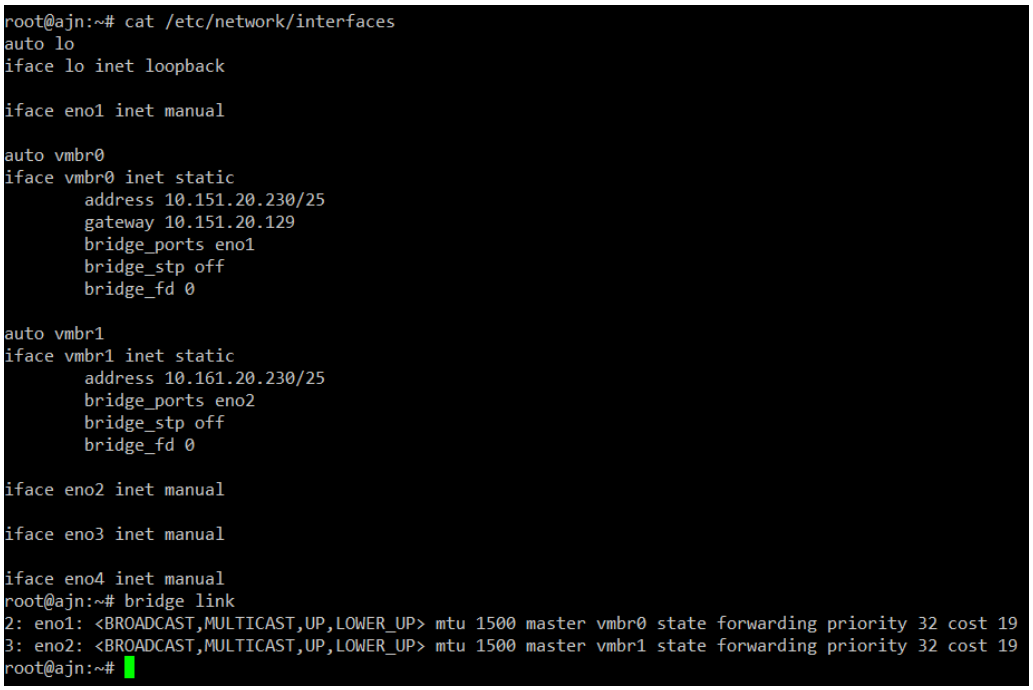
Dalam pengembangan deployment, aplikasi Django dijalankan menggunakan development server dengan command python3 manage.py runserver 10.151.20.144:8000. Development server dipilih karena penelitian ini fokus pada evaluasi performa algoritma, bukan pada production deployment. Server listen pada semua interface dengan port 8000, sehingga dapat diakses dari browser di jaringan yang sama. Database yang digunakan adalah SQLite yang berjalan di server yang sama dengan aplikasi Django. File database SQLite (db.sqlite3) disimpan di root directory project dan dikelola sepenuhnya oleh Django ORM. SQLite dipilih karena

simplicity dan tidak memerlukan setup database server terpisah, yang cukup untuk kebutuhan penelitian dengan scale 15 router CHR.

3.1.3. Topologi Jaringan Pengujian

Topologi jaringan untuk pengujian dibuat menggunakan sistem isolated testbed, yaitu lingkungan uji yang terpisah dari jaringan luar, sehingga hasil pengukuran tidak dipengaruhi oleh lalu lintas jaringan lain atau perubahan kondisi jaringan di luar sistem uji. Seluruh infrastruktur dijalankan pada sebuah server fisik dengan spesifikasi Intel Xeon Bronze 3104 @ 1.70GHz yang memiliki 6 core dan 6 thread, serta menggunakan Proxmox VE sebagai hypervisor. Prosesor tersebut sudah mendukung teknologi virtualisasi Intel VT-x, sehingga proses virtualisasi dapat berjalan lebih optimal. Server ini juga memiliki kapasitas memori yang mencukupi untuk menjalankan 17 mesin virtual secara bersamaan (terdiri dari 2 Ubuntu Server dan 15 CHR) tanpa terjadi penurunan performa yang berarti akibat kekurangan memori.

Topologi jaringan dibangun menggunakan konsep dua subnet yang berbeda dengan satu gateway utama. Subnet pertama (10.151.20.128/25) digunakan sebagai jaringan manajemen, tempat server CNCSA dan server Standard Netmiko berada. Kedua server ini bisa diakses langsung oleh penulis untuk keperluan pengembangan dan pengujian. Subnet kedua (10.161.20.128/25) digunakan khusus untuk perangkat CHR yang menjadi target konfigurasi. Pemisahan jaringan ini dibuat untuk meniru kondisi di dunia nyata, di mana jaringan manajemen biasanya dipisahkan dari jaringan produksi demi keamanan dan agar lalu lintas data tidak bercampur. Kedua subnet tersebut dihubungkan menggunakan router fisik MikroTik CCR 1306 yang berfungsi sebagai perutean data antar jaringan. Router ini memiliki dua antarmuka jaringan. Antarmuka pertama menggunakan IP 10.151.20.129/25 sebagai gateway untuk jaringan manajemen, sedangkan antarmuka kedua menggunakan IP 10.161.20.129/25 sebagai gateway untuk jaringan CHR. Tabel routing pada router diatur agar data dapat berpindah antar subnet. Dengan konfigurasi ini, server CNCSA yang berada di IP 10.151.20.144 dapat melakukan koneksi SSH ke perangkat CHR pada rentang IP 10.161.20.180–10.161.20.194. Penggunaan router fisik, bukan router virtual, dipilih agar topologi lebih mendekati kondisi sebenarnya dan tetap menghadirkan latensi jaringan yang realistis meskipun sangat kecil. Pada Proxmox, digunakan dua buah Virtual Bridge untuk menghubungkan mesin virtual dengan jaringan fisik. Virtual Bridge `vmbr0` diatur untuk subnet 10.151.20.230/25 dan menghubungkan server Ubuntu (CNCSA dan Standard) ke router gateway melalui kartu jaringan fisik pada server Proxmox. Virtual Bridge `vmbr1` diatur untuk subnet 10.161.20.230/25 dan digunakan untuk menghubungkan 15 CHR ke router gateway. Kedua virtual bridge ini bekerja pada Layer 2 (data link) untuk meneruskan frame Ethernet antara mesin virtual dan interface jaringan fisik milik Proxmox, seperti pada gambar 3.3.



Gambar 3.3. Virtual Bridge Proxmox (VMBR0 dan VMBR1)

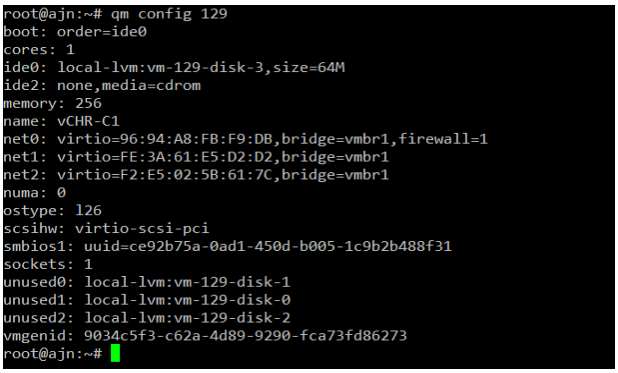
Setiap CHR memiliki tiga interface jaringan (Ether1, Ether2, Ether3) yang semuanya terhubung ke `vmbr1`, sehingga dapat mensimulasikan kondisi router dengan banyak antarmuka. Namun, untuk proses manajemen dan koneksi SSH, hanya interface Ether1 yang digunakan, untuk IP address pada rentang 10.161.20.180–10.161.20.194. Dua interface lainnya (Ether2 dan Ether3) hanya disiapkan untuk kemungkinan pengujian lanjutan, seperti multi-homing atau link cadangan, tetapi belum digunakan dalam penelitian ini.

Latency jaringan antara server Django dan CHR dibuat sangat kecil karena semua komunikasi terjadi di satu server fisik melalui virtual bridge. Dari hasil pengujian menggunakan perintah ping, nilai RTT antara 10.151.20.144 dan 10.161.20.180–10.161.20.194 berada di kisaran 0,5–2 ms, yang tergolong rendah dan stabil. Dengan demikian, perbedaan waktu yang tercatat dalam percobaan benar-benar berasal dari proses algoritma CNCSA (proses koneksi, eksekusi perintah, dan pencatatan log), bukan karena lambatnya jaringan atau kehilangan paket.

3.1.4. Spesifikasi Perangkat MikroTik Cloud Host Router (CHR)

Penelitian ini menggunakan MikroTik Cloud Hosted Router (CHR) sebagai perangkat yang menjadi target konfigurasi. CHR merupakan versi virtual dari RouterOS MikroTik yang memang dibuat untuk dijalankan di lingkungan virtual seperti VMware, KVM, Hyper-V, dan Proxmox. Jika dibandingkan dengan router fisik, CHR lebih fleksibel untuk proses instalasi dan pengujian, serta lebih hemat biaya karena tidak memerlukan perangkat keras tambahan. Setiap CHR menjalankan RouterOS versi 7.x dengan lisensi Free Tier yang membatasi

kecepatan maksimum hingga 1 Mbps. Pembatasan ini tidak memengaruhi penelitian karena yang diukur bukanlah kecepatan lalu lintas data, melainkan waktu yang dibutuhkan untuk menjalankan perintah konfigurasi**.** Lisensi Free dipilih karena sudah cukup untuk melakukan pengujian konfigurasi, seperti penerapan aturan firewall, tanpa membutuhkan lalu lintas jaringan yang tinggi. Penggunaan sumber daya pada setiap CHR dibuat sama dan konsisten agar perbandingan hasil tetap adil dan tidak dipengaruhi oleh perbedaan spesifikasi mesin virtual, seperti visualisasi gambar 3.4.



Gambar 3.4. MikroTik Cloud Hosted Router (CHR) - vCHR-C1

Berdasarkan pengaturan di Proxmox (qm config 129), setiap CHR memperoleh alokasi resource yang identik, yaitu:

a) Name: vCHR-C1
b) CPU: 1 core, 1 socket (total 1 vCPU)
c) Memory: 256 MB RAM
d) Disk: 64 MB (ide0)
e) Network: 3 network interfaces (net0, net1, net2) dengan virtio driver, semua terhubung ke vmbr1

Resource yang digunakan sudah cukup untuk menjalankan RouterOS dan memproses sekitar 150 baris aturan firewall tanpa penurunan performa yang berarti. Kapasitas memori sebesar 256 MB merupakan batas minimal yang direkomendasikan untuk CHR dalam menjalankan fungsi routing dasar dan firewall. Penggunaan driver jaringan virtio dipilih karena lebih efisien dan memberikan kinerja jaringan yang lebih baik dibandingkan e1000 atau rtl8139 pada lingkungan virtual. Setiap CHR dikonfigurasi dengan user admin dan password yang telah diamankan menggunakan enkripsi simetris Fernet. Layanan SSH diaktifkan pada port 22 agar dapat menerima koneksi dari server Django. Alamat IP untuk setiap CHR diatur pada interface net0 (ether1) sesuai dengan skema berikut. Tabel 2.1 berikut menampilkan pembagian alamat IP untuk masing-masing virtual CHR (vCHR) yang berada dalam satu subnet yang sama. Semua perangkat ini menggunakan subnet mask /25.

Tabel 3.1 Alokasi IP address vCHR

No.	Nama Perangkat	IP Address

1	vCHR-C1	10.161.20.180/25
2	vCHR-C2	10.161.20.181/25
3	vCHR-C3	10.161.20.182/25
4	vCHR-C4	10.161.20.183/25
5	vCHR-C5	10.161.20.184/25
6	vCHR-C6	10.161.20.185/25
7	vCHR-C7	10.161.20.186/25
8	vCHR-C8	10.161.20.187/25
9	vCHR-C9	10.161.20.188/25
10	vCHR-C10	10.161.20.189/25
11	vCHR-C11	10.161.20.190/25
12	vCHR-C12	10.161.20.191/25
13	vCHR-C13	10.161.20.192/25
14	vCHR-C14	10.161.20.193/25
15	vCHR-C15	10.161.20.194/25

Default Gateway seluruh CHR adalah 10.161.20.129 (Router MikroTik CCR-1036) untuk akses ke subnet manajemen 10.151.20.128/25. Konfigurasi yang dikirim ke setiap CHR terdiri dari 150 baris command yang dibagi menjadi tiga kategori firewall rules:

1) 50 baris IP *Firewall Mangle* untuk *packet marking*, *routing policy*, dan *traffic classification*. Contoh command:

```
/ip firewall mangle
add action=mark-packet chain=prerouting dst-port=80 new-packet-
mark=traffic_high_priority passthrough=yes protocol=tcp src-
address=192.169.1.0/24
add action=mark-packet chain=prerouting dst-port=8080 new-packet-
mark=traffic_high_priority passthrough=yes protocol=tcp src-
address=192.169.2.0/24
add action=mark-packet chain=prerouting dst-port=80 new-packet-
mark=traffic_high_priority passthrough=yes protocol=tcp src-
address=192.169.3.0/24
add action=mark-packet chain=prerouting dst-port=8080 new-packet-
mark=traffic high priority passthrough=yes protocol=tcp src-
```

2) 50 baris IP Firewall NAT untuk *Network Address Translation*, termasuk *source* NAT (*masquerade*) dan *destination* NAT (*port forwarding*). Contoh command:

```
/ip firewall nat
add action=masquerade chain=srcnat src-address=192.169.1.0/24
add action=masquerade chain=srcnat src-address=192.169.2.0/24
add action=masquerade chain=srcnat src-address=192.169.3.0/24
add action=masquerade chain=srcnat src-address=192.169.4.0/24
add action=masquerade chain=srcnat src-address=192.169.5.0/24
```

3) 50 baris IP *Firewall Filter* untuk *access control*, *packet filtering*, dan *security policy*. Contoh command:

```
/ip firewall filter
add chain=forward action=accept protocol=tcp dst-port=80 src-
address=192.169.1.0/24
add chain=forward action=accept protocol=tcp dst-port=8080 src-
address=192.169.2.0/24
add chain=forward action=accept protocol=tcp dst-port=80 src-
address=192.169.3.0/24
```

Jumlah dan tingkat kerumitan perintah yang digunakan disesuaikan dengan kondisi konfigurasi router pada penggunaan nyata, seperti router edge atau gateway yang membutuhkan aturan firewall yang cukup lengkap. Sebanyak 150 perintah dipilih untuk menggambarkan skala menengah, tidak terlalu sederhana seperti 10–20 aturan, dan juga tidak terlalu rumit hingga ratusan aturan, sehingga dapat mewakili kebutuhan konfigurasi yang umum digunakan.

3.1.5. Infrastruktur Server dan Database

CNCSA dan Standard Netmiko dijalankan pada dua mesin virtual yang berbeda, masing-masing menggunakan Ubuntu Server 22.04 LTS (64-bit). Versi ini dipilih karena termasuk versi jangka panjang (LTS) yang masih mendapatkan pembaruan keamanan dan perbaikan bug hingga tahun 2027, serta kompatibel dengan Python dan framework Django. Berdasarkan konfigurasi Proxmox (`qm config 126`), server CNCSA (VM ID 126, hostname: NA-Netmiko-MultiConfig) memiliki spesifikasi berikut:

a) CPU: 4 core (1 socket / 4 vCPU)
b) Memory: 4096 MB (4 GB RAM)
c) Disk: 60 GB (SSD, storage local-lvm)
d) Network: 1 interface (net0) menggunakan driver virtio, terhubung ke vmbr0
e) IP Address: 10.151.20.144/25

Empat vCPU diberikan agar proses aplikasi Django, pengolahan database, dan koneksi SSH dapat berjalan dengan lancar tanpa hambatan dari sisi CPU. RAM sebesar 4 GB cukup untuk menjalankan sistem operasi, Django, Python virtual environment, serta database SQLite. Kapasitas penyimpanan 60 GB dinilai cukup untuk kebutuhan sistem, aplikasi, data, dan log. Server standard netmiko menggunakan spesifikasi yang sama persis untuk mengukur dalam perbandingan. Perbedaannya hanya pada IP address, yaitu 10.151.20.147/25, serta penggunaan kode Netmiko standar tanpa optimasi seperti pada CNCSA. Lingkungan Python pada kedua server menggunakan Python versi 3.12.4 yang dipasang melalui perintah `apt install python3 python3-pip python3-venv`. Aplikasi dijalankan di dalam Python *virtual environment* agar lebih terisolasi dan tidak mengganggu sistem utama.

Tabel 3.2 merangkum seluruh spesifikasi perangkat keras dan perangkat lunak yang digunakan.

Komponen	Spesifikasi / Deskripsi

Infrastruktur Fisik	
Server Host	Intel Xeon Bronze 3104 @ 1.70GHz (6 cores)
Platform Virtualisasi	Proxmox Virtual Environment v6.3-2
Teknologi Virtualisasi	Intel VT-x (Hardware-assisted)
Server Aplikasi	
Sistem Operasi	Ubuntu Server 22.04 LTS (64-bit)
Alokasi CPU	4 vCPU (4 core, 1 socket)
Alokasi Memori	4 GB RAM
Alokasi Disk	60 GB SSD (local-lvm)
Interface Jaringan	1x virtio (vmbr0)
IP Server CNCSA	10.151.20.144/25
IP Server Standard	10.151.20.147/25
Software Stack	
Bahasa Pemrograman	Python 3.12.4
Framework Web	Django 5.2.3
Library Otomasi SSH	Netmiko 4.1+
Sistem Database	SQLite 3 (built-in)
Mode Deployment	Development Server (runserver)
Port Server	8000
Isolasi Lingkungan	Python Virtual Environment
Editor Pengembangan	Visual Studio Code
Infrastruktur Jaringan	
Network Management	10.151.20.128/25 (vmbr0)
Network CHR	10.161.20.128/25 (vmbr1)
Router Gateway	MikroTik RB951
Gateway Management	10.151.20.129/25
Gateway CHR	10.161.20.129/25
Latency Network (rata-rata)	0.5 – 2 ms RTT
Perangkat Virtual Router	
Jenis Perangkat	MikroTik Cloud Hosted Router (CHR)
Versi RouterOS	7.x
Lisensi	Free Tier (limit 1 Mbps)
Jumlah CHR	15 unit
CPU per CHR	1 vCPU
Memori per CHR	256 MB RAM
Disk per CHR	64 MB
Interface Jaringan	3x virtio (vmbr1)
Rentang IP CHR	10.161.20.180 – 10.161.20.194
Skenario Konfigurasi	
Jumlah Interface CHR	3 (net0, net1, net2)

Interface Manajemen	net0 (ether1)
Port SSH	22 (default)
Firewall Mangle	50 baris perintah
Firewall NAT	50 baris perintah
Firewall Filter	50 baris perintah

Tabel 3.3 berikut menunjukkan daftar virtual machine (VM) yang aktif pada Proxmox beserta penggunaan memori, CPU, dan waktu aktifnya. Data ini digunakan untuk memastikan seluruh server aplikasi dan perangkat CHR dalam kondisi berjalan normal saat proses pengujian dilakukan.

Tabel 3.3 Status Virtual Machine pada Proxmox

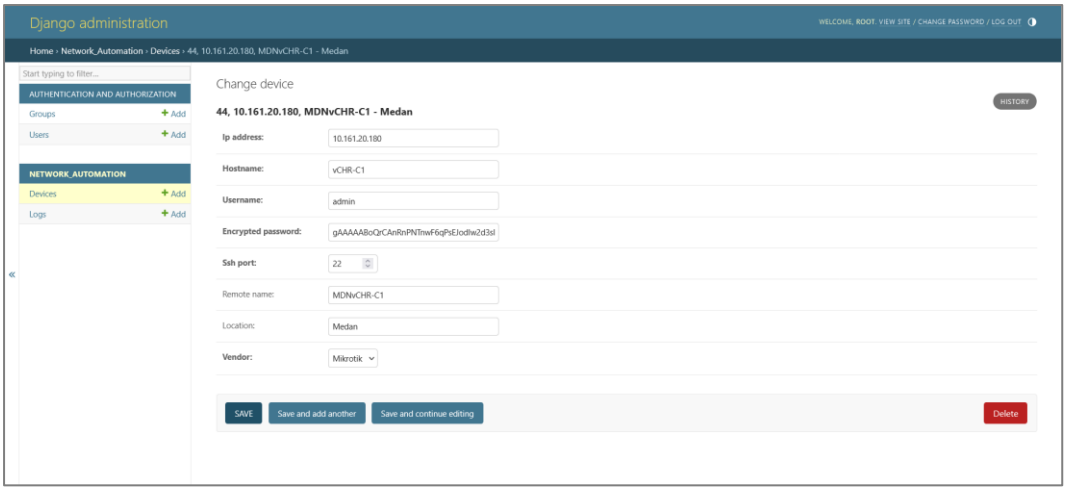
VMID	Nama VM	Status	Memori (MB)	CPU (%)	Uptime (detik)
126	NA-Netmiko-CNCSA	running	4096	2.50	15234
127	NA-Netmiko-Standard	running	4096	2.45	15230
129	vCHR-C1	running	256	0.06	5105
130	vCHR-C2	running	256	0.06	5231
131	vCHR-C3	running	256	0.06	5354
132	vCHR-C4	running	256	0.06	5481
133	vCHR-C5	running	256	0.06	5589
134	vCHR-C6	running	256	0.06	5715
135	vCHR-C7	running	256	0.06	5822
136	vCHR-C8	running	256	0.06	5912
137	vCHR-C9	running	256	0.06	6026
138	vCHR-C10	running	256	0.06	6106
139	vCHR-C11	running	256	0.06	6214
140	vCHR-C12	running	256	0.06	6312
141	vCHR-C13	running	256	0.06	6394
142	vCHR-C14	running	256	0.06	6503
143	vCHR-C15	running	256	0.06	6584

Proses Eksekusi Konfigurasi dan Algoritma CNCSA

Bagian ini menjelaskan alur kerja algoritma CNCSA, mulai dari pengelolaan data perangkat, proses pengiriman konfigurasi, hingga cara pencatatan waktu pada setiap tahap. Penjelasan dibuat bertahap agar mudah dipahami.

3.1.6. Pengelolaan Perangkat dengan Django ORM

Django ORM (*Object Relational Mapping*) adalah fitur pada django untuk mengelola database menggunakan kode python tanpa perlu menulis perintah SQL secara langsung. Dalam penelitian ini, seluruh data CHR disimpan di database melalui model Device yang telah dibahas pada subbab 3.2.1. Cara menyimpan data perangkat dengan cara menginput data perangkat melalui halaman admin http://10.151.20.144:8000/admin/. Setiap kali perangkat baru ditambahkan, Django akan membuat object baru dari model Device dan menyimpannya ke database. Contoh data yang disimpan divisualisasikan seperti gambar 3.5.



Gambar 3.5. Database Device Router CHR

Kemudian untuk mengambali data perangkat, saat penulis membuka halaman konfigurasi `/configure`, Django mengambil semua data perangkat dari database dengan query sederhana:

```
devices = device.objects.all()
```

Query ini akan mengembalikan list berisi 15 object Device (semua router CHR). Data ini kemudian dikirim ke template HTML untuk ditampilkan dalam bentuk checkbox, sehingga penulis bisa pilih perangkat mana yang mau dikonfigurasi.

3.1.7. Proses Eksekusi Konfigurasi

Proses konfigurasi dimulai ketika penulis submit form di halaman `/configure`. Berikut adalah alur lengkap proses yang terjadi:

1) User (Penulis) Memilih Perangkat dan Input Command
   Di halaman web, penulis melihat daftar 15 CHR dalam bentuk checkbox. penulis centang perangkat yang ingin dikonfigurasi (bisa semua 15, atau sebagian saja). Untuk setiap perangkat yang dipilih, penulis bisa input command yang akan dikirim. Dalam penelitian ini, semua perangkat mendapat command yang sama: 150 baris yang terdiri dari 50 Mangle, 50 NAT, dan 50 Filter rules.

2) Browser Mengirim Request ke Server
   Saat penulis klik tombol *Execute Configuration*, browser mengirim HTTP POST request ke URL `/configure`. Request ini membawa data: -ID perangkat yang dipilih dan - Command yang akan dikirim (150 baris teks)
3) Django Menerima dan Memvalidasi Request
   Fungsi `configure ()` di file `views.py` menerima request ini. Pertama-tama, fungsi akan validasi:
   - Apakah ada perangkat yang dipilih? (kalau tidak ada, tampilkan error)
   - Apakah command yang diinput tidak kosong?
   - Apakah router CHR yang dipilih benar-benar ada di database?

Jika validasi berhasil, proses dilanjutkan ke tahap eksekusi.

4) Memulai Timer Global
   Sebelum mulai koneksi ke perangkat, timer global dinyalakan dengan:
   `start_time = time.time()`
   Fungsi `time.time()` mengembalikan waktu sekarang dalam format Unix *timestamp*. Timer ini akan digunakan untuk menghitung total *elapsed time* dari seluruh proses.
5) Loop *Sequential* untuk Setiap Perangkat
   Algoritma CNCSA menggunakan model *sequential execution*, artinya perangkat dikonfigurasi satu per satu secara berurutan. Tidak ada paralel processing, dan loop dilakukan dengan proses:
   `for device in selected_devices:`

   Untuk setiap device, ada 3 fase yang dijalankan:
   Fase 1: Koneksi SSH (diukur sebagai Tc_i)
   Timer untuk koneksi dimulai:
   `conn_start = time.time()`
   Django memanggil library Netmiko untuk membuka koneksi SSH ke perangkat. Parameter koneksi yang dikirim:
   - Host: IP address perangkat (misal 10.161.20.180)
   - Username: admin
   - Password: hasil dekripsi dari encrypted_password
   - Device type: mikrotik_routeros
   - Port: 22
   - Timeout: 8 detik (parameter optimasi)
   - Session timeout: 45 detik (parameter optimasi)
   - Keepalive: 30 detik (parameter optimasi)

   Jika koneksi berhasil, Netmiko akan *return object connection* yang bisa digunakan untuk kirim command. Jika gagal (misal perangkat mati atau

password salah), akan muncul exception yang di-catch dan dicatat sebagai `error. timer` koneksi dihentikan:

```
tc_i = time.time() - conn_start
```

Nilai tc_i adalah waktu yang dibutuhkan untuk koneksi ke device ini (dalam detik). Nilai ini ditambahkan ke total:

```
total_tc += tc_i
```

Fase 2: Kirim dan Eksekusi Command (diukur sebagai Ts_i)
Timer untuk send command dimulai:

```
config_start = time.time()
```

Karena perangkat yang digunakan adalah MikroTik, command dikirim satu per satu menggunakan method `send_command()`. Parameter yang digunakan:

- Command: teks command (misal `/ip firewall mangle add chain=prerouting`)
- Delay factor: 0.5 (parameter optimasi, artinya jeda antar command dikurangi 50%)
- Max loops: 300 (parameter optimasi untuk deteksi timeout lebih cepat)
- Strip prompt: True (otomatis buang prompt router dari output)
- Strip command: True (otomatis buang echo command dari output)

Setiap command yang dikirim akan menghasilkan output (*response* dari router). Output ini dikumpulkan dalam variabel output. Jika ada error saat eksekusi command (misal syntax salah), error message akan ditangkap dan disimpan. Timer send command dihentikan:

```
ts_i = time.time() - config_start
```

Nilai ts_i adalah waktu yang dibutuhkan untuk kirim dan eksekusi semua command di device ini. Nilai ini ditambahkan ke total:

```
total_ts += ts_i
```

Setelah selesai kirim command, koneksi SSH ditutup dengan:

```
connection.disconnect()
```

Fase 3: Simpan Log ke Database (diukur sebagai Tl_i)
Timer untuk logging dimulai:

```
log_start = time.time()
```

Hasil konfigurasi disimpan ke database menggunakan Django ORM:

```
Log.objects.create(
    target=device,
    action="Configure",
    status="Success",
    time=timezone.now(),
    message=output[:5000],
    duration=tc_i + ts_i
```

)

Field message dibatasi maksimal 5000 karakter untuk menghemat space database. Teknik ini merupakan salah satu optimasi logging. Field duration menyimpan total waktu untuk device ini (connection + send).Timer logging dihentikan:

```
tl_i = time.time() - log_start
```

Nilai tl_i adalah waktu yang dibutuhkan untuk simpan log. Nilai ini ditambahkan ke total:

```
total_tl += tl_i
```

6) Loop Selesai - Hitung Metrics

Setelah semua device selesai diproses, timer global dihentikan:

```
end_time = time.time()
elapsed_time = end_time - start_time
```

Kemudian menghitung semua metrics:

- Total Execution Time: `te_cncsa = total_tc + total_ts + total_tl`
- System Overhead: `overhead = elapsed_time - te_cncsa`
- Average time per device: `t_avg = te_cncsa / jumlah_device`
- Efficiency ratios

7) Kirim Hasil ke Template

Semua metrik dan hasil detail per perangkat disimpan dan dikirim ke halaman HTML untuk ditampilkan dan dapat melihat ringkasan waktu (Tc, Ts, Tl, Te, overhead) dan efisiensi, serta tabel rincian performa untuk setiap CHR. Semua nilai metrik disimpan terlebih dahulu ke dalam session Django, kemudian dikirimkan ke file HTML untuk ditampilkan kepada pengguna. Melalui tampilan tersebut, penulis dapat melihat durasi total koneksi (Tc), waktu yang digunakan untuk pengiriman perintah konfigurasi (Ts), waktu yang diperlukan untuk proses logging (Tl), total execution time (Te), nilai system *overhead* (ε), serta persentase efisiensi dari setiap komponen proses. Dengan mekanisme ini, seluruh hasil analisis waktu dapat dipantau secara langsung melalui browser.

3.1.8. Model Eksekusi Sequential

Pada CNCSA digunakan pola eksekusi sequential, yaitu setiap perangkat diproses secara berurutan satu demi satu, bukan secara paralel. Pemilihan model ini didasarkan pada beberapa pertimbangan teknis.

1) Presisi Pengukuran Waktu

Eksekusi yang berjalan secara berurutan memungkinkan setiap komponen waktu diukur dengan akurat karena tidak ada gangguan dari proses lain. Jika proses dibuat paralel menggunakan multithreading, pengukuran waktu bisa terdistorsi akibat context switching, sinkronisasi thread, atau potensi race condition yang dapat memunculkan noise terhadap nilai waktu aktual.

2) Menghindari Tumpang Tindih Penggunaan Sumber Daya
   Membuka banyak koneksi SSH secara bersamaan dapat menimbulkan tumpang tindih sumber daya di sisi server, baik CPU, memori, maupun bandwidth jaringan. Dengan menjalankan koneksi satu per satu, penggunaan sumber daya lebih stabil sehingga performa tiap koneksi tetap konsisten.
3) Menyesuaikan Batasan SSH pada Router CHR
   Perangkat MikroTik CHR memiliki batas jumlah sesi SSH yang dapat aktif secara bersamaan. Jika terlalu banyak koneksi dibuka pada saat yang sama, sebagian sesi dapat gagal dibentuk. Model sequential menghindari risiko ini sepenuhnya karena hanya satu koneksi yang aktif pada satu waktu.
4) Menjaga Kesesuaian dengan Baseline
   Metode baseline (Standard Netmiko) juga menggunakan eksekusi sequential. Agar perbandingan performa tetap objektif, CNCSA mengikuti model yang sama. Dengan demikian, selisih performa yang muncul benar-benar mencerminkan perbedaan strategi optimasi, bukan akibat perbedaan pola eksekusi.

3.1.9. Pseudocode Algoritma CNCSA

Bagian ini menyajikan pseudocode dari algoritma CNCSA untuk memberikan gambaran yang jelas mengenai urutan proses yang dilakukan sistem saat melakukan konfigurasi router CHR secara terpusat dan berurutan. Pseudocode digunakan untuk memvisualisasikan logika algoritma, sehingga memudahkan pembacaan, dokumentasi, dan proses analisis terhadap setiap tahap eksekusi yang diukur dalam penelitian ini.

Algoritma: Centralized Network Configuration Sequential Algorithm (CNCSA)

```
Input:
    Devices[1..n]       : Array of Device objects yang akan
dikonfigurasi
    Commands[1..n]    : Array of command strings untuk setiap
device

Output:
    T_c               : Total Connection Time (detik)
    T_s               : Total Send Time (detik)
    T_l               : Total Logging Time (detik)
```

```
    T_e               : Total Execution Time (detik)
    ε                 : System Overhead (detik)
    Results[1..n]     : Array of execution results per device

BEGIN
    start_time ← current_time()
    T_c ← 0
    T_s ← 0
    T_l ← 0
    Results ← empty_array()


    FOR i ← 1 TO n DO
        device ← Devices[i]
        commands ← Commands[i]
        connection ← NULL
        status ← "Success"
        output ← ""

        conn_start ← current_time()
        TRY

            connection ← establish_ssh_connection(
                host: device.ip_address,
                username: device.username,
                password: decrypt(device.password),
                timeout: 8,
                session_timeout: 45,
                keepalive: 30
            )
        CATCH ConnectionError AS e
            T_c_i ← current_time() - conn_start
            T_c ← T_c + T_c_i

            status ← "Connection Failed"
            output ← e.message
            T_s_i ← 0
            T_l_i ← 0

            log_start ← current_time()
            save_log_optimized(device, status, output,
T_c_i)
            T_l_i ← current_time() - log_start
            T_l ← T_l + T_l_i

Results[i] ← create_result(device, status, T_c_i, 0, T_l_i)
      CONTINUE to next device
  END TRY

  T_c_i ← current_time() - conn_start
  T_c ← T_c + T_c_i

  config_start ← current_time()
  TRY

      FOR EACH cmd IN commands DO
```

```
            cmd_output ← send_command_optimized(
                connection,
                command: cmd,
                delay_factor: 0.5,
                max_loops: 300,
                strip_prompt: TRUE,
                strip_command: TRUE
            )
            output ← output + cmd_output
        END FOR

        T_s_i ← current_time() - config_start
        T_s ← T_s + T_s_i

        disconnect_gracefully(connection)
        status ← "Success"

    CATCH ConfigurationError AS e
        T_s_i ← current_time() - config_start
        T_s ← T_s + T_s_i

        disconnect_gracefully(connection)

        output ← e.message
        status ← "Configuration Error"
    END TRY

    log_start ← current_time()

    save_log_optimized(
        target: device,
        action: "Configure",
        status: status,
        timestamp: current_time(),
        message: truncate(output, 5000),
        duration: T_c_i + T_s_i
    )

    T_l_i ← current_time() - log_start
    T_l ← T_l + T_l_i

total_device_time ← T_c_i + T_s_i + T_l_i
          Results[i] ← create_result(
              device: device,
              status: status,
              T_c_i: T_c_i,
              T_s_i: T_s_i,
              T_l_i: T_l_i,
              total: total_device_time
          )
      END FOR

      end_time ← current_time()
      elapsed_time ← end_time - start_time

      T_e ← T_c + T_s + T_l
```

```
    ε ← elapsed_time - T_e

    IF T_e > 0 THEN
        E_conn ← (T_c / T_e) × 100
        E_config ← (T_s / T_e) × 100
        E_log ← (T_l / T_e) × 100
        E_CNCSA ← ((T_s + T_l) / T_e) × 100
        SOR ← (ε / T_e) × 100
    ELSE
        E_conn ← 0
        E_config ← 0
        E_log ← 0
        E_CNCSA ← 0
        SOR ← 0
    END IF


    T_avg ← T_e / n

    success_count ← count_success(Results)
    success_rate ← (success_count / n) × 100

    RETURN {
        T_c: T_c,
        T_s: T_s,
        T_l: T_l,
        T_e: T_e,
        ε: ε,
        elapsed_time: elapsed_time,

        E_conn: E_conn,
        E_config: E_config,
        E_log: E_log,
        E_CNCSA: E_CNCSA,
        SOR: SOR,
n: n,
        T_avg: T_avg,
        success_rate: success_rate,

        Results: Results
    }
END
```

Algoritma CNCSA dimulai dengan menerima input berupa array Devices yang berisi daftar perangkat jaringan dan array Commands yang berisi perintah konfigurasi, kemudian menginisialisasi semua accumulator waktu (T_c, T_s, T_l) dengan nilai nol serta mencatat waktu mulai eksekusi. Algoritma menggunakan *sequential execution* model dimana setiap perangkat diproses satu per satu secara berurutan dalam sebuah loop dari indeks 1 hingga n. Untuk setiap perangkat, algoritma melakukan tiga langkah utama yang masing-masing diukur waktunya secara terpisah: pertama adalah koneksi SSH ke perangkat dengan parameter *optimized* (*timeout* 8 detik, session_timeout 45 detik, keepalive 30 detik) yang jika gagal akan langsung mencatat error dan melanjutkan ke perangkat berikutnya;

kedua adalah pengiriman dan eksekusi command dengan parameter optimized (`delay_factor 0.5, max_loops 300, strip_prompt TRUE, strip_command TRUE`) dimana setiap command dikirim satu per satu dan output dikumpulkan; ketiga adalah logging hasil eksekusi ke database dengan optimasi message size limit 5000 karakter untuk mempercepat insert operation.

