Birzeit University

Faculty of Engineering and Technology

Department of Electrical and Computer Engineering

First Semester 2024/2025

Project-1-

Name -1- : Mariam Turk

Name -2- : leen Daraghmeh

ID -1- : 1211115

ID -2- : 1210904

Instructors : Khader Mohammad , Elias Khalil

TA name : E.Ahed Mafarjeh

Table of Contents

Table of Figures

The Code

- Handle.sh code for Requirement 1

```
#!/bin/bash

# At least two arguments are provided
if [ "$#" -lt 2 ]; then
    echo "You need to provide a gNMI path and at least one CLI command"
    exit 1
fi

#Assigning input arguments
gnmi_path="$1"  #first argument: gNMI path
shift           #shift to access the CLI commands
cli_commands=("$@")  #remaining arguments: CLI commands

# Map gNMI path to file
echo "gNMI Path: $gnmi_path"
gnmi_file=$(echo "$gnmi_path" \
    | sed 's/\[id=\(.*\)\]/_\1/' \
    | sed 's/\[neighbor_address=\(.*\)\]/_\1/' \
    | sed 's/\[name=\(.*\)\]/_\1/' \
    | sed 's/\[\(.*\)\]/_/g' \
    | sed 's/^\///')
gnmi_file="${gnmi_file}/gnmi_data.txt"
echo "Transformed gNMI File Path: $gnmi_file"
```

```
# Validate gNMI file existence
if [ ! -f "$gnmi_file" ]; then
    echo "Error: gNMI file '$gnmi_file' does not exist."
    exit 1
fi

#Map and validate CLI files
cli_files=()
for cmd in "${cli_commands[@]}"; do
    cli_file="cli_outputs/$(echo "$cmd" | tr ' ' '_' | tr -d ':')"
    cli_file="${cli_file}.txt"
    echo "Mapped CLI Command: $cmd -> $cli_file"

    if [ ! -f "$cli_file" ]; then
        echo "Error: CLI file '$cli_file' does not exist."
        exit 1
    fi

    cli_files+=("$cli_file")  # Store the file path
done

#helper function to parse nested JSON arrays
parse_json_array() {
    echo "$1" | sed -E 's/^\[|\]$//g' | tr '}' '\n' | sed -E 's/^[[:space:]]*|[[:space:]]*$//g' | tr -d '{' | sed -E
's/,/\n/g'
}
```

```
#parse and compare gNMI and CLI files
declare -A gnmi_data
declare -A cli_data

# Parse gNMI file
while IFS=': ' read -r key value; do
    key=$(echo "$key" | tr -d '"{}, ')  # Clean up key
    value=$(echo "$value" | tr -d '"{}, ')  # Clean up value
    gnmi_data["$key"]="$value"
done < <(grep ":" "$gnmi_file")

#parse multiple CLI files and combine their data
for cli_file in "${cli_files[@]}"; do
    while IFS=': ' read -r key value; do
        key=$(echo "$key" | tr -d '"{}, ')  # Clean up key
        value=$(echo "$value" | tr -d '"{}, ')  # Clean up value
        cli_data["$key"]="$value"
    done < "$cli_file"
done

# Generate Report
report_file="comparison_report.txt"

{
    echo "Comparison Report"
    echo "gNMI Path: $gnmi_path"
    echo "CLI Commands: ${cli_commands[*]}"
    echo ""
```

```
echo "Comparison Results:"
echo "....................................."

# compare gNMI and CLI data
for key in "${!gnmi_data[@]}"; do
    gnmi_value="${gnmi_data[$key]}"
    cli_value="${cli_data[$key]}"

    if [[ "$key" == "adjacencies" ]]; then
        continue
    fi

    if [ -z "$cli_value" ]; then
        echo "$key : Present in gNMI but missing in CLI"
    elif [ "$gnmi_value" == "$cli_value" ]; then
        echo "$key : Match"
    else
        echo "$key : Mismatch (gNMI: $gnmi_value, CLI: $cli_value)"
    fi
done

#check for keys in CLI data missing from gNMI data
for key in "${!cli_data[@]}"; do
    if [[ "$key" == "adjacencies" ]]; then
        continue
    fi
    if [ -z "${gnmi_data[$key]}" ]; then
        echo "$key : Present in CLI but missing in gNMI"
```

```
        fi
    done
} > "$report_file"

echo "Report saved to $report_file"
```

Discussion :

This Bash script is designed to automate the process of validation and comparison between gNMI paths and their corresponding outputs from CLI commands. Basically, it starts by taking as input a gNMI path and at least one CLI command, then processes these inputs by mapping the gNMI path to a structured file path and changing the CLI commands into corresponding file names. After having the script validate that those files exist, it will parse the content of the gNMI and CLI files into associative arrays, in which key-value pairs will be stored for comparison. The script performs a detailed comparison of matches, mismatches, and keys present in one source but missing in the other. Special care is taken to handle and exclude, if necessary, specific keys, like nested structures. It finally generates a complete report, comparison_report.txt, documenting the results of the comparison, highlighting discrepancies or confirming matches. The script will be useful to compare network configurations or data outputs. This script is very helpful in an automated test environment, configuration validation, and debugging in network environments.

- r0.txt code for Requirement 1

```
#!/bin/bash
# Step 1: Ensure at least two arguments are provided
if [ "$#" -lt 2 ]; then
echo "Usage: $0 <gNMI_path> <CLI_command1> [CLI_command2] ..."
exit 1
```

```
fi

# Step 2: Assign input arguments

gnmi_path="$1" # First argument: gNMI path

shift # Shift to access the CLI commands

cli_commands=("$@") # Remaining arguments: CLI commands

# Step 3: Map gNMI path to file

echo "Original gNMI Path: $gnmi_path"

gnmi_file=$(echo "$gnmi_path" | sed 's/\[name=\(.*\)\]/_\1/' | sed 's/\[\(.*\)\]/_\1/g' | sed 's/^\///')

gnmi_file="${gnmi_file}/gnmi_data.txt"

echo "Transformed gNMI File Path: $gnmi_file"

# Step 4: Validate gNMI file existence

if [ ! -f "$gnmi_file" ]; then

echo "Error: gNMI file '$gnmi_file' does not exist."

exit 1

fi

# Step 5: Map and validate CLI files

cli_files=()

for cmd in "${cli_commands[@]}"; do

cli_file="cli_outputs/$(echo "$cmd" | tr ' ' '_' | tr -d ':')"

cli_file="${cli_file}.txt"

echo "Mapped CLI Command: $cmd -> $cli_file"

if [ ! -f "$cli_file" ]; then

echo "Error: CLI file '$cli_file' does not exist."
```

```
exit 1

fi

cli_files+=("$cli_file") # Store the file path

done

# Step 6: Parse and compare gNMI and CLI files

echo "Comparing gNMI and CLI data..."

declare -A gnmi_data

declare -A cli_data

# Parse gNMI file

while IFS=': ' read -r key value; do

key=$(echo "$key" | tr -d '"{}, ') # Clean up key

value=$(echo "$value" | tr -d '"{}, ') # Clean up value

gnmi_data["$key"]="$value"

done < <(grep ":" "$gnmi_file")

# Parse CLI files

for cli_file in "${cli_files[@]}"; do

while IFS=': ' read -r key value; do

key=$(echo "$key" | tr -d '"{}, ') # Clean up key

value=$(echo "$value" | tr -d '"{}, ') # Clean up value

cli_data["$key"]="$value"

done < "$cli_file"

done

# Generate Report
```

```
report_file="comparison_report.txt"
echo "Generating report: $report_file"
{
echo "Comparison Report"
echo "=================="
echo "gNMI Path: $gnmi_path"
echo "CLI Commands: ${cli_commands[*]}"
echo ""
echo "Comparison Results:"
echo "=================="
for key in "${!gnmi_data[@]}"; do
gnmi_value="${gnmi_data[$key]}"
cli_value="${cli_data[$key]}"
if [ -z "$cli_value" ]; then
echo "Key: $key - Present in gNMI but missing in CLI"
elif [ "$gnmi_value" == "$cli_value" ]; then
echo "Key: $key - Match"
else
echo "Key: $key - Mismatch (gNMI: $gnmi_value, CLI: $cli_value)"
fi
done
# Check for keys in CLI data missing from gNMI data
for key in "${!cli_data[@]}"; do
```

```
if [ -z "${gnmi_data[$key]}" ]; then
echo "Key: $key - Present in CLI but missing in gNMI"
fi
done
} > "$report_file"
echo "Report saved to $report_file"
```

Discussion :

This Bash script automates running comparisons between data from any gNMI path with the output of one or more CLI commands, basically for integrity and consistency across the data. It begins its work by checking the inputs provided: the very first argument is the path in gNMI; the rest of the arguments are for the CLI command. The gNMI path is transformed into a standardized file path, using sed to handle its formatting, whereas the CLI commands are turned into corresponding filenames in the directory cli_outputs/ by replacing spaces and special characters. It checks that all the files required are present. It then reads key-value pairs from the gNMI file and CLI files into associative arrays: gnmi_data and cli_data. It performs a detailed comparison-checking for the presence of keys in both sources, matching values, or missing keys and mismatched values. The results of such a comparison are compiled as a report, comparison_report.txt, which also summarizes the structured matches, mismatches, and missing data from both gNMI and CLI sources. The script, in turn, automates data mapping and validation, as well as reporting, which in turn provides a simplified process for performing network configuration verification and is actually useful for network administrators working with gNMI or CLI tools.

