







☰  l.. / gr... 🔍 Type / to search      

<> Code ⌚ Issues 🔗 Pull requests ▶ Actions 📁 Projects 🛡 Security 📈 Insights

green-path-optimizer / README.md 



 ramobis Replace svg with png due to font issue d9e9361 · last week  History

Preview

Code

Blame

353 lines (233 loc) · 21.2 KB

Raw



green-path-optimizer @OST

Contents

- [Introduction](#)
- [Project Overview](#)
- [Project Objectives During Hackathon](#)
- [Getting Started](#)
- [Related Work](#)

Introduction

The green-path-optimizer is a framework for simulating a proof-of-concept environment. It shows how to measure the environmental impact of network paths and optimize data flow to reduce the network's carbon footprint.

Currently it contains the following components:

- **Network** based on BMv2 software switches capable of:
 - Collecting inband network telemetry (INT) data regarding energy efficiency of network paths using the [IOAM](#) protocol
 - Exporting the INT data using [IPFIX](#)
- **Configuration** utilities capable of:
 - Provisioning of an arbitrary network topology with information from a declarative YAML definition
 - Dynamic configuration update of network paths and network efficiency which

uses the [nornir](#) network automation framework

- **Monitoring** system based on the Telegraf, InfluxDB, Grafana (TIG) Stack capable of:
 - Collection of IPFIX messages ([Telegraf](#))
 - Parsing of IOAM raw export IPFIX messages ([Telegraf](#))
 - Persistent storage of efficiency data ([InfluxDB](#))
 - Visualization of efficiency data ([Grafana](#)). Two example graphs are:
 - [Heatmap](#) showing endpoint to endpoint flow efficiency
 - [Timeseries graph](#) showing the efficiency of paths over time

In near future it will also contain the following components:

- **Optimizer** to be developed at [RIPE NCC Green Tech Hackathon](#) capable of:
 - Analyzing the given time series data stored in InfluxDB
 - Proposing configuration changes to route traffic over the most efficient paths between every possible ingress and egress router
 - Trigger configuration update to actually implement the proposed optimization in the network.
- **Validator** to be developed after the hackathon and validates the optimiser's configuration changes to e.g. prevent data from being sent through bottlenecks or certain paths from being overprovisioned.

Project Overview

Over the past year, my colleagues and I (Ramon) have delved into the field of sustainable networking, focusing on the critical challenge of improving energy efficiency in computer networks. Our initial research revealed a significant gap: **optimizing network energy efficiency through traffic engineering methods is extremely difficult due to the lack of visibility into the energy efficiency of network paths.**

This realization led us to focus on enhancing visibility into the energy efficiency of network paths and nodes. After several months of research, we published a paper titled [Towards Sustainable Networking: Unveiling Energy Efficiency Through Hop and Path Efficiency Indicators in Computer Networks](#), presented at *IEEE Netsoft 2024*.

Our proof-of-concept (PoC) implementation includes several key capabilities:

- Collection of network telemetry data embedded in packet metadata using the [IOAM Aggregation Trace Option](#)
- Standardized export of the collected network telemetry data to an [IPFIX](#) collector
- Visualization of the collected data on descriptive Grafana dashboards

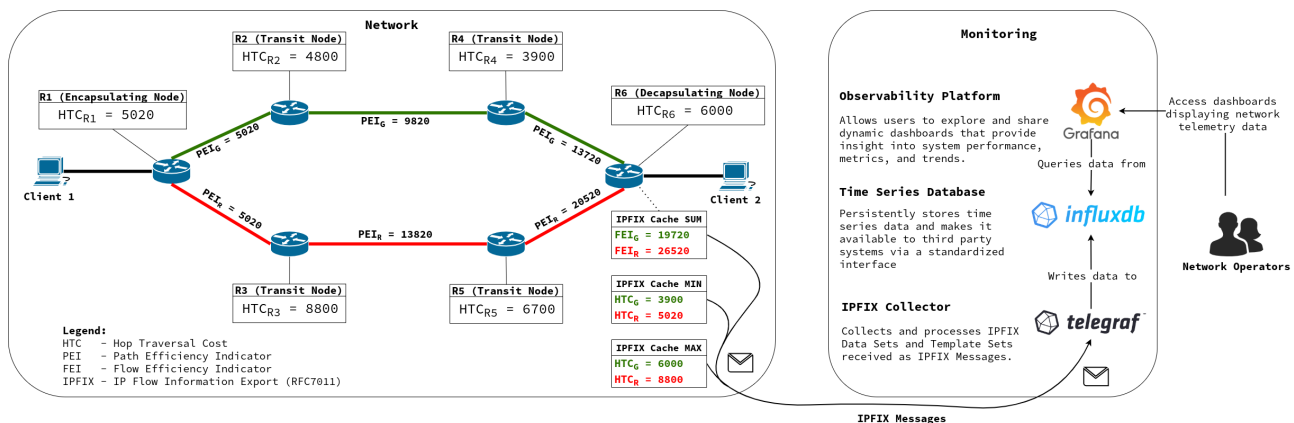
While this provides a strong foundation, our next goal is to act on the collected data by proposing actual improvements to network configuration. This includes the development of the *green-path-optimizer* application and running simulations across different topologies to verify the applicability in various use cases.

