OST
Eastern Switzerland
University of Applied Sciences

Green Networking

Visibility, a first step towards sustainable networking

INS Institute for Network and Security

Semester Thesis HS2023

Advisor:	Prof. Laurent Metzger
Co-Advisor:	Severin Dellsperger
Partner:	Alexander Clemm

Authors: Ramon Bister
Reto Furrer

FUTUREWEI Technologies

Submission: 22. December 2023
Version: 1.0

Revision History

Revision	Date	Author(s)	Description
1.0	22.12.2023	Ramon Bister / Reto Furrer	Release of the initial version of the document for submission

1. Abstract

As the global demand for network connectivity intensifies, the environmental impact of networking infrastructure becomes increasingly significant. This study addresses the critical issue of the lack of visibility into the energy efficiency of networks, hindering the optimization for sustainability. Drawing from the Green Networking Metrics IETF draft, we propose a method to bring transparency to energy efficiency within networks, laying the groundwork for green routing strategies.

The research employs a two-fold approach. First, it involves a theoretical elaboration of mathematical concepts, establishing a foundation for understanding and evaluating energy efficiency in networking environments. Second, a Proof of Concept is implemented on a virtualized network, demonstrating the feasibility of the proposed methodology.

The results showcase the potential for comparing path efficiency and identifying the most efficient and inefficient hops along a network path. The findings underline the necessity for further exploration, advocating for the implementation of the methodology on a physical network. Subsequently, efficiency data should be exported and analyzed centrally, paving the way for informed decision-making towards sustainable networking practices. This study contributes to the emerging field of green networking by providing a structured approach to enhance energy efficiency visibility in network environments, fostering a more sustainable and ecologically responsible networking infrastructure.

2. Management Summary

In the current landscape of computer networks, the visibility into their energy efficiency is notably limited, posing a significant challenge to the pursuit of sustainable networking practices. Recognizing this inherent problem, this research project has undertaken the mission to contribute to a solution to finally overcome these limitations.

At the core of this study is the development of meaningful efficiency indicators at both the hop and path levels, addressing the prevailing dearth of insights into the energy performance of individual nodes and network paths. The introduction of the Hop Efficiency Indicator (HEI) and its calculation algorithm, designed for efficiency and flexibility, lays the groundwork for a comprehensive understanding of node-level efficiency. The study goes further by proposing the Path Efficiency Indicator (PEI) to summarize and compare hop efficiency indicators along different network paths. The following figure uses a simple example to illustrate how HEI and PEI are used in an energy efficiency enabled network to classify paths based on their energy efficiency. In the following example, the PEI assigned to the green path is lower than the PEI assigned to the red path. Considering that a lower PEI indicates better energy efficiency, the green path is more energy efficient than the red path.

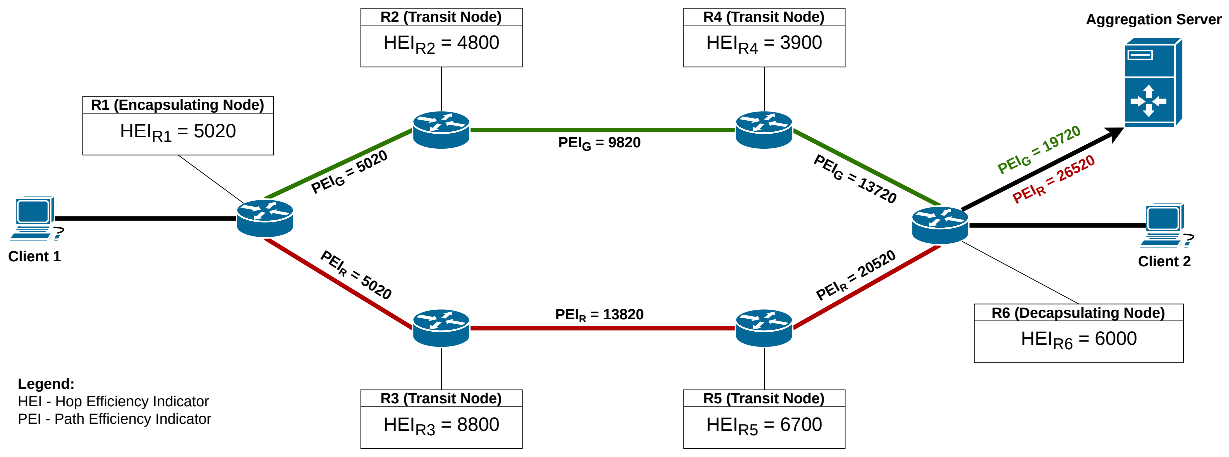


Figure 2.1.: Indicator Calculation Visualization

This research doesn't merely stop at theoretical propositions; it translates these indicators into practical implementation by incorporating them into user packets through the In-situ Operations, Administration, and Maintenance (IOAM) protocol. By doing so, it provides a tangible solution to the challenge of transporting efficiency metrics within network packets.

The implications of this study are profound for the quest towards sustainable networking practices. By embedding efficiency indicators into data forwarding and user packets, the research enables real-time decision-making for optimizing network energy efficiency. This approach facilitates dynamic scaling of the network, routing traffic through the most efficient paths, and selectively activating or deactivating routes based on actual needs.

Recognizing the need for continued progress, the study outlines recommendations for future research. It suggests exploring the export of network energy efficiency data using IPFIX into a time series database, enabling standardized data collection for comprehensive analysis. Additionally, the proposal includes the development of a central monitoring platform and a central management platform. These platforms would not only consolidate energy efficiency metrics but also empower administrators to configure and customize parameters, contributing to the practical implementation of energy efficiency indicators in diverse network environments.

In essence, this research project stands as a pivotal response to the critical issue of limited visibility into the energy efficiency of networks. Through innovative indicators, protocols, and practical implementations, it opens new avenues for sustainable networking practices and lays the foundation for future advancements in the field.

3. Acknowledgement

We would like to express our sincere gratitude to the following individuals and organizations who have supported us throughout the process of completing this semester thesis.

First and foremost, we are deeply thankful to our thesis advisor, **Professor Laurent Metzger**, for his support during this thesis. His commitment, passion and expertise helped and motivated us a lot.

Thanks to **Severin Dellsperger**, for his support during this thesis. We greatly appreciated the constructive discussions, the respectful interaction, his expertise and commitment to our semester thesis.

A special thanks goes to our external partner, **Alexander Clemm** working at Futurewei Technologies, Inc., for his invaluable guidance, encouragement, and expertise. His insightful feedback and constructive criticism played a pivotal role in shaping the direction of our research. We also greatly appreciated the fact that we were involved in the standardisation work right from the start and were given the opportunity to work on the implementation of the IOAM Aggregation Option protocol extension in our simulated network environment. We were very pleased with the uncomplicated nature of the collaboration and his personal visit to us in Rapperswil.

We owe a profound debt of gratitude to **Professor Roman Fuchs** for his exceptional support and expertise in the realm of mathematics. He played a pivotal role during the development and refinement of the energy efficiency indicator mathematical concept.

In conclusion, we are thankful to all those who played a role, big or small, in helping us bring this thesis to fruition. Your support has been invaluable, and we are truly appreciative.

Ramon Bister, Reto Furrer
OST - Eastern Switzerland University of Applied Sciences
22. December 2023

4. Important Terms and Abbreviations

Hop Efficiency Indicator (HEI) Each hop on a network path calculates a value composed of an arbitrary number of raw components. These components include but are not limited to idle power, energy mix, embedded carbon and link cost. All these raw component values are normalized and aggregated by sum operation to the HEI. A hop with a low HEI value is considered more energy efficient.

Path Efficiency Indicator (PEI) The path efficiency indicator is an aggregation by sum operation with all HEI values on a path. This value indicates the efficiency of the path the packet traversed.

IOAM In situ Operations, Administration, and Maintenance (IOAM) collects operational and telemetry information in the packet while the packet traverses a path between two points in the network. [1]

P4 Programming Protocol-independent Packet Processors (P4) is a domain-specific language for network devices, specifying how data plane devices (switches, NICs, routers, filters, etc.) process packets. [8]

5. Introduction

5.1. Background

Climate change and the need to curb greenhouse emissions have been recognized by the United Nations and by most governments as one of the biggest and most urgent challenges of our time. As a result, reducing carbon footprint is becoming of increasing importance for society and all industries. The networking industry is no exception. Future networking advances will thus increasingly need to focus on becoming more sustainable and reducing carbon footprint, both for economic reasons and for reasons of corporate responsibility. This shift has already begun as sustainability is becoming an important concern for network providers. Opportunities exist not only at the level of making networking hardware and transmission technology more energy-efficient, but also at other levels. Examples include inclusion of carbon intensity as a cost factor in routing or optimization of placement of virtual networking functions. Regardless of particular measures being taken, there is a need to have visibility into green networking metrics. Having such visibility allows to assess the effectiveness of measures being taken. In many cases, it also provides the bases for control loops that optimize those metrics as well as for other decisions taken to minimize carbon footprint. Many of the more "obvious" green network metrics are assessed at the level of individual devices, for example measuring their power consumption and that of their port. An aspect that has been much less explored concerns green metrics that allow to assess the carbon footprints of paths, as well as ways in which a network could be instrumented to provide such metrics. This in turn would enable a multitude of new optimization opportunities, for example optimizing paths so that their overall footprint is minimized, then steering traffic across the most carbon-efficient paths. The ability to collect network telemetry across networking paths has received considerable attention in recent years. Specifically, IETF has been standardizing in-situ OAM (iOAM), a set of protocol extensions designed to collect certain telemetry data from nodes across networking paths. Likewise, the P4 consortium has been promoting the collection of In-Network Telemetry (INT) as one of P4's use cases. However, neither approach has been applied to green networking metrics, such as relating to carbon efficiency or power consumption across a path. Likewise, neither approach allow to aggregate metrics that are collected, relying instead of outside applications.

Alexander Clemm, Futurewei Technologies Inc.
27. September 2023

5.2. Thesis Composition

The documentation of the semester thesis is structured in four main parts based on the RUP (Rational Unified Process) project method. The goal of this documentation is to give you some insights about the most important concepts and challenges in this project. To be able to follow some basic knowledge in networking and mathematics is required. The terms Green Networking and Sustainable Networking are used interchangeably in this project.

5.2.1. Inception

The first part outlines the initial situation, our vision and the overall project planning with the defined use cases, requirements and risk assessment.

5.2.2. Elaboration

Our concept for calculating the efficiency indicators is described in the elaboration part. Furthermore data plane programming basics including a focus on the P4 programming language as well as our development environment and testing approach are described in this part.

5.2.3. Construction

The construction part contains the most important facts about the implementation of the P4 efficiency indicators functions, the IOAM Pre-allocated Trace-Option and the IOAM Aggregation Option. Additionally, the implementation of the testing environment and our demo application are covered in this part.

5.2.4. Transition

The transition part contains a summary of the achieved work and outlines the next and upcoming work steps.

5.2.5. Appendix

This part lists the meeting notes, the poster about the semester thesis and contains the IETF Draft that emerged from the semester thesis.

Contents

Part I.

Inception

1. Initial Situation

As already described in the introduction, the networking industry will increasingly need to focus on becoming more sustainable and reducing their carbon footprint. The goal of this thesis is to evaluate a carbon metric with several parameters to calculate a path efficiency indicator per packet and make the traversed paths comparable. With our external partner, Alexander Clemm, we have a good connection to the IETF - Internet Engineering Task Force. With him we can work future oriented and use modern solutions like the In-situ OAM (IOAM), a protocol extension designed for collecting network telemetry from nodes across the network path.

1.1. Existing Research

The project is based on the previous work of the IETF Draft Green Networking Metric [3] and the RFC 9197. Alexander Clemm is an author of the previous mentioned IETF Draft about the Green Networking Metric and the tasks of this semester thesis were defined with him. At the beginning of this project, Alexander Clemm published a new IETF Draft named Aggregation Trace Option for In-situ Operations, Administration, and Maintenance (IOAM) [2]. The protocol extension defined in this draft is the basis for the elaboration of a concept on how to process energy efficiency information in a computer network.

The project is mainly related to the RFC documents and drafts below.

Green Networking Metrics (Draft) Defines the underlying research field [3]

IOAM Aggregation Trace Option Type (Draft) Defines the method used to store the PEI or min/max HEI values in the user packet header data [2]

IOAM Data Fields Defines the IOAM Trace Option type which is used to trace the path a packet traverses RFC 9197

Internet Protocol, Version 6 (IPv6) Defines the Hop by Hop Extension header which is used to encapsulate the IOAM data RFC 2460

2. Vision

Gaining visibility about the energy efficiency and the carbon footprint of networks is only the first step towards sustainable networking. Based on the information gathered with the HEI and PEI, the gained visibility should enable the proposal of concrete measures to improve the overall network energy efficiency.

2.1. Goals

The goal of this work is to define a metric that is used on each network node to determine the hop efficiency indicator. To get an overview of the path, it should be possible to sum up all hop efficiency indicators on a traversed path to obtain the path efficiency indicator. With this path efficiency indicator it is possible to compare different paths and make further decisions based on the efficiency of a path or a node.

In the next sections two possibilities are described what can be done in future based on this work.

2.1.1. Automated Network Efficiency Analysis

A central system should be able to reconfigure the HEI/PEI calculation at runtime. For example the most efficient and most inefficient routers could be discovered on a path using the MIN/MAX aggregator. Another discovery would be to determine the impact of individual HEI components by dynamically adjusting the HEI composition. This would include the addition and removal of components to the HEI on demand which would mitigate the side effect of the loss of information happening due to the aggregation by the sum operation from individual component values to the HEI. All the results should then be displayed on a central dashboard.

2.1.2. Sustainable Networking

Sustainable networking will be the final goal. It does not only include the automated analysis but also the automatic application of improvements to the network in order to increase its efficiency. This could be achieved by:

Traffic Rerouting In case there are multiple paths available between an ingress- and egress router, traffic could be routed via the more efficient path.

Network Scaling In case there are multiple paths available between an ingress- and egress router, the less efficient paths could be disabled taking the current network load into account.

Identification of Network Topology Improvement Potential Based on the information gathered using the green metrics, physical or logical topology changes could be proposed to network administrators to further improve the network efficiency.

3. Use Cases

This chapter covers the use cases for the semester thesis and at the same time sets the framework for our work. The possibilities and further scenarios are discussed in the chapter Outlook.

3.1. UC01: Calculate Hop Efficiency Indicator (HEI)

On each node in the network the hop efficiency indicator (HEI) should be calculated. The value of the hop efficiency indicator states, as the name suggests, how efficient the respective node is. The hop efficiency indicator is composed of different parameters and each of them has a weighting, further referred to as components (parameter with weight). A specific but editable mathematical formula is determined to calculate the HEI from the different components. The addition and removal of components to the HEI should be possible in future. There should be a possibility to control the impact of an individual component to the HEI. The calculation of the HEI should be as efficient as possible (avoid division).

3.2. UC02: Aggregate HEIs to Path Efficiency Indicator (PEI)

In order to compare the different path efficiencies, a representative value is required that can be compared. For this use case, a path efficiency indicator (PEI) is used to summarize the hop efficiency indicators (HEI) of the nodes. The calculation of the PEI should be as efficient as possible (avoid division). In addition, the calculation may include addition, incrementation, and comparison (min/max).

3.3. UC03: Append the PEI to the packet carrying user data

To append the path efficiency indicator value the IOAM protocol shall be used. The implementation should focus on adding the header information to IPv6 packets using the hop by hop extension header.

3.4. UC04: Determine the min/max HEI in a path

Instead of aggregating the hop efficiency indicators (HEIs) to a path efficiency indicator (PEI) the minimum or maximum hop efficiency indicator value and the corresponding node ID shall be recorded. The IOAM header should be used with the intended aggregator.

3.5. UC05: Assign the PEI to the traversed path

The path efficiency indicator (PEI) is only effective if the value can be assigned to a specific path. The IOAM Aggregation Type Option currently does not support the assignment of the aggregated value to a specific path the packet traversed. This leads to the fact that either the IOAM Trace Option Type will be used in conjunction with the IOAM Aggregation Option Type. As using the IOAM Trace Option Type is an option it is not a strict requirement that the added header information must be of a fixed size. Preferred though would be a solution with a fixed header size. This use case is intended to be optional. For demo purposes in our development network we would be allowed to identify the taken path according the source and destination address of the packet.

3.6. UC06: Collect the PEI in demo application

In the demo application, the traveled path and the path efficiency indicator (PEI) should be extracted and displayed for each packet. The path efficiency indicators (PEI) written into the user packets shall be extracted on an endpoint. The efficiency indicators must be mapped to the specific path the packet traversed. A rudimentary output on the CLI containing the following information shall be given (list may not be complete).

For each path in the dev network:

- Path Identification
- Timestamp of last update
- Hops traversed
- Number of packets received
- Amount of data received
- Latest PEI value
- Average PEI value

The output shall be updated in a short interval (e.g. 1s). For demo purposes the values of specific hop efficiency components shall be adjusted during runtime to see an impact.

4. Requirements

This chapter contains the functional and non-functional requirements of the project. The requirements are specified based on the *FURPS* classification method. *FURPS* is an acronym and stands for:

Functionality Capability, Reusability, Security

Usability Human Factors, Aesthetics, Consistency

Reliability Availability, Recoverability, Accuracy

Performance Speed, Efficiency, Resource Consumption, Scalability

Supportability Maintainability, Testability, Flexibility

The information above is based on the *FURPS* Wikipedia entry [4].

4.1. Functional Requirements

The functional requirements below are all related to the *F* in *FURPS*.

ID	FR1
Subject	Calculate the Path Efficiency Indicator (PEI) for each packet
Summary	Each packet traversing the network shall be labeled with an energy rating of the traversed path.
Justification	In order to optimize the energy efficiency of networks in future an insight into how good a specific path of a network performs in regards to the energy efficiency is needed.

ID	FR2
Subject	Calculate the Hop Efficiency Indicator (HEI) for each packet
Summary	Each node the packet traverses the node calculates its own efficiency rating based on the given parameters from the control plane.
Justification	To get the efficiency rating of a path each individual node must participate in the indicator calculation.

ID	FR3
Subject	Add the PEI to each packet as metadata
Summary	The PEI shall be carried by the corresponding packet as part of the header metadata.
Justification	Each node on the way needs to aggregate the current PEI with its own PEI. In order to be able to do this the current PEI value must be present.

ID	FR4
Subject	Determine the node with the highest HEI
Summary	Instead of aggregating the individual HEI values to the PEI, the highest HEI value calculated during the path traversal shall be added to the packet as header metadata.
Justification	The nodes which have a high/bad impact on the PEI should be discoverable.

ID	FR5
Subject	Determine the nodes with the lowest HEI
Summary	Instead of aggregating the individual HEI values to the PEI, the lowest HEI value calculated during the path traversal shall be added to the packet as header metadata.
Justification	The nodes which have a low/good impact on the PEI should be discoverable.

ID	FR6
Subject	Determine the path a packet traversed
Summary	The PEI value is path specific and must be assigned to the path the packet traversed.
Justification	Without the knowledge about the path a packet traversed the PEI value is useless. The primary usage of the PEI value is the comparison of paths in regards to their energy efficiency.

ID	FR7
Subject	HEI parameter weighting
Summary	A network administrator shall be able to individually select the impact of specific parameters to the HEI value.
Justification	Some parameters might be more important than others and this shall be represented in the HEI and finally the PEI value.

ID	FR8
Subject	HEI parameter inversion
Summary	If parameters have a good (energy efficient) value it should have a low impact (small increase) of the HEI. Parameter values which are good when they have a high value need to be inversed in order to have a low impact on the HEI.
Justification	Some parameters are better when they have a low value and other parameters might be better if the have a high value. In order to have a consistent mapping to the HEI impact the parameters which are good when their value is high, an inversion of the parameter value is necessary.