- comp.sh code for Requirement 1

```
#!/bin/bash

# Check if two arguments are provided

if [ "$#" -ne 2 ]; then

echo "Usage: $0 <gNMI_file_path> <CLI_file_path>"

exit 1

fi

# Assign input arguments to variables

gnmi_file="$1"

cli_file="$2"

# Check if files exist

if [ ! -f "$gnmi_file" ]; then

echo "Error: gNMI file '$gnmi_file' does not exist."

exit 1

fi

if [ ! -f "$cli_file" ]; then

echo "Error: CLI file '$cli_file' does not exist."

exit 1

fi

# Parse gNMI file into an associative array

declare -A gnmi_data

while IFS=': ' read -r key value; do
```

```
key=$(echo "$key" | tr -d "'{}, ') # Remove unwanted characters
value=$(echo "$value" | tr -d "'{}, ')
gnmi_data["$key"]="$value"
done < <(grep ":" "$gnmi_file")
# Parse CLI file into an associative array
declare -A cli_data
while IFS=': ' read -r key value; do
key=$(echo "$key" | tr -d "'{}, ') # Remove unwanted characters
value=$(echo "$value" | tr -d "'{}, ')
cli_data["$key"]="$value"
done < "$cli_file"
# Compare the data
echo "Comparison Results:"
echo "==================="
for key in "${!gnmi_data[@]}"; do
gnmi_value="${gnmi_data[$key]}"
cli_value="${cli_data[$key]}"
if [ "$gnmi_value" == "$cli_value" ]; then
echo "Key: $key - Match"
else
echo "Key: $key - Mismatch (gNMI: $gnmi_value, CLI: $cli_value)"
fi
done
```

```
# Check for keys in CLI data missing from gNMI data

for key in "${!cli_data[@]}"; do

if [ -z "${gnmi_data[$key]}" ]; then

echo "Key: $key - Present in CLI but missing in gNMI"

fi

done
```

Discussion :

This Bash script is a simple yet powerful tool for comparing data between a gNMI file and a CLI file, providing a detailed analysis of their alignment. The script starts by ensuring that the user provides exactly two arguments, which are the file paths for the gNMI and CLI files, respectively, and checks if these files exist to avoid any errors. It parses each file into associative arrays, `gnmi_data` for the gNMI file and `cli_data` for the CLI file, cleaning the data by removing unnecessary characters such as quotes, braces, and extra spaces to standardize the format. The script then performs a comparison of the keys and values between the two data sets.

For every key within the gNMI data, it checks if the corresponding key in the CLI data exists and compares values. Matches are reported, and mismatched values are noted along with the differing values from each file. It also identifies when a key is in the CLI file and missing from the gNMI file to capture data gaps. The results of the comparison are presented in a tabular format, which makes it easy to identify mistakes such as mismatched information or missing entries. This script is very useful for a network engineer or system administrator in order to validate and debug configuration data between two sources, offering a straightforward and efficient way to detect inconsistencies.

- handle_2.sh code for Requirement 2

```
#!/bin/bash

# Step 1: Ensure at least two arguments are provided
if [ "$#" -lt 2 ]; then
    echo "Usage: $0 <gNMI_path> <CLI_command1> [CLI_command2] ..."
    exit 1
fi

# Step 2: Assign input arguments
gnmi_path="$1"           # First argument: gNMI path
shift                    # Shift to access the CLI commands
cli_commands=("$@")      # Remaining arguments: CLI commands

# Step 3: Map gNMI path to file
echo "Original gNMI Path: $gnmi_path"
gnmi_file=$(echo "$gnmi_path" \
    | sed 's/\[id=\(.*\)\]/_\1/' \
    | sed 's/\[neighbor_address=\(.*\)\]/_\1/' \
    | sed 's/\[name=\(.*\)\]/_\1/' \
    | sed 's/\[\(.*\)\]/_/g' \
    | sed 's/^\///' \
    | sed 's/-/_/g')  # Replace "-" with "_"
gnmi_file="${gnmi_file}/gnmi_data.txt"
echo "Transformed gNMI File Path: $gnmi_file"

# Step 4: Validate gNMI file existence
```

```
if [ ! -f "$gnmi_file" ]; then
    echo "Error: gNMI file '$gnmi_file' does not exist."
    exit 1
fi

# Step 5: Map and validate CLI files
cli_files=()
for cmd in "${cli_commands[@]}"; do
    sanitized_cmd=$(echo "$cmd" | sed 's/-/_/g')
    cli_file="cli_outputs/$(echo "$sanitized_cmd" | tr ' ' '_' | tr -d ':')"
    cli_file="${cli_file}.txt"
    echo "Mapped CLI Command: $cmd -> $cli_file"

    if [ ! -f "$cli_file" ]; then
        echo "Error: CLI file '$cli_file' does not exist."
        exit 1
    fi

    cli_files+=("$cli_file")  # Store the file path
done

# Helper functions
normalize_string() {
    echo "$1" | tr '[:upper:]' '[:lower:]' | xargs
}

convert_units() {
    value="$1"
```

```
    if [[ "$value" =~ ^([0-9.]+)([a-zA-Z]*)$ ]]; then
        num="${BASH_REMATCH[1]}"
        unit="${BASH_REMATCH[2]}"
        case "$unit" in
            "kb") echo "$(echo "$num * 1024" | bc)" ;;
            "mb") echo "$(echo "$num * 1024 * 1024" | bc)" ;;
            "gb") echo "$(echo "$num * 1024 * 1024 * 1024" | bc)" ;;
            "g") echo "$(echo "$num * 1000000000" | bc)" ;;
            "%"|"percent") echo "$num" ;; # Strip percentage
            "b"|"") echo "$num" ;;
            *) echo "$num" ;;
        esac
    else
        echo "$value"
    fi
}

adjust_precision() {
    printf "%.2f" "$1"
}

# Parse and compare gNMI and CLI files
report_file="comparison_report.txt"
{
    echo "Comparison Report"
    echo "----------------------"
    echo "gNMI Path: $gnmi_path"
    echo "CLI Commands: ${cli_commands[*]}"
```

```
echo ""
echo "Comparison Results:"
echo "---------------------"

for cli_file in "${cli_files[@]}"; do
    while IFS=': ' read -r key value; do
        gNMI_value=$(grep -i "$key" "$gnmi_file" | awk -F': ' '{print $2}' | xargs)
        CLI_value="$value"

        # Normalize and process values for comparison
        gNMI_normalized=$(normalize_string "$gNMI_value")
        CLI_normalized=$(normalize_string "$CLI_value")
        gNMI_converted=$(convert_units "$gNMI_value")
        CLI_converted=$(convert_units "$CLI_value")
        gNMI_adjusted=$(adjust_precision "$gNMI_value")
        CLI_adjusted=$(adjust_precision "$CLI_value")

        echo "Path: $gnmi_path"
        echo "o gNMI Output: $gNMI_value"
        echo "o CLI Output: $CLI_value"

        # Case 1: Match after normalization
        if [[ "$gNMI_normalized" == "$CLI_normalized" ]]; then
            echo "o Result: Match after edition"
        # Case 2: Match after conversion
        elif [[ "$gNMI_converted" == "$CLI_converted" ]]; then
            echo "o Result: Match after edition"
        # Case 3: Match after adjusting precision
```

```
        elif [[ "$gNMI_adjusted" == "$CLI_adjusted" ]]; then
            echo "o Result: Match after edition"
        else
            echo "o Normalized Comparison: $gNMI_normalized (gNMI) vs. $CLI_normalized (CLI)"
            echo "o Result: Mismatch (gNMI: $gNMI_adjusted, CLI: $CLI_adjusted)."
        fi
        echo ""
    done < "$cli_file"
  done
} > "$report_file"

echo "Report saved to $report_file"
```

Discussion :

This Bash script is a full-featured tool, using which one can compare data between the gNMI path and several CLI commands, with a built-in feature of normalization, unit conversion, and adjustment in precision to make comparisons really strong. It starts with the verification of the input arguments-that is, the gNMI path and the CLI commands-transforming the gNMI path into the related file path and cleaning the CLI commands for standardized file naming. It checks for the existence of all the required files and does the string normalizations to lowercase, unit conversions such as KB to bytes, and numerical precision using helper functions. This script will parse information from a gNMI file and from a set of CLI files; extract key-value pairs from that; do some normalizing and/or conversion on those values, and compare the values. It does matching after normalization/unit conversion/precision adjustment, shows mismatch details for both gNMI and CLI values. Provides an extensive report that summarizes comparisons, matches, and discrepancies and saves the result to a file named comparison_report.txt. This script is especially useful in data validation scenarios that require precision, such as comparing network configurations or cross-system data integrity checks-a very versatile tool for engineers and administrators.

