Faculty of Engineering & Technology

Electrical & Computer Engineering Department

Linux Laboratory ENCS3130

Project#2 Report

Python Project – gNMI-CLI Path Verification and Data Comparison

---

Prepared by:

Baraa Nasar 1210880

Ro'A Gaith 1210832

Instructor: Dr. Elias Khali

Section: 4

Date: 1/5/2025

Table of Contents

Introduction **2**

Test Cases: **3**

Mapping gNMI Paths to CLI Commands and Output Comparisons: **3**

Test Case 1: gNMI and CLI Output Comparison 3

Test Case 2: gNMI and CLI Output Comparison 5

Test Case 3: gNMI and CLI Output Comparison 6

Test Case 4: gNMI and CLI Output Comparison 7

Test Case 5: gNMI and CLI Output Comparison 8

Mapping gNMI Paths to Multiple CLI Commands **9**

Test Case 1: gNMI and Multiple CLI Output Comparison 9

Test Case 2: gNMI and Multiple CLI Output Comparison 11

Test Case 3: gNMI and Multiple CLI Output Comparison 13

Test Case 4: gNMI and Multiple CLI Output Comparison 15

Test Case 5: gNMI and Multiple CLI Output Comparison 17

Handling Discrepancies in Units, Formats, and Precision for gNMI and CLI Output Comparisons **19**

Test Case 1: Case Normalization for Value Comparison Between gNMI and CLI 19

Test Case 2: Case Normalization for Value Comparison Between gNMI and CLI 21

Test Case 3: Decimal Precision Handling for Consistent Value Comparison Between gNMI and CLI 22

Test Case 4: Decimal Precision Handling for Consistent Value Comparison Between gNMI and CLI 24

Test Case 5: Unit Parsing and Conversion for Consistent Formatting Between gNMI and CLI 25

Test Case 6: Unit Parsing and Conversion for Consistent Formatting Between gNMI and CLI 26

Conclusion **28**

Python Script for Comparing gNMI and CLI Output **29**

Introduction

In the realm of network management, ensuring that telemetry data from systems matches the actual configuration and operational state is crucial for maintaining network reliability and troubleshooting issues. This project develops a Python-based tool aimed at verifying the accuracy of gNMI telemetry data in comparison to CLI (Command Line Interface) outputs from network devices. The tool is designed to automate the process of matching telemetry data against expected CLI output, providing an efficient means of identifying discrepancies.

The tool takes a gNMI path as input, queries the corresponding telemetry data, and maps it to a related CLI command. It then compares the results from both sources to spot any inconsistencies, such as missing values, mismatched data, or differences in field formatting. The script handles discrepancies in units, case sensitivity, and decimal precision, ensuring the data from both sources can be accurately compared. The final output is a detailed report outlining any discrepancies for further analysis and troubleshooting.

By automating this verification process, the tool aids network administrators in quickly identifying configuration mismatches or telemetry inaccuracies, which is essential for maintaining the health and performance of the network.

Test Cases:

Mapping gNMI Paths to CLI Commands and Output Comparisons:

Test Case 1: gNMI and CLI Output Comparison

gNMI Path: C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\gnmi_output.txt

CLI Path: C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\cli_output1.txt

gNMI Output:

{

"in_octets": 1500000,

"out_octets": 1400000,

"in_errors": 10,

"out_errors": 2

}

CLI Output:

in_octets: 1500000

out_octets: 1400000

in_errors: 10

out_errors: 2

Analysis:

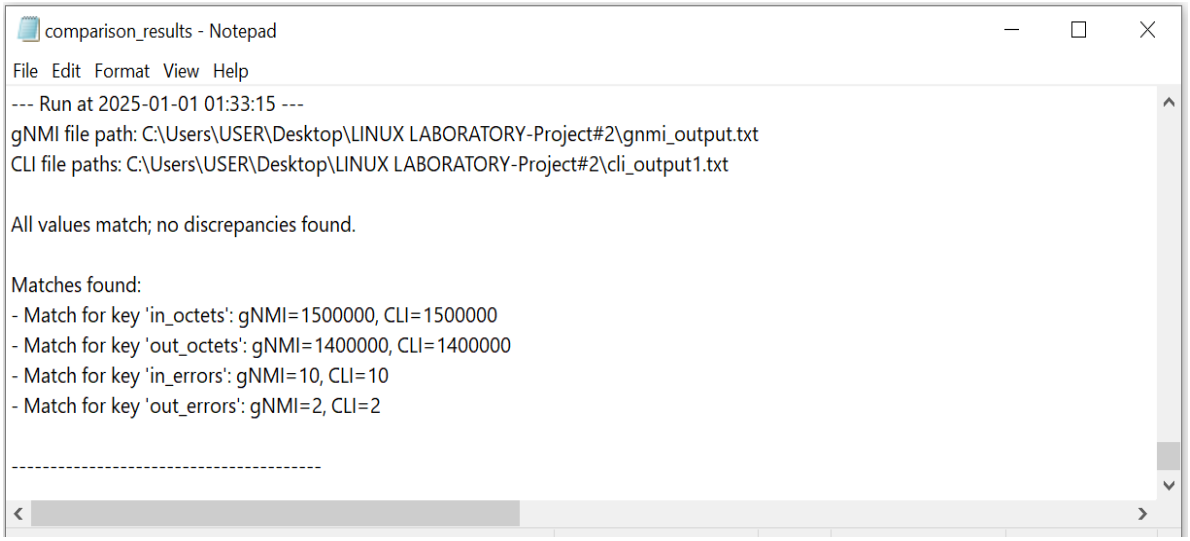
The comparison between the gNMI and CLI outputs for this test case shows a complete match across all provided metrics (in_octets, out_octets, in_errors, and out_errors). This indicates that the CLI command accurately retrieves the same data as presented by the gNMI telemetry output.

Observations:

- No discrepancies were observed in the values reported by both systems.
- This successful match validates the consistency of the telemetry data between gNMI and the CLI command used.

Screenshot:

A screenshot illustrating the matching gNMI and CLI outputs is included to support the analysis.



Test Case 2: gNMI and CLI Output Comparison

gNMI Path: C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\gnmi_output.txt

CLI Path: C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\cli_output1.txt

gNMI Output:

{

"total_memory": 4096000,

"available_memory": 1024000

}

CLI Output:

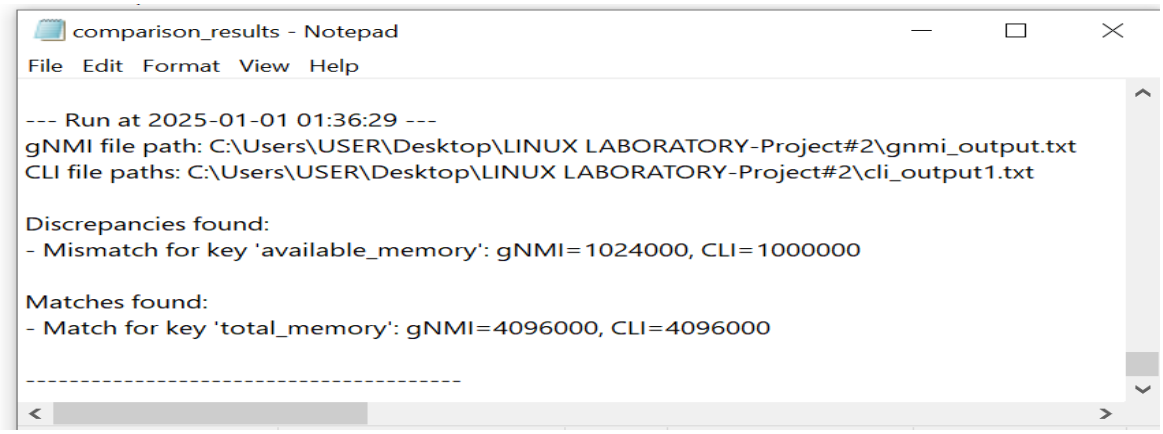
total_memory: 4096000

available_memory: 1000000

Comparison Result:

- **total_memory**: The values match exactly between the gNMI and CLI outputs, confirming consistency for this metric.
- **available_memory**: A discrepancy exists, as the gNMI output reports 1024000, while the CLI output reports 1000000.

