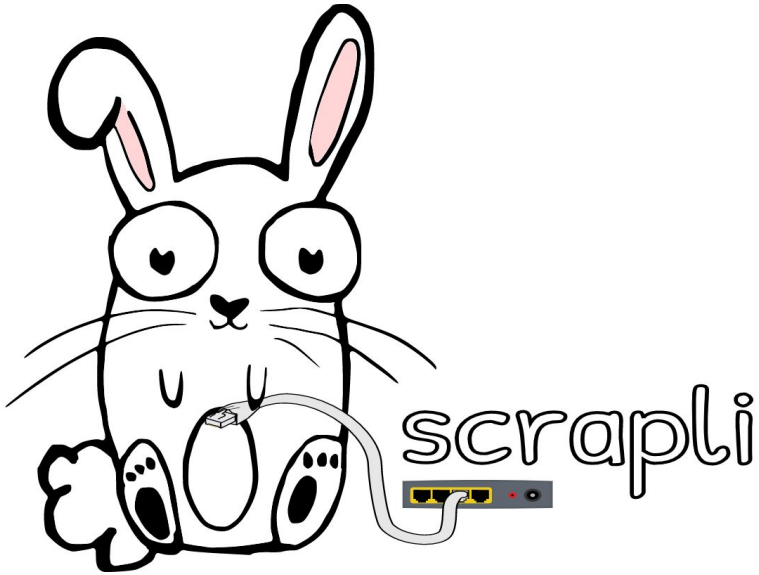
Network Automation



With Scrapli

Who am I ?

20+ years experience

Network Operator and Engineer

Network Architect

Software Developer

Product Developer & Language explorer

Technology Writer

Network Automation

Claus Töpke www.linkedin.com/in/claus-topke claus@telcomanager.com

Disclaimer

Most of the slides are for reference, and we are going to skim.

No intention to give Software Engineering or Coding guidance

Intended for NEs, DevOps and SDEs

No affiliation to any of the language creators or maintainers

A bit shallow but deep enough to offer insights for comparing

Based on years of experience and recent research

Some images were taken from Internet, URLs of original included when possible

An introduction to Scrapli

Scrapli History

Originally created in 2020 for Python by Carl Montanari

Desire to create a tool that addressed some of the limitations he encountered with existing Python libraries for network automation, such as Netmiko and Paramiko.

Montanari sought to build a solution that was faster and more adaptable to different networking environments.

https://github.com/carlmontanari/scrapli

Scrapli History

Was designed from the start with a focus on performance, leveraging Python's **asyncio** for concurrency, which allows it to handle multiple connections with small memory footprint.

This makes it particularly useful for large-scale network automation projects where performance and efficiency are critical.

Scrapli History

The name **scrapli** is a combination of the words **scrape** and **cli**, "*scraping the command-line interface*" ☺

Showing the library's primary function of scraping data from the command-line interfaces of network devices.

Scrapli Library

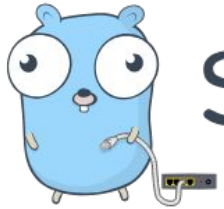
Written in Python

Has MIT license 👍

2 Open Issues (74 closed)
546 stars
54 forks
37 Releases
361 Dependents https://github.com/carlmontanari/scrapli/pulse
13 Contributors

Scrapli world

Has GO implementation (2021/2022)
https://github.com/scrapli/scrapligo
2021/2022
Growing fast has 17 contributors



Has Rust implementation(2023)
https://github.com/scrapli/scraplirs
Just released and is not in crates.io

Scrapli world

Few plugins at: github.com/scrapli

Including, Community, Netconf, Replay, Cfg and Paramiko(windows).

Also Nornir plugin: github.com/scrapli/nornir_scrapli

Problems with Python for Large Networks

Automation problems for large networks

- Python is interpreted
  - CPython (most used) interprets the Python bytecode
- Python has GIL (Global Interpreter Lock)
  - No multi-threading
- Python has low CPU performance
- Python has bigger memory footprint
- Deployment and dependencies knightmare

Python Global Interpreter Lock (GIL)

The Global Interpreter Lock (GIL) in Python ensures thread safety by restricting Python bytecode execution to one thread at a time, preventing memory access conflicts.

Major conflict is reference counting for memory management, which involves assigning a reference count variable to objects created in Python.

