Performance Comparison of SSH Libraries

Sonali Karki[1] , Dr. Kiran V[2]

Electronics and Communication
Dept.[1,2] ,R.V College of
Engineering , Bangalore, India

kiranv@rvce.edu.in[1], sonaliajitkarki.ec17@vce.edu.in[2]

Abstract: *The business industry is evolving. Enterprises have begun a digital transformation path, adopting innovative technologies that enable them to move quickly and change how they cooperate, lowering costs and improving productivity. However, as a result of these technologies, the conventional perimeter has evaporated, and identification has become the new line of defense. New security concerns necessitate modern security measures. Passwords are no longer appropriate for authenticating privileged access to mission-critical assets. Passwords are notorious for being insecure, causing weariness and giving the user a false sense of security. Enterprises must use password-less solutions, which is where SSH key-based authentication comes in.*
The Python language's numerous applications are the consequence of a mixture of traits that offer this language an advantage over others. Some of the advantages of programming with Python are as follows: To enable easy communication between Python and other systems, Python Package Index (PyPI) is used. The package consists of a variety of modules developed by third party developers. It also has the benefit of being an Open Source and Community Development language, as well as having substantial Support Libraries.
There are multiple SSH libraries in python and this paper focuses on each of their pros and cons as well as the time taken for each of them to perform.

Keywords: SSH, Python, Netmiko, Paramiko, SSH2-Python

1. INTRODUCTION

To enable clients to safely establish a connection to a server present in a remote location, a network protocol named SSH (Secure Shell or Secure Socket Shell) was developed. As well as giving secure organization administrations, SSH represents a bunch of devices that execute the SSH convention. Secure Shell gives solid passwords and public key verification and moves scrambled information between two PCs that interface over an open organization like the Internet. As well as giving solid encryption, network overseers use SSH widely to distantly control frameworks and applications, permitting them to sign in to different PCs over the organization, run orders, and move records starting with one PC then onto the next. simulation. Few other notable examples of the use of ssh are to connect to a network device situated in a remote location while connected to a server or remotely executing a graphic session on a Windows XP server. Windows X system graphics session. The SSH server listens on standard port 22 of the Transmission Management Protocol (TCP) by default.

The authors of [1] demonstrates several new techniques for configuring network devices using automation, reducing the time required for equipment setup and making maintenance simpler. It also enhances network security by detecting and repairing security flaws, as well as network reliability.

In [2], for system administrators and end users interested in using this highly common TCP/IP-based approach, this revised book extensively discusses the new SSH-2 protocol. SSH, The Secure Shell: The Definitive Guide, written for a broad technical audience, includes many SSH solutions for various operating systems and programming environments.

The authors of [3] reviews development in SSH main management to date, identifies ongoing issues, and outlines the conditions for a long-term solution. Proposals from the science community are being sought to solve the problem.

The network protocol which has cryptography has a major concept and the tools required to execute this protocol together make up SSH. A client server model is the heart of the SSH. The concept is that a client can safely connect to a server present in a remote location during and after the session. Few application which yield good results when SSH is applied is during file transfer from one workstation to another and during terminal

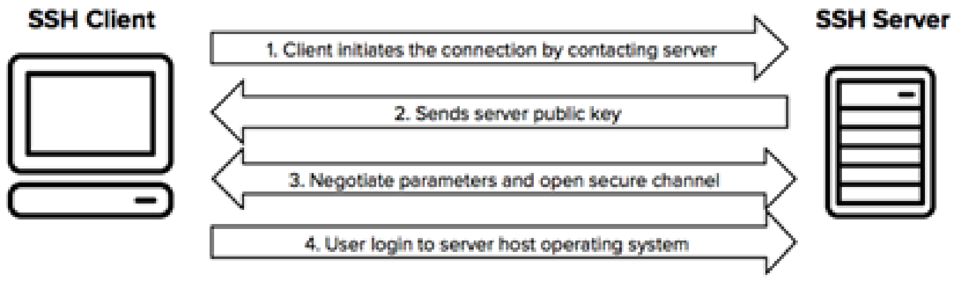
2. FUNDAMENTAL THEORY

2.1 SSH

The main motivation to develop Secure Shell was to ensure an alternate method to the existing login applications such as remote shell and remote login. These applications had a numerous security flaws and hence could not be trusted after a certain point. While TelNet is still being used today, it too has a lot of security issues. To counter the lack of security features in the aforementioned application SSH was developed. It contains enhanced security while retaining the easy-to-use login and logout feature, initializing and disconnecting sessions which need to run on remote servers. Due to its easy-to-use and user-friendly interface, SSH was able to replace RCP and FTP (File Transfer Protocol).