Proof-of-Concept Environment

The figure below illustrates our proof of concept in a simplified network environment.

- When packets travel through the network, an efficiency rating will be attached to it on the ingress node.
- On each subsequent node this energy rating is updated using an aggregation function such as *SUM*, *MIN* or *MAX*. Additionally each node adds himself to the node list to trace the path the packet traversed.
- As a result the packet will contain the path efficiency rating as well as the path taken inside its metadata which is then exported via IPFIX.

On the Monitoring side there is a **Telegraf, Influx, Grafana (TIG) Stack** which collects, stores and displays the collected data.



Efficiency Indication

Efficiency indication is a critical aspect of this project. The following efficiency indicators were introduced:

- **Hop Efficiency Indicator (HEI):** Is an arbitrary number indicating the efficiency of a hop. There can be several HEI values at the same time, which cover different aspects of a hop's energy efficiency.
- **Link Efficiency Indicator (LEI):** Is a dedicated value to indicate the efficiency of an interface. For example a 10Gbit/s copper interface via a twisted pair cable could be indicated to be less expensive in the means of energy efficiency compared to an interface connected to a 10Gbit/s long-haul fiber connection.

- **Hop Traversal Cost (HTC):** Is the result of accumulating the LEI of the ingress link, the HEI and the LEI of the egress link.
- **Path Efficiency Indicator (PEI):** Is the accumulation of HTC values in case the SUM aggregator is used. It indicates the efficiency of the path the packet traversed.
- **Flow Efficiency Indicator (FEI):** Is depending on the aggregator used the average PEI (SUM aggregator), the minimum HEI (MIN aggregator) or maximum HEI (MAX aggregator) considering the network telemetry data of all packets corresponding to the specific flow.

Challenges

Some of the identified challenges are:

1. Collection and processing of inband network telemetry data at line rate
2. Increased packet size as a result of inband network telemetry
3. Aggregation of collected data within the network so that the data is ready to act on and no further computation is required to obtain the path metric
4. Mapping of energy metrics to comparable values
5. Flexibility regarding environmental factors
6. Knowledge about the path a specific path metric belongs to

The challenges were addressed as follows:

1. To ensure data can be processed at line rate, the efficiency data is collected outside the data plane. It is supposed to be exposed to the data plane via a lookup table (similar to the forwarding information base (FIB) tables). Additionally, the aggregation which happens inside the data plane avoids using complex arithmetical operations.
2. The mitigation of the increasing packet size due to INT goes hand in hand with the challenge/requirement that data must be aggregated in transit to retrieve the path metric directly. The aggregation in transit leads to constant header size no matter how many nodes are traversed.
3. To aggregate the efficiency data in transit, the [IOAM Aggregation Trace Option](#) is used.
4. As shown in the figure in section [Proposed Solution](#) one or more energy metrics are mapped to a hop metric. With a consistent energy metric to hop metric configuration on all nodes, the hop metrics of the same type will be comparable.
5. The energy metric mapping as shown in the figure in section [Proposed Solution](#) can be set up as required but needs to be consistent throughout the same IOAM domain. In case energy metrics shall also be exchanged between autonomous systems (AS) with e.g. BGP in future some common configurations need to be standardized.

