2023-04-10

HÖGSKOLAN VÄST

Implementation av Python via NETCONF och RESTCONF. En jämförelsestudie

Philip Ekström

EXAMENSARBETE
Nätverksteknik med IT-säkerhetsprogrammet, 120 hp

Institutionen för ingenjörsvetenskap, Högskolan Väst

EXAMENSARBETE

Implementation av Python via NETCONF och RESTCONF. En jämförelsestudie

Sammanfattning

Det går att hantera nätverksenheter och nätverk på olika sätt. Två av dessa sätt är RESTCONF och NETCONF. Denna undersöknings syfte är att ta reda på vilken av dessa två protokoll som är snabbast och om storlek på nätverket har påverkan på vilket att dessa protokoll som bör användas. Resultatet visar att RESTCONF är snabbare än NETCONF på ett nät med tre routrar och två switchrar. RESTCONF är även snabbare än NETCONF på ett nätverk som består av 17 routrar och fem switchrar. Både NETCONF och RESTCONF tar längre tid i nätverket med fler nätverksenheter än i nätverket med färre nätverksenheter.

Datum: 2023-04-10
Författare: Philip Ekström
Examinator: Andreas de Blanche
Handledare: Simon Olofsson, Högskolan Väst
Program: Nätverksteknik med IT-säkerhet, 120 hp
Huvudområde: Datateknik
Utbildningsnivå: Grundnivå
Kurskod: EXN300, 7,5 hp

Nyckelord: *NETCONF, RESTCONF, PYTHON*

Utgivare: Högskolan Väst, Institutionen för ingenjörsvetenskap
461 86 Trollhättan
Tel: 0520-22 30 00 Fax: 0520-22 32 99, www.hv.se

DEGREE PROJECT

Implementation of Python via NETCONF and REST-CONF. A comparative study

Summary

There are different ways to manage network devices and networks. Two of the ways are RESTCONF and NETCONF. The purpose of this investigation is to find out which of these two protocols is the fastest and if the size of the network has an impact on which of these protocols should be used. The result shows that RESTCONF is faster than NETCONF on a network with three routers and two switches. RESTCONF is also faster than NETCONF on a network consisting of 17 routers and five switches. Both NETCONF and RESTCONF take more time in the network with more network devices than in the network with fewer network devices.

Date:	April 10, 2023
Author:	Philip Ekström
Examiner:	Andreas de Blanche
Advisor:	Simon Olofsson, Högskolan Väst
Programme:	Network Technology and IT-security, 120 HE credits
Main field of study:	Computer Engineering
Education level:	First cycle
Course code:	EXN300, 7.5 HE credits
Keywords:	*NETCONF, RESTCONF, PYTHON*
Publisher:	University West, Department of Engineering Science SE-461 86 Trollhättan, Sweden Phone: + 46 520 22 30 00, Fax: + 46 520 22 32 99, www.hv.se

Förord

Jag vill tacka min handledare Simon Olofsson.

Innehållsförteckning

Bilagor

1 Inledning

Det går att hantera nätverksenheter och nätverk på olika sätt. Den vanligaste metoden de senaste 30 åren har varit att använda command–line interface (CLI). När CLI används finns en större risk att det blir en felkonfiguration av den personen som administrerar nätverket än om ett automationsverkytg, som t.ex. Embedded Event Manager (EEM), används. Utöver detta tar det mycket tid att hantera ett komplext nätverk via CLI [1]. Det går även att undvika användning av CLI, och därmed felkonfiguration och tidsförlust, genom att använda application programming interface (API). För att komma åt API går det att använda protokollen Network Configuration (NETCONF) och Representational State Transfer Configuration Protocol (RESTCONF) [2]. För att veta vilket av dessa protokoll som bör användas behöver användaren veta vilket protokoll som passar för uppgiften.

1.1 Problemformulering, Syfte och Mål

En aspekt som ska beaktas när det ska bestämmas huruvida NETCONF eller RESTCONF ska användas är hur lång tid det tar att hämta konfigurationsdata och skicka ut konfigurationsdata till och från nätverksenheterna. En annan aspekt är om storleken på nätverket har betydelse för tidsåtgången vid valet av RESTCONF eller NETCONF.

Syftet med denna studie är jämföra den genomsnittliga tiden det tar för NETCONF och RESTCONF att hämta konfigurationsdata från nätverksutrustning samt skicka ut konfigurationsdata till nätverksutrustning när dessa protokoll används med Python. Studien görs för att ta reda på vilket av protokollen som tar minst tid att hämta och skicka konfigurationsdata till nätverksenheter. Studien görs också för att ta reda på om storleken på nätverket påverkar huruvida RESTCONF eller NETCONF är snabbast.

Målet med studien är att ta reda på huruvida RESTCONF eller NETCONF bör användas och när de bör användas.

När valet mellan protokollen görs ska andra aspekter, som t.ex. säkerhet och användarvänlighet, tas i beaktning men i denna undersökning mäts och jämförs endast tiden.

2 Bakgrund

Denna del ger bakgrund om YANG, XML, JSON, API, NETCONF, RESTCONF och Python.

2.1 YANG

Yet Another Next Generation (YANG) är ett datamodelleringspråk som används för att beskriva objekt som är relaterade till den data som konfigurationen består av och strukturen på data på objekten. YANG kan t.ex. användas för att beskriva routingprotokoll och portar [2] och det kan användas för konfigurering av nätverk och övervakning av nätverksdata [3]. Det är YANG som t.ex. bestämmer hur märkspråket, Extensible Markup Language (XML), ska se ut och struktureras [4].

YANG är, från början, designat för att modellera data för NETCONF men YANG kan även användas med RESTCONF [5]. YANG består generellt av olika moduler. Dessa moduler beskriver data och bestämmer vad som ska göras [3].

Ett exempel på detta är YANG–modulen ietf–interfaces som innehåller olika containrar. Containrar är en datanod som i sin tur innehåller ett antal datanoder. En container har med andra ord inte ett värde. En av dessa containrar är rw interfaces, rw står för read write. Containern rw interfaces innehåller en lista som heter rw interface* som i sin tur innehåller olika leafs. En leaf är en datanod som har ett värde med en leaf innehåller inte andra datanoder. Strukturen blir således

```
Modul: ietf-interfaces

Container: rw interfaces

list: rw interface*

leaf: enabled
```

Notera att det finns flera leaf i listan rw interface*. Varje leaf har en datatyp. I exemplet är leaf Enabeled. Enabled är boolesk, dvs. att datatypen antingen är sann eller falsk, och om porten är aktiverad är enabled sann och om porten är avaktiverad är enabled falsk [6].

2.2 XML

XML är ett textbaserat format för att representera strukturerad information. Denna information kan t.ex. vara data eller konfiguration. XML används för koda YANG–data. XML använder taggar för att identifiera element och en tagg börjar med < och slutar med >. Till exempel kan en tagg vara `<interfaces>`. Detta kallas för en starttagg. En regel för XML är att en sektion som börjar med en starttagg måste avslutas. Detta görs med en sluttagg. Sluttaggen blir i detta exempel `</interfaces>` [7].

För att identifiera YANG–containrarna i XML används namespace. Namespace används för att unikt identifiera ett element i XML. I exemplet ovan med portar finns dessa i en container som heter interfaces i en YANG–modul som heter ietf–interfaces. YANG–data i XML format blir då

```
<interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
       <interface>
            <enabled>true</enabled>
       </interface>
</interfaces>
```

där xmlns="urn:ietf:params:xml:ns:yang:ietf–interfaces" är namespace, interface är list och enabled är leaf [6].