The GIL in Python allows only one thread to run Python bytecode at a time, ensuring thread safety and simplifying memory management to prevent conflicts when multiple threads accessing objects.

realpython.com/python-gil

Python Global Interpreter Lock (GIL)

```
NUMBER = 1_000_000_000  # 1 billion


def flipsum(args):
    start, end = args
    flipsum = 0
    for n in range(end, start - 1, -1):
        if n % 2 == 0:
            flipsum += n
        else:
            flipsum -= n
    return flipsum
```

```
time -f "%MKB %P %e" python3.12 ./flipsum.py
Flipsum 1000000000
Result = 500000000
Took 79.184543s
10144KB 99% 79.19
```

Python Global Interpreter Lock (GIL)

```
from threading import Thread


def parallel_flip_sum(num_threads):
    chunk_size = NUMBER // num_threads
    ranges = [(i * chunk_size + 1, (i + 1) * chunk_size) for i in range(num_threads)]
    results = []
    threads = []


    for range_tuple in ranges:
        thread = Thread(target=flip_sum_chunk, args=(range_tuple, resu
        threads.append(thread)
        thread.start()
```

```
time -f "%MKB %P %e" python3.12 ./flipsum-threads.py
Flipsum 1000000000 with 4 threads
Result = 500000000
Took 82.527606s
11308KB 100% 82.54
```

Python Global Interpreter Lock (GIL)

```
from multiprocessing import Pool

def parallel_flip_sum(num_processes):
    chunk_size = NUMBER // num_processes
    ranges = [(i * chunk_size + 1, (i + 1) * chunk_size) for i in range(num_processes)]

    with Pool(num_processes) as pool:
        results = pool.map(flip_sum_chunk, ranges)

    return sum(results)

if __name__ == "__main__":
    num_procs = 4

    print("Flipsum {} with {} processes ".format(NUMBER, num_procs))
    start_time = time.time()
    result = parallel_flip_sum(num_procs)
    end_time = time.time()

    print("Result =", result)
    print("Took {:f}s".format(end_time - start_time))
```

```
time -f "%MKB %P %e" python3.12 ./flipsum-multiprc.py

Flipsum 1000000000 with 4 processes

Result = 500000000

Took 20.332754s

17020KB 392% 20.36
```

CPU-bound and I/O-bound tasks

CPU-bound tasks benefit from parallel execution across multiple cores.

I/O-bound tasks benefit from concurrency to minimize idle time during I/O operations, including Network access.

Network automation tasks are mostly I/O-bound, however large networks might impose CPU-bound due to large amount of data to scrape.

Python bytecode

docs.python.org/3/library/dis.html

```
$ python3.12
Python 3.12.0 (main, Oct 21 2023, 17:42:12) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> def sayHello():
...     print("Hello people!");
...
>>> import dis
>>> dis.dis(sayHello)
  1           0 RESUME                   0

  2           2 LOAD_GLOBAL              1 (NULL + print)
             12 LOAD_CONST               1 ('Hello people!')
             14 CALL                     1
             22 POP_TOP
             24 RETURN_CONST             0 (None)
>>>
```

Python bytecode

```
% python3.12 -m py_compile countdown.py

% ls -l
% ls -l
total 7768
drwxr-xr-x  3 claustopke  staff       96 Nov 30 14:54 __pycache__
-rw-rw-r--  1 claustopke  staff      229 Nov 30 13:09 countdown.py

% ls -l __pycache__
total 8
-rw-r--r--  1 claustopke  staff  598 Nov 30 14:54 countdown.cpython-312.pyc
```

Python dependency files

command: `strace -e openat python3.8 <program.py>` # with paramiko & pysnmp)

```
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/x509/__pycache__/extensions.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/__pycache__/ipaddress.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/x509/__pycache__/general_name.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/email/__pycache__/__init__.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/email", O_RDONLY|O_NONBLOCK|O_CLOEXEC|O_DIRECTORY) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/email/__pycache__/utils.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/__pycache__/random.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/__pycache__/bisect.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/email/__pycache__/_parseaddr.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/__pycache__/calendar.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/__pycache__/locale.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/email/__pycache__/charset.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/email/__pycache__/base64mime.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/email/__pycache__/quoprimime.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/email/__pycache__/errors.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/email/__pycache__/encoders.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/__pycache__/quopri.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/x509/__pycache__/name.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/x509/__pycache__/oid.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/backends/openssl/__pycache__/aead.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/backends/openssl/__pycache__/ciphers.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/backends/openssl/__pycache__/cmac.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/backends/openssl/__pycache__/ec.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/backends/openssl/__pycache__/utils.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/backends/openssl/__pycache__/rsa.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/bindings/openssl/__pycache__/__init__.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/bindings/openssl", O_RDONLY|O_NONBLOCK|O_CLOEXEC|O_DIRECTORY) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/bindings/openssl/__pycache__/binding.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/bindings/openssl/__pycache__/_conditional.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/opt/pyca/cryptography/openssl/openssl.cnf", O_RDONLY) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/primitives/serialization/__pycache__/pkcs12.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/paramiko/__pycache__/client.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/paramiko/__pycache__/agent.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/__pycache__/tempfile.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/__pycache__/shutil.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/__pycache__/bz2.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/__pycache__/_compression.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/lib-dynload/_bz2.cpython-38-x86_64-linux-gnu.so", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libbz2.so.1.0", O_RDONLY|O_CLOEXEC) = 3
.
.
.
```