A client-server based protocol is a protocol where one system, the client, can seeks information from another system which is usual termed as a server. When an SSH connection is established between system A (client) and system B (server), system A is able to control system B like a local computer. As the server receives client requests in bulk, to manage all the requests, the server listens to a specify port which is allocated solely for this purpose. A client is able to initialize the connection using only this port. SSH focuses on authentication to enable a safe connection hence the client must have the required permissions after which it can send SSH commands as shown in Figure 1.

Figure 1. SSH Client-Server


SSH Client
SSH Server
1. Client initiates the connection by contacting server
2. Sends server public key
3. Negotiate parameters and open secure channel
4. User login to server host operating system

The first step in connecting to a remote host using SSH is to use the following command:

```
ssh username@remoteserver.com
```

In this example, remoteserver.com is a server which can be connected to by a client to gain information. The Client can connect to this server using SSH, assuming that SSH has been enabled on the server. If this is so then the client needs a username to be able to successfully connect to the server. This is the authentication mechanism added to ensure only authorized personnel have access to the server. The SSH at the begin of the command-line-interface command informs the client, username, to bind to the server, remoteserver.com . The user will be prompted with the remote host's public key fingerprint. If the client has not connected to this server in the past, ie it’s the first time then the client then there is a smooth connection established as the username is used to authenticate the client.

2.2 timeit() Library

Python timeit() is a method in the Python library that calculates the length of time it takes for a given code snippet to execute. The Python library iterates over the code statement a million times and returns the shortest time from the collection of code snippets. Python's timeit() method is useful for testing the output of a programme. Instead of this method, a simple time package is used, which subtracts the resources required pre and post code execution. This methodology, however, is inaccurate since a background process may be executing at the moment of code execution, resulting in significant changes in the processing time of short source code. timeit runs the source code hundreds of thousands of times to provide the highest statistically meaningful assessment of code processing time.

3. PERFORMANCE FEATURES FOR SSH LIBRARY

The SSH libraries' specifications included good consistency, scalability, Python 3 support, and support for the standard authentication and connection models used by the majority of organisations to connect to network devices

3.1 Asynchronous

Asynchronous processing is well suited to network operation. Asynchronous programming became simple in Python 3, particularly starting with Python 3.5. The readability and ease of use of the async mode only improved in subsequent updates, all the way up to Python 3.7, and several new libraries sprung up that did asynchronous versions of popular use cases including REST and SSH. Concurrency can be useful even with a single core, while parallel execution is necessarily dependent on multiple cores, according to a simple model. To put it another way, every task's execution is divided into two parts: CPU and I/O.

When a task performs I/O, it usually waits a long time for the IO to complete and the results to be returned. Waiting for I/O can be exemplified by disc reads or network communication. Other tasks may be scheduled to run while the task is pending. Concurrency is described as any application that can take advantage of this waiting time to work on other tasks. When a task is split into several individual pieces for parallel execution, each piece runs on multiple cores at the same time. Performing a typical operation on multiple elements in a list is an example of a task that lends itself to parallelization.

3.2 Authentication and Connection Method Support

Connecting to network devices in enterprises and most organizations (except hyperscalars who have the ability to put together a good public key infrastructure to use certificate-based authentication), requires support for at least the following features:

1. Excluding Host Key Files
2. Supporting Private Key Files rather than passwords, as well as a private key filewith a password.
3. Compatibility with jump host
4. Compatibility with ssh configuration files.
5. Compatibility with ssh-agent

3.3 SSH and Network Devices

When using SSH to bind to networking devices, they are infamous for not functioning like proper shells. The main issue is how configuration commands interact with network devices. Shells for network devices are context-sensitive, particularly when it comes to configuration. To assign an IP address to an interface one must first run the command "configure," then "interface Ethernet 1/1." As a result, an SSH library designed to facilitate interacting with network devices would be extremely beneficial in this regard.