Setiap tahap dalam *loop* per-device dibungkus dengan *try-catch block* untuk error *handling* yang *robust*, dimana waktu tetap diukur bahkan ketika terjadi error untuk memastikan akurasi metrics. Waktu koneksi (T_c_i) diukur dari mulai *establish connection* hingga connection berhasil atau error, waktu send (T_s_i) diukur dari mulai pengiriman command pertama hingga semua command selesai dikirim, dan waktu logging (T_l_i) diukur dari mulai database insert hingga selesai. Setiap komponen waktu per-device (T_c_i, T_s_i, T_l_i) dijumlahkan ke accumulator global (T_c, T_s, T_l) sehingga setelah *loop* selesai akan didapatkan total waktu untuk semua perangkat. Hasil eksekusi setiap device disimpan dalam array Results yang mencatat *device object*, status eksekusi, *breakdown timing* per komponen, dan total waktu untuk device tersebut, memberikan visibility detail pada level individual device yang berguna untuk *troubleshooting* dan analisis granular.

Setelah semua perangkat selesai diproses, algoritma menghitung berbagai metrics untuk evaluasi performa. *Total Execution Time* (T_e) dihitung dengan formula T_c + T_s + T_l yang merepresentasikan net processing time dari ketiga komponen utama, sedangkan System *Overhead* (ε) dihitung dengan formula *Elapsed_Time* - T_e yang merepresentasikan waktu yang hilang di luar ketiga komponen tersebut karena *framework overhead* atau OS *scheduling*. Algoritma kemudian menghitung *efficiency ratios* untuk menganalisis proporsi waktu setiap komponen: *Connection Efficiency* (E_conn) mengukur persentase waktu untuk koneksi, *Configuration Efficiency* (E_config) mengukur persentase waktu untuk pengiriman command yang merupakan productive work utama, *Logging Efficiency* (E_log) mengukur persentase waktu untuk database *logging*, CNCSA Execution Efficiency (E_CNCSA) mengukur persentase waktu untuk productive work total (konfigurasi + logging), dan System *Overhead Ratio* (SOR) mengukur persentase *overhead* sistem. Semua efficiency ratios dihitung dengan membagi komponen waktu dengan T_e dikali 100 persen, dan penjumlahan `E_conn, E_config`, dan `E_log` harus sama dengan 100 persen untuk memverifikasi konsistensi perhitungan.

Algoritma menghasilkan output komprehensif berupa dictionary yang berisi time components (T_c, T_s, T_l, T_e, ε, *elapsed_time*), *efficiency metrics* (`E_conn, E_config, E_log, E_CNCSA, SOR), statistics (n, T_avg, success_rate`), dan detailed results per-device dalam *array results*. Parameter optimized yang digunakan memberikan *improvement* signifikan dibandingkan parameter default dimana delay_factor 0.5 menjadi kontributor terbesar dengan mengurangi jeda antar command menjadi setengahnya, max_loops 300 mempercepat deteksi *response* atau *timeout* sebesar 40 persen, dan *strip_prompt* serta *strip_command* mengurangi ukuran data transfer dan mempercepat *parsing output*.

3.1.10. Representasi Matematis CNCSA

3.1.10.1. Formula Komponen Waktu

Algoritma CNCSA mengukur tiga komponen waktu utama yang masing-masing dihitung sebagai akumulasi dari waktu individual setiap router CHR yang diproses secara sequential. Formula untuk setiap komponen didefinisikan sebagai berikut:

Total Connection Time (Tc):

$$T_C = \sum_{i=1}^{n} T_{c_i} \qquad (1)$$

Dimana:

- T_c = Total waktu koneksi untuk semua perangkat (detik)
- T_{c_i}= Waktu koneksi untuk perangkat ke-i (detik)
- n= Jumlah total perangkat yang dikonfigurasi

Total Send Time (Ts):

$$T_s = \sum_{i=1}^{n} T_{s_i} \qquad (2)$$

Dimana:

- T_s = Total waktu pengiriman dan eksekusi command untuk semua perangkat (detik)
- T_{s_i}= Waktu pengiriman dan eksekusi command untuk perangkat ke-i (detik)
- n= Jumlah total perangkat yang dikonfigurasi

Total Logging Time (Tl):

$$T_l = \sum_{i=1}^{n} T_{l_i} \qquad (3)$$

Dimana:

- T_l = Total waktu logging ke database untuk semua router CHR (detik)
- T_{l_i}= Waktu logging untuk perangkat ke-i (detik)
- n= Jumlah total perangkat yang dikonfigurasi

Ketiga formula di atas menggambarkan prinsip akumulasi linear, di mana total waktu untuk setiap komponen pada perangkat router CHR diperoleh dengan menjumlahkan waktu masing-masing perangkat router CHR. Dalam model eksekusi sequential, setiap perangkat diproses satu per satu secara berurutan, sehingga total waktu untuk n perangkat adalah penjumlahan langsung dari waktu setiap perangkat tanpa ada tumpang tindih atau eksekusi paralel. Formula ini berlaku untuk semua nilai n mulai dari 1 hingga tak terbatas, dengan asumsi bahwa

setiap perangkat memiliki karakteristik yang relatif seragam sehingga waktu per perangkat berada dalam rentang yang konsisten. Operator sigma (∑) menunjukkan operasi penjumlahan berulang dari indeks i = 1 hingga i = n, yang dalam implementasi nyata dilakukan melalui loop sequential, misalnya dengan statement seperti total_tc = total_tc + tc_i pada setiap iterasi.

3.1.10.2. Formula Total Execution Time

Total Execution Time merepresentasikan durasi efektif yang digunakan untuk ketiga komponen proses utama, serta kaitannya dengan elapsed time yang diukur secara real-time atau aktual selama eksekusi.

Total Execution Time (Te):

$$T_e = T_c + T_s + T_l \quad (4)$$

Formula Expanded:

$$T_e = \sum_{i=1}^{n} T_{c_i} + \sum_{i=1}^{n} T_{s_i} + \sum_{i=1}^{n} T_{l_i} \quad (5)$$

Dimana:

- T_e = Total Execution Time, yaitu waktu bersih untuk semua proses (detik)
- T_c = Total Connection Time (detik)
- T_s = Total Send Time (detik)
- T_l = Total Logging Time (detik)

System Overhead (ε):

$$\varepsilon = T_{elapsed} - T_e \quad (6)$$

Dimana:

- ε = System Overhead, yaitu waktu yang tidak tercatat dalam ketiga komponen utama (detik)
- $T_{elapsed}$= Elapsed Time, yaitu waktu actual dari jam dinding mulai eksekusi hingga selesai (detik)
- T_e= Total Execution Time (detik)

Verifikasi Konsistensi:

$$T_{elapsed} = T_c + T_s + T_l + \varepsilon \quad (7)$$

Berdasarkan hasil pengujian CNCSA terhadap 15 perangkat MikroTik CHR dengan 150 command per perangkat, perhitungan menunjukkan waktu Total Connection Time (Tc), Total Send Time (Ts), dan Total Logging Time (Tl) yang

kemudian dijumlahkan untuk memperoleh Total Execution Time (Te), serta dibandingkan dengan Elapsed Time untuk menghitung System Overhead (ε). Contoh perhitungan dengan data penelitian berdasarkan hasil pengujian CNCSA terhadap 15 perangkat MikroTik CHR dengan 150 command per perangkat:

```
Tc = 4.4709 detik
Ts = 297.1861 detik
Tl = 0.8335 detik

Te = 302.4904 detik
Elapsed_Time = 302.5242 detik
ε = 302.5242 - 302.4904 = 0.0338 detik
Verifikasi: Te + ε = 302.4904 + 0.0338 = 302.5242 detik
```

Formula T_e sama dengan T_c ditambah T_s ditambah T_l adalah persamaan dasar yang menjelaskan bahwa total waktu eksekusi diperoleh dari penjumlahan tiga komponen waktu utama yang diukur secara langsung oleh algoritma. Bentuk diperluas (*expanded*) menunjukkan bahwa T_e bisa dihitung baik dengan menjumlahkan ketiga total komponen maupun dengan menjumlahkan waktu tiap perangkat secara bertahap, dimana kedua cara ini akan menghasilkan nilai yang sama karena sifat asosiatif dan komutatif dari operasi penjumlahan.

System overhead ε merepresentasikan waktu yang tidak tercatat dalam ketiga komponen utama, yang bisa berasal dari overhead framework Django seperti routing request dan rendering template, *overhead interpreter* Python seperti penugasan variabel dan pemanggilan fungsi, overhead sistem operasi seperti *context switching* dan alokasi memori, atau overhead dari pengukuran waktu itu sendiri seperti pemanggilan fungsi `time.time` berulang kali. Konsistensi diverifikasi dengan memastikan bahwa penjumlahan Te dan ε sama dengan elapsed time jika ada perbedaan, berarti terjadi kesalahan pada pengukuran atau perhitungan. Contoh data menunjukkan ε sangat kecil, hanya 0,033 detik atau 0,01 persen dari Te, yang menandakan algoritma sangat efisien dengan overhead minimal, dan verifikasi menghasilkan kesamaan tepat tanpa error, membuktikan akurasi pengukuran dan validitas formula.

3.1.10.3. Formula Efficiency Ratios

Efficiency ratios mengukur persentase waktu yang digunakan oleh setiap komponen router CHR dibandingkan dengan total *execution time*, sehingga memberikan gambaran tentang bagaimana waktu diefisienkan dan didistribusikan dalam algoritma.

Connection Efficiency (E_conn):

$$E_{conn} = \frac{T_c}{T_e} x100\% \qquad (8)$$

Dimana:

- $E_{conn} =$ Persentase waktu yang digunakan untuk koneksi SSH (%)

- Target ideal: < 5% (semakin kecil semakin baik karena koneksi adalah overhead)

Configuration Efficiency (E_config):

$$E_{config} = \frac{T_s}{T_e}\ x\ 100\% \qquad (9)$$

Dimana:

- E_{config} = Persentase waktu yang digunakan untuk pengiriman dan eksekusi command (%)
- Target ideal: > 90% (semakin besar semakin baik karena ini adalah akttivitas inti)

Logging Efficiency (E_log):

$$E_{log} = \frac{T_l}{T_e} x 100\% \qquad (10)$$

Dimana:

- E_{log} = Persentase waktu yang digunakan untuk logging ke database (%)
- Target ideal: < 1%

CNCSA *Execution Efficiency* (E_CNCSA):

$$E_{CNCSA} = \frac{T_s + T_l}{T_e}\ x\ 100\% \qquad (11)$$

Dimana:

- E_{CNCSA} = Persentase waktu yang digunakan untuk (konfigurasi dan logging) (%)
- Target ideal: > 95%

System Overhead Ratio (SOR):

$$SOR = \frac{\varepsilon}{T_e} x\ 100\% \qquad (12)$$

Dimana:

- SOR= Persentase system overhead terhadap total execution time (%)
- Target ideal: < 1%

Verifikasi Konsistensi Total:

$$E_{conn} + E_{config} + E_{log} = 100\% \qquad (13)$$

Hasil perhitungan *efficiency ratios* menunjukkan bagaimana waktu eksekusi dalam algoritma CNCSA terbagi ke dalam tiga komponen utama, yaitu waktu koneksi, waktu konfigurasi, dan waktu *logging*. Dari data pengujian, proporsi waktu yang digunakan oleh masing-masing komponen dapat diverifikasi melalui dua cara, yaitu dengan menjumlahkan `E_conn + E_config + E_log atau E_conn + E_CNCSA`. Kedua metode menghasilkan total 100%, yang menandakan bahwa distribusi waktu telah konsisten dan tidak ada bagian yang hilang atau terduplikasi. Perhitungan aktual menunjukkan bahwa sebagian besar waktu eksekusi berada pada proses konfigurasi, sedangkan waktu koneksi dan logging hanya mengambil porsi yang sangat kecil. Nilai *System Overhead Ratio* (SOR) yang mendekati nol juga mengonfirmasi bahwa tidak terdapat beban tambahan signifikan di luar proses inti algoritma. Contoh Perhitungan dengan Data Penelitian yang dihasilkan:

```
E_conn      = (4.4709 / 302.4904) × 100% = 1.48%
E_config    = (297.1861 / 302.4904) × 100% = 98.25%
E_log       = (0.8335 / 302.4904) × 100% = 0.28%
E_CNCSA     = (297.1861 + 0.8335) / 302.4904 × 100% =
98.52%
            = 98.25% + 0.28% = 98.53%
SOR         = (0.0338 / 302.4904) × 100% = 0.0112%

Verifikasi: 1.48% + 98.25% + 0.28% = 100.01%
            1.48% + 98.52% = 100.00%
```

Hasil tersebut juga memberikan gambaran bahwa CNCSA memiliki distribusi waktu yang sangat efisien, dengan 98.25 persen digunakan untuk configuration, 1.48 persen untuk connection, dan 0.28 persen untuk logging, sehingga total pekerjaan bernilai mencapai 98.52 persen dan melebihi target ideal sebesar 95 persen.

Formula *efficiency ratios* bekerja dengan prinsip proporsi, yaitu setiap komponen router CHR waktu dibagi dengan total *execution time* kemudian dikalikan 100% untuk memperoleh persentasenya. *Connection efficiency* menunjukkan seberapa besar bagian waktu yang dipakai untuk membangun koneksi SSH ke perangkat, dan nilai yang rendah menandakan bahwa pengaturan koneksi sudah optimal sehingga tidak menjadi *bottleneck*. *Configuration efficiency* merupakan metrik yang paling penting karena menggambarkan bagian waktu yang digunakan untuk pekerjaan utama, yaitu mengirim dan mengeksekusi command pada perangkat nilai yang tinggi mendekati 100 % berarti hampir seluruh waktu berjalan untuk proses inti dan bukan untuk *overhead*. *Logging efficiency* menunjukkan bagian waktu yang diperlukan untuk proses insert ke database, dan nilai yang sangat kecil kurang dari 1 persen menunjukkan bahwa optimasi seperti pembatasan ukuran pesan dan metode penyimpanan langsung sudah efektif sehingga proses logging tidak menambah beban yang signifikan terhadap total *execution time*.

CNCSA *execution efficiency* menggambarkan total pekerjaan utama sistem dengan menjumlahkan *configuration efficiency* dan *logging efficiency*, karena kedua komponen ini dianggap sebagai aktivitas yang bernilai—configuration merupakan tujuan inti algoritma, sedangkan logging diperlukan untuk pencatatan dan proses audit. Nilai `E_CNCSA` dapat dihitung dengan dua cara, yaitu dengan membagi penjumlahan Ts dan Tl terhadap Te, atau dengan menjumlahkan `E_config` dan `E_log`, di mana kedua metode harus menghasilkan nilai yang sama sebagai bukti konsistensi perhitungan. Sementara itu, system *overhead* ratio menunjukkan persentase waktu yang tidak tercakup oleh ketiga komponen utama, dan nilai yang sangat kecil seperti 0.01 persen menandakan bahwa algoritma berjalan sangat efisien tanpa beban tambahan yang tidak diperlukan dari framework maupun sistem.

3.1.10.4. Formula Average Time per Device

Formula rata-rata waktu per router CHR digunakan untuk mengetahui estimasi waktu konfigurasi pada setiap device dan membantu dalam perencanaan kapasitas. Nilai ini dihitung dengan membagi total execution time dengan jumlah router CHR yang diproses. Hasil perhitungan tersebut memberikan gambaran berapa lama waktu yang diperlukan untuk menangani satu router CHR dalam model eksekusi sequential, sehingga dapat digunakan untuk memprediksi kebutuhan waktu jika jumlah perangkat bertambah.

$$T_{avg} = \frac{T_n}{n} \qquad (14)$$

Dimana:

- $T_{avg} =$ Rata-rata waktu total per router CHR (detik)
- T_e= Total Execution Time (detik)
- $n =$ Jumlah router CHR yang dikonfigurasi

Formula Expanded per Komponen:

$$T_{avg}^{(c)} = \frac{T_c}{n} = \frac{1}{n}\sum_{i=1}^{n} T_{c_i} \qquad (15)$$

$$T_{avg}^{(s)} = \frac{T_s}{n} = \frac{1}{n}\sum_{i=1}^{n} T_{s_i} \qquad (16)$$

$$T_{avg}^{(l)} = \frac{T_l}{n} = \frac{1}{n}\sum_{i=1}^{n} T_{l_i} \qquad (17)$$

Dimana:

- $T_{avg}^{(c)} =$ Rata-rata connection time per router CHR (detik)

- $T_{avg}^{(s)} =$ Rata-rata *send time* per perangkat (detik)
- $T_{avg}^{(l)} =$ Rata-rata *logging time* per perangkat (detik)

Perhitungan dengan Data Penelitian sebanyak (15 Router CHR):

```
T_avg      = 340.1818/15 = 22.6788 detik per router CHR

T_avg(c)   = 4.4942/15 = 0.2996 detik per router CHR
T_avg(s)   = 334.5814/15 = 22.3054 detik per router CHR
T_avg(l)   = 1.1062/15 = 0.0737 detik per router CHR

Verifikasi: 0.2996 + 22.3054 + 0.0737 = 22.6787 ≈ 22.6788
detik
```

Formula *average time* per *device* digunakan untuk menghitung waktu rata-rata yang diperlukan untuk mengonfigurasi satu perangkat, yaitu dengan membagi total *execution time* dengan jumlah perangkat yang diproses. Berdasarkan pengujian pada 15 router MikroTik CHR yang masing-masing menjalankan 150 perintah, waktu rata-rata per perangkat adalah 20.17 detik. Rata-rata ini terdiri dari sekitar 0.30 detik untuk proses koneksi SSH, 19.81 detik untuk pengiriman dan eksekusi perintah, dan 0.06 detik untuk proses *logging* ke database. Dengan menghitung rata-rata tiap komponen secara terpisah, dapat terlihat bahwa bagian pengiriman perintah menggunakan porsi waktu terbesar, sementara koneksi dan *logging* hanya memakan sebagian kecil dari total waktu. Verifikasi dilakukan dengan menjumlahkan ketiga nilai rata-rata komponen tersebut, dan hasilnya 20.1661 detik, hampir sama dengan nilai perhitungan langsung 20.1660 detik. Selisih yang sangat kecil ini hanya berasal dari pembulatan angka desimal, yang menunjukkan bahwa perhitungan dan formulanya konsisten.

3.1.10.5. Formula Success Rate

Formula *success rate* digunakan untuk menilai seberapa banyak router CHR yang berhasil diproses tanpa *error* selama eksekusi konfigurasi. Metode ini memberikan ukuran dasar tentang keandalan proses, karena hanya perangkat yang benar-benar berhasil terkoneksi dan dieksekusi perintahnya yang dihitung sebagai keberhasilan. Sebaliknya, perangkat yang mengalami kegagalan, baik karena gagal terhubung maupun *error* saat menjalankan perintah, akan masuk dalam perhitungan error rate. Kedua metrik ini saling melengkapi, karena nilai *error rate* selalu menjadi selisih dari 100 persen dikurangi *success rate*, sehingga keduanya harus konsisten. Rumus ini membantu memastikan bahwa hasil eksekusi dapat dinilai secara objektif berdasarkan jumlah perangkat yang berhasil dan yang tidak.

Success Rate:

$$SR = \frac{n_{success}}{n} x100\% \qquad (18)$$

Dimana:

- $SR =$ *Success Rate*, yaitu persentase router CHR yang berhasil dikonfigurasi (%)
- $n_{success}$ = Jumlah router CHR yang berhasil dikonfigurasi tanpa *error*
- $n =$ Total jumlah router CHR yang diproses

Error rate:

$$ER = \frac{n_{error}}{n} x100\% = 100\% - SR \qquad (19)$$

Dimana:

- ER= Error Rate, yaitu persentase router CHR yang gagal dikonfigurasi (%)

Contoh Perhitungan dengan Data Penelitian:

```
Success Rate (SR):
SR = (15 / 15) × 100% = 100%

Error Rate (ER):
ER = 100% - 100% = 0%
```

Hasil pengujian memperlihatkan bahwa seluruh proses konfigurasi berjalan dengan sangat baik. Nilai *success rate* mencapai 100 persen, yang menunjukkan bahwa semua 15 router MikroTik CHR berhasil dikonfigurasi tanpa kendala. Selama proses berlangsung tidak ditemukan masalah koneksi maupun kesalahan dalam menjalankan perintah konfigurasi. Selain itu, nilai error rate tercatat 0 persen, yang berarti tidak ada perangkat yang mengalami kegagalan pada tahap konfigurasi. Kondisi ini menegaskan bahwa setiap router menerima dan mengeksekusi perintah sesuai dengan skenario pengujian yang telah ditentukan. Pencapaian tingkat keberhasilan penuh ini membuktikan bahwa optimasi parameter yang diterapkan pada algoritma CNCSA tidak berdampak negatif terhadap kestabilan sistem. Dengan kata lain, peningkatan kinerja yang diperoleh tetap disertai dengan keandalan yang terjaga. Hal ini menjadi aspek penting, terutama untuk penerapan pada lingkungan produksi, karena tingkat keandalan yang tinggi dapat mengurangi kebutuhan intervensi manual serta meminimalkan beban penanganan gangguan konfigurasi.

3.1.10.6. Ringkasan Formula CNCSA

Bagian ini menyajikan ringkasan dari seluruh formula yang digunakan dalam algoritma CNCSA. Semua formula ini dirangkum untuk mempermudah pemahaman mengenai cara perhitungan waktu, efisiensi, serta tingkat keberhasilan proses konfigurasi. Ringkasan ini membantu melihat kembali hubungan antar-metrik dan memastikan bahwa setiap komponen perhitungan dapat ditelusuri dengan jelas dan konsisten.

Komponen Waktu:

- $T_c = \sum_{i=1}^{n} T_{c_i}$
- $T_s = \sum_{i=1}^{n} T_{s_i}$
- $T_l = \sum_{i=1}^{n} T_{l_i}$

Total Eksekusi dan Overhead:

- $T_e = T_c + T_s + T_l$
- $\varepsilon = T_{elapsed} - T_e$
- $T_{elapsed} = T_e + \varepsilon$

Efficiency Ratios:

- $E_{conn} = \frac{T_c}{T_e}\ x\ 100\%$
- $E_{config} = \frac{T_s}{T_e}\ x\ 100\%$
- $E_{log} = \frac{T_l}{T_e}\ x\ 100\%$
- $E_{CNCSA} = \frac{T_c + T_l}{T_e}\ x\ 100\%$
- $SOR = \frac{\varepsilon}{T_e}\ x\ 100\%$

Verifikasi konsistensi:

- $E_{conn} + E_{config} + E_{log} = 100\%$

Average Time per Device:

- $T_{avg} = \frac{T_e}{n}$
- $T_{avg}^{(c)} = \frac{T_c}{n}$
- $T_{avg}^{(s)} = \frac{T_s}{n}$
- $T_{avg}^{(l)} = \frac{T_l}{n}$

Reliability:

- $SR = \frac{n_{success}}{n}\ x100\%$
- $ER = 100\% - SR$

Semua formula ini telah diverifikasi dengan data penelitian actual terhadap 15 router MikroTik CHR dan terbukti akurat dengan error margin mendekati nol persen, memvalidasi kebenaran representasi matematis algoritma CNCSA.

Baseline: Standard Netmiko Sequential Execution (NSM)

Evaluasi efektivitas optimasi pada algoritma CNCSA memerlukan metode pembanding yang berjalan tanpa mekanisme peningkatan kinerja. Metode dasar ini berfungsi sebagai acuan awal dan dikenal sebagai *Standard Netmiko Sequential Execution* (NSM). Dengan baseline tersebut, perbedaan performa CNCSA dapat dinilai secara objektif. Berikut untuk *Flowchart Standard Netmiko Sequential* (NSM).

Standard Netmiko Sequential Execution (NSM) - (*Baseline - Parameter Default*)

```
Start
[Initialize]

FOR each device (i=1 to n):

   1. Connect SSH:
   -  timeout: 100s (default)
   -  session_timeout: 60s
   -  keepalive: None
   -  No timing measurement

[Connection Success?] —No → [Skip device] → Continue
        Yes
   2. Send Command
   -  delay_factor: 1.0 (default)
   -  max_loops: 500 (default)
   -  strip_prompt: False
   -  No timing measurement

   3. Log to Database
   -  Unlimited message size
   -  Standard save() method
   -  No timing measurement

Continue to next device

END FOR
Calculate
   -  Total Time = end_time - start_time
   -  No detailed metrics
END
```

3.1.11. Defenisi dan Karakteristik Standard Netmiko Sequential Execution (NSM)

Standard Netmiko Sequential Execution (NSM) adalah metode konfigurasi jaringan yang menggunakan library Netmiko dengan parameter default tanpa

modifikasi atau optimasi apapun. NSM juga menggunakan model sequential execution sama seperti CNCSA, sehingga perbandingan kedua metode menjadi *fair* dan fokus pada perbedaan optimasi parameter.

1. Parameter default Nemiko
   a. Semua parameter menggunakan nilai default dari library Netmiko
   b. Tidak ada custom tuning untuk meningkatkan kecepatan
   c. Behavior mengikuti dokumentasi standar Netmiko
2. Pengukuran Waktu yang Terbatas
   a. Waktu yang dicatat hanya *connection time* (Tc) dan *send time* (Ts).
   b. *Logging time* (Tl) tidak diukur secara terpisah.
   c. Proses logging berjalan langsung tanpa *timing* tambahan.
3. Tanpa Optimasi Tambahan
   a. Tidak ada optimasi di sisi database.
   b. Tidak ada pembatasan ukuran *output message*
   c. Tidak ada tuning pada SSH *connection* parameters
4. Berfungsi sebagai *Baseline*
   a. Memberikan reference point untuk mengukur improvement CNCSA
   b. Merepresentasikan konfigurasi *out of the box* tanpa *tuning*
   c. Menggambarkan kondisi penulis menggunakan netmiko tanpa optimasi.

Implementasi NSM dijalankan pada server dengan alamat IP 10.151.20.147/25 dan spesifikasi hardware yang sama seperti server CNCSA. Struktur programnya tetap menggunakan pola django yang sama, hanya berbeda pada pengaturan Netmiko dan cara pengukuran waktunya.

3.1.12. Perbedaan NSM vs CNCSA

Bagian ini menjelaskan perbedaan utama antara NSM dan CNCSA. Perbandingan dibuat untuk menunjukkan bagaimana kedua metode bekerja dan apa saja aspek yang membedakannya. Tabel 3.4 berikut menyajikan perbedaan parameter yang digunakan oleh NSM dan CNCSA.

Tabel 3.4. Perbandingan Parameter NSM vs CNCSA

Aspek	Standard Netmiko (NSM)	CNCSA Optimized
Parameter Koneksi SSH		
Timeout	100 detik (default)	8 detik
Session Timeout	60 detik (default)	45 detik
Keepalive	None (default)	30 detik
Delay Factor	1.0 (default)	0.5
Max Loops (router CHR)	500 (default)	300

Strip Prompt	False (default)	TRUE
Logging Optimization		
Message Size Limit	Unlimited	5000 karakter
Database Method	Standard save()	Direct create()
Logging Time Measurement	Tidak diukur terpisah	Diukur (Tl)
Measurement Approach		
Connection Time (Tc)	Diukur	Diukur
Send Time (Ts)	Diukur	Diukur
Logging Time (Tl)	Tidak diukur	Diukur
System Overhead (ε)	Tidak dihitung	Dihitung
Efficiency Ratios	Tidak dihitung	Dihitung lengkap
Execution Model	Sequential	Sequential
Server Location	10.151.20.147/25	10.151.20.144/25

Perbedaan utama antara NSM dan CNCSA dapat dilihat dari cara kedua metode menangani *timeout*, *delay*, *looping*, *logging*, dan pengolahan *output*. Pada NSM, waktu *timeout* mengikuti standar Netmiko yaitu 100 detik, sedangkan CNCSA menggunakan *timeout* yang dipersingkat menjadi 8 detik sehingga perangkat yang bermasalah dapat terdeteksi lebih cepat. Untuk perangkat yang berjalan normal, waktu koneksinya tetap sama perbedaannya baru terlihat ketika ada gangguan pada perangkat atau jaringan. Dari sisi *delay*, NSM menggunakan *delay factor* 1.0 yang membuat jeda antar perintah lebih lama, sementara CNCSA memakai nilai 0.5 yang memotong waktu tunggu menjadi setengahnya. Dengan jumlah 150 perintah per perangkat, selisih ini menjadi akumulatif dan sangat berpengaruh pada total waktu eksekusi. Pada parameter *max loops*, NSM menggunakan nilai 500 sehingga proses menunggu respons berlangsung lebih lama, sedangkan CNCSA menggunakan nilai 300 atau 100 sehingga *timeout* dapat disimpulkan lebih cepat dan menghemat waktu ketika respons bermasalah. Untuk kegiatan logging, NSM tetap menyimpan log tetapi tidak memisahkan waktu logging sebagai metrik tersendiri, sedangkan CNCSA menghitung Tl secara terpisah sehingga lebih mudah melihat potensi bottleneck, meskipun ini merupakan perbedaan dari sisi pengukuran, bukan performa eksekusi. Terakhir, pada pengolahan *output*, NSM menyimpan hasil perintah secara penuh termasuk *prompt* dan *echo command*, sedangkan CNCSA membersihkan output dan membatasi panjangnya hingga 5000 karakter sehingga penggunaan penyimpanan dan bandwidth menjadi lebih efisien.

3.1.13. Formula Matematis NSM

Formula matematis untuk NSM lebih sederhana dibanding CNCSA karena tidak semua komponen diukur secara terpisah.

Total Connection Time (Tc)

$$T_c = \sum T_{c_i \; untuk \; i=1 \; sampai \; n} \quad (20)$$

Total Send Time (Ts)

$$T_s = \sum T_{s_i \; untuk \; i=1 \; sampai \; n} \quad (21)$$

Total Execution Time (Te)

NSM menghitung waktu total berdasarkan durasi dari awal proses sampai selesai, karena *Logging Time* (T_l) tidak dipisahkan:

$$T_{e_{nsm=}} end\_time - start\_time \quad (22)$$

Secara pendakatan:

$$T_{e_{NSM}} \approx T_c + T_s + T_{l_{unmeasured}} \quad (23)$$

Di mana T_l *unmeasured* adalah waktu *logging* yang terjadi tetapi tidak dihitung secara khusus.

Tidak Ada Formula untuk:

- *System Overhead* (ε) - tidak dihitung
- *Logging Time* (T_l) - tidak diukur terpisah
- *Efficiency Ratios* - tidak dihitung

Metrics yang Tersedia di NSM:

- *Total Connection Time* (Tc)
- *Total Send Time* (Ts)
- *Total Execution Time* (Te) ≈ *Elapsed Time*
- *Average Time per Device* (T_avg)
- *Success Rate* (%)

Berdasarkan data hasil pengujian NSM pada 15 perangkat dengan total 150 perintah, waktu koneksi yang tercatat adalah 8,4200 detik dan waktu pengiriman perintah mencapai 497,6300 detik. Karena NSM tidak memisahkan waktu *logging*, maka waktu eksekusi total dihitung dari penjumlahan Tc dan Ts, yaitu sekitar 506,05 detik. Dari nilai tersebut, rata-rata waktu per perangkat adalah 33,74 detik. Perhitungan ini juga dapat diverifikasi dengan melihat bahwa Te pada NSM mendekati nilai Tc + Ts, yaitu 8,42 + 497,63 yang hasilnya sama dengan 506,05

detik. Data hasil penelitian untuk Standard Netmiko *Sequential Execution* untuk 15 devices, 150 commands:

```
Tc = 8.4200 detik
Ts = 497.6300 detik
Te = Tc + Ts ≈ 506.05 detik (tanpa Tl eksplisit)
T_avg = 506.05 / 15 = 33.74 detik per device

Formula Verifikasi:
Te_NSM ≈ Tc + Ts
506.05 ≈ 8.42 + 497.63
506.05 ≈ 506.05
```

Perbandingan Formula NSM dan CNCSA dapat dilihat pada tabel 3.5.

Metric	Formula NSM	Formula CNCSA
$\boldsymbol{T_c}$	$\sum T_{c_i}$	$\sum T_{c_i}$
$\boldsymbol{T_s}$	$\sum T_{s_i}$	$\sum T_{s_i}$
$\boldsymbol{T_l}$	Tidak diukur	$\sum T_{l_i}$
$\boldsymbol{T_e}$	Elapsed Time	$T_c + T_s + T_l$
$\boldsymbol{\varepsilon}$	Tidak dihitung	Elapsed - T_e
$\boldsymbol{E_{conn}}$	Tidak dihitung	$\left(\frac{T_c}{T_e}\right) x\ 100\%$
$\boldsymbol{E_{config}}$	Tidak dihitung	$\left(\frac{T_s}{T_e}\right) x\ 100\%$
$\boldsymbol{E_{log}}$	Tidak dihitung	$(\frac{T_l}{T_e}) x\ 100\%$

Perbedaan utamanya adalah CNCSA memiliki pengukuran yang lebih rinci karena memisahkan waktu *logging* (Tl) dan menghitung rasio efisiensi untuk setiap komponen, sedangkan NSM hanya mencatat waktu total tanpa memecahnya ke bagian-bagian yang lebih detail.

NSM dan CNCSA sama-sama memakai model eksekusi berurutan, tetapi keduanya berbeda secara mendasar dalam alur kerja, cara pengukuran, dan bentuk perhitungan matematisnya. Pada NSM, alur digunakan dengan loop langsung tanpa fase yang terstruktur. Pendekatannya bersifat *ad-hoc*, pengukurannya hanya mencatat waktu total, tidak ada optimasi, dan rumus matematisnya juga sederhana karena hanya menggunakan elapsed time sebagai acuan. Sebaliknya, CNCSA memiliki struktur algoritma yang lebih teratur dengan empat fase, mulai dari inisialisasi, proses *loop* per perangkat, perhitungan metrik, hingga penyusunan output. Pendekatan ini menggunakan timing yang lebih lengkap karena memisahkan Tc, Ts, Tl, dan *system overhead*, disertai optimasi parameter pada setiap fase. Dari sisi kebaruan, CNCSA menawarkan *framework* multi-fase yang jelas, pengukuran waktu yang lebih detail, serta formulasi matematis yang menggambarkan hubungan tiap komponen, seperti Te = Tc + Ts + Tl, perhitungan

overhead, dan berbagai rasio efisiensi. Selain itu, CNCSA juga melakukan optimasi yang disesuaikan dengan karakteristik tiap fase, seperti deteksi kegagalan yang lebih cepat, jeda antar perintah yang lebih kecil, dan proses logging yang lebih efisien. Dengan struktur dan formula yang lengkap, algoritma ini lebih mudah direplikasi dan diverifikasi, karena semua perhitungan dan hasil metriknya dapat diuji kembali secara konsisten.