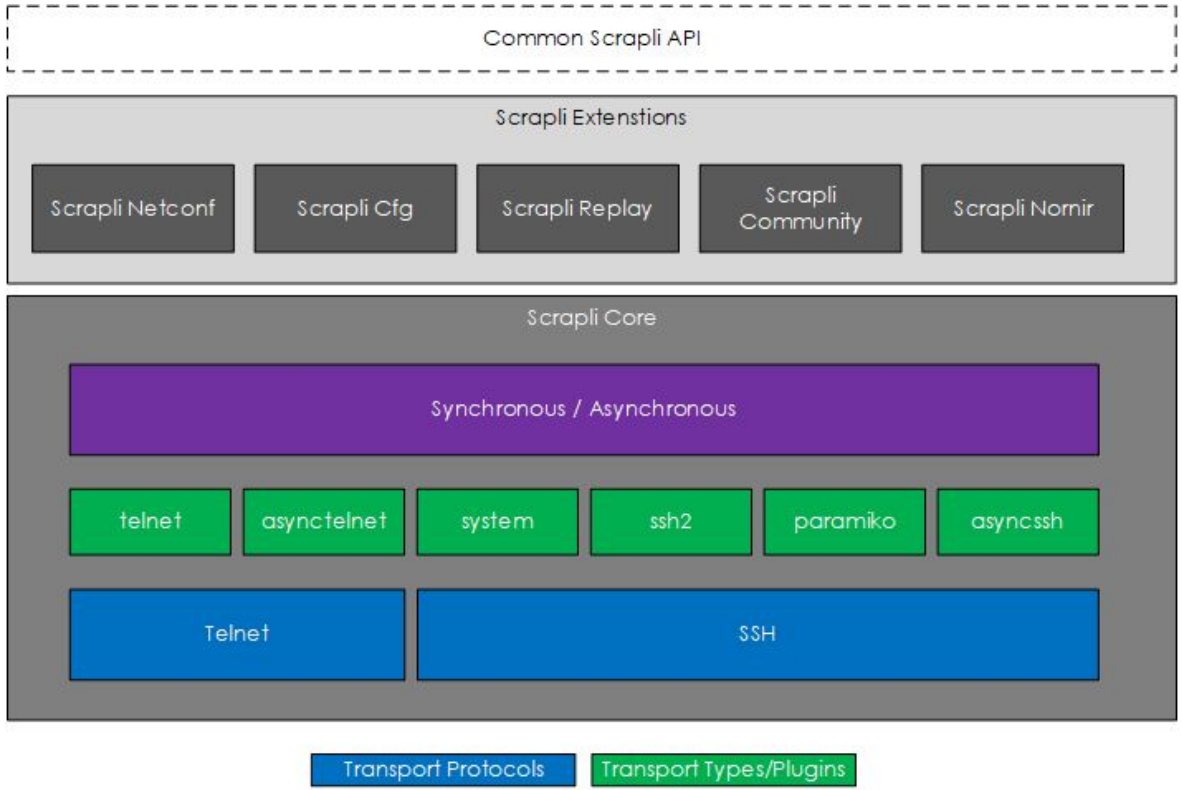
269 files !!

