# BAB 2
# DASAR TEORI

## 2.1 KAJIAN PUSTAKA

Terdapat penelitian yang melakukan penerapan otomasi jaringan menggunakan Ansible dan membandingan dengan metode konvensional pada jaringan EIGRP [6]. Pengujian ini dilakukan untuk memverifikasi keakuratan Ansible untuk mengkonfigurasi protokol *routing* EIGRP dan membandingkan kefektifan otomasi Ansible dengan metode konvensional dari empat skenario berupa konfigurasi alamat IP ke antarmuka, protokol *routing* EIGRP, rute statis *default*, dan konfigurasi EIGRP lanjutan, skenario pengujian dilakukan pada 3 buah Router, sebuah *switch*, dan sebuah ubuntu *Network Automation*. Semua hasil penelitian dilakukan dengan menggunakan *software* GNS3. Hasil data yang diambil dalam penelitian ini yang pertama, skrip *Playbook* yang memungkinkan dijalankan di *Docker* otomasi jaringan untuk memeriksa apakah semua tugas seperti alamat IP, alamat *loopback*, dan EIGRP dasar bekerja secara efisien. Untuk pengujian kedua, operasi EIGRP dilakukan untuk memverifikasi konfigurasi EIGRP yang terdiri dari tetangga EIGRP, informasi protokol *routing* dan tabel *routing*. Perintah ping dan perintah *show run* diterapkan. Tes ketiga adalah tes rute statis *default*, yang menggunakan perintah *show* untuk melihat tugas menyebarkan skrip rute statis *default* pada Router. Tes terakhir adalah tes *Fine-Tune* EIGRP. Pengujian ini dilakukan untuk memeriksa utilisasi *bandwidth, hello Interval* dan *hold timer*. Ansible ditemukan berhasil mengotomatiskan skrip dan menerapkan konfigurasi. Hasil dan analisis menunjukkan bahwa konfigurasi EIGRP menggunakan skrip otomatis telah diverifikasi dan akurat serta lebih efisien dari metode konvesional.

Terdapat penelitian yang melakukan konfigurasi *routing* protokol menggunakan Ansible pada Router cisco dan mikrotik dengan raspberry pi dan [7]. Pengujian ini dilakukan untuk menggambarkan metode baru dalam mengkonfigurasi perangkat jaringan dengan menggunakan otomatisasi, mengurangi waktu konfigurasi peralatan serta lebih mudah untuk pemeliharaan. Skenario pengujian dilakukan terhadap 3 *routing* protokol

yaitu EIGRP, OSPF, dan BGP yang dimana terdapat 5 perangkat cisco, 5 perangkat mikrotik dan raspberry pi sebagai *controller node* yang akan dikonfigurasi secara otomatis, terdapat 2 *File* YAML, yang pertama *File* YAML yang berisi konfigurasi cisco yang akan mengkonfigurasi IP *Address*, EIGRP dengan Autonomous *System* 10 dan eBGP *Routing* dengan Sistem Otonom 200, lalu *File* YAML kedua berisi konfigurasi mikrotik yang akan mengkonfigurasi IP *Address*, *Loopback Interface*, Nama *Host*, dan Perutean OSPF. Hasil menunjukan bahwa semua konfigurasi telah digunakan dengan baik setelah sekitar 3 menit di raspberry pi untuk mengkonfigurasi 10 Router.

Terdapat penelitian yang melakukan perbandingan antara dua protokol *routing*, yaitu *Enhanced Interior Gateway Routing Protocol* (EIGRP) dan *Open Shortest Path First* (OSPF), berdasarkan konvergensi jaringan [5]. Skenario pengujian dilakukan 2 topologi yaitu star dan mesh untuk OSPF dan EIGRP yang dismualasikan di lingkungan GNS3. Adapun parameter untuk pengujian yaitu Durasi Konvergensi, Waktu *Startup* Konvergensi, Kegagalan Konvergensi, dan Konvergensi Tautan Baru. Hasil pengujian menunjukan bahwa EIGRP memiliki kinerja yang lebih tinggi dalam durasi konvergensi, waktu ketika tautan gagal, dan tautan baru, menggunakan 2 topologi yaitu star dan mesh, untuk topologi star 10,25 milidetik lebih cepat dibandingkan dengan OSPF, dan untuk topologi mesh 0,2 milidetik lebih cepat dibandingkan dengan OSPF. Hal ini dikarenakan EIGRP tidak melakukan *routing Update* yang membutuhkan waktu lebih lama dibandingkan *routing protocol* OSPF.

Terdapat penelitian yang melakukan studi perbandingan kinerja *Network Automation* Python menggunakan Library Paramiko dan Netmiko untuk mengaktifkan proses routing dinamis pada perangkat router Cisco dimana protokol routing yang digunakan adalah OSPF [9]. Skenario dilakukan pada topologi jaringan *partial-mesh* yang berisi empat buah router Cisco seri 7200 , satu swith, dan satu *Docker Network Automation* dengan scenario pengujian mengukur waktu pemberian *script* konfigurasi, waktu konvergensi jaringan, serta nilai throughput dan delay dari kedua Library *Network Automation* yaitu Netmiko dan Paramiko. waktu proses pemberian

*script* konfigurasi ke perangkat router dengan menggunakan library Netmiko 4,14 kali lebih lambat dibandingkan jika menggunakan Paramiko. Dengan waktu kirim data lebih lambat, maka penggunaan library Netmiko menghasilkan waktu konvergensi jaringan yang juga lebih lambat dibandingkan Paramiko. Dengan waktu konvergensi lebih lambat, maka nilai delay yang dihasilkan oleh library Netmiko juga lebih lambat dan nilai throughput yang dihasilkan juga lebih kecil.

Terdapat penelitian yang melakukan studi perbandingan *Library* Untuk Implementasi *Network Automation* Menggunakan Paramiko Dan Netmiko Pada Router Mikrotik [8]. Pengujian ini dilakukan untuk mengevaluasi rata-rata waktu eksekusi dari empat skenario yang berbeda, yaitu konfigurasi *Static Routing*, *Dynamic Routing* menggunakan RIPv2, *Firewall* dengan NAT (*Network Address Translation*), dan SNMP (*Simple Network Management Protocol*). Dua topologi yang berbeda digunakan dalam pengujian, yakni topologi pertama menggunakan dua perangkat MikroTik, sedangkan topologi kedua menggunakan tiga perangkat MikroTik. Pengambilan data dilakukan dengan menggunakan perangkat lunak GNS3 dan data yang diper*ole*h berupa eksekusi *script* Python. Hasil pengujian menunjukkan bahwa *library* Paramiko lebih unggul dalam waktu eksekusi *script* dibandingkan dengan Netmiko, dengan selisih rata-rata waktu sebesar 3,66 detik.