4. TYPES OF SSH LIBRARIES

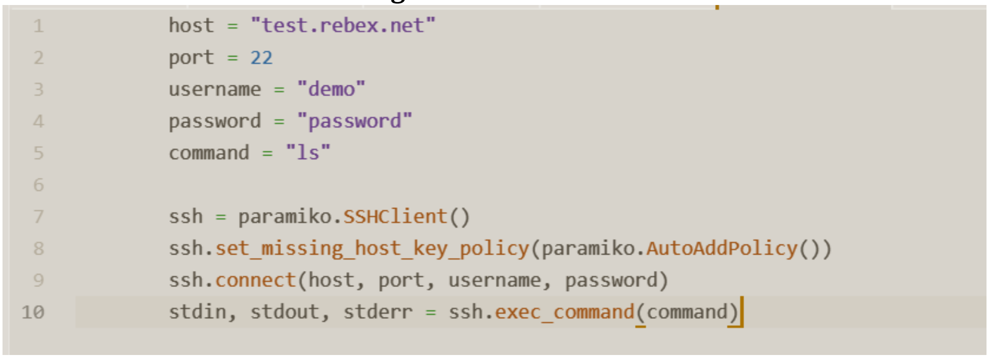
There are a multitude of ssh supporting libraries. These 4 libraries were chosen based on their timely update, timely bug and a large community support.

4.1 Paramiko

"Paramiko is a Python (2.7, 3.4+) implementation of the SSHv2 protocol that provides both client and server functionality."

Paramiko is a python library for interacting with SSH. The paramiko model provides both client(the system which requests for pages from a remote server, AKA user) and server(the system which holds information needed by a user. Usually placed in a remote location hosting multiple websites in one machine) features. It is an example of a library in Python which provides a conception of the SSHv2 protocol. As a client, you can log in with a password or a key, and as a server, you can monitor which users have access and which platforms they can use.

Figure 2. Paramiko

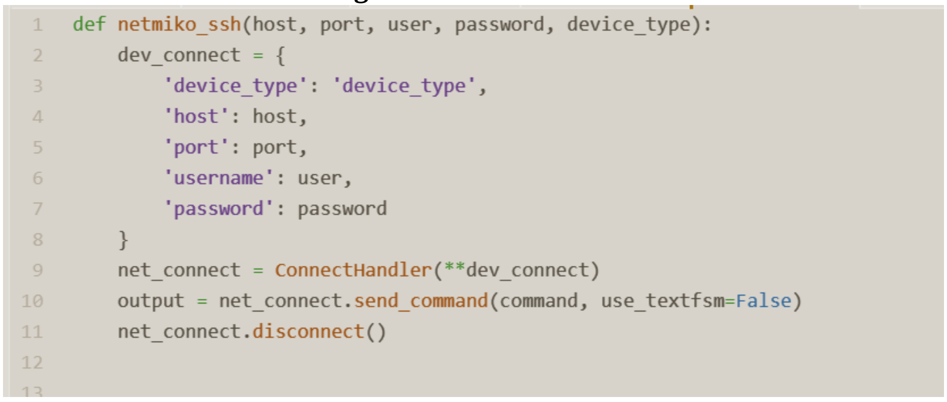


```
host = "test.rebex.net"
port = 22
username = "demo"
password = "password"
command = "ls"

ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh.connect(host, port, username, password)
stdin, stdout, stderr = ssh.exec_command(command)
```

4.2 Netmiko

Netmiko is a common alternative to Paramiko, but there are a few key differences: device support and performance. You'll come across a variety of device types while operating on real-world networks. As a result, you'll need a dependable tool to assist you in automating the process. Paramiko supports a small range of devices which results in failure in connection. In addition, it is much slower than Netmiko.