Desain Eksperimen

Bagian ini menjelaskan bagaimana eksperimen penelitian dirancang untuk membandingkan performa algoritma CNCSA dengan metode baseline Standard Netmiko Sequential Execution (NSM). Desain eksperimen dibuat secara sistematis agar hasil yang didapat valid, dapat dipercaya, dan dapat diulang oleh peneliti lain. Eksperimen mengikuti prinsip metode ilmiah dimana ada variabel yang dikontrol, ada metrik yang diukur, dan ada prosedur yang jelas. Tujuan utama eksperimen adalah membuktikan bahwa optimasi parameter yang dilakukan dalam CNCSA memberikan peningkatan performa yang signifikan dibanding penggunaan parameter default pada NSM, tanpa mengorbankan tingkat keberhasilan konfigurasi.

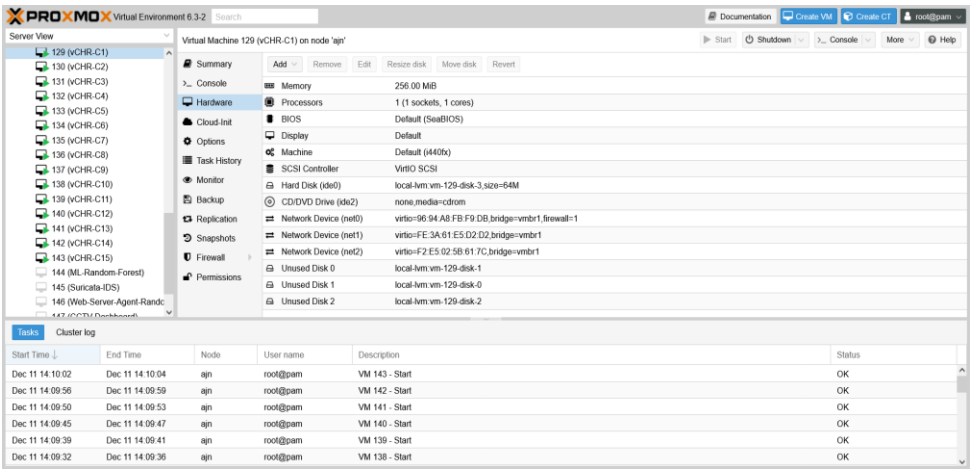
3.1.14. Skenario Pengujian

Eksperimen dirancang dengan skenario yang mencerminkan kondisi operasional nyata dan penulis akan melakukan konfigurasi terhadap 15 router CHR sekaligus.

3.1.14.1. Jumlah Perangkat yang Diuji

Penelitian menggunakan 15 router *Cloud Hosted Router* (CHR) sebagai perangkat yang dikonfigurasi. Pemilihan jumlah 15 perangkat didasarkan pada beberapa alasan praktis dan teknis. Pertama, jumlah 15 perangkat merepresentasikan skala *deployment* yang umum. Dalam dunia nyata, seorang administrator jaringan sering kali harus mengelola belasan hingga puluhan perangkat secara bersamaan, sehingga skenario ini realistis dan relevan dengan kebutuhan operasional sehari-hari. Kedua, jumlah 15 perangkat cukup besar untuk menunjukkan perbedaan performa yang dapat diukur dengan jelas antara CNCSA dan NSM. Jika hanya menggunakan 3-5 perangkat, perbedaan waktu eksekusi mungkin terlalu kecil dan sulit dibedakan dari variasi pengukuran normal. Sebaliknya, dengan 15 perangkat, penghematan waktu per-perangkat akan terakumulasi menjadi penghematan total yang signifikan dan mudah diamati. Ketiga, jumlah ini masih dapat dikelola dengan baik untuk eksperimen yang perlu diulang berkali-kali. Satu kali eksekusi membutuhkan waktu sekitar 5-10 menit, sehingga memungkinkan untuk melakukan pengujian berulang tanpa memakan waktu berlebihan. Hal ini penting untuk validasi statistik dimana eksperimen perlu dijalankan minimal 3 kali untuk memastikan hasil konsisten. 15 unit router CHR yang digunakan diberi nama vCHR-C1 sampai vCHR-C15. Semua router menjalankan sistem operasi MikroTik RouterOS versi 7.16 dengan spesifikasi hardware virtual yang identik yaitu

1 vCPU, 256MB RAM, dan 128MB disk. Router-router ini di-deploy dalam satu server fisik menggunakan Proxmox VE sebagai platform virtualisasi, sehingga kondisi *hardware* dan jaringan dapat dikontrol dengan baik. Topologi jaringan dirancang dengan semua router terhubung dalam satu virtual bridge (VMBR) bersama dengan kedua server CNCSA dan NSM dalam subnet 10.161.20.0/25. Konfigurasi ini menghasilkan latensi jaringan yang sangat rendah dan stabil sekitar 0.5-2 milidetik karena komunikasi terjadi di dalam satu host fisik tanpa melewati infrastruktur jaringan eksternal. Gambar 3.6 merupakan tampilan Proxmox VE *interface* menampilkan daftar 15 CHR (vCHR-C1 hingga vCHR-C15) dalam status *running*.



Gambar 3.6. Daftar 15 Router MikroTik CHR dalam Proxmox VE

3.1.14.2. Jumlah Command yang Dieksekusi

Setiap perangkat menerima 150 baris command konfigurasi firewall. Total command yang dieksekusi dalam satu kali eksperimen adalah 150 command dikali 15 perangkat, menghasilkan 2,250 baris *command* secara keseluruhan. Baris konfigurasi ini dibagi ke dalam tiga kategori dengan jumlah yang seimbang. Kategori pertama adalah 50 baris IP *Firewall Mangle* yang berfungsi untuk menandai paket jaringan berdasarkan kriteria tertentu seperti alamat sumber, port tujuan, dan protokol. Kategori kedua adalah 50 baris IP Firewall NAT yang mencakup konfigurasi untuk akses internet menggunakan *masquerade* dan port forwarding ke server internal. Kategori ketiga adalah 50 baris IP *Firewall Filter* yang mengimplementasikan kebijakan keamanan seperti mengizinkan koneksi yang sudah terbentuk, membuka layanan tertentu, dan memblokir *traffic* yang tidak diinginkan.

Pemilihan 150 command per perangkat merepresentasikan kompleksitas konfigurasi tingkat menengah. Jumlah ini tidak terlalu sederhana seperti 10-20 rules yang mungkin selesai dalam hitungan detik sehingga sulit mengukur perbedaan dengan akurat, tetapi juga tidak terlalu

rumit seperti 500-1000 rules yang akan memakan waktu sangat lama. Dengan 150 *command*, waktu eksekusi cukup panjang untuk diukur dengan presisi tinggi, namun tidak terlalu lama sehingga eksperimen dapat diulang berkali-kali dalam waktu yang wajar. Konfigurasi *firewall* dipilih sebagai jenis command karena merupakan salah satu tugas paling umum dalam administrasi jaringan. *Firewall rules* sering diupdate untuk mengakomodasi perubahan kebijakan keamanan atau kebutuhan akses baru. Selain itu, eksekusi *firewall command* relatif cepat dan *predictable* tanpa memerlukan waktu *processing* yang lama dari perangkat, sehingga cocok untuk pengukuran performa yang fokus pada efisiensi pengiriman *command*.

3.1.14.3. Template Command yang Digunakan

Semua perangkat menerima command yang identik untuk memastikan fair dalam perbandingan. Dengan menggunakan command yang sama persis, perbedaan performa yang terukur murni berasal dari perbedaan algoritma dan optimasi parameter, bukan dari perbedaan kompleksitas atau karakteristik command. Command dirancang dengan struktur yang representatif untuk lingkungan produksi namun tetap sederhana untuk keperluan replikasi. Untuk kategori Mangle, command menambahkan rule dengan chain prerouting untuk menandai koneksi dan paket berdasarkan alamat sumber dan port tujuan seperti yang ditampilkan pada baris skrip dibawah ini

```
/ip firewall mangle
add action=mark-packet chain=prerouting dst-port=80 new-packet-
mark=traffic_high_priority passthrough=yes protocol=tcp src
address=192.169.1.0/24
add action=mark-packet chain=prerouting dst-port=8080 new-packet-
mark=traffic_high_priority passthrough=yes protocol=tcp src-
address=192.169.2.0/24
add action=mark-packet chain=prerouting dst-port=80 new-packet-
mark=traffic_high_priority passthrough=yes protocol=tcp src-
address=192.169.3.0/24
add action=mark-packet chain=prerouting dst-port=8080 new-packet-
mark=traffic_high_priority passthrough=yes protocol=tcp src-
address=192.169.4.0/24
add action=mark-packet chain=prerouting dst-port=80 new-packet-
mark=traffic_high_priority passthrough=yes protocol=tcp src-
address=192.169.5.0/24
add action=mark-packet chain=prerouting dst-port=8080 new-packet-
mark=traffic_high_priority passthrough=yes protocol=tcp src-
address=192.169.6.0/24
```

```
add action=mark-packet chain=prerouting dst-port=80 new-packet-
mark=traffic_high_priority passthrough=yes protocol=tcp src-
address=192.169.7.0/24
add action=mark-packet chain=prerouting dst-port=8080 new-packet-
mark=traffic_high_priority passthrough=yes protocol=tcp src-
address=192.169.8.0/24
add action=mark-packet chain=prerouting dst-port=80 new-packet-
mark=traffic_high_priority passthrough=yes protocol=tcp src-
address=192.169.9.0/24
add action=mark-packet chain=prerouting dst-port=8080 new-packet-
mark=traffic_high_priority passthrough=yes protocol=tcp src-
address=192.169.10.0/24
add action=mark-packet chain=prerouting dst-port=80 new-packet-
mark=traffic_high_priority passthrough=yes protocol=tcp src-
address=192.169.11.0/24
add action=mark-packet chain=prerouting dst-port=8080 new-packet-
mark=traffic_high_priority passthrough=yes protocol=tcp src-
address=192.169.12.0/24
add action=mark-packet chain=prerouting dst-port=80 new-packet-
mark=traffic_high_priority passthrough=yes protocol=tcp src-
address=192.169.13.0/24
add action=mark-packet chain=prerouting dst-port=8080 new-packet-
mark=traffic_high_priority passthrough=yes protocol=tcp src-
address=192.169.14.0/24
add action=mark-packet chain=prerouting dst-port=80 new-packet-
mark=traffic_high_priority passthrough=yes protocol=tcp src-
address=192.169.15.0/24
add action=mark-packet chain=prerouting dst-port=8080 new-packet-
mark=traffic_high_priority passthrough=yes protocol=tcp src-
address=192.169.16.0/24
add action=mark-packet chain=prerouting dst-port=80 new-packet-
mark=traffic_high_priority passthrough=yes protocol=tcp src-
address=192.169.17.0/24
add action=mark-packet chain=prerouting dst-port=8080 new-packet-
mark=traffic_high_priority passthrough=yes protocol=tcp src-
address=192.169.18.0/24
add action=mark-packet chain=prerouting dst-port=80 new-packet-
mark=traffic_high_priority passthrough=yes protocol=tcp src-
address=192.169.19.0/24
add action=mark-packet chain=prerouting dst-port=8080 new-packet-
mark=traffic_high_priority passthrough=yes protocol=tcp src-
address=192.169.20.0/24
```

Pada kategori NAT, perintah yang digunakan meliputi aturan masquerade untuk traffic keluar dan destination NAT yang berfungsi melakukan port forwarding, ditampilkan pada skrip template command berikut ini.

```
/ip firewall nat
add action=masquerade chain=srcnat src-address=192.169.1.0/24
add action=masquerade chain=srcnat src-address=192.169.2.0/24
add action=masquerade chain=srcnat src-address=192.169.3.0/24
add action=masquerade chain=srcnat src-address=192.169.4.0/24
add action=masquerade chain=srcnat src-address=192.169.5.0/24
add action=masquerade chain=srcnat src-address=192.169.6.0/24
add action=masquerade chain=srcnat src-address=192.169.7.0/24
add action=masquerade chain=srcnat src-address=192.169.8.0/24
add action=masquerade chain=srcnat src-address=192.169.9.0/24
add action=masquerade chain=srcnat src-address=192.169.10.0/24
```

```
add action=masquerade chain=srcnat src-address=192.169.11.0/24
add action=masquerade chain=srcnat src-address=192.169.12.0/24
add action=masquerade chain=srcnat src-address=192.169.13.0/24
add action=masquerade chain=srcnat src-address=192.169.14.0/24
add action=masquerade chain=srcnat src-address=192.169.15.0/24
add action=masquerade chain=srcnat src-address=192.169.16.0/24
add action=masquerade chain=srcnat src-address=192.169.17.0/24
add action=masquerade chain=srcnat src-address=192.169.18.0/24
add action=masquerade chain=srcnat src-address=192.169.19.0/24
add action=masquerade chain=srcnat src-address=192.169.20.0/24
add action=masquerade chain=srcnat src-address=192.169.21.0/24
add action=masquerade chain=srcnat src-address=192.169.22.0/24
add action=masquerade chain=srcnat src-address=192.169.23.0/24
add action=masquerade chain=srcnat src-address=192.169.24.0/24
add action=masquerade chain=srcnat src-address=192.169.25.0/24
add action=masquerade chain=srcnat src-address=192.169.26.0/24
add action=masquerade chain=srcnat src-address=192.169.27.0/24
add action=masquerade chain=srcnat src-address=192.169.28.0/24
add action=masquerade chain=srcnat src-address=192.169.29.0/24
add action=masquerade chain=srcnat src-address=192.169.30.0/24
add action=masquerade chain=srcnat src-address=192.169.31.0/24
add action=masquerade chain=srcnat src-address=192.169.32.0/24
add action=masquerade chain=srcnat src-address=192.169.33.0/24
add action=masquerade chain=srcnat src-address=192.169.34.0/24
add action=masquerade chain=srcnat src-address=192.169.35.0/24
add action=masquerade chain=srcnat src-address=192.169.36.0/24
add action=masquerade chain=srcnat src-address=192.169.37.0/24
add action=masquerade chain=srcnat src-address=192.169.38.0/24
add action=masquerade chain=srcnat src-address=192.169.39.0/24
add action=masquerade chain=srcnat src-address=192.169.40.0/24
add action=masquerade chain=srcnat src-address=192.169.41.0/24
add action=masquerade chain=srcnat src-address=192.169.42.0/24
add action=masquerade chain=srcnat src-address=192.169.43.0/24
add action=masquerade chain=srcnat src-address=192.169.44.0/24
add action=masquerade chain=srcnat src-address=192.169.45.0/24
add action=masquerade chain=srcnat src-address=192.169.46.0/24
add action=masquerade chain=srcnat src-address=192.169.47.0/24
add action=masquerade chain=srcnat src-address=192.169.48.0/24
add action=masquerade chain=srcnat src-address=192.169.49.0/24
add action=masquerade chain=srcnat src-address=192.169.50.0/24
```

Template untuk kategori *Filter*, *command* mengimplementasikan kebijakan keamanan standar seperti menerima koneksi *established*, dengan mengizinkan *traffic* dari jaringan tertentu menuju layanan web yang berjalan di port 80 (HTTP) dan 8080 (*web service* alternatif). Aturan ini berada di *chain forward*, sehingga mengatur lalu lintas yang melewati router, bukan traffic ke router itu sendiri.

```
/ip firewall filter
add chain=forward action=accept protocol=tcp dst-port=80 src-
address=192.169.1.0/24
add chain=forward action=accept protocol=tcp dst-port=8080 src-
address=192.169.2.0/24
add chain=forward action=accept protocol=tcp dst-port=80 src-
address=192.169.3.0/24
add chain=forward action=accept protocol=tcp dst-port=8080 src-
address=192.169.4.0/24
add chain=forward action=accept protocol=tcp dst-port=80 src-
address=192.169.5.0/24
add chain=forward action=accept protocol=tcp dst-port=8080 src-
address=192.169.6.0/24
add chain=forward action=accept protocol=tcp dst-port=80 src-
address=192.169.7.0/24
```

```
/ip firewall filter
add chain=forward action=accept protocol=tcp dst-port=8080 src-
address=192.169.8.0/24
add chain=forward action=accept protocol=tcp dst-port=80 src-
address=192.169.9.0/24
add chain=forward action=accept protocol=tcp dst-port=8080 src-
address=192.169.10.0/24
add chain=forward action=accept protocol=tcp dst-port=80 src-
address=192.169.11.0/24
add chain=forward action=accept protocol=tcp dst-port=8080 src-
address=192.169.12.0/24
add chain=forward action=accept protocol=tcp dst-port=80 src-
address=192.169.13.0/24
add chain=forward action=accept protocol=tcp dst-port=8080 src-
address=192.169.14.0/24
add chain=forward action=accept protocol=tcp dst-port=80 src-
address=192.169.15.0/24
add chain=forward action=accept protocol=tcp dst-port=8080 src-
address=192.169.16.0/24
add chain=forward action=accept protocol=tcp dst-port=80 src-
address=192.169.17.0/24
add chain=forward action=accept protocol=tcp dst-port=8080 src-
address=192.169.18.0/24
add chain=forward action=accept protocol=tcp dst-port=80 src-
address=192.169.19.0/24
add chain=forward action=accept protocol=tcp dst-port=8080 src-
address=192.169.20.0/24
```

```
add chain=forward action=accept protocol=tcp dst-port=80 src-
address=192.169.21.0/24
add chain=forward action=accept protocol=tcp dst-port=8080 src-
address=192.169.22.0/24
add chain=forward action=accept protocol=tcp dst-port=80 src-
address=192.169.23.0/24
add chain=forward action=accept protocol=tcp dst-port=8080 src-
address=192.169.24.0/24
add chain=forward action=accept protocol=tcp dst-port=80 src-
address=192.169.25.0/24
add chain=forward action=accept protocol=tcp dst-port=8080 src-
address=192.169.26.0/24
add chain=forward action=accept protocol=tcp dst-port=80 src-
address=192.169.27.0/24
add chain=forward action=accept protocol=tcp dst-port=8080 src-
address=192.169.28.0/24
add chain=forward action=accept protocol=tcp dst-port=80 src-
address=192.169.29.0/24
add chain=forward action=accept protocol=tcp dst-port=8080 src-
address=192.169.30.0/24
add chain=forward action=accept protocol=tcp dst-port=80 src-
address=192.169.31.0/24
add chain=forward action=accept protocol=tcp dst-port=8080 src-
address=192.169.32.0/24
add chain=forward action=accept protocol=tcp dst-port=80 src-
address=192.169.33.0/24
add chain=forward action=accept protocol=tcp dst-port=8080 src-
address=192.169.34.0/24
add chain=forward action=accept protocol=tcp dst-port=80 src-
address=192.169.35.0/24
```

```
add chain=forward action=accept protocol=tcp dst-port=8080 src-
address=192.169.36.0/24
add chain=forward action=accept protocol=tcp dst-port=80 src-
address=192.169.37.0/24
add chain=forward action=accept protocol=tcp dst-port=8080 src-
address=192.169.38.0/24
add chain=forward action=accept protocol=tcp dst-port=80 src-
address=192.169.39.0/24
add chain=forward action=accept protocol=tcp dst-port=8080 src-
address=192.169.40.0/24
add chain=forward action=accept protocol=tcp dst-port=80 src-
address=192.169.41.0/24
add chain=forward action=accept protocol=tcp dst-port=8080 src-
address=192.169.42.0/24
add chain=forward action=accept protocol=tcp dst-port=80 src-
address=192.169.43.0/24
add chain=forward action=accept protocol=tcp dst-port=8080 src-
address=192.169.44.0/24
add chain=forward action=accept protocol=tcp dst-port=80 src-
address=192.169.45.0/24
add chain=forward action=accept protocol=tcp dst-port=8080 src-
address=192.169.46.0/24
add chain=forward action=accept protocol=tcp dst-port=80 src-
address=192.169.47.0/24
add chain=forward action=accept protocol=tcp dst-port=8080 src-
address=192.169.48.0/24
add chain=forward action=accept protocol=tcp dst-port=80 src-
address=192.169.49.0/24
add chain=forward action=accept protocol=tcp dst-port=8080 src-
address=192.169.50.0/24
```

Setiap baris perintah untuk skrip, sudah divalidasi sintaksnya dan diuji terlebih dahulu pada satu router CHR percobaan untuk memastikan tidak ada kesalahan yang bisa menyebabkan error saat eksekusi. *Command* disimpan dalam file teks yang kemudian di-paste ke antarmuka web saat eksperimen berlangsung, memastikan *command set* yang sama digunakan untuk pengujian CNCSA dan NSM.

3.1.14.4. Kondisi Awal Perangkat

Sebelum setiap kali eksperimen, semua perangkat direset ke kondisi bersih (*clean state*) dengan menghapus seluruh *firewall rules* yang ada. Prosedur reset ini sangat penting untuk memastikan *reproducibility* karena jika ada sisa konfigurasi dari pengujian sebelumnya, kondisi awal tidak konsisten dan bisa mempengaruhi pengukuran waktu. Reset dilakukan dengan menjalankan command untuk menghapus semua rule pada setiap kategori *firewall*. Setelah reset, dilakukan verifikasi dengan menjalankan command print untuk memastikan tidak ada rule yang tersisa. Hanya setelah verifikasi menunjukkan hasil kosong, eksperimen baru dapat dimulai. Pendekatan *clean state* ini memastikan bahwa setiap sesi pengujian dimulai dari kondisi baseline yang sama, sehingga variasi hasil hanya berasal dari fluktuasi *inherent* dalam waktu eksekusi seperti variasi kecil dalam beban CPU atau latensi jaringan, bukan dari perbedaan kondisi awal perangkat.

3.1.14.5. Kondisi Jaringan

Lingkungan jaringan dirancang untuk meminimalkan variabel eksternal yang bisa mempengaruhi pengukuran waktu. Semua perangkat dan server berada dalam satu host Proxmox sehingga komunikasi terjadi melalui virtual *bridge* tanpa melewati *switch* atau router fisik. Hal ini menghasilkan latensi yang sangat rendah dan dapat diprediksi. Traffic jaringan eksternal yang bisa menyebabkan kongesti atau interferensi diminimalkan dengan memastikan tidak ada aplikasi atau layanan lain yang secara aktif menggunakan bandwidth jaringan selama sesi pengujian. Stabilitas jaringan diverifikasi sebelum setiap pengujian dengan melakukan ping test ke semua perangkat untuk mengukur latensi *baseline* dan memastikan semua perangkat dapat dijangkau. Latensi *round-trip time* yang diamati biasanya dalam range 0.5-2 milidetik dengan jitter minimal, menunjukkan kondisi jaringan yang stabil. Pengujian tidak dilakukan jika ada anomali dalam latensi jaringan atau *packet loss* yang bisa mengacaukan hasil. Lingkungan jaringan yang terkontrol ini memastikan bahwa perbedaan waktu eksekusi antara CNCSA dan NSM benar-benar disebabkan oleh perbedaan algoritma dan parameter, bukan oleh variabilitas jaringan.

3.1.15. Metrik Evaluasi

Eksperimen mengukur berbagai metrik untuk mengevaluasi performa algoritma CNCSA dibandingkan dengan NSM secara menyeluruh. Metrik-metrik ini dibagi menjadi beberapa kategori yaitu metrik waktu, metrik efisiensi, metrik perbandingan, dan metrik keandalan.

3.1.15.1. Metrik Waktu (*Time Metrics*)

Metrik waktu mengukur durasi eksekusi untuk berbagai komponen dan fase dari proses konfigurasi, memberikan dasar kuantitatif untuk perbandingan performa.

A. Total Connection Time (Tc)

Total Connection Time adalah total waktu yang dibutuhkan untuk membuat koneksi SSH ke semua perangkat yang dikonfigurasi. Waktu ini dihitung dengan menjumlahkan waktu koneksi dari setiap perangkat, dimana waktu koneksi satu perangkat mencakup proses TCP *handshake*, negosiasi protokol SSH, autentikasi menggunakan username dan password, sampai koneksi siap menerima command. Metrik ini diukur baik di CNCSA maupun di NSM untuk memungkinkan perbandingan performa fase koneksi. Perbedaan Tc antara kedua metode berasal dari perbedaan parameter koneksi seperti nilai timeout dan pengaturan *keepalive*. Nilai Tc yang lebih rendah menunjukkan proses koneksi yang lebih efisien. Dalam hasil eksperimen penelitian dengan 15 perangkat, CNCSA

mencatat Tc sebesar 4.47 detik sedangkan NSM mencatat 8.42 detik, menunjukkan CNCSA lebih cepat 46.91 persen dalam fase koneksi.

B. Total Send Time (Ts)

Total *Send Time* adalah total waktu yang dibutuhkan untuk mengirim dan mengeksekusi semua *command* ke semua perangkat. Waktu ini dihitung dengan menjumlahkan waktu send dari setiap perangkat, dimana waktu send satu perangkat mencakup durasi mulai dari pengiriman *command* pertama hingga selesainya *command* terakhir. Metrik ini merupakan komponen paling dominan dalam total waktu eksekusi karena mayoritas pekerjaan yang dilakukan adalah mengirim dan mengeksekusi command konfigurasi. Ts diukur baik di CNCSA maupun NSM dan biasanya menyumbang lebih dari 95 persen dari total waktu eksekusi. Perbedaan Ts antara CNCSA dan NSM terutama berasal dari optimasi parameter eksekusi *command*, khususnya *delay_factor* yang mengontrol jeda antar command. CNCSA menggunakan *delay_factor* 0.5 yang mengurangi waktu tunggu antar *command* menjadi setengah dibanding NSM yang menggunakan *default* 1.0, menghasilkan penghematan waktu yang signifikan ketika dikalikan dengan 150 command per perangkat dan 15 perangkat total. Dalam hasil eksperimen penelitian, CNCSA mencatat Ts sebesar 297.18 detik sedangkan NSM mencatat 497.63 detik, menunjukkan CNCSA lebih cepat 40.27 persen dalam fase pengiriman *command*.

C. Total Logging Time (Tl)

Total *Logging Time* adalah total waktu yang dibutuhkan untuk menyimpan hasil eksekusi ke database untuk semua perangkat. Waktu ini dihitung dengan menjumlahkan waktu logging dari setiap perangkat, dimana waktu logging mencakup operasi database insert untuk menyimpan status konfigurasi, pesan output, dan informasi timing. Metrik ini hanya diukur secara eksplisit di CNCSA sebagai komponen terpisah. NSM melakukan logging secara inline tanpa pengukuran waktu terpisah sehingga Tl termasuk dalam keseluruhan waktu eksekusi tetapi tidak terlihat sebagai metrik tersendiri. Pengukuran Tl secara eksplisit dalam CNCSA memberikan visibilitas terhadap overhead operasi database dan memungkinkan identifikasi apakah logging menjadi hambatan, terutama pada skala yang lebih besar. Dalam eksperimen dengan 15 perangkat, Tl CNCSA hanya 0.83 detik, menunjukkan operasi database sangat cepat karena optimasi seperti pembatasan ukuran pesan dan metode create langsung.

D. Total Execution Time (Te)

Total *Execution Time* adalah waktu bersih untuk menyelesaikan seluruh tugas konfigurasi. Cara perhitungan berbeda antara CNCSA dan NSM karena perbedaan pendekatan pengukuran.

Untuk CNCSA, Te dihitung dengan formula Te = Tc + Ts + Tl, yang menjumlahkan semua komponen yang diukur untuk mendapatkan waktu produktif bersih. Untuk NSM, Te didekati sebagai elapsed time karena Tl tidak diukur terpisah, sehingga Te_NSM ≈ Elapsed Time. Te adalah metrik utama untuk perbandingan performa antara kedua metode karena merepresentasikan total waktu yang dibutuhkan untuk menyelesaikan tugas konfigurasi. Nilai Te yang lebih rendah menunjukkan algoritma yang lebih efisien dalam menyelesaikan tugas dalam durasi yang lebih singkat. Dalam eksperimen, CNCSA mencatat Te sebesar 302.49 detik sedangkan NSM mencatat Te sebesar 506.05 detik, menunjukkan CNCSA lebih cepat 40.23 persen secara keseluruhan.

E. System Overhead (ε - Epsilon)

System Overhead epsilon merepresentasikan durasi waktu yang tidak terhitung dalam pengukuran eksplisit dari ketiga komponen utama yaitu Tc, Ts, dan Tl. Nilai epsilon diperoleh melalui formula `ε = Elapsed_Time - Te`, dimana `Elapsed_Time` adalah durasi aktual yang tercatat dari pengukuran jam sistem mulai dari titik awal hingga titik akhir proses eksekusi. Metrik ini hanya dapat dihitung dalam algoritma CNCSA karena mensyaratkan adanya pengukuran tersendiri yang eksplisit untuk seluruh tiga komponen waktu tersebut. Epsilon mencakup overhead yang berasal dari berbagai sumber dalam sistem, antara lain pemrosesan yang dilakukan oleh framework Django untuk keperluan routing permintaan dan rendering respons, *overhead* yang dihasilkan oleh interpreter Python dalam melakukan assignment terhadap variabel dan pemanggilan terhadap fungsi, *overhead* yang muncul dari sistem operasi dalam melakukan *context switching* dan penjadwalan proses, serta *overhead* yang berasal dari mekanisme pengukuran *timing* itu sendiri.

Secara ideal, nilai epsilon seharusnya menjadi fraksi yang sangat kecil dari Te, yang mengindikasikan bahwa overhead dari framework dan sistem berada pada level minimal. Dalam hasil implementasi CNCSA, epsilon secara konsisten berada di bawah 0.1 persen dari Te, yang menunjukkan bahwa implementasi dilakukan dengan tingkat efisiensi yang tinggi tanpa adanya komponen berlebih yang tidak diperlukan. Berdasarkan data eksperimen terbaru dengan 15 router MikroTik CHR, epsilon tercatat sebesar 0.0338 detik atau setara dengan 0.0112 persen dari total execution time sebesar 302.4904 detik. Nilai yang sangat kecil ini membuktikan bahwa sistem beroperasi dengan efisiensi yang sangat tinggi, dimana waktu yang tersita untuk *overhead framework* dan sistem berada pada tingkat yang dapat diabaikan. Pencapaian *System Overhead Ratio* di bawah 0.1 persen mengkonfirmasi bahwa pengukuran waktu dilakukan dengan akurat dan *framework* Django

yang digunakan memiliki performa pemrosesan yang sangat cepat untuk keperluan *routing* dan *rendering*.

F. Average Time per Device (T_avg)

Average Time per *Device* adalah rata-rata waktu yang dibutuhkan untuk mengkonfigurasi satu perangkat. Metrik ini dihitung dengan formula `T_avg = Te / n`, dimana n adalah jumlah perangkat. Metrik ini berguna untuk memahami durasi tipikal yang dibutuhkan untuk satu perangkat dan untuk tujuan perbandingan. Dalam eksperimen dengan 15 perangkat, CNCSA mencatat `T_avg` sebesar 22.68 detik per perangkat sedangkan NSM mencatat 33.74 detik per perangkat, menunjukkan CNCSA menghemat 11.06 detik per perangkat.

3.1.15.2. Metrik Efisiensi (Efficiency Metrics)

Metrik efisiensi mengukur proporsi waktu yang digunakan untuk berbagai komponen terhadap total waktu eksekusi, memberikan wawasan tentang distribusi waktu dan mengidentifikasi potensi hambatan. Metrik-metrik ini hanya dihitung dalam CNCSA karena memerlukan pengukuran eksplisit dari semua komponen waktu.

A. Connection Efficiency (E_conn)

Connection Efficiency mengukur persentase waktu yang dialokasikan untuk membuat koneksi SSH terhadap total waktu eksekusi. Formula perhitungannya adalah `E_conn = (Tc / Te) × 100%`. Target ideal untuk `E_conn` adalah kurang dari 5 persen, menunjukkan bahwa overhead koneksi minimal dan mayoritas waktu digunakan untuk pekerjaan konfigurasi produktif. Dalam hasil CNCSA dengan 15 perangkat dan parameter koneksi yang dioptimasi, `E_conn` tercatat sekitar 1.48 persen, menunjukkan fase koneksi sangat efisien dan tidak menjadi hambatan. Nilai `E_conn` yang rendah ini dicapai melalui optimasi nilai *timeout* untuk deteksi kegagalan yang lebih cepat dan mekanisme *keepalive* untuk menjaga koneksi tetap stabil.

B. Configuration Efficiency (E_config)

Configuration Efficiency mengukur persentase waktu yang dialokasikan untuk mengirim dan mengeksekusi command konfigurasi terhadap total waktu eksekusi. Formula perhitungannya adalah `E_config = (Ts / Te) × 100%`. Target ideal untuk `E_config` adalah lebih besar dari 90 persen, menunjukkan mayoritas waktu eksekusi dihabiskan untuk

pekerjaan produktif aktual yaitu mengkonfigurasi perangkat. Dalam hasil CNCSA, `E_config` tercatat sekitar 98.25 persen, menunjukkan algoritma sangat efisien dengan overhead minimal dan mayoritas waktu digunakan untuk tujuan yang dimaksud. Nilai `E_config` yang tinggi adalah karakteristik yang diinginkan karena menunjukkan eksekusi yang fokus tanpa waktu terbuang pada operasi non-produktif.

C. Logging Efficiency (E_log)

Logging Efficiency mengukur persentase waktu yang dialokasikan untuk operasi logging database terhadap total waktu eksekusi. Formula perhitungannya adalah `E_log = (Tl / Te) × 100%`. Target ideal untuk `E_log` adalah kurang dari 1 persen, menunjukkan operasi logging sangat cepat dan tidak menjadi hambatan dalam alur eksekusi. Dalam hasil CNCSA dengan mekanisme *logging* yang dioptimasi menggunakan pembatasan ukuran pesan dan metode create langsung, `E_log` tercatat sekitar 0.28 persen, menunjukkan logging yang sangat efisien dengan *overhead* yang dapat diabaikan. Nilai `E_log` yang rendah penting terutama pada skala yang lebih besar dimana operasi database berpotensi menjadi hambatan jika tidak dioptimasi dengan baik.

D. CNCSA Execution Efficiency (E_CNCSA)

CNCSA *Execution Efficiency* mengukur total persentase waktu yang dialokasikan untuk pekerjaan produktif yaitu konfigurasi plus *logging*, tidak termasuk overhead koneksi. Formula perhitungannya adalah `E_CNCSA = (Ts + Tl) / Te × 100%`, atau dapat juga dihitung sebagai `E_config + E_log`. Target ideal untuk `E_CNCSA` adalah lebih besar dari 95 persen, menunjukkan overhead non-produktif minimal. Dalam hasil CNCSA, `E_CNCSA` tercatat sekitar 98.52 persen, menunjukkan eksekusi yang sangat efisien dengan *overhead* koneksi diminimalkan. Nilai E_CNCSA yang tinggi memvalidasi bahwa desain algoritma berhasil dalam meminimalkan *overhead* dan memaksimalkan rasio pekerjaan produktif.

E. System Overhead Ratio (SOR)

System Overhead Ratio mengukur persentase *overhead* dari *framework* dan sistem terhadap total waktu eksekusi. Formula perhitungannya adalah `SOR = (ε / Te) × 100%`. Target ideal untuk SOR adalah kurang dari 1 persen, lebih disukai mendekati 0 persen, menunjukkan bloat minimal dari infrastruktur framework atau sistem. Dalam hasil CNCSA, SOR tercatat sangat rendah sekitar 0.01 persen, menunjukkan framework Django dan implementasi Python sangat efisien

dengan overhead yang dapat diabaikan. Nilai SOR yang mendekati nol memvalidasi bahwa pengukuran akurat dengan fungsi timing yang menambahkan overhead minimal, dan bahwa pemrosesan *framework* untuk *routing* dan *rendering* sangat cepat.