**Tabel 2.1 Kajian Penelitian Sebelumnya**

| *Year* | *Author* | *Objective* | *Configuration* | *Automation* | *Result* |
|---|---|---|---|---|---|
| 2021 | Mohd Faris Fuzi, Khai*run*nisa Abdullah, Imam Hazwam Abd Halim, Rafiza Ruslan | Melakukan penerapan otomasi jaringan menggunakan Ansible dan membandingan dengan metode konvensian untuk memverifikasi keakuratan konfigurasi EIGRP | *Network Automation* pada EIGRP *Network* | Ansible | Penerapan otomasi Ansible berhasil untuk mengkonfigurasi routing protokol EIGRP |
| 2020 | Muhammad Fauzi Islami, Purnawarman Musa, Missa Lamsani | Melakukan konfigurasi *routing* protokol menggunakan Ansible pada Router cisco dan mikrotik dengan raspberry pi | *Network Automation* pada *Routing Protocol* RIP, OSPF, EIGRP, dan BGP | Ansible | Semua konfigurasi telah diterapkan dengan baik setelah 3 menit di raspberry pi untuk mengkonfigurasi 10 Router. |
| 2020 | Ifeanyi Joseph Okonkwo, Ikiomoye Douglas Emmanuel | perbandingan dua protokol *routing*, (EIGRP) dan (OSPF), berdasarkan konvergensi jaringan | *Routing Protocol* EIGRP, dan OSPF | - | EIGRP memiliki kinerja yang lebih tinggi dalam durasi konvergensi, topologi star 10,25 milidetik lebih cepat, topologi mesh 0,2 milidetik lebih cepat dibandingkan dengan OSPF |

| | | | | | |
|---|---|---|---|---|---|
| 2021 | Kukuh Nugroho, Anggi Dzikri Abrariansyah, Syariful Ikhwan | melakukan studi perbandingan Library menggunakan Paramiko dan Netmiko pada perangkat router Cisco | OSPF | Python | menggunakan library Netmiko 4,14 kali lebih lambat dibandingkan menggunakan Paramiko |
| 2022 | Luis Geraldo Mauboy, Theophilus Wellem | Melakukan studi Perbandingan *Library* Menggunakan Paramiko Dan Netmiko Pada Router Mikrotik | *Dynamic routing* menggunakan RIPv2, NAT, dan SNMP | Python | Rata-rata selisih waktu eksekusi antara Netmiko dan Paramiko adalah 3.66 detik |
| 2023 | Ahmad Riyanto | Membandingkan *Network Automation* Ansible dengan *Network Automation* Python | *Network Automation* pada *Routing* Protokol EIGRP | Ansible dan Python | Otomasi Python Paramiko 4,9 kali lebih cepat dibandingkan dengan otomasi Ansible dengan selisih waktu sebesar 185,89 detik, ekstensi Mitogen dapat mmpercepat waktu pengiriman *script* konfigurasi hingga 3,8 kali lipat atau dapat mengurangi waktu sebesar 64,7 detik, modul Python Multi-Thread dapat mempercepat waktu pengiriman *script* otomasi Python Paramiko hingga 3,9 kali lipat atau dapat mengurangi waktu sebesar 34,92 detik. |

## 2.2 *Network Automation*

*Network Automation* adalah proses mengotomatisasi konfigurasi, manajemen, pengujian, penyebaran, dan operasi peralatan jaringan fisik dan virtual. Ini menggunakan logika yang dapat diprogram untuk mengelola sumber daya dan layanan jaringan, memungkinkan tim operasi jaringan untuk mengkonfigurasi, memperluas, melindungi, dan mengintegrasikan infrastruktur jaringan dan layanan aplikasi dengan lebih cepat daripada saat dilakukan secara manual oleh pengguna [11].

Administrator jaringan dan sistem kerap memanfaatkan bahasa skrip untuk mengotomatisasi proses konfigurasi, sehingga dapat mengurangi waktu, tenaga, dan kesalahan manusia. Dalam proses otomasi jaringan, bahasa pemrograman Python dan Ansible sering digunakan. Untuk mengotomatisasi jaringan dengan efektif, para administrator jaringan harus memiliki pemahaman yang mendalam tentang bagaimana perangkat jaringan bekerja dan berperilaku untuk menerapkan otomatisasi pada perangkat jaringan. Mengotomatisasi jaringan dengan pengetahuan semi-pemahaman tentang perangkat yang terdiri dari jaringan tersebut dapat menyebabkan kesalahan besar dan waktu tidak aktif (*downtime*) yang signifikan [12].

Cara kerja *Network Automation* adalah dengan menggunakan perangkat lunak atau *platform* otomatisasi yang digunakan untuk mengelola konfigurasi, manajemen, pengujian, pengiriman, dan operasi perangkat jaringan secara otomatis. *Platform* ini biasanya menggunakan bahasa pemrograman, seperti Python dan Ansible, serta protokol jaringan, seperti SNMP, N*ETC*ONF, dan REST API untuk berkomunikasi dengan perangkat jaringan.

## 2.3 Ansible

Ansible adalah alat IT *open source* yang dapat digunakan untuk *provisioning*, manajemen infrastruktur, konfigurasi sistem, dan juga penyebaran aplikasi. Seorang Administrator Sistem atau bahkan *Engineer* jaringan dapat mengotomatisasi dan menulis penyebaran atau konfigurasi yang biasanya dilakukan secara berulang pada beberapa perangkat seperti server atau Router. Tentu saja, sebagai orang yang bertanggung jawab dalam

manajemen infrastruktur jaringan, baik instalasi layanan, konfigurasi, atau menjaga sistem agar berjalan dengan baik [13].

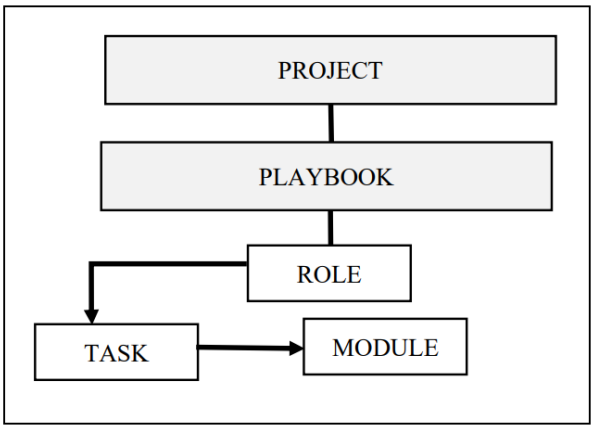
Syarat dari sebuah *system* atau server dapat dikonfigurasi menggunakan Ansible adalah sudah terpasang ansible disana. Ansible menggunakan layanan SSH (*Secure Shelll*) untuk terkoneksi ke server dan menjalankan semua instruksi secara *remote* ditunjukan pada gambar 2.1, tentu hal ini sama halnya dengan yang biasa kita lakukan saat me-*remote* server. dan lebih kerennya lagi, kita tidak perlu memasang Agen di setiap server, kita hanya perlu memasang Ansible di mana Ansible dijalankan [14].


**Gambar 2.1 Komunikasi Otomasi Ansible**

Gambar 2.1 merupakan komunikasi otomasi Ansible yang dimana otomasi Ansible menggunakan layanan SSH (Secure Shelll) untuk terkoneksi atau melakukan otomasi ke server, proses dimulai dari client (otomasi Ansible) yang melakukan koneksi dan menggunakan kunci kriptografi (SSH Key) untuk memverifikasi dan mengidentifikasi SSH server (target) yang dihubungi. Setelah terhubung, SSH server (target) mengirim public key ke client (otomasi Ansible) dan client (otomasi Ansible) harus memiliki key yang sesuai yaitu private key. Jika berhasil, client (otomasi Ansible) dan server (target) melakukan proses autentikasi untuk memverifikasi identitas satu sama lain dan membuka saluran aman. Ketika sudah selesai, client (otomasi Ansible) akan mendapatkan izin akses untuk masuk ke server (target) dan melakukan transmisi data (melakukan otomasi).