Figure 3. Netmiko

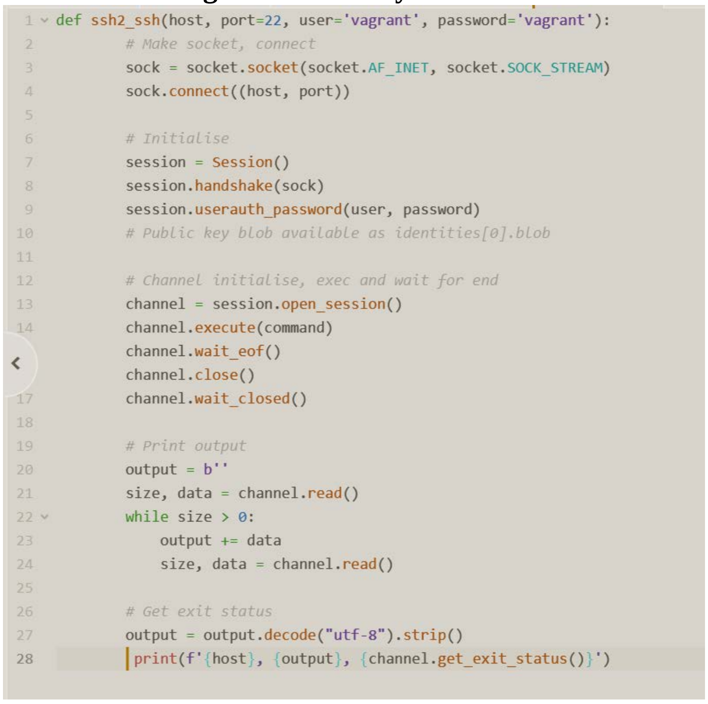


```
def netmiko_ssh(host, port, user, password, device_type):
    dev_connect = {
        'device_type': 'device_type',
        'host': host,
        'port': port,
        'username': user,
        'password': password
    }
    net_connect = ConnectHandler(**dev_connect)
    output = net_connect.send_command(command, use_textfsm=False)
    net_connect.disconnect()
```

4.3 SSH2-Python

SSH2-python is a Python binding for libssh2 and is a super fast SSH2 protocol library. The library is designed to be a thin wrapper of libssh2, implemented in Cython, with as little overhead as possible and, conversely, as high performance as possible. It has no dependencies and includes libssh2 in the Linux, OSX, and Windows binary wheels.

Figure 4. SSH2-Python

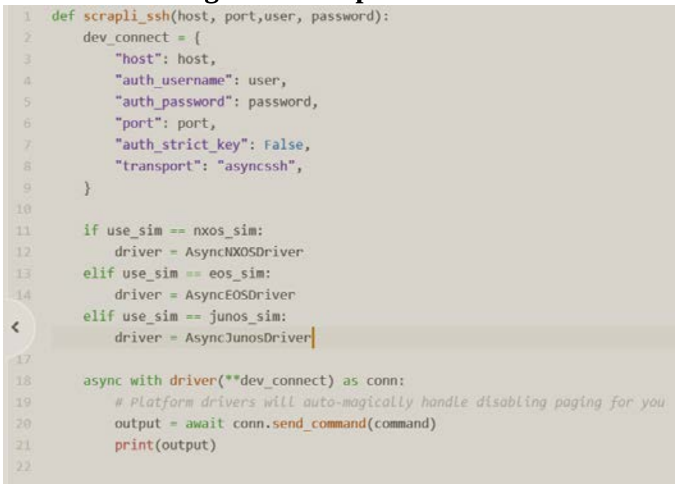


```python
def ssh2_ssh(host, port=22, user='vagrant', password='vagrant'):
        # Make socket, connect
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.connect((host, port))

        # Initialise
        session = Session()
        session.handshake(sock)
        session.userauth_password(user, password)
        # Public key blob available as identities[0].blob

        # Channel initialise, exec and wait for end
        channel = session.open_session()
        channel.execute(command)
        channel.wait_eof()
        channel.close()
        channel.wait_closed()

        # Print output
        output = b''
        size, data = channel.read()
        while size > 0:
            output += data
            size, data = channel.read()

        # Get exit status
        output = output.decode("utf-8").strip()
        print(f'{host}, {output}, {channel.get_exit_status()}')
```

4.4 Scrapli

Scrapli is a wrapper for the paramiko, asyncssh, and ssh2 SSH modules, rather than an SSH library in and of itself. It connects to network devices using both a synchronous and asynchronous SSH connections. Scrapli, on the other hand, aims to make it easy to connect to network devices and issue configuration commands, unlike asyncssh, which doesn't have any additional support for network devices. Scrapli, on the other hand, lacks support for connecting to Linux-based systems such as Cumulus and SONIC, as well as servers, and it supports even fewer devices than Netmiko.