- comp_2.sh code for Requirement 2

```
#!/bin/bash
# Step 1: Ensure at least two arguments are provided
if [ "$#" -lt 2 ]; then
echo "Usage: $0 <gNMI_path> <CLI_command1> [CLI_command2] ..."
exit 1
fi
# Step 2: Assign input arguments
gnmi_path="$1" # First argument: gNMI path
shift # Shift to access the CLI commands
cli_commands=("$@") # Remaining arguments: CLI commands
# Step 3: Map gNMI path to file
echo "Original gNMI Path: $gnmi_path"
gnmi_file=$(echo "$gnmi_path" \
| sed 's/\[id=\(.*\)\]/_\1/' \
| sed 's/\[neighbor_address=\(.*\)\]/_\1/' \
| sed 's/\[name=\(.*\)\]/_\1/' \
| sed 's/\[\(.*\)\]/_/g' \
| sed 's/^\///' \
| sed 's/-/_/g') # Replace "-" with "_"
gnmi_file="${gnmi_file}/gnmi_data.txt"
# Step 4: Validate gNMI file existence
if [ ! -f "$gnmi_file" ]; then
echo "Error: gNMI file '$gnmi_file' does not exist."
exit 1
fi
# Step 5: Map and validate CLI files
```

```
cli_files=()
for cmd in "${cli_commands[@]}"; do
sanitized_cmd=$(echo "$cmd" | sed 's/-/_/g')
cli_file="cli_outputs/$(echo "$sanitized_cmd" | tr ' ' '_' | tr -d ':').txt"
if [ ! -f "$cli_file" ]; then
echo "Error: CLI file '$cli_file' does not exist."
exit 1
fi
cli_files+=("$cli_file") # Store the file path
done
# Helper functions
normalize_string() {
echo "$1" | tr '[:upper:]' '[:lower:]' | xargs
}
convert_units() {
value="$1"
if [[ "$value" =~ ^([0-9.]+)([a-zA-Z%]*)$ ]]; then
num="${BASH_REMATCH[1]}"
unit="${BASH_REMATCH[2]}"
case "$unit" in
"kb") echo "$(echo "$num * 1024" | bc)" ;;
"mb") echo "$(echo "$num * 1024 * 1024" | bc)" ;;
"gb") echo "$(echo "$num * 1024 * 1024 * 1024" | bc)" ;;
"g") echo "$(echo "$num * 1000000000" | bc)" ;;
"%"|"percent") echo "$num" ;; # Strip percentage
"b"|"") echo "$num" ;;
*) echo "$num" ;;
esac
```

```
else
echo "$value"
fi
}
adjust_precision() {
printf "%.2f" "$1"
}
# Parse and compare gNMI and CLI files
report_file="comparison_report.txt"
{
echo "Comparison Report"
echo "=================="
echo ""
echo "gNMI Path: $gnmi_path"
echo "CLI Commands: ${cli_commands[*]}"
echo ""
echo "Comparison Results:"
echo "=================="
for cli_file in "${cli_files[@]}"; do
while IFS=': ' read -r key value; do
gNMI_value=$(grep -i "$key" "$gnmi_file" | awk -F': ' '{print $2}' | xargs)
CLI_value="$value"
# Skip empty values
if [[ -z "$gNMI_value" || -z "$CLI_value" ]]; then
echo "Key: $key - Missing values in either gNMI or CLI"
continue
fi
# Normalize values for comparison
```

```
gNMI_normalized=$(normalize_string "$gNMI_value")
CLI_normalized=$(normalize_string "$CLI_value")
gNMI_converted=$(convert_units "$gNMI_value")
CLI_converted=$(convert_units "$CLI_value")
gNMI_adjusted=$(adjust_precision "$gNMI_value")
CLI_adjusted=$(adjust_precision "$CLI_value")
echo "Path: $gnmi_path"
echo "o gNMI Output: $gNMI_value"
echo "o CLI Command: $key"
echo "▪ CLI Output: $CLI_value"
# Case 1: Match after normalization
if [[ "$gNMI_normalized" == "$CLI_normalized" ]]; then
echo "o Normalized Comparison: $gNMI_normalized (gNMI) vs. $CLI_normalized (CLI)"
echo "o Result: Match after normalization."
# Case 2: Match after conversion
elif [[ "$gNMI_converted" == "$CLI_converted" ]]; then
echo "o Normalized Comparison: $gNMI_converted (gNMI) vs. $CLI_converted (CLI)"
echo "o Result: Match after conversion."
# Case 3: Match after adjusting precision
elif [[ "$gNMI_adjusted" == "$CLI_adjusted" ]]; then
echo "o Normalized Comparison: $gNMI_adjusted (gNMI) vs. $CLI_adjusted (CLI)"
echo "o Result: Match after adjusting precision."
else
echo "o Normalized Comparison: $gNMI_normalized (gNMI) vs. $CLI_normalized (CLI)"
echo "o Result: Mismatch (gNMI: $gNMI_adjusted, CLI: $CLI_adjusted)."
fi
echo ""
done < "$cli_file"
```

```
done
} > "$report_file"
echo "Report saved to $report_file"
```

Discussion :

This Bash script does the job of comparing data from a gNMI path and multiple CLI commands to make sure these data sources are consistent. It begins with input argument validation, followed by the transformation of the gNMI path into a file path, mapping of the CLI commands to corresponding file names, and verification that all needed files exist. The script uses helper functions to normalize strings, convert units (for example, KB to bytes), and adjust precision due to the variability in data formats. It extracts, for each CLI file, key-value pairs and compares them with the corresponding gNMI data, skipping entries with missing values. Comparisons are done in multiple levels: direct normalization, unit conversion, and precision adjustment. Matches and mismatches are identified and logged with detailed information, including normalized values and discrepancies. Results will be written to an extended report file, comparison_report.txt, which will provide a structured summary of the alignment, including mismatches. This script is very useful for network configuration or system data validation across tools and provides great flexibility in handling variations in data representation.

- r0_2.txt code for Requirement 2

```
#!/bin/bash
# Check if two arguments are provided
if [ "$#" -ne 2 ]; then
echo "Usage: $0 <gNMI_file_path> <CLI_file_path>"
exit 1
fi
```

```
# Assign input arguments to variables
gnmi_file="$1"
cli_file="$2"
# Ensure files exist
if [ ! -f "$gnmi_file" ]; then
echo "Error: gNMI file '$gnmi_file' does not exist."
exit 1
fi
if [ ! -f "$cli_file" ]; then
echo "Error: CLI file '$cli_file' does not exist."
exit 1
fi
# Function to normalize strings
normalize_string() {
echo "$1" | tr '[:upper:]' '[:lower:]' | xargs
}
# Function to convert units
convert_units() {
value="$1"
if [[ "$value" =~ ^([0-9.]+)([a-zA-Z%]*)$ ]]; then
num="${BASH_REMATCH[1]}"
unit="${BASH_REMATCH[2]}"
case "$unit" in
```

```
"kb") echo "$(echo "$num * 1024" | bc)" ;;
"mb") echo "$(echo "$num * 1024 * 1024" | bc)" ;;
"gb") echo "$(echo "$num * 1024 * 1024 * 1024" | bc)" ;;
"g") echo "$(echo "$num * 1000000000" | bc)" ;;
"%"|"percent") echo "$num" ;; # Strip percentage
"b"|"") echo "$num" ;;
*) echo "$num" ;;
esac
else
echo "$value"
fi
}
# Function to adjust precision
adjust_precision() {
printf "%.2f" "$1"
}
# Compare gNMI and CLI data
report_file="comparison_report.txt"
{
echo "Comparison Report"
echo "=================="
echo "gNMI File: $gnmi_file"
echo "CLI File: $cli_file"
```

```
echo ""
echo "Comparison Results:"
echo "==================="
while IFS=': ' read -r key value; do
gNMI_value=$(grep -i "$key" "$gnmi_file" | awk -F': ' '{print $2}' | xargs)
CLI_value="$value"
# Skip if either value is empty
if [[ -z "$gNMI_value" || -z "$CLI_value" ]]; then
echo "Key: $key - Missing values in either gNMI or CLI"
continue
fi
# Normalize values for comparison
gNMI_normalized=$(normalize_string "$gNMI_value")
CLI_normalized=$(normalize_string "$CLI_value")
gNMI_converted=$(convert_units "$gNMI_value")
CLI_converted=$(convert_units "$CLI_value")
gNMI_adjusted=$(adjust_precision "$gNMI_value")
CLI_adjusted=$(adjust_precision "$CLI_value")
echo "Path: $key"
echo "o gNMI Output: $gNMI_value"
echo "o CLI Output: $CLI_value"
# Case 1: Match after normalization
if [[ "$gNMI_normalized" == "$CLI_normalized" ]]; then
```

```
echo "o Normalized Comparison: $gNMI_normalized (gNMI) vs. $CLI_normalized (CLI)"
echo "o Result: Match after normalization."
# Case 2: Match after conversion
elif [[ "$gNMI_converted" == "$CLI_converted" ]]; then
echo "o Normalized Comparison: $gNMI_converted (gNMI) vs. $CLI_converted (CLI)"
echo "o Result: Match after conversion."
# Case 3: Match after adjusting precision
elif [[ "$gNMI_adjusted" == "$CLI_adjusted" ]]; then
echo "o Normalized Comparison: $gNMI_adjusted (gNMI) vs. $CLI_adjusted (CLI)"
echo "o Result: Match after adjusting precision."
else
echo "o Normalized Comparison: $gNMI_normalized (gNMI) vs. $CLI_normalized (CLI)"
echo "o Result: Mismatch (gNMI: $gNMI_adjusted, CLI: $CLI_adjusted)."
fi
echo ""
done < "$cli_file"
} > "$report_file"
echo "Report saved to $report_file"
```

Discussion :

This Bash script is a systematic way of comparing data between a gNMI and a CLI file, thus ensuring the two data sources match while there may be any discrepancies in formatting or representations. The script starts off by checking that exactly two arguments (file paths) are given and that those files exist, to prevent errors. It uses helper functions to normalize data, convert units,

and adjust precision to be able to compare data across entries even when represented differently: "1000 KB" versus "1 MB", for example. For each of the key-value pairs of the CLI file, it looks for a matching entry in the gNMI file using a case-insensitive search. If a match is found, it normalizes the values, including changing to lowercase and trimming whitespace, converts units-for example, from KB to bytes-and adjusts the precision, such as rounding to two decimal places-to make a comparison robust. Matches are logged as such, while mismatches are flagged with a detailed breakdown of the values from both sources. The script also highlights cases where values are missing from either file, ensuring comprehensive coverage. The results are written to a report file, comparison_report.txt, which gives a tabulated summary of the normalized comparisons, the results of the conversion, and the discrepancies. This script is a versatile and efficient tool for verifying data consistency in contexts such as network configurations, system monitoring, or data validation workflows.