Solving problems for large networks

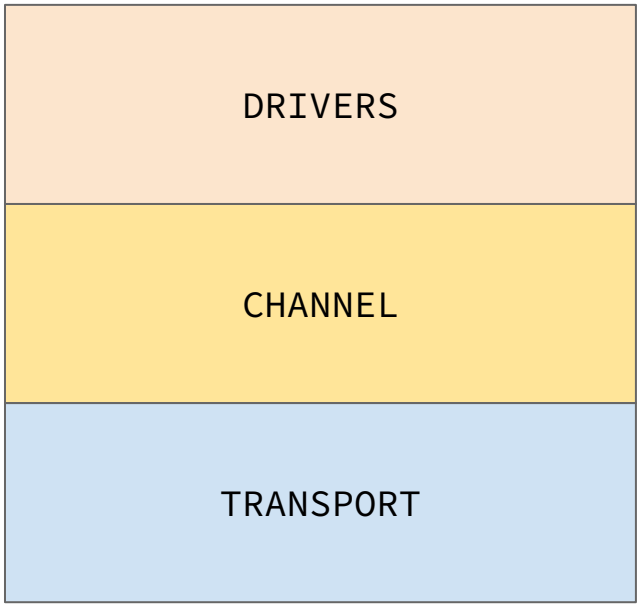
- Python is interpreted => JIT with PyPy
- Python has GIL (Global Interpreter Lock) => Using Asyncio
  - No multi-threading
- Python has low CPU performance => Using PyPy
- Python has bigger memory footprint => Using Asyncio
- Deployment and dependencies knightmare => Using Virtenv or containers

Scrapli architecture

Scrapli structure



Scrapli Core



Scrapli Transport

- **system**: wrapper around SSH/System available SSH binary
- **telnet**: Python standard library telnetlib
- **asynctelnet**: Python standard library asyncio stream
- **asyncssh**: wrapper around asyncssh library
- **ssh2**: wrapper around ssh2-python library (fast)
- **paramiko**: wrapper around paramiko library

Scrapli Channel

The "channel" acts as a vital link between transports and drivers, facilitating tasks such as **prompt detection**, **command execution**, and **device interaction**, independent of the chosen transport's type (asynchronous or synchronous). It operates by interacting with the transport layer for each connection.

Scrapli Driver

The "driver," merges transport and channel to initiate scrapli objects.

The allow to understand the context of the device being connected, using **`IOSXEDriver`** targeting Cisco IOS-XE context, like privileged enable prompt. Other devices not commonly used can use **`GenericDriver`** with less context.

Ideally, a driver should understand the system that is being connected to interpret privileged and other features, like saving or checking config.

Scrapli Driver

The core drivers are designed for specific platforms such as Arista EOS, Cisco IOS-XE, IOS-XR, NX-OS, and Juniper Junos, each available in synchronous and asynchronous.

The Scrapli Community extends this support to other devices and platforms, like Fortinet, Mikrotik, Siemens, Nokia SR OS (SROS), Palo Alto Networks PAN-OS, Raisecom ROS, Ruckus FastIron and Unleashed, Siemens ROXII, Versa FlexVNF, and VyOS, among others.

https://scrapli.github.io/scrapli_community/reference/

Scrapli Driver

async_driver

scrapli_community.fortinet.fortios.async_driver

AsyncFortinetFortiOSDriver

Bases: `AsyncGenericDriver`

Fortinet FortiOS platform class

Attributes:

Name	Type	Description
`_vdoms_enabled`	`bool`	True when device is in multi-VDOM mode
`_vdom_list`	`List[str]`	list of VDOMs read from device when needed
`_original_console`	`str`	more|standard, read from device in order to restore it by cleanup

Source code in `fortinet/fortios/async_driver.py`

`cleanup_session() -> None` async

Restore paging if necessary

Source code in `fortinet/fortios/async_driver.py`

`context(context: str) -> Union[None, str]` async

Claus Töpke

Installation and usage

Scrapli Install

From pip:
```
pip install scrapli
```

From Source:
```
git clone https://github.com/carlmontanari/scrapli
cd scrapli
python setup.py install
```

For all extras and plugins:
```
pip install scrapli[full]
```

Scrapli Usage Basics

```
from scrapli.driver.core import IOSXEDriver

my_device = {
    "host": "172.18.0.11",
    "auth_username": "scrapli",
    "auth_password": "scrapli",
    "auth_strict_key": False,
}

with IOSXEDriver(**my_device) as conn:
    response = conn.send_command("show version")
```

Scrapli Usage Basics

For SSH

```
from scrapli.driver.core import IOSXEDriver

my_device = {
    "host": "172.18.0.11",
    "auth_username": "scrapli",
    "auth_password": "scrapli",
    "auth_strict_key": False,
}

with IOSXEDriver(**my_device) as conn:
    response = conn.send_command("show version")
```

Scrapli Usage Basics

For Telnet

```
from scrapli.driver.core import IOSXEDriver

my_device = {
    "host": "172.18.0.11",
    "auth_username": "scrapli",
    "auth_password": "scrapli",
    "transport": "telnet",
}

with IOSXEDriver(**my_device) as conn:
    response = conn.send_command("show version")
```