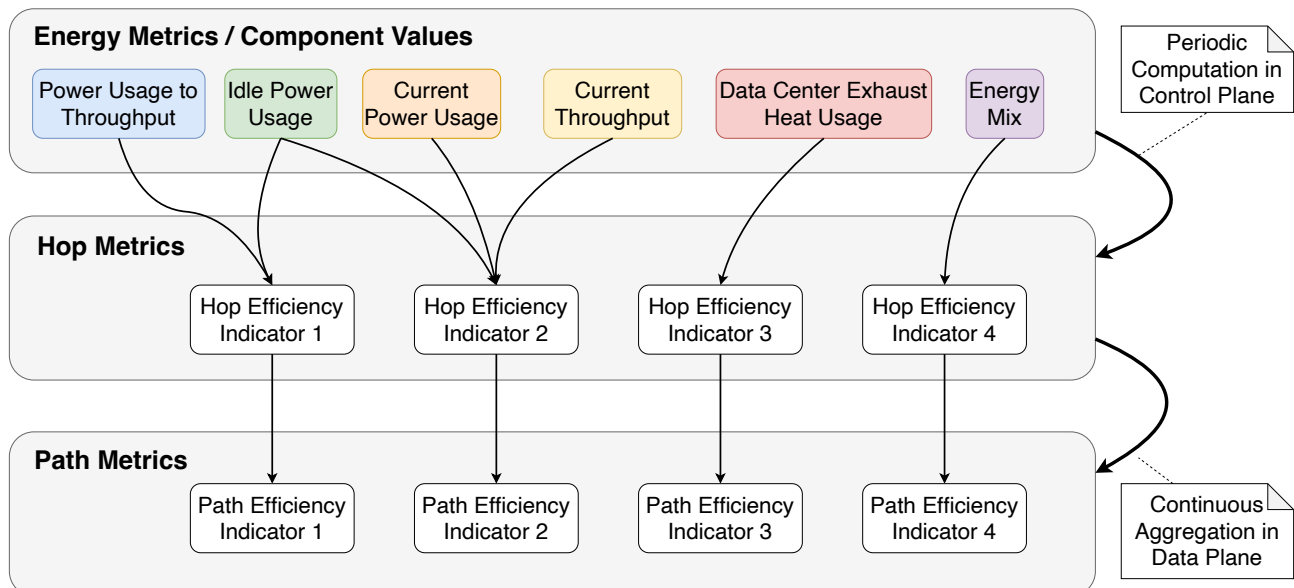
6. To determine to which path a specific path metric belongs the [Pre-allocated IOAM Trace Option](#) is used in combination with the [IOAM Aggregation Trace Option](#).

Proposed Solution

The proposed solution depicted in the figure below works as follows:

1. Map one or more energy metrics to a hop metric. This process is carried out periodically on each node outside of its data plane.
2. Store the determined hop metrics in lookup tables accessible by the data plane (similar to the FIB).
3. The data plane continuously reads these hop metrics from the lookup tables during the forwarding operation of IP packets.
 - i. If **ingress node**: Add relevant header fields and initialize with the **hop metric** retrieved from the lookup tables.
 - ii. If **transit node**: Update/aggregate header fields with the **hop metric** retrieved from the lookup tables.
 - iii. If **egress node**: Same as on the transit node. Additionally, export the relevant header fields representing the **path metric** to a collector using IPFIX.

Notice that the energy metrics and the mappings chosen in the figure below are examples only.



Network Telemetry (IOAM)

The IOAM protocol is used to store the efficiency data as inband network telemetry data. The following sections show how the telemetry data is included in an IPv6 packet.

IPv6 Hop-by-Hop Option Extension Header

An ingress node pushes a Hop-by-Hop extension header and initializes the IOAM option header fields.

The packet capture below shows an IPv6 packet carrying network telemetry data.

```

- Internet Protocol Version 6, Src: 2001:db8:64::10, Dst: 2001:db8:c8::20
  0110 .... = Version: 6
  .... 0000 0000 .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
  .... 0000 1010 1010 0110 1111 = Flow Label: 0x0aa6f
  Payload Length: 96
  Next Header: IPv6 Hop-by-Hop Option (0)
  Hop Limit: 60
  Source Address: 2001:db8:64::10
  Destination Address: 2001:db8:c8::20
  [Stream index: 1]
- IPv6 Hop-by-Hop Option
  Next Header: UDP (17)
  Length: 6
  [Length: 56 bytes]
  IOAM Option ← IOAM Pre-allocated Trace Option
  IOAM Option ← IOAM Aggregation Trace Option
  PadN
- User Datagram Protocol, Src Port: 50665, Dst Port: 443
- QUIC IETF

```

IOAM Pre-allocated Trace Option

The first IOAM Option carried inside the Hop-by-Hop Option extension header is used to trace the path the packet traversed.

Besides other fields it contains a list of nodes in the order traversed with the corresponding node id and hop limit on that specific node.