2.3 JSON

JavaScript Object Notation (JSON) är ett lättviktigt datautbytesformat, dvs. det används för datautbyte. Det är läsbart och skrivbart för människor. Det är ursprungligen utvecklat från JavaScript. JSON är ett textbaserat format som är oberoende av programmeringsspråk.

JSON baseras på två grundstrukturer:

- En samling av nyckel– och värdepar. Beroende på programmeringsspråk har det olika namn. I Python kallas det dictionary.
- En ordnad lista av värden. Beroende på programmeringsspråk, har det olika namn. I Python kallas det lista.[8].

YANG–modeller kan representeras i JSON–format. I exemplet nedan representerar ietf–interfaces YANG–modulen, interface är list och name,description, type och enabled är leaf [9].

```
"ietf-interfaces:interface": {
            "name": "Loopback1",
            "description": "Loopback_Ett",
            "type": "iana-if-type:softwareLoopback",
            "enabled": true,
}
```

Notera att containern saknas i detta exempel då denna kan identifieras i URI, det står mer om detta under rubriken RESTCONF [10].

2.4 API

API står för application programming interface. I detta fall betyder application en mjukvara som utför en specifik uppgift samt en uppsättning regler som gör att olika mjukvaror kan kommunicera med varandra. API gör med andra ord så att två program kan utbyta information [11].

Det finns olika typer av API. Två av dessa är Northbound API och Southbound API. Northbound API är ett interface mellan controller och mjukvara för kommunikation och hantering [12]. Denna trafik kan krypteras med Transport Layer Security (TLS) [1].

Southbound API ger interface mellan controllern och dataplanet [12]. När en ändring görs i en routers eller switchs konfiguration via controller kommer denna konfigurationsdata skickas via Southbound API [1].

För att få åtkomst till data anger Representational State Transfer (REST) en uppsättning av HTTP–metoder GET, PUT, POST och DELETE. Detta gör att klienter och servrar utbyter data. REST API är stateless vilket gör att servrar inte sparar klientinformation mellan förfrågningar [11].

2.5 NETCONF

NETCONF–protokollet är en standard som kan användas för hantering och konfiguration av nätverksenheter. NETCONF är utformad för att stödja flera funktioner som krävs för effektiv konfigurationshantering som tidigare saknades i andra nätverkshanteringsprotokoll. NETCONF arbetar på datalagret och ger konfigurationen av enheten som en strukturerat text, som är i XML–format. NETCONF tillhandahåller stöd för datafiltrering, alternativ för autentisering och kan ge noteringar om händelser.

NETCONF är mer komplext och har en större struktur jämfört med RESTCONF–protokollet. Detta innebär att mer komplexa uppgifter kan utföras av NETCONF. NETCONF kan endast att använda XML–format, detta innebär att data som skickas med NETCONF måste vara i XML–format. Metoderna för att hämta och skicka konfiguration med NETCONF är GET, GET–CONFIG och EDIT–CONFIG.

Kommunikationen med NETCONF sker med av Secure Shell–protokollet (SSH) på port 830. I och med att SSH används innebär det att överförd data krypteras. Det är går att kommunicera med NETCONF med programmeringsspråk, som t.ex. Python, såväl som ett CLI med t.ex. Putty [2].

2.6 RESTCONF

RESTCONF är ett protokoll baserat på hypertext transfer protocol (HTTP). För att kryptera överförd data går det att använda hypertext transfer protocol secure (HTTPS). RESTCONF används för att tillhandahålla ett applikationsgränssnitt för åtkomst till data. Det använder av YANG för att komma åt denna data. Det går, på grund av detta, att hantera enheter via REST API. Konfiguration kan läsas, raderas och ändras via HTTP–metoderna GET, POST, PUT och DELETE.

Både JSON–formatet och XML–formatet kan användas för det data som överförs till och från enheten som hanteras. RESTCONF använder Uniform Resource Identifier (URI) för att komma åt YANG–data.

Ett exempel på URI är `https://192.168.1.1/restconf/data/ietf-interfaces:interfaces/interface=GigabitEthernet0/0/0` där 192.168.1.1 är IP–adressen till enheten, ietf–interface hänvisar till YANG–modulen interfaces är containern och interface är listan [2].

2.7 Python

Python är ett interpreterat programspråk. Utmärkande för dessa typer av programmeringsspråk är att ingen översättning görs. Datorns hårdvara tolkar de interpreterade programspråken med hjälp av en interpretator. De interpreterade programspråken kompileras inte innan de körs. Detta innebär att program skrivna i interpreterade språk är långsammare än program skrivna med kompilerade språk [13].

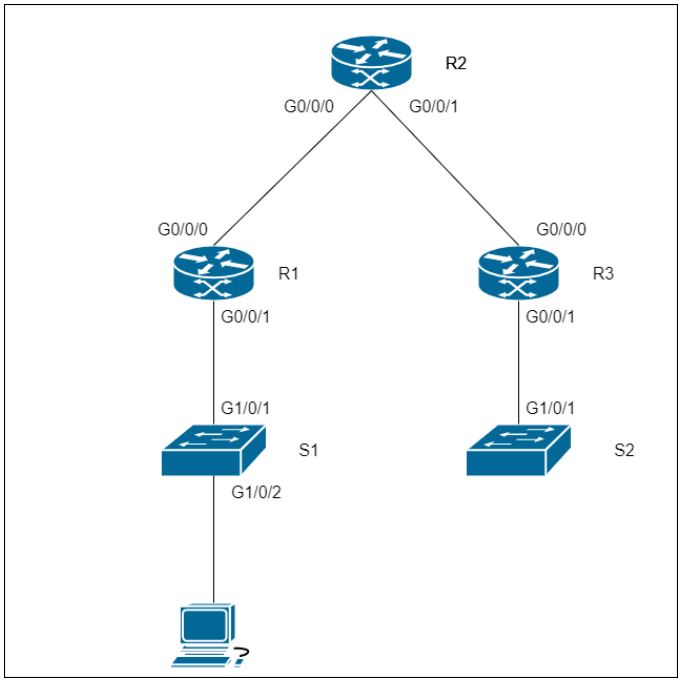
Genom att använda Pythonbiblioteket Ncclient går det att upprätta en NETCONF–session med nätverksutrustningen [4]. Ncclient ger ett API som mappar den XML–kod som NETCONF använder till den syntax som Python använder, och gör det underlättar att skriva program för att hämta eller skicka konfigurationsdata. När Ncclient importeras måste olika specificeras parametrar i Pythonprogramet. Exempel på dessa parametrar är vilken TCP–port som ska användas, vilken användare som ska logga in samt dennes lösenord [14].

Pythonbiblioteket Requests är till för att göra HTTP–förfrågningar i Python. Det förenklar att göra förfrågningar genom ett API. Requestsbiblioteket gör så att det går att använda HTTP–metoder i ett Pythonprogram och det tillåter också att, genom Pythonprogram, returnera status–koder för exempelvis en Webbserver [15].