### 2.3.1 *Playbook* Ansible

*Playbook* Ansible ditulis dalam format YAML. *Playbook* adalah *File* YAML yang berisi urutan perintah. Tugas (*tasks*), modul (*modules*), dan *File* semuanya merupakan bagian dari struktur peran (*role*). Peran terdiri dari direktori dengan subdirektori, masing-masing memiliki *File* main.yml yang menggambarkan urutan operasi yang harus dilakukan. Modul adalah program pendek yang melakukan aktivitas khusus pada sistem. Modul dapat digunakan sendiri atau sebagai bagian dari skrip yang lebih besar yang dikenal sebagai *Playbook*. Gambar 2.2 menunjukkan struktur Skrip Ansible.




**Gambar 2.2 Struktur Skrip Ansible**

### 2.3.2 *YAML Aint a Markup Languange File*

Automasi jaringan memudahkan insinyur jaringan untuk membuatnya sederhana karena hanya melakukan eksekusi pada satu *File* untuk menerapkan banyak konfigurasi yang ditulis dalam format tertentu seperti YAML. Pada awal pengembangannya, YAML dimaksudkan sebagai singkatan dari "*Yet Another Markup Language*". Dalam perkembangannya, untuk menegaskan tujuannya yang terfokus pada data dan bukan markah dokumen, YAML diubah menjadi singkatan rekursif dari "YAML *Ain't a Markup Language*". YAML adalah *File* yang ditulis dalam format yang mudah dibaca oleh manusia dengan penggunaan indentasi sebagai alternatif kurung kurawal yang biasanya ada dalam bahasa pemrograman umum. Ini mirIP dengan aturan Python tetapi ada perbedaan di antara keduanya. Indentasi dalam *File* YAML tidak mendukung tabulasi yang digunakan kecuali spasi. Ini akan lebih mudah dibaca bahkan oleh pemula [7].

*File* YAML memiliki beberapa aturan yang harus kita patuhi, di antaranya:

1. Spasi digunakan untuk menandai struktur penulisan
2. Tabulasi dilarang dan tidak diizinkan seperti pada bahasa Python
3. Penggunaan tanda — ditulis pada baris pertama
4. Penggunaan tanda ... digunakan jika kita ingin mengakhiri dokumen dalam *File* yang sama
5. Menggunakan tanda # (pagar) untuk memberikan komentar pada baris tertentu
6. Tanda - (strip) menunjukkan anggota daftar pada setiap subdivisi objek
7. Menggunakan tanda : (titik dua) untuk memberikan nilai pada setiap kunci atau *variable*
8. Menggunakan tanda ; (titik koma) untuk membuat *array*

### 2.3.3 *Benefit of Using* Ansible

Manfaat penggunaan Ansible sebagai alat otomatisasi dalam infrastruktur dijelaskan di bawah ini [13]:

1. Sederhana

   Ansible menggunakan sintaks sederhana yang ditulis dalam format YAML yang disebut sebagai *Playbook*.
2. Tanpa Agen

   *Software* sebagai agen tidak perlu diinstal pada *Host* target.
3. Kuat & Fleksibel

   Ansible memiliki banyak modul untuk mengelola infrastruktur jaringan dan layanannya di sistem operasi.
4. Efisien

   Ansible tidak memerlukan ruang pada *hard drive Host* karena bersifat tanpa agen.

### 2.3.4 *Terminology of* Ansible

Dalam Ansible, ada beberapa terminologi yang penting untuk dipahami karena akan memudahkan penggunaan Ansible. Berikut adalah beberapa terminologi beserta penjelasannya [13]:

1. *Node Controller node* atau juga disebut mesin tempat Ansible terpasang dan bertanggung jawab untuk melakukan *provisioning* pada infrastruktur yang dikelola.
2. Inventaris *File* inisialisasi yang berisi informasi tentang perangkat pada infrastruktur yang dikelola.
3. *Playbook Node* masuk untuk Ansible melakukan *provisioning* di mana otomatisasi didefinisikan melalui tugas-tugas yang ditulis dalam format YAML.
4. Modul Abstraksi dari seluruh sistem, seperti sesuatu yang terkait dengan paket atau dalam membuat dan memodifikasi *File*. Ansible memiliki banyak modul bawaan atau beberapa modul yang telah diinstal saat instalasi.
5. Peran ini adalah bagaimana *Playbook* dikelola dan *File* lain untuk memfasilitasi penggunaan lain dan menggunakannya kembali dari langkah *provisioning*.
6. *Play Provisioning* yang dieksekusi dari awal sampai akhir disebut sebagai *Play*. Dengan kata lain, eksekusi *Playbook* disebut *Play*.
7. Fakta Variabel global yang berisi informasi tentang sistem, seperti antarmuka yang terpasang atau operasi sistem.
8. *Handler Handler* digunakan untuk memicu perubahan status dari layanan, seperti me-restart atau menghentikan layanan.
9. *Vault Vault* digunakan untuk mengamankan informasi sensitif seperti nama pengguna dan kata sandi dengan enkrIPsi AES256. 2.1.4 Struktur Ansible net *protocol* jaringan yang membaginya menjadi beberapa sistem otonom.

## 2.4 Python

Python *Network Automation* adalah penggunaan bahasa pemrograman Python untuk mengotomatisasi tugas-tugas pada jaringan komputer seperti konfigurasi perangkat jaringan, manajemen jaringan, pemantauan, dan lain-lain [15].

Python memiliki *library* dan *framework* yang kuat untuk melakukan otomatisasi pada jaringan diantaranya[16]:

1. Netmiko, merupakan perpustakaan multi-vendor yang dibangun di atas Paramiko yang menyediakan antarmuka yang disederhanakan dan konsisten untuk terhubung dan berinteraksi dengan perangkat jaringan. Ini mendukung berbagai vendor jaringan dan memungkinkan Anda mengotomatisasi tugas-tugas seperti pengelolaan konfigurasi, pencadangan perangkat, dan peningkatan firmware.
2. Paramiko, Implementasi Python dari protokol SSHv2 yang memungkinkan komunikasi yang aman dengan perangkat jaringan melalui SSH. Ini memungkinkan Anda mengotomatisasi tugas-tugas seperti menjalankan perintah, mentransfer file, dan mengelola sesi SSH.
3. NAPALM, Perpustakaan yang tidak bergantung pada vendor yang mengabstraksi konfigurasi dan pengelolaan perangkat jaringan. Ini mendukung berbagai sistem operasi jaringan dan menyediakan API yang terpadu untuk mengotomatisasi tugas-tugas seperti konfigurasi perangkat, validasi, dan pemantauan.
4. Nornir, library Python yang digunakan untuk otomatisasi jaringan dan manajemen infrastruktur. Nama "NORNIR" sendiri berasal dari karakter fiksi dalam buku seri "The Lord of the Rings" karya J.R.R. Tolkien, yaitu ras Norn. NORNIR dirancang khusus untuk mempermudah otomatisasi tugas-tugas jaringan yang kompleks dan repetitif.
5. Telnetlib, merupakan modul bawaan Python yang digunakan untuk membuat koneksi Telnet, mengirimkan perintah, dan menerima respons dari perangkat jaringan melalui protokol Telnet. Modul ini tidak mendukung enkripsi atau keamanan yang kuat, sehingga sebaiknya digunakan dengan hati-hati dan hanya untuk tujuan pengujian atau dalam lingkungan yang aman.
6. pyATS/Genie, pyATS (Python Automation Testing System) adalah kerangka kerja yang dikembangkan oleh Cisco yang menyederhanakan otomatisasi dan pengujian jaringan. Ini menyediakan sejumlah perpustakaan, termasuk Genie, yang memungkinkan Anda berinteraksi dengan perangkat jaringan, mengumpulkan data, dan melakukan pengujian dan verifikasi otomatis.

