Available *online* at : http://bit.ly/InfoTekJar

InfoTekJar : Jurnal Nasional Informatika dan Teknologi Jaringan

ISSN (Print) 2540-7597 | ISSN (Online) 2540-7600

Teknologi Jaringan

Perbandingan Kinerja *Library Paramiko* dan *Netmiko* Dalam Proses Otomasi Jaringan

Kukuh Nugroho, Anggi Dzikri Abrariansyah, Syariful Ikhwan

Program Studi S1 Teknik Telekomunikasi, Fakultas Teknik Telekomunikasi dan Elektro, Institut Teknologi Telkom Purwokerto, 53147

KATA KUNCI

Otomasi jaringan, *Paramiko*, *Netmiko*, OSPF (*Open Shortest Path First)*

CORRESPONDENCE

Phone: +62 81321829010

E-mail: kukuh@ittelkom-pwt.ac.id

A B S T R A C T

Configuring network devices in a large network needs a relatively long time when manually configured by a network administrator. Thus, it will make the possibility of error in writing configuration commands becomes even greater. The concept of network automation can be a solution to these problems. The configuration process will be centralized. A programming language is needed in order to make the concept of network automation to communicate between the central computer and all devices in the network. In this research, two libraries in the Python programming language are used, namely Paramiko and Netmiko to enable dynamic routing processes on routers wherein the routing protocol used is Open Shortest Path First (OSPF). The experiment results of a comparative test of the performance of the network automation process show that the use of the Paramiko library provides a 4.14 time faster configuration time than Netmiko.

ABSTRAK

Proses konfigurasi perangkat jaringan dalam skala besar memerlukan waktu konfigurasi yang relatif lama dan apabila hal tersebut dilakukan secara manual oleh seorang network administrator, maka kemungkinan terjadinya kesalahan dalam menuliskan perintah konfigurasi menjadi semakin besar. Konsep otomasi jaringan dapat menjadi solusi dari permasalahan tersebut. Proses konfigurasi akan dilakukan secara terpusat. Dalam membuat konsep otomasi jaringan diperlukan bahasa pemrograman untuk mengkomunikasikan antara komputer pusat dengan semua perangkat dalam jaringan. Pada penelitian ini digunakan dua library dalam bahasa pemrograman Python yaitu Paramiko dan Netmiko untuk mengaktifkan proses routing dinamis pada perangkat router dimana protokol routing yang digunakan adalah OSPF. Dari hasil uji perbandingan performansi proses otomasi jaringan memperlihatkan bahwa penggunaan library Paramiko memberikan waktu konfigurasi ke perangkat router 4,14 kali lebih cepat dari Netmiko.

PENDAHULUAN

Otomasi merupakan sebuah konsep dalam menggantikan peran manusia dalam sebuah aktifitas atau pekerjaan tertentu, terlebih apabila pekerjaan tersebut adalah pekerjaan yang dilakukan secara berulang [1]. Sebagai contoh adalah pekerjaan seorang *Network Engineer* dalam mengkonfigurasi sebuah perangkat dalam sebuah jaringan. Biasanya pada saat awal perangkat dipasang, konfigurasi awal yang diberikan pada perangkat relatif sama dan jumlah perangkat yang relatif banyak. Karena konfigurasi yang diberikan relatif sama dan dalam jumlah perangkat yang relatif banyak, maka untuk mempermudah dan mempercepat proses konfigurasi diperlukan proses otomasi jaringan. Otomasi jaringan menggunakan konsep pemrograman dan menyediakan konsep manajemen jaringan yang terpusat. Hal ini memungkinkan admin jaringan dapat mengkonfigurasi dan menyatukan jaringan dan aplikasi yang digunakan didalamnya [2]. Salah satu contoh aplikasi yang digunakan untuk proses otomasi jaringan adalah aplikasi untuk mengkonfigurasi mekanisme *routing* baik secara *static* dan *dynamic*, membuat VLAN (*Virtual* LAN), melakukan proses *backup* data hasil konfigurasi dan mengembalikan data tersebut ke perangkat. Dengan adanya otomasi jaringan, maka proses dalam membuat konfigurasi perangkat-perangkat dalam jaringan menjadi terpusat. Selain dalam hal konfigurasi perangkat, proses otomasi jaringan juga dilakukan pada pekerjaan pendaftaran pengguna *Hotspot* di sebuah kampus [3]. Pekerjaan tersebut bisa digantikan dengan sistem otomasi dengan memanfaatkan API (*Application Programming Interface*) yang terdapat pada perangkat Mikrotik.

Perangkat jaringan yang biasanya digunakan untuk proses otomasi adalah perangkat penghubung seperti router atau *multi-*

https://doi.org/10.30743/infotekjar.v5i1.2758

layer switch. Fungsi utama perangkat router adalah menghubungkan antar komputer pada jaringan yang berbeda. Data dapat dikirimkan dari satu komputer di sisi wilayah jaringan satu ke komputer pada wilayah jaringan yang lain. Kecepatan dalam proses transfer data tergantung dari *bandwidth* yang terpasang pada perangkat router tersebut. Agar proses aliran *transfer* data data dapat maksimal, diperlukan proses manajemen *bandwidth* yang baik [4]. Diperlukan Bahasa pemrograman tertentu untuk bisa membuat sistem konfigurasi perangkat router dalam jaringan menjadi otomatis [5]. Sistem otomasi perangkat dalam sebuah jaringan menjadi sebuah solusi untuk mempercepat proses pembangunan jaringan itu sendiri dibandingkan dengan proses konfigurasi manual yang dilakukan secara langsung pada masing-masing perangkat [6]. Proses ini memerlukan waktu konfigurasi yang relatif lama, karena admin jaringan harus memasukkan satu-per-satu baris perintah konfigurasi ke perangkat router. Dengan menggunakan sistem otomatis jaringan, konfigurasi perangkat router dalam jaringan dalam dilakukan pada satu komputer saja secara terpusat. Tentu saja, pada perangkat komputer tersebut sudah diaktifkan program tertentu yang digunakan untuk memasukkan perintah konfigurasi dalam perangkat router. Salah satu pilihan Bahasa pemrograman yang digunakan adalah *Python* [7].