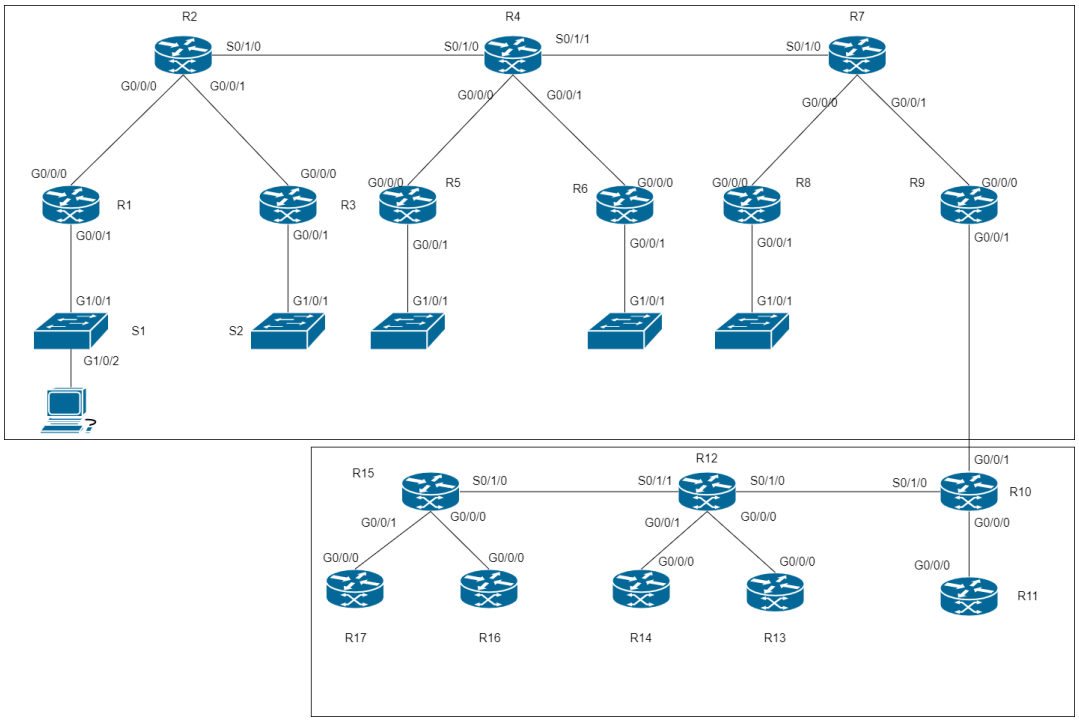
För att kunna arbeta med JSON–format behövs biblioteket json [16] och för att arbeta med XML behövs biblioteket xml.dom.minidom [17].

3 Metod

För att mäta tiden det tar för RESTCONF och NETCONF att skicka ut och hämta in konfiguration gjordes mätningar på två olika nätverk. Ett nätverk med tre routrar och två switchar i ett rum som kallas nätverk A och ett nät med 17 routrar och fem switchar som kallas nätverk B. Topologi för nätverk A finns i figur 1 och topologi för nätverk B finns i figur 2. Tiden som mäts är från att Pythonprogramet körs till det att alla enheter som Pythonprogramet avser har svarat.



Figur 1. Topologikarta över nätverk A



Figur 2. Topologikarta över nätverk B

I nätverk A och nätverk B användes routingprotokollet Open Shortest Path First (OSPF). Adresseringen för enheterna på nätverk A visas i bilaga A och adressering för nätverk B visas i bilaga B. Enheter som börjar på R är routrar, enheter som börjar på S är switchar och PC är den kontrollstation som användes för att köra Pythonprogramen.

För grundkonfiguration på switchar se bilaga E. Konfigurationen som finns i bilaga C är till Switch S1. Övriga switchar har samma konfiguration förutom hostnamn, IP–adress samt default gateway. För grundkonfiguration på routrar se bilaga F. Konfigurationen som finns i bilaga D är till Router R1. Övriga routrar har samma konfiguration förutom hostnamn, IP–adresser, router–ID i OSPF–processen samt att DHCP–tjänsten inte är i gång.

Konfigurationen på routrarna som hämtades och skickades på en och samma gång där IP–adresserna för alla routrar fanns i en lista i Pythonprogramen. Konfigurationen på switchrar som hämtades inhämtades på en och samma gång där IP–adresserna för alla switchrar fanns i en lista Pythonprogramen. I Pythonprogrammen finns en for–loop där konfigurationen som ska skickas eller hämtas itereras över respektive lista med IP–adresser till nätverksenheterna. I de Pythonprogram där konfiguration hämtas finns fyra listor:

- Lista för routrar i nätverk A
- Lista för routrar i nätverk B
- Lista för switchrar i nätverk A
- Lista för switchrar i nätverk B

I de Pythonprogram där konfiguration skickas finns två listor:

- Lista för routrar i nätverk A
- Lista för routrar i nätverk B

När mätningarna gjordes användes respektive lista för respektive nätverk och de listorna som inte användes kommenterades ut.

Pythonprogramen som användes finns i bilagorna E–N. Python modulen Time användes för att mäta tiden som det tog att hämta respektive skicka konfigurationsdata.

Konfiguration som hämtas på routrar är:

- Hostnamn på var enhet
- Vilka ethernetport som är aktiverade och har IP–adresser
- Vilka router–id samt vilka nätverk som deltar i OSPF–processen

Konfiguration som hämtas på switchar är:

- Hostnamn på var enhet
- Vilka ethernetport och SVI som är aktiverade och har IP–adresser

Konfiguration som skickas ut till routrar är:

- Sätta ingång loopbackport och sätta IP–adress på dem.
- Skapa användare med lösenord som har privilege level 5

Under laborationen hämtades informationen på routrarna och switchrarna först. Varje Pythonprogram kördes 20 gånger. Tiden som redovisas i resultatdelen är medelvärdet på de 20 testen som gjorts på varje Pythonprogram.

Efter det skapades Loopbackportarna 1–20 med olika IP–adresser med RESTCONF och sedan loopbackportarna 21–40 med olika IP–adresser med NETCONF. Varje loopbackport skapades separat vilket innebär att Pythonprogramen ändras vid varje körning.

Användarna user1–user20 skapades med RESTCONF och användarna user21–user40 skapades med NETCONF. Varje användare skapades separat vilket innebär att Pythonprogramen ändras vid varje körning.

När användarna och loopbackportarna hade skapats verifierades detta genom att kontrollera detta via CLI på varje router.

Datorn som används är Ubuntu 18.04.6 LTS på en virtuell maskin, via VirtualBox, som kördes på en dator med Windows 10. Python version 3.9 har använts för att skriva kod. De övriga närverksenheter som har använts för laboration är Cisco Catalyst 9300L med IOS–XE version 17.03.05 och Cisco 4321 med IOS–XE version 16.06.04

4 Resultat

Tabell 1 redovisar den genomsnittliga tiden att hämta hostnamn på nätverksenheterna.

Tabell 1. Genomsnittlig tid för att hämta hostnamn på enheter

Hämta hostnamn	**Genomsnittlig tid i sekunder för att hämta på nätverk A**		**Genomsnittlig tid i sekunder för att hämta på nätverk B**	
Enhet	**Routrar**	**Switchar**	**Routrar**	**Switchar**
NETCONF	2,69	1,39	19,03	4,46
RESTCONF	0,63	0,15	5,28	0,40

Tabell 2 redovisar den genomsnittliga tiden att hämta vilka ethernetportar och SVI som är aktiverade och vilken IP–adress de har.

Tabell 2. Genomsnittlig tid för att hämta aktiverade SVI och ethernetportar

Hämta ethernetport och SVI	**Genomsnittlig tid i sekunder för att hämta på nätverk A**		**Genomsnittlig tid i sekunder för att hämta på nätverk B**	
Enhet	**Routrar**	**Switchar**	**Routrar**	**Switchar**
NETCONF	2,92	1,52	20,22	4,82
RESTCONF	1,01	0,99	7,69	2,57

Tabell 3 redovisar den genomsnittliga tiden att hämta OSPF–information på routrarna.

Tabell 3. Genomsnittlig tid att hämta OSPF-information på routrar

Hämta OSPF–information på routrar	Genomsnittlig tid i sekunder för att hämta på nätverk A	Genomsnittlig tid i sekunder för att hämta på nätverk B
NETCONF	2,71	19,20
RESTCONF	1,03	7,80

Tabell 4 redovisar den genomsnittliga tiden att aktivera Loopbackport och sätta IP–adress på den.

Tabell 4. Genomsnittlig tid att sätta IP–adress och aktivera Loopbackportar på routrar.