4.2. Non Functional Requirements

Based on the FURPS classification the non functional requirements are categorized in the following classes:

- Usability
- Reliability
- Performance

- Supportability

ID	NFR1
Subject	Performant calculation of the HEI and PEI
Requirement	Performance
Priority	High
Summary	The calculation of the HEI is performed on any node for every packet traversing the network on its path. This implies that the calculation is performed unimaginably often. Therefore the calculation operations should only include integer arithmetic with a focus on bit shifting and other logical operations. Floating point division is not allowed.
Justification	The calculation is done very often and it should have negligible impact on network performance. Floating point arithmetic is very costly and additionally it is not supported on FPGA hardware.

ID	NFR2
Subject	All components mapped to the HEI must be comparable
Requirement	Supportability
Priority	High
Summary	The HEI value is based on different components. Each of the components has its own specific value range. In order to compare the component values they need to be mapped on a common value range using a normalization technique.
Justification	Each component has its own value range. To simplify the calculation of the HEI and to ensure that the values of the components have the same effect on the HEI.

ID	NFR3
Subject	Removal and addition of parameters to the HEI calculation process
Requirement	Supportability
Priority	High
Summary	It should be possible to add and remove parameters to the dataplane program without the need to adjust datastructures and existing actions.
Justification	This project is an initial attempt on how to do the calculations and how to store the calculated data inside the header of the packet. The selection of all the relevant components is not the main focus of this project and the availability of components on different hardware might vary. A future proof solution must be easily adjustable in regards to which parameters are taken into consideration.

ID	NFR4
Subject	Forwarding of network traffic is not impaired by the indicator processing
Requirement	Reliability
Priority	High
Summary	Whether the efficiency indicator processing is enabled or disabled in a network should not affect the actual data forwarding. Packets which can not carry additional metadata in the header because they already reached there maximum size shall be forwarded without the efficiency indicator processing.
Justification	Determining the efficiency of a network is not the core responsibility of a network. The reliability of a network is one of the most important requirements. This should not be undermined with the determination of the efficiency of the network.

ID	NFR5
Subject	The overhead of the energy indicator shall be constant over an arbitrary number of nodes.
Requirement	Performance
Priority	Medium
Summary	Metadata related to the energy efficiency indicator shall only be added on the ingress node. Transit nodes are not allowed to add new data. Data may only be aggregated in transit. The header storing the indicator data is of fixed size. Data related to path tracing is not considered to be part of the energy efficiency indicator metadata.
Justification	A packet is not allowed to grow beyond the maximum transmission unit (MTU). Adding metadata to a packet on every single node has the disadvantage that it is likely that the MTU will be reached and no more data can be added. This would result in an incomplete calculation of the PEI.

Part II.

Elaboration

1. Design Decisions

The following design decisions are documented as so called Y-Statements. The statements will refer to the functional- and non functional requirements specified in chapter 4 in the inception part.

- In the context of the energy metric calculation process (FR1-FR8), facing the need to not impact network performance (NFR1), we decided to use bit shifting as an alternative to the division operation, and neglected the usage of floating point arithmetic's, to achieve the maximum performance during packet processing, accepting that numbers not belonging to the basis two can not be computed without a rounding error.

- In the context of the energy metric calculation process (FR1-FR8), facing the need of the comparability of HEI values (NFR2), we decided that all nodes in a network process the same set of components and neglected to implement functionality to indicate which components where used on which node, to achieve comparability of the HEI values with no additional processing overhead, accepting that only the subset of components which are supported on all nodes can be used in a network.

- In the context of the energy metric calculation process (FR1-FR8), facing the need to remove and add components on demand (NFR3), we decided to define a calculation method which can be applied to any parameter and neglected to implement specific calculation methods for each specific parameter, to achieve a maximum flexibility with the addition and removal of parameters because almost no code changes are necessary, accepting that individual characteristics of parameters cannot be taken into account.

- In the context of the energy metric calculation process (FR1-FR8), facing the need to not impact the forwarding process of network traffic (NFR4), we decided to handle all efficiency indicator related processing in the egress pipeline, and neglected the implementation of all functionality inside the ingress pipeline, to achieve a logical division of functionality and to avoid interference with the forwarding operations carried out in the ingress pipeline, accepting that the application can only be run on a target which supports at least two pipelines.

- In the context of the energy metric calculation process (FR1-FR8), facing the need of a constant energy efficiency header size (NFR5), we decided to use the IOAM aggregation option to store efficiency indicator related data, and neglected the usage of the IOAM trace option, to achieve a constant size of efficiency indicator metadata over an arbitrary number of nodes, accepting that information of individual hops is lost due to aggregation.

2. Efficiency Indicators

This chapter describes the Hop and Path Efficiency Indicator and how the calculation of the HEI and PEI works in this project. As transport method for the PEI value the IOAM Aggregation Option Type is used as described in chapter 5 in the construction part.

2.1. Gain

Calculating the HEI and PEI values provides visibility into a network. The gained transparency enables the identification of the most inefficient node on a specific path or the optimization of the packet transfer based on the HEI and PEI.

2.2. Hop Efficiency Indicator - HEI

The efficiency of each node is measured by the HEI, which is composed of various weighted components. A mathematical formula that is flexible is used to calculate the components, allowing for future adjustments. Another benefit is the immediate feedback on changes in the network.

2.2.1. HEI Components

The hop efficiency indicator (HEI) is calculated using various components.

Examples for HEI components are:

Energy Mix Proportion of the electricity generated from renewable energy sources

Power Idle Power consumption when the device is powered on and not forwarding any data

Power to Bandwidth Ratio Coefficient how the device performs with increasing bandwidth

Link Cost Efficiency of the link the packet was received or sent from

i Information

The definition of a practical composition of HEI components is out of scope for this project. The aim is to do a prove of concept with a small subset of possible components and build the solution flexible enough for the future expansion of HEI components.

2.3. Path Efficiency Indicator - PEI

The path efficiency indicator is an aggregation by sum operation with all HEI values on a path. This value indicates the efficiency of the path the packet has traversed. Using this information about the different paths, it is possible to monitor the network and make active changes based on the efficiency of the different nodes.

2.4. Minimum / Maximum HEI on Traversed Path

The IETF Draft about the IOAM Aggregation Option Type [2] specifies a min and a max aggregator which obtain the node with the minimum or maximum aggregate value on a traversed path. In this project and for this use case the aggregate value in the IOAM Aggregation Option Type header represents the minimum or maximum HEI value on a specific path. In this case, not every node on the traversed path will add its HEI value to the aggregate value field in the header. Instead, each node checks whether the current value of the HEI is smaller or larger than the previous aggregate value. If the condition is satisfied, the HEI value and node ID of the current node are added to the IOAM Aggregation Option Type header to capture the node with the smallest or largest value. Additionally the current node ID and hop limit are added to the IOAM Pre-Allocated Trace-Option Type header to ensure the traceability of the traversed path. This use case for determining the worst or most efficient node on a traversed path relates to the functional requirement (FR4, FR5) specified in section 4.1 in the inception part.

2.5. Average of HEIs on Traversed Path

The IETF Draft about the IOAM Aggregation Option Type [2] specifies an aggregator to perform the average of the aggregate values on a traversed path. In this thesis, the average aggregator is not used and is not described in detail.

2.6. Mathematical Concept

This section explains the mathematical concept used to calculate the HEI and the PEI. In order to follow this, some basic mathematical knowledge is required, in particular regarding normalization and bit shifting.

i Information

The mathematical concept is based on the following calculation steps. No simulation, such as the Monte Carlo method, was performed as part of this thesis to analyze how the calculation procedure behaves over a wide range of values. The validation of the calculation method is to be endeavoured in further research on this topic.

The following requirements were defined that the mathematical concept should cover:

Variable input size The input size of the parameters used in the calculation of the HEI value should be variable.

256 components It should be possible to include up to 256 weighted component values in the calculation of the HEI value.

256 HEI values It should be possible to include up to 256 HEI values in the calculation of the PEI value.

Performant calculation of the HEI and PEI The calculation is done very often and it should have negligible impact on network performance

On the following page is a graphical overview that illustrates the mathematical concept of the HEI calculation which is added to the PEI and stored in the Aggregate field of the IOAM

Aggregation Option Type header. In the sections following the overview, there is a step-by-step calculation example that shows the individual steps for determining the HEI value.

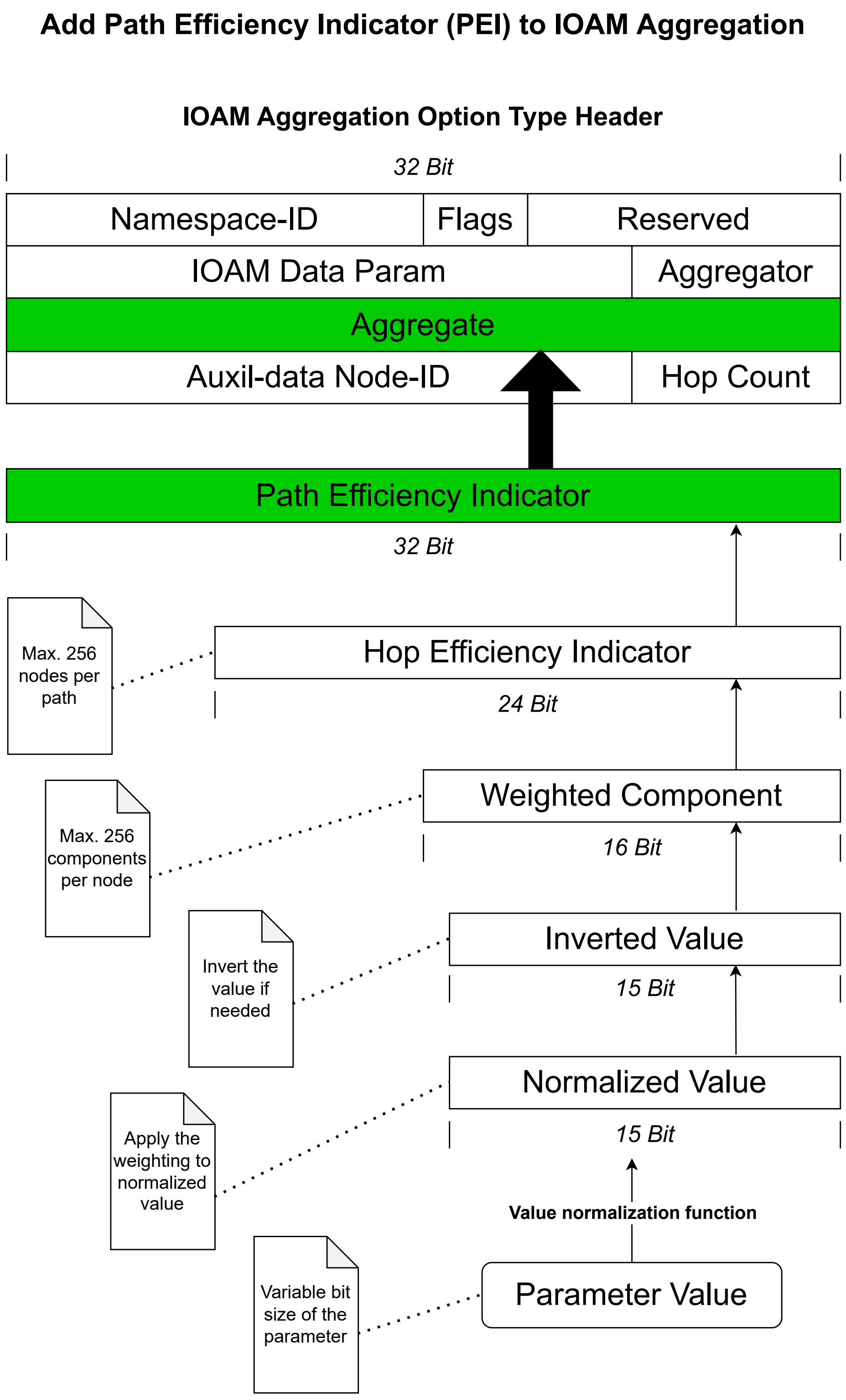

Add Path Efficiency Indicator (PEI) to IOAM Aggregation
IOAM Aggregation Option Type Header
32 Bit
Namespace-ID
Flags
Reserved
IOAM Data Param
Aggregator
Aggregate
Auxil-data Node-ID
Hop Count
Path Efficiency Indicator
32 Bit
Max. 256 nodes per path
Hop Efficiency Indicator
24 Bit
Max. 256 components per node
Weighted Component
16 Bit
Invert the value if needed
Inverted Value
15 Bit
Apply the weighting to normalized value
Normalized Value
15 Bit
Value normalization function
Variable bit size of the parameter
Parameter Value

Figure 2.1.: Exploded view drawing of the PEI composition

2.6.1. Get the Parameters

The parameters used to calculate the HEI value can be obtained from the control plane, set and configured via gRPC, or obtained directly from the hardware via extern functions. In the P4 language, an extern function is a function that is defined outside the P4 program, but can be called from inside the program. For example, this could be a function on specific hardware that returns the current idle power of the device. The method that is used in this project to obtain the parameters is described in the construction part.

2.6.2. Normalization

The obtained parameter values are within an arbitrary value range and must be mapped to a predefined/normalized range in order to be comparable. This process is called normalization and is a prerequisite for the subsequent calculations to work.

$$C_n = f(v, x, y) = v * \frac{2^x}{2^y}$$

Cn Normalized component value

v Parameter value that is used to calculate the HEI

x Number of bits in the normalized range

y Number of bits of the input range (value bit size)

2.6.3. Inversion

By definition a lower PEI value is considered more energy efficient. As the HEI component values directly influence the PEI value, the higher the component value, the higher the increment to the PEI value by that component. Some of the component values behave inversely to this definition. For example, Energy Mix indicates the percentage of electricity that comes from renewable sources. A higher energy mix value is better. Therefore, the energy mix value needs to be inverted within the normalized range.

$$C_i = g(C_n, x, i) = \begin{cases} (2^x - 1) - C_n, & \text{if } i = \mathit{true} \\ C_n, & \text{otherwise} \end{cases}$$

Ci Inverted component value

Cn Normalized component value

x Number of bits in the normalized range

i Boolean indicating if the component is inverse

2.6.4. Weighting

Finally, weights are applied to the normalized and potentially inverted values.

- A positive weight (*weight=2*) indicates that the component is more relevant than others and doubles the calculated value which increases its effect on the HEI and PEI.
- A neutral weight (*weight=1*) does not change the value at all.

- A negative weight (*weight=0*) indicates that the component is less relevant than others and halves the calculated value which decreases its effect on the HEI and PEI.

$$C_w = h(C_i, w) = \begin{cases} C_i \cdot 2, & \text{if } w = 2 \\ C_i, & \text{if } w = 1 \\ \frac{C_i}{2}, & \text{if } w = 0 \end{cases}$$

Cw Weighted, inverted and normalized component value

Ci Inverted and normalized component value

w Weight to be applied

2.6.5. HEI Calculation

The figure contains an overview of the calculation steps for determining the HEI value. The sum of the various weighted components represents the HEI of a particular node.

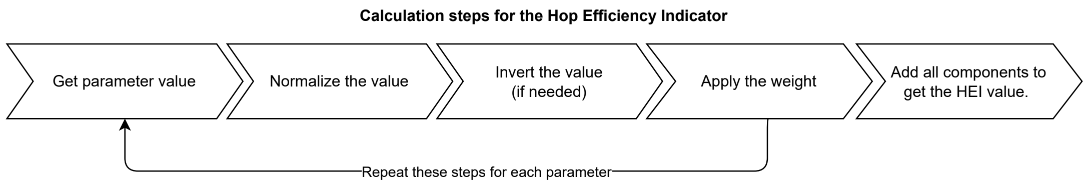

Calculation steps for the Hop Efficiency Indicator
Get parameter value
Normalize the value
Invert the value (if needed)
Apply the weight
Add all components to get the HEI value.
Repeat these steps for each parameter

Figure 2.2.: HEI calculation steps

The following formula defines the Hop Efficiency Indicator.

$$\texttt{HEI} = \left\{ \sum_{i=0}^{n} Comp_i \;\middle|\; n < 256 \right\}$$

2.6.6. PEI calculation

The path efficiency indicator is the sum of the HEI values of all nodes in a specific path.

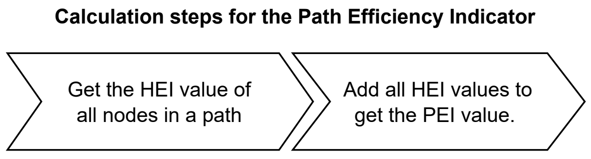

Calculation steps for the Path Efficiency Indicator
Get the HEI value of all nodes in a path
Add all HEI values to get the PEI value.

Figure 2.3.: PEI calculation steps

The following formula defines the Path Efficiency Indicator.

$$\texttt{PEI} = \left\{ \sum_{i=0}^{n} HEI_i \;\middle|\; n < 256 \right\}$$

2.7. Calculation Example

The hop efficiency indicator calculation is shown in the following step-by-step instructions. In step 2, the value is normalized to a predefined range of 15 bits. The constant that defines the range of the normalized value is: VALUE_NORMALIZED_BIT_SIZE = 15. That means the maximum parameter value in this project can be 32767.

Step 1 Get the parameters that are used to calculate the hop efficiency indicator.
Energy mix: value = 34, type size = 7 bit, weight = 2, invert = 1

Step 2 The type of a parameter is in a variable range, so the value must be normalized to be comparable to other parameters. The following python normalize function shows how the normalization is done. The function gets the parameter value and the size of the parameter type and will illustrate the normalized value.
normalize(34, 7) = 8704

```python
def normalize(value, type_size):
    if type_size < VALUE_NORMALIZED_BIT_SIZE:
        return value << (VALUE_NORMALIZED_BIT_SIZE - type_size)
    elif type_size > VALUE_NORMALIZED_BIT_SIZE:
        return value >> (type_size - VALUE_NORMALIZED_BIT_SIZE)
```

Step 3 Parameter values which are considered more efficient if the value is higher, need to be inverted due to the definition of the HEI. For example the parameter energy mix, that shows how much of the used energy is from renewable energy sources. In that case a value of 100% is better than 1% so the value must be inverted to represent the impact to our calculation correctly.
invert(8704, 1) = 24063

```python
def invert(value_normalized, inverse):
    if inverse == True:
        return (2 ** VALUE_NORMALIZED_BIT_SIZE - 1) - value_normalized
    return value_normalized
```

Step 4 After the normalization process and inverting the parameter value if needed the component is weighted.

- If the weight is set to the value `2`, the component is shifted by 1 position to the left, which doubles the value.
- If the weight is set to the value `0`, the component is shifted by 1 position to the right, which halves the value.
- If the weight is set to the value `1`, nothing happens, the value stays as it is.

Example: weight(24063, 2) = 48126

```python
def weight(value_inverted, weight):
    if weight > 1:
        return value_inverted << 1
    elif weight < 1:
        return value_inverted >> 1
    return value_inverted
```

Step 5 Repeat the above `Step 1-4` for each parameter.

Step 6 Sum all previously calculated components to the hop efficiency indicator.

2.8. Requirements

There are several requirements that must be met to ensure that the aggregation works efficiently and does not interfere the regular network traffic. The related non functional requirements are NFR1 and NFR4 which are specified in section 4.2 in the inception part.

2.9. Decisions

The mathematical calculation is based on the following design decisions:

Type sizes of the elements in the calculation The sizes of the different element types are all determined by specific considerations. The following type names of the elements included in the calculation are named as they are used in the P4 code.

Path Efficiency Indicator The calculation is designed to allow each PEI value to be calculated based on a maximum of 256 HEI values. To be sure that no overflow occurs the type path_efficiency_indicator_t is set to 32 bits what is also the Aggregate size within the IOAM Aggregation Option. If 256 HEI values, all with the highest possible value, are aggregated by sum, no overflow will occur because the result would be the maximum value of path_efficiency_indicator_t.

Hop Efficiency Indicator The calculation is designed to be able to calculate the HEI value based on a maximum of 256 components. To be sure that no overflow occurs the type hop_efficiency_indicator_t is set to 24 bits. If 256 components, all with the highest possible value, are aggregated by sum, no overflow will occur because the result would be the maximum value of hop_efficiency_indicator_t.