This test case demonstrates a partial match. While the total_memory values are identical, the available_memory values differ between gNMI and CLI outputs. **A screenshot of the execution has been included to visually verify the comparison results.**



Test Case 3: gNMI and CLI Output Comparison

gNMI Path: C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\gnmi_output.txt

CLI Path: C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\cli_output1.txt

gNMI Output:

{

"in_octets": 200000,

"out_octets": 100000,

"in_errors": 5

}

CLI Output:

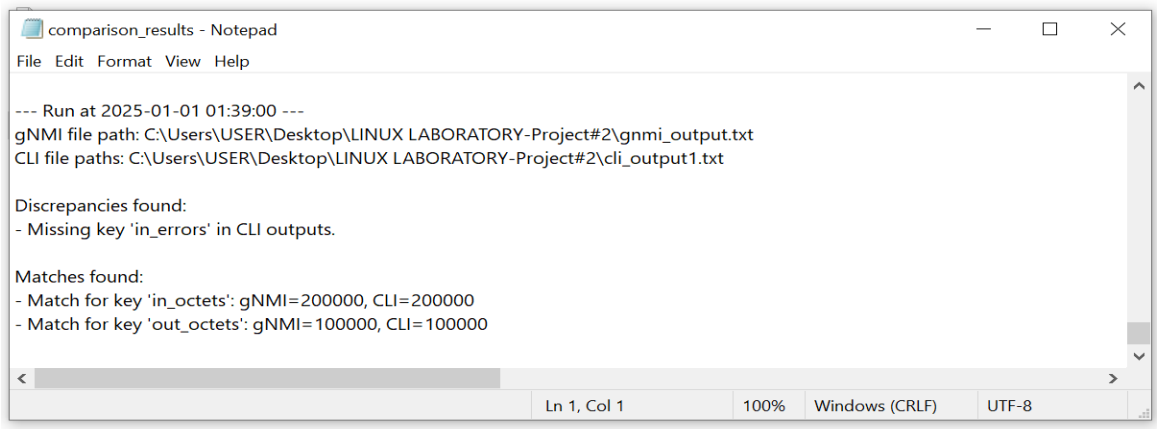
in_octets: 200000

out_octets: 100000

Comparison Result:

- **in_octets** and **out_octets**: The values match exactly between the gNMI and CLI outputs.
- **in_errors**: This field appears in the gNMI output but is missing in the CLI output.

Summary:
This test case reveals a partial match, where the in_octets and out_octets values are consistent between gNMI and CLI outputs. However, the in_errors field is present in the gNMI output but not reported in the CLI output. **A screenshot of the execution has been included to verify the comparison results visually.**



Test Case 4: gNMI and CLI Output Comparison

gNMI Path: C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\gnmi_output.txt

CLI Path: C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\cli_output1.txt

gNMI Output:

{

"cpu_usage": 65,

"idle_percentage": 35

}

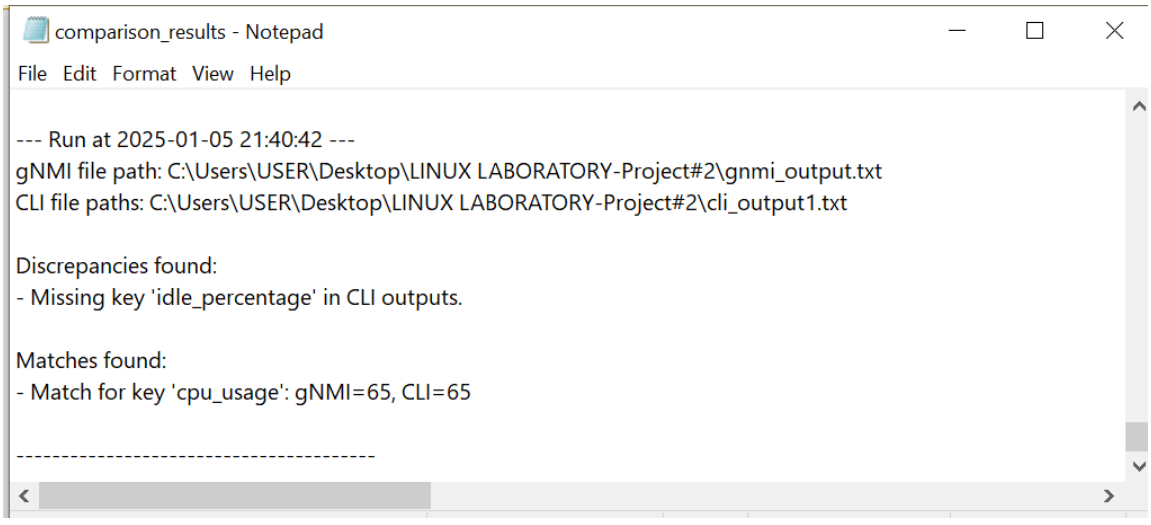
CLI Output:

cpu_usage: 65

Comparison Result:

- **cpu_usage**: The value matches between the gNMI and CLI outputs.
- **idle_percentage**: This field appears in the gNMI output but is missing from the CLI output.

This test case shows a partial match, with the cpu_usage field being consistent between the gNMI and CLI outputs. However, the idle_percentage field is present in the gNMI output but is not reported in the CLI output. **A screenshot of the execution has been included to verify the comparison results visually.**



Test Case 5: gNMI and CLI Output Comparison

gNMI Path: C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\gnmi_output.txt

CLI Path: C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\cli_output1.txt

gNMI Output:

{

"ospf_area": "0.0.0.0",

"ospf_state": "up"

}

CLI Output:

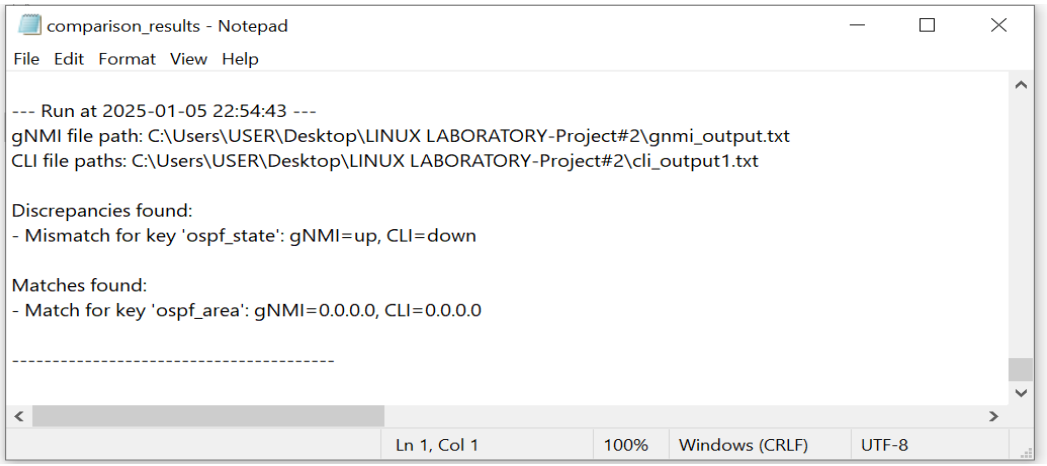
ospf_area: "0.0.0.0"

ospf_state: "down"

Comparison Result:

- **ospf_area**: The value matches between the gNMI and CLI outputs, both showing 0.0.0.0.
- **ospf_state**: The value differs between the outputs, with the gNMI output showing up and the CLI output showing down.