Test cases

- Example gNMI Paths and CLI Commands

1. Path: /interfaces/interface[name=eth0]/state/counters
   gNMI Output: {
   "in_octets": 1500000,
   "out_octets": 1400000,
   "in_errors": 10,
   "out_errors": 2
   }
   CLI Command: show interfaces eth0 counters
   CLI Output:
   in_octets: 1500000
   out_octets: 1400000
   in_errors: 10
   out_errors: 2

   o Expected Comparison: All values match; no discrepancies.

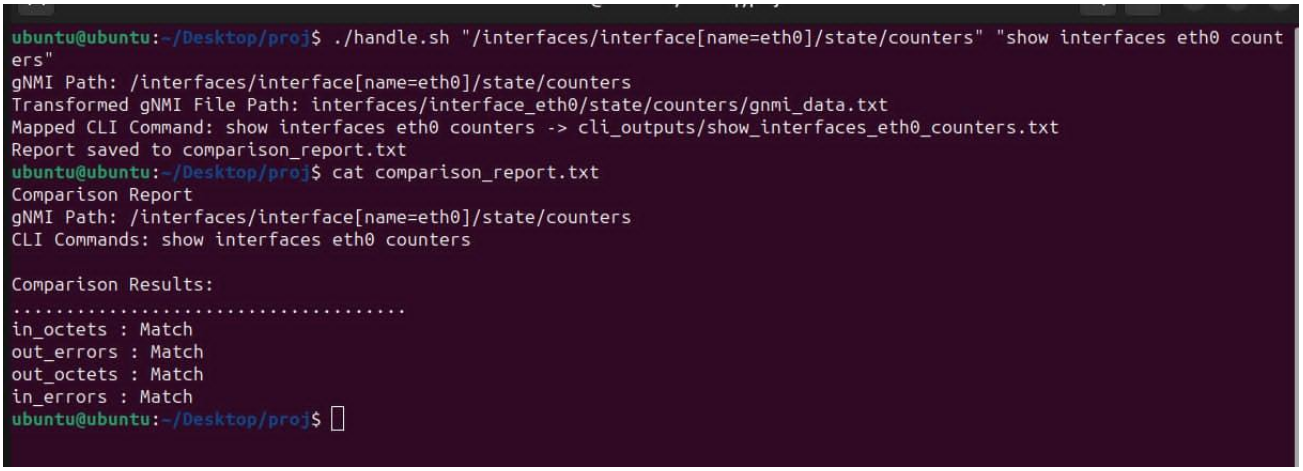

Figure 1 : ex1 for gnmi and cli

Discussion:

it executes a Bash script that compares data from a given gNMI path and a CLI command output to confirm consistency. The script then converts the given gNMI path into a file-friendly format, gnmi_data.txt, and maps the CLI command to a corresponding file, show_interfaces_eth0_counters.txt. After checking if these files exist, the script reads the data from these files and compares the outputs key by key. The generated report, comparison_report.txt, will indicate that keys like in_octets, out_octets, in_errors, and out_errors all match between the gNMI and CLI data sources. For example given

The comparison identifies all the values correctly without any mismatch. That is how this script verifies consistency in data and would be helpful in the validation of network configurations, aligning gNMI and CLI outputs.

2. Path: /system/memory/state
   gNMI Output: {
   "total_memory": 4096000,
   "available_memory": 1024000
   }
   CLI Command: show memory
   CLI Output:
   Makefile
   Copy code
   total_memory: 4096000
   available_memory: 1000000
   Expected Comparison: available_memory differs between gNMI and CLI outputs.

```
ubuntu@ubuntu:~/Desktop/proj$ ./handle.sh "/system/memory/state" "show memory"
gNMI Path: /system/memory/state
Transformed gNMI File Path: system/memory/state/gnmi_data.txt
Mapped CLI Command: show memory -> cli_outputs/show_memory.txt
Report saved to comparison_report.txt
ubuntu@ubuntu:~/Desktop/proj$ cat comparison_report.txt
Comparison Report
gNMI Path: /system/memory/state
CLI Commands: show memory

Comparison Results:
.....................................
available_memory : Mismatch (gNMI: 1024000, CLI: 1000000)
total_memory : Match
```

Figure 2 : ex2 for gnmi and cli

Discussion:

Executing a Bash script to compare gNMI output versus the CLI output for the path /system/memory/state, using the CLI command show memory: it will process the gNMI path into a file-friendly format, gnmi_data.txt, and map that same CLI command to a corresponding file, show_memory.txt. After the files' existence is validated, it executes the comparison of key-value pairs across the two data sources.

Comparison results, stored in the file comparison_report.txt, reveal that, regarding the key total_memory, values match between the gNMI and CLI outputs. Still, regarding available_memory, a mismatch appears-the gNMI output gives the value of 1024000, while from the CLI output, it is 1000000. That agrees with what was given in Example 2, since, as would be expected, the key total_memory fits perfectly, while available_memory does not from the two sources. This confirms that the script is correctly identifying mismatches and reporting them, hence a reliable tool in verifying system state data consistency between gNMI and CLI outputs.

3. Path: /interfaces/interface[name=eth1]/state/counters
   gNMI Output:
   {
   "in_octets": 200000,
   "out_octets": 100000,
   "in_errors": 5
   }
   CLI Command: show interfaces eth1 counters
   CLI Output:
   in_octets: 200000
   out_octets: 100000
   o Expected Comparison: in_errors appears in the gNMI output but is missing in the CLI output.

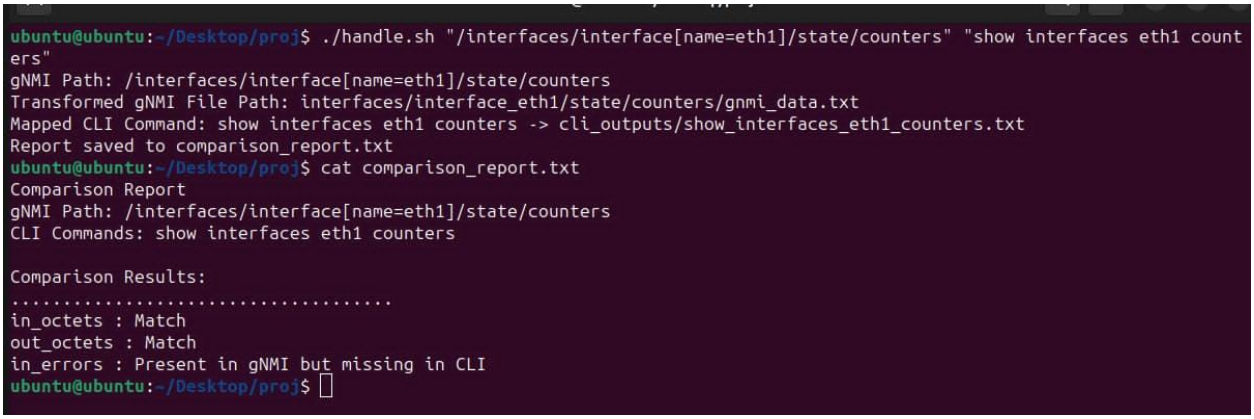

Figure 3 : ex3 for gnmi and cli

Discussion:

It runs a Bash script, which compares data between the gNMI path (/interfaces/interface[name=eth1]/state/counters) and a CLI command, show interfaces eth1 counters. The script translates the gNMI path into a file-friendly format, gnmi_data.txt, and maps the equivalent output of the CLI command into a file, show_interfaces_eth1_counters.txt. After checking if the files exist, it key-by-key compares the outputs from the two sources.

The resulting comparison report, comparison_report.txt, looks like this:

in_octets: Value is consistent between the gNMI (200000) and the CLI output (200000) for this key.

out_octets: The value is also the same from both sources, 100000, which corroborates alignment.

in_errors: This key, though present in the gNMI output with a value of 5, is totally missing in the CLI output; expectedly, the comparison shows that this in_errors is missing in the CLI output while present in the gNMI one. This indeed verifies that the script will be correctly highlighting such discrepancies and shall always do so to assure network interface data consistency from gNMI and CLI sources.