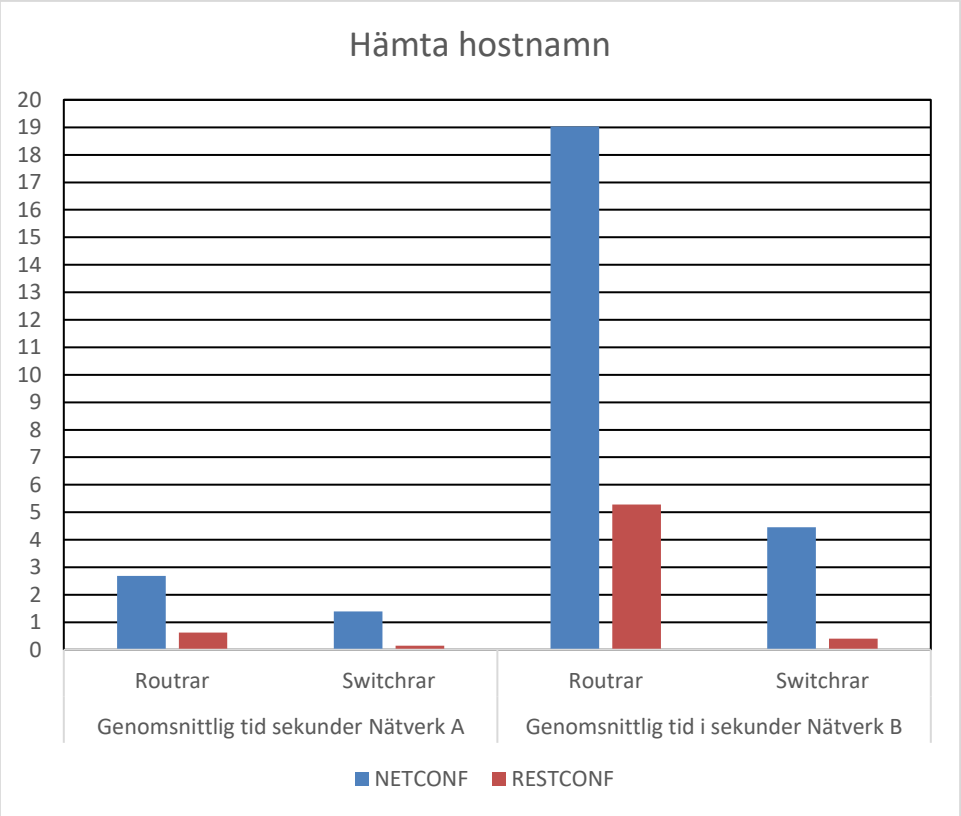
Konfigurera Loopback–port på routrar	Genomsnittlig tid i sekunder för att få svar på nätverk A	Genomsnittlig tid i sekunder för att få svar på nätverk B
NETCONF	4,98	32,60
RESTCONF	4,08	22,61

Tabell 5 redovisar den genomsnittliga tiden att skapa en användare på routrarna.

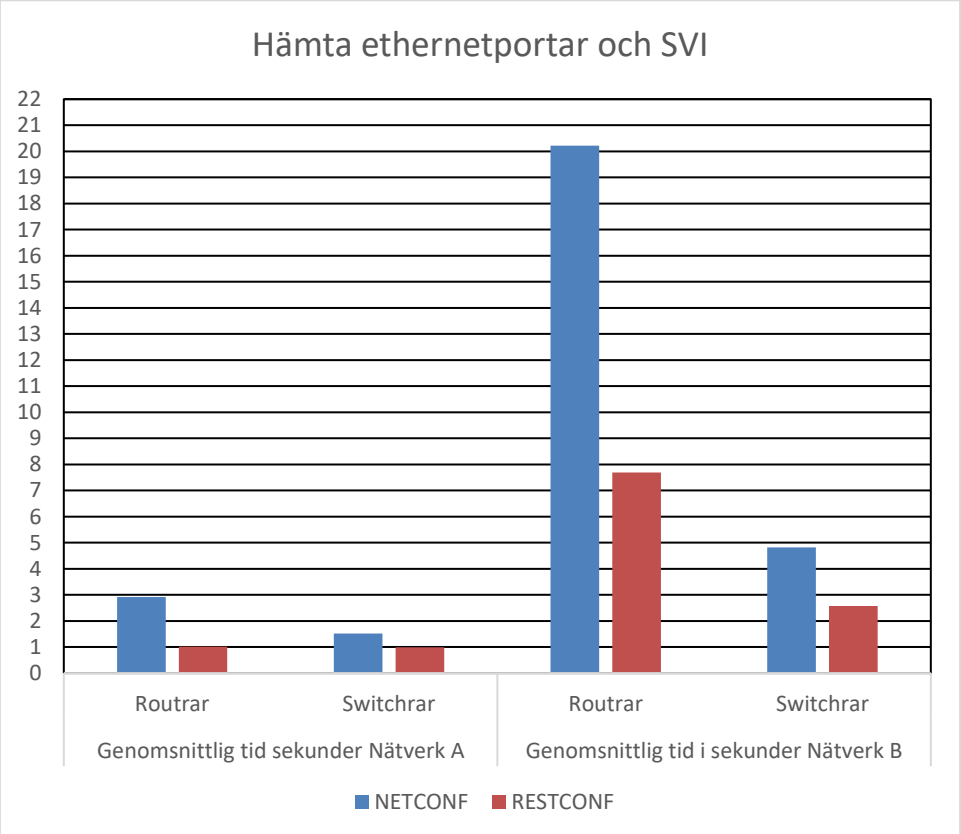
Tabell 5. Genomsnittlig tid att konfigurera användare på routrar

Konfigurera användare på routrar	Genomsnittlig tid i sekunder för att få svar på nätverk A	Genomsnittlig tid i sekunder för att få svar på nätverk B
NETCONF	3,07	21,05
RESTCONF	1,68	10,27

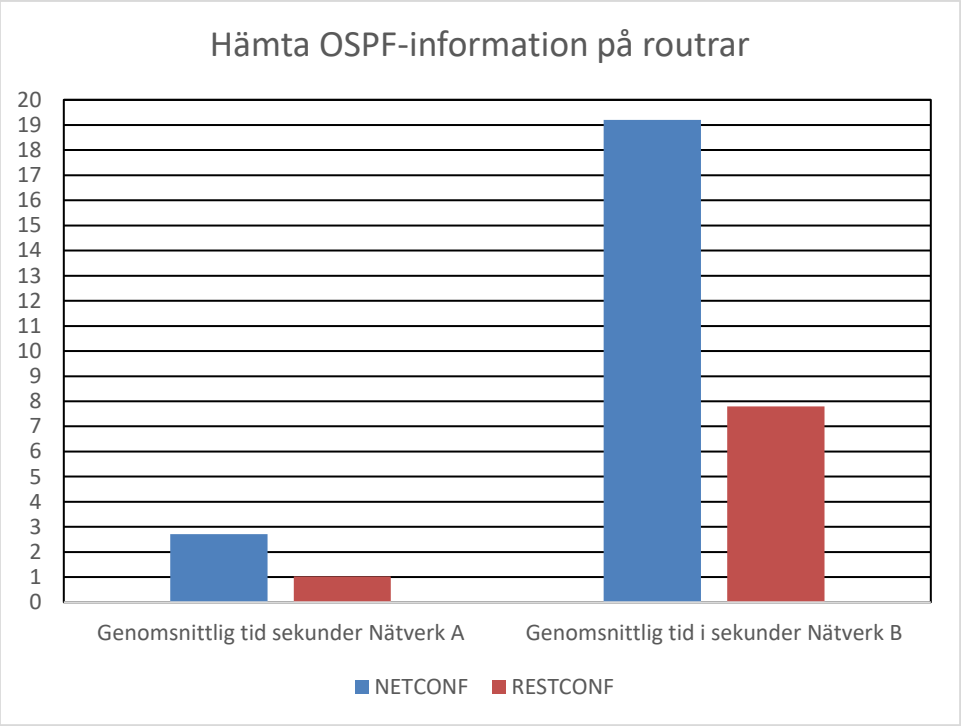
Motsvarande resultat presenteras med stapeldiagram nedan.