Dengan semua library tersebut maka memungkinkan pengguna untuk mengakses perangkat jaringan secara *remote* dan mengirimkan perintah-perintah secara otomatis. Dalam praktiknya, Python *Network Automation* dapat digunakan untuk beberapa tujuan, antara lain[12]:

1. Konfigurasi Perangkat Jaringan: Dengan menggunakan pustaka seperti netmiko atau paramiko, Python dapat digunakan untuk mengirimkan perintah konfigurasi ke perangkat jaringan seperti router atau switch. Ini memungkinkan otomatisasi pengaturan perangkat yang berulang dan mempercepat penyebaran konfigurasi jaringan.
2. Pengumpulan Data Jaringan: Dengan menggunakan pustaka seperti pyATS atau NAPALM, Python dapat digunakan untuk mengumpulkan data operasional dari perangkat jaringan seperti status antarmuka, tabel routing, atau statistik kinerja. Data ini dapat digunakan untuk memonitor jaringan, analisis, atau pemecahan masalah.
3. Manajemen Perangkat Jaringan: Python juga dapat digunakan untuk melakukan tugas-tugas manajemen jaringan seperti mengonfigurasi perangkat jaringan baru, memonitor keamanan jaringan, atau mengotomatisasi tugas-tugas rutin seperti backup konfigurasi perangkat.

Python *Network Automation* dapat meningkatkan efisiensi dalam mengelola jaringan, mengurangi kesalahan manusia, meningkatkan keamanan, dan memungkinkan tim IT untuk fokus pada tugas-tugas yang lebih kompleks. Beberapa contoh penggunaan Python *Network Automation* adalah mengkonfigurasi *switch*, Router, *firewall*, *load balancer*, dan perangkat jaringan lainnya, serta melakukan pemantauan jaringan dan mengirimkan laporan otomatis jika terjadi masalah pada jaringan[12].

## 2.5 *Library* Paramiko

Paramiko adalah sebuah *library* Python yang digunakan untuk mengakses perangkat jaringan secara *remote* melalui protokol SSH (*Secure Shell*). Paramiko menyediakan fitur seperti transfer *File*, eksekusi perintah, dan tunel jaringan yang memungkinkan pengguna untuk melakukan otomatisasi pada perangkat jaringan[8].

Paramiko juga menyediakan antarmuka Python yang mudah digunakan dan dokumentasi yang lengkap sehingga pengguna dapat menggunakannya dengan mudah. *Library* ini juga memiliki fitur keamanan yang kuat, termasuk enkripsi AES, autentikasi dengan kunci publik, dan proteksi terhadap serangan *brute force* [17].

Beberapa kegunaan Paramiko adalah untuk mengotomatisasi tugas-tugas seperti konfigurasi perangkat jaringan, *backup* konfigurasi, dan pemantauan jaringan. Paramiko juga dapat digunakan untuk transfer *File* secara otomatis antara perangkat jaringan dan server. Selain itu, Paramiko dapat digunakan untuk mengakses perangkat yang menjalankan sistem operasi berbasis Unix, Linux, atau macOS, dan dapat digunakan untuk mengakses perangkat seperti Router, *switch*, *firewall*, dan server[17].

### 2.5.1 *Benefit of Using* Paramiko

Keunggulan ataupun manfaat penggunaan Paramiko sebagai alat otomatisasi dalam infrastruktur dijelaskan di bawah ini [18]:

1. Kemudahan penggunaan

   Paramiko menyediakan antarmuka yang sederhana dan mudah digunakan untuk melakukan koneksi SSH dan menjalankan perintah pada server jarak jauh
2. Banyak fitur dan fleksibilitas

   Modul paramiko menyediakan banyak seperti autentikasi kunci publik, transfer file, forwarding port fitur yang memungkinkan pengguna untuk mengelola koneksi SSH, menjalankan perintah, mentransfer file.
3. Aktif dan didukung oleh komunitas

   Paramiko adalah proyek open-source yang aktif dan didukung oleh komunitas Python.

### 2.5.2 Cara Kerja Paramiko

Library Paramiko bekerja dengan mengimplementasikan protokol SSH (Secure Shell) dalam bahasa pemrograman Python. Protokol SSH digunakan untuk melakukan koneksi ke server jarak

jauh secara aman dan menjalankan perintah-perintah pada server tersebut[18].

Adapun langkah umun tentang cara kerja Paramiko:

1. Mengimport modul Paramiko: untuk menggunakan paramiko langkah pertama yaitu mengimport modul Paramiko ke dalam program Python dengan menggunakan pernyataan import paramiko.
2. Membuat objek SSHClient: pembuatan objek SSHClient yang akan digunakan untuk melakukan koneksi ke server SSH. Objek ini dapat dibuat dengan memanggil paramiko.SSHClient().
3. Mengatur kebijakan keamanan: sebelum melakukan koneksi, perlu mengatur kebijakan keamanan untuk objek SSHClient. Kebijakan keamanan menentukan bagaimana objek SSHClient akan menangani kunci *host* yang tidak dikenal. Dengan cara menggunakan kebijakan AutoAddPolicy() yang secara otomatis menambahkan kunci *host* yang tidak dikenal ke dalam daftar kunci yang diterima.
4. Membuat koneksi SSH: Setelah mengatur kebijakan keamanan, pembuatan koneksi SSH dilakukan dengan cara memanggil metode connect() pada objek SSHClient. Metode ini menerima argumen seperti alamat IP atau nama *host*, username, dan password.
5. Menjalankan perintah: Setelah berhasil terhubung ke server SSH, maka dapat menjalankan perintah-perintah pada server tersebut. Dapat menggunakan metode exec_command() pada objek SSHClient untuk menjalankan perintah dan mendapatkan outputnya.

## 2.6 Mitogen

Mitogen adalah sebuah ekstensi yang dirancang untuk meningkatkan kinerja dan skalabilitas saat menjalankan playbook atau tugas Ansible. Mitogen adalah lapisan koneksi UNIX dan waktu runtime modul yang

sepenuhnya didesain ulang untuk Ansible, yang memungkinkan proses yang lebih cepat dan lebih efisien [19].