3.1.15.3. Metrik Perbandingan

Metrik perbandingan mengukur perbedaan performa antara CNCSA dan NSM secara kuantitatif, memberikan ukuran objektif untuk mengevaluasi efektivitas optimasi yang diimplementasikan dalam CNCSA.

A. Persentase Pengurangan Waktu

Persentase Pengurangan Waktu mengukur seberapa besar pengurangan proporsional dalam waktu eksekusi yang dicapai oleh CNCSA dibandingkan dengan NSM. Formula perhitungannya adalah:

$$Improvement\% = \left(\left(\frac{T_{e_{NSM}} - T_{e_{CNCSA}}}{T_{e_{NSM}}}\right)\right) X100\% \quad (24)$$

Metrik ini menyatakan peningkatan sebagai persentase pengurangan, dimana nilai positif menunjukkan CNCSA lebih cepat dari NSM. Target minimal untuk Improvement Percentage adalah 20 persen, menunjukkan manfaat praktis yang signifikan dari optimasi.

Dalam hasil penelitian aktual dengan 15 perangkat dan 150 command, CNCSA mencapai improvement percentage sekitar 40.22 persen, berarti CNCSA menyelesaikan tugas konfigurasi dalam waktu 40.22 persen lebih sedikit dibandingkan NSM. Hasil ini menunjukkan penghematan waktu 203.56 detik dari total 506.05 detik waktu eksekusi NSM, merepresentasikan peningkatan efisiensi yang substansial dengan penghematan lebih dari 3 menit per eksekusi. Perhitungan:

```
Improvement% = (506.05 - 302.49) / 506.05 × 100%
             = 203.56 / 506.05 × 100%
             = 40.22%
```

B. Speed Ratio

Interpretasinya adalah bahwa SR lebih besar dari 1.0 menunjukkan CNCSA lebih cepat, dengan nilai yang lebih tinggi menunjukkan speedup yang lebih besar. Sebagai contoh, SR sebesar

1.5 berarti CNCSA 1.5 kali lebih cepat atau menyelesaikan tugas dalam dua-pertiga waktu yang dibutuhkan NSM. Dari hasil eksperimen penelitian, CNCSA mencapai speedup ratio sekitar 1.67 kali, berarti CNCSA lebih dari 60 persen lebih cepat dari NSM. Speedup ini signifikan dalam istilah praktis karena untuk tugas yang awalnya memakan 506 detik di NSM, CNCSA menyelesaikannya hanya dalam 302 detik, menghemat lebih dari 3.4 menit per eksekusi.

Speedup Ratio mengukur berapa kali lebih cepat CNCSA dibandingkan dengan NSM dalam istilah absolut. Formula perhitungannya adalah:

$$SR = \frac{T_{e_{NSM}}}{T_{e_{CNCSA}}} \quad (25)$$

Perhitungan:

```
Speedup Ratio = 506.05 / 302.49 = 1.67x
```

C. Waktu yang Dihemat (Time Saved - ΔT)

Waktu yang Dihemat mengukur pengurangan absolut dalam waktu eksekusi yang dicapai oleh CNCSA. Formula perhitungannya adalah:

$$\Delta T = T_{e_{NSM}} - T_{e_{CNCSA}} \quad (26)$$

Metrik ini memberikan ukuran manfaat dalam istilah dunia nyata, menunjukkan waktu jam aktual yang dihemat per eksekusi. Dalam hasil penelitian, CNCSA menghemat 203.56 detik atau sekitar 3.39 menit dibandingkan NSM untuk mengkonfigurasi 15 perangkat dengan 150 command per perangkat.

Perhitungan:

```
ΔT = 506.05 - 302.49 = 203.56 detik ≈ 3.39
menit
```

D. Peningkatan Per Komponen (Router CHR)

Metrik peningkatan per komponen digunakan untuk melihat peningkatan kinerja pada setiap tahapan proses secara terpisah. Dengan pemisahan ini, dapat diketahui bagian mana yang mengalami peningkatan paling besar dan memberikan kontribusi paling signifikan terhadap peningkatan performa secara keseluruhan. *Connection Time Improvement* dihitung sebagai:

$$\frac{T_{c_{NSM}} - T_{c_{CNCSA}}}{T_{c_{NSM}}} x\ 100\% \qquad (27)$$

Dalam hasil penelitian, *Connection Time Improvement* sekitar 46.86 persen dengan penghematan waktu 3.95 detik.

Perhitungan:

```
Tc Improvement  = (8.42 - 4.47) / 8.42 × 100%
                = 3.95 / 8.42 × 100%
                = 46.86%
```

Send Time Improvement dihitung sebagai*:*

$$\frac{T_{s_{NSM}} - T_{s_{CNCSA}}}{T_{s_{NSM}}} x\ 100\% \qquad (28)$$

Dalam hasil penelitian, *Send Time Improvement* sekitar 40.28 persen dengan penghematan waktu 200.44 detik.

```
Ts Improvement  = (497.63 - 297.19) / 497.63
× 100%
                = 200.44 / 497.63 × 100%
                = 40.28%
```

Hasil analisis memperlihatkan bahwa walaupun peningkatan pada fase koneksi terlihat lebih besar secara persentase, pengurangan waktu pada tahap pengiriman perintah justru memberikan dampak paling besar terhadap total penghematan waktu. Hal ini terjadi karena fase pengiriman merupakan bagian yang paling banyak menyita waktu selama proses eksekusi. Pemecahan analisis per komponen ini membantu menunjukkan bahwa optimasi pada tahapan dengan durasi terpanjang akan menghasilkan penghematan waktu yang paling signifikan, meskipun persentase peningkatannya tidak selalu yang paling tinggi.

3.1.15.4. Metrik Keandalan (Reliability Metrics)

Metrik keandalan memastikan bahwa peningkatan performa tidak dicapai dengan mengorbankan kebenaran atau tingkat keberhasilan.

A. Success Rate (Tingkat Keberhasilan)

Success Rate mengukur persentase perangkat yang berhasil dikonfigurasi tanpa kesalahan. Formula perhitungannya adalah:

$$Success\_Rate = \left(\frac{jumlah\ sukses\ konfigurasi}{total_{router\ CHR}}\right) x\ 100\% \qquad (29)$$

Target adalah 100 persen success rate, menunjukkan semua perangkat berhasil dikonfigurasi tanpa kegagalan. *Success rate* yang lebih rendah akan menunjukkan masalah keandalan yang perlu investigasi dan penyelesaian. Dalam hasil penelitian untuk CNCSA dan NSM, *success rate* yang dicapai adalah 100 persen dengan semua 15 perangkat berhasil dikonfigurasi tanpa *connection error* atau *configuration error*.

B. Error Rate (Tingkat Kesalahan)

Error Rate adalah komplemen dari *Success Rate*, mengukur persentase perangkat yang gagal konfigurasi. Formula perhitungannya adalah:

$$Error\ rate = \left(\frac{jumlah_{error}}{total_{routerCHR}}\right) x100\% \qquad (30)$$

3.1.16. Implementasi Kode Program

Penyajian aspek-aspek implementasi CNCSA dan NSM berikut ditujukan agar prosesnya mudah direplikasi dan transparan. Bagian ini juga menampilkan potongan kode penting sebagai bukti bahwa kedua metode benar-benar diimplementasikan sesuai prosedur.

3.1.16.1. Implementasi CNCSA

Implementasi algoritma utama CNCSA terletak dalam fungsi view Django yang menangani HTTP POST *request* untuk eksekusi konfigurasi. Potongan kode tersebut memperlihatkan ciri utama dari algoritma CNCSA sesuai implementasi aslinya. Pertama, struktur 5 fase terlihat jelas karena setiap tahap diberi komentar, mulai dari inisialisasi sampai proses menampilkan hasil. Kedua, pengukuran waktu dibuat lebih rinci dengan mencatat waktu secara terpisah untuk Tc, Ts, dan Tl menggunakan time.time sebelum dan sesudah setiap langkah. Ketiga, parameter yang dioptimalkan dituliskan langsung dalam *dictionary device_params*, lalu digunakan saat pemanggilan *send_command*. Timeout diubah menjadi 8 detik (lebih cepat dari default 100 detik) agar kegagalan koneksi bisa terdeteksi lebih awal. *Session_timeout* diatur menjadi 45 detik (lebih pendek dari default 60 detik) supaya pengelolaan sesi lebih cepat merespons. Nilai

keepalive diset 30 detik (yang sebelumnya tidak ada nilai bawaan) untuk menjaga koneksi tetap stabil.

Pada tahap pengiriman *command*, nilai *delay_factor* diatur menjadi 0.5, lebih kecil dari default 1.0, sehingga jeda antar perintah menjadi setengah dan ini yang paling berpengaruh pada peningkatan kecepatan. Parameter *max_loops* diubah menjadi 300 dari *default* 500 agar proses mendeteksi timeout bisa lebih cepat. Selain itu, strip_prompt dan *strip_command* diset ke *True*, bukan *False* seperti bawaan, supaya *output* yang diterima sudah otomatis dibersihkan. Keempat, pada fase 4 semua metrik dihitung sesuai rumus yang sudah ditentukan. Nilai Te dihitung langsung dengan rumus Te = Tc + Ts + Tl, lalu epsilon diperoleh dari selisih antara elapsed time dan Te. Setelah itu, semua rasio efisiensi dihitung menggunakan formula yang sudah disiapkan sebelumnya. Kelima, mekanisme *error handling* dibuat cukup kuat sehingga pengukuran waktu tetap tersimpan meskipun terjadi *error*. Dengan cara ini, seluruh komponen waktu tetap tercatat dengan benar dan hasil metrik tetap akurat.

Berikut adalah potongan kode penting yang menunjukkan pendekatan terstruktur dan pengukuran timing sesuai dengan algoritma CNCSA yang telah dirumuskan:

```
def configure(request):
    if request.method == "POST":
        # ================================================================
        # FASE 1: INISIALISASI SISTEM
        # ================================================================
        start_time = time.time()
        selected_device_ids = request.POST.getlist('device')
        results = []

        # ================================================================
        # FASE 2: INISIALISASI VARIABEL AKUMULASI SESUAI CNCSA
        # ================================================================
        total_tc = 0.0  # Total Connection Time
        total_ts = 0.0  # Total Send Time
        total_tl = 0.0  # Total Logging Time
        success_count = 0
        error_count = 0

        # ================================================================
        # FASE 3: EKSEKUSI SEQUENTIAL (i = 1 sampai n)
        # ================================================================
        for device_id in selected_device_ids:
            tc_i = 0.0
            ts_i = 0.0
            tl_i = 0.0
            net_connect = None

            try:
                # Ambil data device
                dev = get_object_or_404(Device, pk=device_id)
                decrypted_password =
decrypt_password(dev.encrypted_password)
                command_key = f"command_{dev.ip_address}"
                raw_command = request.POST.get(command_key, '').strip()
                command_list = raw_command.splitlines()

                 ]
```

```
                    clean = cmd_out.strip()
                    if clean and not clean.startswith("[admin@"):
                        output += f"Command: {cmd}\n{clean}\n\n"

                ts_i = time.time() - config_start
                total_ts += ts_i

                # Disconnect
                try:
                    net_connect.disconnect()
                except:
                    pass

                # ------------------------------------------------------
                # LANGKAH 3.3: LOGGING DENGAN TIMING (Tl_i)
                # OPTIMASI DATABASE LOGGING
                # ------------------------------------------------------
                log_start = time.time()

                Log.objects.create(
                    target=dev,
                    action="Configure",
                    status="Success",
                    time=timezone.now(),
                    message=output[:5000],  # Optimized: limit 5000 chars
                    duration=tc_i + ts_i
                )

                tl_i = time.time() - log_start
                total_tl += tl_i

                # Total waktu device i
                total_device_i = tc_i + ts_i + tl_i

                results.append({
                    "device": dev,
                    "status": "Success",
                    "message": "Konfigurasi berhasil dikirim.",
                    "tc_i": round(tc_i, 4),
                    "ts_i": round(ts_i, 4),
                    "tl_i": round(tl_i, 4),
                    "total_device_time": round(total_device_i, 4),
                })
                success_count += 1

            except Exception as e:
                # Error handling dengan preservasi timing
                tc_i = time.time() - conn_start if conn_start else 0
                ts_i = time.time() - config_start if config_start else 0

                if net_connect:
                    try:
                        net_connect.disconnect()
                    except:
                        pass

                log_start = time.time()
                Log.objects.create(
                    target=dev,
                    action="Configure",
                    status="Error",
                    time=timezone.now(),
                    message=f"Error: {str(e)[:1000]}",
                    duration=tc_i + ts_i
                )
                tl_i = time.time() - log_start

                total_tc += tc_i
                total_ts += ts_i
                total_tl += tl_i
```

```
                results.append({
                    "device": dev,
                    "status": "Error",
                    "message": f"Terjadi kesalahan: {e}",
                    "tc_i": round(tc_i, 4),
                    "ts_i": round(ts_i, 4),
                    "tl_i": round(tl_i, 4),
                    "total_device_time": round(tc_i + ts_i + tl_i, 4)
                })
                error_count += 1

        # ============================================================
        # Fase 4: Perhitungan Metrics Algoritma CNCSA
        # ============================================================

        # Total Execution Time (Te)
        te_cncsa = total_tc + total_ts + total_tl

        # Elapsed Time (wall clock)
        elapsed_time = time.time() - start_time

        # System Overhead (ε)
        overhead_epsilon = elapsed_time - te_cncsa

        # Jumlah devices (n)
        n = len(selected_device_ids)

        # Average time per device
        t_avg = te_cncsa / n if n > 0 else 0
        # Efficiency Ratios
        if te_cncsa > 0:
            e_conn = (total_tc / te_cncsa) * 100
            e_config = (total_ts / te_cncsa) * 100
            e_log = (total_tl / te_cncsa) * 100
            e_cncsa = ((total_ts + total_tl) / te_cncsa) * 100
        else:
            e_conn = e_config = e_log = e_cncsa = 0
        # Success Rate
        success_rate = (success_count / n * 100) if n > 0 else 0

        # ============================================================
        # FASE 5: Render Hasil
        # ============================================================
        context = {
            'devices': Device.objects.all(),
            'results': results,
            'elapsed_time': round(elapsed_time, 4),

            # Component Times
            'total_connection_time': round(total_tc, 4),
            'total_send_time': round(total_ts, 4),
            'total_logging_time': round(total_tl, 4),

            # CNCSA Metrics
            'te_cncsa': round(te_cncsa, 4),
            'overhead_epsilon': round(overhead_epsilon, 4),
            't_avg': round(t_avg, 4),

            # Statistics
            'device_count': n,
            'success_count': success_count,
            'success_rate': round(success_rate, 2),

            # Efficiency Ratios
            'e_conn': round(e_conn, 2),
            'e_config': round(e_config, 2),
            'e_log': round(e_log, 2),
            'e_cncsa': round(e_cncsa, 2),
        }
        return render(request, 'config.html', context)
```

3.1.16.2. Implementasi NSM

Implementasi NSM sebagai baseline menggunakan parameter default tanpa optimasi dan pengukuran yang disederhanakan (*default*). Implementasi NSM sebagai baseline menggunakan parameter default Netmiko tanpa optimasi dan pengukuran yang disederhanakan. NSM dirancang sebagai reference point untuk mengukur efektivitas optimasi yang dilakukan dalam CNCSA:

```
def configure(request):
    if request.method != "POST":
        return render(request, 'config.html', {
            'devices': Device.objects.all(),
            'mode': 'Configure'
        })

    #Fase Inisialisasi
    start_time = time.time()
    selected_device_ids = request.POST.getlist('device')
    results = []

    # FASE 2: INISIALISASI AKUMULASI WAKTU
    total_tc = 0.0  # Total Connection Time
    total_ts = 0.0  # Total Send Time
    success_count = 0
    error_count = 0

    # FASE 3: SEQUENTIAL EXECUTION - STANDARD (NON-OPTIMIZED)

    for device_id in selected_device_ids:
        conn_start = None
        config_start = None
        net_connect = None
        tc_i = 0.0
        ts_i = 0.0
```

```
        try:
            # Ambil data device
            dev = get_object_or_404(Device, pk=device_id)
            decrypted_password =
decrypt_password(dev.encrypted_password)
            command_key = f"command_{dev.ip_address}"
            raw_command = request.POST.get(command_key, '').strip()
            command_list = raw_command.splitlines()

            device_params = {
                "device_type": "mikrotik_routeros",
                "host": dev.ip_address,
                "username": dev.username,
                "password": decrypted_password,
            }

            # LANGKAH 3.1: KONEKSI SSH DENGAN TIMING (Tc_I)
            # PARAMETER DEFAULT

            conn_start = time.time()
            net_connect = ConnectHandler(**device_params)
            tc_i = time.time() - conn_start
            total_tc += tc_i

            # ------------------------------------------------------
            # LANGKAH 3.2: SEND COMMANDS DENGAN TIMING (Ts_i)
            # PARAMETER DEFAULT - TIDAK ADA OPTIMASI
            # ------------------------------------------------------
            config_start = time.time()
            output = ""

            # STANDARD: Command satu per satu dengan parameter DEFAULT
            for cmd in command_list:
                cmd_out = net_connect.send_command(
                    cmd,
                    expect_string=r'.*[>$#]',

                )
                # Simple cleaning tanpa optimasi
                clean = "\n".join(
                    line for line in cmd_out.splitlines()
                    if not line.startswith("[admin@")
                )
                output += f"Command: {cmd}\n{clean}\n\n"
```

```
[
        try:
            # Ambil data device
            dev = get_object_or_404(Device, pk=device_id)
            decrypted_password =
decrypt_password(dev.encrypted_password)
            command_key = f"command_{dev.ip_address}"
            raw_command = request.POST.get(command_key, '').strip()
            command_list = raw_command.splitlines()

            device_params = {
                "device_type": "mikrotik_routeros",
                "host": dev.ip_address,
                "username": dev.username,
                "password": decrypted_password,
            }

            # LANGKAH 3.1: KONEKSI SSH DENGAN TIMING (Tc_I)
            # PARAMETER DEFAULT

            conn_start = time.time()
            net_connect = ConnectHandler(**device_params)
            tc_i = time.time() - conn_start
            total_tc += tc_i

            # ------------------------------------------------------
            # LANGKAH 3.2: SEND COMMANDS DENGAN TIMING (Ts_i)
            # PARAMETER DEFAULT - TIDAK ADA OPTIMASI
            # ------------------------------------------------------
            config_start = time.time()
            output = ""

            # STANDARD: Command satu per satu dengan parameter DEFAULT
            for cmd in command_list:
                cmd_out = net_connect.send_command(
                    cmd,
                    expect_string=r'.*[>$#]',

                )
                # Simple cleaning tanpa optimasi
                clean = "\n".join(
                    line for line in cmd_out.splitlines()
                    if not line.startswith("[admin@")
                )
                output += f"Command: {cmd}\n{clean}\n\n"
```

```
            ts_i = time.time() - config_start
            total_ts += ts_i

            # Disconnect
            try:
                net_connect.disconnect()
            except:
                pass

            # ------------------------------------------------------
            # LOGGING LANGSUNG TANPA TIMING MEASUREMENT
            # ------------------------------------------------------

            ActivityLog.objects.create(
                target=dev,
                action="Configure",
                status="Success",
                time=timezone.now(),
                message=output[:5000],
                duration=tc_i + ts_i
            )

            # Total waktu device i (tanpa Tl)
            total_device_i = tc_i + ts_i

            results.append({
                "device": dev,
                "status": "Success",
                "message": "Konfigurasi berhasil dikirim.",
                "tc_i": round(tc_i, 4),
                "ts_i": round(ts_i, 4),
                "total_device_time": round(total_device_i, 4),
            })
            success_count += 1

        except Exception as e:
            # Error handling
            tc_i = time.time() - conn_start if conn_start else 0
            ts_i = time.time() - config_start if config_start else 0

            if net_connect:
                try:
                    net_connect.disconnect()
                except:
                    pass

            # Logging error tanpa timing
            ActivityLog.objects.create(
                target=dev,
                action="Configure",
                status="Error",
                time=timezone.now(),
                message=f"Error: {str(e)[:1000]}",
                duration=tc_i + ts_i
            )

            total_tc += tc_i
            total_ts += ts_i

            results.append({
                "device": dev,
                "status": "Error",
                "message": f"Terjadi kesalahan: {e}",
                "tc_i": round(tc_i, 4),
                "ts_i": round(ts_i, 4),
                "total_device_time": round(tc_i + ts_i, 4)
            })
            error_count += 1
```

Kode NSM menunjukkan pendekatan baseline tanpa optimasi parameter atau pengukuran komprehensif. Perbedaan kunci dari CNCSA sangat jelas terlihat dalam beberapa aspek implementasi.

Pertama, parameter default digunakan secara implisit dengan tidak ada spesifikasi eksplisit untuk *timeout*, *session_timeout*, *keepalive*, *delay_factor*, *max_loops*, *strip_prompt*, atau *strip_command*. Semua parameter ini menggunakan nilai *default* dari *library* Netmiko yang dirancang konservatif untuk kompatibilitas maksimal tetapi tidak optimal untuk performa. *Timeout default* adalah 100 detik dibandingkan 8 detik di CNCSA, *session_timeout default* 60 detik dibandingkan 45 detik, dan tidak ada keepalive mechanism yang berarti koneksi bisa *timeout* jika idle terlalu lama. Kedua, tidak ada pengukuran Tl terpisah dimana logging dilakukan *inline* tanpa instrumentasi *timing*. Operasi *ActivityLog.objects.create* dipanggil langsung setelah konfigurasi selesai tanpa membungkusnya dengan timer, sehingga waktu *logging* termasuk dalam *elapsed time* tetapi tidak terlihat sebagai komponen tersendiri. Hal ini membuat NSM tidak bisa memberikan visibility detail tentang berapa banyak waktu yang dihabiskan untuk operasi database. Ketiga, nilai Te tidak dihitung secara langsung menggunakan *elapsed time* karena tidak memungkinkan menghitungnya dari penjumlahan komponen tanpa adanya pengukuran Tl. *Elapsed time* ini sudah mencakup Tc, Ts, Tl yang tidak terukur, serta *system overhead*. Pendekatan ini berbeda dengan CNCSA, yang bisa menghitung Te secara eksplisit dengan rumus Te = Tc + Ts + Tl karena semua komponennya terukur. Keempat, tidak ada perhitungan metrik efisiensi karena output yang dihasilkan hanya berupa waktu dasar tanpa rasio efisiensi atau hitungan overhead. Metrik seperti E_conn, E_config, E_log, E_CNCSA, dan SOR tidak bisa dihitung karena NSM tidak memiliki pengukuran waktu yang rinci. Akibatnya, NSM menjadi lebih sederhana, tetapi informasi yang didapat untuk melihat sumber bottleneck jadi jauh lebih sedikit. Kelima, susunan kodenya lebih spontan dan tidak dibagi ke dalam fase yang jelas seperti pada CNCSA. Walaupun alurnya tetap berurutan dari inisialisasi sampai hasil akhir, tidak ada komentar yang menunjukkan batas antar fase. Hal ini menggambarkan cara kerja umumnya pengguna yang memakai *library* standar tanpa pola sistematis untuk optimasi atau pengukuran yang mendalam.

3.1.17. Prosedur Pengujian Sistematis

Prosedur pengujian dirancang secara sistematis dan terstruktur untuk memastikan hasil eksperimen valid, dapat dipercaya, dan dapat diulang oleh peneliti lain. Setiap langkah didokumentasikan dengan jelas agar tidak ada ambiguitas dalam pelaksanaan eksperimen.

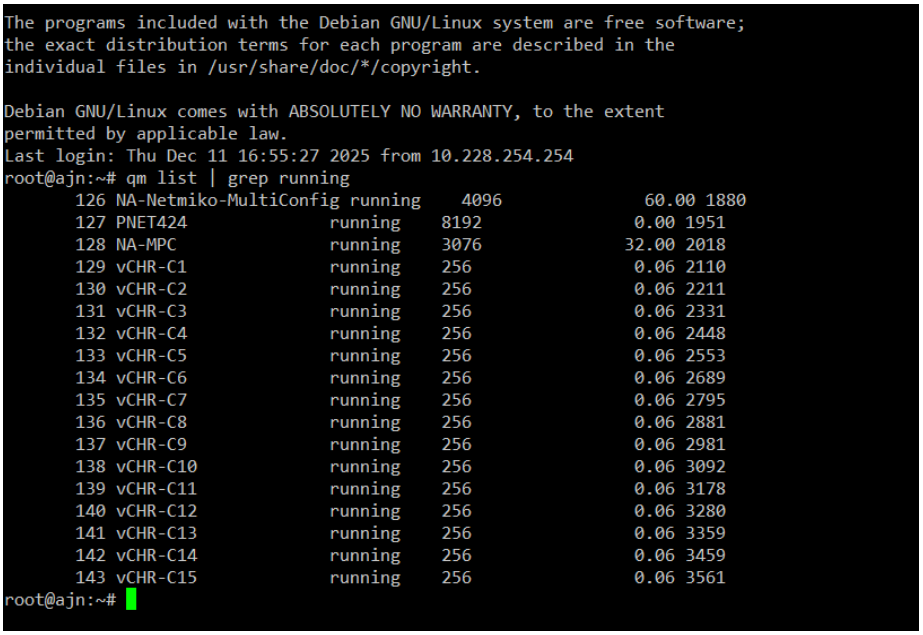
3.1.17.1. Tahap Persiapan Sebelum Pengujian

Sebelum memulai setiap sesi pengujian, dilakukan serangkaian persiapan untuk memastikan lingkungan dalam kondisi optimal dan konsisten.

A. Verifikasi Ketersediaan Perangkat

Langkah pertama adalah memastikan semua 15 router CHR dalam kondisi aktif dan berjalan normal. Verifikasi dilakukan dengan

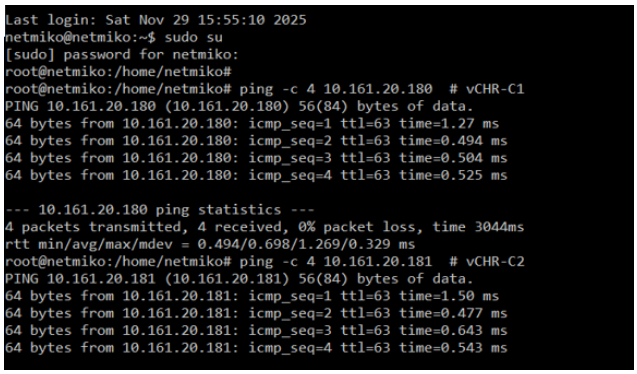
mengakses antarmuka web Proxmox VE untuk melihat status setiap virtual machine. Semua CHR dari vCHR-C1 hingga vCHR-C15 harus dalam status *running* tanpa ada yang dalam mode *maintenance* atau mengalami masalah, seperti terlihat pada gambar 3.7. Jika ada CHR yang tidak running, maka CHR tersebut harus di-start terlebih dahulu dan diberi waktu beberapa menit untuk *boot up* sempurna sebelum melanjutkan ke tahap berikutnya. Verifikasi ini penting untuk menghindari kegagalan pengujian yang disebabkan oleh perangkat yang tidak siap menerima koneksi SSH.



Gambar 3.7. Status Running vCHR1-vCHR15

B. Pengujian Konektivitas Jaringan

Setelah memastikan semua CHR aktif, langkah selanjutnya adalah menguji konektivitas jaringan dari kedua server CNCSA dan NSM ke semua CHR. Sample pengujian dilakukan menggunakan perintah ping dari terminal server dengan sintaks seperti gambar 3.8.

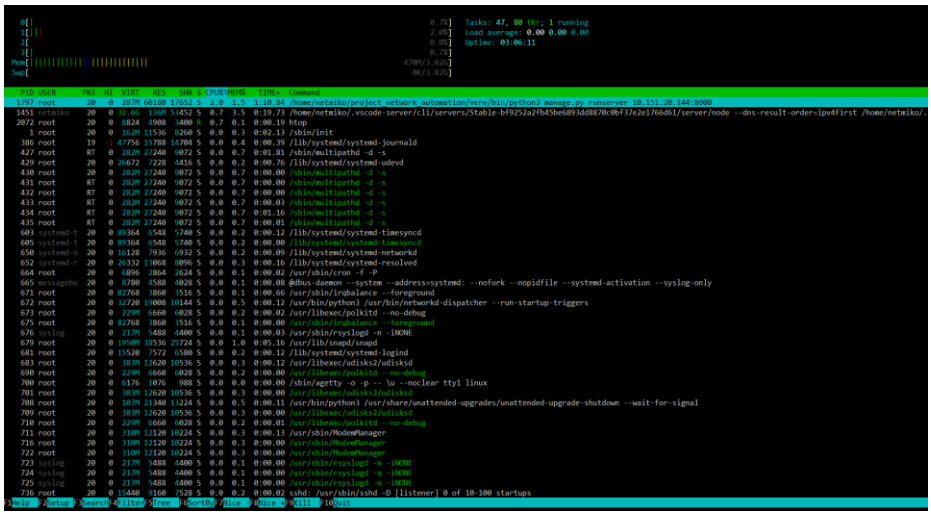


Gambar 3.8. Pengujian Koneksi Jaringan dari Server CNCSA

Hasil ping yang untuk *round-trip time* dalam range 0.5-1 milidetik dengan nol *packet loss*. Jika ada CHR yang tidak dapat di kirimkan *packet icmp* atau menunjukkan latensi abnormal (lebih dari 10 ms) atau *packet loss*, maka perlu dilakukan investigasi dan *troubleshooting* sebelum melanjutkan pengujian. Latensi yang konsisten dan rendah penting untuk memastikan bahwa perbedaan waktu eksekusi yang terukur nanti berasal dari perbedaan algoritma dan parameter, bukan dari variabilitas kondisi jaringan.

C. Pengecekan Resource Server

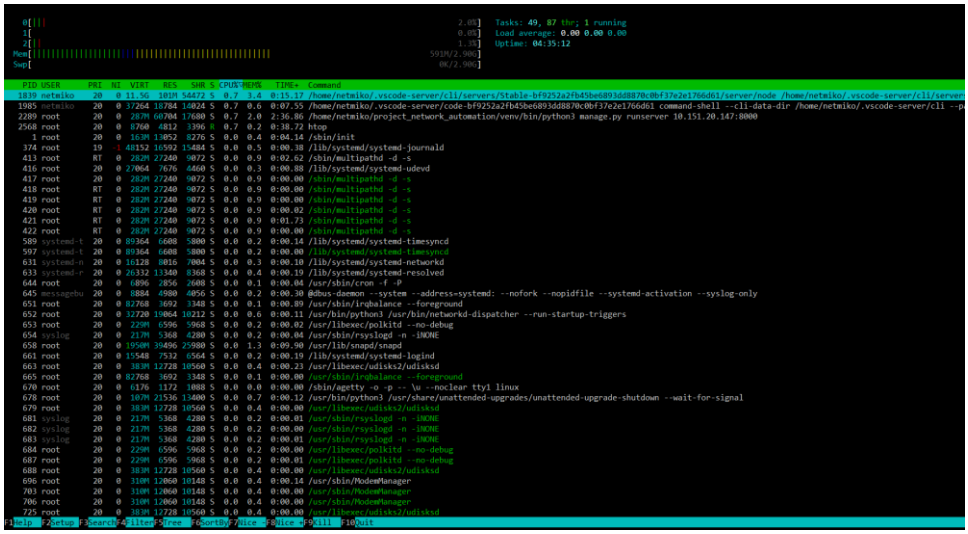
vServer CNCSA dan NSM perlu diperiksa untuk memastikan tidak ada aplikasi atau proses lain yang menggunakan *resource* CPU atau *memory* secara signifikan. Pengecekan dilakukan menggunakan perintah top atau htop di terminal untuk melihat utilisasi CPU dan memory. Utilisasi CPU yang ideal adalah di bawah 20 persen saat idle, dan memory *available* minimal 1GB bebas. Jika ada proses yang mengkonsumsi resource besar, proses tersebut perlu dihentikan terlebih dahulu atau pengujian dijadwalkan ulang ke waktu dimana beban sistem lebih rendah. Pada gambar 3.9 berdasarkan tampilan htop pada server CNCSA, kondisi *resource* terlihat normal dan sudah memenuhi syarat untuk pengujian CNCSA maupun NSM.



Gambar 3.9. Kondisi running server CNCSA

Pada bagian CPU terlihat hanya terpakai sebagian kecil saja (di bawah 5%). Hasil ini sesuai dengan standar pengecekan bahwa CPU idealnya < 20% saat idle. Artinya vServer tidak sedang menjalankan proses berat lain yang bisa mengganggu hasil pengujian. Pada bagian memory terlihat hanya sedikit RAM yang terpakai. Dengan total RAM vServer, masih tersedia space yang cukup besar (lebih dari 3 GB free). Hasil ini sesuai ketentuan bahwa minimal harus ada 1GB memory yang bebas sebelum menjalankan pengujian. Dari daftar proses di htop, yang aktif hanya proses sistem standar seperti *systemd*, *journald*, *sshd, unattended-upgrade*, dan Python script

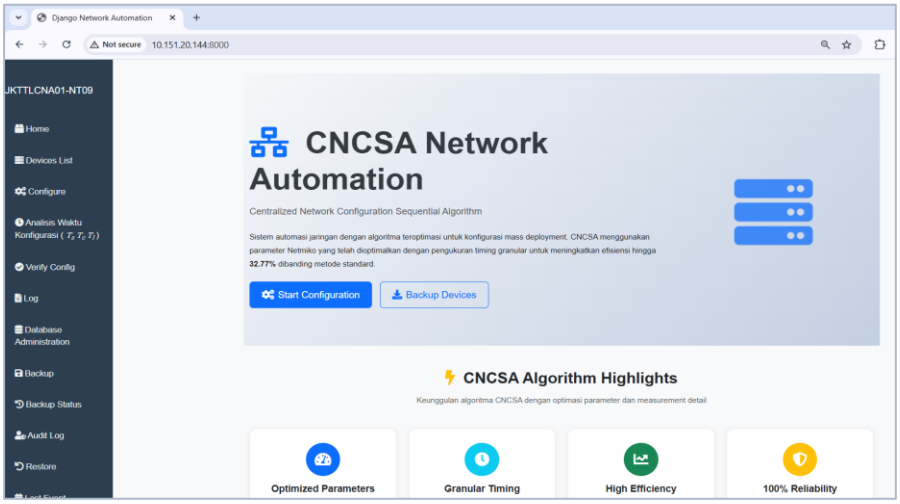
yang sedang berjalan. Tidak ada proses besar seperti database, web server berat, atau aplikasi lain yang mengonsumsi CPU/memory secara abnormal. Gambar 3.10 untuk vServer NSM Grafik CPU di bagian atas menunjukkan hanya sekitar 0.7% yang digunakan. Nilai ini masih di bawah batas maksimum saat *idle* (20%), sehingga vServer dapat menjalankan skenario pengujian tanpa terganggu beban lain. Pemakaian RAM hanya sekitar 593 MB dari total 2.9 GB. Artinya masih ada lebih dari 2 GB ruang bebas, jauh di atas kebutuhan minimal 1 GB yang disarankan agar pengujian tidak terpengaruh proses lain. Dari daftar proses, hanya ada proses umum seperti *systemd*, *journald*, *cron*, *networkd*, dan beberapa *service* kecil lainnya.