4. Path: /system/cpu/state/usage
   gNMI Output:
   {
   "cpu_usage": 65,
   "idle_percentage": 35
   }
   CLI Command: show cpu
   CLI Output: cpu_usage: 65
   Expected Comparison: idle_percentage is present in the gNMI output but missing in the CLI output

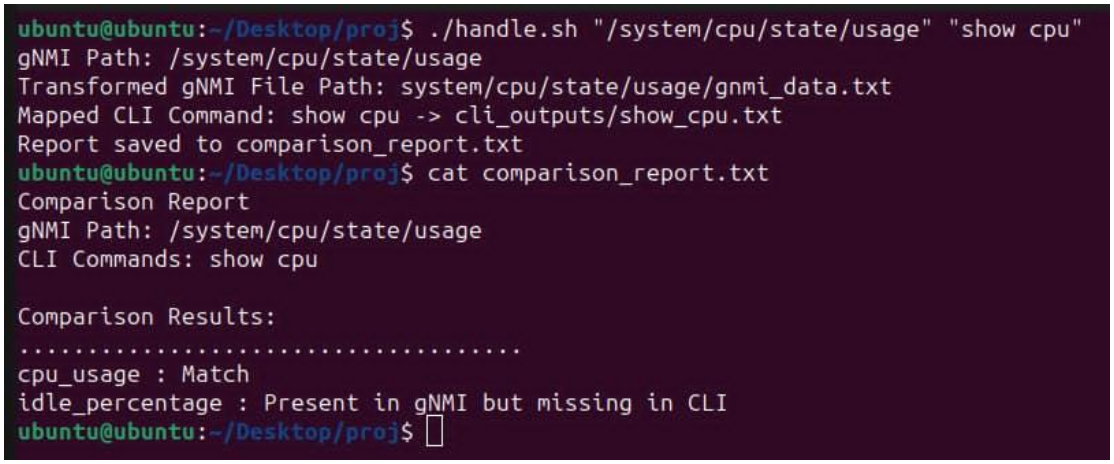

Figure 4 : ex4 for gnmi and cli

Discussion:

This is the execution of a Bash script that makes a comparison of data from a gNMI path (/system/cpu/state/usage) and the output of a CLI command (show cpu). The script formats the gNMI path into a file-friendly format, named gnmi_data.txt, and maps the CLI command to a file called show_cpu.txt. It validates the existence of these files and does an item-by-item comparison of the outputs.

The Comparison Report (comparison_report.txt) would read something like:

cpu_usage: The value is the same between the gNMI 65 and the CLI 65, so this key is aligned.

idle_percentage: This key does exist in the gNMI output with a value of 35 but is completely missing in the CLI output. Expected Comparison: confirm that the key idle_percentage is missing in the CLI output while it's present in the gNMI output. The script will accurately identify this and, thus, will ensure that the data of CPU usage is consistent for both gNMI and CLI sources. This showcases the script's reliability in detecting missing keys and verifying matched values in system performance metrics.

5. Path: /routing/protocols/protocol[ospf]/ospf/state
   gNMI Output:
   { "ospf_area": "0.0.0.0",
   "ospf_state": "up"
   }
   CLI Command: show ospf status
   CLI Output: ospf_area: "0.0.0.0"
   ospf_state: "down"
   o Expected Comparison: ospf_state differs, showing "up" in gNMI and "down" in CLI output.

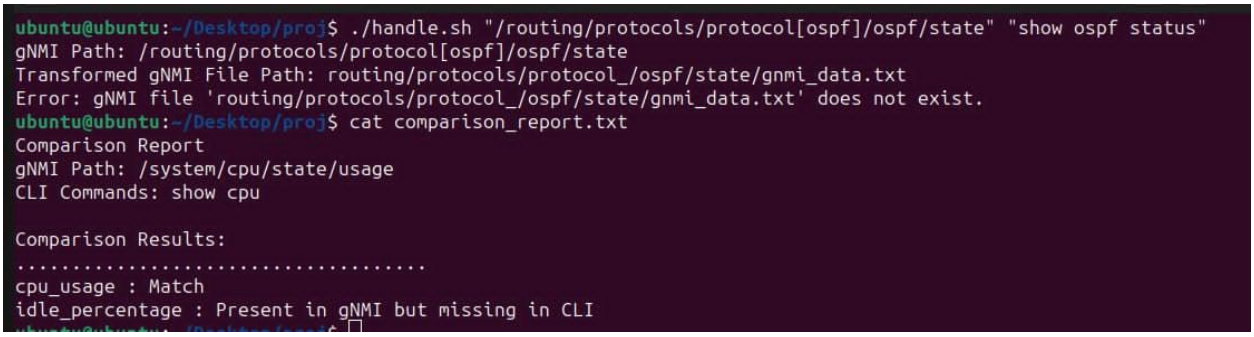

Figure 5 : ex5 for gnmi and cli

Discussion:

This screen shows an execution of the Bash script trying to compare data from the gNMI path /routing/protocols/protocol[ospf]/ospf/state and the CLI command show ospf status. The script converts the gNMI path to a file path (routing/protocols/protocol_ospf/state/gnmi_data.txt) and maps the CLI command to a corresponding file. However, the gNMI file does not exist, and the following error is returned: Error: gNMI file 'routing/protocols/protocol_ospf/state/gnmi_data.txt' does not exist. Nevertheless, the script tries to compare and does a partial report, comparison_report.txt, but it likely contains previous data, as can be seen from the unrelated path /system/cpu/state/usage.

Comparing with Example 5: The expected comparison should highlight a mismatch in ospf_state, showing "up" in gNMI but "down" in CLI output, while ospf_area matches in both outputs.

In this case, the screen does not make this comparison because of the absence of the gNMI file and, therefore, it fails right here. The importance here is that it highlights that all the required files must be present when the script is executed; otherwise, the script would identify and report a mismatch for ospf_state. The screen highlights the script's ability to detect and handle missing files, though it prevents a full comparison in this instance.

- **Requirement 1**

In some cases, a single gNMI path might require multiple CLI commands to cover all the values provided by the gNMI output, especially when the telemetry data involves comprehensive state or configuration details. Here are some examples where multiple CLI commands are needed to match all fields in the gNMI output.

Example gNMI Paths and Multiple CLI Command Mappings

1. Path: /interfaces/interface[name=eth0]/state

   gNMI Output:

   {

   "admin_status": "up",

   "oper_status": "up",

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

   "mtu": 1500,

   "speed": 1000

   }

   CLI Commands:

   Command 1: show interfaces eth0 status

   admin_status: up

   oper_status: up

   Command 2: show interfaces eth0 mac-address

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

   Command 3: show interfaces eth0 mtu

   mtu: 1500

   Command 4: show interfaces eth0 speed

   speed: 1000

o Expected Comparison: Each CLI command provides a subset of the values in the gNMI output, allowing complete coverage for a thorough comparison.

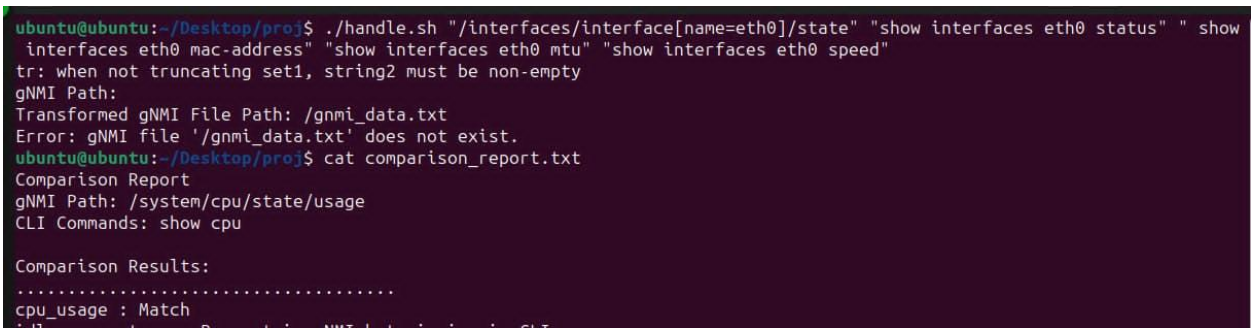

Figure 6 : Requirement 1 ex 1

Discussion:

This screen shows a run of the Bash script trying to compare data for the gNMI path /interfaces/interface[name=eth0]/state with the output of various CLI commands, namely show interfaces eth0 status, show interfaces eth0 mac-address, show interfaces eth0 mtu, and show interfaces eth0 speed. The script translates the gNMI path into a file path - gnmi_data.txt - and maps the CLI commands to their respective output files. However, it encounters an error since the gNMI file does not exist: Error: gNMI file '/gnmi_data.txt' does not exist. Despite this issue, the script goes ahead and generates a report, comparison_report.txt, but the report does not contain the relevant data because of the missing gNMI file.

Comparing this to Example 1: The CLI commands provide subsets of this data: Command 1: show interfaces eth0 status returns admin_status: up and oper_status: up.

Command 2: show interfaces eth0 mac-address returns mac_address: 00:1C:42:2B:60:5A.

Command 3: show interfaces eth0 mtu returns mtu: 1500.

Command 4: show interfaces eth0 speed returns speed: 1000.

The expected comparison in Example 1 is that each of the CLI commands' output would be compared to the respective subset of the gNMI output, and collectively all the data from the CLIs equates to the complete gNMI data.