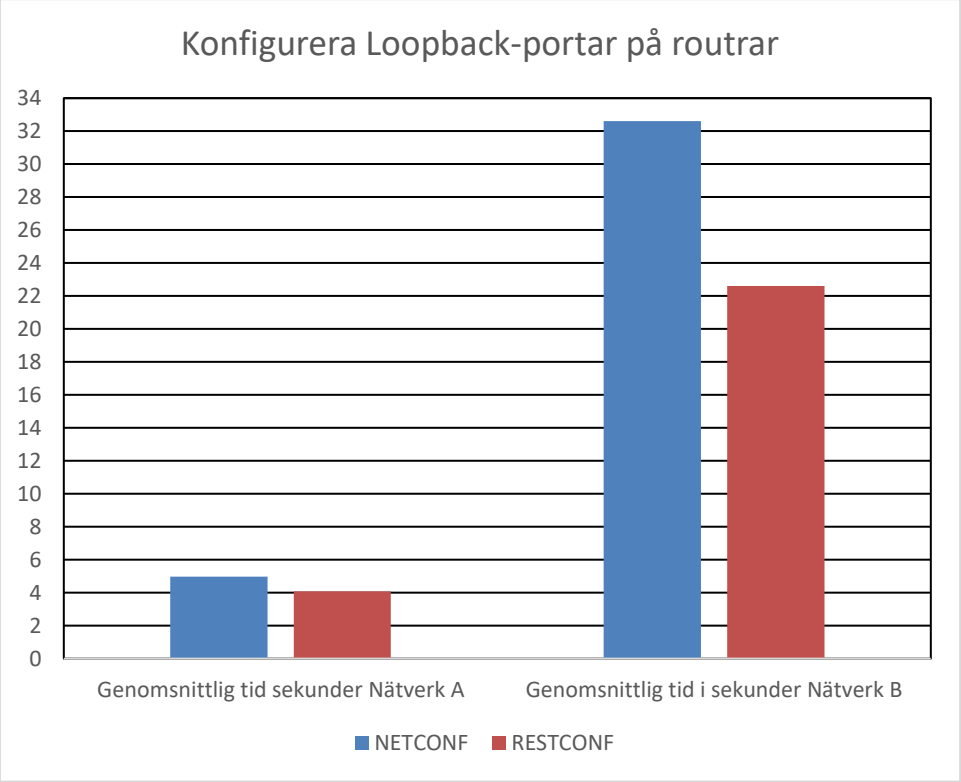
Figur 3. Genomsnittlig tid för att hämta hostnamn på enheter



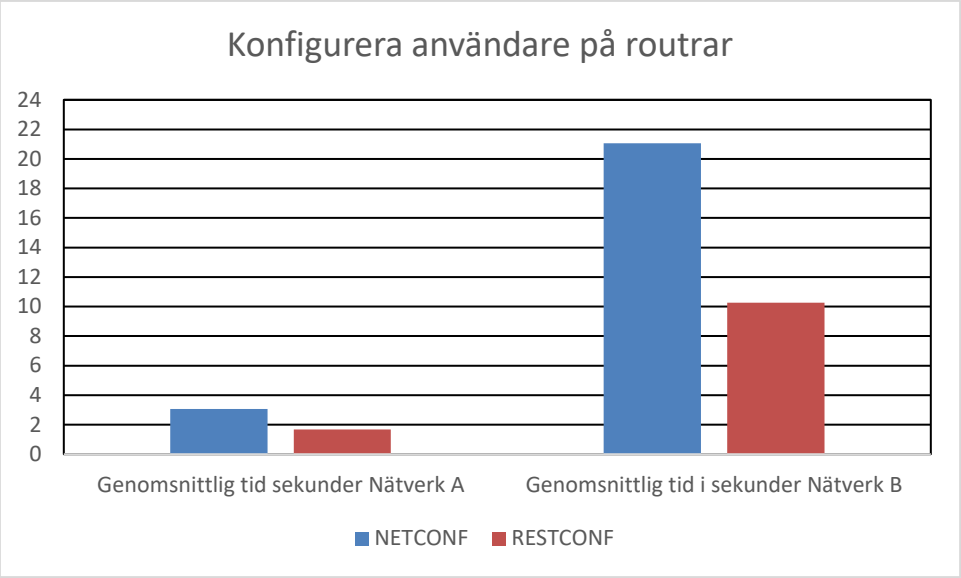
Figur 4. Genomsnittlig tid för att hämta aktiverade SVI och ethernetportar



Figur 5. Genomsnittlig tid att hämta OSPF-information på routrar



Figur 6. Genomsnittlig tid att sätta IP–adress och aktivera Loopbackportar på routrar.



Figur 7. Genomsnittlig tid att konfigurera användare på routrar

5 Analys och diskussion

Denna del ger en analys om resultatet samt en diskussion kring anledningen till resultatet samt förslag på vidare undersökningar där NETCONF och RESTCONF jämförs.

5.1 Analys

Resultatet visar att RESTCONF tar mindre tid än NETCONF i alla undersökningar som gjorts. Det fall där tidsskillnaden är som störst är att hämta konfigurationsdata från routrarna på nätverk B.

Inhämtning av data om ethernetportar och SVI på switchrar på nätverk A är skillnaden, i den genomsnittliga tiden, 0,53 sekunder. Detta är den minsta tidsskillnad mellan RESTCONF och NETCONF som uppmätts i denna undersökning. Det största tidsskillnaden som uppmättes var 13,25 sekunder. Detta var att hämta hostnamn på routrarna på nätverk B

Båda protokollen tar längre tid när data skickas och hämtas på nätverk B jämfört med nätverk A.

Det framgår av resultatet att det tar längre tid att skicka ut konfigurationsdata än det tar att hämta konfigurationsdata. Detta gäller för både NETCONF och RESTCONF.

5.2 Diskussion

En anledning till att RESTCONF är snabbare än NETCONF i denna undersökning kan vara att RESTCONF är stateless dvs. att ingen information om sessionen sparas.

I nätverk A är RESTCONF snabbare men det är med en liten marginal jämfört med NETCONF. Om ett nätverk med få nätverksenheter ska hanteras spelar det mindre roll huruvida RESTCONF eller NETCONF används.

Det är mindre troligt att en organisation hämtar och skickar konfiguration Just in time. Däremot sparar en organisation tid t.ex. om den regelbundet hämtar statusen på ethernetportar på routrar i ett nätverk med 17 routrar. Om detta görs var tionde minut visar resultaten att det sparas över en halvtimme per dygn.

Ett möjligt fortsättningsarbete är att göra jämförelsen på en ännu större nätverk än den i denna studie. Detta för att jämföra huruvida tidsskillnaden ökar mellan protokollen ju större näten är eller om skillnaden mattas av. Det är även intressant att fastslå varför det finns en tidsskillnad mellan NETCONF och RESTCONF.

För att undersöka varför tidsskillnaden finns ska transporten av data tas i beaktning. RESTCONF använder HTTP eller HTTPS och NETCONF använder SSH och skillnaderna mellan transport av data för HTTP och SSH bör stå i fokus för en sådan undersökning.