Konsep konfigurasi jaringan awal (tradisional) masih dilakukan secara langsung pada perangkat. Seorang administrator jaringan harus memberikan perintah konfigurasi pada semua perangkat dalam jaringan. Hal ini menjadi tidak efisien dan kurang cepat apabila diimplementasikan pada jaringan skala besar, dimana terdapat banyak perangkat jaringan yang harus dikonfigurasi dan jumlah baris perintah yang diberikan untuk mengkonfigurasi perangkat juga relatif banyak. Adanya konsep otomasi jaringan menjadikan kerja dari seorang administrator jaringan menjadi lebih efisien karena hanya dengan menjalankan sebuah program dalam komputer pusat dan waktu konfigurasi perangkat dalam jaringan juga relatif cepat karena tidak perlu mengetikkan satu-per-satu baris perintah dalam mengkonfigurasi perangkat tersebut. Permasalahan lamanya waktu konfigurasi perangkat dan adanya kesalahan dalam menuliskan perintah menjadi berkurang dengan adanya proses otomatis jaringan. Selain itu, proses otomasi jaringan menjadikan sebuah solusi untuk menghemat pengeluaran dilihat dari sisi OPEX (*Operating Expenditure*) [8]. Pada awalnya, kebutuhan tenaga *network administrator* sebanding dengan jumlah perangkat yang akan dikonfigurasi. Dengan adanya konsep konfigurasi secara terpusat, maka seorang *network administrator* dapat menangani beberapa perangkat sekaligus. Selain itu, dengan menggunakan konsep otomasi jaringan dapat mempercepat waktu konfigurasi perangkat penghubung dalam jaringan, tetapi juga dalam menghemat waktu perawatan perangkat secara berkala, terlebih apabila ditemui kasus jaringan dengan skala besar.

Sistem otomasi jaringan membutuhkan sebuah *software* atau program untuk menjalankan perintah ke dalam sebuah perangkat. Salah satu program yang dapat digunakan adalah *Python* [9],[10]. Program tersebut akan dijalankan pada perangkat komputer yang difungsikan sebagai komputer pusat untuk mengkonfigurasi semua perangkat penghubung dalam jaringan. Komputer pusat akan mengatur pola aliran trafik data yang dilakukan oleh perangkat penghubung dengan cara memberikan perintah konfigurasi untuk perangkat yang dimasukkan ke dalam *script Python*. Penelitian ini ditujukan untuk mengkonfigurasi perangkat penghubung jaringan yaitu router secara otomatis dengan menggunakan *script Python*. Perintah konfigurasi yang akan dimasukkan dalam *script Pyhon* adalah perintah untuk mengaktifkan proses *routing* dengan menggunakan protokol routing OSPF (*Open Shortest Path First*). Proses *routing* merupakan sebuah proses yang dilakukan oleh perangkat router dalam menentukan rute dari paket yang diterima untuk bisa diteruskan ke tujuan paket tersebut [11]. Informasi rute tentang alamat network dalam jaringan akan dipertukarkan antar router yang sama-sama mengaktifkan protokol routing OSPF. Jaminan pertukaran informasi rute antar router OSPF aman jika ditambahkan mekanisme VPN (*Virtual Private Network*) [12]. Pola aliran trafik data akan bisa dilakukan oleh perangkat router apabila mekanisme *routing* sudah diaktifkan pada masing-masing perangkat router tersebut. Proses otomasi mekanisme *routing* tidak langsung dilakukan pada perangkat router, melainkan pada sebuah komputer terpusat yang didalamnya sudah dijalankan program *Python* untuk memberikan perintah konfigurasi ke semua perangkat router dalam jaringan. *Python* sendiri memiliki beberapa jenis *library* yang digunakan untuk implementasi sistem otomasi jaringan diantaranya adalah *Paramiko* dan *Netmiko* [13],[14]. Namun kedua jenis *library Python* tersebut memiliki perbedaan ketika akan digunakan untuk proses otomasi jaringan. *Library Paramiko* dapat langsung menjalankan *script* yang berisi konfigurasi perangkat, kemudian dikirimkan ke perangkat yang akan dikonfigurasi menggunakan protokol SSH (*Secure Shell*). Sedangkan penggunaan *library Netmiko* masih menggunakan modul tambahan yaitu *yaml* yang berfungsi untuk menyimpan data hasil konfigurasi perangkat. Modul *yaml* nantinya akan dipanggil dan dibaca oleh *library Netmiko*. Data hasil konfigurasi akan dikirimkan ke perangkat melalui jaringan menggunakan protokol yang sama yaitu SSH. Pada penelitian yang dilakukan oleh Wiryawan [15], berhasil dibuat aplikasi dengan menggunakan Bahasa pemrograman *Python* dan *library* yang digunakan adalah *Paramiko*. Aplikasi tersebut berhasil digunakan untuk melakukan proses otomasi jaringan dengan melakukan proses konfigurasi mekanisme *routing* secara statis maupun dinamis menggunakan protokol *routing* OSPF, RIP, dan BGP. Namun pada penelitian tersebut belum sampai pada tahap pengecekan performansi jaringan. Pada penelitian ini akan dilakukan proses otomasi jaringan dengan menggunakan Bahasa pemrograman yang sama yaitu *Python* dengan menambahkan *library Netmiko* selain *Paramiko*. Nantinya akan dibandingkan performansi jaringan hasil dari proses otomasi dengan menggunakan parameter uji waktu pemberikan perintah konfigurasi ke router, waktu konvergensi jaringan, *throughput*, dan *delay*.