Weighted Component In Step 4 above, the normalized value is weighted using one of the weighting parameters. If the weight is applied that causes a left shift to double the value, it needs 1 bit more, that's why the type `component_t` is size of 16 bits.

Normalized Value The parameter will be normalized to a predefined type with a range of 15 bits to be comparable to other parameter values.

Parameter Value The size of the parameter type can be of arbitrary size.

Transport Concept The IPv6 Hop by Hop Extension header with the IOAM Aggregation Option and the IOAM Pre-Allocated Trace Option Type will be used as transport the HEI and PEI values. Further details about the aggregation of efficiency indicators, the IOAM Aggregation Option and the IOAM Pre-Allocated Trace Type Option are described in detail in chapter 3 in the construction part.

2.10. Limitations

Information

The current implementation of the mathematical calculation has some limitations. However, these limitations do not interfere with the basic use case of our work.

- The parameter type must be the smallest possible where the maximum value fits. If the parameter's type size is not set to the smallest possible size, it cannot be compared accurately with the other values, and the result will be distorted.
  The following example with the energy mix component, which has a range of 0 to 100, and the value 50 shows the consequences of not setting the type size to the smallest possible size, which in this case is 7 bits:

  Warning

  Type size 7 bit

  $$EngergyMix_{normalized} = f(50, 15, 7) = 50 * \frac{2^{15}}{2^7} = \mathbf{12800}$$

  Type size 8 bit

  $$EngergyMix_{normalized} = f(50, 15, 8) = 50 * \frac{2^{15}}{2^8} = \mathbf{6400}$$

  This minor detail changes the normalized value by half and leads to a distortion of the entire calculation.

- The weighting of the different components may cause a rounding error during a right shift because the least significant bit is lost.

- Normalization can cause loss of value granularity.

3. Programmable Forwarding Planes

In the past network devices had a fixed set of available features defined by the manufacturer. The implementation of new protocols and additional functionality could only be achieved by the manufacturer. Usually the implementation of new features required a redesign of the hardware chip. As modifications to hardware are costly and very time intense this had a negative impact on innovations in the network industry. [5]

As it happened in other domains, today also in networking there is a domain specific processor following the Protocol Independent Switch Architecture (PISA). PISA describes a pipeline of identical match action stages and is a generic forwarding engine designed to be very fast. At the beginning of the pipeline is a parser extracting packet information and at the end of the pipeline there is a deparser writing the modified header information back into the packet. Section 3.3 in the elaboration part, contains a description of PISA in regards to the domain specific language P4. [5]

Figure 3.1 contains an overview of domain specific processors.

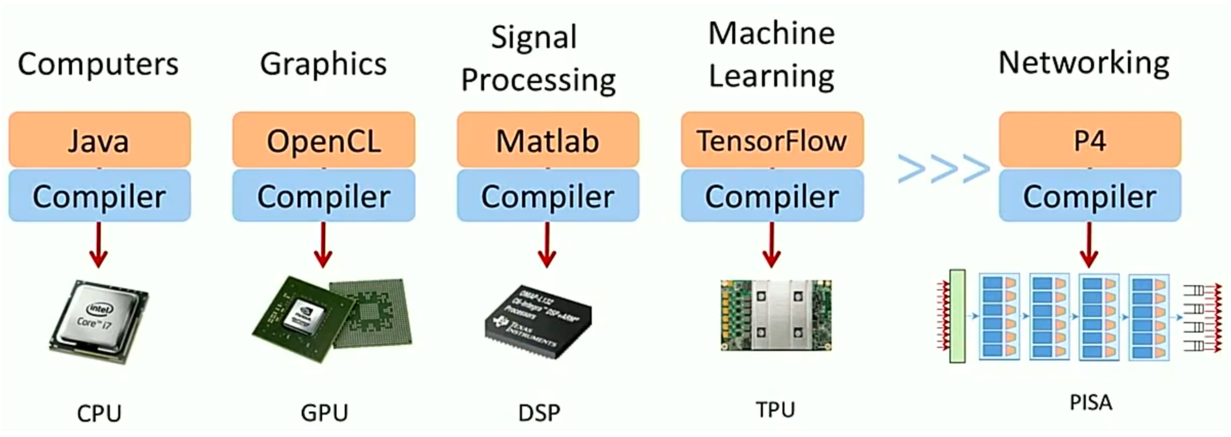


Figure 3.1.: P4 Workflow [5]

P4 programs following the PISA architecture can be compiled down to a binary which can be run on field programmable gate arrays which are chips, which circuits can be modified on the field. With those chips the forwarding pipelines written in P4 achieve the same performance as fixed functions chips. The pipeline can be executed at line rate. [5]

3.1. Control Plane / Data Plane

At this point the difference between the control plane and data plane shall be clarified. The understanding of this basic concept is very important in order to be able to follow the descriptions below.

Control Plane The control plane is the component gathering the information about neighboring routers and the network topology in general. This information is used to build the routing tables also called the forwarding information base (FIB) which contain the forwarding rule-set for each available network. In todays networks protocols like for example OSPF, IS-IS and BGP are responsible to generate the FIB. In the simulated network used in this project the FIB is statically defined without the use of routing protocols.

Data Plane The data plane is the component actually forwarding the data based on forwarding information in the FIB. The FIB is the linkage between the control plane and data plane. During the forwarding operation the data plane updates the header fields of all involved protocols. One can think of the data plane as a pipeline where each packet passes through during forwarding.

3.2. Introduction to P4

The domain specific programming language for network devices Programming Protocol-independent Packet Processors (P4) is used in this project. P4 is a language which can be used to define the forwarding pipeline of network devices.

With P4 in combination with a programmable target arbitrary network features and protocols can be implemented and operated on testing and productive networks. In this project part of the IOAM protocol are implemented in P4 as well as the calculation of the green networking metrics.

These are the components involved in a P4 workflow. [8]

P4 Target Is a network device which can be hardware-based (FPGA, Programmable ASICs) or software (running on x86).

P4 Program (prog.p4) Is target specific and classifies packets by header and the actions to take on incoming packets (e.g., forward, drop).

P4 Compiler Is target specific and generates the runtime mapping metadata to allow the control and data planes to communicate using P4Runtime (prog.p4info). It also generates an executable for the target data plane (target_prog.bin), specifying the header formats and corresponding actions for the target device.

Figure 3.2 contains a logical overview of the components involved in a P4 program.

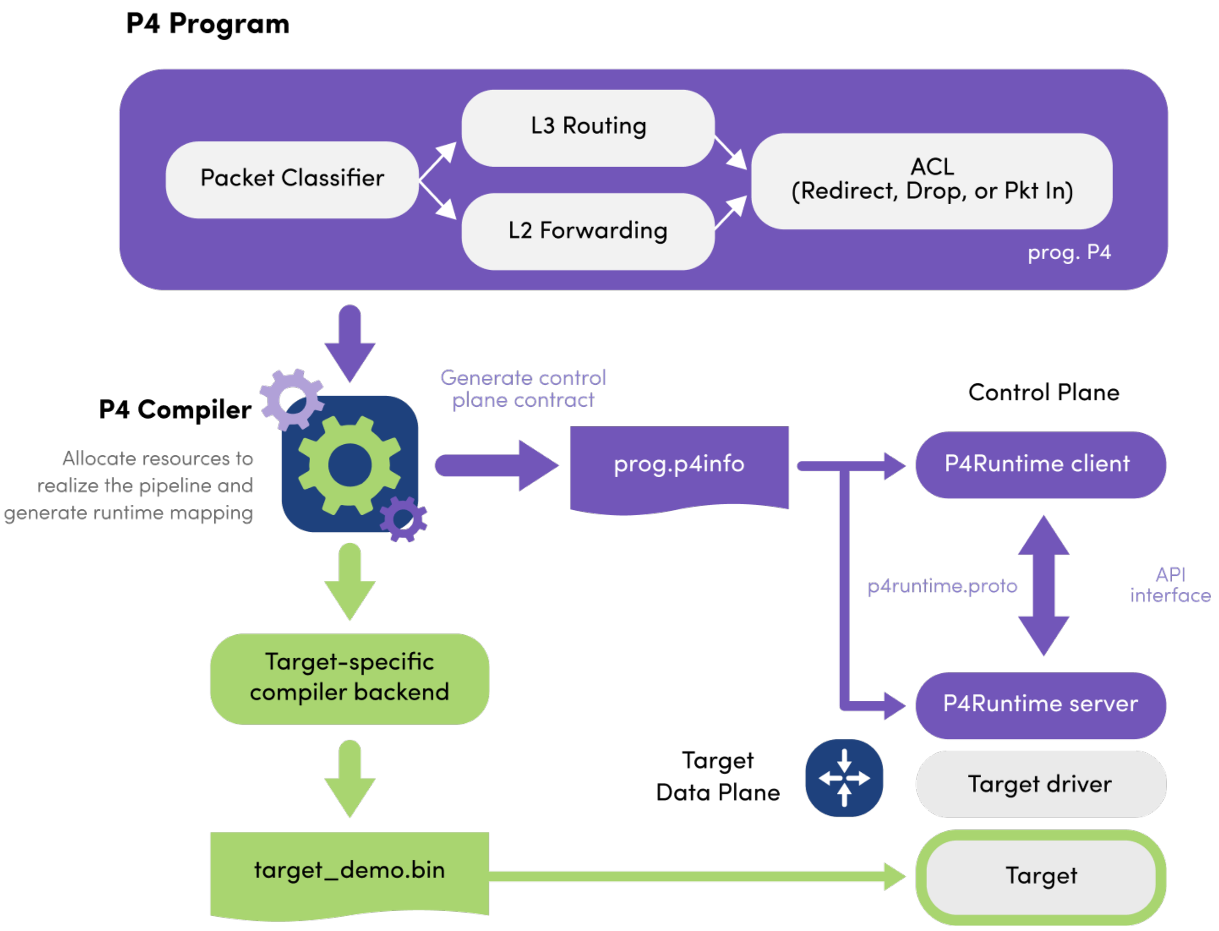


Figure 3.2.: P4 Workflow [8]

For detailed information about P4 refer to the P4 Language Specification.

3.3. P4 Processing Pipeline

The P4 processing pipeline is based on the PISA architecture. Therefore the forwarding of packets by a programmable network device always follows the same structure. Figure 3.3 illustrates the packet forwarding process in a programmable network device.

1. Packet is received as INPUT
2. The headers of the packet is parsed into predefined data structures
3. The predefined data structures holding the header information is passed to the ingress pipeline where forwarding decisions are made
4. The predefined data structures holding the header information is passed to the egress pipeline where additional calculations are performed
5. The updated header information is deparsed into the outgoing packet format
6. The packet is sent as OUTPUT through the port which was determined as part of the forwarding decisions

The following sections will describe the individual components in more detail.

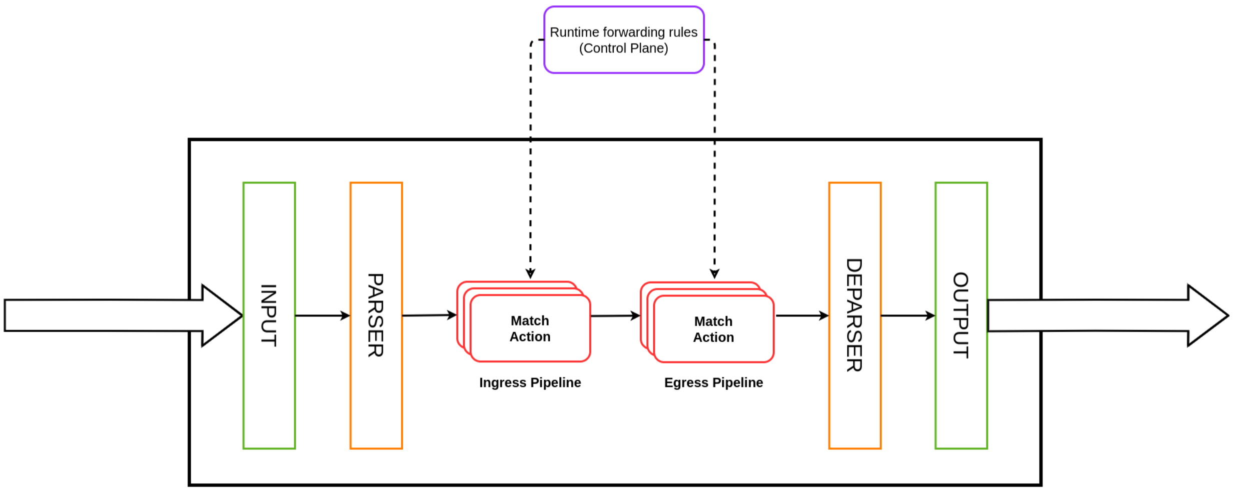


Figure 3.3.: P4 Processing Pipeline

3.3.1. Runtime Forwarding Rules (Control Plane Tables)

With P4 one can define how a network device does the forwarding operation but the behavior of the control plane (e.g. routing protocols such as OSPF) are not programmed using P4. But with P4 one can query information gathered by the control plane doing so called table lookups. These table lookups are the interaction between the data plane and the control plane.

Information inside the control plane tables is accessed once a packet passes the ingress or egress pipelines.

3.3.2. Parser

A parser is a finite state machine which maps the headers and metadata of the incoming packet to predefined data structures. In the code snippet below there are two states.

- start
- parse_ethernet

Once a packet is received the parser is started and immediately transitions to the *parse_ethernet* state. Inside the *parse_ethernet* state based on the ethernet header length (14 bytes) the extract methods writes the same number of bytes into the predefined ethernet header data structure. Depending on the information inside the type field of the already parsed ethernet header, the parser transitions to the state *parse_ipv4* or *parse_ipv6* to continue parsing the IPv4 or IPv6 header respectively. Once all headers are parsed the parser will end up in the accept or reject state.

In case the accept state was reached the program will continue executing the ingress pipeline otherwise the packet will be dropped.

The code listing 3.1 defines the parser described above and additionally defines all related constants, types and headers.

Listing 3.1: P4 Parsing Example

```
// Constants definition
const bit<16> TYPE_IPV4 = 0x800;
const bit<16> TYPE_IPV6 = 0x86DD;

// Type definition
```

```
typedef bit<48> macAddr_t;

// Header definition
header ethernet_t {
    macAddr_t dstAddr;
    macAddr_t srcAddr;
    bit<16>   etherType;
}

// Parser
state start {
    transition parse_ethernet;
}

state parse_ethernet {
    packet.extract(hdr.ethernet);
    transition select(hdr.ethernet.etherType) {
        TYPE_IPV4: parse_ipv4;
        TYPE_IPV6: parse_ipv6;
        default: accept;
    }
}

state parse_ipv4 {
  <-- omitted -->
}

state parse_ipv6 {
  <-- omitted -->
}
```

3.3.3. Ingress Pipeline

The ingress pipeline does the IPv4 and IPv6 forwarding. The information required to do the forwarding correctly is stored inside the runtime forwarding rules in the control plane.

3.3.4. Egress Pipeline

The egress pipeline does additional processing related to network telemetry data stored within the IOAM header. The following operations are performed within the egress pipeline.

- Initialization of network telemetry related headers
  - IPv6 Hop by Hop Extension Header
  - IOAM Trace Option Header
  - IOAM Aggregation Option Header
  - PadN Option Header
- Add trace data of the current node to the IOAM Trace Option Header
- Calculate the HEI
- Aggregate the HEI with the aggregate stored in the IOAM Aggregation Option Header to set the current PEI value in the packet

Similar to the ingress pipeline the egress pipeline uses the runtime forwarding rules in the control plane for IOAM related processing. All related information regarding path tracing and HEI calculation are defined in the control plane.

3.3.5. Deparser

The deparser is used to write the updated header information that was updated in the ingress and egress pipelines back into the packet before it is resent. The emit function, which is defined in the P4 core library for the packet_out data type, is used for this purpose. The emit function writes all headers that are set as valid in the defined sequence to the packet.

4. Development Environment

Before one can get started programming in general, one does need a development environment. Dataplane-Programming in P4 does not only include the installation of an IDE but also the provisioning of network devices - referred to as *targets* from now on - and the design and deployment of a development network. The programmable targets can either be hardware devices with a *field-programmable gate array (FPGA)* or can be virtualized.

The following section will discuss the most important decisions taken concerning the development environment.

4.1. Decisions

Given the circumstance that currently no programmable network hardware is available at OST Eastern Switzerland University of Applied Sciences, it was clear that a virtualized environment will be used. Alexander Clemm, our external advisor, suggested to use Open vSwitch as a programmable target. The setup and usage of Open vSwitch as P4 programmable targets is rather complex and the P4-OVS project is not very well maintained. Without further investigation it would have been a high risk to base our development platform on Open vSwitch.

As an alternative a fully simulated software switch called Behavioral Model Version 2 (BMv2) is available and is the suggested platform to be used to develop P4 programs.

The following sections will further describe the BMv2 programmable target and will discuss other important decisions taken.

4.1.1. Behavioral Model Version 2 (BMv2)

The BMv2 target is a software switch written in C++ not meant to be a production switch but designed for P4 programming with flexibility in mind. One of the key advantage of this target is, that the *control plane* can be defined with arbitrary tables using JSON syntax. This implies that one has great flexibility during development as there are no limitations in the means of available data in the control plane. For the development of efficiency indicators, which are most likely based on data which is not available in todays control plane tables, this is an optimal starting point.

For further information refer to the project behavioral-model on GitHub.

4.1.2. Mininet

In our project Mininet is used to run the development network including BMv2 nodes and Linux endpoints. There is some very nice tooling already available by P4Lang to run networks based on the BMv2 programmable target in Mininet mostly out of the box.

Mininet is an open-source software emulator that is primarily used for creating virtual networks on a single physical or virtual machine. It allows users to simulate complex network topologies and experiment with network configurations in a controlled and isolated environment. Mininet's main purpose is to provide a platform for testing, development, and research in the field of computer networking, enabling users to study and evaluate network protocols and applications without the need for physical network infrastructure.

4.1.3. Virtual Machine

In order to effectively work together in the team without struggling with compatibility issues during development, a dedicated virtual machine is used by both team members. The virtual machine is based on Ubuntu 23.04 which is compatible with all the required software components.

4.1.4. Development Network

As discussed in the previous chapter our development network is run virtually based on BMv2 on Mininet. This chapter describes the topology and the control plane definition of the BMv2 targets in more detail.

The requirements for the development network are:

- Predictable forwarding of traffic
- There should be paths with different lengths
- For direct comparability of PEI values there shall be at least two different paths between an ingress- and an egress router
- Good troubleshooting capabilities (automatic captures and detailed logging)

4.1.5. Topology

The topology of the development network is illustraded in figure 8.1. It consists of six BMv2 programmable targets and four Linux endpoints.