6 Slutsatser

Det längre tid att båda hämta information och konfigurera med NETCONF jämfört med RESTCONF när Python används för kommunikationen. Det tar längre tid både hämta information och konfigurera på nätverk B jämfört med nätverk A med båda protokollen. Slutsatsen är således att RESTCONF bör användas om tiden är en viktig faktor för valet mellan RESTCONF och NETCONF. En annan slutsats är också att storleken på nätverket har betydelse för tiden som går åt att hämta och skicka data till routrar och switchrar. Ett nätverk med färre nätverksenheter svarar snabbare än ett nätverk med fler nätverksenheter.

Referenser

[1] Edgeworth, B., Garza Rios, R., Gooley, J., Hucaby, D "Foundational Network Programmability Concepts" in *CCNP and CCIE Enterprise Core ENCORE 350–401*. Cisco Press, 2020, ch.28 pp. 818–819.

[2] E.A. Katonová, R. Petija, F. Jakab, P. Fecil'ak, M. Michalko "Integration of the programmability and automation concepts into the teaching of computer networks", 19th International Conference on Emerging eLearning Technologies and Applications (ICETA), Košice, Slovakia, Nov 11–12 2021

[3] Park, P., Na T., Kim, T., "Device Independent YANG Auto–generation Mechanism", 2021 International Conference on Information and Communication Technology Convergence (ICTC), Jeju Island, Korea, Oct. 20–22 2021

[4] F. Jeppsson "Konfigurering av VPN med Netconf/Yang och Python", Självständigt arbete på grundnivå, Avdelning för informationssystem och –teknologi, MIttuniversitetet, Sundsvall, 2017.

[5] Yigsit, Y., Huseynov, K., Ahmadi, H. Canberk, B. "YA–DA: YAng–Based DAta Model for Fine–Grained IIoT Air Quality Monitoring" 2022 IEEE Globecom Workshops (GC Wkshps) Globecom Workshops (GC Wkshps), . Rio de Janeiro, Brazil, Dec. 4–8 2022

[6] *The YANG 1.1 Data Modeling Language,* RFC 7950, Internet Engineering Task Force (IETF), August 2016

[7] Bray, T., Paoli, J., Sperberg–McQueen, C.M., "Extensible Markup Language (XML) 1.0 (Fifth edition)" Accessed Feb. 3, 2023 [Online] Available: https://www.w3.org/TR/xml/

[8] Json.org "Introduktion till JSON" Accessed Feb. 3, 2023 [Online] Available: https://www.json.org/json–en.html

[9] Granderath, M., Schonwalder, J., "A Resource Efficient Implementation of the RESTCONF Protocol for OpenWrt Systems", 2020 IEEE/IFIP Network Operations and Management Symposium Network Operations and Management Symposium, Budapest, Hungary Apr. 20–24 2022

[10] *JSON Encoding of YANG Data,* RFC 7951, Internet Engineering Task Force (IETF), August 2016

[11]N, Baluprithviraj.K. S, Monesh.M. C, Pranesh Raj. S, Varuna. "Application Programming Interface (API) based Student Activity Tracking System, " 2022 International Conference on Augmented Intelligence and Sustainable Systems (ICAISS), Trichy, India, Nov. 24–26 2022

[12] Nishtha, Sood, M., "Software Defined Network – Architectures" 2014 International Conference on Parallel, Distributed and Grid Computing Parallel, Distributed and Grid Computing (PDGC), Solan, India, Dec. 11–13 2014

[13] Skansholm, J., "Intepreterade Språk" in *Python från början*, Studentlitteratur AB, Lund, 2019, ch.1 pp. 10–11.

[14]Ncclient, "Welcome" Accessed Mar. 11, 2023 [Online] Available: https://ncclient.readthedocs.io/en/latest/

[15] Real Python "Python's Requests Library (Guide)" Accessed Mar. 3, 2023 [Online] Available: https://realpython.com/python-requests/

[16] W3 Schools "Python Json" Accessed Mar. 3, 2023 [Online] Available: https://www.w3schools.com/python/python_json.asp

[17] Python.org "xml.dom.minidom–Minimal DOM implementation" Accessed Mar. 11, 2023 [Online] Available: https://docs.python.org/3/library/xml.dom.minidom.html

A. Adressering för Nätverk A

Nätverk A		
Enhet	**Port**	**IP-adress**
R1	G0/0/0	10.10.12.1/24
	G0/0/1	192.168.1.1/24
R2	G0/0/0	10.10.12.2/24
	G0/0/1	10.10.23.2/24
R3	G0/0/0	10.10.23.3/24
	G0/0/1	192.168.3.1/24
S1	VLAN 1	192.168.1.2/24
S2	VLAN 1	192.168.3.2/24
PC	NIC	DHCP

B. Adressering för Nätverk B

Nätverk B		
Enhet	**Port**	**IP-adress**
R1	G0/0/0	10.10.12.1/24
	G0/0/1	192.168.1.1/24
R2	G0/0/0	10.10.12.2/24
	S0/1/0	10.10.24.1/24
	G0/0/1	10.10.23.2/24
R3	G0/0/0	10.10.23.3/24
	G0/0/1	192.168.3.1/24
R4	G0/0/0	10.10.45.2/24
	S0/1/0	10.10.24.2/24
	S0/1/1	10.10.47.1/24
	G0/0/1	10.10.46.2/24
R5	G0/0/0	10.10.45.3/24
	G0/0/1	192.168.5.1/24
R6	G0/0/0	10.10.46.3/24
	G0/0/1	192.168.6.1/24
R7	G0/0/0	10.10.78.2/24
	S0/1/0	10.10.47.2/24
	G0/0/1	10.10.79:2/24
R8	G0/0/0	10.10.78.1/24
	G0/0/1	192.168.8.1/24
R9	G0/0/0	10.10.79.1/24
	G0/0/1	DHCP
R10	G0/0/0	10.10.101.1/24
	S0/1/0	10.10.102.1/24
	G0/0/1	DHCP
R11	G0/0/0	10.10.101.2/24
R12	G0/0/0	10.13.12.1/24
	S0/1/0	10.10.102.2/24
	S0/1/1	10.12.15.1/24
	G0/0/1	10.14.12.1/24
R13	G0/0/0	10.13.12.2/24
R14	G0/0/0	10.14.12.2/24
R15	G0/0/0	10.15.16.1/24
	S0/1/0	10.12.15.2/24
	G0/0/1	10.15.17.1/24
R16	G0/0/0	10.15.16.2/24
R17	G0/0/0	10.15.17.2/24
S1	VLAN 1	192.168.1.2/24

S2	VLAN 1	192.168.3.2/24
S3	VLAN 1	192.168.5.2/24
S4	VLAN 1	192.168.6.2/24
S5	VLAN 1	192.168.8.2/24
PC	NIC	DHCP

C. Grundkonfiguration Switchrar

```
host S1

banner motd #KEEP OUT#

no ip domain lookup

ip domain name cisco.com

username admin privilege 15 secret cisco

crypto key generate rsa general-keys modulus 1024

line con 0

exec-timeout 0 0

logg sync

exit

line vty 0 15

exec-timeout 0 0

logg sync

transport input ssh

login local

exit

inter vlan 1

ip add 192.168.1.2 255.255.255.0

no shut

exit

ip default-gateway 192.168.1.1

netconf-yang

restconf

ip http server-secure

ip http authentication local
```