Figure 5. Scrapli

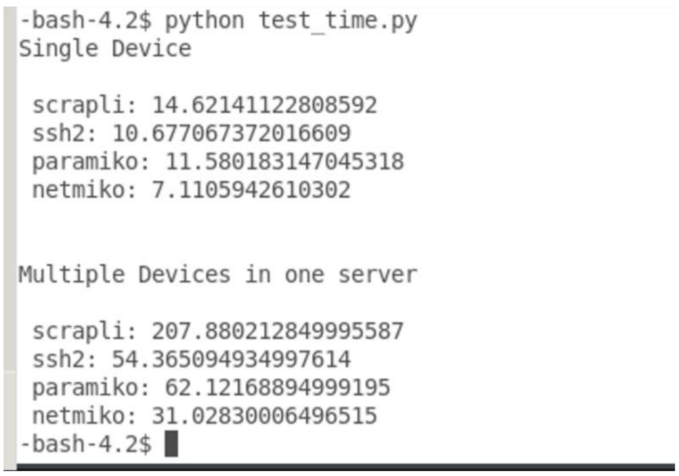


```python
def scrapli_ssh(host, port,user, password):
    dev_connect = {
        "host": host,
        "auth_username": user,
        "auth_password": password,
        "port": port,
        "auth_strict_key": False,
        "transport": "asyncssh",
    }

    if use_sim == nxos_sim:
        driver = AsyncNXOSDriver
    elif use_sim == eos_sim:
        driver = AsyncEOSDriver
    elif use_sim == junos_sim:
        driver = AsyncJunosDriver

    async with driver(**dev_connect) as conn:
        # Platform drivers will auto-magically handle disabling paging for you
        output = await conn.send_command(command)
        print(output)
```

5. Results

The libraries were tested for 2 scenarios. The first scenario has only one device in a server whereas the second one has multiple device per server which is usually the case. The time of execution was mapped for each of the libraries with Netmiko performing the fastest in both cases.

Figure 6. Results



```
-bash-4.2$ python test_time.py
Single Device

 scrapli: 14.62141122808592
 ssh2: 10.677067372016609
 paramiko: 11.580183147045318
 netmiko: 7.1105942610302


Multiple Devices in one server

 scrapli: 207.880212849995587
 ssh2: 54.365094934997614
 paramiko: 62.12168894999195
 netmiko: 31.02830006496515
-bash-4.2$ 
```

Scrapli was dismissed as it not only performed the slowest, it does not support asynchronous polling. While Paramiko and ssh2-python came in close to Netmiko, 2 errors were encountered.

SSh2-python supported only 9 of the devices. If a library does not support a device, connection to this device is not possible. This issue was faced in many servers as each server has a variety of devices.

An important feature of the library is being able to configure the router. The Management IP is one such important configuration.

The interface which is a primary line for the controller is the Mgmt interface aka management interface. This is the go-to interface for all activities such as accessing networks connected to businesses. A common example of this is the AAAA servers. Control and Provisioning of Wireless Access Points aka CAPWAP is also accessed using the Mgmt interface. The interface also finds itself being used in controller_to_access_point communications. Traffic Tunnelling and intercontroller mobility signalling are also applications of Mgmt interface.

A common method of accessing the graphical user interface for the controller is to use the browser. The Uniform Resource Locator bar on the browser is key. The Mgmt console Internet Protocol can be inserted into the search bar.A default support found on the Mgmt GUI is management of the AP. Every controller needs 2 essential components. One is the interface for the Mgmt to control communications related to inter-controller activities and a manger for the AP.

REFERENCES

10.1 Journal Articles

[1] P. Mihˇailˇa, T. Balan, R. Curpen, and F. Sandu, "Network automation andabstraction using python programming methods,"MACRo 2015, vol. 2,10 2017.
[2] D. Barrett, R. Silverman, and R. Byrnes, "Ssh, the secure shell: Thedefinitive guide," 01 2005.
[3] T. Ylonen, "Ssh key management challenges and requirements," in201910th IFIP International Conference on New Technologies, Mobility andSecurity (NTMS), 2019, pp. 1–5.
[4] T. Yl¨onen. (1995). "Ssh protocol," [Online]. Available:https://ssh.com
[5] J. Forcier. (2009). "Welcome to paramiko's documentation!" [Online]. Available: http://docs.paramiko.org/en/stable/index.html.
[6] K. Byers. (2014). "Module netmiko," [Online]. Available: https : / / ktbyers . github.io/netmiko/docs/netmiko/index.html.
[7] Panos. (2009). "Ssh2-python documentation," [Online]. Available: https://ssh2- python.readthedocs.io/en/latest/
[8] C. Montanari. (2018). "Scrapli," [Online]. Available: https : / / carlmontanari . github.io/scrapli