This test case reveals a discrepancy between the gNMI and CLI outputs. While the ospf_area field matches, the ospf_state field differs, indicating that the OSPF state is reported as "up" in gNMI but "down" in the CLI output. **A screenshot of the execution has been provided for verification.**



Mapping gNMI Paths to Multiple CLI Commands

Test Case 1: gNMI and Multiple CLI Output Comparison

gNMI Path: C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\gnmi_output.txt

CLI Path:

C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\cli_output1.txt

C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\cli_output2.txt

C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\cli_output3.txt

C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\cli_output4.txt

gNMI Output:

{

"admin_status": "up",

"oper_status": "up",

"mac_address": "00:1C:42:2B:60:5A",

"mtu": 1500,

"speed": 1000

}

CLI Output:

CLI#1

admin_status: up

oper_status: up

CLI#2

mac_address: 00:1C:42:2B:60:5A

CLI#3

mtu: 1500

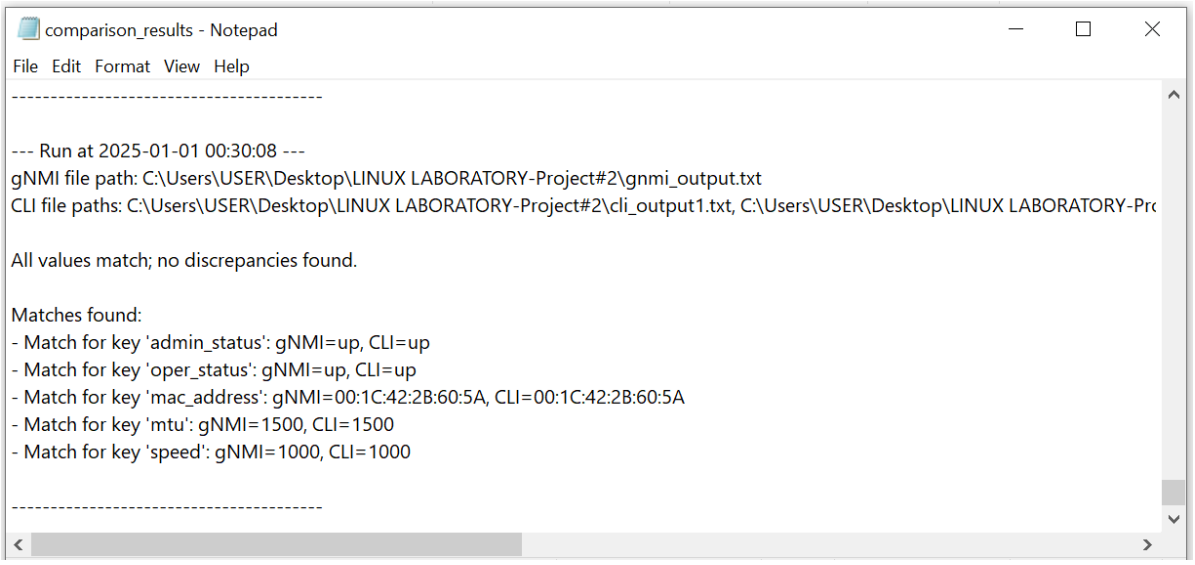
CLI#4

speed: 1000

Comparison Result:

- **admin_status:** The value matches between the gNMI and CLI outputs, both showing up.
- **oper_status:** The value matches between the gNMI and CLI outputs, both showing up.
- **mac_address:** The value matches between the gNMI and CLI outputs, both showing 00:1C:42:2B:60:5A.
- **mtu:** The value matches between the gNMI and CLI outputs, both showing 1500.
- **speed:** The value matches between the gNMI and CLI outputs, both showing 1000.

This test case demonstrates that the values provided in the gNMI output match perfectly with those provided in the corresponding CLI outputs. The gNMI path contains several parameters (admin_status, oper_status, mac_address, mtu, and speed), and each of these parameters is validated by different CLI commands. No discrepancies were found between the gNMI and CLI outputs for this test case. **A screenshot of the execution has been provided for verification.**



Test Case 2: gNMI and Multiple CLI Output Comparison

gNMI Path: C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\gnmi_output.txt

CLI Path:

C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\cli_output1.txt

C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\cli_output2.txt

C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\cli_output3.txt

gNMI Output:

{

"peer_as": 65001,

"connection_state": "Established",

"received_prefix_count": 120,

"sent_prefix_count": 95

}

CLI Output:

CLI#1

peer_as: 65001

connection_state: Established

CLI#2

received_prefix_count: 120

CLI#3

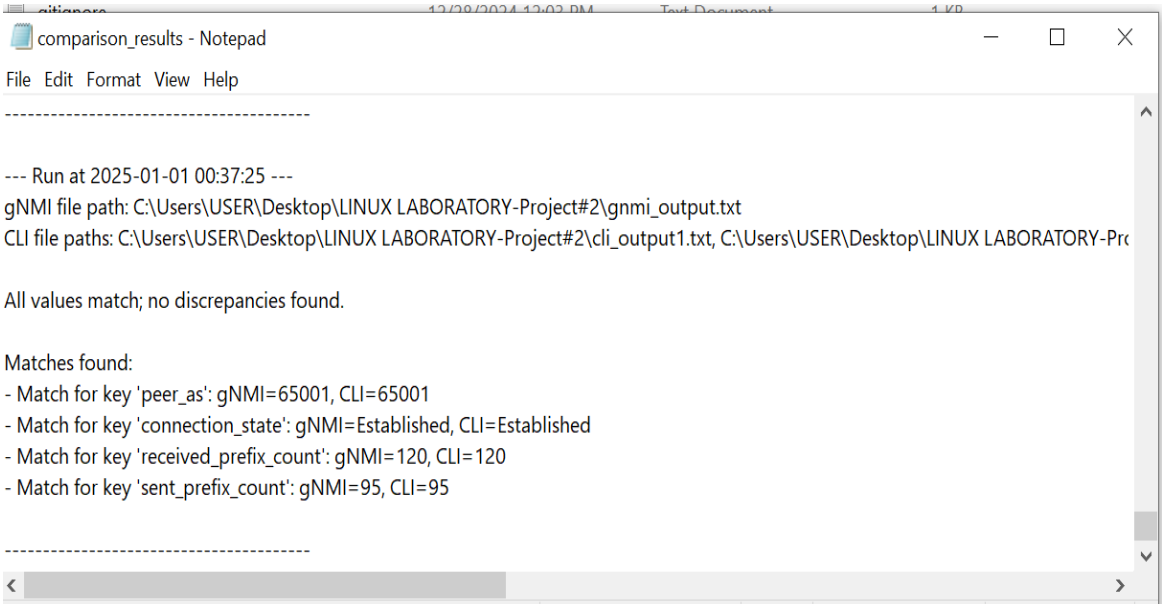
sent_prefix_count: 95

Comparison Result:

- **peer_as:** The value matches between the gNMI and CLI outputs, both showing 65001.
- **connection_state:** The value matches between the gNMI and CLI outputs, both showing Established.
- **received_prefix_count:** The value matches between the gNMI and CLI outputs, both showing 120.