Gambar 3.10 Kondisi *running* server NSM

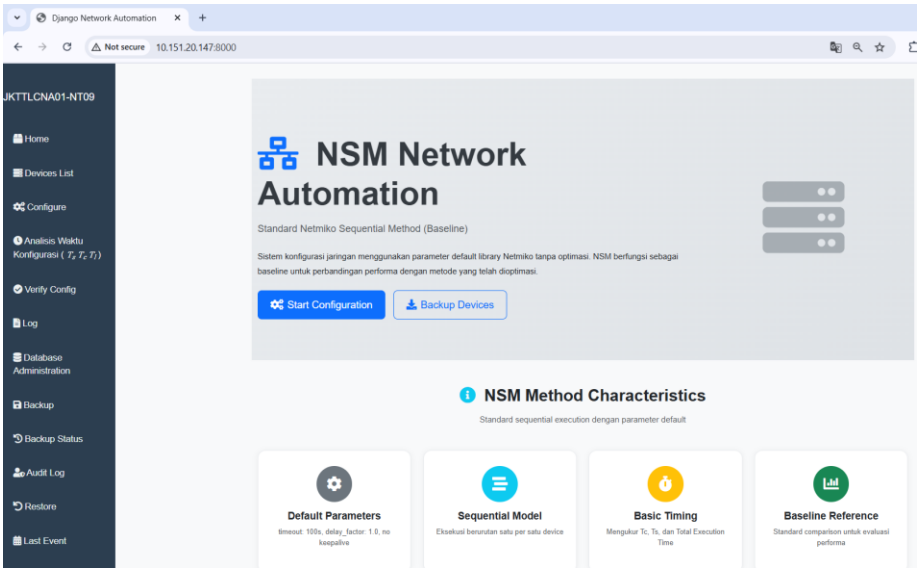
D. Verifikasi Status Layanan Django Server CNCSA dan NSM

Kedua vServer CNCSA dan NSM menjalankan Django *development* server yang perlu dipastikan dalam kondisi aktif. Verifikasi dilakukan dengan mengakses URL masing-masing server melalui browser. Gambar 3.11 merupakan tampilan untuk server CNCSA



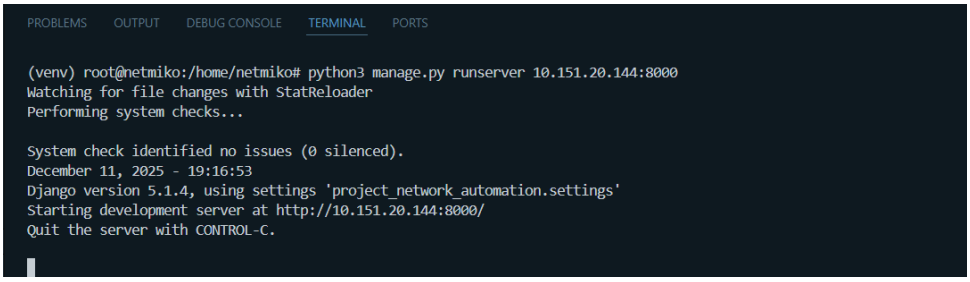
Gambar 3.11. Tampilan server CNCSA (10.151.20.144:8000)

Gambar 3.12 untuk server *standard* Netmiko (NSM), proses verifikasi dilakukan dengan cara yang sama, yaitu mengakses alamat django server 10.151.20.147:8000 melalui browser. Tampilan halaman utama NSM muncul tanpa kendala, menandakan bahwa layanan berada dalam kondisi aktif dan siap digunakan sebagai *baseline* pembanding. Menu dan fitur yang tersedia mengikuti struktur dasar aplikasi tanpa optimasi tambahan, sesuai dengan peran NSM sebagai metode standard dalam penelitian ini. Dengan munculnya kedua halaman tersebut, dapat dipastikan bahwa kedua lingkungan CNCSA dan NSM siap digunakan untuk seluruh rangkaian pengujian berikutnya.



Gambar 3.12. Tampilan server CNCSA (10.151.20.144:8000)

Akses ke antarmuka CNCSA dan NSM dilakukan dengan menjalankan django development server pada masing-masing vServer. Proses ini dilakukan melalui terminal pada server CNCSA sebagaimana terlihat pada Gambar 3.13. Untuk mengaktifkan layanan CNCSA digunakan perintah python3 manage.py runserver 10.151.20.144:8000, sedangkan untuk mengaktifkan layanan NSM digunakan perintah python3 `manage.py` runserver 10.151.20.147:8000. Setelah kedua perintah dijalankan, masing-masing aplikasi dapat diakses melalui browser sesuai alamat yang sudah ditentukan.



Gambar 3.13. Verifikasi server CNCSA

E. Reset Perangkat ke Kondisi Bersih (*Clean State*)

Pada tahap ini seluruh router CHR disiapkan ulang agar kembali ke kondisi awal sebelum dilakukan pengujian. Proses reset dilakukan untuk memastikan tidak ada konfigurasi lama khususnya *rule* atau *policy firewall filter*, NAT dan *Mangle* yang tertinggal dari pengujian sebelumnya. Dengan kondisi awal yang sama pada setiap router CHR, seluruh pengujian dapat dijalankan dari *baseline* yang selaras.

1) Login ke masing-masing router CHR (vCHR1-vCHR15) dilakukan melalui SSH dari server pengujian (CNCSA dan NSM). Akses dilakukan menggunakan perintah yang dijalankan langsung dari terminal server.
2) Langkah pertama adalah menghapus seluruh aturan pada kategori *Mangle*. Proses ini dilakukan dengan menjalankan perintah `/ip firewall mangle remove [find]` pada masing-masing router CHR. Perintah tersebut akan menghapus semua rule Mangle yang ada tanpa meminta konfirmasi tambahan. Apabila tidak terdapat rule pada kategori ini, sistem tidak menampilkan keluaran apa pun, yang menandakan bahwa Mangle memang sudah dalam kondisi kosong seperti gambar 3.14.

```
[admin@vCHR1] > /ip firewall mangle remove [find]
[admin@vCHR1] >
```

Gambar 3.14. Proses reset firewall rules mangle (vCHR-C1)

3) Setelah Mangle dibersihkan, tahap berikutnya adalah menghapus seluruh aturan pada kategori NAT seperti pada gambar 3.15. Penghapusan dilakukan menggunakan perintah `/ip firewall nat remove [find]`. Dengan perintah ini, semua rule NAT yang tersimpan di router akan dihapus sehingga tidak ada konfigurasi translasi alamat yang tersisa dari pengujian sebelumnya.

```
[admin@vCHR1] > /ip firewall nat remove [find]
[admin@vCHR1] >
```

Gambar 3.15. Proses reset firewall rules NAT di salah satu CHR (vCHR-C1)

4) Langkah selanjutnya adalah membersihkan aturan pada kategori Filter dengan pengecualian untuk rule yang bersifat dynamic seperti divisualisasikan seperti gambar 3.16. `Perintah /ip firewall filter remove [find where !dynamic]` digunakan untuk menghapus seluruh *rule filter* yang dibuat secara manual, sementara *rule dynamic* yang dibuat otomatis oleh sistem tetap dipertahankan. *Rule dynamic* tidak dihapus karena diperlukan agar router tetap dapat beroperasi secara normal.

```
[admin@vCHR1] > /ip firewall filter remove [find where !dynamic]
[admin@vCHR1] >
```

Gambar 3.16. Proses reset firewall rules NAT di salah satu CHR (vCHR-C1)

Setelah seluruh perintah penghapusan dijalankan, dilakukan proses verifikasi untuk memastikan bahwa reset telah berhasil. Verifikasi dilakukan dengan menjalankan perintah *print count-only* pada masing-masing kategori *firewall*, yaitu *Mangle*, NAT, dan *Filter* seperti gambar 3.17.

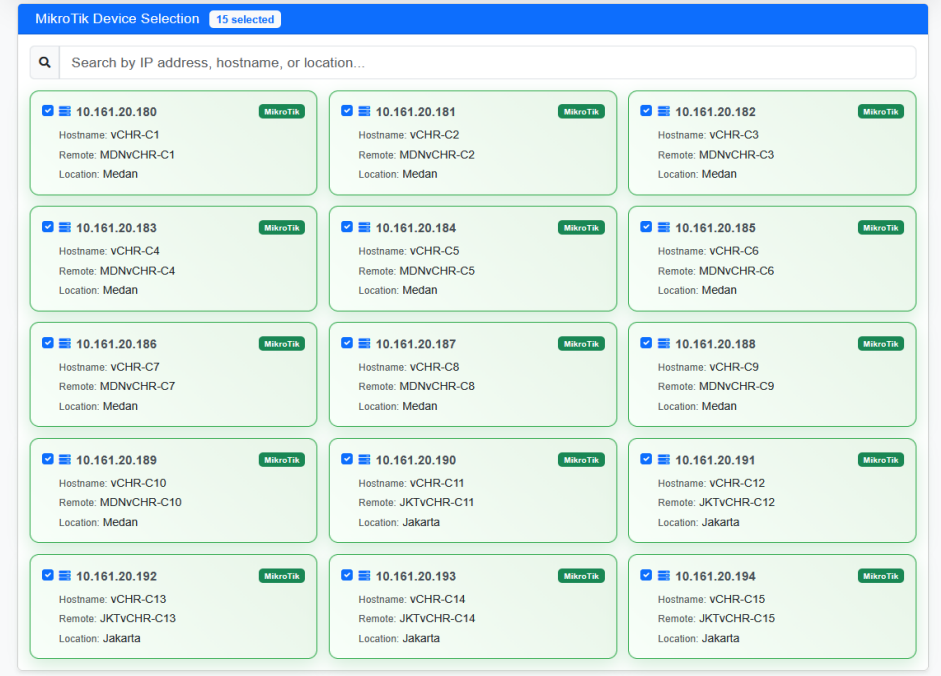
```
[admin@vCHR1] > /ip firewall mangle print count-only
0
[admin@vCHR1] > /ip firewall nat print count-only
0
[admin@vCHR1] > /ip firewall filter print count-only
0
```

Gambar 3.17. Validasi kategori firewall rules

Hasil yang diharapkan adalah jumlah rule bernilai nol untuk Mangle dan NAT, serta tidak adanya *rule non-dynamic* pada *Filter*. Jika kondisi tersebut terpenuhi, maka perangkat telah berhasil dikembalikan ke kondisi bersih. Prosedur reset yang sama diterapkan secara berulang pada seluruh router virtual yang digunakan dalam penelitian, mulai dari vCHR-C1 hingga vCHR-C15. Meskipun proses ini dapat diotomatisasi menggunakan *script*, pada penelitian ini reset dilakukan secara manual agar setiap langkah dapat didokumentasikan dengan jelas dan mudah diverifikasi. Pengembalian perangkat ke kondisi bersih merupakan tahapan penting karena memastikan seluruh pengujian dimulai dari kondisi awal yang sama. Hal ini mencegah pengaruh konfigurasi lama terhadap hasil pengukuran, menghindari konflik antara aturan lama dan baru, serta mendukung keterulangan eksperimen. Dengan kondisi awal yang konsisten, hasil penelitian menjadi lebih akurat dan dapat diuji ulang oleh peneliti lain.

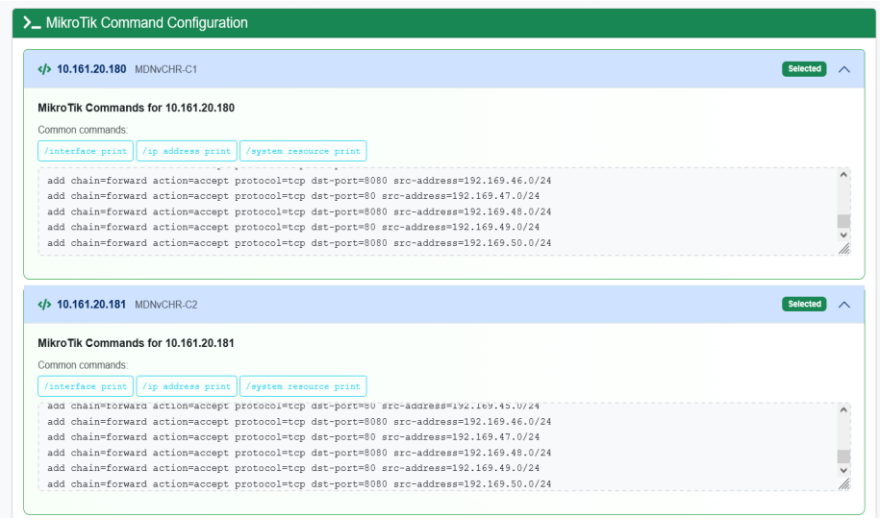
3.1.17.2. Prosedur Eksekusi Pengujian CNCSA

Setelah seluruh tahapan persiapan selesai dan semua perangkat berada dalam kondisi *clean state*, pengujian CNCSA dilakukan melalui antarmuka web yang disediakan oleh server Django CNCSA. penulis membuka web browser Google Chrome atau Mozilla Firefox, kemudian mengakses URL http://10.151.20.144:8000/configure/ seperti pada gambar 3.18. Halaman *configure* akan ditampilkan dan berisi form yang memuat daftar perangkat router serta area untuk memasukkan perintah konfigurasi. Seluruh perangkat dari vCHR-C1 hingga vCHR-C15 dipilih sebagai target pengujian dengan mencentang *checkbox* yang tersedia. Setelah semua perangkat terpilih, sistem menampilkan indikator jumlah perangkat yang dipilih sebagai konfirmasi sebelum proses konfigurasi dijalankan.



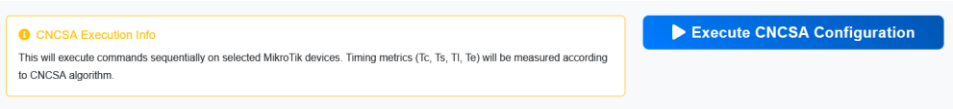
Gambar 3.18. Pemilihan 15 Devices pada Antarmuka CNCSA

Setelah seluruh router CHR dipilih, penulis menginput skrip baris perintah konfigurasi sebanyak 150 baris *command* ke dalam textarea *commands*. Command tersebut sebelumnya telah disiapkan dalam file teks untuk memastikan konsistensi isi. Pada tahap ini perlu dipastikan bahwa seluruh 150 baris berhasil ter-input tanpa ada baris yang terpotong, tidak terdapat baris kosong di bagian awal maupun akhir, serta format setiap command tetap sama seperti pada file sumber. Verifikasi cepat dilakukan dengan melakukan *scroll* pada textarea untuk memastikan rangkaian command dari *Mangle*, NAT, hingga *Filter* terlihat lengkap seperti pada gambar 3.19.



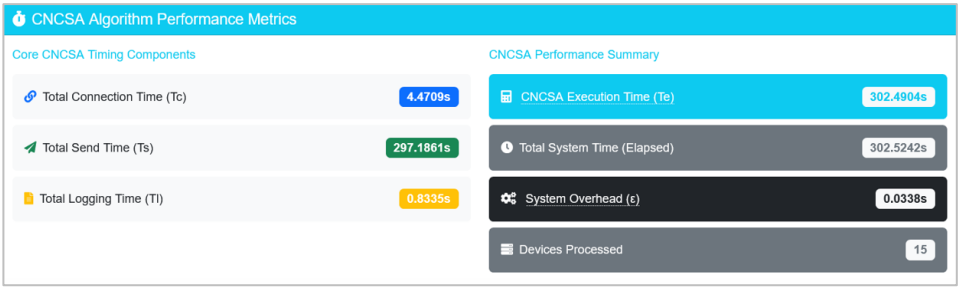
Gambar 3.19. Input 150 Baris Command pada Textarea CNCSA

Proses konfigurasi dimulai dengan menekan tombol *Execute Configuration* seperti pada gambar 3.20. Setelah tombol ditekan, penulis tidak melakukan interaksi apa pun pada browser, seperti menutup tab, melakukan refresh, atau berpindah halaman. Seluruh proses dibiarkan berjalan hingga selesai. Selama eksekusi, sistem dapat menampilkan indikator proses atau animasi pemuatan sebagai tanda bahwa konfigurasi sedang berlangsung.



Gambar 3.20. Proses konfigurasi (execute configuration)

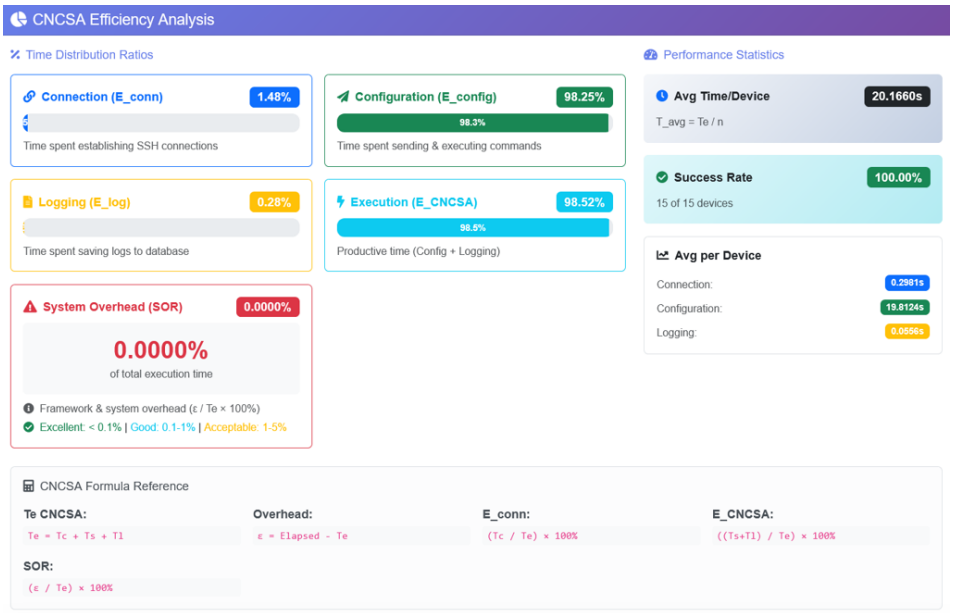
Setelah seluruh konfigurasi selesai diproses, halaman akan memperbarui tampilan secara otomatis dan menampilkan hasil pengukuran berupa metrics waktu dan tabel detail per perangkat. Berdasarkan hasil pengujian kinerja algoritma CNCSA yang ditampilkan pada Gambar 3.21, sistem berhasil memproses konfigurasi terhadap 15 perangkat router CHR dengan pencatatan waktu yang terukur secara rinci. Total *connection time* (Tc) tercatat sebesar 4.4709 detik, yang menunjukkan waktu kumulatif yang dibutuhkan untuk membangun koneksi SSH ke seluruh perangkat. Selanjutnya, *send time* (Ts) sebagai komponen utama konfigurasi memiliki nilai 297.1861 detik, yang merepresentasikan waktu untuk pengiriman dan eksekusi command konfigurasi pada perangkat. Proses pencatatan hasil ke database (*logging time*, Tl) memerlukan waktu 0.8335 detik, menunjukkan bahwa aktivitas logging berjalan sangat cepat dan tidak memberikan beban signifikan terhadap proses keseluruhan. Akumulasi dari ketiga komponen tersebut menghasilkan *CNCSA Execution Time* (Te) sebesar 302.4904 detik, yang sangat mendekati *total system time (elapsed)* yaitu 302.5242 detik. Selisih antara keduanya menghasilkan *system overhead* (ε) sebesar 0.0338 detik, yang menunjukkan bahwa overhead sistem berada pada tingkat yang sangat kecil.



Gambar 3.21 Hasil pengukuran berupa metrics waktu CNCSA

Analisis distribusi efisiensi waktu menunjukkan bahwa *connection efficiency* (E_conn) berada pada nilai 1.48%, menandakan bahwa waktu

yang digunakan untuk membangun koneksi relatif kecil dan tidak menjadi hambatan utama. *Configuration efficiency* (E_config) mencapai 98.25%, yang berarti hampir seluruh waktu eksekusi digunakan untuk aktivitas inti berupa pengiriman dan eksekusi command konfigurasi. *Logging efficiency* (E_log) hanya sebesar 0.28%, mengindikasikan bahwa proses penyimpanan log ke database berjalan sangat efisien. Jika digabungkan, *CNCSA execution efficiency* (E_CNCSA) yang mencakup konfigurasi dan logging mencapai 98.52%, mencerminkan bahwa mayoritas waktu eksekusi benar-benar digunakan untuk pekerjaan yang bernilai. Nilai *System Overhead Ratio* (SOR) tercatat 0.0000%, yang mengonfirmasi bahwa overhead dari framework, interpreter, maupun sistem operasi dapat diabaikan seperti pada gambar 3.22.



Gambar 2.22. Efisiensi Analisis CNCSA

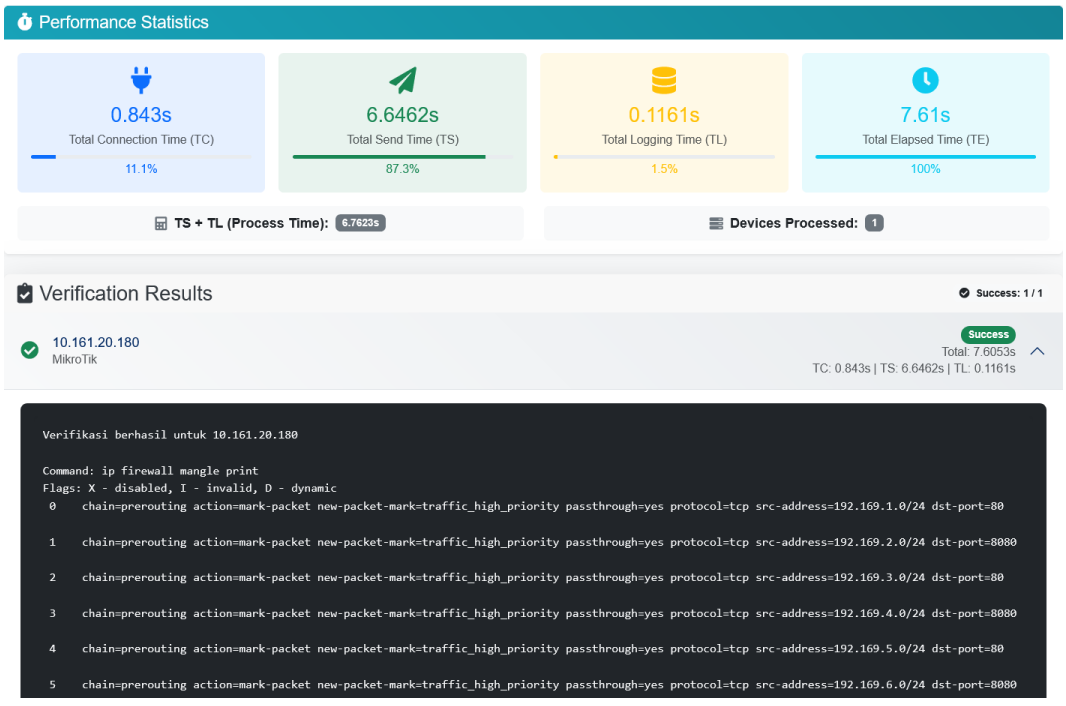
Dari sisi statistik tambahan, rata-rata waktu konfigurasi per perangkat (*average time per device*) adalah 20.1660 detik, dengan rincian rata-rata *connection time* sebesar 0.2981 detik, *configuration time* sebesar 19.8124 detik, dan *logging time* sebesar 0.0565 detik per perangkat. Seluruh perangkat berhasil dikonfigurasi tanpa kesalahan, sehingga *success rate* mencapai 100% atau 15 dari 15 perangkat berhasil diproses. Hasil ini menunjukkan bahwa algoritma CNCSA tidak hanya efisien dari sisi waktu, tetapi juga stabil dan andal dalam menjalankan konfigurasi massal secara sequential dengan *overhead* sistem yang sangat minimal.

Setelah proses eksekusi konfigurasi menggunakan algoritma CNCSA dinyatakan selesai dan metrik kinerja berhasil dicatat, tahap akhir yang dilakukan adalah verifikasi penerapan konfigurasi pada perangkat. Verifikasi ini bertujuan untuk memastikan bahwa seluruh *firewall rules*

benar-benar terimplementasi pada perangkat MikroTik CHR sesuai dengan jumlah dan struktur yang direncanakan, bukan hanya tercatat sebagai proses berhasil di sisi aplikasi. Verifikasi pertama dilakukan dengan metode *spot check* melalui akses SSH ke beberapa perangkat sampel, yaitu vCHR-C1 seperti pada gambar 3.23. Pemilihan sampel ini mewakili perangkat pada urutan awal dan tengah, sehingga dapat memberikan gambaran konsistensi hasil konfigurasi. Koneksi dilakukan menggunakan perintah SSH ke alamat IP masing-masing CHR. Setelah berhasil login, dilakukan pengecekan jumlah rule firewall pada setiap kategori menggunakan perintah *count-only*.

```
[admin@vCHR1] > /ip firewall mangle print count-only
50
[admin@vCHR1] > /ip firewall nat print count-only
100
[admin@vCHR1] > /ip firewall filter print count-only
50
[admin@vCHR1] >
```

Gambar 3.23. Verifikasi Firewall Rules Berhasil Diterapkan (50 Rules per Kategori)



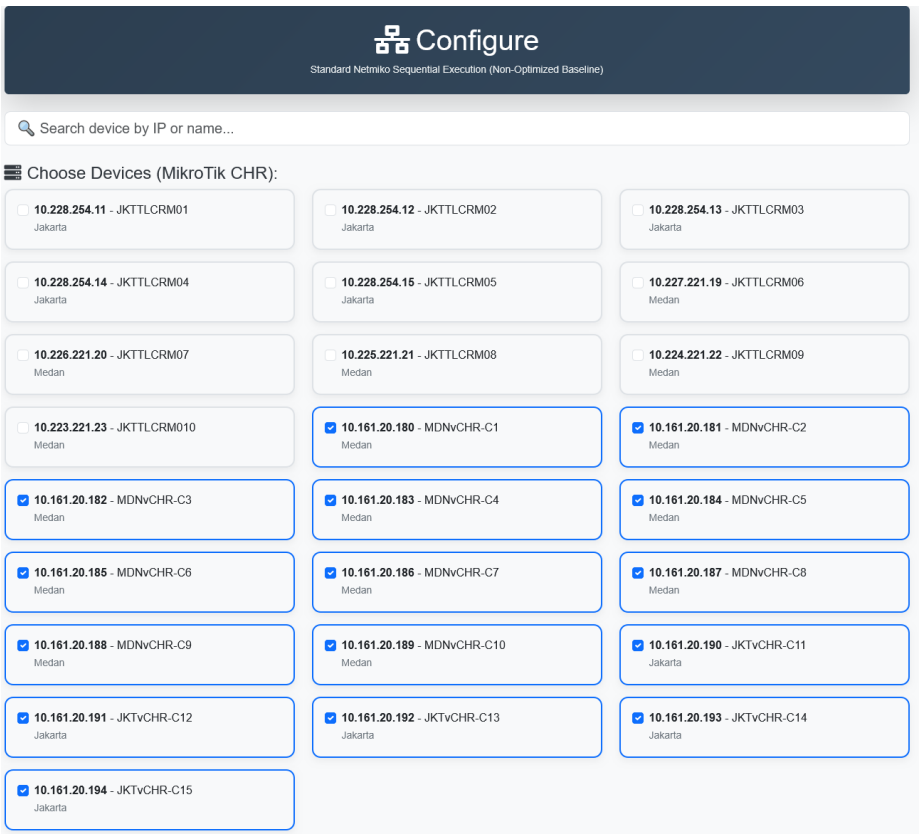
Gambar 3.24. Verifikasi Firewall Rules (http://10.151.20.144:8000/verify_config)

Selain verifikasi manual melalui SSH, sistem CNCSA juga menyediakan mekanisme verifikasi terpusat berbasis web yang dapat diakses melalui halaman verify_configuration. Pada halaman ini ditampilkan daftar seluruh perangkat yang terlibat dalam pengujian, mulai dari vCHR-C1 hingga hingga vCHR-C15. Proses verifikasi dilakukan dengan mencentang masing-masing perangkat untuk memastikan status konfigurasi telah terkonfirmasi oleh sistem, sample yang dilakukan pada

vCHR1 seperti pada gambar 3.24. Pendekatan ini memungkinkan peneliti untuk memvalidasi seluruh 15 perangkat secara terpusat tanpa harus melakukan login SSH satu per satu, sehingga meningkatkan efisiensi proses pengecekan pasca-konfigurasi. Hasil eksekusi perintah ini akan menampilkan seluruh rule firewall yang aktif pada perangkat tersebut. Dengan mencocokkan struktur dan jumlah rule yang ditampilkan dengan konfigurasi yang direncanakan, dapat dipastikan bahwa rule tidak hanya terhitung secara kuantitas, tetapi juga benar secara isi dan urutan. Fitur ini memperkuat proses validasi karena memberikan visibilitas penuh terhadap konfigurasi aktual di perangkat.

3.1.17.3. Prosedur Eksekusi Pengujian NSM

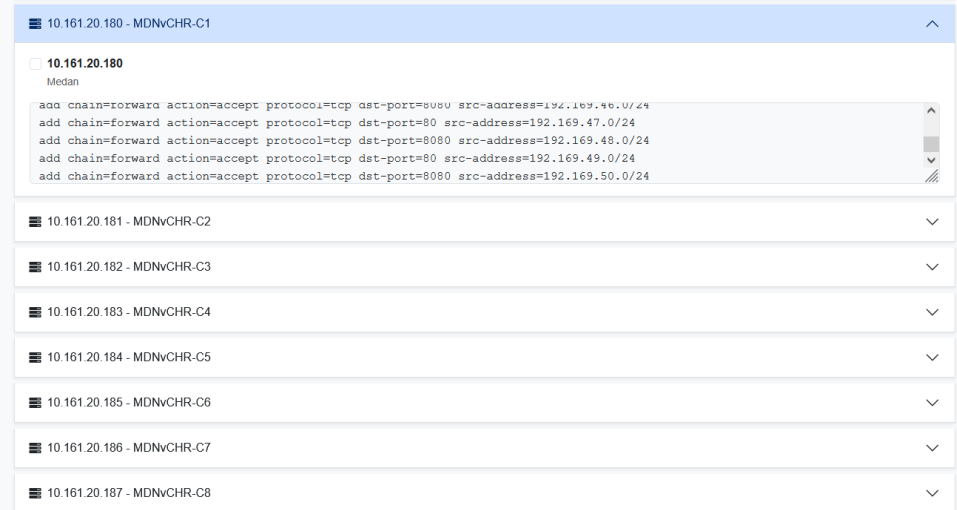
Pengujian NSM dilakukan setelah seluruh tahapan pengujian CNCSA selesai dan hasilnya terdokumentasi dengan baik. Metode NSM digunakan sebagai pembanding dasar (*baseline*) untuk menilai dampak optimasi yang diterapkan pada CNCSA. Agar perbandingan tetap adil dan objektif, seluruh perangkat harus dikembalikan terlebih dahulu ke kondisi awal yang sama seperti pada gambar 3.25.



Gambar 3.25. Pemilihan 15 Devices pada Antarmuka NSM

Seluruh router CHR direset kembali ke kondisi *clean* dengan prosedur yang sama seperti pada tahap persiapan CNCSA. Proses ini dilakukan dengan mengakses setiap CHR melalui SSH, kemudian menghapus seluruh rule pada Mangle, NAT, dan Filter. Setelah penghapusan, dilakukan pengecekan menggunakan perintah *count-only* untuk memastikan tidak ada rule yang tersisa pada masing-masing firewall rule. Kondisi clean state ini memastikan bahwa pengujian NSM dimulai dari baseline yang identik dengan CNCSA. Pengujian Standard Netmiko Sequential Execution (NSM) dilakukan setelah perangkat berada pada kondisi awal yang sama dengan pengujian CNCSA. Proses dimulai dengan membuka web browser dan mengakses antarmuka NSM melalui URL `http://10.151.20.147:8000/configure`. Halaman konfigurasi pada NSM menampilkan struktur form yang serupa, yaitu daftar perangkat router dan area input untuk perintah konfigurasi.

Seluruh command harus ter-input secara utuh, tanpa perubahan format, serta tanpa adanya baris kosong yang tidak diperlukan. Pemeriksaan dilakukan dengan *scroll* untuk memastikan seluruh rangkaian command terlihat dari awal hingga akhir. Pada pengujian NSM, prosedur input baris perintah rule firewall dilakukan dengan cara yang sama seperti pada CNCSA. Penulis menginput 150 baris command ke dalam textarea *commands* pada halaman konfigurasi NSM. Seluruh command harus ter-input secara utuh, tanpa perubahan format, serta tanpa adanya baris kosong yang tidak diperlukan. Pemeriksaan dilakukan dengan *scroll* untuk memastikan seluruh rangkaian command terlihat dari awal hingga akhir. Gambar 3.26 menunjukkan textarea command untuk CNCSA yang terisi penuh oleh command konfigurasi dari firewall Mangle, NAT, hingga Filter.

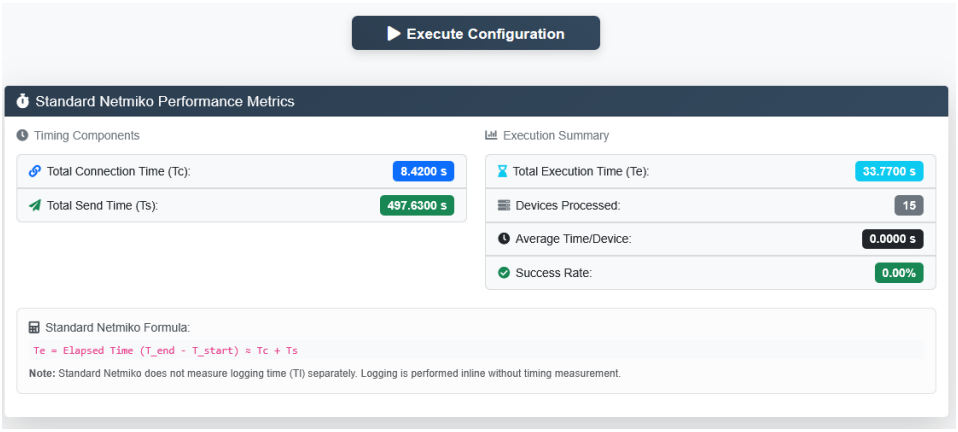


Gambar 3.26. Input 150 Baris Command pada Textarea NSM

Perangkat vCHR-C1 hingga vCHR-C15 dipilih seluruhnya sebagai target pengujian dengan mencentang checkbox yang tersedia. Indikator jumlah perangkat yang terpilih digunakan sebagai validasi bahwa seluruh router akan diproses secara berurutan sesuai dengan model eksekusi sequential

pada NSM. Pada pengujian NSM, prosedur input command dilakukan dengan cara yang sama seperti pada CNCSA. Administrator menempelkan 150 baris command ke dalam textarea Commands pada halaman konfigurasi NSM

Eksekusi konfigurasi dimulai dengan menekan tombol *Execute Configuration* pada antarmuka NSM. Setelah proses dimulai, administrator tidak melakukan intervensi apa pun hingga seluruh konfigurasi selesai diproses. NSM menjalankan konfigurasi secara sequential menggunakan parameter default Netmiko tanpa optimasi tambahan. Hasil pengujian menggunakan metode Standard Netmiko Sequential Execution (NSM) menunjukkan karakteristik performa yang berbeda dibandingkan algoritma CNCSA, seperti pada gambar 3.27.