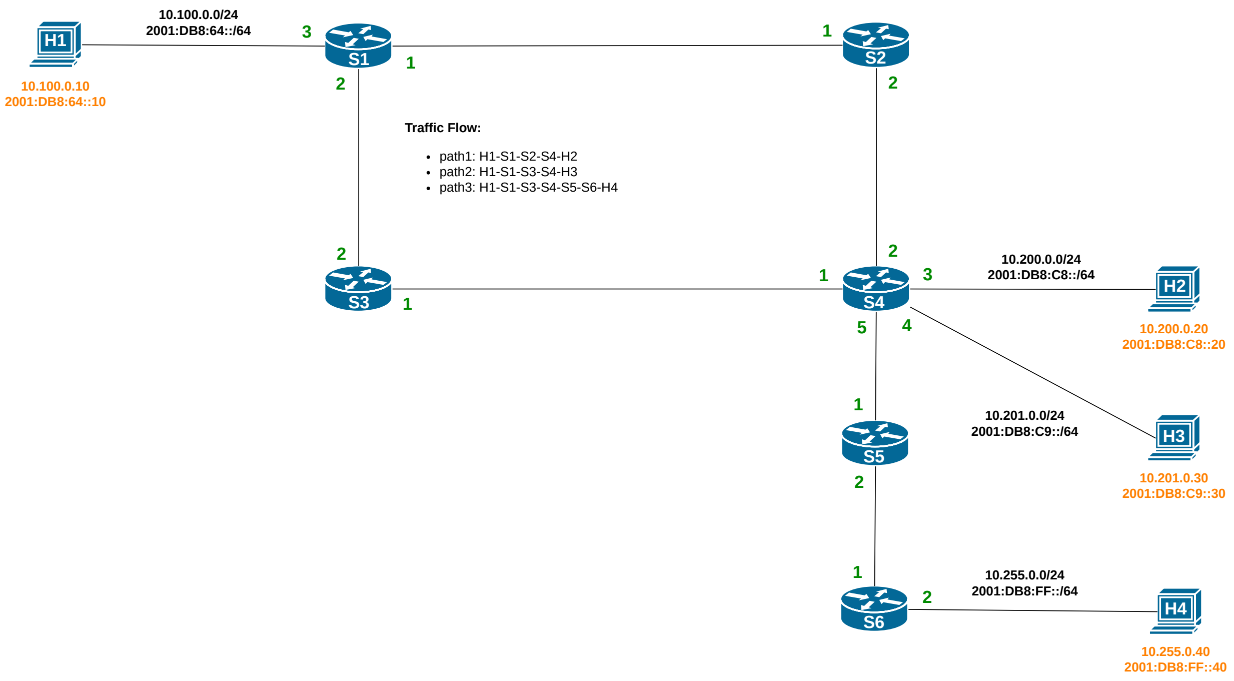


Figure 4.1.: Development Network Topology

Each endpoint is in a different network and connected to one of the BMv2 targets. The endpoints are configured with IPv4 only but IPv6 traffic can still be sent and received using Scapy which is sufficient to test our protocol on IPv6.

The BMv2 targets have no IP addresses assigned to their interfaces as they are not relevant for IP forwarding when using a static control plane.

4.1.6. Control Plane Definition

One of the requirements to the development network is that the traffic forwarding is predictable.

Information

Traffic can flow in both directions. To simplify the testing in a later stage, traffic usually flows from left to right. That implies that *H1* is the sender and *H2, H3 and H4* are considered receivers.

The resulting paths of the situation described in the info box are:

H1 - H2 Traffic on this path flows via S1-S2-S4.

H1 - H3 Traffic on this path flows via S1-S3-S4 and together with the previous path it fulfils the requirement of two different paths between the same ingress- and egress routers.

H1 - H4 Traffic on this path flows via S1-S3-S4-S5-S6 and fulfils the requirement that there must be paths with different lengths.

Information

Traffic sent to *H1* will always flow via *S2*.

The following JSON snippet is part of the control plane definition of S1. It adds a new entry for the IP prefix `10.201.0.0/24` to the *MyIngress.ipv4_lpm* table. Part of the entry is the specification of the action to call, in this case *MyIngress.ipv4_forward* and the action parameters. The action parameters required for IPv4 forwarding is the destination MAC address of the next hop and the egress port used to forward the packet.

```json
{
  "table": "MyIngress.ipv4_lpm",
  "match": {
    "hdr.ipv4.dstAddr": ["10.201.0.0", 24]
  },
  "action_name": "MyIngress.ipv4_forward",
  "action_params": {
    "dstAddr": "08:00:00:00:03:00",
    "port": 2
  }
}
```

4.1.7. IPv6 Functionality

With the tooling provided by P4lang IPv6 forwarding did not work out of the box. Besides the implementation of IPv6 forwarding in P4 and the definition of IPv6 forwarding tables, the p4_runtime library had to be expanded. The library did not support the encoding of IPv6 addresses.

Similar to the IPv4 forwarding table, the IPv6 entry looks as follows.

```json
{
  "table": "MyIngress.ipv6_lpm",
  "match": {
    "hdr.ipv6.dstAddr": ["2001:DB8:C9::", 64]
  },
  "action_name": "MyIngress.ipv6_forward",
  "action_params": {
    "dstAddr": "08:00:00:00:03:00",
    "port": 2
  }
},
```

4.1.8. Traffic simulation

In order to be able to efficiently test our P4 implementation, there is the need to easily simulate traffic. A simple python CLI-application was written which can be used on the Linux endpoints to send UDP traffic from a station A to a station B.

The CLI-application supports the following command line options.

–ipv4 Send only IPv4 traffic

–ipv6 Send only IPv6 traffic

–src The source of the traffic

–dst The destination of the traffic

–count Number of packets to be sent

Information

The application sends real user traffic captured earlier. The usage of real user traffic is a more realistic test scenario as packets of arbitrary size are transmitted over the network. This is particularly interesting to analyze the behavior on exceedance of the maximum transmission unit when adding metadata to a very large packet.

To send 50 packets from h2 to h3 with IPv4 and IPv6 each, the following command would be used within the Mininet CLI.

```
> h2 python3 ./dev-network/utils/testing/send.py --src "h2" --dst "h3" --count 50
```

4.1.9. Automatic Packet Capture

For optimal insight and efficient debugging the Mininet environment is configured to automatically capture traffic on all interfaces of the BMv2 targets. The captures are recreated every time the topology is started. There is an individual pcap file for each interface and direction.

Consider the target S2 with two interfaces eth1 and eth2. The following four pcap files would be created and traffic the traffic captured accordingly.

- s2-eth1_in.pcap
- s2-eth1_out.pcap
- s2-eth2_in.pcap
- s2-eth2_out.pcap

4.2. Installation

All required software is installed on our development virtual machine. As mentioned earlier the virtual machine is based on Ubuntu 23.04.

To get the development environment up and running the following components need to be installed.

4.2.1. Install Dependencies

Some of the installation steps require ‘curl‘ to be installed. Curl is available from the standard Ubuntu apt repository and can be installed with the ‘apt-get install‘ command.

```
sudo apt-get install curl
```

4.2.2. Install Mininet

Mininet is available from the standard Ubuntu apt repository and can be installed with the ‘apt-get install‘ command.

```
sudo apt-get install mininet
```

4.2.3. Install BMv2

The BMv2 software switches are not available in the standard Ubuntu apt repository but after adding the p4lang repository to the apt sources the required software can be installed with the ‘apt-get install‘ command.

```
. /etc/os-release
echo "deb
    http://download.opensuse.org/repositories/home:/p4lang/xUbuntu_${VERSION_ID}/ /" |
    sudo tee /etc/apt/sources.list.d/home:p4lang.list
curl -fsSL
    "https://download.opensuse.org/repositories/home:p4lang/xUbuntu_${VERSION_ID}/Release.key"
    | gpg --dearmor | sudo tee /etc/apt/trusted.gpg.d/home_p4lang.gpg > /dev/null
sudo apt update
sudo apt install p4lang-bmv2
```

4.2.4. Install P4 Compiler (p4c)

The P4 compiler is not available in the standard Ubuntu apt repository but after adding the p4lang repository to the apt sources the required software can be installed with the ‘apt-get install‘ command.

```
source /etc/lsb-release
echo "deb
    http://download.opensuse.org/repositories/home:/p4lang/xUbuntu_${DISTRIB_RELEASE}/
    /" | sudo tee /etc/apt/sources.list.d/home:p4lang.list
curl -fsSL
    https://download.opensuse.org/repositories/home:p4lang/xUbuntu_${DISTRIB_RELEASE}/Release.key
    | gpg --dearmor | sudo tee /etc/apt/trusted.gpg.d/home_p4lang.gpg > /dev/null
sudo apt-get update
sudo apt install p4lang-p4c
```

4.2.5. Install Python Dependencies

The python library scapy is used to send test traffic over the network. Install scapy using the following command.

```
sudo apt-get install python3-scapy
```

4.3. Operation

To start and stop the development network with the BMv2 targets programmed with the current version of the P4 application the 'make' command can be used.

```
# start development network
make run
# stop the development network
make stop
# run automated tests
make test
# run automated demos
make demo
```

Information

This only works inside the root directory of our P4 projects. The Makefile and the tooling behind is provided by P4lang.

5. Testing

Testing is an essential part in software development. Find bugs before they end up in the production environment. In the software engineering practices 2 module at the Eastern Switzerland University of Applied Sciences the citation below was mentioned and it emphasizes the importance of tests in software development.

Legacy code is code without tests. - Michael Feathers

As P4 is a domain-specific language and the code can only run on an actual target, there is no option to perform unit tests on individual actions. As explained earlier with P4 one describes the layout of the forwarding pipeline and this pipeline can only be tested in one piece. This leads to the fact, that performing integration tests by sending traffic through a test network and validating the packet contents on the receiver side, is the way to test the proper functioning of P4 applications. In P4 there is no built in test framework as it is the case in other languages so a solution to test our code needs to be elaborated in the scope of this project.

5.1. Scope

With the elaborated testing platform all IOAM Aggregation Option and IOAM Pre-allocated Trace-Option header fields shall be validated for correctness. All other header fields are not part of the validation process.

5.2. Requirements

The following requirements shall be met by the test application in order to ensure the testability of the elaborated P4 application. Additionally the elaborated solution shall be usable as a traffic generator and data exporter for our demo application.

- The test execution must be fully automated
- The testcases must be definable in a flexible and declarative way so that:
  - The test path can be selected
  - The number of packets can be selected
  - The values in the control plane of an arbitrary table can be set to an arbitrary value (also referred to as parameter patching)
  - Chose whether to automatically continue to the next testcase or wait for user input (especially relevant for the demo application use case)
- The control plane of the software switches must be configurable by testcase
- The receiving station must be able to precalculate the results by testcase
- The test results must be displayed clearly
- The test results must be logged in detail to ensure maximum transparency

- All implemented IOAM Aggregation Option aggregators must be testable
- The IOAM Trace-Option must be testable (including overflow)
- The received data shall be exported to a JSON file readable by the demo application

5.3. Architecture

The specification of a well-designed architecture during the elaboration phase is key for the success in the construction phase. Figure 5.1 illustrates the elaborated architecture for the testing platform.

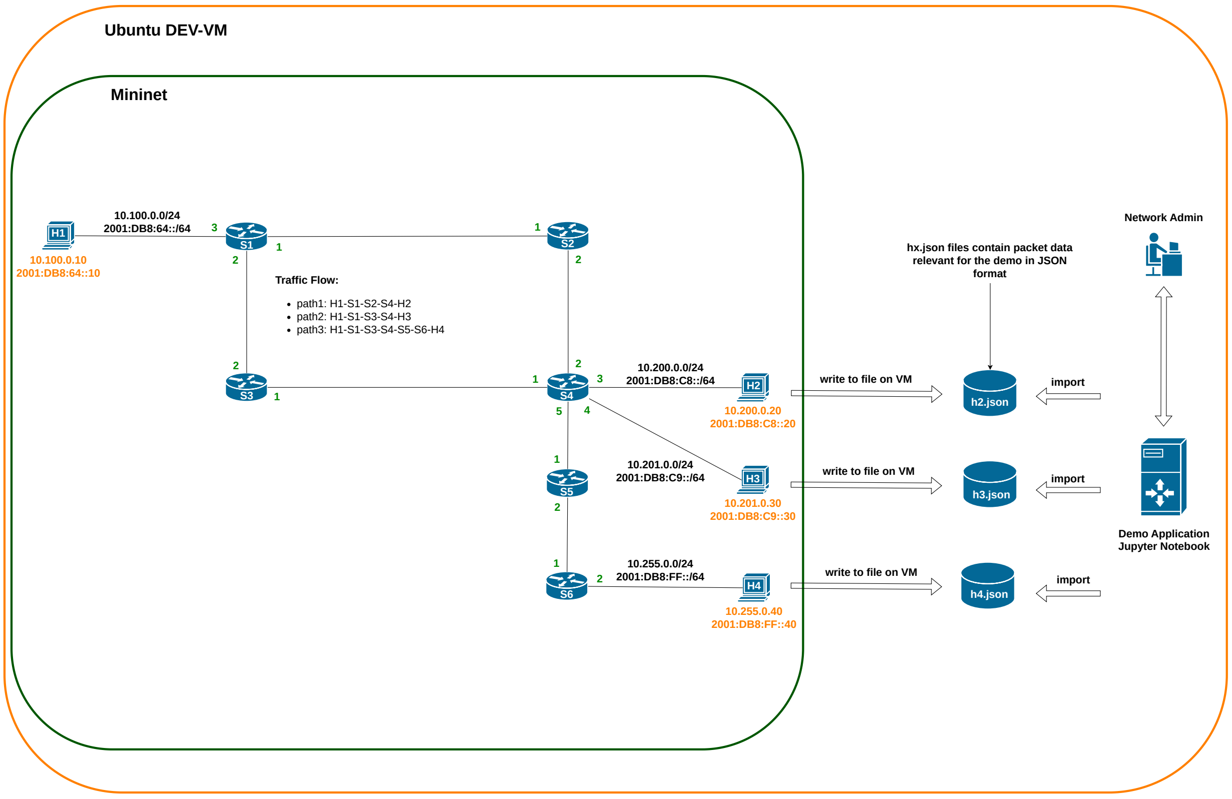


Figure 5.1.: Testing Architecture

5.3.1. Design Decisions

- H2 and H3 are connected to the same switch to ensure that traffic can be sent via the network from S1 (ingress) to S4 (egress) via two different paths.
- The path to H4 is longer than 4 hops, which is more than the pre-allocated node list length, to be able to test the overflow case of the IOAM Pre-allocated Trace-Option.
- Traffic is always sent by H1 and received by H2, H3 or H4 as this will reduce the number of possible paths and therefore reduce overall complexity.
- Data received by H2, H3 and H4 is exported using the JSON format, to a file on the host virtual machine which can be imported by a demo application for further investigation. We neglected the export of data using a network connection, to avoid interference with the test execution and reduce overall complexity.

- The sender and the receivers shall be configured with the exact same testcases file.

5.3.2. Context

The orange boundary in figure 5.1 clarifies that the whole environment is executed on the Ubuntu development virtual machine. Inside that virtual machine the development environment illustrated by the green boundary described in chapter 4 in the elaboration part is run using Mininet.

5.3.3. Test Procedure

This section describes on a high level how the tests are carried out and again refers to figure 5.1.

A user on the Ubuntu DEV-VM, typically the developer who wants to run the test, types `make test` to run the test job defined in the *Makefile*. The test job will start the Mininet development environment. Once the environment is started the testcases definition is read. On all destinations of all testcases *H2, H3 and H4* an instance of the test receiver program is started.

Test Sender According to the parameter patches defined in the current testcase the BMv2 software switches are reprogrammed. Once the network is ready, a process on *H1* is started to send traffic to the destination specified in the testcase. Once the traffic was sent the BMv2 configuration is reset to default. This happens again for every testcase defined.

Test Receiver According to the paths defined inside the testcases json file the relevant testcases are extracted. The receiver now waits for the number of packets expected for the first test case. As soon as the expected number of packets is received the program will carry on and calculate the expected values of all header fields which need to be verified. Each verification is logged. Finally each packet is exported in json format.

The test view collects the information out of the log files.

6. Demo Application

As defined in UC06 in chapter 3 in the inception part, energy efficiency related data shall be collected in a demo application. The use case originally specified that the statistics should be displayed on a CLI interface, but to present the collected data more clearly, it was decided to use a Jupyter Notebook instead, which supports plotting the collected data in diagrams. For demonstration purposes, the specific parameter values will be updated during runtime to see the impact on the PEI of a path.

6.1. Jupyter Notebook

A decision was made to use a Jupyter Notebook as a demo application, which brings several advantages to our project:

- Jupyter Notebooks are interactive, which means that users can play around with the code and see the results in real-time.
- The Jupyter Notebook allows to create visualizations, such as graphs, charts and plots to illustrate complex concepts and data. This makes it easier to understand and remember the demo.
- Jupyter Notebook is relatively easy to set up and it's not necessary to install huge testing framework to visualize the findings of the project.

6.2. Demonstration Content

The following scenarios should be provided in the demo application:

- The variation of the energy mix value on path 1 (S1-S2-S4) and path 2 (S1-S3-S4) is to be visualized during a simulated day where the sustainable energy generated by solar power fluctuates.
- A comparison should visualize the different PEI values between path 1 and path 2. For this demo case it is necessary that the ingress and egress nodes are the same because paths with different origins and destinations are not directly comparable.
- The most efficient and most inefficient node of path 1 should be visualized in a diagram.
- A diagram should visualize the composition of the PEI value divided into the individual components which contributed to the PEI value.
- A path statistic shows the most important information of a path as defined in UC06 in chapter 3 in the inception part.

6.2.1. Example Calculation

The demo application should provide a step-by-step guide explaining the calculation of the PEI value based on a specified demo case.

Part III.

Construction

1. Parser

This chapter describes the basic concepts related to a packet parser in P4. Furthermore the parser logic implemented in this project to parse network packets in the format introduced in the previous chapter will be described in detail.

1.1. Concepts

A P4 parser describes a state machine with one start state and two final states. The start state is always named start. The two final states are named accept (indicating successful parsing) and reject (indicating a parsing failure). The start state is part of the parser, while the accept and reject states are distinct from the states provided by the programmer and are logically outside of the parser. Figure 1.1 illustrates the general structure of a parser state machine. [9]

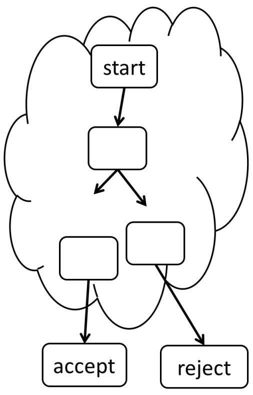


Figure 1.1.: P4 Parser Structure

1.1.1. Parser Declaration

To declare a parser in P4 the parser keyword is used. The arguments provided to a parser are the incoming packet, the target headers data structure and metadata which might be updated during parsing.

Listing 1.1: P4 Parser Declaration

```
parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata) {}
```

1.1.2. Parser State Definition

An example state definition is given in section 3.3.2 in the elaboration part within the introduction to P4. For even more information refer to the P4 language specification section Packet parsing.

1.2. Parser State Machine

This section outlines the state machine of the parser elaborated during this project. Figure 1.2 contains the diagram of the finite state machine of the parser.

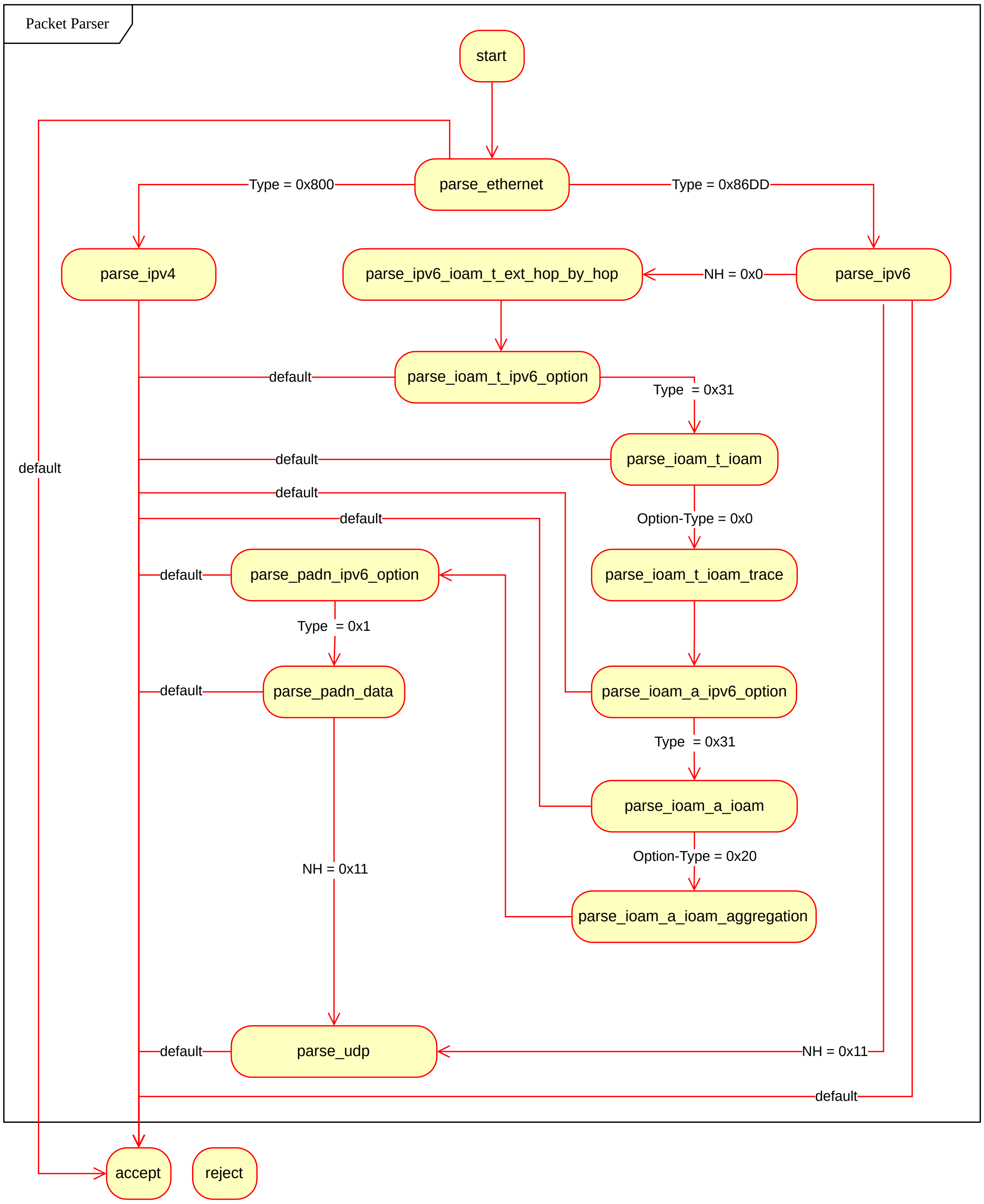


Figure 1.2.: Packet Parser State Machine

1.2.1. Parsing Flow

This section explicates in prose the flow of packet parsing given the state machine in figure 1.2.