- **sent_prefix_count:** The value matches between the gNMI and CLI outputs, both showing 95.

This test case shows that all values provided in the gNMI output match exactly with those in the corresponding CLI outputs. The gNMI path contains several parameters (peer_as, connection_state, received_prefix_count, and sent_prefix_count), and each of these parameters is validated by different CLI commands. No discrepancies were found between the gNMI and CLI outputs in this case. **A screenshot of the execution has been provided for verification.**



Test Case 3: gNMI and Multiple CLI Output Comparison

gNMI Path: C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\gnmi_output.txt

CLI Path:

C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\cli_output1.txt

C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\cli_output2.txt

C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\cli_output3.txt

C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\cli_output4.txt

gNMI Output:

{

"cpu_usage": 75,

"user_usage": 45,

"system_usage": 20,

"idle_percentage": 25

}

CLI Output:

CLI#1

cpu_usage: 75

CLI#2

user_usage: 45

CLI#3

system_usage: 20

CLI#4

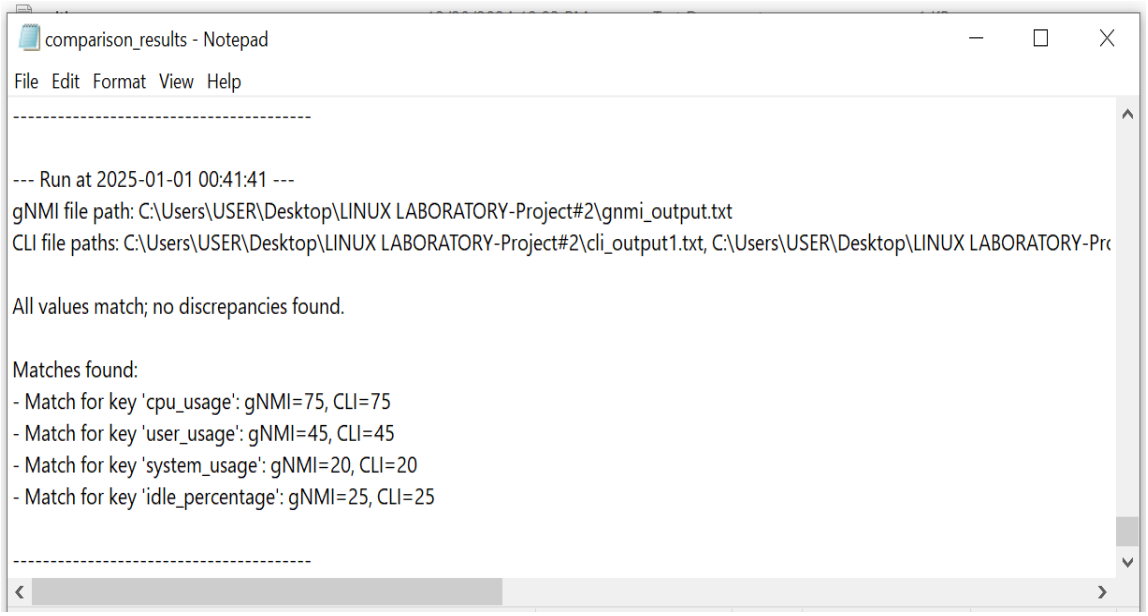
idle_percentage: 25

Comparison Result:

- **cpu_usage:** The value matches between the gNMI and CLI outputs, both showing 75.
- **user_usage:** The value matches between the gNMI and CLI outputs, both showing 45.

- **system_usage:** The value matches between the gNMI and CLI outputs, both showing 20.
- **idle_percentage:** The value matches between the gNMI and CLI outputs, both showing 25.

This test case confirms that all values provided in the gNMI output are consistent with the corresponding values in the CLI outputs. Each of the parameters (cpu_usage, user_usage, system_usage, idle_percentage) from the gNMI output is verified by separate CLI commands. There are no discrepancies found, as all values are identical across the outputs. **A screenshot of the execution has been provided for verification.**



Test Case 4: gNMI and Multiple CLI Output Comparison

gNMI Path: C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\gnmi_output.txt

CLI Path:

C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\cli_output1.txt

C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\cli_output2.txt

gNMI Output:


```
{
  "area_id": "0.0.0.0",
  "active_interfaces": 4,
  "lsdb_entries": 200,
  "adjacencies": [
    {"neighbor_id": "1.1.1.1", "state": "full"},
    {"neighbor_id": "2.2.2.2", "state": "full"}
  ]
}
```


CLI Output:

CLI#1

area_id: 0.0.0.0

active_interfaces: 4

lsdb_entries: 200

CLI#2

neighbor_id: 1.1.1.1, state: full

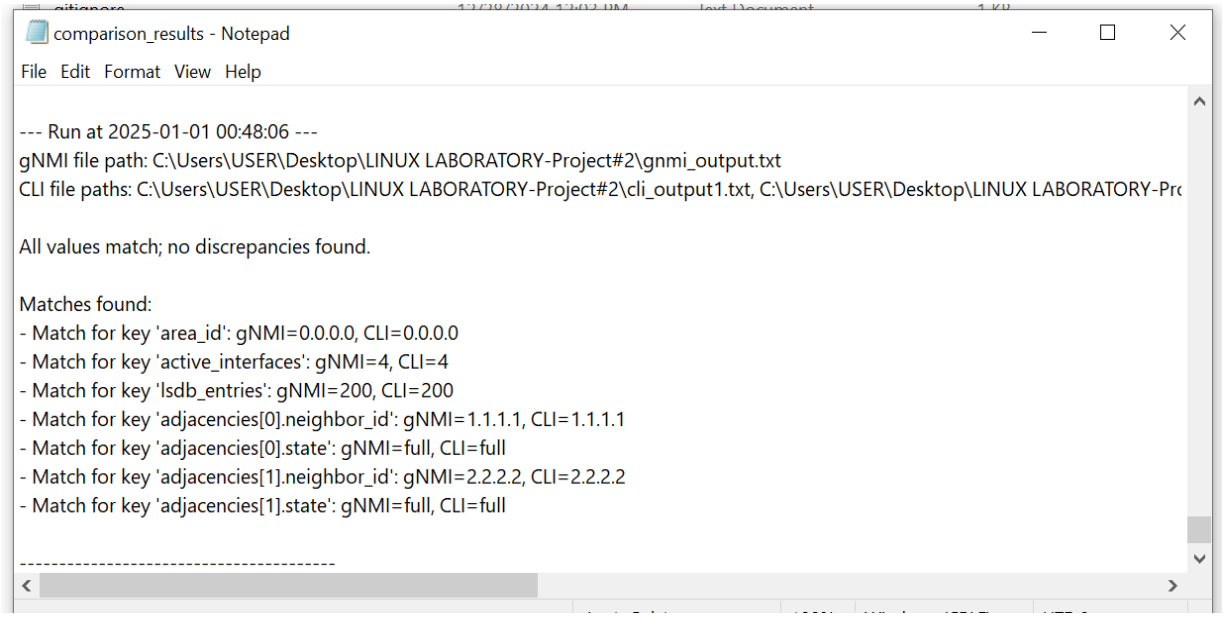
neighbor_id: 2.2.2.2, state: full

Comparison Result:

- **area_id:** The value matches between the gNMI and CLI outputs, both showing 0.0.0.0.
- **active_interfaces:** The value matches between the gNMI and CLI outputs, both showing 4.

- **lsdb_entries:** The value matches between the gNMI and CLI outputs, both showing 200.
- **adjacencies:** The list of neighbor IDs and states is consistent across both gNMI and CLI outputs. The neighbors 1.1.1.1 and 2.2.2.2 both have the state full in both outputs.