D. Grundkonfiguration Routrar

```
clock set 10:00:00 1 jan 2018

conf t

host R1

banner motd #KEEP OUT#

no ip domain-lookup

ip domain-name cisco.com

username admin privilege 15 secret cisco

crypto key generate rsa general-keys modulus 1024

line con 0

exec-timeout 0 0

logg sync

exit

line vty 0 15

exec-timeout 0 0

logg sync

transport input ssh

login local

exit

inter g0/0/0

ip add 10.10.12.1 255.255.255.0

no shut

inter g0/0/1

ip add 192.168.1.1 255.255.255.0

no shut

exit

ip dhcp excluded-address 192.168.1.1 192.168.1.10

ip dhcp pool LAN

network 192.168.1.0 /24

default-router 192.168.1.1

domain-name cisco.com
```

```
exit
router ospf 1
router-id 1.1.1.1
network 192.168.1.0 0.0.0.255 area 0
network 10.10.12.0 0.0.0.255 area 0
passive-interface g0/0/1
exit
netconf-yang
restconf
ip http server-secure
ip http authentication local
```

E. Python för att konfigurera Loopback-portar med RESTCONF

```
###CONF LOOPBACK INTERFACE RESTCONF

import json

import requests

import urllib3

requests.packages.urllib3.disable_warnings()

import time

nytt_ord=''

start=time.time()

listaroutrarA=['10.10.12.2','192.168.1.1','192.168.3.1']

listaroutrarB=['10.10.12.2','192.168.1.1','192.168.3.1','10.10.45.2','10.10.45
.3','10.10.46.3',
'10.10.78.2','192.168.8.1','10.10.79.1','10.10.101.1','10.10.101.2','10.13.12.
1','10.13.12.2','10.14.12.2','10.15.16.1','10.15.16.2','10.15.17.2',]


headers = { "Accept": "application/yang-data+json",

            "Content-type":"application/yang-data+json"

            }

basicauth = ("admin", "cisco")


payload = '''

{{

            "ietf-interfaces:interface": {{

            "name": "Loopback20",

            "description": "Loopback",

            "type": "iana-if-type:softwareLoopback",

            "enabled": true,

            "ietf-ip:ipv4":{{

                        "address":[

                                    {{
```

```
                                    "ip": "{ip}",
                                    "netmask": "255.255.255.0"
                                    }}
                                    ]
            }},
            "ietf-ip:ipv6": {{}}
            }}
}}
'''
for i in range(len(listaroutrarA)):
            api_url= "https://"+ listaroutrarA [i]+"/restconf/data/ietf-inter-
faces:interfaces"
            resp = requests.post(api_url, data=payload.for-
mat(ip="1.2.20."+str(i+1)), auth=basicauth, headers=headers, verify=False)


stop=time.time()
final=stop-start
print(final)
```

F. Python för att konfigurera Loopback-portar med NETCONF

```
###CONF LOOPBACK INTERFACE NETCONF

from ncclient import manager

import time

import xml.dom.minidom

import json

start_time=time.time()

listaroutrarB=['10.10.12.2','192.168.1.1','192.168.3.1','10.10.45.2','10.10.45
.3','10.10.46.3',
'10.10.78.2','192.168.8.1','10.10.79.1','10.10.101.1','10.10.101.2','10.13.12.
1','10.13.12.2','10.14.12.2','10.15.16.1','10.15.16.2','10.15.17.2',]

listaroutrarA=['10.10.12.2','192.168.1.1','192.168.3.1']

nytt_ord2=''

PORT = 830

USER = 'admin'

PASS = 'cisco'

netma='255.255.255.0'


for i in range(len(listaroutrarA)):

            ipadd='1.2.40.'+str(i+1)

            HOST2 = listaroutrarA [i]

            FILTER2 = f'''

            <config>

            <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">

            <interface>

            <Loopback>

            <name>40</name>

            <ip>

            <address>

            <primary>
```

```
            <address>{ipadd}</address>
            <mask>{netma}</mask>
            </primary>
            </address>
            </ip>
            </Loopback>
            </interface>
            </native>
            </config>

       '''

            with manager.connect(host=HOST2, port=PORT, username=USER,
                        password=PASS, hostkey_verify=False,
                        device_params={'name': 'default'},
                        allow_agent=False, look_for_keys=False) as m:
                       results2 = m.edit_config(target='running', config=FIL-
TER2)
end_time=time.time()
final_time=end_time-start_time
print(final_time)
```

G. Python för att konfigurera användare med NETCONF

```
###CONF LOOPBACK INTERFACE NETCONF

from ncclient import manager

import time

import xml.dom.minidom

import json

start_time=time.time()

listaroutrarA=['10.10.12.2','192.168.1.1','192.168.3.1']

listaroutrarB=['10.10.12.2','192.168.1.1','192.168.3.1','10.10.45.2','10.10.45
.3','10.10.46.3',
'10.10.78.2','192.168.8.1','10.10.79.1','10.10.101.1','10.10.101.2','10.13.12.
1','10.13.12.2','10.14.12.2','10.15.16.1','10.15.16.2','10.15.17.2',]


nytt_ord2=''

PORT = 830

USER = 'admin'

PASS = 'cisco'

netma='255.255.255.0'


for i in range(len(listaroutrarA)):

            HOST2 = listaroutrarA [i]

            FILTER2 = f'''


            <config>

            <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">

            <username>

            <name>user40</name>

            <privilege>5</privilege>

            <password>

            <password>philip</password>
```

```
            </password>
            </username>
            </native>
            </config>

        '''

            with manager.connect(host=HOST2, port=PORT, username=USER,
                        password=PASS, hostkey_verify=False,
                        device_params={'name': 'default'},
                        allow_agent=False, look_for_keys=False) as m:
                       results2 = m.edit_config(target='running', config=FIL-
TER2)



end_time=time.time()
final_time=end_time-start_time
print(final_time)
```

H. Python för att konfigurera användare med RESTCONF

```
###CREATE user RESTCONF

import json

import requests

import urllib3

requests.packages.urllib3.disable_warnings()

import time

nytt_ord=''

start=time.time()


listaroutrarB=['10.10.12.2','192.168.1.1','192.168.3.1','10.10.45.2','10.10.45
.3','10.10.46.3',
'10.10.78.2','192.168.8.1','10.10.79.1','10.10.101.1','10.10.101.2','10.13.12.
1','10.13.12.2','10.14.12.2','10.15.16.1','10.15.16.2','10.15.17.2',]

listaroutrarA=['10.10.12.2','192.168.1.1','192.168.3.1']

for i in range(len(listaroutrarA)):

            api_url= "https://"+listaroutrarA[i]+"/restconf/data/Cisco-IOS-XE-
native:native/username=user20"


            headers = { "Accept": "application/yang-data+json",

            "Content-type":"application/yang-data+json"

            }

            basicauth = ("admin", "cisco")



            payload ={

            "Cisco-IOS-XE-native:username": {

            "name": "user20",

            "privilege": 5,

            "password": {
```

```
            "encryption": "0",
            "password": "philip"
            }
            }
            }

            resp=requests.put(api_url, data=json.dumps(payload), auth=basi-
cauth, headers=headers, verify=False)


stop=time.time()
final=stop-start
print(final)
```

I. Python för att hämta hostname med NETCONF

```
###GET HOSTNAME NETCONF

from ncclient import manager

import time

import xml.dom.minidom

start_time=time.time()

listaroutrarB=['10.10.12.2','192.168.1.1','192.168.3.1','10.10.45.2','10.10.45
.3','10.10.46.3',
'10.10.78.2','192.168.8.1','10.10.79.1','10.10.101.1','10.10.101.2','10.13.12.
1','10.13.12.2','10.14.12.2','10.15.16.1','10.15.16.2','10.15.17.2',]

listaroutrarA=['10.10.12.2','192.168.1.1','192.168.3.1']

listaswitcharA=['192.168.1.2','192.168.3.2']

listaswitcharB=['192.168.1.2','192.168.3.2','192.168.5.2','192.168.6.2','192.1
68.8.2']

nytt_ord=''

for i in range(len(listaroutrarA)):

     HOST = listaroutrarA[i]

     PORT = 830

     USER = 'admin'

     PASS = 'cisco'


     FILTER = '''

                <filter xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

                  <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-na-
tive">

                    <hostname></hostname>

                  </native>

                </filter>

            '''
```

```
    with manager.connect(host=HOST, port=PORT, username=USER,
                         password=PASS, hostkey_verify=False,
                         device_params={'name': 'default'},
                         allow_agent=False, look_for_keys=False) as m:



        results = m.get_config('running', FILTER)
        nytt_ord+=results.xml


end_time=time.time()
final_time=end_time-start_time
print(nytt_ord)
print(final_time)
```