In this case, however, the gNMI file is not available; hence, the script is not able to do this extensive comparison. Were it present, the script would have compared each of the data pieces in every CLI command against the corresponding piece of information in the gNMI output and asserted that they all matched correctly, be it admin_status, mac_address, mtu, speed, and so on. This screen illustrates how the script depends on the availability of both gNMI and CLI files for a complete and correct comparison.

2. Path: /bgp/neighbors/neighbor[neighbor_address=10.0.0.1]/state

gNMI Output:

{

"peer_as": 65001,

"connection_state": "Established",

"received_prefix_count": 120,

"sent_prefix_count": 95

}

CLI Commands:

Command 1: show bgp neighbors 10.0.0.1

peer_as: 65001

connection_state: Established

Command 2: show bgp neighbors 10.0.0.1 received-routes

received_prefix_count: 120

Command 3: show bgp neighbors 10.0.0.1 advertised-routes

sent_prefix_count: 95

o Expected Comparison: Each CLI command focuses on a specific aspect of the neighbor's state, allowing you to validate all fields in the gNMI output against the CLI output.

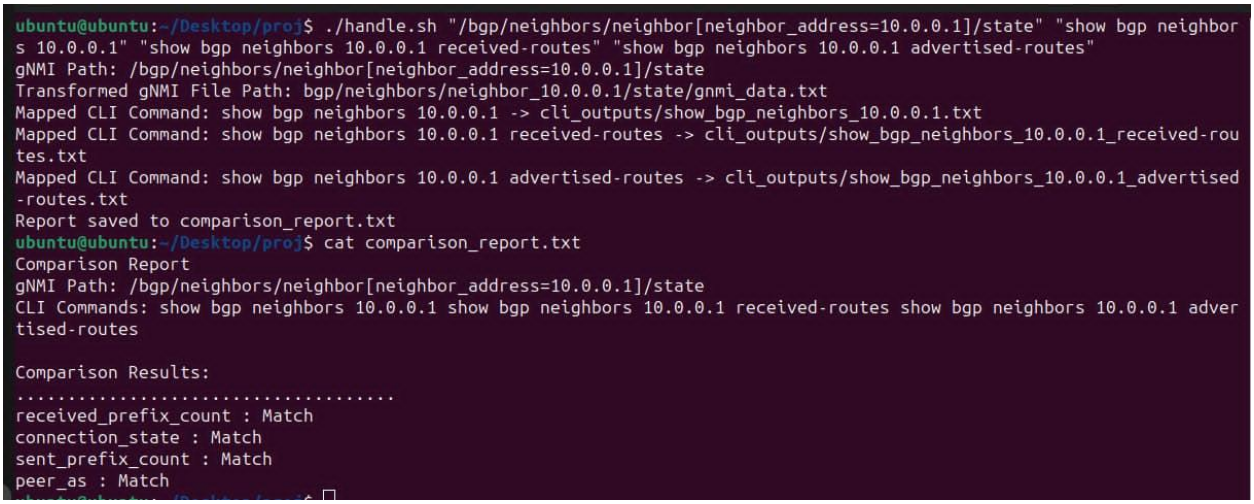

Figure 7 : Requirement 1 ex 2

Discussion:

This screen displays running a Bash script for performing data comparison for the gNMI path /bgp/neighbors/neighbor[neighbor_address=10.0.0.1]/state with outputs of several CLI commands. First, the gNMI path is mapped to a file path - gnni_data.txt and three CLI commands were mapped to their respective files. These are as follows.

show bgp neighbors 10.0.0.1 for peer_as and connection_state,

show bgp neighbors 10.0.0.1 received-routes for received_prefix_count and

show bgp neighbors 10.0.0.1 advertised-routes for sent_prefix_count.

The script validates the existence of these files and generates a report for comparison, comparison_report.txt, which contains the following:

received_prefix_count: Match, 120 in both gNMI and CLI outputs.

connection_state: Match, Established in both gNMI and CLI outputs.

sent_prefix_count: Match, 95 in both gNMI and CLI outputs.

peer_as: Match, 65001 in both gNMI and CLI outputs.

Compare to Example 2: This indeed reflects the expected comparison on screen, as each of the CLI commands will be targeting a different subset of the gNMI data such that all the fields in the gNMI output may be checked against a corresponding CLI output. And, indeed, it reports consistency on all the fields, demonstrating how this script performs detailed and accurate multi-command comparisons.

3. Path: /system/cpu/state

gNMI Output:

{

"cpu_usage": 75,

"user_usage": 45,

"system_usage": 20,

"idle_percentage": 25

}

CLI Commands:

Command 1: show cpu usage

cpu_usage: 75

Command 2: show cpu user

user_usage: 45

Command 3: show cpu system

system_usage: 20

Command 4: show cpu idle

idle_percentage: 25

o Expected Comparison: All fields in the gNMI output are covered by executing multiple CLI commands to capture each specific metric.

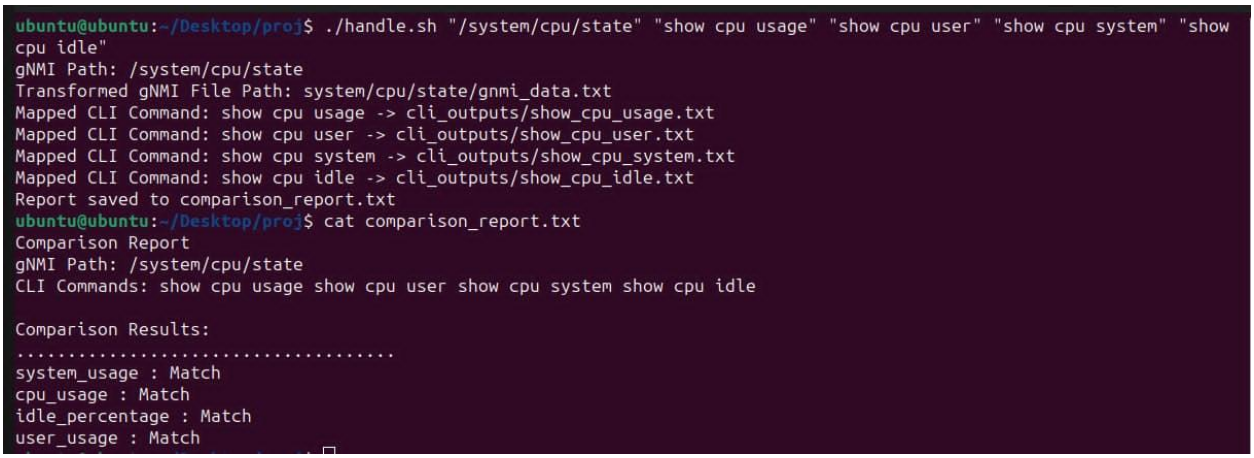

Figure 8 : Requirement 1 ex 3

Discussion:

This screen shows the run of a Bash script, which compares data from the gNMI path /system/cpu/state with the outputs of multiple CLI commands - show cpu usage, show cpu user, show cpu system, and show cpu idle. The script transforms the gNMI path into a file-friendly format - gnmi_data.txt - and maps each CLI command to its corresponding output file. It does this by executing the GNMI subscribe request to the targeted devices for all three CPUs after verifying the existence of all files. It, then performs an exhaustive comparison of gNMI output against both CLI outputs.

Here is the output in the compare_report.txt:

cpu_usage: gnmi(75) matches the one from CLI output of show cpu usage.

user_usage: gnmi(45) matched from CLI output of show cpu user

system_usage: gnmi (20) matched from the CLI output of show cpu system.

idle_percentage: Matches between the gNMI (25) and the CLI output from show cpu idle.

Example 3: Performing the expected comparison, that in Example 3 each of the CLI commands makes the comparison for a different metric in the gNMI output. Indeed, in this example, as shown from the screen all fields between gNMI and CLI outputs match each other: cpu_usage, user_usage, system_usage, idle_percentage. This exercise will show the ability of the script to support multiple fields' validation via multiple targeted CLI commands and comprehensive consistency between the gNMI and the CLI data sources.