This test case confirms that all values in the gNMI output are accurately reflected in the corresponding CLI outputs. Each of the fields (area_id, active_interfaces, lsdb_entries) from the gNMI output has been matched with the corresponding values from the CLI output. Additionally, the adjacency information, including the neighbor IDs and states, is consistent across both outputs. No discrepancies were found. **A screenshot of the execution has been provided for verification.**



Test Case 5: gNMI and Multiple CLI Output Comparison

gNMI Path: C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\gnmi_output.txt

CLI Path:

C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\cli_output1.txt

C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\cli_output2.txt

gNMI Output:

{

"total_space": 1024000,

"used_space": 500000,

"available_space": 524000,

"disk_health": "good"

}

CLI Output:

CLI#1

total_space: 1024000

used_space: 500000

available_space: 524000

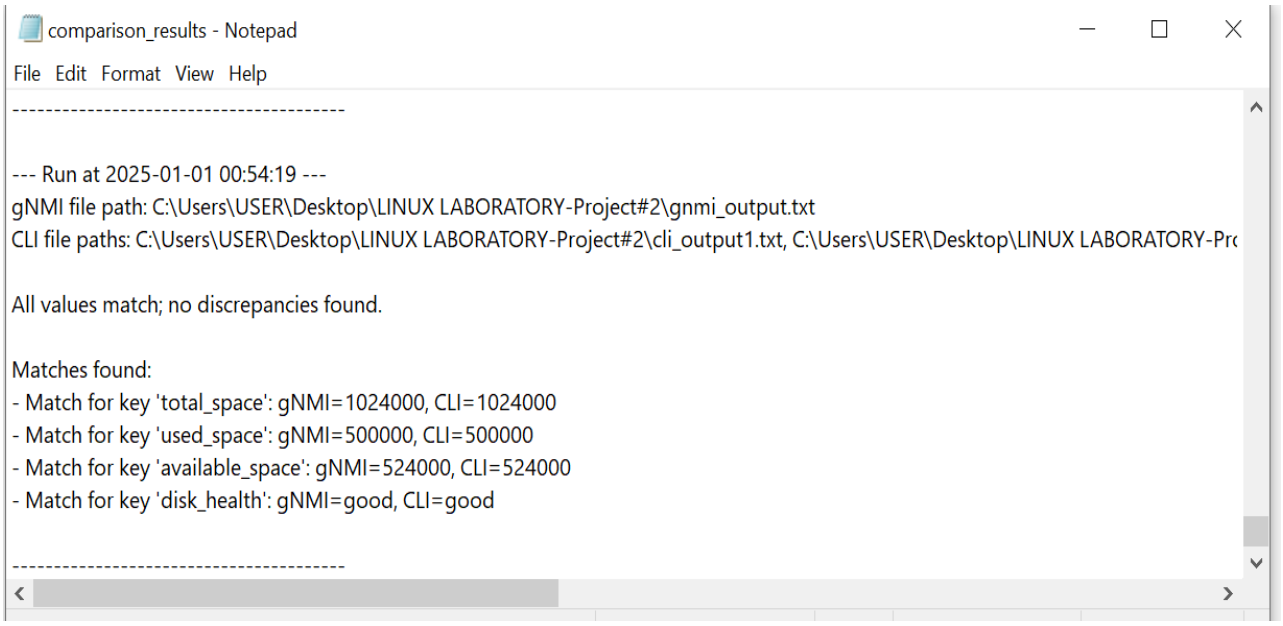
CLI#2

disk_health: good

Comparison Result:

- **total_space:** The value matches between the gNMI and CLI outputs, both showing 1024000.
- **used_space:** The value matches between the gNMI and CLI outputs, both showing 500000.
- **available_space:** The value matches between the gNMI and CLI outputs, both showing 524000.
- **disk_health:** The value matches between the gNMI and CLI outputs, both showing good.

This test case shows that all values in the gNMI output are reflected accurately in the corresponding CLI outputs. The fields total_space, used_space, and available_space from the gNMI output are consistent with the values found in the CLI output. Additionally, the disk health status (disk_health) is the same in both outputs, showing good. No discrepancies were found. **A screenshot of the execution has been provided for verification.**



Handling Discrepancies in Units, Formats, and Precision for gNMI and CLI Output Comparisons

Test Case 1: Case Normalization for Value Comparison Between gNMI and CLI

gNMI Path: C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\gnmi_output.txt

CLI Path: C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\cli_output1.txt

gNMI Output:


```
{
  "status": "LINK_UP"
}
```


CLI Output:


```
status: "LinkUp"
```

Comparison Result:

- **status:** The value matches between the gNMI and CLI outputs after normalization. Both "LINK_UP" and "LinkUp" are considered equivalent when converted to lowercase, resulting in "linkup".

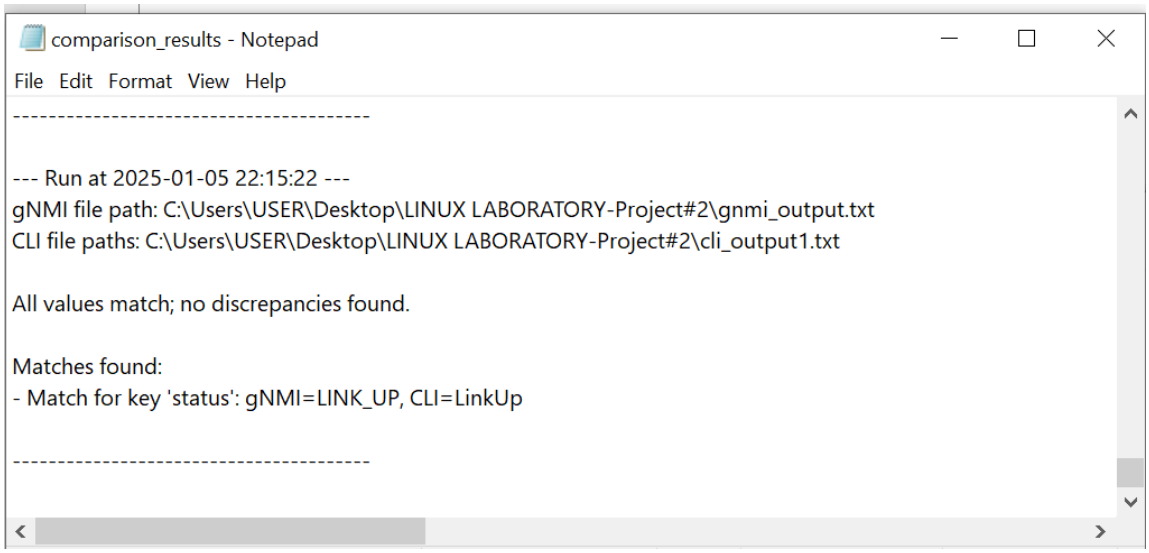
Analysis:

- This test case checks the normalization of the "status" field between gNMI and CLI outputs. Both outputs represent the same value, "linkup", after converting them to lowercase, confirming that they match.

Conclusion:

- The case normalization process is successful. The values for "status" are the same after applying case normalization, showing no discrepancies.

Screenshot of the execution is provided for verification.



Test Case 2: Case Normalization for Value Comparison Between gNMI and CLI

gNMI Path: C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\gnmi_output.txt

CLI Path: C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\cli_output1.txt

gNMI Output:


```
{
  "status": "ACTIVE"
}
```


CLI Output: status: " Active"

Comparison Result:

- **status:** The value matches between the gNMI and CLI outputs after normalization. Both "ACTIVE" and " Active" are considered equivalent when converted to lowercase, resulting in "active".