METODOLOGI

Analisa perbandingkan kinerja *library Paramiko* dan *Netmiko* dalam membuat sistem otomasi jaringan menggunakan topologi jaringan yang sama seperti yang terlihat pada keterangan Gambar 1. Kedua *library* tersebut akan dijalankan pada komputer *Docker*, sehingga proses konfigurasi dilakukan pada satu komputer secara terpusat. Proses pengambilan data menggunakan bantuan *software* emulasi GNS3 yang sudah mendukung proses otomasi jaringan (*network automation*). Semua perangkat yang terhubung yang digunakan untuk proses pengambilan data dilakukan pada *software* GNS3.

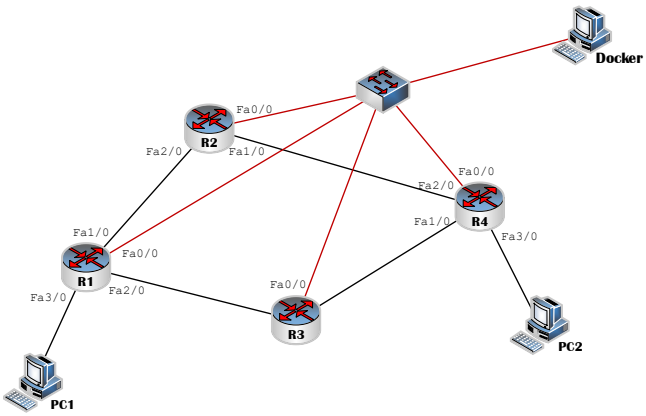
 https://doi.org/10.30743/infotekjar.v5i1.2758

Fungsi dari komputer *Docker* seperti yang terlihat pada keterangan Gambar 1 adalah untuk menjalankan program/*script Python* yang menggunakan dua *library* yang berbeda yaitu *Paramiko* dan *Netmiko*. Fungsi dari *script* tersebut adalah untuk mengirimkan perintah konfigurasi perangkat, dalam hal ini adalah router, melalui jaringan. Proses konfigurasi tidak dilakukan pada masing-masing router, melainkan dilakukan pada satu komputer secara terpusat yaitu *Ubuntu Docker*. Perintah konfigurasi terlebih dahulu diberikan pada *script Python* yang dijalankan pada *Ubuntu Docker*. *Script* hasil konfigurasi kemudian dikirimkan melalui media ke masing-masing router. Media yang digunakan untuk menghubungkan antar perangkat pada topologi jaringan adalah kabel *twisted pair*.

Penelitian ini dilakukan hanya dengan menjalankan dua modul (*library*) yang terdapat pada *Python* yang difungsikan untuk proses otomasi jaringan yaitu *Paramiko* dan *Netmiko*. Kinerja proses otomasi jaringan yang dilakukan oleh kedua *library* tersebut kemudian akan dibandingkan dengan menggunakan parameter konvergensi jaringan, nilai *throughput* dan *delay*. Waktu konvergensi jaringan menggunakan acuan protokol *routing* OSPF. Nantinya protokol *routing* OSPF akan diaktifkan pada masing-masing router dengan cara memanfaatkan dua *library Python* yaitu *Netmiko* dan *Paramiko*.

A. *Rancangan Topologi Jaringan*

Skema dari rancangan topologi jaringan terlihat pada keterangan Gambar 1. Topologi jaringan uji menggunakan empat buah router Cisco seri 7200 yang dihubungkan satu sama lain dengan menggunakan topologi *partial-mesh*. Terdapat mekanisme jalur backup (cadangan) apabila pertukaran trafik data antara PC1 dan PC2 mengalami masalah. Trafik data akan dialihkan ke jalur cadangan, sehingga proses komunikasi masih bisa berlangsung. Keberadaan komputer PC1 dan PC2 digunakan untuk menguji performansi jaringan dilihat dari sisi parameter waktu konvergensi jaringan, *throughput* dan *delay*. Sedangkan peran dari perangkat komputer *Docker* adalah untuk mengirimkan perintah konfigurasi untuk mengaktifkan protokol *routing* OSPF pada masing-masing router yang dimasukkan ke dalam *script library Python*.



Gambar 1. Topologi jaringan uji

Topologi jaringan uji seperti yang terlihat pada keterangan Gambar 1 akan dibuat menggunakan bantuan *software* GNS3 dan dengan memanfaatkan proses virtualisasi komputer. Pada pembangunan sistem menggunakan dua komputer virtual dalam satu komputer fisik. *Software* yang digunakan untuk membuat sistem virtualisasi adalah VMWare *Workstation*. Satu komputer virtual digunakan untuk membuat topologi jaringan yang menggunakan empat buah router, satu switch dan dua buah komputer yaitu PC1 dan PC2. Seperti yang terlihat pada keterangan gambar 1, komputer Docker akan dibuat secara *virtual* terletak pada komputer yang terpisah. Pada komputer tersebut akan dijalankan sistem virtualiasi tingkat OS (*Operating System*) yaitu dengan menjalankan *Docker* yang mendukung proses otomasi jaringan. Sistem virtualisasi menggunakan *Docker* juga dapat digunakan untuk membuat konsep jaringan *virtual* yang dinamakan sebagai NFV (*Network Function Virtualization*) [16].

B. *Konfigurasi Perangkat*

Protokol *routing* OSPF akan diaktifkan secara otomatis pada semua router. Proses konfigurasi akan dimulai dari komputer *Docker*. Program *Python* yang menggunakan *library Paramiko* dan *Netmiko* akan dijalankan pada komputer *Docker*. Program tersebut berisi perintah konfigurasi OSPF pada masing-masing router. Proses *routing* akan dilakukan oleh perangkat router ketika semua informasi rute lawan sudah dikumpulkan oleh protokol *routing* OSPF yang aktif pada router tersebut. Proses pencatatan informasi alamat *network* lawan akan dilakukan secara otomatis dengan menggunakan bantuan protokol *routing* OSPF. Data akan dikirimkan dari PC1 ke PC2 yang digunakan sebagai data uji. Proses komunikasi antara komputer *Docker* dengan masing-masing router menggunakan protokol SSH (*Secure Shell*). Jaminan proses transfer data antara komputer *Docker* dengan masing-masing router yang aman terjadi ketika digunakan protokol SSH [17]. Proses komunikasi antara komputer *Docker* dengan perangkat router dapat dilakukan ketika masing-masing *interface* router sudah diberikan alamat IP secara manual. Versi dari pengalamatan IP menggunakan IPv4 (IP versi 4). Hal ini seperti yang terlihat pada keterangan Tabel 1.