In the figure below:

- The packet traversed the nodes 1-3-4
- The node list has free space to trace one more node

Refer to [RFC9197](#) for more information about individual header fields.

```

- IOAM Option
  Type: IOAM Option (0x31)
  Length: 26
  Reserved: 0
  Option-Type: Pre-allocated Trace (0)
- Pre-allocated Trace
  Namespace ID: 10
  0000 1... = Node Length: 1
  .... .000 0... = Flags: 0x0
  .000 0001 = Remaining Length: 1
  Trace Type: 0x800000, Hop_Lim and Node ID (short)
  Reserved: 0
- Trace Data
  Free space: 00000000
- Node 1
  - Hop_Lim and Node ID (short)
    Hop Limit: 62
    ID: 0x000001
- Node 2
  - Hop_Lim and Node ID (short)
    Hop Limit: 61
    ID: 0x000003
- Node 3
  - Hop_Lim and Node ID (short)
    Hop Limit: 60
    ID: 0x000004

```

} IOAM Pre-allocated Node List

IOAM Aggregation Option

The second IOAM Option carried inside the Hop-by-Hop Option extension header is used to store the aggregated energy efficiency data of all nodes on the path.

In the figure below:

- The HEI with ID 255 is being collected
- The *SUM* aggregator is used
- The PEI is 80120

Refer to [draft-cxx-ippm-ioamaggr-02](#) for more information about individual header fields.

```
- IOAM Option
  · Type: IOAM Option (0x31)
  Length: 18
  Reserved: 0
  Option-Type: Aggregation (32)
- Aggregation
  Namespace ID: 10
  · 0000 .... = Flags: 0
  .... 0000 0000 0000 = Reserved: 0
  IOAM Data Param: 255
  · 0000 0001 = Aggregator: 1, SUM
  Aggregate: 80120
  Auxil-data Node-ID: 4
  Hop Count: 3
```

Export Mechanism (IPFIX)

Egress routers export the collected path metrics from the IPv6 packet as network telemetry data using IPFIX (as standardized in [RFC7011](#)). As already stated earlier the BMv2 software switches are used in the PoC network environment. By default the BMv2 switches are not capable of exporting data via IPFIX. A fork with the corresponding IPFIX implementation is available [here](#). The implementation can be found in the `externs` directory.

There are two different export mechanisms which fulfill different needs.

Aggregated Export

The aggregated export is used to make **flow statistics** available to a network operator as it aggregates efficiency data stored in packets on a per flow basis. For example the data is used to generate a heat map containing endpoint to endpoint network efficiency information.

Data is grouped by flow and IOAM aggregator. The network telemetry data of all packets within the same group is then aggregated using the IOAM aggregator specified and exported within a single IPFIX message.

Raw Export

The raw export is used to make **path statistics** available to a network operator. This data allows to identify inefficient paths and nodes within a network.

The Raw Export mechanism exports the entire IPv6 header including all extension headers. That means that not only the efficiency metric is going to be available but also the corresponding path information the metric belongs to.

The implementation always exports the first packet of a flow and then every nth packet based on the configured sampling rate per flow.

Dashboard

The Grafana dashboards visualize the time series data present in InfluxDB. These dashboards make it easy for humans to interpret how the network as a whole, individual paths, individual nodes, and individual flows perform in the context of the sustainability metrics. It is also helpful to see how the metrics change over time. **During the hackathon these dashboards will be very helpful to see if the optimizer application actually increases the efficiency of the network.**

Simulation Network Statistics

This dashboard contains general information about the simulation network such as:

- Overview about which IOAM aggregator has been used (MIN, MAX, SUM) for a percentage of packets
- Number of packets per flow distribution
- Number of packets per receiver time series
- IOAM Aggregation Option error statistics

The purpose of this dashboard is mainly the debugging of issues with the network as a whole.

Flow Statistics

This dashboard contains information about the efficiency of individual flows. The data displayed in this dashboard was exported via IPFIX with the **aggregated export** mechanism. Information contained is:

- A last 5min average time series which displays how the efficiency of flows change over time
- A heatmap to indicate the efficiency of flows between each host
- Some other time series and distributions about flow efficiencies

Hop Statistics

This dashboard contains information about the efficiency of individual hops. The data displayed in this dashboard was exported via IPFIX with the **raw export** mechanism and includes only data of packets which used *MIN* or *MAX* aggregation. Information contained is:

- An absolute and relative discovery about how many times a router is the most efficient or most inefficient on a specific path
- Most efficient hop discovery per path
- Most inefficient hop discovery per path

Path Statistics

This dashboard contains information about the efficiency of individual paths. The data displayed in this dashboard was exported via IPFIX with the **raw export** mechanism and includes only data of packets which used *SUM*. Information contained is:

- A last 5min average time series which displays how the efficiency of paths changes over time
- A PEI distribution
- A mapping of which path is being used between which end hosts
- A mapping of PEI value to each discovered path
- A time series which shows how the efficiency of a specific path changes over time

This dashboard could be completed with a heat map which shows the path efficiency between hops similar to the heat map in the flow statistics dashboard.