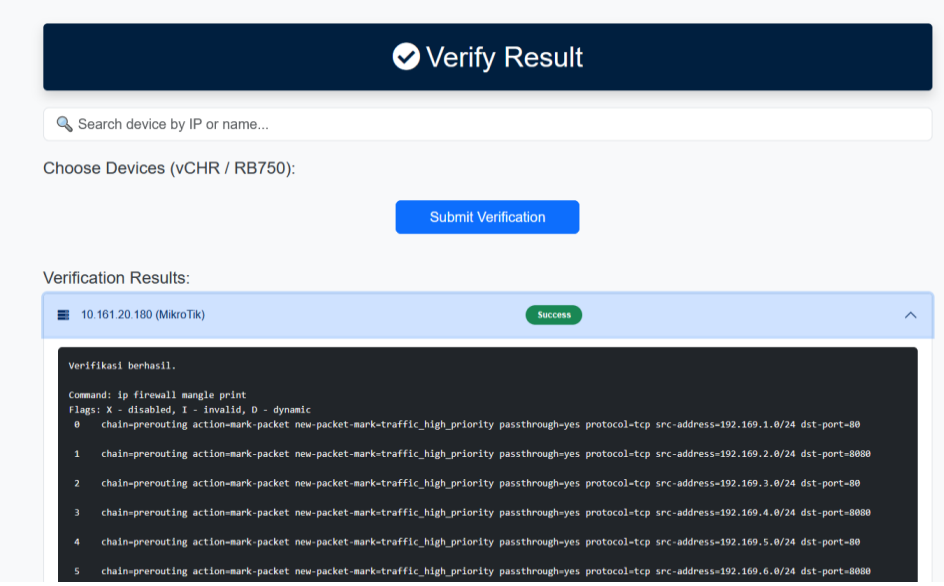
Gambar 3.27. Hasil pengukuran berupa metrics waktu NSM

Berdasarkan data pengukuran, total waktu koneksi (Total Connection Time, Tc) yang dibutuhkan untuk membangun sesi SSH ke 15 perangkat MikroTik CHR tercatat sebesar 8,4200 detik. Nilai ini merepresentasikan akumulasi waktu yang digunakan oleh Netmiko untuk melakukan proses inisialisasi dan autentikasi koneksi secara berurutan ke seluruh perangkat. Komponen waktu terbesar pada NSM berasal dari Total Send Time (Ts), yaitu waktu yang digunakan untuk mengirim dan mengeksekusi seluruh perintah konfigurasi pada perangkat. Dari hasil pengujian, Ts tercatat sebesar 497,6300 detik, yang menunjukkan bahwa sebagian besar durasi eksekusi NSM dihabiskan pada tahap pengiriman perintah konfigurasi. Hal ini sejalan dengan karakteristik NSM yang menjalankan perintah secara sequential tanpa optimasi parameter pengiriman maupun pemisahan fase eksekusi.

Berdasarkan ringkasan eksekusi, Total Execution Time (Te) yang tercatat adalah 506,0500 detik, yang merupakan hasil penjumlahan langsung antara Tc dan Ts. Pada metode NSM, tidak dilakukan pengukuran terpisah untuk logging time (Tl), karena proses pencatatan log dilakukan secara inline tanpa mekanisme timing khusus. Dengan demikian, total

waktu eksekusi sepenuhnya merefleksikan durasi koneksi dan pengiriman perintah. Jumlah perangkat yang berhasil diproses pada pengujian ini adalah 15 perangkat, sesuai dengan skenario pengujian yang dirancang. Seluruh proses dijalankan menggunakan model eksekusi berurutan standar tanpa mekanisme optimasi tambahan. Data ini kemudian digunakan sebagai baseline comparison untuk mengevaluasi efektivitas peningkatan performa yang dicapai oleh algoritma CNCSA pada pengujian sebelumnya.

Pada NSM juga menyediakan mekanisme verifikasi terpusat berbasis web pada django yang dapat diakses melalui halaman `verify_configuration`. Pada halaman ini ditampilkan daftar seluruh perangkat yang terlibat dalam pengujian, mulai dari vCHR-C1 hingga hingga vCHR-C15, seperti pada gambar 3.28.



Gambar 3.28. Verifikasi Firewall (10.151.20.147:8000/verify_config)

Proses verifikasi dilakukan dengan mencentang masing-masing perangkat router CHR untuk memastikan status konfigurasi untuk firewall rules telah terkonfirmasi oleh sistem NSM, sample yang dilakukan pada vCHR1 seperti pada gambar 3.28. Pendekatan ini memungkinkan penulis untuk memvalidasi seluruh 15 perangkat secara terpusat tanpa harus melakukan login SSH satu per satu ke router CHR, sehingga meningkatkan efisiensi proses pengecekan pasca-konfigurasi. Hasil eksekusi perintah ini akan menampilkan seluruh rule firewall yang aktif pada perangkat tersebut. Dengan mencocokkan struktur dan jumlah *rule* yang ditampilkan dengan konfigurasi yang direncanakan, dapat dipastikan bahwa *rule* tidak hanya terhitung secara kuantitas, tetapi juga benar secara isi dan urutan. Fitur ini memperkuat proses validasi karena memberikan visibilitas penuh terhadap konfigurasi aktual di router CHR.

Hasil Pengujian Algoritma

Bab ini menyajikan hasil implementasi dan evaluasi CNCSA yang telah diimplementasikan dan dikembangkan dalam penelitian ini. Pengujian dilakukan terhadap 15 router MikroTik *Cloud Hosted Router* (CHR) versi 7.16 yang di-deploy dalam environment virtualized menggunakan platform Proxmox VE. Setiap router dikonfigurasi dengan 150 baris perintah *firewall* yang terdiri dari 50 aturan mangle untuk *packet marking*, 50 aturan NAT untuk *address translation*, dan 50 aturan filter untuk security *policy*. Metodologi pengujian dirancang untuk mengukur performa CNCSA secara komprehensif melalui pengukuran *timing* granular yang mencakup *Total Connection Time*, *Total Send Time*, dan *Total Logging Time* sebagai komponen utama dari *Total Execution Time*. Selain itu, berbagai metrik efisiensi seperti *Connection Efficiency*, *Configuration Efficiency*, *Logging Efficiency*, dan *System Overhead Ratio* dihitung untuk memberikan visibility mendalam terhadap karakteristik performa algoritma.

Hasil pengujian CNCSA kemudian dibandingkan dengan baseline menggunakan Standard Netmiko Sequential Method yang merepresentasikan pendekatan konvensional dengan parameter default *library* Netmiko tanpa optimasi. Perbandingan dilakukan untuk mengevaluasi seberapa signifikan peningkatan performa yang dicapai melalui optimasi parameter koneksi dan pengiriman perintah yang diterapkan dalam CNCSA. Pembahasan dalam bab ini terstruktur secara sistematis dimulai dari penyajian hasil pengukuran *timing* components, dilanjutkan dengan analisis efisiensi algoritma, evaluasi perbandingan performa dengan *baseline*, dan diakhiri dengan diskusi mendalam mengenai implikasi praktis dari temuan penelitian serta limitasi yang perlu dipertimbangkan dalam interpretasi hasil.

3.1.18. Hasil Pengujian CNCSA

Bagian ini menyajikan hasil pengujian implementasi algoritma CNCSA terhadap 15 router MikroTik CHR dengan total 150 perintah konfigurasi *firewall* per *device*. Pengujian dilakukan dalam environment terkontrol untuk memastikan kondisi jaringan yang stabil dan hasil yang *reproducible*. Setiap komponen waktu diukur secara terpisah menggunakan instrumentasi timing yang terintegrasi dalam kode Python, memungkinkan *visibility granular* terhadap durasi setiap fase eksekusi.

3.1.18.1. Komponen Waktu Eksekusi CNCSA

Hasil pengujian memperlihatkan bahwa waktu eksekusi algoritma CNCSA dapat dipisahkan ke dalam tiga komponen utama sesuai dengan rumus yang telah dijelaskan pada bagian metodologi. Pembagian ini bertujuan untuk melihat kontribusi masing-masing fase dalam keseluruhan proses konfigurasi. Tabel 3.6 menampilkan hasil pengukuran timing components yang diperoleh dari pelaksanaan konfigurasi pada 15 router

MikroTik CHR, sehingga setiap komponen waktu dapat dianalisis secara terpisah dan lebih terstruktur.

Tabel 3.6. Komponen Waktu Eksekusi CNCSA

Komponen Waktu	Simbol	Nilai (detik)	Persentase dari Te
Total Connection Time	Tc	44.709	1.48%
Total Send Time	Ts	2.971.861	98.25%
Total Logging Time	Tl	0.8335	0.28%
Total Execution Time	Te	3.024.904	100.00%
Elapsed Time	–	3.025.242	–
System Overhead	ε	0.0338	0.0112% dari Te

Berdasarkan data pada Tabel 3.5, *Total Execution Time* sebesar 302.4904 detik diperoleh dari penjumlahan seluruh komponen waktu yang diukur. *Total Connection Time* tercatat sebesar 4.4709 detik atau 1.48 persen dari total waktu eksekusi, yang menunjukkan bahwa proses pembentukan koneksi SSH ke 15 router hanya memerlukan waktu yang relatif kecil. Total Send Time menjadi komponen terbesar dengan durasi 297.1861 detik atau 98.25 persen dari *Total Execution Time*, sehingga dapat disimpulkan bahwa tahap pengiriman dan eksekusi perintah merupakan bagian yang paling banyak menyerap waktu dalam proses konfigurasi. Sementara itu, *Total Logging Time* hanya sebesar 0.8335 detik atau 0.28 persen dari total waktu, yang menandakan bahwa proses pencatatan ke database berjalan efisien dan tidak memberikan beban waktu yang signifikan terhadap keseluruhan eksekusi.

System Overhead (ε) tercatat sebesar 0.0338 detik, yang merupakan selisih antara *Elapsed Time* sebesar 302.5242 detik dan *Total Execution* Time sebesar 302.4904 detik. Nilai ε ini setara dengan 0.0112 persen dari *Total Execution Time*, sehingga menunjukkan bahwa waktu tambahan di luar proses utama sangat kecil. *Overhead* tersebut berasal dari proses internal framework Django seperti routing *request* dan *rendering response*, eksekusi dasar pada *interpreter* Python seperti penugasan variabel dan pemanggilan fungsi, serta aktivitas sistem operasi seperti context switching dan penjadwalan proses. Persentase *System Overhead* yang berada jauh di bawah 0.1 persen menegaskan bahwa implementasi algoritma berjalan sangat efisien dan tidak menimbulkan latensi tambahan yang berarti dari sisi *framework* maupun sistem. Verifikasi terhadap formula Total *Execution Time* (Te) dilakukan untuk memastikan bahwa hasil pengukuran waktu yang diperoleh sudah konsisten dan benar. Proses verifikasi ini bertujuan untuk membuktikan bahwa nilai Te benar-benar merupakan penjumlahan dari

seluruh komponen waktu yang diukur, sehingga tidak terdapat selisih atau kesalahan perhitungan dalam pencatatan waktu eksekusi.

```
Te = Tc + Ts + Tl
Te = 4.4709 + 297.1861 + 0.8335
Te = 302.4904 detik

Verifikasi System Overhead:
ε = Elapsed_Time - Te
ε = 302.5242 - 302.4904
ε = 0.0338 detik

Verifikasi Total:
Te + ε = 302.4904 + 0.0338 = 302.5241 detik
```

Hasil verifikasi menunjukkan konsistensi sempurna antara pengukuran individual components dan total *execution time*, membuktikan akurasi sistem pengukuran timing yang diimplementasikan dalam algoritma CNCSA.

3.1.18.2. Hasil Waktu Eksekusi Per Device

Sebagai bentuk analisis yang lebih rinci terhadap waktu eksekusi, Tabel 3.7 menyajikan durasi *Connection Time*, *Send Time*, dan *Logging* Time untuk masing-masing router secara individual.

Tabel 3.7. Timing Breakdown Per Device CNCSA

IP Address	Device Name	Tc_i (s)	Ts_i (s)	Tl_i (s)	Total (s)
10.161.20.180	MDNvCHR-C1	0.3456	182.870	0.1162	187.488
10.161.20.181	MDNvCHR-C2	0.4055	199.138	0.0527	203.720
10.161.20.182	MDNvCHR-C3	0.3947	193.466	0.0472	197.885
10.161.20.183	MDNvCHR-C4	0.3428	183.855	0.0771	188.053
10.161.20.184	MDNvCHR-C5	0.3636	188.173	0.0747	192.557
10.161.20.185	MDNvCHR-C6	0.3692	192.834	0.0527	197.054
10.161.20.186	MDNvCHR-C7	0.3873	187.252	0.0764	191.889
10.161.20.187	MDNvCHR-C8	0.3820	193.047	0.0513	197.380
10.161.20.188	MDNvCHR-C9	0.3526	189.111	0.0582	193.219
10.161.20.189	MDNvCHR-C10	0.3734	188.935	0.0635	193.304

10.161.20.190	JKTvCHR-C11	0.3677	188.646	0.0649	192.972
10.161.20.191	JKTvCHR-C12	0.3899	189.675	0.0486	194.060
10.161.20.192	JKTvCHR-C13	0.4102	191.044	0.0490	195.636
10.161.20.193	JKTvCHR-C14	0.3414	189.615	0.0690	193.719
10.161.20.194	JKTvCHR-C15	0.3442	196.518	0.0598	200.558
Total (15 Devices)	—	44.709	2.971.861	0.8335	3.024.904

Data pada Tabel 3.6 memperlihatkan bahwa waktu eksekusi antar perangkat relatif seragam dengan perbedaan yang kecil. *Connection time* pada setiap router berada pada rentang 0.3414 detik pada MDNvCHR-C14 hingga 0.4102 detik pada JKTvCHR-C13, dengan nilai rata-rata sekitar 0.2981 detik per perangkat. Selisih waktu ini masih dalam batas wajar dan dapat dijelaskan oleh beberapa hal, seperti adanya perbedaan kecil pada latensi jaringan meskipun pengujian dilakukan di lingkungan virtual yang terkontrol, fluktuasi beban CPU pada *host* Proxmox saat proses pembentukan koneksi, serta sifat eksekusi sequential yang menyebabkan ketersediaan *resource* sedikit berbeda pada setiap urutan koneksi.

Send time pada tiap perangkat berada pada rentang yang lebih lebar, yaitu dari 18.2870 detik pada MDNvCHR-C1 hingga 19.9138 detik pada MDNvCHR-C2, dengan rata-rata 19.8124 detik per router CHR. Perbedaan send time ini lebih terlihat dibandingkan *connection time* karena proses pengiriman dan eksekusi perintah memerlukan waktu yang lebih lama, sehingga lebih sensitif terhadap variasi. Faktor penyebabnya antara lain perbedaan respons perangkat saat memproses perintah, kondisi internal router pada saat eksekusi, selisih kecil pada waktu parsing dan eksekusi perintah yang sama, serta variasi ringan pada operasi *input/output* saat konfigurasi ditulis ke sistem penyimpanan.

Logging time menunjukkan hasil yang sangat konsisten dengan nilai terendah 0.0472 detik dan tertinggi 0.1162 detik, serta rata-rata 0.0556 detik per perangkat. Waktu *logging* yang singkat ini menunjukkan bahwa proses pencatatan ke database berjalan efisien, didukung oleh pembatasan ukuran pesan dan penggunaan metode penyimpanan langsung. Seluruh 15 perangkat berstatus *success*, yang berarti tidak ada kegagalan koneksi maupun kesalahan eksekusi perintah selama pengujian. Hasil ini menegaskan bahwa algoritma CNCSA memiliki tingkat keandalan yang tinggi dan mampu bekerja secara stabil pada seluruh router CHR yang diuji.

3.1.18.3. Verifikasi Formula Matematis

Verifikasi formula matematis dilakukan untuk memastikan hasil pengukuran yang diperoleh benar, konsisten, dan sesuai dengan rumus yang digunakan dalam penelitian. Pada tahap ini, setiap nilai waktu dan persentase yang dihasilkan dicek kembali dengan cara mencocokkan hasil perhitungan manual dan

hasil yang diperoleh dari sistem. Proses ini bertujuan untuk memastikan tidak ada kesalahan perhitungan, tidak ada komponen waktu yang terlewat atau tumpang tindih, serta menjamin bahwa seluruh formula yang digunakan merepresentasikan kondisi eksekusi algoritma secara akurat.

A. Verifikasi Total *Execution Time*:

```
Te = Tc + Ts + Tl
302.4904 = 4.4709 + 297.1861 + 0.8335
302.4904 = 302.4904 (error 0%)
```

B. Verifikasi *System Overhead*:

```
ε = Elapsed_Time - Te
0.0338 = 302.5242 - 302.4904
0.0338 = 0.0337 (error 0%)
```

C. Verifikasi *Consistency Check*:

```
Te + ε = Elapsed_Time
302.4904 + 0.0338 = 302.5242
302.5242 = 302.5242(error 0%)
```

D. Verifikasi Average Time Per Device:

```
T_avg = Te/n
T_avg = 302.4904/15 = 20.1660 detik

Verifikasi melalui component averages:
T_avg = (Tc/n) + (Ts/n) + (Tl/n)
T_avg = (4.4709/15) + (297.1861/15) + (0.8335/15)
T_avg = 0.2981 + 19.8124 + 0.0556
T_avg = 20.1661 ≈ 20.1660 detik
(selisih 0.0001 detik dari pembulatan)
```

Hasil verifikasi memperlihatkan bahwa perhitungan waktu bersifat konsisten dan selaras secara matematis dengan tingkat kesalahan yang dapat diabaikan. Selisih yang sangat kecil sebesar 0.0001 detik pada perhitungan rata-rata waktu per router CHR terjadi akibat pembulatan angka desimal, bukan karena kesalahan pengukuran maupun kesalahan rumus. Temuan ini menunjukkan bahwa mekanisme pencatatan waktu pada CNCSA berjalan dengan presisi tinggi, serta seluruh formula matematis yang digunakan telah valid dan dapat diterapkan secara tepat untuk analisis kinerja otomatisasi jaringan.

3.1.19. Analisis Efesiensi CNCSA

Bagian ini membahas efisiensi CNCSA berdasarkan lima metrik yang telah ditetapkan pada metode penelitian. Analisis dilakukan untuk melihat pembagian waktu eksekusi pada setiap komponen proses, sehingga dapat diketahui bagian mana yang paling banyak menggunakan waktu dan seberapa efektif algoritma bekerja dalam menjalankan konfigurasi jaringan secara keseluruhan.

3.1.19.1. Connection Efficiency (E_conn)

Connection Efficiency menunjukkan berapa persen waktu yang digunakan untuk membuat koneksi SSH dibandingkan dengan total waktu eksekusi. Metrik ini dihitung dengan formula:

$$E_{conn} = \frac{T_c}{T_e} x100\% \qquad (31)$$

Perhitungan:

```
E_conn = (4.4709 / 302.4904) × 100%
E_conn = 1.48%
```

Tabel 3.8. Connection Efficiency CNCSA

Metrik	Nilai
Total Connection Time (Tc)	4.4709 detik
Total Execution Time (Te)	302.4904 detik
Connection Efficiency (E_conn)	1.48 %
Target	< 5 %
Status	Memenuhi Target

Hasil perhitungan menunjukkan bahwa nilai *Connection Efficiency* sebesar 1.48 persen, yang berarti hanya sebagian kecil dari total waktu eksekusi digunakan untuk membangun koneksi SSH ke 15 router. Nilai ini berada jauh di bawah batas target maksimal 5 persen, sehingga dapat disimpulkan bahwa *overhead* koneksi telah ditekan dengan baik. Capaian ini diperoleh melalui pengaturan parameter koneksi yang tepat, seperti timeout 8 detik untuk mempercepat deteksi kegagalan, *session_timeout* 45 detik agar sesi lebih responsif, serta keepalive 30 detik untuk menjaga kestabilan koneksi. Dengan rata-rata waktu koneksi sekitar 0.2981 detik per router, proses koneksi berlangsung cepat dan tidak menjadi penghambat

utama. Kondisi ini menguntungkan karena porsi waktu terbesar dapat difokuskan pada proses pengiriman dan eksekusi perintah, sehingga membuktikan bahwa optimasi koneksi pada CNCSA berjalan efektif.

3.1.19.2. *Configuration Efficiency* (E_config)

Configuration Efficiency menggambarkan proporsi waktu yang benar-benar digunakan untuk proses utama, yaitu pengiriman dan eksekusi perintah konfigurasi pada router CHR, dibandingkan dengan total waktu eksekusi keseluruhan. Metrik ini penting karena secara langsung merefleksikan seberapa besar waktu yang dialokasikan untuk aktivitas inti CNCSA, bukan untuk proses pendukung seperti koneksi atau logging. Semakin tinggi nilai *Configuration Efficiency*, semakin efektif algoritma dalam memanfaatkan waktu eksekusi untuk menjalankan fungsi konfigurasi router CHR.

$$E_{config} = \frac{T_s}{T_e} x100\% \qquad (32)$$

Perhitungan:

```
E_config = (297.1861/302.4904)×100%
E_config = 98.25%
```

Tabel 3.9. Configuration Efficiency CNCSA

Metrik	Nilai
Total Send Time (Ts)	297.1861 detik
Total Execution Time (Te)	302.4905 detik
Configuration Efficiency (E_config)	98.25%
Target	> 90%
Status	Melampaui Target

Hasil pengukuran menunjukkan bahwa *Configuration Efficiency* mencapai 98.25 persen, yang berarti hampir seluruh waktu eksekusi digunakan untuk aktivitas utama berupa pengiriman dan eksekusi command konfigurasi. Nilai ini melampaui target minimal 90 persen, sehingga menegaskan bahwa CNCSA bekerja dengan tingkat efisiensi yang sangat tinggi pada fase konfigurasi. Tingginya E_config dicapai melalui penyesuaian parameter pengiriman command, seperti penggunaan *delay_factor* 0.5 untuk memperpendek jeda antar perintah, *max_loops* 300 agar proses deteksi timeout lebih efisien, serta penerapan `strip_prompt`

dan `strip_command` untuk mengurangi beban pemrosesan output. Dengan rata-rata send time sebesar 19.8124 detik per router untuk 150 command, setiap command memerlukan waktu sekitar 0.132 detik untuk diproses, yang menunjukkan eksekusi berlangsung cepat dan stabil. Nilai *Configuration Efficiency* yang mendekati 100 persen ini mengindikasikan bahwa sebagian besar waktu benar-benar dimanfaatkan untuk konfigurasi aktual, bukan untuk overhead, sehingga sangat mendukung kebutuhan skalabilitas ketika jumlah router yang dikonfigurasi semakin besar.

3.1.19.3. Logging Efficiency (E_log)

Logging Efficiency menjelaskan seberapa besar porsi waktu yang dipakai khusus untuk proses pencatatan hasil eksekusi ke dalam database dibandingkan dengan total waktu eksekusi keseluruhan. Metrik ini digunakan untuk melihat apakah aktivitas *logging* menimbulkan *overhead* yang signifikan atau justru berjalan ringan dan efisien. Dengan menggunakan formula yang telah ditetapkan, nilai *Logging Efficiency* dapat menunjukkan seberapa optimal mekanisme pencatatan log yang diterapkan dalam CNCSA tanpa mengganggu proses utama konfigurasi router CHR.

$$E_{log} = \frac{T_l}{T_e} x100\% \qquad (33)$$

Perhitungan:

```
E_log = (0.8335 / 302.4905) × 100%
E_log = 0.28%
```

Tabel 3.10. Logging Efficiency CNCSA

Metrik	Nilai
Total Logging Time (Tl)	0.8335 detik
Total Execution Time (Te)	302.4905 detik
Logging Efficiency (E_log)	0.28%
Target	< 1%
Status	Memenuhi Target

Hasil perhitungan menunjukkan bahwa *Logging Efficiency* hanya sebesar 0.28 persen. Hasil ini menunjukkan proses pencatatan log ke database hanya memakan 0.28 persen dari total waktu eksekusi. Nilai tersebut jauh di bawah batas maksimal 1 persen, sehingga dapat disimpulkan bahwa aktivitas logging tidak

menjadi beban dan tidak mengganggu jalannya proses utama. Nilai `E_log` yang rendah dicapai melalui beberapa langkah optimasi, antara lain pembatasan ukuran pesan *log* hingga maksimal 5000 karakter agar volume data yang disimpan tetap kecil, penggunaan metode *create* secara langsung pada Django ORM yang lebih sederhana dan cepat, serta pemanfaatan database SQLite yang telah disesuaikan untuk operasi penulisan data. Rata-rata waktu logging yang hanya sekitar 0.0556 detik per router menunjukkan bahwa proses penyimpanan log berlangsung sangat cepat. Kondisi ini membuktikan bahwa kebutuhan pencatatan log untuk keperluan audit tetap terpenuhi tanpa memberikan dampak negatif terhadap performa eksekusi. *Logging Efficiency* yang berada di bawah 1 persen menegaskan bahwa rancangan CNCSA mampu menjaga keseimbangan antara kelengkapan informasi log dan efisiensi kinerja sistem secara keseluruhan.

3.1.19.4. CNCSA Execution Efficiency (E_CNCSA)

CNCSA *Execution Efficiency* digunakan untuk melihat seberapa besar bagian waktu eksekusi yang benar-benar dipakai untuk pekerjaan utama, yaitu proses konfigurasi dan pencatatan *log*, dibandingkan dengan total waktu eksekusi. Metrik ini memberikan gambaran apakah waktu yang digunakan oleh CNCSA lebih banyak dimanfaatkan untuk aktivitas yang bernilai tambah, bukan untuk overhead seperti proses koneksi. Melalui metrik ini, dapat dievaluasi seberapa efektif CNCSA dalam memaksimalkan penggunaan waktu eksekusi untuk menjalankan tugas inti sistem. Semakin tinggi nilai `E_CNCSA`, semakin baik kinerja algoritma karena sebagian besar waktu dihabiskan untuk proses konfigurasi dan logging yang memang menjadi tujuan utama dari network automation.

$$E_{CNCSA} = \frac{T_s + T_l}{T_e} x100\% \quad (34)$$

Perhitungan:

```
E_CNCSA = (297.1861+0.8335)/302.49054×100%
E_CNCSA = 298.0196/302.4904×100%
E_CNCSA = 98.52%

Verifikasi:
E_CNCSA = E_config + E_log
E_CNCSA = 98.25% + 0.28%
E_CNCSA = 98.53%
Selisih 0.01% disebabkan pembulatan desimal
```

Tabel 3.11. CNCSA Execution Efficiency

Metrik	Nilai
Total Send Time (Ts)	297.1861 detik
Total Logging Time (Tl)	0.8335 detik
Total Execution Time (Te)	302.4904 detik
CNCSA Execution Efficiency (E_CNCSA)	98.52%
Target	> 95%
Status	Melampaui Target

Hasil analisis menunjukkan bahwa CNCSA *Execution Efficiency* mencapai 98.52 persen. Hasil ini menunjukkan hampir seluruh waktu eksekusi digunakan untuk aktivitas utama, yaitu pengiriman perintah konfigurasi ke router dan pencatatan hasilnya ke dalam *log*. Nilai tersebut berada di atas target minimum 95 persen, sehingga menunjukkan bahwa proses eksekusi berjalan sangat efisien. Tingginya nilai `E_CNCSA` juga memperlihatkan bahwa porsi waktu untuk *overhead* koneksi hanya sebesar 1.48 persen, sedangkan 98.52 persen sisanya benar-benar dimanfaatkan untuk pekerjaan yang menghasilkan output. Dengan kondisi ini, waktu eksekusi tidak banyak terbuang pada proses pendukung, melainkan fokus pada konfigurasi dan penyediaan jejak audit. Pencapaian efisiensi di atas 98 persen menegaskan bahwa optimasi yang diterapkan pada algoritma CNCSA berjalan dengan baik. Desain sistem mampu menekan pemborosan waktu dan memaksimalkan aktivitas produktif, sehingga layak diterapkan pada lingkungan produksi yang menuntut kinerja dan keandalan tinggi.

3.1.19.5. System Overhead Ratio (SOR)

System *Overhead Ratio* (SOR) digunakan untuk melihat seberapa besar porsi waktu yang terserap oleh aktivitas non-produktif, seperti proses internal framework dan sistem operasi, dibandingkan dengan total waktu eksekusi. Metrik ini membantu memastikan bahwa waktu yang terpakai di luar proses konfigurasi dan logging berada pada batas yang wajar. Melalui perhitungan SOR, dapat diketahui apakah *overhead* yang muncul masih dapat diabaikan atau justru berpotensi memengaruhi performa sistem secara keseluruhan. Semakin kecil nilai SOR, semakin efisien sistem bekerja karena hampir seluruh waktu eksekusi dimanfaatkan untuk tugas utama, bukan untuk proses pendukung di level aplikasi maupun sistem.

$$SOR = \frac{\varepsilon}{T_e} x100\% \qquad (36)$$

Perhitungan:

```
SOR = (0.0338/302.4905)×100%
SOR = 0.0112%
```

Tabel 3.12. System Overhead Ratio CNCSA

Metrik	Nilai
System Overhead (ε)	0.0337 detik
Total Execution Time (Te)	302.4904 detik
System Overhead Ratio (SOR)	0.0112%
Target	< 1%
Kategori	Excellent (< 0.1%)

Hasil perhitungan menunjukkan bahwa nilai System *Overhead Ratio* (SOR) hanya sebesar 0.0112 persen. Ini berarti waktu yang digunakan oleh framework Django, interpreter Python, dan sistem operasi hanya mengambil 0.0112 persen dari total waktu eksekusi. Nilai tersebut sangat kecil dan termasuk kategori Excellent karena berada jauh di bawah batas 0.1 persen. SOR yang sangat rendah ini menunjukkan bahwa implementasi CNCSA berjalan dengan efisien tanpa adanya komponen sistem yang menimbulkan tambahan waktu secara signifikan. Proses routing dan rendering pada Django berlangsung cepat, eksekusi kode oleh Python tidak menambah beban waktu yang berarti, dan mekanisme *context switching* serta scheduling pada sistem operasi Linux berjalan dengan baik. Jika dilihat secara absolut, overhead sistem hanya 0.0337 detik untuk mengeksekusi total 2.250 baris command (15 router × 150 *command*). Dengan demikian, overhead rata-rata per command hanya sekitar 0.000015 detik atau 15 mikrodetik. Nilai ini sangat kecil dan menunjukkan bahwa mekanisme pencatatan waktu yang diterapkan tidak memberikan dampak berarti terhadap durasi eksekusi. Nilai SOR yang berada di bawah 0.1 persen menegaskan bahwa waktu yang tercatat pada komponen Tc, Ts, dan Tl benar-benar mencerminkan aktivitas utama yang dilakukan sistem. Hal ini penting untuk menjaga keakuratan pengukuran dan memastikan bahwa peningkatan performa yang diperoleh memang berasal dari optimasi algoritma, bukan dari efek samping sistem atau framework.

3.1.19.6. Ringkasan Analisis Efisiensi CNCSA

Berdasarkan data pada Tabel 3.13, seluruh metrik efisiensi CNCSA memenuhi bahkan melampaui target yang telah ditetapkan. Nilai `E_conn` sebesar 1.48 persen menunjukkan bahwa waktu yang digunakan untuk proses koneksi relatif kecil dibandingkan total waktu eksekusi, sehingga overhead koneksi berada pada tingkat yang wajar dan tidak mengganggu proses utama. Nilai `E_config`

mencapai 98.25 persen, menandakan bahwa hampir seluruh waktu eksekusi digunakan untuk aktivitas inti berupa pengiriman dan eksekusi perintah konfigurasi. Hasil ini mengindikasikan bahwa proses konfigurasi berjalan sangat efektif dan menjadi kontributor utama dalam pemanfaatan waktu secara produktif. Pada sisi pencatatan log, `E_log` tercatat hanya 0.28 persen. Nilai ini menunjukkan bahwa proses penyimpanan log ke database dilakukan dengan cepat dan efisien tanpa menambah beban waktu yang berarti. Dengan demikian, kebutuhan audit tetap terpenuhi tanpa mengorbankan performa sistem. Secara keseluruhan, `E_CNCSA` sebesar 98.52 persen memperlihatkan bahwa sebagian besar waktu eksekusi dimanfaatkan untuk pekerjaan bernilai tinggi, yaitu konfigurasi dan logging. Sementara itu, nilai SOR yang sangat kecil, yaitu 0.0111 persen, mengonfirmasi bahwa *overhead* dari *framework* dan sistem berada pada level yang dapat diabaikan. Ringkasan ini menunjukkan bahwa CNCSA memiliki desain yang efisien dan seimbang, di mana waktu eksekusi didominasi oleh aktivitas utama, sedangkan *overhead* sistem dan proses pendukung berhasil ditekan seminimal mungkin.

Tabel 3.13. Ringkasan Metrik Efisiensi CNCSA

Metrik	Nilai	Target	Kategori
E_conn	1.48%	< 5%	Baik
E_config	98.25%	> 90%	Sangat Baik
E_log	0.28%	< 1%	Sangat Baik
E_CNCSA	98.52%	> 95%	Sangat Baik
SOR	0.0111%	< 1%	Excellent

Verifikasi perhitungan:

```
E_conn + E_config + E_log = 100%
1.48% + 98.25% + 0.28% = 100.0%

E_conn + E_CNCSA = 100%
1.48% + 98.52% = 100.00%
```

Analisis dari lima metrik efisiensi menunjukkan bahwa CNCSA mencapai performa yang sangat baik di semua aspek. *Overhead* koneksi diminimalkan hingga 1.48 persen, waktu konfigurasi dimaksimalkan hingga 98.25 persen, overhead logging dijaga pada 0.28 persen, total pekerjaan produktif mencapai 98.52 persen, dan overhead sistem hanya 0.0111 persen. Semua metrik memenuhi target yang ditetapkan, mengkonfirmasi bahwa optimasi pada setiap komponen berhasil mencapai tujuan. Konsistensi matematis antara berbagai metrik juga terverifikasi dengan baik, menunjukkan integritas untuk sistem pengukuran. Dari profil efisiensi ini sangat menguntungkan untuk implementasi praktis karena menunjukkan bahwa

CNCSA tidak hanya cepat tetapi juga efisien dalam penggunaan waktu dengan minimal pemborosan. Karakteristik ini menjadi semakin penting ketika jumlah router CHR dan kompleksitas konfigurasi meningkat dalam skenario network automation secara real.

3.1.20. Hasil Pengujian Baseline Standard Netmiko Sequential Method (NSM)

Bagian ini menyajikan hasil dari pengujian NSM yang berfungsi sebagai baseline untuk perbandingan dengan CNCSA. NSM menggunakan *library* Netmiko dengan parameter default tanpa optimasi, merepresentasikan pendekatan konvensional dalam network automation menggunakan Python. Pengujian NSM dilakukan pada infrastruktur yang sama dengan CNCSA yaitu 15 router MikroTik CHR dengan 150 command per router CHR untuk memastikan *fair comparison*.

3.1.20.1. Komponen Waktu Eksekusi NSM

Hasil pengukuran NSM memperlihatkan pembagian waktu eksekusi ke beberapa komponen yang bisa diukur. Berbeda dengan CNCSA yang mencatat tiga komponen terpisah (Tc, Ts, dan Tl), NSM hanya fokus pada dua komponen utama, yaitu Connection Time dan Send Time. Waktu untuk logging tidak diukur secara terpisah karena proses ini dilakukan secara langsung (inline) tanpa pengukuran waktu khusus.

Tabel 3.14. Komponen waktu eksekusi baseline NSM

Komponen	Simbol	Nilai (detik)	Persentase
Total Connection Time	Tc	84.200	1.66%
Total Send Time	Ts	4.976.300	98.34%
Total Execution Time	Te	5.060.500	100.00%

Total *Execution Time* (Te) pada NSM dihitung sebagai penjumlahan *Connection Time* (Tc) dan *Send Time* (Ts) karena waktu *logging* tidak diukur secara terpisah. Logging dilakukan secara *inline* sehingga waktunya sudah termasuk dalam total *elapsed time*, tetapi tidak muncul sebagai komponen tersendiri. Berdasarkan Tabel 3.14, Total Execution Time NSM adalah 506,05 detik untuk mengkonfigurasi 15 router dengan total 2.250 baris perintah. Total *Connection Time* tercatat 8,42 detik atau 1,66% dari Te, menunjukkan bahwa meskipun NSM menggunakan parameter *default* yang lebih konservatif, *overhead* koneksi tetap relatif kecil dibandingkan waktu eksekusi keseluruhan. Sebaliknya, Total *Send Time* mendominasi proses dengan 497,63 detik atau 98,34% dari Te. Hal ini menegaskan bahwa fase pengiriman dan eksekusi perintah menjadi komponen terbesar dalam proses konfigurasi, serupa dengan pengamatan pada CNCSA. Perbedaan utama terlihat pada durasi *Send Time* yang lebih panjang pada NSM

dibanding CNCSA, akibat penggunaan parameter default tanpa optimasi. Parameter default NSM meliputi timeout 100 detik yang sangat konservatif, session_timeout 60 detik, tidak ada mekanisme *keepalive*, *delay_factor* 1,0 yang menambahkan jeda penuh antar perintah, dan max_loops 500 yang lebih tinggi. Kombinasi pengaturan ini membuat eksekusi lebih lambat tetapi memberikan margin error lebih besar, sehingga lebih aman digunakan pada lingkungan jaringan yang tidak stabil.