Mitogen dapat melakukan peningkatan kecepatan sebesar 1,25x hingga 7x dan pengurangan penggunaan CPU setidaknya 2x, tergantung pada kondisi jaringan, modul yang dieksekusi. Mitogen tidak dapat meningkatkan kinerja modul ketika sudah dieksekusi, tetapi dapat memastikan modul dieksekusi dengan secepat mungkin. [19].

Cara kerja Mitogen adalah sebagai berikut:

1. Dalam Mitogen, setiap *host* target hanya menggunakan satu koneksi, ditambah satu panggilan sudo (panggilan atau perintah yang digunakan untuk menjalankan tugas atau komando sebagai superuser (root) di sistem operasi) untuk setiap akun pengguna. Pendekatan ini jauh lebih efisien dibandingkan menggunakan multiplexing SSH yang digabungkan dengan pipelining. Keuntungannya adalah informasi penting yang diperlukan dapat dijaga di memori RAM antara langkah-langkah tugas, sehingga tidak perlu melakukan proses otentikasi berulang-ulang yang mengisi log sistem. Ini membantu menghemat waktu dan sumber daya yang digunakan dalam menjalankan tugas.
2. Ketika menggunakan Mitogen, hanya dibutuhkan satu kali perjalanan jaringan untuk mengeksekusi langkah tugas yang kode programnya sudah ada di RAM pada *host* target. Dengan menghilangkan pembuatan saluran SSH multiplex, waktu eksekusi dapat dihemat hingga 4 milidetik untuk setiap 1 milidetik laten jaringan yang biasanya dibutuhkan dalam setiap langkah playbook. Dengan cara ini, Mitogen berusaha mempercepat proses eksekusi tugas dengan mengoptimalkan pertukaran data melalui jaringan.
3. Proses dalam Mitogen digunakan secara agresif untuk diulang, yang artinya proses yang sudah ada akan digunakan kembali untuk tugas-tugas berikutnya. Dengan menghindari biaya pemanggilan dan mengompilasi impor ulang, Mitogen dapat menghemat waktu sekitar 300-800 milidetik untuk setiap langkah playbook. Dengan cara ini, Mitogen berupaya untuk mengurangi beban proses eksekusi tugas, sehingga proses dapat berjalan lebih efisien dan cepat.

4. Dalam Mitogen kode yang diperlukan untuk menjalankan tugas di-*host* target akan di-cache (disimpan sementara) di dalam memori RAM. Dengan cara ini, penggunaan bandwidth (lebar pita) dapat dikurangi hingga sepuluh kali lipat jika dibandingkan dengan metode pipelining SSH. Selain itu, karena kode sudah ada di RAM, hanya sedikit data yang perlu dikirimkan melalui jaringan, sehingga jumlah "frame" (paket data) yang berlalu-lintas melalui jaringan akan berkurang hingga lima kali lipat dari jumlah yang biasanya terjadi dalam satu jalankan tugas yang biasa. Dengan kata lain, Mitogen berusaha mengurangi penggunaan bandwidth dan mempercepat proses eksekusi tugas dengan cara mengoptimalkan pertukaran data melalui jaringan.
5. Dengan Mitogen, jumlah tulisan ke sistem file target dapat dikurangi. Biasanya, pada konfigurasi Ansible yang umum, Ansible akan berulang kali menulis ulang dan mengekstrak file ZIP ke beberapa direktori sementara pada *host* target. Namun, dengan menggunakan Mitogen, masalah keamanan terkait file sementara dalam situasi akun lintas dapat sepenuhnya dihindari. Dengan kata lain, Mitogen membantu mengurangi aktivitas menulis berlebih ke sistem file pada *host* target, sehingga meningkatkan keamanan dan mencegah masalah terkait file sementara yang mungkin muncul dalam skenario penggunaan yang kompleks. Hal ini membantu mengoptimalkan proses eksekusi tugas dan meningkatkan efisiensi keseluruhan dalam pengelolaan infrastruktur dengan Ansible.

   Mitogen memberikan efek paling kuat pada playbook yang berisi banyak tindakan singkat. Ini terjadi ketika Ansible memiliki banyak tugas kecil yang harus dilakukan pada *host* target. Dalam situasi ini, overhead Ansible menjadi beban utama yang mempengaruhi waktu eksekusi tugas . Dengan Mitogen, overhead tersebut dapat dikurangi, sehingga waktu eksekusi tugas menjadi lebih cepat. Mitogen membantu mengoptimalkan proses eksekusi, memastikan tugas-tugas kecil selesai dengan lebih efisien, dan mengurangi beban yang harus ditanggung oleh Ansible. Sehingga, penggunaan Mitogen sangat menguntungkan dalam mengelola playbook

dengan banyak tindakan singkat, yang akan memberikan hasil yang lebih optimal dalam penggunaan Ansible [19].

## 2.7 Multi-Threading

Python Multi-Threading adalah kemampuan dalam bahasa pemrograman Python untuk menjalankan beberapa thread secara bersamaan di dalam satu proses. Thread adalah unit eksekusi kecil yang dapat bekerja secara paralel dalam sebuah program. Dengan menggunakan multi-threading, program Python dapat menjalankan beberapa tugas secara simultan, meningkatkan efisiensi dan responsifitas program[20].

Dalam Python, modul threading digunakan untuk mengimplementasikan multi-threading. Modul ini menyediakan kelas Thread yang memungkinkan pembuatan dan pengelolaan thread. Setiap thread dijalankan secara independen, memiliki alur eksekusi sendiri, tetapi berbagi sumber daya yang sama dengan thread lain dalam proses yang sama[20].

Beberapa kegunaan multi-threading dalam Python meliputi:

1. Peningkatan responsifitas: Dengan menjalankan tugas-tugas yang membutuhkan waktu lama secara paralel dalam thread terpisah, program dapat tetap responsif dan tidak terblokir saat menunggu operasi selesai.
2. Pemrosesan bersamaan: Jika ada beberapa tugas yang dapat dikerjakan secara bersamaan, multi-threading memungkinkan pemanfaatan sumber daya CPU yang lebih efisien dengan menjalankan thread-thread tersebut secara paralel.
3. Pemrosesan IO-bound: Dalam kasus tugas yang banyak bergantung pada operasi input/output (misalnya membaca atau menulis file), multi-threading dapat meningkatkan kinerja dengan menjalankan tugas-tugas tersebut secara paralel.

Untuk memulai thread baru, perlu memanggil metode berikut yang tersedia dalam modul thread “thread.start_new_thread (function, args[, kwargs])” Panggilan metode ini memungkinkan cara yang cepat dan efisien untuk membuat thread baru baik di Linux maupun Windows. Panggilan metode ini mengembalikan segera dan thread anak dimulai dan memanggil fungsi dengan daftar argumen yang dilewatkan. Ketika fungsi selesai, thread

berakhir. Di sini, args adalah tuple dari argumen; gunakan tuple kosong untuk memanggil fungsi tanpa melewatkan argumen apa pun. kwargs adalah kamus opsional dari argumen kata kunci[21].