J. Python för att hämta hostname med RESTCONF

```
###GET HOSTNAME REST

import json

import requests

requests.packages.urllib3.disable_warnings()

import time

nytt_ord=''

start=time.time()

listaroutrarB=['10.10.12.2','192.168.1.1','192.168.3.1','10.10.45.2','10.10.45
.3','10.10.46.3',
'10.10.78.2','192.168.8.1','10.10.79.1','10.10.101.1','10.10.101.2','10.13.12.
1','10.13.12.2','10.14.12.2','10.15.16.1','10.15.16.2','10.15.17.2']

listaswitcharB=['192.168.1.2','192.168.3.2','192.168.5.2','192.168.6.2','192.1
68.8.2']

listaroutrarA=['10.10.12.2','192.168.1.1','192.168.3.1']

listaswitcharA=['192.168.1.2','192.168.3.2']

for i in range(len(listaroutrarA)):

            api_url="https://"+listaroutrarA[i]+"/restconf/data/Cisco-IOS-XE-
native:native/hostname"

            headers = { "Accept": "application/yang-data+json",

            "Content-type":"application/yang-data+json"

            }

            basicauth = ("admin", "cisco")


            resp = requests.get(api_url, auth=basicauth, headers=headers, ver-
ify=False)

            response_json = resp.json()


            nytt_ord+=str(response_json)

            #print(response_json)

stop=time.time()

final=stop-start
```

```
print(nytt_ord)

print(final)
```

K. Python för att hämta interface med NETCONF

```
###GET INTERFACE NETCONF

from ncclient import manager

import time

import xml.dom.minidom

import json

start_time=time.time()

listaroutrarB=['10.10.12.2','192.168.1.1','192.168.3.1','10.10.45.2','10.10.45
.3','10.10.46.3',
'10.10.78.2','192.168.8.1','10.10.79.1','10.10.101.1','10.10.101.2','10.13.12.
1','10.13.12.2','10.14.12.2','10.15.16.1','10.15.16.2','10.15.17.2']

listaswitcharB=['192.168.1.2','192.168.3.2','192.168.5.2','192.168.6.2','192.1
68.8.2']

listaroutrarA=['10.10.12.2','192.168.1.1','192.168.3.1']

listaswitcharA=['192.168.1.2','192.168.3.2']

nytt_ord2=''

PORT = 830

USER = 'admin'

PASS = 'cisco'

for i in range(len(listaroutrarA)):

     HOST2 = listaroutrarA[i]

     FILTER2 = '''

<filter xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"><interfaces
xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"><interface><ena-
bled>true</enabled></interface></interfaces></filter>

       '''

     with manager.connect(host=HOST2, port=PORT, username=USER,

                          password=PASS, hostkey_verify=False,

                          device_params={'name': 'default'},

                          allow_agent=False, look_for_keys=False) as m:

      results2 = m.get_config('running', FILTER2)

      nytt_ord2+=results2.xml
```

```
end_time=time.time()

final_time=end_time-start_time

print(nytt_ord2)

print(final_time)
```

L. Python för att hämta interface med RESTCONF

```
###GET INTERFACE RESTCONF

import json

import requests

requests.packages.urllib3.disable_warnings()

import time

nytt_ord=''

start=time.time()

listaroutrarB=['10.10.12.2','192.168.1.1','192.168.3.1','10.10.45.2','10.10.45
.3','10.10.46.3',
'10.10.78.2','192.168.8.1','10.10.79.1','10.10.101.1','10.10.101.2','10.13.12.
1','10.13.12.2','10.14.12.2','10.15.16.1','10.15.16.2','10.15.17.2',]

listaswitcharB=['192.168.1.2','192.168.3.2','192.168.5.2','192.168.6.2','192.1
68.8.2']

listaroutrarA=['10.10.12.2','192.168.1.1','192.168.3.1']

listaswitcharA=['192.168.1.2','192.168.3.2']

for i in range(len(listaswitcharA)):

            api_url="https://"+listaswitcharA[i]+"/restconf/data/ietf-inter-
faces:interfaces/"

            headers = { "Accept": "application/yang-data+json",

            "Content-type":"application/yang-data+json"

            }

            basicauth = ("admin", "cisco")


            resp = requests.get(api_url, auth=basicauth, headers=headers, ver-
ify=False)

            response_json = resp.json()

            interfaces=response_json['ietf-interfaces:interfaces']['inter-
face']


            for interface in interfaces:
```

```
                        if bool(interface['enabled']) or bool(interface['ietf-
ip:ipv4']):
                                    nytt_ord+=str(interface)


stop=time.time()
final=stop-start
print(nytt_ord)
print(final)
```

M. Python för att hämta OSPF med RESTCONF

```
####GET OSPF CONF REST

import json

import requests

requests.packages.urllib3.disable_warnings()

import time

nytt_ord=''

start=time.time()

listaroutrarB=['10.10.12.2','192.168.1.1','192.168.3.1','10.10.45.2','10.10.45
.3','10.10.46.3',
'10.10.78.2','192.168.8.1','10.10.79.1','10.10.101.1','10.10.101.2','10.13.12.
1','10.13.12.2','10.14.12.2','10.15.16.1','10.15.16.2','10.15.17.2',]

listaroutrarA=['10.10.12.2','192.168.1.1','192.168.3.1']


for i in range(len(listaroutrarB)):

            api_url="https://"+listaroutrarB[i]+"/restconf/data/Cisco-IOS-XE-
native:native/router"

            headers = { "Accept": "application/yang-data+json",

            "Content-type":"application/yang-data+json"

            }

            basicauth = ("admin", "cisco")


            resp = requests.get(api_url, auth=basicauth, headers=headers, ver-
ify=False)

            response_json = resp.json()

            nytt_ord+=str(response_json)


stop=time.time()

final=stop-start

print(nytt_ord)

print(final)
```

N. Python för att hämta OSPF med NETCONF

```
###GET HOSTNAME NETCONF

from ncclient import manager

import time

import xml.dom.minidom

import json

start_time=time.time()

listaroutrarB=['10.10.12.2','192.168.1.1','192.168.3.1','10.10.45.2','10.10.45
.3','10.10.46.3',
'10.10.78.2','192.168.8.1','10.10.79.1','10.10.101.1','10.10.101.2','10.13.12.
1','10.13.12.2','10.14.12.2','10.15.16.1','10.15.16.2','10.15.17.2',]

listaroutrara=['10.10.12.2','192.168.1.1','192.168.3.1']

nytt_ord2=''

PORT = 830

USER = 'admin'

PASS = 'cisco'

for i in range(len(listaroutrarA)):

     HOST2 = listaroutrarA[i]

     FILTER2 = '''

<filter xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"><native
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native"><router></router></na-
tive></filter>

       '''

     with manager.connect(host=HOST2, port=PORT, username=USER,

                          password=PASS, hostkey_verify=False,

                          device_params={'name': 'default'},

                          allow_agent=False, look_for_keys=False) as m:

      results2 = m.get_config('running', FILTER2)

      nytt_ord2+=results2.xml
```

```
end_time=time.time()

final_time=end_time-start_time

print(nytt_ord2)

print(final_time)
```