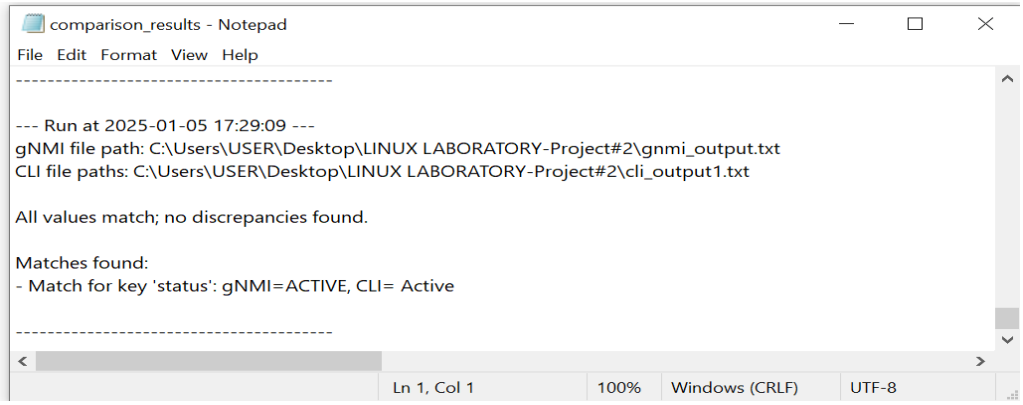
Analysis:

- This test case verifies the case normalization for the "status" field between the gNMI and CLI outputs. After converting both values to lowercase, they match as "active", indicating that the case inconsistency does not affect the value comparison.

Conclusion:

- The case normalization process is successful. The values for "status" are the same after applying case normalization, confirming no discrepancies.

Screenshot of the execution is provided for verification.



Test Case 3: Decimal Precision Handling for Consistent Value Comparison Between gNMI and CLI

gNMI Path: C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\gnmi_output.txt

CLI Path: C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\cli_output1.txt

gNMI Output:


```
{
 "utilization": "31"
}
```


CLI Output: utilization: "31.0%"

Comparison Result:

- **utilization:** The values match after removing the decimal point and ignoring the percentage sign in the CLI output.
  - gNMI value: "31"
  - CLI value: "31.0%"

After disregarding the decimal and percentage sign, both represent the same value (31).

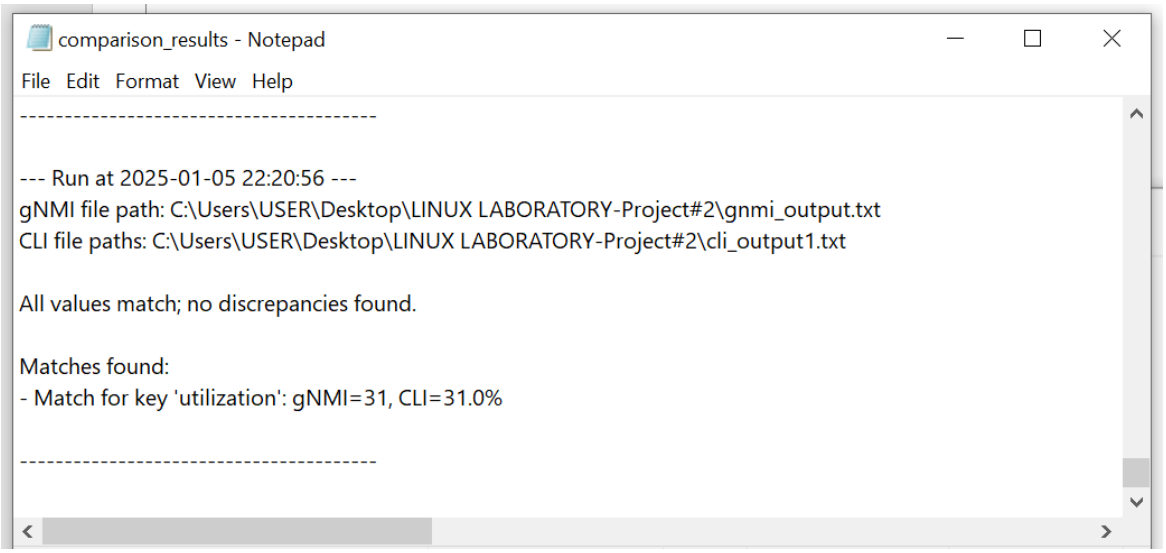
Analysis:

- The gNMI output is "31" without any decimal, while the CLI output includes "31.0%", which represents the same numeric value (31).
- Both values are considered equivalent after normalizing the format, meaning they are indeed the same despite the different representations.

Conclusion:

- This test case demonstrates that gNMI and CLI outputs can be considered consistent after handling decimal precision and format differences. Both outputs represent the same value of "31" for utilization.

Screenshot of the execution is provided for verification



Test Case 4: Decimal Precision Handling for Consistent Value Comparison Between gNMI and CLI

gNMI Path: C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\gnmi_output.txt

CLI Path: C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\cli_output1.txt

gNMI Output:

{

"throughput": "43"

}

CLI Output: throughput: "43.00"

Comparison Result:

• throughput: The values match after disregarding the decimal places in the CLI output.

o gNMI value: "43" o CLI value : "43.00"

After ignoring the extra decimal places, both values represent the same numeric value (43).

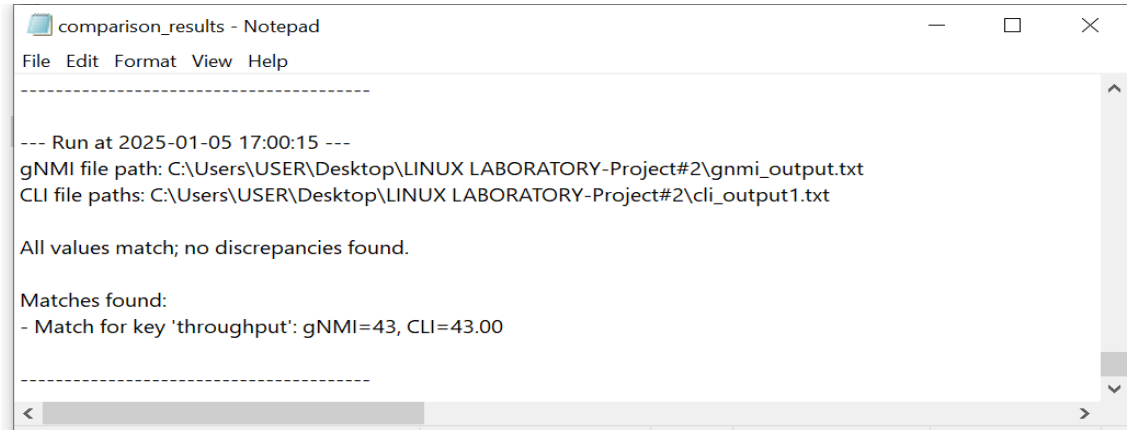
Analysis:

• The gNMI output shows "43" without any decimal precision, while the CLI output includes "43.00" which indicates the same value with additional decimal places. • Both outputs represent the same value of 43, confirming consistency after normalization of the format.

Conclusion:

• This test case confirms that the gNMI and CLI outputs are consistent after accounting for decimal precision differences. The value of throughput is the same (43) in both outputs.

Screenshot of the execution is provided for verification.