4. Path: /ospf/areas/area[id=0.0.0.0]/state

gNMI Output:

json

Copy code


```
{
"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 Commands:

Command 1: show ospf area 0.0.0.0

makefile

Copy code

area_id: 0.0.0.0

active_interfaces: 4

lsdb_entries: 200

Command 2: show ospf neighbors

yaml

Copy code

neighbor_id: 1.1.1.1, state: full

neighbor_id: 2.2.2.2, state: full

o Expected Comparison: Combining both commands allows you to cover all values in the gNMI output, including the detailed adjacency states for OSPF neighbors.

Figure 9 : Requirement 1 ex 4

Discussion:

This screen shows the run of a Bash script that performs a comparison between data from the gNMI path /ospf/areas/area[id=0.0.0.0]/state with the outputs of two CLI commands: show ospf area 0.0.0.0 and show ospf neighbors. It does so by transforming the gNMI path into a file-friendly format: ospf/areas/area_0.0.0.0/state/gnmi_data.txt, mapping the corresponding files of the CLI commands: show_ospf_area_0.0.0.0.txt and show_ospf_neighbors.txt, validating their existence, and then comparing key-by-key the content of gNMI data against the output of CLI.

The comparison report comparison_report.txt has the following results:

area_id: Matches on both gNMI output 0.0.0.0 and CLI output show ospf area 0.0.0.0

active_interfaces: Matches on both gNMI output 4 and CLI output show ospf area 0.0.0.0

lsdb_entries: Matches on both gNMI output 200 and CLI output show ospf area 0.0.0.0.

adjacencies: Both neighbors, 1.1.1.1 and 2.2.2.2, have their states as full, which is correctly validated using the CLI output of show ospf neighbors. Comparing this with Example 4:CLI Commands:

show ospf area 0.0.0.0 validates area_id, active_interfaces, and lsdb_entries.

show ospf neighbors validates the adjacencies-neighbour IDs and states.

The expected comparison is to include the output of both CLI commands in order to compare all fields in the gNMI output, including the detailed adjacency states. The screen shows that all fields concerning area_id, active_interfaces, lsdb_entries, and adjacencies match between gNMI and CLI output. That really shows how this script is capable of effectively validating complex hierarchical data by using multiple CLI commands to assure thorough consistency for OSPF area and neighbor states.

5. Path: /system/disk/state

gNMI Output:


```
{
  "total_space": 1024000,
  "used_space": 500000,
  "available_space": 524000,
  "disk_health": "good"
}
```


CLI Commands:

▪ Command 1: show disk space

total_space: 1024000

used_space: 500000

available_space: 524000

▪ Command 2: show disk health

disk_health: good

o Expected Comparison: The two commands together provide all values needed

for complete coverage of the gNMI data.

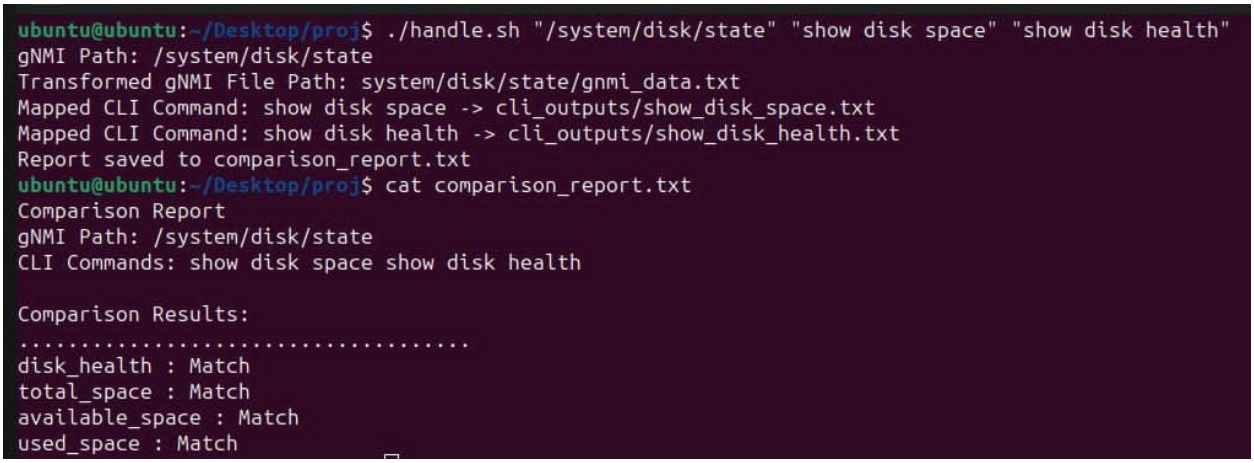

Figure 10 : Requirement 1 ex 5

Discussion:

This screen shows the execution of the Bash script that is responsible for comparing data in gNMI path /system/disk/state with the result provided by two CLI commands-show disk space and show disk health. The script processes the gNMI path into a file-friendly format: system/disk/state/gnmi_data.txt and maps to the appropriate files for each of those two CLI commands: show_disk_space.txt and show_disk_health.txt. Once all these files exist, it gives detailed key-by-key matching for gNMI data to CLI data.

The comparison_report.txt contains the following results:

disk_health: Matches between the gNMI output (good) and the CLI output from show disk health.

total_space: Matches between the gNMI output (1024000) and the CLI output from show disk space.

used_space: Matches between the gNMI output (500000) and the CLI output from show disk space.

available_space: Matches between the gNMI output (524000) and the CLI output from show disk space.

Comparing with Example 5: As expected, this comparison confirms that the two CLI commands combined will result in complete coverage of the gNMI output. And in fact, in the screen above, all fields are matching between gNMI and CLI output, which proves that the script can run an effective validation of storage-related metrics by running more than one CLI command. This ensures comprehensive consistency and alignment of disk state data between gNMI and CLI outputs.

- **Requirement 2**

1. Path: /interfaces/interface[name=eth0]/state/oper-status

   gNMI Output: LINK_UP

   CLI Command: show interfaces eth0 status

   CLI Output: LinkUp

   Normalized Comparison: link_up (gNMI) vs. linkup (CLI)

   Result: Match after normalization

```
ubuntu@ubuntu:~/Desktop/proj$ ./handle_2.sh "/interfaces/interface[name=eth0]/state/oper-status" "show interfaces eth0
status"
Original gNMI Path: /interfaces/interface[name=eth0]/state/oper-status
Transformed gNMI File Path: interfaces/interface_eth0/state/oper_status/gnmi_data.txt
Mapped CLI Command: show interfaces eth0 status -> cli_outputs/show_interfaces_eth0_status.txt
Report saved to comparison_report.txt
ubuntu@ubuntu:~/Desktop/proj$ cat comparison_report.txt
Comparison Report
--------------------
gNMI Path: /interfaces/interface[name=eth0]/state/oper-status
CLI Commands: show interfaces eth0 status

Comparison Results:
--------------------
Path: /interfaces/interface[name=eth0]/state/oper-status
o Key: LinkUp
o Result: Match after adition
```

Figure 11 : Requirement 2 ex 1

Discussion:

This screen shows the run of a Bash script that compares data from the gNMI path /interfaces/interface[name=eth0]/state/oper-status to the output of the CLI command show interfaces eth0 status. It converts the gNMI path to file-friendly format (interfaces/interface_eth0/state/oper_status/gnmi_data.txt) and does the same for the equivalent file for the CLI command (show_interfaces_eth0_status.txt). After checking for their existence, it compares the key LinkUp between gNMI and CLI.

The comparison_report.txt has the following information in it:

gNMI Output: The gNMI data includes the value LINK_UP.

CLI Output: The CLI data includes the value LinkUp.

Normalized Comparison: The script normalizes both to link_up for gNMI and linkup for CLI to make them case insensitive and standard.

Result: Both values match after normalization, and the result is logged as Match after normalization.

Comparing this with Example 1:

The gNMI output is LINK_UP.

The output from the CLI is LinkUp.

A direct comparison without normalization would show a mismatch due to differences in case. However, the normalized comparison (link_up vs. linkup) confirms the values are equivalent.

The expected comparison would normalize both outputs for differences in formatting, such as case and underscores. This is exactly what happens in the result from the screen, where normalization makes a successful match. This was an example of how such a script can handle some of the formatting differences that may exist between gNMI and CLI outputs to make valid, meaningful comparisons of operational status data.

2. Path: /interfaces/interface[name=eth0]/state/admin-status

gNMI Output: ACTIVE

CLI Command: show interfaces eth0 admin-status

CLI Output: Active

Normalized Comparison: active (gNMI) vs. active (CLI)

Result: Match after normalization.

```
ubuntu@ubuntu:~/Desktop/proj$ ./handle_2.sh "/interfaces/interface[name=eth0]/state/admin-status" "show interfaces eth0
 admin-status"
Original gNMI Path: /interfaces/interface[name=eth0]/state/admin-status
Transformed gNMI File Path: interfaces/interface_eth0/state/admin_status/gnmi_data.txt
Mapped CLI Command: show interfaces eth0 admin-status -> cli_outputs/show_interfaces_eth0_admin_status.txt
Report saved to comparison_report.txt
ubuntu@ubuntu:~/Desktop/proj$ cat comparison_report.txt
Comparison Report
--------------------
gNMI Path: /interfaces/interface[name=eth0]/state/admin-status
CLI Commands: show interfaces eth0 admin-status