Kelas alamat IPv4 yang digunakan untuk memberikan alamat IP pada semua interface perangkat jaringan uji adalah kelas A dan C. Misalnya pada pengalamatan *interface* Fa0/0 di router R1 menggunakan kelas A dengan alamat IP 10.10.10.2/24 dan pada pengalamatan IP pada *interface* Fa3/0 menggunakan kelas C yaitu 192.168.1.1/24. Perangkat komputer *Docker* terhubung dengan semua router dengan menggunakan alamat *network* yang sama yaitu 10.10.10.0/24. Hal ini sesuai dengan skema perencanaan topologi jaringan yaitu komputer *Docker* terhubung dengan router R1, R2, R3, dan R4 dengan menggunakan bantuan perangkat switch. Mekanisme *routing* tidak diperlukan untuk

mengkomunikasikan antara komputer *Docker* dengan masing-masing router karena perangkat penghubung yang digunakan adalah switch.

Tabel 1. Pengalamatan IP Pada *Interface* Router dan Komputer

Perangkat	Interface	Alamat IP	Perangkat	Interface	Alamat IP
R1	Fa0/0	10.10.10.2/24	R3	Fa0/0	10.10.10.4/24
	Fa1/0	20.20.20.1/30		Fa1/0	50.50.50.2/30
	Fa2/0	40.40.40.2/30		Fa2/0	40.40.40.1/30
	Fa3/0	192.168.1.1/24	R4	Fa0/0	10.10.10.5/24
R2	Fa0/0	10.10.10.3/24		Fa1/0	50.50.50.1/24
	Fa1/0	30.30.30.1/30		Fa2/0	30.30.30.2/30
	Fa2/0	20.20.20.2/30		Fa1/0	192.168.2.1/24
Host B	Eth0	192.168.2.2/24	*Host* A	Eth0	192.168.1.2/24
Ubuntu Docker	Eth0	10.10.10.1/24			

Proses konfigurasi dimulai dari sisi komputer *Docker* yang akan mengirimkan perintah utuk mengkonfigurasi protokol *routing* OSPF pada masing-masing router. Sebelum *script Python* yang berisi perintah konfigurasi routing dikirimkan, terlebih dahulu *server* SSH diaktifkan pada masing-masing router. *Library Python Paramiko* dan *Netmiko* menggunakan protokol SSH dalam proses komunikasi dengan perangkat lawan, sehingga konsep komunikasi sistem *client-server* akan diaktifkan pada komputer *Docker* yang diposisikan sebagai *client* dan router sebagai *server* SSH. Gambar 2 berikut merupakan contoh konfigurasi *server* SSH di router R1.

```
R1(config)#username kukuh secret cisco
R1(config)#username kukuh privilege 15
R1(config)#ip domain-name telkom.com
R1(config)#crypto key generate rsa modulus 1024
R1(config)#line vty 0 4
R1(config-line)#transport input ssh
R1(config-line)#login local
```

Gambar 2. Konfigurasi SSH di router R1

Konfigurasi *server* SSH dilakukan pada semua router. Gambar 2 menjelaskan perintah untuk mengkonfigurasi *server* SSH di router R1. Proses komunikasi antara komputer *Docker* dengan router menggunakan jalur virtual (*vty*). Komputer *Docker* harus menggunakan *username* "kukuh" dan *password* "cisco" untuk bisa masuk ke perangkat router. Data yang dipertukarkan antara komputer *Docker* dengan router akan di-enkripsi dengan menggunakan algoritma adalah RSA (*Rivest–Shamir–Adleman*) dengan menggunakan kunci sebesar 1024 bit.

Username dan kata kunci yang sudah dibuat pada saat mengaktifkan *server* SSH digunakan sebagai acuan dalam membuat program *Python* untuk proses otomasi jaringan menggunakan *library Paramiko*. Contoh dari penggunaan *library Paramiko* dalam mengirimkan perintah untuk mengkonfigurasi alamat IP terdapat pada keterangan Gambar 3. Deklarasi awal dari program adalah dengan cara memanggil modul *paramiko* (*import paramiko*). Modul tersebut yang nantinya digunakan untuk proses konfigurasi alamat IP secara otomatis pada router R1. Alamat IP tujuan dari penerima paket yang berisi *script* program adalah 10.10.10.2 yang merupakan alamat IP yang digunakan oleh *interface* Fa0/0 di router R1. Isian data *text* untuk variabel "*username*" dan "*password*" dibuat sama seperti yang dikonfigurasi pada *server* SSH di router R1.

```
import paramiko
import time

def r1() :
        ip_address = "10.10.10.2"
        username = "kukuh"
        password = "cisco"

        ssh_client = paramiko.SSHClient()
ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh_client.connect(hostname=ip_address,username=username,
password=password)

        print "Login Sukses di {0}".format(ip_address)
        conn = ssh_client.invoke_shell()

        conn.send ("conf t\n")
        conn.send ("int f1/0\n")
conn.send ("ip add 20.20.20.1 255.255.255.252\n")
        conn.send ("no sh\n")
        time.sleep (1)

r1()
```