## 2.8 *Secure Shell* (SSH)

*Secure Shell* (SSH) adalah sebuah protokol jaringan yang digunakan untuk mengakses perangkat jaringan secara aman melalui jaringan yang tidak terpercaya, seperti Internet. SSH memungkinkan pengguna untuk mengirimkan dan menerima data dengan aman dengan menggunakan enkripsi yang kuat [22].

Protokol SSH (Secure Shell) berada di lapisan aplikasi dalam model referensi OSI (Open Systems Interconnection) yang terdiri dari tujuh lapisan. Lapisan aplikasi adalah lapisan tertinggi dalam model OSI dan berinteraksi langsung dengan aplikasi pengguna[23].

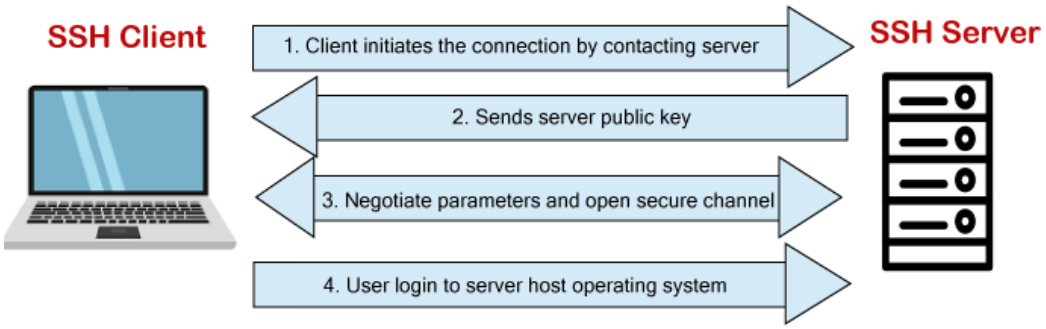
Protokol SSH (Secure Shell) secara umum bekerja pada layet aplikasi, tetapi juga berinteraksi dengan tiga lapisan atau layer, yang disebut sebagai "SSH protocol stack" atau "SSH protocol architecture". Berikut adalah tiga lapisan protokol SSH [24]:

1. Transport Layer (Lapisan Transportasi): Ini adalah lapisan terbawah dalam protokol SSH. Lapisan ini bertanggung jawab untuk menyediakan layanan enkripsi, integritas, dan autentikasi pada tingkat koneksi. Lapisan Transportasi menggunakan enkripsi simetris untuk melindungi data yang dikirimkan antara client dan server SSH.
2. User Authentication Layer (Lapisan Autentikasi Pengguna): Lapisan ini terletak di atas lapisan Transportasi. Lapisan Autentikasi Pengguna mengatur proses autentikasi pengguna saat mereka mencoba mengakses server SSH. Protokol SSH mendukung berbagai metode autentikasi, seperti menggunakan kata sandi, kunci publik-privat, dan metode autentikasi lainnya.
3. Connection Layer (Lapisan Koneksi): Lapisan ini berada di atas lapisan Autentikasi Pengguna dan menyediakan layanan yang berkaitan dengan koneksi SSH yang sudah terautentikasi. Lapisan Koneksi mengelola sesi

SSH dan menyediakan berbagai layanan, seperti forwarding port, transfer file, dan eksekusi perintah pada server jarak jauh.

Secara keseluruhan, protokol SSH memiliki tiga lapisan utama yang saling berinteraksi untuk menyediakan komunikasi aman dan autentikasi pada koneksi SSH [24].

SSH dapat digunakan untuk berbagai keperluan seperti mengakses server jarak jauh, melakukan transfer *File*, dan mengakses perangkat jaringan seperti Router, *switch*, dan *firewall*. Dalam proses penggunaannya, SSH memerlukan autentikasi yang kuat menggunakan metode kunci publik dan kunci privat yang memungkinkan pengguna untuk mengakses perangkat jaringan dengan aman [22].




**Gambar 2.3 Komunikasi Protokol SSH**

Cara kerja SSH adalah dengan menerapkan model *client-server* dapat dilihat pada gambar 2.3, dimana SSH *client* melakukan koneksi ke SSH server. Proses koneksi dimulai dari *client* yang melakukan koneksi dan menggunakan kunci kriptografi (SSH *Key*) untuk memverifikasi dan mengidentifikasi SSH server yang dihubungi. Setelah terhubung, SSH menyediakan otentikasi kata sandi yang kuat dan otentikasi kunci publik, serta komunikasi data terenkrIPsi antara dua komputer yang terhubung melalui jaringan terbuka seperti *Inter*net. Selain itu, SSH memungkinkan pengguna untuk mengakses dan memodifikasi berbagai macam pengaturan maupun *File* yang ada di dalam server [25].

## 2.9 *Routing Protocol*

*Routing* adalah proses berpindahnya data pada jaringan dengan menggunakan perangkat yang disebut Router. Router akan memilihkan rute data yang benar berdasarkan arah yang ingin dituju. Pada aplikasinya, Router akan mengelola informasi mengenai rute data tersebut menjadi suatu skema yang dinamakan tabel *routing*. Tabel ini akan berisi informasi tentang *Interface* Router pada jaringan yang digunakan pada proses pengiriman data [26].

Protokol *routing* adalah serangkaian aturan dan prosedur yang digunakan oleh perangkat jaringan untuk memilih jalur terbaik untuk mengirimkan paket data dari sumber ke tujuan melalui jaringan. Protokol *routing* memungkinkan perangkat jaringan untuk berkomunikasi dan berkoordinasi satu sama lain untuk mengoptimalkan pengiriman data melalui jaringan dengan cara yang efisien dan dapat diandalkan [27]. Terdapat beberapa jenis protokol *routing*, seperti OSPF, EIGRP, RIP, IS-IS, BGP dan lain-lain, yang masing-masing memiliki kelebihan dan kekurangan dalam mengelola lalu lintas jaringan dan terbagi menjadi 2 jenis protokol yaitu:

1. *Interior Gateway Protocol* (IGP) merupakan sebuah *routing protocol* yang digunakan untuk *routing* di dalam sebuah AS. IGP biasa disebut sebagai *Intra* AS *routing* yang berarti proses per*routing*an berada di dalam AS. Protokol *Routing* IGP diklasifikasikan menjadi dua yaitu:
   A. *Distance-vector Routing protocol*s yang berarti bahwa rute yang dihitung adalah jarak dan arah, dimana jarak akan ditentukan berdasarkan *metric* dan arah adalah arah *hop* berikutnya dari Router. Contoh dari *Distance Vector Routing protocol*s adalah RIPv1, IGRP, RIPv2 dan EIGRP.
   B. *Link-state Routing protocol* sering disebut dengan protokol *Shortest Path First* (SPF). Protokol *routing link-state* merupakan sebuah proses *routing* yang membangun topologi databasenya sendiri, dengan kata lain setiap Router akan menerima map dari Router tetangga. Contoh dari *Link State Routing protocol* adalah OSPF dan IS-IS

2. *Exterior Gateway Protocol* (EGP) merupakan sebuah *protocol routing* yang digunakan untuk *routing* antara AS. EGP biasa disebut sebagai *Inter*-AS yang berarti proses *routing* antara AS. Klasifikasi dari EGP hanya terdapat satu yaitu berupa *Path-Vector Routing protocol* yang terdiri dari BGP.