Once a network packet enters the programmable target it is parsed using the given state machine.

- The parsing process begins in the **start** state and to parse the first header it immediately transitions to the **parse ethernet** state because the very first header present is the Ethernet header. The Ethernet header contains a type field which identifies the following protocol. Our implementation supports the forwarding of IPv4 and IPv6 packets or native Ethernet frames. In the **parse ethernet** state the transition to the next state depends on the value in the type field.
- If the value is equal to *0x800* the state machine transitions to the **parse ipv4** state. In this state the IPv4 header data is extracted and because no efficiency indicator specific implementation was attempted for IPv4 in this project, after extraction the parsing ends in the **accept** state.
- If the value is equal to *0x86DD* the state machine transitions to the **parse ipv6** state. In this state the IPv6 header data is extracted and the transition to the next state is based on the *Next Header* field inside the IPv6 header.
- If the value is neither *0x11* nor *0x0* the state machine transitions to the **accept** state.
- If the value is equal to *0x11* the state machine transitions to the **parse udp** state where the data is extracted into the provided UDP header data structure. After the UDP header was extracted the state machine transitions to the **accept** state.
- If the value is equal to *0x0* the state machine transitions to the **parse ipv6 ioam t ext hop by hop** state where the *Next Header* and *Length* of the Hop-by-Hop Options extension header is extracted. Once the data is extracted the state machine transitions to the **parse ioam t ipv6 option** state where the *Option Type* and *Option Length* is extracted. The transition to the next state is based on the *Option Type* value.
- If the value is anything else than *0x31* the state machine transition to the **accept** state.
- If the value is equal to *0x31* the option type is IOAM and the state machine transitions to the **parse ioam t ioam** state. In this state the *IOAM Option Type* and the *Reserved* field are extracted. The transition to the next state is based on the *IOAM Option Type* value.
- If the value is anything else than *0x0* the state machine transition to the **accept** state.
- If the value is equal to *0x0* the IOAM option type is the *Pre-allocated Trace-Option* and the state machine transitions to **parse ioam t ioam trace**. In this state the IOAM Trace Option data is extracted into the given data structure. Once the data is extracted the state machine transitions to the **parse ioam a ipv6 option** state where the *Option Type* and *Option Length* is extracted. The transition to the next state is based on the *Option Type* value.
- If the value is anything else than *0x31* the state machine transition to the **accept** state.
- If the value is equal to *0x31* the option type is IOAM and the state machine transitions to the **parse ioam a ioam** state. In this state the *IOAM Option Type* and the *Reserved* field are extracted. The transition to the next state is based on the *IOAM Option Type* value.

- If the value is anything else than *0x0* the state machine transition to the **accept** state.
- If the value is equal to *0x20* the IOAM option type is the *Aggregation-Option* and the state machine transitions to **parse ioam a ioam aggregation**. In this state the IOAM Aggregation Option data is extracted into the given data structure. Once the data is extracted the state machine transitions to the **parse padn ipv6 option** state where the *Option Type* and *Option Length* is extracted. The transition to the next state is based on the *Option Type* value.
- If the value is anything else than *0x1* the state machine transition to the **accept** state.
- If the value is equal to *0x1* the option type is the *PadN Option* and the state machine transitions to the **parse padn data** state. In this state the padding data is extracted and depending on the *Next Header* field of the Hop-by-Hop Options extension header the state machine transitions to the next state.
- If the value is anything else than *0x11* the state machine transitions to the accept state.
- If the value is equal to *0x11* the state machine transitions to the **parse udp** state where the data is extracted into the provided UDP header data structure. After the UDP header was extracted the state machine transitions to the **accept** state.

1.2.2. States

This section briefly explains each state of the state machine mentioning the target data structure and transition conditions.

parse_ethernet Parses the Ethernet header. 14 bytes are extracted into an instance of the *ethernet_t* data structure and the transition to the next state is based on the Ethernet type field.

parse_ipv4 Parses the IPv4 header. 20 bytes are extracted into an instance of the *ipv4_t* data structure. There are no transition conditions.

parse_ipv6 Parses the IPv6 header. 40 bytes are extracted into an instance of the *ipv6_t* data structure. The transition is based on the next header field.

parse_ipv6_ioam_t_ext_hop_by_hop Parses IPv6 extension header related fields. The Hop-by-Hop Option data parsing is not part of this state. 2 bytes are extracted into the *ipv6_ext_hop_by_hop_t* data structure. There are no transition conditions.

parse_ioam_t_ipv6_option Parses the fields identifying the IPv6 Hop-by-Hop Option. In this state the expected option type is the IOAM option. 2 bytes are extracted into an instance of the *ipv6_option_t* data structure. The transition is based on the option type field.

parse_ioam_t_ioam Parses the fields identifying the IOAM option type. In this state the expected option type is the Pre-allocated IOAM Trace-Option. 2 bytes are extracted into an instance of the *ioam_t* data structure and the transition is based on the IOAM option type.

parse_ioam_t_ioam_trace Parses the Pre-allocated IOAM Trace-Option data. 8 bytes plus the length of the node list are extracted into an instance of the *ioam_trace_t* data structure. There are no transition conditions.

parse_ioam_a_ipv6_option Parses the fields identifying the IPv6 Hop-by-Hop Option. In this state the expected option type is the IOAM option. 2 bytes are extracted into an instance of the *ipv6_option_t* data structure. The transition is based on the option type field.

parse_ioam_a_ioam Parses the fields identifying the IOAM option type. In this state the expected option type is the IOAM Aggregation-Option. 2 bytes are extracted into an instance of the *ioam_t* data structure and the transition is based on the IOAM option type.

parse_ioam_a_ioam_aggregation Parses the IOAM Aggregation-Option data. 8 bytes plus the length of the node list are extracted into an instance of the *ioam_aggregation_t* data structure. There are no transition conditions.

parse_padn_ipv6_option Parses the fields identifying the IPv6 Hop-by-Hop Option. In this state the expected option type is the PadN option. 2 bytes are extracted into an instance of the *ipv6_option_t* data structure. The transition is based on the option type field.

parse_padn_data Parses the PadN NULL bytes. The number of NULL bytes depends on the amount of padding required. In this scenario 6 bytes padding is required in total, less the 2 bytes required for the PadN option identification, so a total of 4 NULL bytes are present. 4 bytes are extracted into an instance of the *option_padn_data_t* data structure. The transition is based on the next header field of the Hop-by-Hop Options extension header.

parse_udp Parses the UDP header. 8 bytes are extracted into an instance of the *udp_t* data structure. There are no transition conditions.

accept Final state after successful parsing.

reject Final state after failed parsing.

2. Forwarding Pipelines

This chapter describes the implementation of the two forwarding pipelines which are present in the P4 program. For more information about conceptual decisions refer to the section 3.3 in the elaboration part.

2.1. Ingress Pipeline

As soon as the parsing of the packet is complete, it is processed by the ingress pipeline. The ingress pipeline is responsible for the proper IPv4 and IPv6 forwarding. Header fields are updated and forwarding decisions are made based on the information inside the runtime configuration of the BMv2 targets.

2.1.1. Actions

The following actions are defined in the ingress pipeline.

drop This action is used to drop a packet in case no matching forwarding entry was found in the control plane tables.

ipv4_forward This action is used to update the IPv4 header fields and the metadata accordingly. It takes the destination MAC address and the egress port as input. The *egress_spec* field of the metadata is set to the value of the egress port parameter. The Ethernet header fields are updated accordingly. The time to live header field is decremented by one.

ipv6_forward This action is used to update the IPv6 header fields and the metadata accordingly. It takes the destination MAC address and the egress port as input. The *egress_spec* field of the metadata is set to the value of the egress port parameter. The Ethernet header fields are updated accordingly. The hop limit header field is decremented by one.

2.1.2. Tables

The following tables are defined in the ingress pipeline.

ipv4_lpm This table is used to lookup the forwarding information required to forward an IPv4 packet given the IPv4 destination address. The required forwarding information is the destination MAC address and the egress port. The matching is based on the longest prefix match algorithm which is applied on the specified key which is the destination IPv4 address. On a match the action *ipv4_forward* is called and on a miss the *drop* action is called.

ipv6_lpm This table is used to lookup the forwarding information required to forward an IPv6 packet given the IPv6 destination address. The required forwarding information is the destination MAC address and the egress port. The matching is based on the longest prefix match algorithm which is applied on the specified key which is the destination IPv6 address. On a match the action *ipv6_forward* is called and on a miss the *drop* action is called.

2.1.3. Apply

The apply block of the ingress pipeline applies the appropriate table based on the IP header validity. In case the IPv4 header is valid, which means that the header was parsed successfully, the *ipv4_lpm* table is applied. In case the IPv6 header is valid, the *ipv6_lpm* table is applied.

Listing 2.1: Ingress pipeline apply block

```
apply {
    // IP Forwarding
    if (hdr.ipv4.isValid()) {
        ipv4_lpm.apply();
    }
    if (hdr.ipv6.isValid()) {
        ipv6_lpm.apply();
    }
}
```

2.2. Egress Pipeline

Once a packet has passed the ingress pipeline, it is processed by the egress pipeline. The egress pipeline does not specify any actions or tables but calls dedicated controllers instead. In the egress pipeline energy efficiency indicator related process is carried out.

2.2.1. Apply

As specified in UC03 in chapter 3 in the inception part, the energy efficiency related data should be stored inside the Hop-by-Hop Option extension header. This implies that the controllers specified in the apply block of the egress pipeline are only applied in case the IPv6 header is valid. If the incoming packet is a valid IPv6 packet the controllers are called in the specific order.

1. Initialize the IPv6 Hop-by-Hop Options extension header
2. Add path tracing information using the IOAM Pre-Allocated Trace Option
3. Calculate the HEI and initialize related IOAM Aggregation Option metadata
4. Aggregate the HEI stored inside the metadata to the data packet with the specified aggregator

Listing 2.2: Egress pipeline apply block

```
apply {
    if (hdr.ipv6.isValid()) {
        // Initialize IPv6 Extension Headers
        process_ipv6_ext_header_init.apply(hdr, meta, standard_metadata);
        // IOAM Tracing
        process_ioam_tracing.apply(hdr, meta, standard_metadata);
        // Efficiency Indicator
        process_efficiency_indicator.apply(hdr, meta, standard_metadata);
        // IOAM Aggregation
        process_ioam_aggregation.apply(hdr, meta, standard_metadata);
    }
}
```

The controllers are described in more detail in the following chapters.

3. IPv6 Extension Header

As specified in the use case description of UC03 in chapter 3 in the inception part, the IOAM header information shall be added to IPv6 packets using the Hop-by-Hop Options header defined in RFC 2460. This chapter briefly describes the header format elaborated during the construction phase of this project.

3.1. Hop-by-Hop Options Header

The Hop-by-Hop Options header is used to carry optional information that must be examined by every node along a packet's delivery path. The Hop-by-Hop Options header is identified by a Next Header value of 0 in the IPv6 header, and has the following format: RFC 2460

Information

The Options field in the format specification in listing 3.1 is of variable length and carries option specific data.

Listing 3.1: Hop-by-Hop Option Format

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Option Type  | Opt Data Len  |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
|                                                               |
.                                                               .
.                            Options                            .
.                                                               .
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Warning

Each extension header should occur at most once, except for the Destination Options header which should occur at most twice (once before a Routing header and once before the upper-layer header). **This means that there will be one Hop-by-Hop Options header carrying multiple options inside an IPv6 packet.**

3.1.1. Options

The options carried by an IPv6 packet in an efficiency indicator enabled network, contains at least the following options inside the Hop-by-Hop Options header.

IOAM Trace Option 224-bit (including the fields: Option Type and Opt Data Len) header area used to store path tracing related data. More detailed information about the specific header fields of that option are described in chapter 4 in the construction part.

IOAM Aggregation Option 160-bit (including the fields: Option Type and Opt Data Len) header area used to store energy efficiency indicator data More detailed information about the specific header fields of that option are described in chapter 5 in the construction part.

PadN 48-bit (including the fields: Option Type and Opt Data Len) header area used to pad out the containing header to a multiple of 8 octets in length.

3.2. Header Structure Overview

The header structure specification in listing 3.2 shows the structure of the Hop-by-Hop Options header used in this project. Each line has a length of 4 bytes and the total header has a size of 56 bytes.

Information

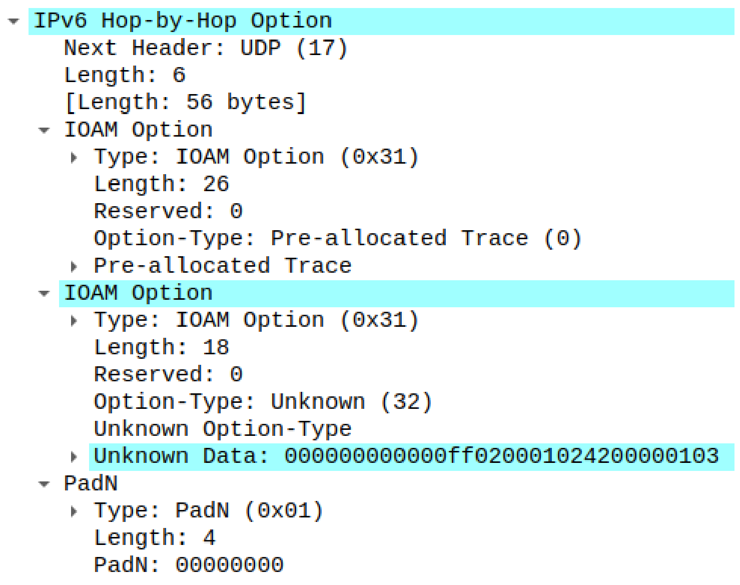
The three Options already explained above are preceded by the two fields Option Type and Opt Data Len.

Listing 3.2: IPv6 Hop-by-Hop Options Header Structure

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Next Header  |  Hdr Ext Len  |  Option Type  | Opt Data Len |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       IOAM Trace Option                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       IOAM Trace Option                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       IOAM Trace Option                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       IOAM Trace Option                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       IOAM Trace Option                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       IOAM Trace Option                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       IOAM Trace Option       |  Option Type  | Opt Data Len |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     IOAM Aggregation Option                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     IOAM Aggregation Option                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     IOAM Aggregation Option                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     IOAM Aggregation Option                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   IOAM Aggregation Option     |  Option Type  | Opt Data Len |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              PadN                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

3.2.1. Wireshark

In the following Wireshark capture, the previously described IPv6 Hop-by-Hop Option header is visible in the structure of a packet. As one can see, there is one IOAM Option for the IOAM Pre-allocated Trace-Option and another one for the IOAM Aggregation Option. At the end there is the PadN option, adding the required padding for the extension header alignment.



```
▾ IPv6 Hop-by-Hop Option
    Next Header: UDP (17)
    Length: 6
    [Length: 56 bytes]
  ▾ IOAM Option
    ▸ Type: IOAM Option (0x31)
      Length: 26
      Reserved: 0
      Option-Type: Pre-allocated Trace (0)
    ▸ Pre-allocated Trace
  ▾ IOAM Option
    ▸ Type: IOAM Option (0x31)
      Length: 18
      Reserved: 0
      Option-Type: Unknown (32)
      Unknown Option-Type
    ▸ Unknown Data: 000000000000ff020001024200000103
  ▾ PadN
    ▸ Type: PadN (0x01)
      Length: 4
      PadN: 00000000
```

Figure 3.1.: Wireshark capture of the IPv6 Hop-by-Hop Option header

3.3. Header Initialization in P4

The initialization of the IPv6 Hop-by-Hop extension header is implemented in a seperate controller called *process_ipv6_ext_header_init* located in *efficiency-indicator-p4/includes/ipv6_ext_header.p4*.

3.3.1. Actions

init_ipv6_ext_hop_by_hop This action is used to initialize the header fields of the IPv6 Hop-by-Hop Option extension header. Additionally the next header field of the IPv6 header is updated to the value of 0, which is defined to identify the following header as a Hop-by-Hop extension header.

3.3.2. Apply

The apply block calls the action described above in case the IPv6 Hop-by-Hop extension header is not yet initialized.

Listing 3.3: Hop-by-Hop extension header controller apply block

```
apply {
    if (!hdr.ipv6_ext_hop_by_hop.isValid()) {
        init_ipv6_ext_hop_by_hop();
    }
}
```

4. IOAM Tracing

As specified in section 4.1 in the inception part in FR6, the calculated PEI value must be assignable to the path the packet traversed. In order to fulfill that functional requirement the IOAM Trace-Option type is used to store the node identifiers of the nodes the packet traversed as metadata in the packet.

The IOAM Trace-Option is stored inside the IPv6 Hop-By-Hop extension header as described in chapter 3 in the construction part.

4.1. Header Structure

As described in chapter 3 in the construction part, the Pre-allocated IOAM Trace-Option header is inserted into the IOAM option as part of the IPv6 Hop-by-Hop Options extension header. The Pre-allocated IOAM Trace-Option header is of fixed size and consists of two components. The first component, illustrated by listing 4.1, contains general IOAM metadata required during processing. The second component, illustrated by listing 4.2, is the pre-allocated node data list containing the actual trace data in the format of a defined IOAM Trace-Type according to RFC 9197.

In our test environment the number of pre-allocated node data list entries is four, which means that a maximum number of four hops can be traced.

For more information about the individual fields refer to RFC 9197.