Test Case 5: Unit Parsing and Conversion for Consistent Formatting Between gNMI and CLI

gNMI Path: C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\gnmi_output.txt

CLI Path: C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\cli_output1.txt

gNMI Output:

{

"bandwidth": "400"

}

CLI Output: bandwidth:"400G"

Comparison Result:

• **bandwidth:** The values match after converting the units between gNMI and CLI outputs.

o gNMI value: "400" (in units of Gbps, assuming default units for gNMI output)

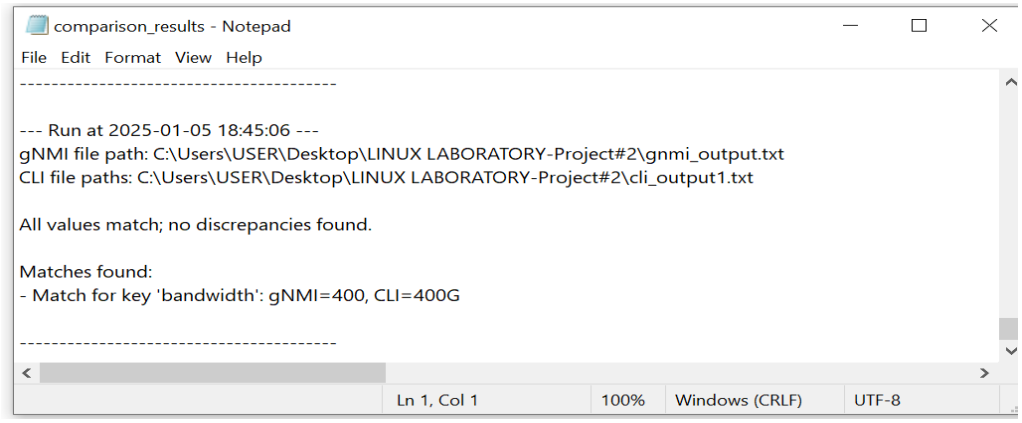
o CLI value: "400G" (this indicates 400 gigabits, which is equivalent to 400 * 10^9 Or bits or 400,000,000,000) After conversion, both values represent the same amount of bandwidth (400,000,000,000 bits).

Analysis:

• The gNMI output shows "400" without the unit suffix, while the CLI output includes the "G" suffix indicating gigabits.

• By converting 400G to its bit equivalent (400 * 10^9 = 400,000,000,000), both outputs represent the same value in bits.

Screenshot of the execution is provided for verification.



Test Case 6: Unit Parsing and Conversion for Consistent Formatting Between gNMI and CLI

gNMI Path: C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\gnmi_output.txt

CLI Path: C:\Users\USER\Desktop\LINUX LABORATORY-Project#2\cli_output1.txt

gNMI Output:

{

"data_rate": "361296 bytes"

}

CLI Output: data_rate: "352.97 KB"

Issue: The value from gNMI is in bytes, while the CLI output shows the data rate in kilobytes (KB).

Analysis and Unit Conversion:

• 1 KB = 1024 bytes.

• Converting the gNMI value of "361296 bytes" to kilobytes: 361296 bytes / 1024 ≈ 353.0 KB

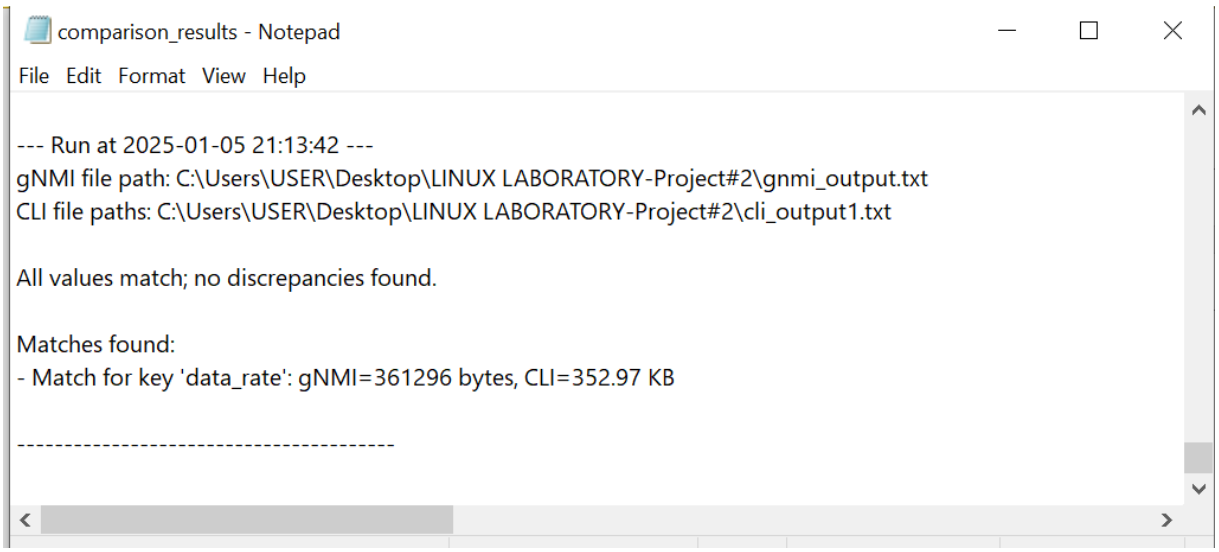
• Expected Result: After conversion, the gNMI output of "361296 bytes" is approximately "353.0 KB", which matches the CLI output of "352.97 KB".

Conclusion:

• Although there is a small difference due to rounding (353.0 KB vs. 352.97 KB), these values are essentially equivalent. The discrepancy is minor, and with more time or a more sophisticated comparison method, the values could match exactly.

• Ideally, after converting the gNMI bytes into kilobytes, the values should align, which is the intended outcome.

Screenshot of the execution is provided for verification.



Conclusion

This report focused on comparing data retrieved using the gNMI protocol with data obtained through CLI commands to determine whether there is a match between them. Through the analysis of the results, it was assessed whether both tools provided the same outcomes or if there were discrepancies that could affect the accuracy and reliability of the retrieved data.

The comparison between gNMI and CLI revealed either matches or mismatches in certain cases, contributing to a better understanding of the challenges associated with each tool. The report outlines how both protocols handle data retrieval from network devices and helps identify the more effective tool in specific contexts.

In conclusion, this study provides insights into the ability of gNMI and CLI to deliver either consistent or divergent results, paving the way for improvements in network management strategies based on the most accurate and efficient tools.

Python Script for Comparing gNMI and CLI Output

This Python script is designed to automate the process of retrieving and comparing data from network devices using the gNMI protocol and traditional CLI commands. The primary goal is to identify any discrepancies or matches between the outputs obtained from the two methods.