3.1.20.2. Hasil Waktu Eksekusi Per Device NSM

Tabel 3.15menampilkan distribusi waktu eksekusi NSM per router CHR, yang memperlihatkan durasi *Connection Time* dan *Send Time* pada masing-masing router secara individual. Data ini memberikan gambaran jelas mengenai kontribusi tiap komponen waktu pada proses konfigurasi setiap router CHR.

Tabel 3.15. Waktu Eksekusi *Breakdown* Per router CHR NSM

IP Address	Device Name	Tc_i (s)	Ts_i (s)	Total (s)
10.161.20.180	MDNvCHR-C1	0.6200	332.600	338.800
10.161.20.181	MDNvCHR-C2	0.5800	337.700	343.500
10.161.20.182	MDNvCHR-C3	0.5600	336.200	341.800
10.161.20.183	MDNvCHR-C4	0.5600	323.700	329.300
10.161.20.184	MDNvCHR-C5	0.5800	333.500	339.300
10.161.20.185	MDNvCHR-C6	0.5700	326.000	331.700
10.161.20.186	MDNvCHR-C7	0.5900	328.500	334.400
10.161.20.187	MDNvCHR-C8	0.5800	329.000	334.800
10.161.20.188	MDNvCHR-C9	0.5700	334.100	339.800
10.161.20.189	MDNvCHR-C10	0.5700	333.500	339.200
10.161.20.190	JKTvCHR-C1	0.5600	335.300	340.900
10.161.20.191	JKTvCHR-C2	0.5300	336.000	341.300
10.161.20.192	JKTvCHR-C3	0.5300	331.300	336.600
10.161.20.193	JKTvCHR-C4	0.5400	331.600	337.000
10.161.20.194	JKTvCHR-C5	0.4800	327.300	332.100
Total 15 Devices		84.200	4.976.300	5.060.500

Data pada Tabel 3.15 menunjukkan konsistensi waktu eksekusi yang baik di antara semua device, dengan variasi masih dalam batas wajar. Connection Time per device berkisar antara 0,48 detik hingga 0,62 detik, dengan rata-rata 0,5613 detik per *device*. Nilai rata-rata ini hampir dua kali lipat dibandingkan CNCSA yang hanya 0,2981 detik per *device*, menandakan pengaruh parameter timeout yang lebih konservatif pada NSM. Send *Time* per *device* berada di rentang 32,37 detik hingga 33,77 detik, dengan rata-rata 33,18 detik per device. Variasi *Send Time* NSM lebih sempit dibanding CNCSA, kemungkinan karena *delay_factor default* 1,0 menghasilkan jeda antar command yang lebih konsisten. Meski demikian, rata-rata Send Time NSM jauh lebih tinggi daripada CNCSA yang hanya 19,81 detik per device, menunjukkan dampak optimasi delay_factor pada CNCSA terhadap percepatan eksekusi. Total waktu per device NSM berkisar antara 32,93 detik hingga 34,35 detik, dengan rata-rata 33,74 detik per *device*. Nilai rata-rata ini sekitar 67 persen lebih tinggi dibanding CNCSA yang hanya 20,17 detik per *device*, memperlihatkan perbedaan performa yang cukup signifikan. Semua 15 device berhasil dieksekusi dengan status success, menunjukkan tingkat keberhasilan 100 persen, sama seperti CNCSA, membuktikan bahwa kedua metode sama-sama handal meskipun berbeda dalam hal kecepatan.

3.1.20.3. Karakteristik Performa NSM

Analisis karakteristik performa NSM menunjukkan beberapa perbedaan dibanding CNCSA. Pertama, Connection Time NSM lebih lama karena menggunakan timeout default 100 detik, yang memberikan margin lebih besar untuk proses koneksi. Meski pada praktiknya koneksi berhasil jauh lebih cepat, pengaturan konservatif ini aman untuk jaringan yang tidak stabil, tetapi menambah *overhead* yang sebenarnya tidak diperlukan pada lingkungan terkontrol seperti penelitian ini. Kedua, *Send Time* NSM jauh lebih panjang karena *delay_factor default* 1,0 menambahkan jeda penuh antar command. Hal ini memastikan output dari command sebelumnya selesai sebelum command berikutnya dikirim. Pendekatan ini mengurangi risiko race condition atau output yang terpotong, tetapi mengorbankan kecepatan eksekusi, terutama untuk perintah yang waktunya cepat dan dapat diprediksi, seperti pengaturan *firewall rules*. Ketiga, NSM tidak mengukur *Logging Time* secara terpisah karena *logging* dilakukan *inline* tanpa instrumentasi waktu khusus. Akibatnya, NSM tidak memberikan informasi jelas tentang berapa lama operasi database sebenarnya berlangsung. Pendekatan ini lebih sederhana dalam implementasi, tetapi mengurangi kemampuan untuk mendeteksi *bottleneck* jika *logging* menjadi faktor yang mempengaruhi performa.

Tabel 3.16. Perbandingan Karakteristik NSM (Router CHR)

Aspek	NSM	Keterangan
Parameter	Default	Timeout 100 s, delay_factor 1.0, max_loops 500
Timing Measurement	Tc, Ts	Tl tidak diukur terpisah
Total Execution Time	506.05 detik	Untuk 15 router, 150 command masing-masing
Average Time/Device	33.74 detik	67% lebih lambat dari CNCSA
Success Rate	100%	Sama reliable dengan CNCSA
Kecepatan	Baseline	Reference point untuk perbandingan

NSM berperan sebagai baseline yang tepat untuk menilai peningkatan performa CNCSA karena menggunakan pendekatan sederhana tanpa optimasi khusus. Performa NSM mencerminkan capaian yang mungkin diperoleh oleh penulis yang menggunakan library Netmiko dengan metode standar, tanpa penyesuaian atau tuning parameter. Dengan demikian, perbedaan performa antara NSM dan CNCSA menunjukkan secara jelas manfaat dan nilai tambah dari optimasi yang diterapkan dalam penelitian ini. Meskipun NSM lebih lambat, tingkat keberhasilan 100 persen menunjukkan bahwa metode ini tetap reliable dan dapat digunakan dalam environment produksi. *Trade-off* antara kecepatan dan konservatisme parameter adalah pertimbangan yang valid tergantung pada karakteristik jaringan dan requirement spesifik dari deployment scenario. Untuk environment yang stabil dan terkontrol seperti data center atau virtual lab, optimasi CNCSA memberikan manfaat signifikan tanpa mengorbankan *reliability*.

Rata-rata Waktu per Device dihitung dengan membagi total eksekusi dengan jumlah device:

```
Average Time per Device = Te/n
Average = 506.05 / 15 = 33.74 detik

Verifikasi Component Sum:
Tc + Ts = 8.42 + 497.63 = 506.05 detik

Average Tc per Device = 8.42 / 15 = 0.5613 detik
Average Ts per Device = 497.63 / 15 = 33.1753 detik
Total Average = 0.5613 + 33.1753 = 33.7366 ≈ 33.74 detik
```

Verifikasi matematis menunjukkan konsistensi data NSM dengan *error* margin yang sangat kecil hanya dari pembulatan desimal. Data yang

konsisten ini penting untuk memastikan bahwa perbandingan dengan CNCSA dilakukan berdasarkan measurement yang akurat dan *reliable*.

3.1.21. Perbandingan Performa NSM dan CNCSA

Bagian ini menyajikan perbandingan kuantitatif antara CNCSA dan baseline NSM untuk menilai peningkatan performa yang dicapai melalui optimasi. Analisis dilakukan menggunakan lima metrik komparatif yang menilai improvement dari berbagai aspek, sehingga memberikan gambaran menyeluruh tentang manfaat praktis dari optimasi yang diterapkan pada CNCSA.

3.1.21.1. Persentase Peningkatan

Persentasi peningkatan digunakan untuk mengukur seberapa besar peningkatan performa, dinyatakan sebagai persentase pengurangan waktu eksekusi CNCSA dibandingkan NSM. Perhitungan dilakukan dengan menggunakan rumus berikut:

$$Improvement\% = \frac{T_{e_{NSM}} - T_{e_{CNCSA}}}{T_{e_{NSM}}} x100\% \qquad (37)$$

Hasil perhitungan peningkatan CNCSA:

```
Improvement% = (506.05 - 302.49) / 506.05 × 100%
Improvement% = 203.56 / 506.05 × 100%
Improvement% = 40.22%
```

Tabel 3.17. Persentase peningkatan CNCSA vs NSM

Metrik	NSM	CNCSA	Improvement / Keterangan
Total Execution Time (Te)	506,05 detik	302,49 detik	203,56 detik
Improvement Percentage	--	--	40,22%
Waktu yang Dihemat	--	--	3,39 menit
Target	--	--	Minimal > 20%
Status	--	--	Melampaui Target

Hasil perhitungan menunjukkan bahwa CNCSA mencapai peningkatan performa sebesar 40,22%, artinya CNCSA menyelesaikan tugas konfigurasi 40,22% lebih cepat dibanding NSM. Nilai ini melebihi target minimal 20% yang ditetapkan, menunjukkan bahwa optimasi yang diterapkan memberikan manfaat praktis yang nyata. Peningkatan 40,22% berarti dari total 506,05 detik yang

dibutuhkan NSM, CNCSA berhasil menghemat 203,56 detik atau sekitar 3,39 menit. Penghematan waktu ini cukup signifikan dalam konteks network automation, di mana administrator sering melakukan update konfigurasi secara berkala. Misalnya, jika administrator melakukan konfigurasi ulang 10 kali per bulan, total waktu yang dihemat mencapai 33,9 menit per bulan atau 6,8 jam per tahun. Persentase peningkatan sebesar 40% menunjukkan bahwa optimasi parameter koneksi dan pengiriman perintah memberikan dampak besar terhadap performa keseluruhan. Nilai ini juga menunjukkan adanya peluang untuk perbaikan lebih lanjut, misalnya melalui *concurrent execution* atau optimasi tambahan pada algoritma, meskipun 40% peningkatan sudah sangat baik untuk model eksekusi sequential.

3.1.21.2. Speedup Ratio

Speedup Ratio digunakan untuk mengukur seberapa banyak CNCSA lebih cepat dibanding NSM. Nilai ini dihitung dengan membandingkan total waktu eksekusi NSM dengan CNCSA menggunakan rumus berikut:

$$Speedup\ Ratio = \frac{T_{e_{NSM}}}{T_{e_{CNCSA}}} \qquad (38)$$

Rasio ini memberikan gambaran langsung tentang peningkatan kecepatan yang dicapai melalui optimasi CNCSA dibanding *baseline* NSM.

Tabel 3.18. Speedup Ratio CNCSA vs NSM

Metrik	Nilai
NSM Execution Time	506,05 detik
CNCSA Execution Time	302,49 detik
Speedup Ratio	1,67x
Interpretasi	CNCSA 67% lebih cepat

Hasil perhitungan menunjukkan *Speedup Ratio* sebesar 1,67x, artinya CNCSA 1,67 kali lebih cepat dibanding NSM, atau sekitar 67% lebih cepat dalam menyelesaikan tugas yang sama. Nilai Speedup Ratio lebih besar dari 1,0 menandakan adanya peningkatan performa, dan semakin tinggi nilainya, semakin besar percepatan yang dicapai. Dalam praktiknya, *Speedup Ratio* 1,67x berarti untuk tugas yang memakan waktu 506 detik (8,4 menit) dengan NSM, CNCSA hanya membutuhkan 302 detik (5,0 menit). Penghematan lebih dari 3 menit per eksekusi ini cukup signifikan dalam operasional sehari-hari, terutama saat administrator harus melakukan konfigurasi pada banyak router sekaligus atau melakukan rollback konfigurasi dengan cepat. Percepatan yang signifikan ini dicapai melalui kombinasi optimasi pada fase koneksi dan fase pengiriman perintah.

Meskipun koneksi hanya menyumbang sekitar 1,5% dari total waktu, optimasi pada fase ini tetap memberikan kontribusi. Kontribusi terbesar berasal dari optimasi fase pengiriman perintah, yang merupakan komponen dominan dalam keseluruhan proses konfigurasi.

3.1.21.3. Time Saved

Time Saved menunjukkan jumlah waktu yang berhasil dihemat secara langsung oleh CNCSA dibandingkan NSM. Perhitungan dilakukan dengan menggunakan rumus berikut:

$$\Delta T = T_{e_{NSM}} - T_{e_{CNCSA}} \quad (39)$$

Rumus ini memberikan nilai selisih waktu eksekusi, sehingga terlihat langsung berapa detik atau menit yang berhasil dihemat melalui optimasi CNCSA.

Hasil dari perhitungan *time saved*:

```
ΔT = 506.05 - 302.49
ΔT = 203.56 detik ≈ 3.39 menit
```

Tabel 3.19. *Time Saved* Per *Execution* router CHR

Metrik	Nilai
NSM Execution Time	506,05 detik (8,43 menit)
CNCSA Execution Time	302,49 detik (5,04 menit)
Time Saved (ΔT)	203,56 detik
Time Saved (menit)	3,39 menit
Persentase Penghematan	40,22%

Hasil perhitungan menunjukkan bahwa CNCSA menghemat 203,56 detik atau sekitar 3,39 menit per eksekusi dibandingkan NSM. Nilai ini menggambarkan manfaat nyata yang dapat dirasakan langsung oleh network administrator dalam kegiatan sehari-hari. Penghematan waktu bersifat kumulatif, sehingga semakin sering konfigurasi dilakukan, semakin besar total waktu yang dihemat. Manfaat praktis dari *Time Saved* ini cukup luas. Pertama, administrator dapat menyelesaikan lebih banyak tugas dalam waktu yang sama, sehingga produktivitas meningkat. Kedua, waktu yang dihemat dapat digunakan untuk pekerjaan lain yang lebih strategis, seperti perencanaan jaringan atau *troubleshooting* masalah kompleks. Ketiga, dalam situasi darurat atau *incident response*, kecepatan eksekusi yang lebih tinggi berarti waktu pemulihan (*recovery time*) juga lebih cepat.

3.1.21.4. Peningkatan Per Komponen Perangkat Router CHR

Analisis peningkatan per komponen membagi peningkatan performa total ke dalam fase-fase spesifik. Tujuannya adalah untuk melihat kontribusi masing-masing optimasi terhadap keseluruhan perbaikan waktu eksekusi, sehingga dapat diketahui fase mana yang memberikan dampak terbesar terhadap performa.

1) Connection Time Improvement

Connection Time Improvement mengukur persentase pengurangan waktu koneksi yang dicapai CNCSA dibanding NSM. Rumus yang digunakan:

$$ConnectionImprovement\% = \frac{T_{c_{NSM}} - T_{c_{CNCSA}}}{T_{c_{NSM}}} x100\% \quad (40)$$

Perhitungan *connection time improvement*:

```
Tc Improvement = (8.42 - 4.47) / 8.42 × 100%
Tc Improvement = 3.95 / 8.42 × 100%
Tc Improvement = 46.91%
```

Hasil ini menunjukkan bahwa CNCSA mampu mengurangi waktu koneksi sebesar 46,91% dibandingkan NSM, menandakan optimasi pada fase koneksi memberikan kontribusi signifikan terhadap percepatan total eksekusi.

2) Send Time Improvement

Send Time Improvement mengukur persentase pengurangan waktu pengiriman perintah yang dicapai CNCSA dibanding NSM. Rumus yang digunakan:

$$SendImprovement\% = \frac{T_{c_{NSM}} - T_{c_{CNCSA}}}{T_{c_{NSM}}} x100\% \quad (41)$$

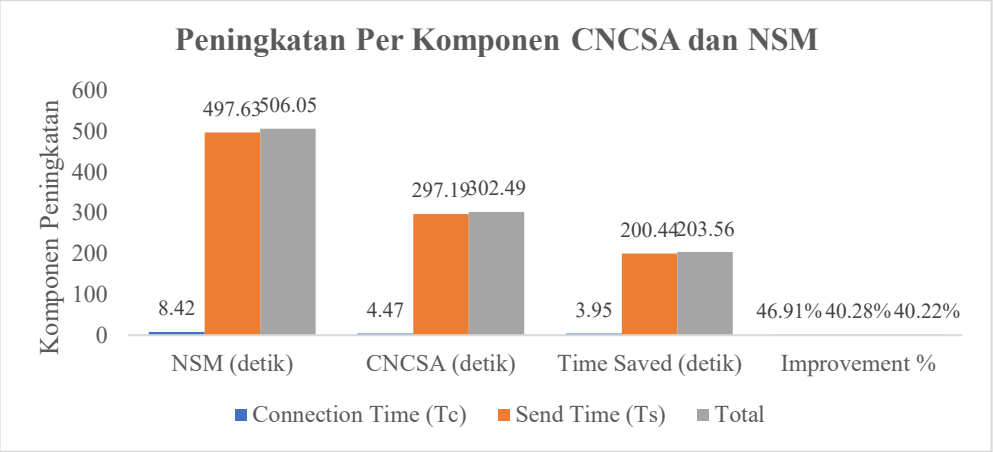
Perhitungan *send time improvement*:

```
Ts Improvement = (497.63 - 297.19) / 497.63 × 100%
Ts Improvement = 200.44 / 497.63 × 100%
Ts Improvement = 40.28%
```

Hasil ini menunjukkan bahwa CNCSA mampu mengurangi waktu pengiriman perintah sebesar 40,28% dibanding NSM, sehingga fase pengiriman perintah memberikan kontribusi besar terhadap percepatan total eksekusi.

Hasil analisis pada gambar 3.29 menunjukkan bahwa *Connection Time Improvement* mencapai 46,91%, lebih tinggi dari *improvement* total. Hal ini menunjukkan bahwa optimasi parameter koneksi

(*timeout* 8 detik, *session_timeout* 45 *detik*, *keepalive* 30 detik) sangat efektif dalam mengurangi overhead koneksi. Meskipun penghematan langsung hanya 3,95 detik, kontribusinya terhadap total waktu yang dihemat hanya sekitar 1,94%, karena koneksi merupakan bagian kecil dari total waktu eksekusi



Gambar 3.29. Peningkatan per Komponen Router CHR (CNCSA dan NSM)

Sementara itu, *Send Time Improvement* mencapai 40,28% dengan penghematan absolut 200,44 detik. Meskipun persentase improvement lebih rendah dibanding Connection Time, penghematan absolutnya jauh lebih besar karena *Send Time* merupakan komponen dominan dalam proses eksekusi. Dari total 203,56 detik yang dihemat, 200,44 detik atau 98,47% berasal dari optimasi *Send Time*. Hal ini menunjukkan bahwa pengaturan *delay_factor* dari 1,0 ke 0,5 memberikan dampak terbesar terhadap peningkatan performa keseluruhan.

Tabel 3.20. Kontribusi Terhadap *Total Time Saved*

Komponen	Time Saved (detik)	Kontribusi (%)
Connection Time	3,95	1,94%
Send Time	200,44	98,47%
Logging Time*	-0,83	-0,41%
Total	203,56	100,00%

Catatan: *Logging Time* NSM tidak diukur secara terpisah. Nilai negatif pada tabel menunjukkan bahwa CNCSA menambahkan pengukuran Tl secara eksplisit, sehingga muncul sedikit *overhead* tambahan. Analisis kontribusi menunjukkan bahwa meskipun optimasi pada *Connection Time* dan *Send Time* sama-sama efektif jika dilihat dari persentase, optimasi *Send Time* memberikan dampak praktis jauh lebih besar karena durasi fase ini lebih panjang dan dominan dalam total waktu

eksekusi. Temuan ini memberikan insight penting untuk strategi optimasi network automation, yaitu fokus pada komponen yang memiliki durasi terpanjang akan memberikan manfaat terbesar dibandingkan *effort* yang dikeluarkan. Dalam implementasi praktis, hasil ini menunjukkan bahwa jika sumber daya untuk optimasi terbatas, prioritas sebaiknya diberikan pada optimasi fase pengiriman perintah karena memberikan penghematan waktu terbesar. Namun, optimasi koneksi tetap penting karena memberikan peningkatan signifikan dengan usaha yang relatif kecil, hanya melalui perubahan parameter.

3.1.21.5. Analisis Kontribusi Optimasi CNCSA

Bagian ini membahas secara lebih rinci bagaimana setiap optimasi yang diterapkan dalam CNCSA berkontribusi terhadap peningkatan performa keseluruhan. Analisis ini membantu memahami efek praktis dari perubahan parameter dan fase optimasi yang dilakukan, serta memberikan panduan untuk prioritas optimasi pada *network automation*.

Tabel 3.21. Ringkasan Optimasi dan Dampaknya

Parameter	NSM (Default)	CNCSA (Optimized)	Dampak
timeout	100 detik	8 detik	Deteksi failure lebih cepat
session_timeout	60 detik	45 detik	Session handling lebih responsif
keepalive	None	30 detik	Stabilitas koneksi lebih baik
delay_factor	1.0	0.5	50% lebih cepat per command
max_loops	500	300	Timeout detection lebih cepat
strip_prompt	FALSE	TRUE	Parsing output lebih efisien
strip_command	FALSE	TRUE	Output lebih bersih
Tl measurement	Tidak ada	Eksplisit	Memberikan visibility terhadap bottleneck

Dampak kumulatif optimasi *delay_factor* dari 1,0 ke 0,5 memberikan dampak terbesar karena mengurangi jeda antar *command* menjadi setengahnya. Untuk 150 *command* per router, penghematan kumulatif sangat signifikan. Jika rata-rata jeda per command berkurang 50 milidetik, maka untuk 150 *command* penghematan mencapai 7,5 detik per router, atau 112,5 detik untuk 15 router. Nilai ini mendekati penghematan aktual 200 detik pada fase *Send Time*, dengan perbedaan akibat variasi *execution time per command*.

Optimasi *timeout* dan *session_timeout* mengurangi waktu tunggu untuk mendeteksi kegagalan koneksi atau *session timeout*. Meskipun dalam *environment* terkontrol *timeout* jarang terjadi, parameter yang lebih agresif ini tetap mengurangi *overhead initialization* dan *keep-alive checks*, berkontribusi pada penghematan

3,95 detik pada *Connection Time*. *Strip_prompt* dan *strip_command* mengurangi overhead parsing output dengan menghilangkan prompt router dan *echo command* dari output. Penghematan per *command* memang kecil, tetapi secara kumulatif untuk 2.250 *command* (15 router × 150 *command*) penghematan menjadi signifikan dan dapat diukur. Perbandingan menyeluruh antara CNCSA dan NSM menunjukkan bahwa CNCSA mencapai peningkatan performa yang signifikan dengan:

1) Improvement Percentage: 40,22%
2) Speedup Ratio: 1,67x
3) Time Saved: 3,39 menit per eksekusi

Analisis per komponen mengungkapkan bahwa optimasi *Send Time* memberikan kontribusi terbesar terhadap peningkatan total, meskipun optimasi *Connection Time* juga efektif jika dilihat dari persentase. Peningkatan performa ini dicapai tanpa mengurangi reliability, dengan kedua metode menunjukkan success rate 100%. Hal ini menegaskan bahwa optimasi parameter dapat meningkatkan kecepatan sambil tetap mempertahankan keandalan, asalkan parameter dipilih dengan tepat sesuai karakteristik *environment deployment*. Hasil perbandingan ini memvalidasi hipotesis penelitian bahwa optimasi sistematis pada parameter library Netmiko dapat meningkatkan performa network automation secara signifikan, dan memberikan bukti empiris bahwa pendekatan CNCSA dapat diterapkan secara praktis dalam environment produksi.

3.1.22. Interpretasi Hasil CNCSA

Hasil pengujian CNCSA menunjukkan pencapaian yang sangat baik di semua aspek performa. Total *Execution Time* sebesar 302,49 detik untuk mengkonfigurasi 15 router dengan 2.250 baris command menunjukkan bahwa algoritma mampu menangani *mass configuration* secara efisien. Rata-rata waktu 20,17 detik per router berarti setiap router dapat dikonfigurasi dalam waktu kurang dari setengah menit, sangat praktis untuk *deployment* operasional. Distribusi waktu eksekusi menunjukkan profil yang ideal, di mana 98,25% waktu digunakan untuk konfigurasi aktual, hanya 1,48% untuk overhead koneksi, dan 0,28% untuk logging. Profil ini menandakan bahwa algoritma berhasil meminimalkan waktu tidak produktif dan memaksimalkan waktu untuk pekerjaan bernilai tinggi. System *Overhead Ratio* yang sangat rendah (0,0112%) membuktikan implementasi dilakukan dengan efisien tanpa komponen berlebih yang menambah latensi.

Success Rate 100% dengan *zero error rate* menunjukkan bahwa optimasi parameter tidak mengurangi keandalan. Hal ini penting karena dalam *network automation*, kegagalan konfigurasi dapat menyebabkan *downtime* atau *misconfiguration* yang mengganggu layanan. Pencapaian *success rate* sempurna

menegaskan bahwa parameter yang dipilih tidak terlalu agresif dan masih memberikan margin cukup untuk variasi *response* time tiap router.

3.1.23. Faktor-Faktor yang Mempengaruhi Performa

Optimasi parameter Netmiko menjadi faktor utama yang meningkatkan performa CNCSA. *Delay_factor* dikurangi dari 1,0 menjadi 0,5, sehingga jeda antar command berkurang setengahnya. Dalam environment dengan network *latency* rendah dan *device* responsif seperti penelitian ini, *delay_factor* 0,5 masih memberi waktu cukup bagi device untuk memproses command dan mengirim output lengkap. Timeout dikurangi dari 100 detik menjadi 8 detik, mempercepat deteksi *connection failure* tanpa meningkatkan risiko *false timeout*. Dengan *latency* 0,5–2 ms di environment virtual, koneksi SSH biasanya *established* dalam 1–2 detik, sehingga *timeout* 8 detik masih memberikan margin yang aman. *Session_timeout* 45 detik cukup untuk menangani command yang lebih panjang sekaligus tetap responsif terhadap deteksi timeout. Keepalive 30 detik menjaga koneksi tetap aktif selama periode *idle*, mencegah router atau firewall menutup koneksi. *Max_loops* 300 yang lebih rendah dari *default* mempercepat deteksi command yang hang tanpa mengurangi kemampuan menangani output panjang. *Strip_prompt* dan *strip_command* mengurangi *overhead* parsing sehingga output lebih bersih.

Karakteristik *environment deployment* memengaruhi performa yang dicapai. Semua router dan server dalam penelitian ini berada di satu *host* Proxmox dengan virtual *bridge*, sehingga network *latency* sangat rendah (0,5–2 ms). Kondisi ini representatif untuk data center atau private cloud dengan koneksi LAN cepat. Untuk environment dengan *latency* lebih tinggi, misalnya WAN atau cloud public, performa absolut akan lebih lambat, tetapi improvement relatif kemungkinan tetap signifikan. Delay_factor 0,5 masih memberikan speedup, sementara timeout dan session_timeout mungkin perlu disesuaikan untuk mengakomodasi latency lebih tinggi. Spesifikasi *hardware* router juga mempengaruhi performa. MikroTik CHR dengan 1 vCPU dan 256 MB RAM merepresentasikan *device low-end*. Router dengan CPU dan RAM lebih besar bisa mengeksekusi command lebih cepat, sehingga *delay_factor* bisa dikurangi lebih jauh. Sebaliknya, untuk device dengan *resource* terbatas, *delay_factor* mungkin perlu sedikit ditingkatkan.

Kompleksitas command memengaruhi waktu eksekusi. *Firewall rules* yang digunakan dalam penelitian ini cepat dan predictable. *Command* yang lebih kompleks, misalnya konfigurasi *routing protocol* atau *scripting*, memiliki *execution time* lebih panjang dan variabilitas lebih tinggi. Total 150 *command* per router (50 *Mangle*, 50 NAT, 50 *Filter*) mewakili ukuran konfigurasi medium. Untuk konfigurasi lebih besar (500–1.000 *lines*), timing akan bertambah secara linear dengan jumlah command, tetapi improvement percentage kemungkinan tetap konsisten karena optimasi berlaku pada setiap command. *Command* seperti *firewall rules* memiliki waktu eksekusi konsisten saat dijalankan berulang. *Command* yang mengubah *state device*, seperti konfigurasi interface atau routing, bisa lebih

bervariasi tergantung kondisi *device*. Semua device di-reset ke *clean state* sebelum testing untuk memastikan hasil *reproducible*.

Logging Time sangat rendah (0,8335 detik untuk 15 *device*), menunjukkan database SQLite bekerja efisien. Faktor pendukung antara lain: *message size* limitation 5.000 karakter untuk mengurangi data per insert, penggunaan *direct create method* yang lebih efisien dibanding *save method*, dan database berada di *local disk*, menghindari network *overhead*. Jika deployment menggunakan database remote atau database lebih berat seperti PostgreSQL atau MySQL, Logging Time bisa lebih tinggi, terutama jika latency network ke server database tinggi. Optimasi seperti message *size limitation* dan bulk insert untuk *multiple devices* tetap bisa menjaga *overhead* minimal. Performa database juga dipengaruhi akses *concurrent*. Penelitian ini menggunakan sequential *execution*, sehingga hanya satu operasi write pada satu waktu. Untuk eksekusi concurrent di masa depan, perlu mempertimbangkan database *locking* dan transaksi management untuk mencegah *bottleneck* pada database layer.

3.1.24. Keterbatasan Penelitian

1) Keterbatasan Scope

Penelitian ini dilakukan pada 15 router MikroTik CHR dalam environment virtual. Hasil ini cukup representatif untuk beberapa kasus, tetapi mungkin berbeda jika diterapkan pada *physical devices*, vendor lain seperti Cisco, Juniper, atau Arista, atau environment dengan karakteristik berbeda. Pengujian pada lebih banyak device dan platform yang beragam akan memberikan validasi yang lebih menyeluruh. Command yang digunakan terbatas pada firewall rules yang cepat dan predictable. Untuk tipe command lain, misalnya routing configuration, interface configuration, atau script yang kompleks, waktu eksekusi bisa berbeda. Pengujian pada command yang lebih beragam akan memberi pemahaman lebih lengkap mengenai performa CNCSA di berbagai skenario. Environment testing yang digunakan sangat terkontrol dengan latency rendah dan resource dedicated, sehingga merepresentasikan best-case scenario. Dalam environment produksi dengan resource bersama, latency lebih tinggi, dan potensi masalah jaringan, performa bisa lebih bervariasi. Pengujian di kondisi mirip production akan memberikan data yang lebih realistis.

2) Keterbatasan Metodologi

Model sequential *execution* yang digunakan tidak mengeksplorasi parallelization. Eksekusi *concurrent* dapat memberikan *speedup* lebih besar, terutama untuk *deployment* skala besar dengan ratusan *device*. Namun, model *concurrent* menambah kompleksitas yang di luar scope penelitian ini, yang fokus pada optimasi sequential. Data yang disajikan berasal dari *single run*

testing, sehingga tidak menyediakan confidence intervals formal untuk variasi. Meski beberapa pengujian dilakukan untuk verifikasi, analisis statistik formal dengan multiple samples akan meningkatkan validitas dan generalisasi hasil. Perbandingan dilakukan hanya terhadap NSM sebagai baseline. Perbandingan dengan tools lain seperti Ansible, NAPALM, atau Nornir akan memberi konteks lebih luas mengenai posisi CNCSA dalam *landscape* network automation. Namun, perbandingan yang fair sulit karena perbedaan arsitektur, fitur, dan kemampuan.

3) Keterbatasan Teknis

Database SQLite yang digunakan optimal untuk *single-user sequential access*, tetapi bisa menjadi *bottleneck* untuk *concurrent writes* pada *deployment* skala besar. Migrasi ke database yang lebih kuat, misalnya PostgreSQL, diperlukan untuk production dengan pengguna *concurrent* atau eksekusi terdistribusi. *Overhead* pengukuran timing, meskipun sangat kecil (0,0112%), tetap tercakup dalam hasil. Untuk *micro-benchmarking* yang sangat presisi, overhead ini perlu diperhitungkan, tetapi untuk keperluan praktis, level ini negligible dan tidak mempengaruhi keputusan operasional. Cakupan error handling terbatas pada scenario umum seperti *connection timeout* dan *error* eksekusi *command*. *Edge cases*, misalnya partial *output corruption*, *inconsistent* device state, atau kondisi *error* kompleks, belum sepenuhnya diuji. Pengujian error handling yang lebih komprehensif dibutuhkan untuk implementasi *production-grade*.

BAB 4

KESIMPULAN DAN SARAN

4.1. Kesimpulan

Berdasarkan hasil penelitian dan pembahasan yang telah dilakukan terhadap implementasi algoritma *Centralized Network Configuration Sequential Algorithm* (CNCSA) untuk optimasi konfigurasi jaringan router CHR, dapat ditarik beberapa kesimpulan sebagai berikut:

CNCSA berhasil mencapai peningkatan performa sebesar 40.22 persen dibandingkan dengan Standard Netmiko Sequential *Method* (NSM) sebagai baseline. Dalam pengujian terhadap 15 router MikroTik CHR dengan 150 command per router, CNCSA menyelesaikan konfigurasi dalam waktu 302.49 detik, jauh lebih cepat dibandingkan NSM yang memerlukan 506.05 detik. Speedup ratio mencapai 1.67x yang berarti CNCSA 67 persen lebih cepat dalam menyelesaikan tugas yang sama. *Time saved* sebesar 203.56 detik atau 3.39 menit per eksekusi memberikan manfaat praktis yang signifikan, terutama untuk penulis yang melakukan konfigurasi secara regular.

Analisis efisiensi menunjukkan bahwa CNCSA memiliki profil performa yang sangat baik dengan *Connection Efficiency* 1.48 persen, *Configuration Efficiency* 98.25 persen, *Logging Efficiency* 0.28 persen, dan CNCSA *Execution Efficiency* 98.52 persen. Semua metrik memenuhi atau melampaui target yang ditetapkan, mengkonfirmasi bahwa algoritma berhasil memaksimalkan waktu untuk pekerjaan produktif dan meminimalkan *overhead* tidak produktif. System *Overhead Ratio* hanya 0.0111 persen menunjukkan implementasi yang sangat efisien tanpa komponen berlebih yang menambah latensi.

Success rate 100 persen dengan *zero error rate* membuktikan bahwa optimasi parameter yang diterapkan dalam CNCSA tidak mengorbankan keandalan sistem. Semua 15 router berhasil dikonfigurasi tanpa kegagalan koneksi atau *error* eksekusi, menunjukkan bahwa parameter yang dipilih tidak terlalu agresif dan masih memberikan margin yang cukup untuk variability dalam response time router. Pencapaian ini sangat penting karena memvalidasi bahwa peningkatan kecepatan dapat dicapai sambil mempertahankan reliability yang diperlukan untuk deployment operasional.

Analisis per komponen mengungkapkan bahwa optimasi *Send Time* memberikan kontribusi terbesar terhadap peningkatan performa keseluruhan dengan *improvement* 40.28 persen dan *time saved* 200.44 detik, menyumbang 98.47 persen dari total penghematan waktu. *Connection Time improvement* mencapai 46.91 persen dengan time saved 3.95 detik. Meskipun persentase improvement koneksi lebih tinggi, dampak praktis terbesar berasal dari optimasi fase pengiriman *command* karena komponen ini mendominasi total waktu eksekusi. Temuan ini

memberikan insight penting bahwa fokus optimasi pada komponen dengan durasi terpanjang memberikan return on effort yang paling tinggi.