Gambar 3. Penggunaan *library Paramiko* dalam konfigurasi perangkat router

Dalam membuat *script Python* untuk proses otomasi jaringan dengan menggunakan *library Paramiko* berbeda dengan *Netmiko*. Penggunaan *library Paramiko* menggunakan proses kirim yang lebih sederhana dibandingkan dengan *Netmiko*. Dalam satu *script Python* sudah termasuk modul *Paramiko* dan perintah konfigurasi, seperti yang terlihat pada keterangan gambar 3. Pada penggunaan *library Netmiko*, letak dari *script* yang berisi perintah konfigurasi perangkat dibuat terpisah. Letak dari perintah konfigurasi akan tersimpan dalam satu *file* tersendiri dengan nama "*inventory.yml*". Terdapat program utama yang difungsikan untuk memanggil modul *yaml* sebagai modul untuk mengaktifkan *library Netmiko*.

```
import yaml

def read_yaml(path="inventory.yml"):
    with open(path) as f:
        yaml_content = yaml.safe_load(f.read())
    return yaml_content

def main():
    parsed_yaml = read_yaml()
    print(parsed_yaml)

if __name__ == "__main__":
    main()
```

Gambar 4. *Script Python* untuk menjalankan modul *yaml*

Gambar 4 merupakan program utama ketika menggunakan *library Netmiko*. Program tersebut akan diarahkan untuk mengambil *file* konfigurasi router yang tersimpan dalam modul

https://doi.org/10.30743/infotekjar.v5i1.2758

"*yaml*". Deklarasi awal program adalah memanggil dengan modul "*yaml*". Kemudian program tersebut akan mengambil isi konfigurasi yang tersimpan dalam *file* "*inventory.yml*".

```
Core:
  vars:
    username: kukuh
    password: cisco

    hostname: r1
    host: 10.10.10.2
    int_config:
      - interface: FastEthernet1/0
        ip_address: 20.20.20.1 255.255.255.252

    ospf_config:
      - area: 0
        network:
          - 20.20.20.0 0.0.0.3
```

Gambar 5. Hasil konfigurasi *routing* yang tersimpan dalam *file* "*inventory.yml*"

Perintah konfigurasi yang terlihat pada Gambar 5 digunakan untuk mengaktifkan protokol *routing* OSPF yang diberikan ke router R1. Sebelum *file* konfigurasi OSPF dikirimkan ke router R1, terlebih dahulu proses koneksi antara komputer *Docker* dengan router R1 dilakukan dengan menggunakan protokol SSH. Router R1 akan melakukan proses verifikasi dengan mencocokkan isian variabel "*username*" dan "*password*". Jika proses verifikasi berhasil, maka data yang berisi *script Python* akan dikirimkan ke router oleh komputer *Docker*.

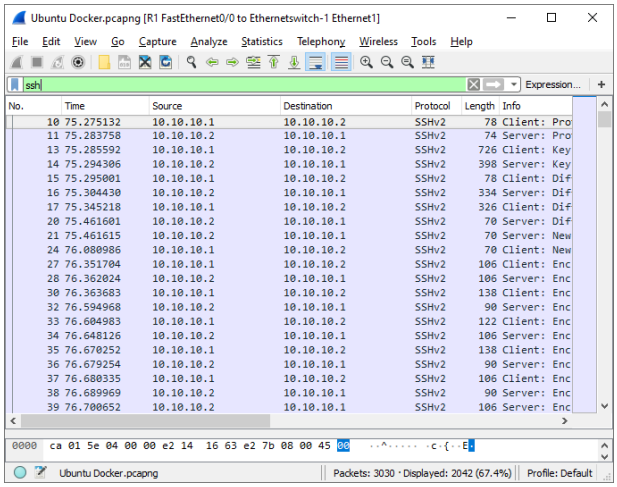
HASIL DAN PEMBAHASAN

Pengujian dua *library Python* yaitu *Paramiko* dan *Netmiko* dalam proses otomasi jaringan dilakukan dengan menggunakan skenario topologi jaringan yang terdapat pada Gambar 1. Komputer yang akan memberikan data berupa *script Python* untuk mengkonfigurasi perangkat router adalah komputer *Docker*. Ketika komputer *Docker* mengirimkan data *script Python*, performansi dari *library Paramiko* dan *Netmiko* akan diujikan. Parameter yang digunakan untuk menguji kedua *library* tersebut adalah waktu dalam memberikan data *script Python* tersebut ke perangkat router. Parameter berikutnya yang akan diujikan adalah waktu konvergensi jaringan, *throughput* dan *delay*.

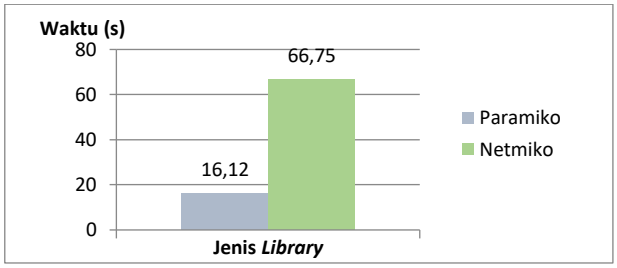
A. Pengukuran Waktu Pemberian Perintah Konfigurasi ke Router

Program *Python* yang dijalankan pada komputer *Docker* digunakan untuk memberikan perintah konfigurasi ke masing-masing router. Perintah konfigurasi yang akan diberikan diantara adalah konfigurasi dalam memberikan alamat IP pada *interface* router dan mengaktifkan protokol *routing* OSPF yang digunakan oleh perangkat router untuk proses *routing*. Lamanya waktu dari data *script Python* tersebut dikirimkan dari komputer *Docker* ke perangkat router akan dihitung. *Software* tambahan yang digunakan untuk mengitung waktu pemberian perintah konfigurasi ke router adalah *Wireshark*. Cara kerja dari *software Wireshark* yaitu akan memproses semua data yang diterima atau dikirim oleh sebuah *interface* perangkat. Gambar 6 berikut menjelaskan proses yang dilakukan oleh komputer *Docker* ketika mengirimkan dan menerima data *script Python* dengan menggunakan *software Wireshark*.