Listing 4.1: Pre-allocated and Incremental Trace-Option header RFC 9197

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Namespace-ID           |NodeLen  | Flags | RemainingLen|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               IOAM Trace-Type                 |  Reserved     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Listing 4.2: 4 octets aligned option data RFC 9197

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+<-+
|                                                               |  |
|                       node data list [0]                      |  |
|                                                               |  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  D
|                                                               |  a
|                       node data list [1]                      |  t
|                                                               |  a
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                             ...                               ~  S
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  p
|                                                               |  a
|                       node data list [n-1]                    |  c
|                                                               |  e
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  |
|                                                               |   |
|                       node data list [n]                      |   |
|                                                               |   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+<-+
```

4.1.1. IOAM Trace-Type

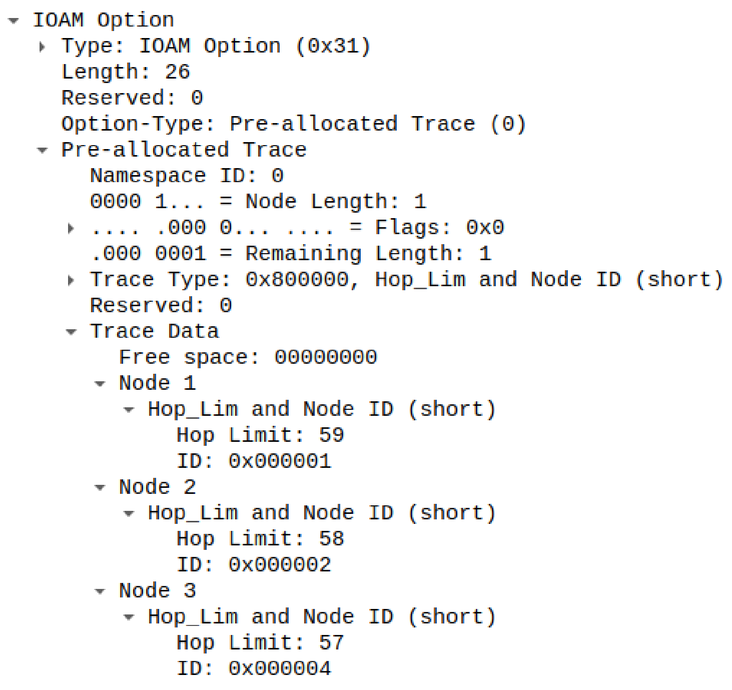
The format of the node data list entries follows the Hop_Lim and node_id Short format as specified in RFC 9197. The first 8 bits of each entry is used to store the current hop limit and the last 24 bits are used to store the node id of the current node.

Listing 4.3: Hop_Lim and node_id Short Format RFC 9197

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Hop_Lim     |              node_id                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

4.1.2. Wireshark

In the following Wireshark capture, the previously described IOAM Pre-allocated Trace-Option is visible in the header structure of a packet. As one can see, three nodes have already been traced and it's possible to trace one more due to the pre-allocation of four entries in the node list.



```
▾ IOAM Option
  ▸ Type: IOAM Option (0x31)
    Length: 26
    Reserved: 0
    Option-Type: Pre-allocated Trace (0)
  ▾ Pre-allocated Trace
      Namespace ID: 0
      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: 59
            ID: 0x000001
      ▾ Node 2
        ▾ Hop_Lim and Node ID (short)
            Hop Limit: 58
            ID: 0x000002
      ▾ Node 3
        ▾ Hop_Lim and Node ID (short)
            Hop Limit: 57
            ID: 0x000004
```

Figure 4.1.: Wireshark capture of the IOAM Trace-Type

4.2. Implementation in P4

The following sections will describe the individual P4 software components related to the IOAM Pre-allocated Trace-Option in more detail excluding the parser which is described in chapter 1 in the construction part.

4.2.1. Constants

The constants definition can be found in *efficiency-indicator-p4/includes/constants.p4*. The following values related to the IOAM Trace-Option are defined as constants.

Listing 4.4: IOAM Trace-Option Constants

```
const bit<8> IOAM_OPTION_TYPE_CHG = 0x31;
const bit<8> IOAM_PRE_ALLOC_TRACE_OPTION_TYPE = 0x0;
const bit<8> IOAM_TRACE_NUM_NODES = 4;
const bit<8> IOAM_TRACE_DATA_LIST_LEN = IOAM_TRACE_NUM_NODES * 32;
const bit<8> IOAM_TRACE_OPTION_DATA_LEN = 10 + IOAM_TRACE_NUM_NODES * 4;
```

IOAM_OPTION_TYPE_CHG This value is used in the IPv6 Hop-By-Hop extension header to identify the IOAM Option. This specific value specifies that the IOAM Option data is allowed to change in transit. The value is defined by IANA [7].

IOAM_PRE_ALLOC_TRACE_OPTION_TYPE This value is used to identify the IOAM Pre-allocated Trace-Option type. The value is defined by IANA [6].

IOAM_TRACE_NUM_NODES This value defines the length in number of nodes of the pre-allocated node_list.

IOAM_TRACE_DATA_LIST_LEN This value defines the length in bits of the pre-allocated node_list. One node_list entry is 32 bits in size. To calculate the value apply the formula below.

$$IOAM\_TRACE\_DATA\_LIST\_LEN = IOAM\_TRACE\_NUM\_NODES \cdot 32 \text{ Bits}$$

IOAM_TRACE_OPTION_DATA_LEN This value defines the total length of the IOAM Pre-allocated Trace-Option. It includes IOAM Option identifying information and the IOAM Pre-allocated Trace-Option header. The value is the sum of the following headers and header fields, where the last summand of the calculation represents the size of the node list in bytes containing the trace data.

$$\begin{aligned} IOAM\_TRACE\_OPTION\_DATA\_LEN = {} & 1 \text{ Byte IOAM Opt-Type} \\ & + 1 \text{ Byte Reserved} \\ & + 8 \text{ Byte IOAM Trace-Option Header} \\ & + (\text{IOAM\_TRACE\_NUM\_NODES} \cdot 4 \text{ Byte}) \end{aligned}$$

The constants described above are used in the following type, header and controller definition.

4.2.2. Types

There are no IOAM Trace-Option specific type aliases.

4.2.3. Headers

This section describes the definition of the IOAM Trace-Option header. The header definition can be found in *efficiency-indicator-p4/includes/headers.p4*. The header fields are defined according to RFC 9197.

The IOAM Trace-Option header type is called *ioam_trace_t* and contains seven fixed sized header fields. The size of the *dataList* field is defined at compile time using one of the constants described in the previous section.

Listing 4.5: IOAM Trace-Option Header

```
header ioam_trace_t {
    bit<16> namespaceID;
    bit<5> nodeLen;
    bit<4> flags;
    bit<7> remainingLen;
    bit<24> ioamTraceType;
    bit<8> reserved;
    bit<(IOAM_TRACE_DATA_LIST_LEN)> dataList;
}
```

4.2.4. Controller

The *process_ioam_tracing* controller is responsible for IOAM Trace-Option related processing. It takes the header structures initialized by the parser, custom metadata and standard metadata as input.

The controller definition can be found in *efficiency-indicator-p4/includes/ioam_tracing.p4*.

In the following section the controllers actions are described which give information about how header fields are updated in this state of the match action pipeline.

4.2.4.1. Actions

Actions are the place where the actual manipulation of header data occurs. In the case of the IOAM Trace-Option controller there are mainly two actions where one is responsible to initialize the IOAM Trace-Option header if it does not exist and the other one is used to add an additional trace entry.

ioam_trace_push This action sets the IPv6 Hop-by-Hop option, IOAM option and the IOAM Trace-Option headers to valid and all related header fields are set to the initial value.

ioam_trace_node This action is used to add the 32 bit node entry in the hop_lim node_id short format according to RFC 9197. The content of the *dataList* field is shifted to the left by 32 bits and afterwards the last 32 bits of the *dataList* are set to the new node entry using the logical *OR* operation. Finally the remaining length field is decremented by one.

4.2.4.2. Tables

This section describes the tables relevant for the IOAM Trace-Option. As described in section 3.3.1 in the elaboration part, the control plane tables are the linkage between the control plane and the data plane. In this case the nodeID value needed in the *ioam_trace_node* action is gathered from the control plane through a table lookup.

The table is called *ioam_trace_node_exact* and is defined as follows for s1 in *efficiency-indicator-p4/dev-network/s1-runtime.json*.

Listing 4.6: S1 IOAM Trace-Option Table Definition

```
{
  "table": "MyEgress.process_ioam_tracing.ioam_trace_node_exact",
  "match": {
    "hdr.ethernet.srcAddr": [
      "08:00:00:00:01:00"
    ]
  },
  "action_name": "MyEgress.process_ioam_tracing.ioam_trace_node",
  "action_params": {
    "nodeID": 1
  }
}
```

The complete table name consists of multiple parts separated by a dot.

MyEgress Name of the pipeline

process_ioam_tracing Name of the controller

ioam_trace_node_exact Name of the actual table

Each table must specify at least one match entry. In this case the source ethernet address is matched on the given address. The given address is the address of s1.

Each table must specify an action which will be called on a match. In this case the action *ioam_trace_node* will be called on a match and the parameter nodeID with the value set to 1 will be passed to the specified action.

In P4 the table match action is specified as follows.

Listing 4.7: IOAM Trace-Option Match Action

```
    table ioam_trace_node_exact {
        key = {
            hdr.ethernet.srcAddr: exact;
        }
        actions = {
            ioam_trace_node;
            NoAction;
        }
        size = 1;
    }
```

4.2.4.3. Apply

The apply block in a P4 program is used to make decision which table to apply or action to call based on conditions most likely given by header values or header validity state. To initialize the IOAM Trace-Option header, based on the statement in the apply block, the controller calls the *ioam_trace_push* action in case the IOAM trace header is invalid. In case the remainingLen is greater than 0 the table *ioam_trace_node_exact* is applied to add the current node to the nodeList. Otherwise the overflow bit is set to 1.

Listing 4.8: IOAM Trace-Option Apply Block

```
apply {
    if (!hdr.ioam_t_ioam_trace.isValid()) {
        ioam_trace_push();
    }
    if (hdr.ioam_t_ioam_trace.remainingLen > 0) {
        // add node id and hop count to array
        ioam_trace_node_exact.apply();
    } else {
        // set overflow bit
        hdr.ioam_t_ioam_trace.flags = hdr.ioam_t_ioam_trace.flags | 0x8;
    }
}
```

4.3. Limitations

- The current implementation supports a maximum number of 4 traceable nodes which is enough for testing purposes in the simulated environment. This limitation arises due to the maximum header field size of 128 bits on the BMv2 target. Each entry in the node list has a size of 32 bits. In case similar limitations exist on hardware targets, a possible mitigation could the usage of a dedicated header stack type for the node list.

5. IOAM Aggregation

As specified in section 4.1 in the inception part, in FR3, FR4 and FR5, the energy efficiency related data shall be stored in the corresponding packet as part of the header metadata. The non functional requirement NFR5 specified in 4.2 in the inception part additionally states that the energy efficiency related data shall be of constant size regardless of the number of hops. In order to fulfill this requirement data of an individual node must be aggregated and that is where the IOAM Aggregation Option comes in. The IOAM Aggregation Option is used to aggregate the HEI to the PEI. The IOAM Aggregation Option supports different aggregators with which all three mentioned functional requirements can be fulfilled.

5.1. Header Structure

As described in chapter 3 in the construction part, the IOAM Aggregation Option header is inserted into the IOAM option as part of the IPv6 Hop-by-Hop Options extension header. The structure of the IOAM Aggregation Option header is illustrated in listing 5.1. The header has a constant size of 16 bytes.

Listing 5.1: IOAM Aggregation Option Type Format [2]

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Namespace-ID           | Flags |       Reserved        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               IOAM Data Param                 |  Aggregator   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Aggregate                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Auxil-data Node-ID                 |   Hop Count   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

A selection of header fields are further explained in the following sections. For more information refer to the IETF draft-cxx-ippm-ioamaggr [2].

5.1.1. IOAM Data Param

This 24-bit field identifies the data parameter that is to be aggregated across the nodes. As of today there is no data parameter defined by IANA for the IOAM Aggregation Option. In the scope of this project the value *0xFF* was selected to identify the efficiency indicator data parameter.

5.1.2. Aggregator

This 8-bit field specifies the aggregator to be used for aggregation. The IOAM Aggregation Option supports four different aggregators which are sum, minimum, maximum and average. Three of the four aggregators are implemented and used in this project to fulfill the functional requirements specified in 4.1 in the inception part.

SUM (0x1) Aggregates the HEI values to the PEI value (FR3).

MIN (0x2) Determines the node with the minimum HEI (most efficient) in a path (FR5).

MAX (0x4) Determines the node with the maximum HEI (most inefficient) in a path (FR4).

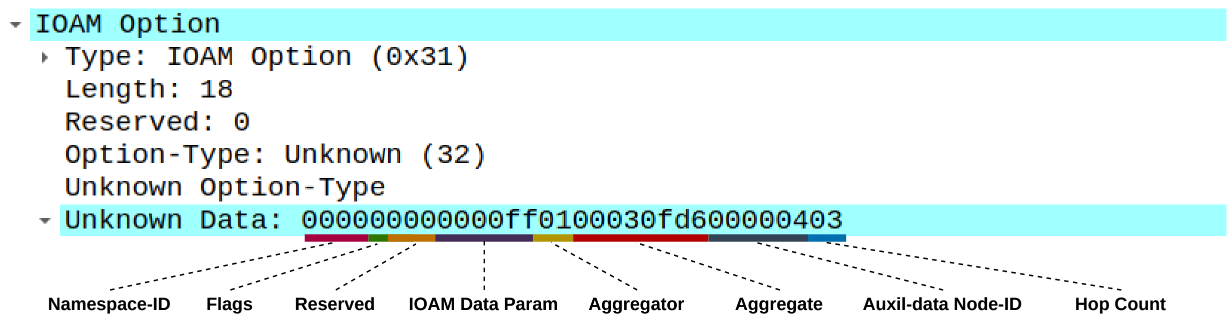
AVG (0x8) No use case in this project for this aggregator.

5.1.3. Aggregate

This 32-bit field contains the aggregated value. In case the SUM aggregator is used the aggregated value corresponds to the PEI otherwise to the minimum or maximum HEI on a path.

5.1.4. Wireshark

In the following Wireshark capture, the raw data of the previously described IOAM Aggregation Option header is visible. Wireshark does currently not support the decoding of the IOAM Aggregation Option header format as the IOAM Aggregation Option is not yet standardized and still in a draft state. Therefore a legend was added to identify the individual header fields inside the byte-stream.



```
IOAM Option
  Type: IOAM Option (0x31)
  Length: 18
  Reserved: 0
  Option-Type: Unknown (32)
  Unknown Option-Type
  Unknown Data: 000000000000ff0100030fd600000403
```

Figure 5.1.: Wireshark Capture of the IOAM Aggregation Option

5.2. Implementation in P4

The following sections will describe the individual P4 software components related to the IOAM Aggregation Option in more detail excluding the parser which is described in chapter 1 in the construction part.

5.2.1. Constants

The constants definition can be found in *efficiency-indicator-p4/includes/constants.p4*. The following values related to the IOAM Aggregation Option are defined as constants.

Listing 5.2: IOAM Aggregation Option Constants

```
const bit<8> IOAM_OPTION_TYPE_CHG = 0x31;
const bit<8> IOAM_AGGREGATION_OPTION_TYPE = 0x20;
const bit<8> IOAM_AGGREGATION_OPTION_DATA_LEN = 18;
const bit<8> IOAM_AGGREGATOR_SUM = 0x1;
const bit<8> IOAM_AGGREGATOR_MIN = 0x2;
const bit<8> IOAM_AGGREGATOR_MAX = 0x4;
const bit<8> IOAM_AGGREGATOR_AVG = 0x8;
```

IOAM_OPTION_TYPE_CHG This value is used in the IPv6 Hop-By-Hop extension header to identify the IOAM Option. This specific value specifies that the IOAM Option data is allowed to change in transit. The value is defined by IANA [7].

IOAM_AGGREGATION_OPTION_TYPE This value is used to identify the IOAM Aggregation Option type. The value is not yet defined by IANA so the unassigned value *0x20* was chosen for identification.

IOAM_AGGREGATION_OPTION_DATA_LEN This value defines the total length of the IOAM Aggregation Option. It includes IOAM Option identifying information and the IOAM Aggregation Option header. The value is the sum of the following headers and header fields.

$$\begin{aligned} IOAM\_AGGREGATION\_OPTION\_DATA\_LEN = {} & 1 \text{ Byte IOAM Opt-Type} \\ & + 1 \text{ Byte Reserved} \\ & + 16 \text{ Byte IOAM Aggregation Header} \end{aligned}$$

IOAM_AGGREGATOR_SUM This value specifies that the sum aggregation function is to be applied.

IOAM_AGGREGATOR_MIN This value specifies that the minimum aggregation function is to be applied.

IOAM_AGGREGATOR_MAX This value specifies that the maximum aggregation function is to be applied.

IOAM_AGGREGATOR_AVG This value specifies that the average aggregation function is to be applied.

5.2.2. Types

The types definition can be found in *efficiency-indicator-p4/includes/types.p4*. The following types related to the IOAM Aggregation Option are defined.

Listing 5.3: IOAM Aggregation Option Types

```
typedef bit<24> ioamNodeID_t;
typedef bit<8> ioamAggregator_t;
typedef bit<32> ioamAggregate_t;

struct ioamAggrMeta_t {
    ioamAggregate_t aggregate;
    ioamNodeID_t nodeID;
}
```

ioamNodeID_t Is a 24-bit unsigned number used to store the node identifier.

ioamAggregator_t Is a 8-bit unsigned number used to store the aggregator identifier.

ioamAggregate_t Is a 32-bit unsigned number used to store the aggregate value.

ioamAggrMeta_t Structured type with two fields which are relevant for IOAM Aggregation specific processing. An instance of this struct is instantiated as an attribute of the metadata struct. The values of that instance inside the metadata struct are initialized before the execution of IOAM Aggregation specific actions.

5.2.3. Headers

This section describes the definition of the IOAM Aggregation Option header. The header definition can be found in *efficiency-indicator-p4/includes/headers.p4*. The header fields are defined according to draft-cxx-ippm-ioamaggr [2].

The IOAM Aggregation Option header type is called *ioam_aggregation_t* and contains eight fixed sized header fields.

Listing 5.4: IOAM Aggregation Option Header

```
header ioam_aggregation_t {
    bit<16> namespaceID;
    bit<4> flags;
    bit<12> reserved;
    bit<24> dataParam; // identifies the type of data being aggregated
    ioamAggregator_t aggregator;
    ioamAggregate_t aggregate;
    ioamNodeID_t auxilDataNodeID;
    bit<8> hopCount;
}
```

5.2.4. Controller

The *process_ioam_aggregation* controller is responsible for IOAM Aggregation Option related processing. It takes the header structures initialized by the parser, custom metadata and standard metadata as input.

The controller definition can be found in *efficiency-indicator-p4/includes/ioam_tracing.p4*.

In the following section the controllers actions are described which give information about how header fields are updated in this state of the match action pipeline.

5.2.4.1. Actions

Actions are the place where the actual manipulation of header data occurs. The IOAM Aggregation Controller includes an action to initialize the IOAM Aggregation header data structure and actions to perform the already described aggregation operations.

ioam_aggr_push This action sets the IOAM option and the IOAM Aggregation Option headers to valid. All associated header fields are set to the initial value.

ioam_aggr_sum In this action, the current value stored in the aggregate field is aggregated with the HEI value stored in the metadata using the sum aggregation function. Additionally the auxil data node ID field is updated to the ID of the current node.

ioam_aggr_min In this action, the current value stored in the aggregate field is aggregated with the HEI value stored in the metadata using the min aggregation function. The aggregate value and the auxil data node ID fields are only updated in case the HEI value inside the metadata is smaller than the current aggregate value.

ioam_aggr_max In this action, the current value stored in the aggregate field is aggregated with the HEI value stored in the metadata using the max aggregation function. The aggregate value and the auxil data node ID fields are only updated in case the HEI value inside the metadata is larger than the current aggregate value.

get_node_id This action is used to initialize the node ID inside the metadata. It can be thought of a helper action that gets the required information from the control plane. In this case the node ID.

5.2.4.2. Tables

This section describes the tables relevant for the IOAM Aggregation Option. As described in section 3.3.1 in the elaboration part, the control plane tables are the linkage between the control plane and the data plane.

The IOAM Aggregation controller involves two tables which are defined in the runtime definition of each BMv2 software switch accordingly. For example for *s1* the definition can be found in *efficiency-indicator-p4/dev-network/s1-runtime.json*.

Two of the tables defined inside the runtime definition are directly related to the IOAM Aggregation Option which are further described below.

ioam_aggr_push_exact This table is used to lookup the aggregator value and to call the action *ioam_aggr_push* in case of a match.

node_id This table is used to lookup the node ID value and to call the action *get_node_id* in case of a match.

For more information about the table declaration syntax for the BMv2 model and table statements in P4 refer to the IOAM Pre-allocated Trace-Option table description in section 4.2.4.2 in the construction part.

5.2.4.3. Apply

The apply block in a P4 program is used to make decisions which table to apply or action to call based on conditions most likely given by header values or header validity state. In the apply block of the IOAM Aggregation Option first the action *node_id* is applied to initialize the metadata with the appropriate node ID. If the IOAM Aggregation Option header is already initialized the aggregation functions are called based on the aggregator specified inside the aggregator header field. In case the IOAM Aggregation Option header is not initialized (most likely on the ingress node) the table *ioam_aggr_push_exact* is applied which will initialize the IOAM Aggregation Option header.

Listing 5.5: IOAM Aggregation Option Apply Block