Project Objectives During Hackathon

The primary goal is to demonstrate that energy-efficient traffic routing can be achieved using our collected data, while also identifying any gaps in the current dataset that would be crucial for further optimizations.

This includes the following work items:

- Analysis of given raw data in InfluxDB
- **Development of green-path-optimizer application which suggests path updates to improve efficiency**
- **Integration of the configuration-update-utility to actually deploy the improvements in the PoC network environment**
- In case there is time:

- Parameter evaluation
- Interpretation of improvements and what is needed to apply it in practice
- Validation of the suggested improvements also regarding throughput / bottlenecks etc.
- **Conception of efficiency discovery mechanism of all paths of a given length in an arbitrary topology.**

Getting Started

Installation

Network Environment

To collaborate in the hackathon **you won't need to run the network environment on your PC**. I will bring two devices where we can run two instances of the network environment which should be enough for our purposes.

If you would like to install it, though, you will need the following software components:

- Mininet
- P4 Compiler (p4c)
- BMv2
 - You need to compile our [fork](#) from source as we had to do a little tweak in the main source code which was not contributed upstream.
- BMv2 IPFIX Extension
 - You also need to compile it from source (located in the `externs` folder in our [fork](#).
 - Do not forget to set to correct path to the shared object file in the Makefile of this repository.

I tested the setup on Ubuntu 22.04 and 22.10 where it works fine but I was unable to set it up on other versions of Ubuntu. The P4 tooling and dependencies seem to be quite picky about the underlying OS.

Monitoring Environment

To collaborate in the hackathon **you won't need to run the monitoring environment on your PC**. It will run on the same system as the network environment.

The installation would be easy though and it should run out of the box in case you have docker installed on your system.

Configuration Utilities

Both the *configuration generator* and the *configuration updater* are written in Python and are managed in a single virtual environment with the package manager [uv](#).

The installation of the dependencies is a two step procedure:

1. Verify that you have [uv](#) installed on you local system. If not follow the [Getting Started](#) guide.
2. Once [uv](#) is installed, open a terminal and change to the repository root directory and use the command `uv sync .`

Done! All dependencies are installed.

Running the Environment

To use the tools and components installed previously you will most likely be using the `make` command to execute the predefined targets.

Network Environment

As already mentioned I will provide two systems which operate the network environment. So you may skip this section.

To run the network there are two different `make` targets available:

- `run`: Will spool up the network environment and the monitoring system after building the project and generating the configurations.
- `run-debug`: Will do the same as `run` but additionally the BMv2 switches will write verbose log files and packet capture files. This mode should only be used in case you need the detailed information provided, otherwise it will fill your disk sooner or later.

To run only the network environment, you may use the following targets.

- `run-network`
- `run-network-debug`

Monitoring Environment

As I will provide two systems which operate the monitoring environment, you may skip this section.

If the network environment was already started with the `make run` command the monitoring environment is already started.

Otherwise the following `make` target can be used:

- `run-monitoring`: This will bootstrap the TIG stack based on the docker compose file.

Configuration Utilities

To run the configuration utilities, the following `make` targets are available:

- `config`: Executes the `generate-config` and then the `update-config` target.
- `generate-config`: Regenerates all configuration files based on the resource file specified in the Makefile. This task is automatically executed as part of the `run` target.
- `update-config`: Pushes the generated configurations to the BMv2 software switches. Make sure to select the correct inventory in the `config.yaml` file inside the config updater.

Stopping the Environment

To stop the environment there are dedicated `stop` `make` targets:

- `stop`: Stops the network and the monitoring environment
- `stop-network`: Stops the network environment
- `stop-monitoring`: Stops the monitoring environment

Cleaning the Environment

To clean the environment there are dedicated `clean` `make` targets:

- `clean`: Removes all files generated by any `make` target
- `clean-network`: Stops the network environment and removes all files related to the network environment such as log files and packet captures
- `clean-monitoring`: Stops the monitoring environment and removes all files and docker volumes related to the monitoring environment
- `clean-config`: Removes the generated configuration files and log files

Related Work

- **IEEE Publication:** [Towards Sustainable Networking: Unveiling Energy Efficiency Through Hop and Path Efficiency Indicators in Computer Networks](#)
- **Internet - Draft:** [Aggregation Trace Option for In-situ Operations, Administration, and](#)

[Maintenance IOAM](#)

- **Internet - Draft:** [Challenges and Opportunities in Management for Green Networking](#)
- **RFC - Proposed Standard:** [Data Fields for In Situ Operations, Administration, and Maintenance \(IOAM\)](#)