## 2.10 *Enhanced Interior Gateway Routing Protocol* (EIGRP)

EIGRP (*Enhanced Interior Gateway Routing Protocol*) adalah hasil pengembangan dari *protocol* pendahulunya, yaitu *Interior Gateway Routing Protocol* (IGRP). Keduanya adalah *routing* yang dikembangkan oleh CISCO. EIGRP hanya dapat diaplikasikan pada Router milik CISCO saja, sedangkan untuk Router yang lain tidak bisa. EIGRP disebut sebagai *Hybrid Routing Protocol* karena cara kerjanya yang berdasarkan dua tipe *routing protocol* [5].

EIGRP bekerja membangun sebuah *routing protocol* dengan sebuah algoritma yang disebut dengan DUAL. DUAL dapat menghitung dan membangun sebuah tabel *routing*. DUAL juga memberi izin untuk Router yang menggunakan EIGRP agar dapat menentukan jalur alternatif sendiri, tanpa menunggu informasi dari Router lainnya [28].

Ada 5 tipe paket data yang digunakan oleh EIGRP untuk berkomunikasi dengan perangkatperangkat penyusun jaringan, antara lain:

1. EIGRP mengirimkan *hello packet* agar dapat mengetahui apakah Router-Router tetangganya masih dalam keadaan aktif atau sudah dalam keadaan tidak aktif.
2. *Update packets,* berfungsi untuk memberitahu tujuan yang dapat dijangkau oleh Router.
3. *Acknowledgement packet* digunakan sebagai pemberitahuan pada saat data sudah dtiterima.
4. *Query packet*s adalah sebuah permintaan untuk mengirimkan sebuah rute.
5. *Reply packet*s digunakan untuk menjawab permintaan *Query packet*, dimana ini akan memberitahu bahwa Router pengirim tidak melakukan perhitungan ulang untuk jalurnya karena *feasible Successors* masih dapat dilalui.

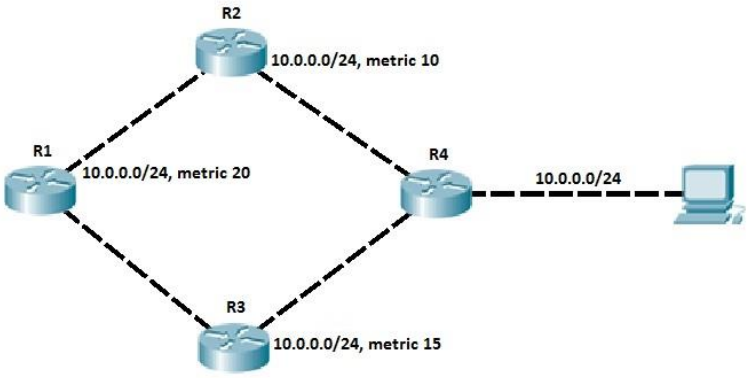
## 2.11 Cara Kerja EIGRP

Protokol EIGRP akan bekerja berdasarkan algoritma yang disebut dengan DUAL (*Diffusing Update Algorithm*). DUAL akan mengirimkan sebuah *Query packet* kepada Router untuk membantu mengirim data ke sebuah jaringan. Router-Router lainnya akan meneruskan *Query packet* tersebut, sampai ada Router yang mengirim *Reply packet* yang berisi cara untuk menuju ke jaringan yang dituju. Untuk menentukan rute terbaik menuju jaringan yang dituju, EIGRP menggunakan sebuah metrik dimana nilai metrik terendah akan dipilih dan dimasukkan pada tabel *routing* [28].

Setelah *routing* table telah terbentuk, EIGRP lalu mengirim *hello packet* agar dapat mengetahui kondisi Router-Router disekitarnya apakah masih aktif atau tidak aktif. *Hello packet* dikirim serentak, dan didalam paket tersebut terdapat hold time. *Hold time* adalah waktu maksimum yang diberikan saat menunggu paket balasan yang dikirmkan oleh Router tetangga. Jika sampai batas waktu tertentu Router tetangga tidak membalas *hello packet* tersebut, maka Router tersebut dianggap tidak aktif. Setelah itu EIGRP akan melakukan pembaharuan pada tabel *routing*-nya. *Update* pada tabel *routing* akan dilakukan apabila terjadi perubahan pada jaringan. *Update packet* yang berisi informasi tentang perubahan rute akan dikirimkan menuju Router-Router tetangganya [28].

EIGRP menentukan rute terbaik untuk mencapai tujuan berdasarkan algoritma DUAL. DUAL akan melakukan perhitungan agar dapat memilih Router yang akan menjadi *succesor* dan *feasible Successor*. *Successor* adalah rute yang paling dekat dan paling efisien untuk menuju ke jaringan tujuan, rute disimpan di tabel perutean. Sedangkan *feasible Successor* adalah rute cadangan (*backup*) yang akan dilalui apabila rute utamanya tidak dapat dilewati [28].

Untuk memilih rute sebagai *feasible Successor*, satu syarat harus dipenuhi, yaitu jarak yang diiklankan tetangga atau *neighbor's advertised distance* (AD) untuk rute harus kurang dari jarak layak penerus atau *successor's feasible distance* (FD) ditunjukan pada gambar 2.4.




**Gambar 2.4 Konsep *successor* dan *feasible successor***

R1 memiliki dua jalur untuk mencapai subnet 10.0.0.0/24. Jalur melalui R2 memiliki metrik terbaik (20) dan disimpan di tabel perutean R1. Rute lain, melalui R3, adalah rute pengganti yang layak, karena kondisi kelayakan telah terpenuhi (jarak yang diumumkan R3 sebesar 15 kurang dari jarak layak R1 sebesar 20). R1 menyimpan rute itu di tabel topologi. Rute ini dapat langsung digunakan jika rute utama mengalami kegagalan.

## 2.12 Waktu Pengiriman Script Konfigurasi

Waktu pengiriman script adalah waktu yang dibutuhkan oleh client atau *Network Automation* untuk memberikan perintah konfigurasi ke server atau target pada jaringan dan memanfaatkan traffic protokol SSH yang ditangkap oleh Wireshark dari jalur Docker Network Automation menuju switch dikarenakan otomasi Ansible dan Python Paramiko menggunakan layanan SSH (*Secure Shelll*) untuk terkoneksi ke server. Filterisasi protokol SSH dilakukan menggunakan hasil yang dihasilkan oleh Wireshark, yang didapatkan dari waktu akhir SSH dikurangi dengan waktu awal SSH.

## 2.13 Konvergensi Jaringan

Konvergensi jaringan routing mengacu pada proses di mana semua router dalam jaringan secara konsisten mempelajari dan menyebarkan informasi routing yang akurat setelah terjadi perubahan topologi atau keadaan jaringan. Proses konvergensi memastikan bahwa semua router memiliki pengetahuan yang konsisten tentang bagaimana mengirimkan paket melalui jaringan[29].