```
apply {
  node_id.apply();
  if (hdr.ioam_a_ioam_aggregation.isValid()) {
    switch (hdr.ioam_a_ioam_aggregation.aggregator) {
      IOAM_AGGREGATOR_SUM: {ioam_aggr_sum();}
      IOAM_AGGREGATOR_MIN: {ioam_aggr_min();}
      IOAM_AGGREGATOR_MAX: {ioam_aggr_max();}
    }
  } else {
    ioam_aggr_push_exact.apply();
  }
}
```

5.3. Limitations

- This implementation of the IOAM Aggregation Option does not fully comply with draft-cxx-ippm-ioamaggr as the average aggregator is currently not implemented [2].

6. Efficiency Indicators

6.1. Implementation in P4

The following sections will describe the individual P4 software components related to the calculation of the HEI and PEI value.

6.1.1. Constants

The constants definition can be found in *efficiency-indicator-p4/includes/constants.p4*. The following values related to the Efficiency Indicators are defined as constants.

Listing 6.1: Efficiency Indicator Constants

```
 const bit<15> MAX_VALUE_15_BIT = 32767;
```

MAX_VALUE_15_BIT This value is used during calculation. This specific value specifies the maximum value of a 15 bit size variable.

6.1.2. Types

The types definition can be found in *efficiency-indicator-p4/includes/types.p4*. The following types related to the Efficiency Indicators are defined.

Listing 6.2: Efficiency Indicator Types

```
typedef bit<1> inverse_t;
typedef bit<2> weight_t;
typedef bit<32> parameter_t;
typedef bit<5> parameterSize_t;
typedef bit<16> component_t;
typedef bit<15> normValue_t;
```

inverse_t Is a 1-bit unsigned number used to store the inverse parameter, `0 = false, 1 = true`.

weight_t Is a 2-bit unsigned number used to store the weight parameter.

parameter_t Is a 32-bit unsigned number used to store the parameter value.

parameterSize_t Is a 5-bit unsigned number used to store the type size of the parameter.

component_t Is a 16-bit unsigned number used to store the weighted normalized value.

normValue_t Is a 15-bit unsigned number used to store the normalized value.

6.1.3. Controller

The *process_efficiency_indicator* controller is responsible for Efficiency Indicators related processing. It takes the header structures initialized by the parser, custom metadata and standard metadata as input.

The controller definition can be found in *efficiency-indicator-p4/includes/efficiency_indicator.p4*.

In the following section the controllers actions are described which give information about how header fields are updated in this state of the match action pipeline.

6.1.3.1. Actions

Actions are the place where the actual manipulation of header data occurs. The Efficiency Indicator Controller includes an actions to get the needed parameter values to calculate the HEI and PEI from the control plane table and actions to perform the calculation.

calc_carbon_metric_parameter This action calculates the component value and adds it to the HEI value.

get_carbon_metric_energy_mix This action is called from the *carbon_metric_component_energy_mix* table and get the parameter values from the control plane table. After receiving the parameter the function calls the *calc_carbon_metric_parameter* action to calculate the energy mix component and add this component value to the HEI.

get_carbon_metric_idle_power This action is called from the *carbon_metric_component_idle_power* table and get the parameter values from the control plane table. After receiving the parameter the function calls the *calc_carbon_metric_parameter* action to calculate the idle power component and add this component value to the HEI.

get_carbon_metric_embedded_carbon This action is called from the *carbon_metric_component_embedded_carbon* table and get the parameter values from the control plane table. After receiving the parameter the function calls the *calc_carbon_metric_parameter* action to calculate the embedded carbon component and add this component value to the HEI.

Listing 6.3: Action calc_carbon_metric_parameter

```
action calc_carbon_metric_parameter(parameter_t parameter, parameterSize_t
    parameterSize, weight_t weight, inverse_t inverse) {
  // Normalization
  normValue_t normValue = 0;
  if(parameterSize < normValue_t.minSizeInBits()) {
    normValue = (normValue_t)parameter << (normValue_t.minSizeInBits() -
    parameterSize);
  } else if(parameterSize > normValue_t.minSizeInBits()){
    parameter_t tempNormValue = parameter >> (parameterSize -
    normValue_t.minSizeInBits());
    normValue = (normValue_t)tempNormValue;
  }

  // Inversion
  if(inverse == 1) {
    normValue = MAX_VALUE_15_BIT - normValue;
  }

  // Weighting normValue
  component_t component = 0;
  if(weight == 2) {
    component = (component_t)normValue << 1;
  } else if(weight == 1) {
    component = (component_t)normValue;
  } else if(weight == 0) {
    component = (component_t)normValue >> 1;
  }

  // Add component to HEI value
  meta.ioamAggrMeta.aggregate = meta.ioamAggrMeta.aggregate +
    (ioamAggregate_t)component;
    }
```

6.1.3.2. Tables

This section describes the relevant tables for calculating the efficiency indicators. As described in section 3.3.1 in the elaboration part, the control plane tables are the linkage between the control plane and the data plane.

The Efficiency Indicator Controller involves three tables which are defined in the runtime definition of each BMv2 software switch accordingly. For example for *s1* the definition can be found in *efficiency-indicator-p4/dev-network/s1-runtime.json.*

carbon_metric_component_energy_mix This table is used to lookup the value, weight and inverse parameter of the energy mix and to call the action *get_carbon_metric_energy_mix* in case of a match.

carbon_metric_component_idle_power This table is used to lookup the value, weight and inverse parameter of the idle power and to call the action *get_carbon_metric_idle_power* in case of a match.

carbon_metric_component_embedded_carbon This table is used to lookup the value, weight and inverse parameter of the embedded carbon and to call the action *get_carbon_metric_embedded_carbon*

The following table is used for the parameter lookup of the energy mix parameter. In each parameter table are the three parameters defined:

value The value is the specific parameter value of this node.

weight The weight defines how important this parameter is for the calculation of the HEI and PEI. The weight for a parameter type is on all nodes the same.

inverse The inverse parameter defines if it's needed to inverse the value during calculation. The reason for that is specified in chapter 2 in the elaboration part.

Listing 6.4: Energy Mix Table Definition

```
{
  "table": "MyEgress.process_efficiency_indicator.carbon_metric_component_energy_mix",
  "match": {
  "hdr.ethernet.srcAddr": [
    "08:00:00:00:01:00"
  ]
  },
  "action_name": "MyEgress.process_efficiency_indicator.get_carbon_metric_energy_mix",
  "action_params": {
  "value": 10,
  "weight": 2,
  "inverse": 1
  }
}
```

For more information about the table declaration syntax for the BMv2 model and table statements in P4 refer to the IOAM Pre-allocated Trace-Option table description in section 4.2.4.2 in the construction part.

6.1.3.3. Apply

The apply block in a P4 program is used to make decisions which table to apply or action to call based on conditions most likely given by header values or header validity state.

Listing 6.5: Efficiency Indicator Apply Block

```
apply {
    // IOAM Carbon Metric Aggregation
    carbon_metric_component_energy_mix.apply();
    carbon_metric_component_idle_power.apply();
    carbon_metric_component_embedded_carbon.apply();
}
```

6.1.4. IOAM Aggregation

After the *process_efficiency_indicator* controller is processed and the HEI was calculated the following apply block of the *process_ioam_aggregation* controller is called, which will handle what further happens with the HEI value. The IOAM Aggregation Option is described in detail in chapter 5 in the construction part.

6.2. Add new Component

The following steps describe how to implement a new component that is added to the calculation of HEI. The example is made with the component named Energy Mix.

Step 1 Add a new control plane definition for the new component in all *sx-runtime.json*. For example for *s1* the definition can be found in *efficiency-indicator-p4/dev-network/s1-runtime.json*. This entry defines the parameter value, weight, and whether it should be inverted.

Listing 6.6: Energy Mix Table Definition

```
{
  "table":
  "MyEgress.process_efficiency_indicator.carbon_metric_component_energy_mix",
  "match": {
  "hdr.ethernet.srcAddr": [
    "08:00:00:00:01:00"
  ]
  },
  "action_name":
  "MyEgress.process_efficiency_indicator.get_carbon_metric_energy_mix",
  "action_params": {
  "value": 10,
  "weight": 2,
  "inverse": 1
  }
}
```

Step 2 A new type for the component must be initialized. The size of the type must be as small as possible as described in 2.10 in the elaboration part.

Listing 6.7: Energy Mix Table Definition

```
typedef bit<7> energyMix_t;
```

Step 3 Create a new Action for the component in the file *efficiency-indicator-p4/includes/efficiency_indicator.p4*. This action gets the parameter from the control table look up and will call the *calc_carbon_metric_parameter* function to calculate the component value and add them to the HEI.

Listing 6.8: Action to get the Energy Mix parameters

```
action get_carbon_metric_energy_mix(energyMix_t value, weight_t weight,
  inverse_t inverse) {
      calc_carbon_metric_parameter((parameter_t)value, value.minSizeInBits(),
  weight, inverse);
  }
```

Step 4 Add a new table that is used to lookup the value, weight and inverse parameter of the energy mix and to call the action *get_carbon_metric_energy_mix* in case of a match.

Listing 6.9: Energy Mix Table

```
table carbon_metric_component_energy_mix {
      key = {
          hdr.ethernet.srcAddr: exact;
      }
      actions = {
          get_carbon_metric_energy_mix;
          drop;
          NoAction;
      }
      size = 1;
      default_action = drop();
  }
```

Step 5 Add the following line of code to the apply block to retrieve the new component data from the control plane and to use it in the HEI calculation.

Listing 6.10: Add the Table to the apply block

```
carbon_metric_component_energy_mix.apply();
```

7. Testing

This chapter describes how the elaborated testing approach described in chapter 5 in the elaboration part is implemented.

The implementation is an extension of the already existing P4 language tutorial environment available on GitHub, written in the Python programming language by p4lang. The extension of the tutorial environment includes the following challenges which had to be overcome during the development phase:

- Define a structured format to specify test cases containing all the relevant information for both the sending and receiving component.
- Reprogram the BMv2 targets at runtime based on patches defined for the individual test case.
- Synchronize the sending and receiving component.
- Send data through the network based on the test case specifications.
- Validate the received data on correctness considering the test case specific condition of the control plane of the BMv2 targets.
- Display the results to the developer executing the automated tests.

The following sections go into further detail on how these challenges where mastered.

7.1. Software Components

Before diving into any implementation detail the role and purpose of all related software components must be clear. All relevant components are shown in the context diagram in figure 7.1 and are related to each other.

Test Sender Main component to trigger the bootstrapping and reconfiguration of the test environment, send test traffic and to start the needed test listener processes on the specific Mininet endpoints.

Test Receiver Validate the IOAM header fields of the received test traffic for correctness.

P4 Tutorial Utilities Python application developed by p4lang available on GitHub. Called by the *Test Sender* to provision and reprogram the test network. For the reconfiguration of the BMv2 targets control plane, the P4 runtime shell is used and the connection to the targets is done via gRPC. Refer to the p4runtime-shell project for more information.

Test Network (BMv2 Targets) Corresponds to the development environment described in chapter 4 in the elaboration part.

Runtime Definitions JSON definition of the control plane of the BMv2 targets.

Test Specific Runtime Definitions Copies of the patched runtime definitions specific for a test case.

Test Case Definition JSON definition of the test cases further described in section 7.2 in the construction part.

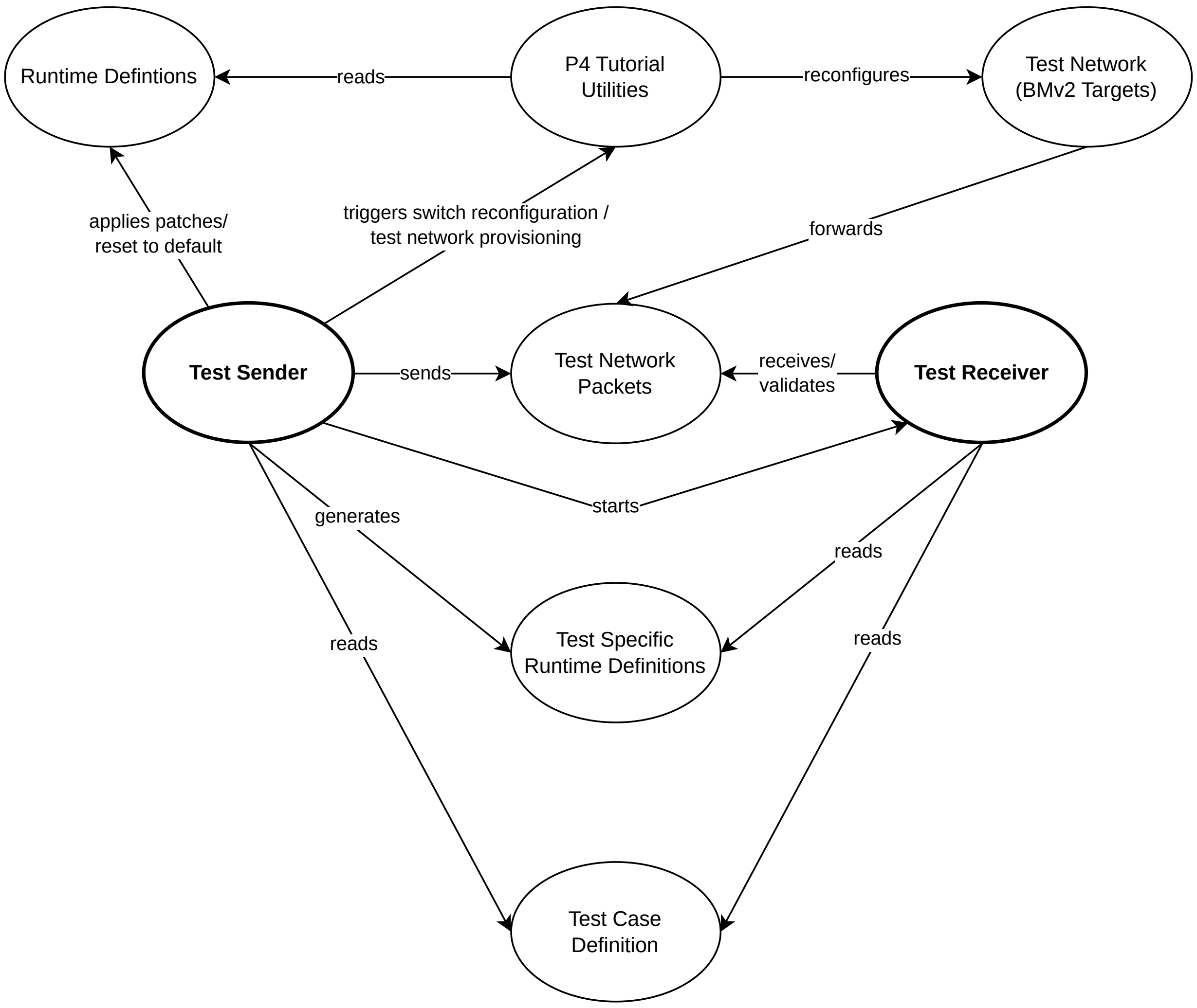


Figure 7.1.: Testing Context Diagram

7.1.1. Component Interaction Flow

Figure 7.1 shows the relation between components but gives no information about the flow of interactions between the actual components. The enumeration below describes the component interactions and the resulting test procedure.

1. The *Test Sender* triggers the provisioning of the *Test Network* by calling the corresponding *P4 Tutorial Utilities*.
2. The *Test Sender* reads the *Test Case Definitions*.
3. The *Test Sender* starts a *Test Receiver* process on every endpoint which is specified as the destination of test traffic at least once inside the *Test Case Definition*.
4. Each *Test Receiver* started by the *Test Sender* reads the *Test Case Definitions* and determines the test cases which are relevant for the specific *Test Receiver*. Relevant test cases are those, in which the *Test Receiver* is specified as destination.

5. For each test case specified inside the *Test Case Definitions* the *Test Sender*:
    - Applies patches to the *Runtime Definitions*, in case parameter patches are specified in the test case definition.
    - Generates the *Test Specific Runtime Definitions* by copying the patched *Runtime Definitions*, in case parameter patches are specified in the test case definition.
    - Triggers a reconfiguration of the *Test Network* by calling the corresponding *P4 Tutorial Utilities*, in case parameter patches are specified in the test case definition.
    - Sends the specified number of *Test Network Packets* to the *Test Receiver* specified in the *Test Case Definition*.
    - Resets the *Runtime Definitions* to default.
    - Triggers a reconfiguration of the *Test Network* by calling the corresponding *P4 Tutorial Utilities*, in case parameter patches are specified in the test case definition.
6. For each relevant test case specified inside the *Test Case Definitions* the *Test Receiver*:
    - Listens for the packets of the current test case.
    - Calculates the expected IOAM header values based on the control plane table values specified in the *Test Specific Runtime Definition*.
    - Validates the IOAM header fields for correctness after reception. The result of each validation step is logged to a *Test Receiver* specific log file.
7. The *Test Sender* shuts down the *Test Network* by calling the corresponding *P4 Tutorial Utilities*.

7.1.2. Source Code

The source code is located inside the *efficiency-indicator-p4* repository as part of the *dev-network* resources. The directory structure is depicted below.