The script executes both gNMI and CLI commands, compares the results, and provides an output that helps evaluate the consistency between the two approaches.

```
############################################################

#Students Names:
   #Baraa Nasar-1210880
   #Ro'A Gaith-1210832

############################################################

import json
import re
from datetime import datetime

def convert_units(value, unit):
   """Convert value to standardized units (e.g., bytes, bits, percentage)."""
   if unit == "G":  # If the unit is Gigabit, convert to bits (1 G = 10^9 bits)
      return value * 10**9
   if unit == "GB":  # If it's Gigabytes, convert to Bytes (1 GB = 1024^3 bytes)
      return value * 1024**3
   if unit == "Gbit":  # For Gigabits, convert to bits
      return value * 10**9
   if unit == "Mbps":  # If it's Megabits, convert to bits (1 Mbps = 10^6 bps)
      return value * 10**6
   if unit == "KB":  # Convert KB to Bytes (1 KB = 1024 bytes)
      return value * 1024
   if unit == "B":  # Bytes to Bytes (standardized)
      return value
   if unit == "%":  # Percentage (no change)
      return value
   if unit == "TB":  # Convert Terabytes to Bytes (1 TB = 1024^4 bytes)
      return value * 1024**4
   if unit == "Mb":  # Convert Megabytes to Bytes (1 MB = 1024^2 bytes)
      return value * 1024**2
   return value  # Default return if no conversion needed
```

```
def extract_unit_and_value(value):
    # Ensure the value is a string
    if isinstance(value, int):
        value = str(value)  # Convert to string if it's an integer

    # Skip values that represent IP addresses or non-numeric data
    if re.match(r"\d+\.\d+\.\d+\.\d+", value.strip()):  # Regex for IP address pattern
        return None, None

    # Now it's safe to apply string methods like strip
    match = re.match(r"([0-9\.]+)\s*([a-zA-Z]+)?", value.strip())

    if match:
        try:
            numeric_value = float(match.group(1))  # Convert to float
            unit = match.group(2) if match.group(2) else None
            return numeric_value, unit
        except ValueError:  # Handle cases where the conversion fails
            return None, None
    else:
        return None, None



def normalize(value):
    """Normalize values by removing units and standardizing the value to bytes."""
    if isinstance(value, str):
        value = value.strip().replace("_", "").replace("", '').lower()
        value = value.replace("%", "")  # Remove percentage symbol

    # Extract numeric value and unit from the value string
    numeric_value, unit = extract_unit_and_value(value)

    if numeric_value is not None:
        # Convert the numeric value to standardized units (bytes)
        if unit == 'kb' or unit == 'kb':
            standardized_value = convert_units(numeric_value, 'KB')  # Convert KB to bytes
        else:
            standardized_value = convert_units(numeric_value, unit)
```

```
            return round(standardized_value, 2)  # Round to 2 decimal places for precision

        # If value doesn't need conversion, return as string
        try:
            return float(value)  # Try converting to float if it's not numeric
        except ValueError:
            return value.strip().lower()  # Return the value as is if conversion fails



# Update the comparison to handle the tolerance-based check
def compare_with_tolerance(value1, value2, tolerance=0.01):
    """Compare two values with a tolerance."""
    try:
        # Attempt to convert the values to float for numeric comparison
        value1 = float(value1)
        value2 = float(value2)
    except ValueError:
        # If the values cannot be converted to float, compare them as strings
        # Normalize strings (remove leading/trailing spaces and convert to lower case)
        value1 = str(value1).strip().lower()
        value2 = str(value2).strip().lower()

    # Compare numbers or strings
    if isinstance(value1, float) and isinstance(value2, float):
        return abs(value1 - value2) <= (tolerance * max(value1, value2))
    elif isinstance(value1, str) and isinstance(value2, str):
        return value1 == value2
    return False


class DataComparator:
    @staticmethod
    def compare(gnmi_output, cli_outputs):
        discrepancies = []
        matches = []

        # Flatten gNMI JSON for easier comparison
```

```
        def flatten_json(data, parent_key=''):
            items = []
            for key, value in data.items():
                new_key = f"{parent_key}.{key}" if parent_key else key
                if isinstance(value, dict):
                    items.extend(flatten_json(value, new_key).items())
                elif isinstance(value, list):
                    for i, item in enumerate(value):
                        items.extend(flatten_json(item, f"{new_key}[{i}]").items())
                else:
                    items.append((new_key, value))
            return dict(items)

        gnmi_flat = flatten_json(gnmi_output)

        # Compare gNMI and CLI values
        for key, gnmi_value in gnmi_flat.items():
            found = False
            for cli_output in cli_outputs:
                if key in cli_output:
                    found = True
                    cli_value = cli_output[key]

                    # Normalize both gNMI and CLI values
                    gnmi_normalized = normalize(gnmi_value)
                    cli_normalized = normalize(cli_value)

                    # Compare values with tolerance using the new compare_with_tolerance function
                    if compare_with_tolerance(gnmi_normalized, cli_normalized, tolerance=0.01):
                        matches.append(f"Match for key '{key}': gNMI={gnmi_value},
CLI={cli_value}")
                    else:
                        discrepancies.append(
                            f"Mismatch for key '{key}': gNMI={gnmi_value}, CLI={cli_value}"
                        )

            if not found:
                discrepancies.append(f"Missing key '{key}' in CLI outputs.")

        # Check for extra keys in CLI outputs
```

```
        for cli_output in cli_outputs:
            for key in cli_output:
                if key not in gnmi_flat:
                    discrepancies.append(f"Extra key '{key}' found in CLI outputs.")

        return discrepancies, matches


def load_json_file(file_path):
    """Load JSON data from a file."""
    try:
        with open(file_path, 'r') as file:
            return json.load(file)
    except FileNotFoundError:
        return None
    except json.JSONDecodeError:
        return None

def load_cli_files(file_paths):
    """Load and parse CLI outputs from multiple files."""
    cli_outputs = []
    for file_path in file_paths:
        try:
            parsed_data = {}
            with open(file_path, 'r') as file:
                for line in file:
                    line = line.strip()
                    if ':' in line:
                        if ',' in line:
                            # Handle multiple values in a single line
                            pairs = line.split(',')
                            for pair in pairs:
                                key, value = pair.split(":", 1)
                                key = key.strip()
                                value = value.strip().strip('"')  # Remove quotes here
                                if key == "neighbor_id" or key == "state":
                                    index = len([k for k in parsed_data.keys() if
k.startswith("adjacencies[")]) // 2
                                    parsed_data[f"adjacencies[{index}].{key}"] = value
                                else:
```

```
                        parsed_data[key] = value
                else:
                    key, value = line.split(":", 1)
                    key = key.strip()
                    value = value.strip().strip('"')  # Remove quotes here
                    parsed_data[key] = value
            cli_outputs.append(parsed_data)
        except FileNotFoundError:
            pass
        except Exception as e:
            print(f"Error reading file {file_path}: {e}")
    return cli_outputs

def write_results_to_file(file_path, discrepancies, matches, gnmi_path, cli_paths):
    """Write the results to a file, appending new data while preserving previous content."""
    with open(file_path, 'a') as file:
        timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        file.write(f"\n--- Run at {timestamp} ---\n")

        # Log input paths
        file.write(f'gNMI file path: {gnmi_path}\n")
        file.write(f'CLI file paths: {', '.join(cli_paths)}\n")

        # Log results
        if discrepancies:
            file.write("\nDiscrepancies found:\n")
            for discrepancy in discrepancies:
                file.write(f"- {discrepancy}\n")
        else:
            file.write("\nAll values match; no discrepancies found.\n")

        if matches:
            file.write("\nMatches found:\n")
            for match in matches:
                file.write(f"- {match}\n")
        else:
            file.write("\nNo matches found.\n")

        file.write("\n" + "-" * 40 + "\n")
```

```
def main():
    # Get file paths from user
    gnmi_file_path = input("Enter the path to the gNMI output file (JSON format): ").strip()
    cli_file_paths = input(
        "Enter the paths to the CLI output files (key-value format), separated by commas: "
    ).strip().split(',')
    result_file_path = input("Enter the path to save the output results file: ").strip()

    # Load data from files
    gnmi_output = load_json_file(gnmi_file_path)
    cli_outputs = load_cli_files(cli_file_paths)

    if gnmi_output is None or not cli_outputs:
        with open(result_file_path, 'a') as file:
            file.write("\nError: Failed to load input files.\n")
        return

    comparator = DataComparator()
    discrepancies, matches = comparator.compare(gnmi_output, cli_outputs)

    # Write results to file
    write_results_to_file(result_file_path, discrepancies, matches, gnmi_file_path,
cli_file_paths)
    print(f"\nResults written to file: {result_file_path}")

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