Ketika terjadi perubahan topologi, seperti penambahan atau penghapusan koneksi atau perangkat jaringan, atau jika ada kegagalan pada saluran komunikasi, router-router dalam jaringan harus beradaptasi dengan perubahan tersebut. Proses konvergensi memastikan bahwa setiap router memiliki tabel routing yang diperbarui dan telah mengambil langkah-langkah yang diperlukan untuk mengirimkan paket melalui rute yang benar.

Waktu konvergensi adalah waktu yang dibutuhkan oleh jaringan komputer atau protokol routing untuk mencapai kesepakatan dan stabilitas dalam hal informasi topologi setelah terjadi perubahan pada jaringan. Perubahan tersebut dapat berupa perubahan status link, kegagalan perangkat, atau perubahan lain dalam topologi jaringan[30].

Proses konvergensi terjadi ketika semua perangkat di jaringan menyelaraskan informasi topologi mereka sehingga mereka memiliki gambaran yang konsisten tentang bagaimana mengirimkan paket data. Selama waktu konvergensi, protokol routing bekerja untuk menghitung jalur terbaik yang akan digunakan oleh setiap router untuk mencapai tujuan tertentu, dan tabel perutean di-update di seluruh jaringan.

Waktu konvergensi sangat penting dalam jaringan karena selama proses ini, beberapa paket data mungkin tidak dapat diarahkan secara optimal, dan ada kemungkinan paket hilang atau terjebak. Semakin cepat konvergensi terjadi, semakin sedikit dampak negatif yang dialami oleh jaringan akibat perubahan topologi atau keadaan.[31]

## 2.14 Failover Convergence

Failover Konvergensi adalah waktu yang dibutuhkan oleh jaringan atau protokol routing untuk mencapai kestabilan setelah terjadi perubahan topologi akibat failover. Failover terjadi ketika terjadi kegagalan pada jalur utama (primary) atau perangkat utama yang menyediakan layanan. Selama proses failover, jaringan harus beradaptasi dengan perubahan topologi ini dan mencapai kesepakatan tentang jalur baru yang harus digunakan untuk pengiriman data[32].

Proses failover konvergensi serupa dengan proses konvergensi biasa, yaitu melibatkan deteksi perubahan, pemberitahuan, perhitungan jalur

alternatif, dan pembaruan tabel perutean. Namun, dalam kasus failover, prioritas utama adalah untuk secepat mungkin memulihkan layanan agar jaringan tetap berfungsi dengan benar meskipun terjadi kegagalan[33].

Protokol routing dan mekanisme failover yang efisien sangat penting dalam situasi ini untuk memastikan waktu konvergensi setelah failover minimal, sehingga layanan jaringan dapat kembali beroperasi dengan cepat dan terus berfungsi dengan baik.

## 2.15 GNS3 (*Graphic Network Simulator* 3)

GNS3 adalah emulator jaringan dengan antarmuka grafis (GUI) yang dirilis pada tahun 2008. Dengan GNS3, kita dapat mensimulasikan dan mengeksemplikan seperti menggunakan perangkat sungguhan. Emulator ini dapat dijalankan pada banyak sistem operasi seperti Windows, Linux, dan OSX. Perbedaan antara GNS3 dengan simulator jaringan lain seperti Cisco Packet Tracert adalah GNS3 dapat mensimulasikan dan mengeksemplikan perangkat dari vendor yang berbeda. Oleh karena itu, tidak hanya perangkat Cisco yang dapat disimulasikan di GNS3. Selain itu, pengguna dapat mengeksplorasi lebih banyak di GNS3 karena emulator ini diciptakan sebanyak mungkin seperti di dunia nyata rekayasa jaringan dengan perangkat sungguhan [7].

### 2.15.1 *Docker Network Automation*

*Appliance Docker* GNS3 merupakan penggunaan kontainer *Docker* yang telah dikonfigurasi khusus untuk mengotomatisasi tugas-tugas jaringan dalam simulasi jaringan menggunakan perangkat lunak GNS3. Kontainer *Docker* ini dirancang untuk menyediakan lingkungan yang dapat dijalankan di dalam GNS3 dan mengintegrasikan alat atau skrip yang mendukung automasi jaringan [34].

*Appliance Docker* digunakan untuk menyediakan solusi yang dapat dijalankan di dalam jaringan simulasi GNS3 dan mengotomatisasi tugas-tugas jaringan, seperti konfigurasi perangkat

jaringan, pengujian jaringan, pemantauan jaringan, atau implementasi kebijakan jaringan.

Beberapa keuntungan penggunaan *Appliance Docker* dalam automasi jaringan GNS3 meliputi:

1. Keterpisahan: *Appliance Docker* menjalankan tugas automasi jaringan dalam kontainer terpisah yang tidak mempengaruhi *host* GNS3 atau perangkat virtual lainnya dalam simulasi. Ini memungkinkan pengujian dan pengembangan automasi yang aman tanpa risiko merusak atau mengganggu simulasi jaringan yang ada.
2. Reusabilitas: Kontainer *Docker* sebagai *Appliance* automasi jaringan dapat dikemas, didistribusikan, dan digunakan kembali dalam berbagai skenario simulasi. Ini memungkinkan portabilitas dan efisiensi dalam penggunaan solusi automasi jaringan di berbagai topologi jaringan yang berbeda.
3. Integrasi yang Mudah: *Appliance Docker* dapat diintegrasikan dengan GNS3 melalui fitur *Docker* yang disediakan oleh GNS3. Ini memungkinkan penggunaan dan manajemen *Appliance Docker* langsung dari antarmuka GNS3, membuatnya lebih mudah untuk memanfaatkan automasi jaringan menggunakan solusi *Docker* dalam simulasi jaringan.
4. Skalabilitas: Dengan menggunakan *Appliance Docker* dalam automasi jaringan GNS3, Anda dapat dengan mudah menggandakan atau mengurangi jumlah instance yang dijalankan dalam simulasi. Ini memberikan fleksibilitas dan skalabilitas yang lebih baik dalam mengelola tugas automasi yang kompleks atau skenario jaringan yang membutuhkan sumber daya tambahan.

Melalui penggunaan *Appliance Docker* dalam automasi jaringan GNS3, pengguna dapat mengoptimalkan dan memperluas kemampuan GNS3 dengan mengintegrasikan alat-alat atau skrip automasi jaringan yang dijalankan dalam kontainer *Docker*.

## 2.16 Wireshark

Wireshark adalah perangkat lunak yang digunakan untuk melakukan analisis paket data pada jaringan secara *real-time* dan menampilkan hasilnya dalam format yang mudah dipahami oleh pengguna. Wireshark dapat melakukan paket *filtering*, paket *color coding*, *sniffing*, dan fitur-fitur lain yang memungkinkan penggunanya untuk melihat detail dari setiap paket data yang dikirim dan diterima oleh jaringan [35].

Wireshark secara umum digunakan untuk memecahkan masalah jaringan serta mengembangkan dan menguji coba suatu *software*. Wireshark dapat menampilkan data dari ratusan protokol yang berbeda pada semua jenis jaringan utama. Wireshark adalah penganalisa protokol jaringan gratis yang digunakan oleh profesional keamanan, pengembang, atau administrator sistem untuk memantau lalu lintas jaringan dan menganalisis paket data yang dikirim dan diterima oleh jaringan [35].