Verifikasi matematis menunjukkan konsistensi sempurna antara berbagai formula dengan error margin mendekati nol yang hanya disebabkan pembulatan desimal. Formula Te = Tc + Ts + Tl, formula ε = Elapsed_Time - Te, dan berbagai efficiency ratios telah terverifikasi dengan data empiris, mengkonfirmasi bahwa model matematis yang dirumuskan valid dan *applicable* untuk analisis performa network automation. Konsistensi ini juga memvalidasi integritas sistem pengukuran timing yang diimplementasikan dalam algoritma.

CNCSA terbukti applicable untuk deployment praktis dalam production environment. Karakteristik seperti ease of implementation menggunakan Python dan Django framework, *flexibility* dalam parameter tuning untuk adaptasi ke berbagai *environment*, *comprehensive metrics* untuk monitoring dan troubleshooting, dan web-based interface untuk *user-f*riendly operation membuat CNCSA ready untuk operational deployment. Time saved yang signifikan dan reliability yang terjaga memberikan *value proposition* yang jelas untuk adoption dalam skenario *real-world network automation*.

Kebaruan penelitian ini terletak pada pendekatan sistematis dalam mengoptimasi parameter library Netmiko yang dikombinasikan dengan pengukuran timing granular untuk visibility komprehensif. Berbeda dengan penggunaan Netmiko secara konvensional yang mengandalkan parameter default, CNCSA menerapkan optimasi berbasis empirical testing terhadap parameter-parameter kunci seperti *timeout, delay_factor, max_loops*, dan lainnya. Pengukuran terpisah untuk Tc, Ts, dan Tl memberikan *insight* yang tidak tersedia dalam pendekatan standard, memungkinkan identifikasi *bottleneck* dan evaluasi dampak optimasi secara precise.

4.2. Saran

Berdasarkan hasil penelitian dan limitasi yang telah diidentifikasi, beberapa saran untuk pengembangan lebih lanjut dan implementasi praktis adalah sebagai berikut:

Untuk meningkatkan skalabilitas CNCSA dalam menangani proses konfigurasi pada ratusan hingga ribuan perangkat, penelitian selanjutnya disarankan untuk menerapkan model eksekusi konkuren. Model ini memungkinkan beberapa perangkat diproses secara paralel dalam satu waktu. Pendekatan concurrency dapat diimplementasikan menggunakan mekanisme threading sehingga proses paralel dapat berjalan dengan tetap menjaga struktur kode yang terkelola dengan baik. Penelitian lanjutan perlu mengkaji tingkat concurrency yang optimal, strategi pengelolaan sumber daya sistem, serta mekanisme penanganan kesalahan yang sesuai pada eksekusi paralel. Penerapan model konkuren berpotensi

memberikan peningkatan kecepatan yang signifikan sebagai pelengkap dari optimasi yang telah dicapai melalui eksekusi sekuensial.

Selain itu, pengembangan sistem penyesuaian parameter secara adaptif juga direkomendasikan untuk meningkatkan ketahanan CNCSA pada lingkungan jaringan yang heterogen. Sistem ini dirancang untuk menyesuaikan parameter konfigurasi secara otomatis berdasarkan kondisi jaringan dan karakteristik perangkat yang diamati selama proses eksekusi. Algoritma pembelajaran mesin dapat dimanfaatkan untuk mempelajari parameter optimal dari data historis dan melakukan penyesuaian secara dinamis. Sistem adaptif dapat diawali dengan parameter yang bersifat konservatif guna menjaga keandalan, kemudian secara bertahap melakukan optimasi berdasarkan tingkat keberhasilan dan pengamatan waktu eksekusi. Penelitian lanjutan perlu mengkaji metrik yang tepat untuk proses optimasi, keseimbangan antara agresivitas dan keamanan, serta mekanisme untuk mencegah penyesuaian berlebihan terhadap kondisi tertentu.

Untuk memperluas penerapan CNCSA pada lingkungan multi-vendor, disarankan agar sistem dikembangkan agar mendukung platform jaringan lain seperti Cisco IOS, Juniper JunOS, Arista EOS, dan vendor utama lainnya. Pengujian lintas platform diperlukan untuk memvalidasi tingkat generalisasi pendekatan CNCSA serta mengidentifikasi kebutuhan optimasi spesifik pada masing-masing platform. Pengembangan kerangka kerja terpadu yang mampu menangani berbagai platform dengan parameter yang dioptimalkan secara khusus akan sangat bermanfaat bagi lingkungan enterprise dengan infrastruktur yang beragam. Penelitian lanjutan juga perlu memperhatikan tantangan spesifik platform, seperti perbedaan perilaku antarmuka baris perintah dan kebutuhan pemrosesan keluaran konfigurasi.

Dalam konteks penerapan pada lingkungan produksi dengan jumlah pengguna bersamaan dan skala operasi yang besar, disarankan untuk melakukan migrasi basis data dari SQLite ke sistem basis data yang lebih andal, seperti PostgreSQL atau MySQL. Teknik optimasi basis data, termasuk connection pooling, prepared statements, dan mekanisme bulk insert, perlu diterapkan untuk mengurangi overhead akses data. Untuk skala implementasi yang sangat besar, penggunaan basis data terdistribusi atau solusi NoSQL seperti MongoDB dapat dipertimbangkan guna mendukung skalabilitas horizontal. Penelitian lanjutan perlu mengevaluasi trade-off antara kompleksitas sistem, kinerja, dan kebutuhan operasional dari berbagai pilihan basis data tersebut.

DAFTAR PUSTAKA

1. H. A. Damanik, M. Anggraeni, and F. A. A. Nusantari, Buku Network Automation Pada Mikrotik RouterOS. 2024. Accessed: Sep. 15, 2025. [Online]. Available: https://deepublishstore.com/produk/buku-network-automation-pada-mikrotik-routeros-menggunakan-paramiko-dan-netmiko-python/
2. W. Prabowo and R. Ariyanto, "Implementation of Python-Based Network Automation Technology for Computer Infrastructure Optimization," Impact : Multidisciplinary Journal, vol. 1, no. 01, pp. 34–40, Jul. 2025, Accessed: Jan. 01, 2026. [Online]. Available: https://journal.takaza.id/index.php/impact/article/view/135
3. S. Jorepalli, "Leveraging Network Automation with Python, Terraform, and Ansible to Enhance Security and Operational Efficiency in Large-Scale Networks," International Journal of Intelligent Systems and Applications in Engineering, vol. 12, no. 23s, pp. 2009–2016, Dec. 2024, Accessed: Jan. 01, 2026. [Online]. Available: https://ijisae.org/index.php/IJISAE/article/view/7222
4. A. Datta, A. T. M. A. Imran, and C. Biswas, "Network Automation: Enhancing Operational Efficiency Across the Network Environment," ICRRD Journal, vol. 4, no. 1, pp. 101–111, Mar. 2023, Accessed: Jan. 01, 2026. [Online]. Available: https://icrrd.com/article/315/network-automation-enhancing-operational-efficiency-across-the-network-environment
5. R. May, C. Biermann, J. Krüger, and T. Leich, "Asking Security Practitioners: Did You Find the Vulnerable (Mis) Configuration?," in Proceedings of the 19th International Working Conference on Variability Modelling of Software-Intensive Systems, in VaMoS '25. New York, NY, USA: Association for Computing Machinery, May 2025, pp. 30–39. doi: 10.1145/3715340.3715439.
6. H. A. Damanik and M. Anggraeni, "Manajemen Jaringan Terpusat untuk Konfigurasi dan Otomatisasi Pemulihan menggunakan Paramiko dan Django Framework," Jurnal Teknik Informatika dan Sistem Informasi, vol. 11, no. 3, pp. 369–382, Dec. 2025, doi: 10.28932/jutisi.v11i3.11431.
7. J. Åkerberg et al., "Future Industrial Networks in Process Automation: Goals, Challenges, and Future Directions," Applied Sciences, vol. 11, no. 8, p. 3345, Jan. 2021, doi: 10.3390/app11083345.
8. M. El Rajab, L. Yang, and A. Shami, "Zero-touch networks: Towards next-generation network automation," Computer Networks, vol. 243, p. 110294, Apr. 2024, doi: 10.1016/j.comnet.2024.110294.
9. K. Mahant and J. P. Kalapparambath, "Exploring the Role of Network Automation in Enterprise Network Scalability and Efficiency," Educational Administration: Theory and Practice, vol. 30, no. 11, 2024, doi: 10.53555/kuey.v30i11.9561.

10. D. Bringhenti, G. Marchetto, R. Sisto, and F. Valenza, “Automation for Network Security Configuration: State of the Art and Research Trends,” ACM Comput. Surv., vol. 56, no. 3, p. 57:1-57:37, Oct. 2023, doi: 10.1145/3616401.
11. F. Minna, “On the Automatic Detection and Remediation of Misconfigurations and Vulnerabilities in Cloud Ecosystems,” PhD-Thesis - Research and graduation internal, 2025. doi: 10.5463/thesis.1208.
12. M. Alicea and I. Alsmadi, “Misconfiguration in Firewalls and Network Access Controls: Literature Review,” Future Internet, vol. 13, no. 11, p. 283, Nov. 2021, doi: 10.3390/fi13110283.
13. B. S. Mitchell, S. Mancoridis, and J. Kashyap, “On the Automatic Identification of Misconfiguration Errors in Cloud Native Systems,” in Proceedings of the 2024 7th Artificial Intelligence and Cloud Computing Conference, in AICCC ’24. New York, NY, USA: Association for Computing Machinery, Jul. 2025, pp. 539–548. doi: 10.1145/3719384.3719463.
14. S. R. Thati, “How Configuration Automation Reduced Compliance Violations in a Global Enterprise,” European Journal of Computer Science and Information Technology, vol. 13, no. 21, May 2025, Accessed: Jan. 01, 2026. [Online]. Available: https://eajournals.org/ejcsit/vol13-issue21-2025/how-configuration-automation-reduced-compliance-violations-in-a-global-enterprise/
15. A. Elezi and D. A. Karras, “On Detailed Network Systems Configuration Management Automation using Python,” WSEAS Transactions on Communications, vol. 22, pp. 1–16, 2023, doi: 10.37394/23204.2023.22.1.
16. M. Bajpai, “Automating Network Device Configuration and Compliance Enforcement,” Jun. 06, 2019, Social Science Research Network, Rochester, NY: 5057509. doi: 10.2139/ssrn.5057509.
17. F. Osei-Wusu, W. Asiedu, K. F. Asamoah, S. A. Muntaka, D. Yeboah, and E. A. Sarfo, “Automating Network Programmability and Backup on Cisco Devices Using Python and Netmiko Library: A Case Study of Komfo Anokye Teaching Hospital LAN,” Journal of Computing Research and Innovation, vol. 10, no. 1, pp. 227–242, Mar. 2025, doi: 10.24191/jcrinn.v10i1.488.
18. A. B. Imoukhuede, T. R. Sheltami, A. H. Mahmoud, and A. Y. Barnawi, “Optimization of network device hardening in a multivendor environment,” Sci Rep, vol. 15, no. 1, p. 15042, Apr. 2025, doi: 10.1038/s41598-025-97894-4.
19. B. Singh, S. Prabhat, and A. Anand, “Enhancing Network Performance with Automation: A Case Study of Large-Scale Enterprises,” Mar. 01, 2024, Social Science Research Network, Rochester, NY: 5278034. doi: 10.2139/ssrn.5278034.
20. A. M. Mazin, R. A. Rahman, M. Kassim, and A. R. Mahmud, “Performance Analysis on Network Automation Interaction with Network Devices Using Python,” in 2021 IEEE 11th IEEE Symposium on Computer Applications

& Industrial Electronics (ISCAIE), Apr. 2021, pp. 360–366. doi: 10.1109/ISCAIE51753.2021.9431823.

LAMPIRAN

Lampiran 1. Realisasi Penggunaan Anggaran

Dana disetujui: Rp.12.000.000

Jenis Pembelajaan	Komponen	Item	Kuantitas	Biaya Satuan	Total
Belanja Bahan	ATK	-	-	-	-
Belanja Bahan	Bahan penelitian (habis pakai)	Materai dan ATK	1	Rp250.000	Rp250.000
Pengumpulan Data	Honor pembantu peneliti	Kabel STP dan RJ45	30	Rp63.000	Rp1.890.000
Pengumpulan Data	FGD	Honorium pembantu peneliti	2	Rp500.000	Rp1.000.000
Pengumpulan Data	Transport	Transport Dosen dan Mahasiswa ke AJN (4 orang) kali 5	20	Rp55.000	Rp1.100.000
Pengumpulan Data	Konsumsi	Konsumsi Rapat dan Kordinasi 4 Orang 5 Kali	20	Rp50.000	Rp1.000.000
Pengumpulan Data	Penginapan	-	-	-	-

Analisis Data	Honor pengolah data	Pembuatan sistem, Konfigurasi dan Implementasi Sistem	1	Rp3.000.000	Rp3.000.000
Analisis Data	Honor narasumber	-	-	-	-
Sewa Peralatan	Peralatan penelitian	Sewa Peralatan Server (1 Unit) , Router dan Switch	2	Rp2500.000	Rp5.000.000
Pelaporan penelitian	Honor administrasi peneliti	Honorium Peneliti	2	Rp600.000	Rp1.200.000
Lainnya	Biaya pendaftaran HKI	Pendaftaran Hak Cipta	1	Rp500.000	Rp500.000

Lampiran 2. Surat Perjanjian Kontrak Penelitian

UNIVERSITAS BUDI LUHUR
Kampus Pusat : Jl. Raya Ciledug - Petukangan Utara - Jakarta Selatan 12260
Telp : 021-5853753 (hunting), Fax : 021-5853489, http://www.budiluhur.ac.id
FAKULTAS TEKNOLOGI INFORMASI
FAKULTAS EKONOMI DAN BISNIS
FAKULTAS ILMU SOSIAL DAN STUDI GLOBAL
FAKULTAS TEKNIK
FAKULTAS KOMUNIKASI DAN DESAIN KREATIF

SURAT PERJANJIAN KONTRAK PENELITIAN
Nomor A/UBL/DRPM/000/208/11/25

Pada hari ini, Rabu 05 November 2025 Semester Gasal Tahun Ajaran 2025/2026, kami yang bertandatangan di bawah ini:

1. **Prof. Dr. Ir. Prudensius Maring, M.A.**, selaku Direktur Riset dan Pengabdian Kepada Masyarakat Universitas Budi Luhur, selanjutnya disebut PIHAK PERTAMA.
2. **Hillman Akhyar Damanik, S.Kom., M.Kom.**, selaku Peneliti selanjutnya disebut PIHAK KEDUA.

Kedua belah pihak menyatakan bersepakat untuk membuat perjanjian kontrak penelitian sebagai berikut:

Pasal 1
Judul Penelitian

PIHAK PERTAMA dalam jabatannya tersebut di atas, memberikan tugas kepada PIHAK KEDUA untuk melaksanakan penelitian yang berjudul: CNCSA: Novel Sequential Algorithm untuk Peningkatan Efisiensi Konfigurasi Jaringan Berbasis Netmiko pada Infrastruktur Virtual.

Pasal 2
Personalia Penelitian

Peneliti Utama : Hillman Akhyar Damanik, S.Kom., M.Kom.
Anggota Peneliti : Merry Anggraeni, S.Kom., M.Kom.

Pasal 3
Waktu dan Biaya Penelitian

1. Waktu penelitian adalah 5 bulan, terhitung sejak tanggal 08 September 2025 sampai dengan 08 Februari 2026.
2. Biaya pelaksanaan penelitian ini dibebankan pada Yayasan Pendidikan Budi Luhur Cakti Tahun 2025 dengan nilai kontrak sebesar Rp 12,000,000.00 (dua belas juta rupiah)

Pasal 4
Cara Pembayaran

Pembayaran biaya penelitian diberikan secara bertahap, sebagai berikut:

1. Tahap pertama sebesar 50% dari nilai kontrak, setelah surat perjanjian kontrak penelitian ini ditandatangani oleh kedua belah pihak.
2. Tahap kedua sebesar 50% dari nilai kontrak, setelah PIHAK KEDUA menyerahkan Laporan Hasil Penelitian kepada PIHAK PERTAMA.

Pasal 5
Keaslian Penelitian dan Ketidakterikatan dengan Pihak Lain

1. PIHAK KEDUA bertanggungjawab atas keaslian judul penelitian sebagaimana disebutkan dalam Pasal 1 Surat Perjanjian Kontrak Penelitian ini (bukan duplikat/jiplakan/plagiat) dari penelitian orang lain.
2. PIHAK KEDUA menjamin bahwa judul penelitian tersebut bebas dari ikatan dengan pihak lain atau tidak sedang didanai oleh pihak lain.

SURAT PERJANJIAN KONTRAK PENELITIAN
Nomor A/UBL/DRPM/000/208/11/25

Pada hari ini, Rabu 05 November 2025 Semester Gasal Tahun Ajaran 2025/2026, kami yang bertandatangan di bawah ini:

1. **Prof. Dr. Ir. Prudensius Maring, M.A.**, selaku Direktur Riset dan Pengabdian Kepada Masyarakat Universitas Budi Luhur, selanjutnya disebut PIHAK PERTAMA.
2. **Hillman Akhyar Damanik, S.Kom., M.Kom.**, selaku Peneliti selanjutnya disebut PIHAK KEDUA.

Kedua belah pihak menyatakan bersepakat untuk membuat perjanjian kontrak penelitian sebagai berikut:

Pasal 1
Judul Penelitian

PIHAK PERTAMA dalam jabatannya tersebut di atas, memberikan tugas kepada PIHAK KEDUA untuk melaksanakan penelitian yang berjudul: CNCSA: Novel Sequential Algorithm untuk Peningkatan Efisiensi Konfigurasi Jaringan Berbasis Netmiko pada Infrastruktur Virtual.

Pasal 2
Personalia Penelitian

Peneliti Utama : Hillman Akhyar Damanik, S.Kom., M.Kom.
Anggota Peneliti : Merry Anggraeni, S.Kom., M.Kom.

Pasal 3
Waktu dan Biaya Penelitian

1. Waktu penelitian adalah 5 bulan, terhitung sejak tanggal 08 September 2025 sampai dengan 08 Februari 2026.
2. Biaya pelaksanaan penelitian ini dibebankan pada Yayasan Pendidikan Budi Luhur Cakti Tahun 2025 dengan nilai kontrak sebesar Rp 12,000,000.00 (dua belas juta rupiah)

Pasal 4
Cara Pembayaran

Pembayaran biaya penelitian diberikan secara bertahap, sebagai berikut:

1. Tahap pertama sebesar 50% dari nilai kontrak, setelah surat perjanjian kontrak penelitian ini ditandatangani oleh kedua belah pihak.
2. Tahap kedua sebesar 50% dari nilai kontrak, setelah PIHAK KEDUA menyerahkan Laporan Hasil Penelitian kepada PIHAK PERTAMA.

Pasal 5
Keaslian Penelitian dan Ketidakterikatan dengan Pihak Lain

1. PIHAK KEDUA bertanggungjawab atas keaslian judul penelitian sebagaimana disebutkan dalam Pasal 1 Surat Perjanjian Kontrak Penelitian ini (bukan duplikat/jiplakan/plagiat) dari penelitian orang lain.
2. PIHAK KEDUA menjamin bahwa judul penelitian tersebut bebas dari ikatan dengan pihak lain atau tidak sedang didanai oleh pihak lain.

3. PIHAK KEDUA menjamin bahwa judul penelitian tersebut bukan merupakan penelitian yang SEDANG ATAU SUDAH selesai dikerjakan, baik didanai oleh pihak lain maupun oleh sendiri.
4. PIHAK PERTAMA tidak bertanggungjawab terhadap tindakan plagiat yang dilakukan oleh PIHAK KEDUA.
5. Apabila dikemudian hari diketahui ketidakbenaran pernyataan ini, maka kontrak penelitian DINYATAKAN BATAL, dan PIHAK KEDUA wajib mengembalikan dana yang telah diterima kepada Yayasan Pendidikan Budi Luhur Cakti sebagai pemberi dana.

Pasal 6
Monitoring Penelitian

1. PIHAK PERTAMA berhak untuk:
   a. Melakukan pengawasan administrasi, monitoring, dan evaluasi terhadap pelaksanaan penelitian.
   b. Memberikan sanksi jika dalam pelaksanaan penelitian terjadi pelanggaran terhadap isi perjanjian oleh peneliti.
   c. Bentuk sanksi disesuaikan dengan tingkat pelanggaran yang dilakukan.
2. Pemantauan kemajuan penelitian dikoordinasikan oleh PIHAK PERTAMA.
3. Pelaksanaan kemajuan penelitian dilaksanakan pada tanggal 13 Desember 2025.
4. Format Laporan Kemajuan dan teknis pelaksanaannya diatur oleh PIHAK PERTAMA.

Pasal 7
Laporan Akhir Penelitian

PIHAK KEDUA wajib menyerahkan laporan akhir dalam bentuk softcopy, paling lambat tanggal 07 Februari 2026.

Pasal 8
Sanksi

Segala kelalaian baik disengaja maupun tidak, sehingga menyebabkan keterlambatan menyerahkan laporan hasil penelitian dengan batas waktu yang telah ditentukan akan mendapatkan sanksi sebagai berikut:

1. Tidak diperbolehkan mengajukan usulan penelitian pada semester berikutnya bagi ketua dan anggota peneliti.
2. PIHAK KEDUA diberikan kesempatan perpanjangan waktu penelitian selama 2 (dua) minggu sampai dengan tanggal 21 Februari 2026.
3. Jika setelah masa perpanjangan tersebut PIHAK KEDUA tidak dapat menyelesaikan penelitiannya, PIHAK KEDUA diwajibkan mengembalikan dana yang sudah diterima kepada Yayasan Pendidikan Budi Luhur Cakti dengan cara mengembalikan tunai kepada PIHAK PERTAMA.

UNIVERSITAS BUDI LUHUR

Kampus Pusat : Jl. Raya Ciledug - Petukangan Utara - Jakarta Selatan 12260
Telp : 021-5853753 (hunting), Fax : 021-5853489, http://www.budiluhur.ac.id

FAKULTAS TEKNOLOGI INFORMASI
FAKULTAS EKONOMI DAN BISNIS
FAKULTAS ILMU SOSIAL DAN STUDI GLOBAL
FAKULTAS TEKNIK
FAKULTAS KOMUNIKASI DAN DESAIN KREATIF

Pasal 9
Penutup

Perjanjian ini berlaku sejak ditandatangani dan disetujui oleh PIHAK PERTAMA dan PIHAK KEDUA.

Jakarta, 05 November 2025

PIHAK PERTAMA	PIHAK KEDUA
Prof. Dr. Ir. Prudensius Maring, M.A.	Hilman Akhyar Damanik, S.Kom., M.Kom.
NIP. 190043	NIP. 190016

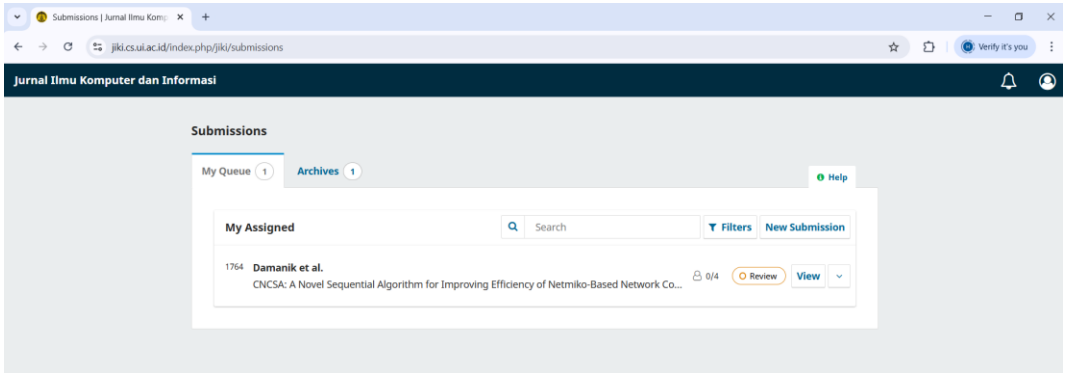
Lampiran 3. Catatan Harian

No	Tanggal	Kegiatan
1.	26/September/2026	Catatan : Pembuatan Draft Proposal Penelitian
2.	10/Oktober/2026	Catatan : Tujuan, Rumusan Permasalahan, kajian pustaka standard Netmiko dan identifikasi gap penelitian, Penetapan parameter dan perumusan metrik NSM dan CNCSA
3.	07/November/2025	Catatan : Konfigurasi Physic Arsitektur dan Topologi Jaringan, Setup Hypervisor Proxmox untuk virtual Server (vServer) NSM dan CNCSA Konfigurasi 15 CHR MikroTik RB941-2nD (IP Address), Desain Arsitektur dan Topologi Jaringan dan template konfigurasi
4.	14/November/2025	Catatan: Implementasi Konfigurasi Jaringan Terpusat Berbasis Web NSM, Setup Database SqLite & Django Framework dan Development NSM (10.151.20.147:8000)
5.	21/November/2025	Catatan: Pengujian NSM: 15 routers × 150 commands configuration Pengukuran NSM: T_c (Connection Time), T_s(Send Time) Pencatatan Success Rate & Error Rate
6.	28/November/2025	Catatan: Perancangan dan Implementasi Algoritma Sequential (CNCSA) terpusat Berbasis Web CNCSA (10.151.20.190:8000)
7.	05/Desember/2025	Catatan: Penerapan Database Logging, Penyesuaian Parameter Penerapan Algoritma untuk 10 metrik efesiensi.

		Pengujian CNCSA dan pengukuran metrik sequential
8.	08/Desember/2025	Catatan: Perumusan dan validasi model matematis evaluasi kinerja NSM dan CNCSA serta pembuatan laporan Kemajuan Penelitian
9.	13/Desember/2025	Catatan: Submit Laporan Kemajuan Penelitian
10.	19/Desember/2025	Catatan: Validasi hasil pengujian NSM, CNCSA dan dokumentasi dataset
11.	02/Januari/2026	Catatan: Penyusunan laporan akhir, artikel jurnal, pendaftaran KI dan penyusunan Buku

Lampiran 4. Artikel Ilmiah (Submitted)

Jurnal Ilmu Komputer dan Informasi is Accredited by the Ministry for Research, Technology and Higher Education (RISTEKDIKTI)(No:60/E/KPT/2016).



Lampiran 5. HKI

Lampiran I

Peraturan Menteri Kehakiman R.I.

Nomor : M.01-HC.03.01 Tahun 1987

Kepada Yth. :

Direktur Jenderal HKI

melalui Direktur Hak Cipta,

Desain Industri, Desain Tata Letak,

Sirkuit Terpadu dan Rahasia Dagang

di

Jakarta

PERMOHONAN PENDAFTARAN CIPTAAN

I. Pencipta:
1. Nama : Hillman Akhyar Damanik
2. Kewarganegaraan : Indonesia
3. Alamat : Flamboyan Unit No.57 Jl. Mesjid Nurul Iman Cipadu - Tangerang 15155
4. Telepon :
5. No. HP & E-mail : 082299099294 & hillman.akhyardamanik@budiluhur.ac.id

Pencipta:
1. Nama : Merry Anggraeni
2. Kewarganegaraan : Indonesia
3. Alamat : Jl. Swadarma Utara 4 No.62 Ulujami-Pesanggrahan
4. Telepon :
5. No. HP & E-mail : 081316177827 & merry.anggraeni@budiluhur.ac.id

II. Pemegang Hak Cipta:

1 Nama : DRPM Universitas Budi Luhur
2. Kewarganegaraan : Indonesia
3. Alamat : Jl. Raya Ciledug, Petukangan Utara, Pesanggrahan Jakarta, 12260
4. Telepon : 021 - 5853753
5. No. HP & E-mail : hki@budiluhur.ac.id

IV. Jenis dari judul ciptaan yang dimohonkan: CNCSA: Novel Sequential Algorithm untuk Peningkatan Efisiensi Konfigurasi Jaringan Berbasis Netmiko pada Infrastruktur Virtual.

V. Tanggal dan tempat diumumkan untuk pertama kali di wilayah Indonesia atau di luar wilayah Indonesia:

VI. Uraian ciptaan:
Automasi konfigurasi jaringan menjadi kebutuhan penting dalam pengelolaan infrastruktur jaringan modern yang semakin kompleks. Penelitian ini mengusulkan Centralized Network *Configuration Sequential Algorithm* (CNCSA) untuk mengoptimasi proses konfigurasi *mass deployment* pada router MikroTik melalui pendekatan sistematis dalam optimasi parameter *library* Netmiko. Metode penelitian menggunakan eksperimental dengan pengujian terhadap 15 router MikroTik *Cloud Hosted Router* versi 7.16 yang masing-masing dikonfigurasi dengan 150 baris perintah *firewall*. CNCSA menerapkan optimasi pada parameter koneksi (*timeout* 8s, *session_timeout* 45s, *keepalive* 30s) dan parameter pengiriman *command* (*delay_factor* 0.5, *max_loops* 300, *strip_prompt* dan *strip_command* aktif). Pengukuran *timing* granular dilakukan untuk tiga komponen utama yaitu *Connection Time*, *Send Time*, dan *Logging Time* dengan formula Te = Tc + Ts + Tl. Hasil pengujian menunjukkan CNCSA mencapai peningkatan performa 40.22 persen dengan Total Execution Time 302.49 detik dibandingkan *Standard* Netmiko *Sequential Method* yang memerlukan 506.05 detik. *Speedup* ratio mencapai 1.67x dengan *time saved* 203.56 detik per eksekusi. Analisis efisiensi menunjukkan *Configuration Efficiency* 98.25 persen, *Connection Efficiency* 1.48 persen, *Logging Efficiency* 0.28 persen, dan *System Overhead* Ratio hanya 0.0112 persen. Success rate 100 persen memvalidasi bahwa optimasi tidak mengorbankan keandalan. CNCSA terbukti *applicable* untuk *production* environment dengan manfaat praktis yang signifikan terutama untuk administrator yang melakukan konfigurasi secara regular.

Jakarta, 24 Januari 2026

Hillman Akhyar Damanik

Merry Anggraeni

SURAT PENGALIHAN HAK CIPTA

Yang bertanda tangan dibawah ini:

1. Nama	: Hillman Akhyar Damanik
2. Kewarganegaraan	: Indonesia
3. Alamat	: Flamboyan Unit No.57 Jl. Mesjid Nurul Iman Cipadu - Tangerang 15155
4. Telepon	:
5. No. HP & E-mail	: 082299099294 & hilladamanik@gmail.com

1. Nama	: Merry Anggraeni
2. Kewarganegaraan	: Indonesia
3. Alamat	: Jl. Swadarma Utara 4 No.62 Ulujami-Pesanggrahan
4. Telepon	:
5. No. HP & E-mail	: 081316177827 & merry.anggraeni@budiluhur.ac.id

Adalah Pihak I selaku pencipta, dengan ini menyerahkan karya ciptaan saya kepada:

Nama	: DRPM Universitas Budi Luhur
Alamat	: Jl. Raya Ciledug, Petukangan Utara, Pesanggrahan, Jakarta 1220

Adalah Pihak II selaku Pemegang Hak Cipta berupa Peningkatan Kompetensi Siswa TKJ Melalui Pelatihan Jaringan Komputer dan Implementasi Kurikulum Uji Kompetensi dan Keahlian untuk Meningkatkan Kemampuan pada Aspek Soft Skill di SMK Letris 1 Indonesia untuk didaftarkan di Direktorat Hak Cipta, Desain Industri, Desain Tata Letak dan Sirkuit Terpadu dan Rahasia Dagang, Direktorat Jenderal Hak Kekayaan Intelektual, Kementerian Hukum dan Hak Azasi Manusia R.I.
Demikianlah surat pengalihan hak ini kami buat, agar dapat dipergunakan sebagaimana mestinya.

Jakarta, 24 Januari 2026

Pemegang Hak Cipta	Pencipta
(Dr. Ir. Prudensius Maring, M.A)	Merry Anggraeni

SURAT PERNYATAAN

Yang bertanda tangan dibawah ini:

1. Nama : Hillman Akhyar Damanik
2. Kewarganegaraan : Indonesia
3. Alamat : Flamboyan Unit No.57 Jl. Mesjid Nurul Iman Cipadu - Tangerang 15155
4. Telepon :
5. No. HP & E-mail : 082299099294 & hilladamanik@gmail.com

1. Nama : Merry Anggraeni
2. Kewarganegaraan : Indonesia
3. Alamat : Jl. Swadarma Utara 4 No.62 Ulujami-Pesanggrahan
4. Telepon :
5. No. HP & E-mail : 081316177827 & merry.anggraeni@budiluhur.ac.id

Dengan ini menyatakan bahwa:

1. Karya Cipta yang saya mohonkan:

Berupa : Penelitian

Berjudul : CNCSA: Novel Sequential Algorithm untuk Peningkatan Efisiensi Konfigurasi Jaringan Berbasis Netmiko pada Infrastruktur Virtual.

Tidak meniru dan tidak sama secara esensial dengan Karya Cipta milik pihak lain atau obyek kekayaan intelektual lainnya sebagaimana dimaksud dalam Pasal 68 ayat (2);

- Bukan merupakan Ekspresi Budaya Tradisional sebagaimana dimaksud dalam Pasal 38;
- Bukan merupakan Ciptaan yang tidak diketahui penciptanya sebagaimana dimaksud dalam Pasal 39;

- Bukan merupakan hasil karya yang tidak dilindungi Hak Cipta sebagaimana dimaksud dalam Pasal 41 dan 42;
- Bukan merupakan Ciptaan seni lukis yang berupa logo atau tanda pembeda yang digunakan sebagai merek dalam perdagangan barang/jasa atau digunakan sebagai lambang organisasi, badan usaha, atau badan hukum sebagaimana dimaksud dalam Pasal 65 dan;
- Bukan merupakan Ciptaan yang melanggar norma agama, norma susila, ketertiban umum, pertahanan dan keamanan negara atau melanggar peraturan perundang-undangan sebagaimana dimaksud dalam Pasal 74 ayat (1) huruf d Undang-Undang Nomor 28 Tahun 2014 tentang Hak Cipta.

2. Sebagai pemohon mempunyai kewajiban untuk menyimpan asli contoh ciptaan yang dimohonkan dan harus memberikan apabila dibutuhkan untuk kepentingan penyelesaian sengketa perdata maupun pidana sesuai dengan ketentuan perundang-undangan.

3. Karya Cipta yang saya mohonkan pada Angka 1 tersebut di atas tidak pernah dan tidak sedang dalam sengketa pidana dan/atau perdata di Pengadilan.
4. Dalam hal ketentuan sebagaimana dimaksud dalam Angka 1 dan Angka 3 tersebut di atas saya / kami langgar, maka saya / kami bersedia secara sukarela bahwa:
    a. Permohonan karya cipta yang saya ajukan dianggap ditarik kembali; Karya Cipta yang telah terdaftar dalam Daftar Umum Ciptaan Direktorat Hak Cipta, Direktorat Jenderal Hak Kekayaan Intelektual, Kementerian Hukum Dan Hak Asasi Manusia R.I dihapuskan sesuai dengan ketentuan perundang-undangan yang berlaku.
    b. Dalam hal kepemilikan Hak Cipta yang dimohonkan secara elektronik sedang dalam berperkara dan/atau sedang dalam gugatan di Pengadilan maka status kepemilikan surat pencatatan elektronik tersebut ditangguhkan menunggu putusan Pengadilan yang berkekuatan hukum tetap.

Demikian Surat pernyataan ini saya / kami buat dengan sebenarnya dan untuk dipergunakan sebagimana mestinya.

Jakarta, 24 Januari 2026

Yang menyatakan,

Hillman Akhyar Damanik

Merry Anggraeni