Scrapli multiple routers (serial)

```
from scrapli.driver.core import IOSXEDriver

devices = ["10.0.0.1", "10.0.0.10", "10.0.0.11", "10.0.0.12", "10.0.0.30", "10.0.0.31",
"10.0.0.40", "10.0.0.41", "10.0.0.50", "10.0.0.51"]

common_params = {
    "auth_username": "username",
    "auth_password": "password",
    "transport": "telnet",
}

def connect_and_execute(device_ip):
    device = {"host": device_ip, **com
    with IOSXEDriver(**device) as conn:
        response = conn.send_command("show version")
        print(f"Device: {device_ip}, Response: {response.result}")

if __name__ == "__main__":
    for ip in devices:
        connect_and_execute(ip)
```

```
$ time -f "%MKB %P %e" python3
show-version-multiple.py

21192KB 11% 5.26
```

Scrapli multiple routers with multi-threading

```
def connect_and_execute(device_ip):
    device = {"host": device_ip, **common_params}
    with IOSXEDriver(**device) as conn:
        response = conn.send_command("show version")
        print(f"Device: {device_ip}, Response: {response.result}")

def run_threads():
    threads = []
    for ip in devices:
        thread = threading.Thread
        threads.append(thread)
        thread.start()

    for thread in threads:
        thread.join()

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

```
$ time -f "%MKB %P %e" python3
show-version-multiple-threading.py

23128KB 112% 1.04
```

Scrapli multiple routers with AsyncIO

```
from scrapli.driver.core import AsyncIOSXEDriver

async def connect_and_execute(device_ip):
    device = {"host": device_ip, **common_params}
    async with AsyncIOSXEDriver(**device) as conn:
        response = await conn.send_command("show version")
        print(f"Device: {device_ip}, Response: {response.result}")

async def main():
    tasks = [connect_and_execute(ip) for ip in devices]
    await asyncio.gather(*tasks)


if __name__ == "__main__":
    asyncio.run(main())
```

```
$ time -f "%MKB %P %e" python3
show-version-multiple-asyncio.py

20620KB 52% 0.37
```

ADVANCED FEATURES

Scrapli Advanced features/options

- **Platform Regex**: obtained with comms_prompt_pattern
- **on_open**: example disable CLI paging
- **on_close**: sane operations can be added
- **Diff timeouts**: socket, transport or ops
- **Privilege levels**: Specific for each driver
- **Exclusive/Private/Session**: configuration

Scrapli using Netconf

```
from scrapli_netconf.driver import NetconfDriver

device = {
    "host": "10.0.0.20",
    "auth_username": "username",
    "auth_password": "password",
    "auth_strict_key": False,
    "port": 830,  # Default Netconf port
}

with NetconfDriver(**device) as conn:
    # Getting configuration data
    response = conn.get_config(source="running")
    print(response.result)
```

Scrapli using Replay

Tools to enable easy testing of scrapli programs and to create semi-interactive SSH servers that look and feel like "real" network devices

```
@pytest.mark.scrapli_replay
def test_something_else():
    with IOSXEDriver(**MY_DEVICE) as conn:
        result = conn.send_command("show run | i hostname")
```

Scrapli using Nornir plugin

It integrates Scrapli's fast, asyncio-capable SSH and Telnet connections with Nornir's task execution model, enabling efficient and flexible network automation tasks.

```
from nornir import InitNornir
from nornir_scrapli.tasks import send_command


nr = InitNornir(config_file="nornir_data/config.yaml")
command_results = nr.run(task=send_command, command="show version")

print("send_command result:")
print(command_results["iosxe-1"].result)
```

Scrapli using CFG

scrapli_cfg makes merging or replacing device configurations over Telnet or SSH easy, all while giving you the scrapli behaviour you know and love.

```
with open("myconfig", "r") as f:
    my_config = f.read()

with Scrapli(**device) as conn:
  cfg_conn = ScrapliCfg(conn=conn)
  cfg_conn.prepare()
  cfg_conn.load_config(config=my_config, replace=True)
  diff = cfg_conn.diff_config()
  print(diff.side_by_side_diff)
  cfg_conn.commit_config()
  cfg_conn.cleanup()
```

What about Opening the can of worms with go

CONCLUSIONS

Conclusions

Not as mature as Netmiko, but growing community

Born to be faster

Plugins are growing and have interesting features

Have GO version

Have Rust version

Thank you !

Drop a message and let's talk about network automation, discuss solutions for network performance, network simulation, traffic analysis, network management, and more.

www.telcomanager.com

Network Automation Go/Python book: a.co/d/iO7iXMe

Github: github.com/brnuts

Linkedin : www.linkedin.com/in/claus-topke/

Work email: claus@telcomanager.com

Personal email: claus.topke@gmail.com