Dalam mengirimkan data *script Python*, komputer *Docker* akan menggunakan protokol SSH. Data yang dikirim atau diterima oleh *interface* komputer *Docker* akan disaring berdasarkan kriteria protokol SSH. Hal ini terlihat pada keterangan Gambar 6. Semua data yang ditampilkan pada *software Wireshark* merupakan data yang dipertukarkan antara komputer *Docker* dengan alamat IP 10.10.10.1 dengan perangkat tujuan yaitu router R1 yang menggunakan alamat IP 10.10.10.2. Proses komunikasi antara komputer *Docker* sebagai SSH *client* dan router R1 sebagai SSH *server* terus-menerus dilakukan sampai semua segmen dari data *script Python* berhasil terkirim. Sebelum proses pertukaran paket, proses autentikasi antara kedua perangkat tersebut akan dilakukan. Waktu yang dibutuhkan dari awal data dikirimkan sampai data tersebut diterima oleh router R1 dinamakan sebagai waktu pemberian perintah konfigurasi ke router. Hasil pengukuran dari waktu pemberian perintah konfigurasi ke router terdapat pada keterangan Gambar 7.



Gambar 6. Penyaringan data berdasarkan protokol SSH



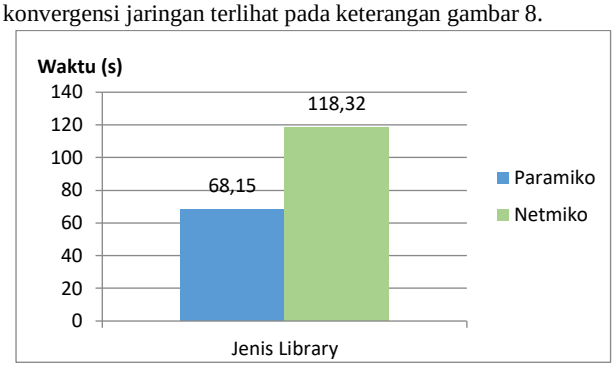
Gambar 7. Waktu pemberian perintah konfigurasi ke router

Hasil pengukuran dengan menggunakan *software wireshark* memperlihatkan bahwa penggunaan *library Paramiko* memberikan waktu pengiriman data *script Python* ke perangkat router yang lebih cepat dibandingkan dengan *Netmiko*. Penggunaan *library Paramiko* menghasilkan waktu pengiriman data sebesar 16,12 detik, sedangkan *Netmiko* menghasilkan waktu sebesar 66,75 detik. Penggunaan *library Paramiko* memberikan waktu yang lebih cepat dibandingkan *Netmiko*. Pada saat penggunaan *library Netmiko*, *script* perintah yang berisi konfigurasi protokol *routing* OSPF tidak diikutkan dalam program utama. Perintah konfigurasi OSPF berada pada *file* yang berbeda yaitu "*inventory.yml*". Ketika program utama dijalankan, program tersebut masih akan mengambil isi dari *file* konfigurasi yang tersimpan dalam "*inventory.yml*", kemudian mengirimkan ke perangkat router. Hal ini menyebabkan waktu pengiriman ketika menggunakan *library Netmiko* lebih besar dibandingkan *Paramiko*.

B. Pengukuran Waktu Konvergensi Jaringan OSPF

Waktu konvergensi menjadi salah satu faktor penentu sebuah paket bisa diteruskan oleh sebuah router ke perangkat tujuan. Sebuah router akan bisa melakukan proses *routing* jika informasi rute yang tersimpan dalam tabel routing sudah lengkap. Jika jaringan belum mencapai kondisi konvergen, dalam artian informasi rute yang tersimpan dalam tabel *routing* masing-masing belum lengkap, maka tidak ada jaminan dari sebuah data bisa diteruskan oleh sebuah router. Semua informasi alamat network lawan harus tersimpan dalam tabel *routing* semua router dalam jaringan. Alamat *network* lawan disini adalah alamat *network* yang tidak terhubung langsung dengan sebuah router. Ketika semua informasi alamat *network* lawan sudah tersimpan dalam tabel routing sebuah router, maka kondisi tersebut dinamakan sebagai kondisi konvergen. Waktu dari awal router menghimpun atau mengumpulkan informasi alamat *network* lawan sampai semua alamat network lawan tercatat ke dalam tabel *routing* sebuah router dinamakan dengan istilah waktu konvergensi.

Proses perhitungan waktu konvergensi dimulai ketika PC1 mengirimkan paket ICMP (*Internet Control Message Protocol*) ke PC2. Proses pengiriman paket dilakukan secara terus-menerus. Pada saat kondisi jaringan belum mencapai konvergen, paket dari PC1 belum bisa diteruskan ke PC2. Pesan yang muncul dalam jendela command prompt ketika proses tes koneksi dari PC1 ke PC2 dengan cara mengirimkan paket ICMP adalah "*destination host unreachable*". Pesan ini akan terus muncul dalam tampilan jendela command prompt sampai dicapai kondisi jaringan konvergen. Dalam membuat kondisi konvergen, komputer *Docker* akan mengirimkan data *script Python* dari dua *library* yang berbeda. Data tersebut berisi informasi konfigurasi OSPF yang diberikan pada masing-masing router. Ketika data diberikan dan diproses oleh masing-masing router, kemudian protokol *routing* OSPF akan aktif di dalam router tersebut. OSPF akan bekerja dalam menghimpun informasi alamat *network* lawan sampai dicapai kondisi jaringan konvergen. Pesan yang muncul pada jendela *command prompt* akan berbeda ketika dicapai kondisi konvergen. Informasi alamat *network* lawan akan lengkap. Paket masuk *interface* router, kemudian akan dilakukan proses routing dan dicarikan *interface* keluaran yang sesuai dengan informasi yang terdapat dalam tabel *routing*. Ketika data diproses oleh router dan dikirimkan ke perangkat tujuan, pesan yang muncul pada jendela *command prompt* berubah menjadi "*reply*". Waktu ketika tampilan jendela *command prompt* yaitu "*network unreachable*" sampai berubah menjadi "*reply*" digunakan sebagai acuan dalam menentukan waktu konvergensi jaringan. *Software Wireshark* digunakan untuk menghitung waktu konvergensi jaringan. Hasil dari perhitungan waktu konvergensi jaringan terlihat pada keterangan gambar 8.