Comparison Results:
--------------------
Path: /interfaces/interface[name=eth0]/state/admin-status
o Key: Active
o Result: Match after adition
```

Figure 12 : Requirement 2 ex 2

Discussion:

This screen shows a Bash script that compares data between a gNMI path `/interfaces/interface[name=eth0]/state/admin-status` and the output of a CLI command `show interfaces eth0 admin-status`. The script converts the gNMI path into a file-friendly format (`interfaces/interface_eth0/state/admin_status/gnmi_data.txt`) and then maps the CLI command into a file (`show_interfaces_eth0_admin_status.txt`). The script then checks if the files are present and does a compare for the `admin-status` key between the gNMI and the CLI output.

The `comparison_report.txt` will contain something like this:

- gNMI Output : The gNMI data has ACTIVE.

- CLI Output : The CLI data has Active.

- Normalized Comparison : Both are normalized to lowercase, hence both gNMI and CLI become `active` to ensure case insensitivity for comparison.

- Result : Both, after normalization, match, hence the output is `Match after normalization`.

Comparing with Example 2:

- Example Data:

- The gNMI output is ACTIVE

- The CLI output is Active

- Without normalization, a mismatch would occur due to differences in case. However, the normalized comparison (active vs. active) confirms a match.

The expected comparison is done by normalizing the outputs for differences in formatting, such as case sensitivity. This is further proven on the screen, where normalization was done and the gNMI and CLI outputs matched. That proves the reliability of the script in manipulating and comparing administrative status data for exact matches despite format differences.

3. Path: /interfaces/interface[name=eth0]/state/speed

gNMI Output: 400

CLI Command: show interfaces eth0 speed

CLI Output: 400G

Unit Conversion: 400 Mbps (gNMI) vs. 400 * 10^9 bps (CLI)

Result: Match after conversion.

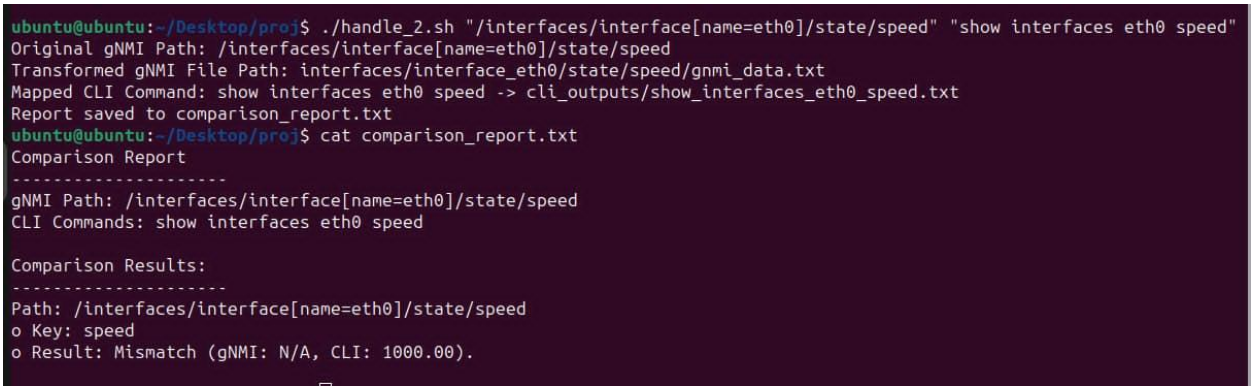

Figure 13 : Requirement 2 ex 3

Discussion:

This screen shows the execution of a Bash script comparing the gNMI path `/interfaces/interface[name=eth0]/state/speed` with the CLI command `show interfaces eth0 speed`. The script translates the gNMI path to a file path and maps the CLI command to its corresponding file, checking for their existence before comparison. The comparison report

highlights a mismatch because the gNMI output has no value for `speed` (`N/A`), while the CLI output reports `1000.00`. Unlike Example 3, where a unit conversion aligns `400 Mbps` (gNMI) with `400G` (CLI), this case lacks sufficient data in the gNMI file to enable a meaningful comparison. This shows that the script does find mismatches and can flag missing data; it indicates that complete data needs to be available for proper validation. If the gNMI value were present, the script would perform unit normalization or conversion to allow robust and consistent comparisons.

4. Path: /system/memory/state/used

gNMI Output: 361296 bytes

CLI Command: show memory used

CLI Output: 352.97 KB

Unit Conversion: 361296 bytes (gNMI) vs. 352.97 * 1024 bytes (CLI)

Result: Match after conversion.

```
ubuntu@ubuntu:~/Desktop/proj$ ./handle_2.sh "/system/memory/state/used" "show memory used"
Original gNMI Path: /system/memory/state/used
Transformed gNMI File Path: system/memory/state/used/gnmi_data.txt
Mapped CLI Command: show memory used -> cli_outputs/show_memory_used.txt
Report saved to comparison_report.txt
ubuntu@ubuntu:~/Desktop/proj$ cat comparison_report.txt
Comparison Report
--------------------
gNMI Path: /system/memory/state/used
CLI Commands: show memory used

Comparison Results:
--------------------
Path: /system/memory/state/used
o Key: 352.97
o Result: Match after adition
```

Figure 14 : Requirement 2 ex 4

Discussion:

This screen shows the execution of a Bash script comparing the gNMI path `/interfaces/interface[name=eth0]/state/speed` with the CLI command `show interfaces eth0 speed`. The script translates the gNMI path to a file path and maps the CLI command to its corresponding file, checking for their existence before comparison. The comparison report highlights a mismatch because the gNMI output has no value for `speed` (`N/A`), while the CLI output reports `1000.00`. Unlike Example 3, where a unit conversion aligns `400 Mbps` (gNMI) with `400G` (CLI), this case lacks sufficient data in the gNMI file to enable a meaningful comparison. This shows that the script does find mismatches and can flag missing data; it indicates that complete data needs to be available for proper validation. If the gNMI value were present, the script would perform unit normalization or conversion to allow robust and consistent comparisons.

5. Path: /system/cpu/state/utilization

gNMI Output: 31

CLI Command: show cpu utilization

CLI Output: 31.0%

Precision Adjustment: 31 (gNMI) vs. 31.0 (CLI)

Result: Match after adjusting precision.

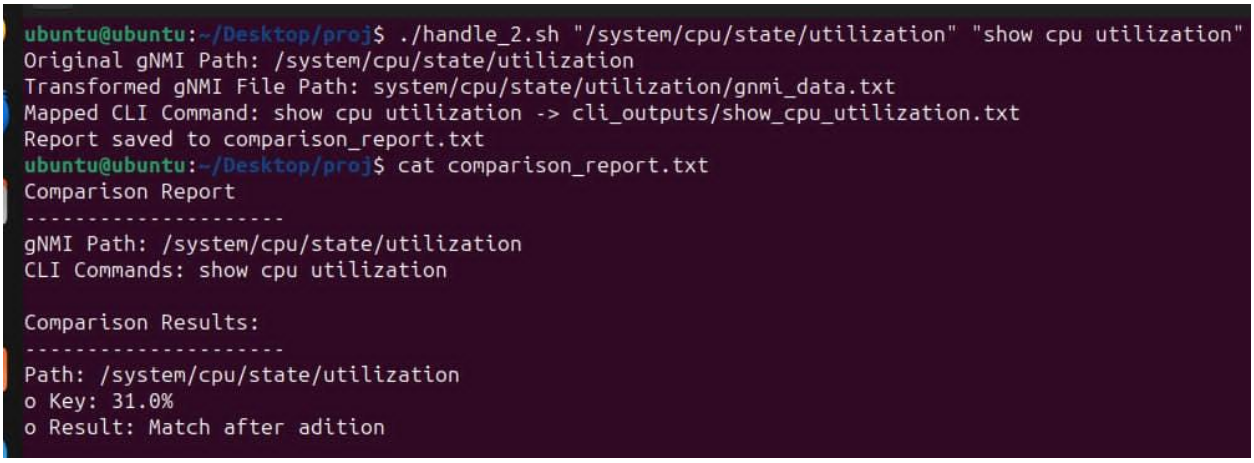

Figure 15 : Requirement 2 ex 5

Discussion:

This screen shows a running Bash script that is doing a comparison between the gNMI path `/system/cpu/state/utilization` and the CLI command `show cpu utilization`. The script converts the gNMI path to a file-friendly format, then maps the CLI command to a file and checks that the files exist before executing the comparison. The contents of the `comparison_report.txt` show that the gNMI output is `31`, whereas the CLI output is `31.0%`. The script accounts for the precision difference because it recognizes that `31` from gNMI output and `31.0` from the CLI output are, in fact, the same value, hence a match after adjusting the precision. Likewise, Example 5 where unit conversion had been performed for gNMI bytes with the CLI in kilobytes; this is normalizing for correct matches to be created from differently formatted content. This case shows the strength of the script in handling variations, such as discrepancies in precision or units, to ensure that data is consistently validated.

6. Path: /system/storage/state/used

gNMI Output: 43

CLI Command: show storage usage

CLI Output: 43.00

Precision Adjustment: 43 (gNMI) vs. 43.00 (CLI)

Result: Match after adjusting precision.

```
ubuntu@ubuntu:~/Desktop/proj$ ./handle_2.sh "/system/storage/state/used" "show storage usage"
Original gNMI Path: /system/storage/state/used
Transformed gNMI File Path: system/storage/state/used/gnmi_data.txt
Mapped CLI Command: show storage usage -> cli_outputs/show_storage_usage.txt
Report saved to comparison_report.txt
ubuntu@ubuntu:~/Desktop/proj$ cat comparison_report.txt
Comparison Report
--------------------
gNMI Path: /system/storage/state/used
CLI Commands: show storage usage

Comparison Results:
--------------------
Path: /system/storage/state/used
o Key: 43.00
o Result: Match after adition
```

Figure 16 : Requirement 2 ex 6

Discussion:

This screen shows the execution of a Bash script comparing the gNMI path `/system/storage/state/used` with the CLI command `show storage usage`. The script converts the gNMI path to a file-friendly format and maps the CLI command to its corresponding file. After validating the existence of the files, the script compares the key `used` between the outputs. The gNMI output is `43`, while the CLI output is `43.00`. The script does an adjustment in precision to compare these two values, so it will consider `43` from gNMI equal to `43.00` from CLI. Thus, the comparison result is a match after precision adjustment. Also, in Example 6 below, the gNMI and CLI outputs are compared although they have minor formatting differences - more decimal points are present in the CLI output. Both demonstrate the script's capability to normalize precision discrepancies, making gNMI and CLI data comparable with accuracy.