Gambar 8. Waktu konvergensi jaringan

Hasil pengukuran waktu pemberian perintah konfigurasi ke masing-masing router oleh dua *library Python* yaitu *Paramiko* dan *Netmiko* akan berbanding lurus terhadap hasil pengukuran waktu konvergensi jaringan. Dari hasil pengukuran waktu konvergensi jaringan seperti yang terlihat pada keterangan Gambar 8 terlihat bahwa waktu konvergensi jaringan dengan menggunakan *library Paramiko* sebagai program untuk memberikan perintah konfigurasi *routing* ke masing-masing router memiliki nilai waktu konvergensi jaringan yang relatih lebih cepat dibandingkan jika digunakan *library Netmiko*. Penggunaan *library Paramiko* menghasilkan waktu konvergensi sebesar 68,15 detik, sedangkan waktu konvergensi menjadi lebih besar ketika digunakan *library Netmiko* yaitu sebesar 118,32 detik.

C. Pengukuran Nilai Throughput

Pengukuran nilai *throughput* dilakukan pada saat data *script Python* dikirimkan dari komputer *Docker* ke masing-masing router melalui perangkat switch. Pengukuran nilai *throughput* dilakukan pada *interface* switch. *Software wireshark* akan diaktifkan pada *interface* switch tersebut. Hasil pengukuran nilai *throughput* dengan menggunakan dua *library Python* berbeda terdapat pada keterangan Tabel 2.

Tabel 2 Hasil Pengukuran Nilai *Throughput*

Jenis *Library*	Router tujuan				Rata-Rata Nilai *Throughput* (Kbps)
	R1	R2	R3	R4	
Paramiko	89,375	91,782	91,363	85,77	**89,57**
Netmiko	28,062	28,49	24,456	27,45	**27,11**

Pengukuran nilai *throughput* dilakukan pada masing-masing router. Data diambil dari setiap router ketika data tersebut dikirimkan oleh komputer *Docker*. Karena pola pengiriman data *script Python* dilakukan secara *unicast*. Data akan dikirimkan dari satu perangkat sumber menuju ke satu perangkat tujuan. Sebagai perangkat pengirim adalah komputer *Docker* dan perangkat penerima adalah router. Komputer *Docker* akan mengirimkan data tersebut ke masing-masing router secara bergantian. Sehingga dalam mengukur nilai *throughput* akan dilakukan pada masing-masing pengiriman. Nilai *throughput* yang dihasilkan merepresentasikan besaran nilai *bandwidth* yang digunakan dalam proses pengiriman data *script Python* yang dilakukan oleh komputer *Docker*. Dari informasi yang terdapat pada Tabel 2, proses pertukaran data antara komputer *Docker* dengan masing-masing router menghasilkan nilai *throughput* yang berbeda. Data tersebut merupakan *script Python* dari penggunaan *library Paramiko* dan *Netmiko* yang berisi konfigurasi OSPF yang akan diaktifkan pada masing-masing router.

 https://doi.org/10.30743/infotekjar.v5i1.2758

Available *online* at : http://bit.ly/InfoTekJar

InfoTekJar : Jurnal Nasional Informatika dan Teknologi Jaringan

ISSN (Print) 2540-7597 | ISSN (Online) 2540-7600

Hasil pengukuran nilai *throughput* merupakan nilai rata-rata dari empat kali pengiriman data ke router yang berbeda. Hasil pengukuran menunjukkan bahwa penggunaan *library Paramiko* menghasilkan rata-rata nilai *throughput* yang lebih besar dibandingkan *Netmiko*. Parameter nilai *throughput* dikatakan lebih baik jika mempunyai nilai yang lebih besar. Penggunaan *library Paramiko* menghasilkan rata-rata nilai *throughput* sebesar 89,57 Kbps, sedangkan jika menggunakan *library Netmiko* hanya menghasilkan rata-rata nilai *throughput* sebesar 27,11 Kbps. Perbedaan konsep pemrograman yang digunakan oleh *library Paramiko* menghasilkan nilai *throughput* yang lebih baik dibandingkan dengan *Netmiko*. *Throughput* yang digunakan oleh *library Netmiko* lebih kecil dibandingkan *Paramiko* dikarenakan ukuran *file* dari program utama yang digunakan oleh *Netmiko* lebih kecil dibandingkan dengan *Paramiko*. Program utama pada *library Netmiko* hanya berisi *script Python* untuk memanggil modul *yaml*. *Script* yang berisi perintah konfigurasi protokol *routing* OSPF diletakkan pada *file* yang berbeda. Nama dari *file* tersebut adalah "*inventory.yml*". Berbeda dengan konsep pemrograman yang digunakan oleh *library Paramiko*. *Script* yang berisi konfigurasi OSPF dijadikan satu dengan program utama.

D. Pengukuran Nilai Delay

Pengujian dengan menggunakan parameter nilai *delay* dilakukan untuk mengetahui hubungan dengan hasil pengukuran sebelumnya, baik itu antara hasil nilai *throughput*, waktu konvergensi, maupun waktu pengiriman data *script Python* ke masing-masing router. Pengukuran nilai *delay* dilakukan pada *interface* switch. Proses pengukuran dilakukan ketika komputer *Docker* ketika komputer tersebut mengirimkan data *script Python* ke masing-masing perangkat router. Hasil pengukuran nilai *delay* dari masing-masing *library Python* terlihat pada Tabel 3.

Tabel 3 Hasil Pengukuran Nilai *Delay*

Jenis *Library*	Router tujuan				Rata-Rata Nilai *Delay* (ms)
	R1	R2	R3	R4	
Paramiko	0,0896	0,0872	0,0876	0,0933	**0,09**
Netmiko	0,285	0,2807	0,3271	0,291	**0,30**

Pengukuran nilai *delay* juga dilakukan pada empat router tujuan yang berbeda. Proses pengukuran dilakukan ketika komputer *Docker* mengirimkan data *script Python* ke masing-masing router. Data yang diterima oleh perangkat switch kemudian ditangkap oleh *software wireshark*. Seperti pada pengukuran parameter sebelumnya. Pengukuran nilai *delay* juga menggunakan *software wireshark*. Hasil dari pengukuran nilai *delay* memperlihatkan bahwa penggunaan *library Paramiko* menghasilkan nilai *delay* yang lebih baik dibandingkan dengan *Netmiko*. Rata-rata nilai waktu yang dibutuhkan data *script Python* untuk bisa sampai ke *interface* switch dari komputer *Docker* ketika digunakan *library Paramiko* adalah sebesar 0,09 ms. Sedangkan penggunaan *library Netmiko* menghasilkan nilai *delay* sebesar 0,3 ms. Perbedaan konsep pemrograman antara *library Paramiko* dan *Netmiko* memperlihatkan penggunaan *library Netmiko* membutuhkan waktu kirim data *script Python* ke perangkat router yang lebih lama dibandingkan ketika menggunakan *library Paramiko*.

PENUTUP

Perbedaan dalam proses mengeksekusi program antara penggunaan *library Paramiko* dan *Netmiko* menjadikan performansi *library Paramiko* dalam proses otomasi jaringan lebih baik dari *Netmiko*. *Library Paramiko* menggunakan satu tahap proses eksekusi perintah, sedangkan pada penggunaan *library Netmiko* terdapat dua tahap eksekusi perintah. *Library Netmiko* terlebih dahulu akan mencari *file* hasil konfigurasi yang terdapat "*inventory.yml*" sebelum dapat mengirimkan data *script Python* ke perangkat router. Dengan adanya dua kali tahap eksekusi perintah tersebut maka waktu proses kirim data *script Python* 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.

DAFTAR PUSTAKA

[1] M. P. Groover, *Automation, production systems, and computer-integrated manufacturing*. India: Pearson Education, 2016.

[2] F5 Networks and Red Hat, *NetOps Meets DevOps: The State of Network Automation*. 2018.

[3] A. Zakaria, A. Prihantara, and A. A. Hartono, "Integrasi Application Programing Interface, PHP, dan MySQL untuk Otomatisasi Verifikasi dan Aktifasi Pengguna Layanan Hotspot MikroTik," *JUITA J. Inform.*, vol. 7, no. 2, p. 63, 2019.

[4] P. Ferdiansyah, R. Indrayani, and S. Subektiningsih, "Analisis Manajemen Bandwidth Menggunakan Hierarchical Token Bucket Pada Router dengan Standar Deviasi," *J. Nas. Teknol. dan Sist. Inf.*, vol. 6, no. 1, pp. 38–45, 2020.

[5] E. Chou, *Mastering Python Networking*. Birmingham: Packt Publishing Ltd, 2017.

[6] K. Jambunatha, "Design and implement Automated

https://doi.org/10.30743/infotekjar.v5i1.2758

Procedure to upgrade remote network devices using Python.," *IEEE Int. Adv. Comput. Conf.*, pp. 217–221, 2015.

[7] P. Mihăilă, T. Bălan, R. Curpen, and F. Sandu, "Network Automation and Abstraction using Python Programming Methods," *MACRo 2015*, vol. 2, no. 1, pp. 95–103, 2017.

[8] K. Katiyar, R., Pawar, P., Gupta, A., & Kataoka, "Auto-configuration of SDN switches in SDN/non-SDN hybrid network," *Proc. Asian Internet Eng. Conf.*, no. ACM, pp. 48–5, 2015.

[9] D. M. O. F. Sarker, *Python Network Programming Cookbook*. 2014.

[10] B. Aly, *Hands-On Enterprise Automation with Python: Automate common administrative and security tasks with Python*. Packt Publishing Ltd, 2018.

[11] K. Nugroho, *IP Routing Menggunakan Cisco & Mikrotik Dalam Teori & Praktik*. Bandung: INFORMATIKA, 2016.

[12] H. A. Musril, "Desain Virtual Private Network (VPN) Berbasis Open Shortest Path First (OSPF)," *InfoTekJar (Jurnal Nas. Inform. dan Teknol. Jaringan)*, vol. 3, no. 2, pp. 83–88, 2019.

[13] M.-I. Candrea-Bogza and P. Ciotîrnae, "Integrated Management of Transport and Commutation Resources over the Network Layer," *J. Mil. Technol.*, vol. 2, no. 1, pp. 27–30, 2019.

[14] Achmad Komarudin, *Otomatisasi Administrasi Jaringan Dengan Script Python*. Jakarta: Jasakom, 2018.

[15] R. A. Wiryawan and N. R. Rosyid, "Pengembangan Aplikasi Otomatisasi Administrasi Jaringan Berbasis Website Menggunakan Bahasa Pemrograman Python," *Simetris J. Tek. Mesin, Elektro dan Ilmu Komput.*, vol. 10, no. 2, pp. 741–752, 2019.

[16] M. I. DJOMI, R. MUNADI, and R. M. NEGARA, "Analisis Performansi Layanan FTP danVideo Streaming berbasis Network Function Virtualization menggunakan Docker Containers," *ELKOMIKA J. Tek. Energi Elektr. Tek. Telekomun. Tek. Elektron.*, vol. 6, no. 2, p. 180, 2018.

[17] K. Maurice, C., Weber, M., Schwarz, M., Giner, L., Gruss, D., Boano, C. A., ... & Römer, "Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud," *NDSS*, vol. 17, pp. 8–11, 2017.

 https://doi.org/10.30743/infotekjar.v5i1.2758