```
dev-network/
├── test/
│   └── cases.json
├── tmp_runtimes/
│   └── _omitted_
├── utils/
│   ├── captures/
│   │   ├── quic_ipv4.pcapng
│   │   └── quic_ipv6.pcapng
│   ├── mininet/
│   │   └── _omitted_
│   ├── p4_runtime_lib/
│   │   └── _omitted_
│   ├── testing/
│   │   ├── calc.py
│   │   ├── send.py
│   │   └── utils.py
│   ├── run_test_receiver.py
│   └── run_test_sender.py
└── s1-runtime.py
```

```
│
├── _omitted_
├── s6-runtime.py
└── topology.json
```

The following description gives information about the most important files listed in the directory structure above and how they are related to the individual software components and interactions.

test/cases.json This file holds the *Test Case Definition*. It is read by both the *Test Sender* and *Test Receiver*. The definition of the test cases is further described in section 7.2 in the construction part.

tmp_runtimes/ This directory contains the *Test Specific Runtime Definitions* generated by the *Test Sender* and read by the *Test Receiver*.

utils/ This directory contains all related Python utilities and resources.

utils/captures/ This directory contains wireshark captures which contain template packets, which are modified and sent by the *Test Sender*.

utils/mininet/ Mininet specific Python package and part of the *P4 Tutorial Utilities*.

utils/p4_runtime_lib P4 and BMv2 specific Python package and part of the *P4 Tutorial Utilities*.

utils/testing Python package containing helper functions used by the *Test Sender* and *Test Receiver*.

utils/testing/run_test_sender.py Implementation of the *Test Sender* and caller of specific *P4 Tutorial Utilities*.

utils/testing/run_test_receiver.py Implementation of the *Test Receiver*.

7.1.3. Scapy Library

Scapy is a powerful interactive packet manipulation library written in Python. Scapy is able to forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies, and much more. [10] For more information about Scapy refer to the official documentation: https://scapy.net/

- Scapy is used by the *Test Sender* to forge and send packets over the network. The *Test Sender* reads the capture containing real user traffic from *utils/capture*, modifies the source and destination MAC and IP addresses and emits the packets into the *Test Network*.
- Scapy is used by the *Test Receiver* to decode the packets which where sent via the *Test Network*. Before the IOAM header data validation occurs a dummy packet is forged and initialized with the expected values. The actual packet and the dummy packet are then compared for equality.

7.2. Definition of Tests

In this section the *Test Case Definition* is examined. According to figure 7.1 the *Test Case Definition* is read by both the *Test Sender* and *Test Receiver*.

The tests are defined in JSON format. The file follows the structure below which is defined as a list of paths followed by a list of testcases. Details about the path and testcase objects are documented in the following two sections.

Listing 7.1: Tests definition

```
{
  "paths": [
    // path1,
    // path2,
    // ...
  ],
  "testcases": [
    // testcase1,
    // testcase2,
    // ...
  ]
}
```

7.2.1. Path Object

A path object contains all path relevant information. The information is used by the *Test Sender* to determine the correct source and destination MAC and IP addresses. The *Test Receiver* reads the nodes list to be able to perform IOAM Aggregation and IOAM Pre-allocated Trace-Option calculations.

The following JSON snippet is a valid path object definition. The snippet defines the path from *h1* to *h2* which passes via the switches *s1-s2-s4*.

Listing 7.2: Path object definition

```
{
    "id": 1,
    "src": "h1",
    "dst": "h2",
    "nodes": [
        {
            "name": "s1",
            "id": 1
        },
        {
            "name": "s2",
            "id": 2
        },
        {
            "name": "s4",
            "id": 4
        }
    ]
}
```

The following description gives more information about the individual object properties.

id Identification number of the path

src Name of the source host as defined in the Mininet topology (*Test Sender*)

dst Name of the sending host as defined in the Mininet topology (*Test Receiver*)

nodes List of nodes

nodes / name Name of the node as defined in the Mininet topology

nodes / id Identification of the node as defined in the runtime definition (corresponds to the IOAM node ID)

7.2.2. Testcase Object

A testcase object contains all testcase relevant information. The information is used by the *Test Sender* to reconfigure the *Test Network* and send the appropriate number of packets to the destination of the specified path. The information is used by the *Test Receiver* to determine the relevant testcases, number of relevant packets, *Test Specific Runtime Definition* (based on the id of the test) and to calculate the IOAM Aggregation and IOAM Pre-allocated Trace-Option based on the defined path for this testcase.

Information

The id field must not be defined manually. During the first execution of a testcase an ID is automatically defined and set inside the testcase definition.

The following JSON snippet is a valid testcase object definition. The snippet defines a testcase to:

- send 1 packet from the *Test Sender* via *path 1* without the application of parameter patches to the BMv2 targets in the *Test Network*
- validate the packets on the *Test Receiver* whether the IOAM Aggregation was performed correctly using the sum aggregator
- validate the packets on the *Test Receiver* whether the IOAM Tracing corresponds to the trace via *path 1*

Listing 7.3: Testcase object definition

```
{
    "name": "Test with default control plane on path H1 to H2",
    "path": 1,
    "protocol": "ipv6",
    "num_packets": 1,
    "ioam_aggregation": {
        "aggregator": "sum"
    },
    "id": "4be8d9aa-8c75-11ee-99de-0800270cf606"
}
```

The testcase definition above in listing 7.3 does not cover the requirement of having the possibility to flexibly define the control plane of BMv2 targets specifically for a testcase. The testcase definition below in listing 7.4 introduces the specification of parameter patches. With the provided specification syntax one can set any parameter value of any action in any table on any switch by desire.

The following JSON snippet is a valid testcase object definition. The snippet defines a testcase to:

- send 5 packets from the *Test Sender* via *path 2*
- set the *weight* parameter value of the *get_carbon_metric_energy_mix* action in the *carbon_metric_energy_mix* table to 1 on *s1, s2 and s4*.

- validate the packets on the *Test Receiver* whether the IOAM Aggregation was performed correctly using the sum aggregator
- validate the packets on the *Test Receiver* whether the IOAM Tracing corresponds to the trace via *path 2*

Listing 7.4: Testcase object definition with parameter patches

```
{
    "name": "Test with neutral weight for energy mix parameter on path H1 to H3",
    "path": 2,
    "protocol": "ipv6",
    "num_packets": 5,
    "ioam_aggregation": {
        "aggregator": "sum",
        "parameter_patches": [
            {
                "switches": [
                    "s1",
                    "s3",
                    "s4"
                ],
                "table":
    "MyEgress.process_efficiency_indicator.carbon_metric_component_energy_mix",
                "action":
    "MyEgress.process_efficiency_indicator.get_carbon_metric_energy_mix",
                "parameters": [
                    {
                        "weight": 1
                    }
                ]
            },
        ]
    },
}
```

name Name of the testcase (only descriptive no technical use)

path Identification number of the path used in this test case

protocol Protocol to use (only ipv6 is implemented)

num_packets Number of packets to send

ioam_aggregation IOAM Aggregation Option specific configuration parameters

ioam_aggregation/aggregator IOAM Aggregation Option aggregator (sum/min/max) to be used for validation only

ioam_aggregation/parameter_patches List of patches to apply for this testcase

ioam_aggregation/parameter_patches/switches List of switches to apply the patch on

ioam_aggregation/parameter_patches/table The table to apply the patch on

ioam_aggregation/parameter_patches/action The action to apply the patch on

ioam_aggregation/parameter_patches/parameters List of parameters (arbitrary key value objects) to be set by the patch

id Identification number of the testcase (automatically generated UUID)

Information

Setting the aggregator option has no impact to the control plane tables of the BMv2 targets. In case a test is set to validate the min or max aggregator, the parameter_patches to reconfigure the control plane of the BMv2 targets to use the specific aggregator, must be specified explicitely.

7.3. Test Execution

As stated in the requirements the test execution should be fully automated. To execute the specified tests a developer has to enter the command `make test`. This will trigger the job called *test*, specified in the Makefile, to run.

7.3.1. Makefile

The make utility is used to automate the necessary steps for the test execution. The required action to execute the tests are:

1. Build the P4 project
2. Run the test utility
3. Reset file permission

Listing 7.5: Test job definition

```
test: build
  sudo bash ./run_tests.sh
  sudo chown -R ${USER}:${USER} ./
```

The *test* job defined in listing 7.5 is executed after the build job has completed. The test utilities are started by running the *run_test.sh* bash script. The script initializes required variables, starts the *run_test_sender.py* Python utility with the required arguments. Additionally a watch screen is displayed which shows the output of the command below updated every second.

Listing 7.6: Watch command console output

```
watch -n 1 "grep -r -E 'TEST (FAILED|PASSED|RUN)' "${log_dir}" | awk -F 'log:' '{print
    \$2}' | sort"
```

grep Filter for the given pattern in the log directory of the current test run

awk Returns the second row of the grep output delimited at *log:* which removes the preceding file name added by grep

sort Sorts the output alphabetically which leads to a chronological ordering of the events because each event begins with the timestamp

After the completion of the test the file ownership is reset to the user named boss.

7.3.2. Results

All test results are logged in the log directory of the current test run. Each host has its own log file.

7.3.2.1. Console Output

The console output is generated by the `watch` command described in the previous section. The output lists the test results. For illustrative purpose the P4 program was modified, which caused the last test with the ID *1c6a122-8d01-11ee-99de-0800270cf606* to fail.

Listing 7.7: Console outpuut of a sample test run

```
2023-12-14 20:34:22,860:INFO:TEST RUN STARTED (please wait this might take a few
    seconds)
2023-12-14 20:34:31,415:INFO:TEST PASSED (id: 4be8d9aa-8c75-11ee-99de-0800270cf606)
2023-12-14 20:34:33,962:INFO:TEST PASSED (id: 4be8d9ab-8c75-11ee-99de-0800270cf606)
2023-12-14 20:34:36,573:INFO:TEST PASSED (id: 4be8d9ac-8c75-11ee-99de-0800270cf606)
2023-12-14 20:34:39,314:INFO:TEST PASSED (id: 4be8d9ad-8c75-11ee-99de-0800270cf606)
2023-12-14 20:34:42,156:INFO:TEST PASSED (id: 604e1b4e-95d5-11ee-85ca-0800270cf606)
2023-12-14 20:34:44,966:INFO:TEST PASSED (id: 77ddef58-8cfe-11ee-99de-0800270cf606)
2023-12-14 20:34:47,798:ERROR:TEST FAILED (id: a1c6a122-8d01-11ee-99de-0800270cf606)
2023-12-14 20:34:51,005:INFO:TEST RUN COMPLETED (detailed logs about the results are
    located at: /home/boss/git/efficiency-indicator-p4/logs/testing/20231214203422)
```

One can now search in the log directory given in the last line for either *ERROR* events or the ID of the failed test.

7.3.2.2. Logs

Each host involved, has its own log file located in a subdirectory with the current timestamp. The log directory follows the structure below.

```
efficiency-indicator-p4/
└─ logs/
   └─ testing/
      ├─ 20231209105915/
      │  ├─ h1-test-sender.log
      │  ├─ h2-test-receiver.log
      │  ├─ h3-test-receiver.log
      │  └─ h4-test-receiver.log
      └─ 20231214203422/
         ├─ h1-test-sender.log
         ├─ h2-test-receiver.log
         ├─ h3-test-receiver.log
         └─ h4-test-receiver.log
```

To search for *ERROR* messages of the test run attempted at *20231214203422* one would enter the following command.

Listing 7.8: Search for errors

```
grep -r "ERROR" /home/boss/git/efficiency-indicator-p4/logs/testing/20231214203422
```

```
2023-12-14 20:34:47,798:ERROR:Test failed, packet 1/1 field invalid
    (auxil_data_node_id: expected = 4 / received = 5) (id:
    a1c6a122-8d01-11ee-99de-0800270cf606)
2023-12-14 20:34:47,798:ERROR:TEST FAILED (id: a1c6a122-8d01-11ee-99de-0800270cf606)
```

The log message shows that an error occurred validating the *auxil_data_node_id* field.

To filter for all messages regarding the failed test with the ID *a1c6a122-8d01-11ee-99de-0800270cf606* the same command could be used replacing *ERROR* with the ID of the test to be filtered for.

8. Demo Application

The demo application is based on the network topology as described in the chapter 4 in the elaboration part. The following paths are used in the demo application and will be further named Path 1 and Path 2.

Path 1 H1 - S1 - S2 - S4 - H2

Path 2 H1 - S1 - S3 - S4 - H3

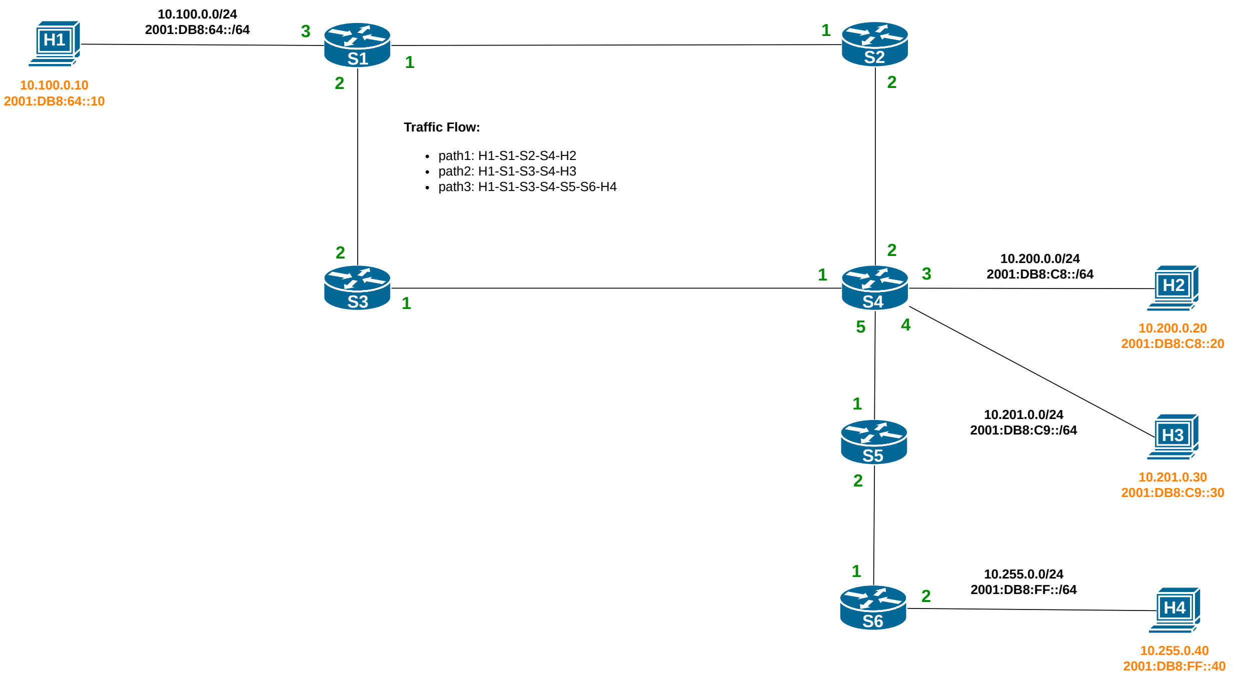


Figure 8.1.: Development Network Topology

8.1. Demo Execution

The demo application is based on the testing framework described in chapter 7 in the construction part. The make utility automates the necessary steps to execute the demo, as described in chapter 4 in the elaboration part. The main difference to the testing is that not the same `Test Case Definitions` are used for the demo application.

test/cases.json This file holds the *Test Case Definition*. It is read by both the *Test Sender* and *Test Receiver*. The definition of the test cases is further described in section 7.2 in the construction part.

demo/cases.json This file holds the *Demo Case Definition*. It is read by both the *Test Sender* and *Test Receiver*. The definition of the demo cases is further described in section 7.2 in the construction part and also applies to the demo cases.

Additionally, in demo cases, it is possible to add a stop attribute that will pause the demo until the `demo executor` hits enter.

8.2. Data Export

Each receiving host exports the received packets into the following data structures and stores them in a json file named after the host name. The file for the host 2 with the exported data is stored for example in the following directory *dev-network/demo/h2.json*

Listing 8.1: Data structure for export data

```json
{
    "test_id": "d15b0dd2-9b80-11ee-99de-0800270cf606",
    "timestamp": 1702741414627986932,
    "size": 1455,
    "ethernet": {
        "src": "08:00:00:00:04:00",
        "dst": "08:00:00:20:00:20"
    },
    "ipv6": {
        "src": "2001:db8:64::10",
        "dst": "2001:db8:c8::20",
        "hop_limit": 57
    },
    "hop_by_hop_option": {
        "ioam_option_tracing": {
        "type": 49,
        "length": 26,
        "reserved": 0,
        "option_type": 0,
        "namespace_id": 0,
        "node_length": 1,
        "flags": 0,
        "remaining_length": 1,
        "trace_type": 8388608,
        "ioam_reserved": 0,
        "node_list": [
            {
            "hop_limit": 0,
            "node_id": 0
            },
            {
            "hop_limit": 59,
            "node_id": 1
            },
            {
            "hop_limit": 58,
            "node_id": 2
            },
            {
            "hop_limit": 57,
            "node_id": 4
            }
        ]
        },
        "ioam_option_aggregation": {
        "type": 49,
        "length": 18,
        "reserved": 0,
        "option_type": 32,
        "namespace_id": 0,
        "flags": 0,
        "ioam_reserved": 0,
        "ioam_data_param": 255,
        "aggregator": 1,
        "aggregate": 196162,
        "auxil_data_node_id": 4,
        "hop_count": 3
        }
    }
    }
```

8.3. Demo Cases

This section describes all demo scenarios which are implemented in the demo application. All demo scenarios are defined in the section 6.2 in the elaboration part.

8.3.1. Timeseries per Path

In the first demonstration the default runtime configuration is initially loaded and five packets will be sent over path 1. For each demo case listed below, five packets will be sent with a different energy mix parameter value. After all packets are sent via path 1 the same amount of packets with the same energy mix values are sent over path 2. Both diagrams display individual packets and their corresponding PEI values as dots.

Path 1 Simulated day where the sustainable energy generated by solar power fluctuates

- **Packets 0 to 4** Demo with the default energy mix value on each node.
- **Packets 5 to 9** Demo with energy mix value in the morning, set value to `30` on all nodes in path 1.
- **Packets 10 to 14** Demo with energy mix value at lunch time, set value to `90` on all nodes in path 1.
- **Packets 15 to 19** Demo with energy mix value in the afternoon, set value to `65` on all nodes in path 1.
- **Packets 20 to 24** Demo with energy mix value in the evening, set value to `15` on all nodes in path 1.

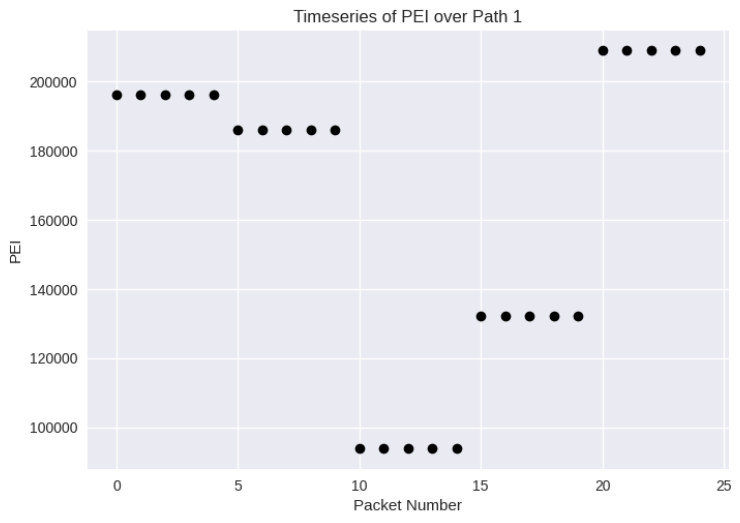


Figure 8.2.: Diagram about changing the energy mix value on all nodes in path 1

Path 2 Simulated day where the sustainable energy generated by solar power fluctuates

Packets 0 to 4 Demo with the default energy mix value on each node.

Packets 5 to 9 Demo with energy mix value in the morning, set value to 30 on all nodes in path 2.

Packets 10 to 14 Demo with energy mix value at lunch time, set value to 90 on all nodes in path 2.

Packets 15 to 19 Demo with energy mix value in the afternoon, set value to 65 on all nodes in path 2.

Packets 20 to 24 Demo with energy mix value in the evening, set value to 15 on all nodes in path 2.

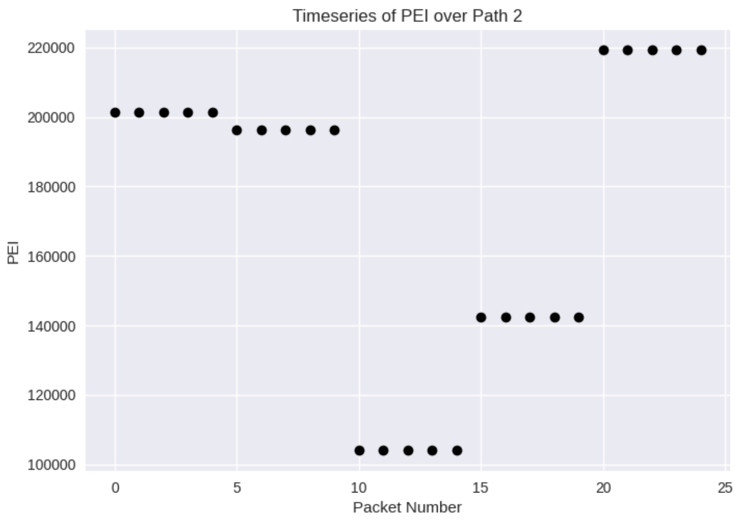


Figure 8.3.: Diagram about changing the energy mix value on all nodes in path 2

8.3.2. Path Comparison

In this example, it is important that the weights are set to the same value at all nodes, otherwise the result will be distorted. The graphic below shows that path 1 has a slightly lower PEI value than path 2, which indicates that packets sent over path 1 are using the more efficient path. The difference is fairly small because the energy mix values are all set to the same values on each node, but the idle power and embedded carbon parameter values vary.

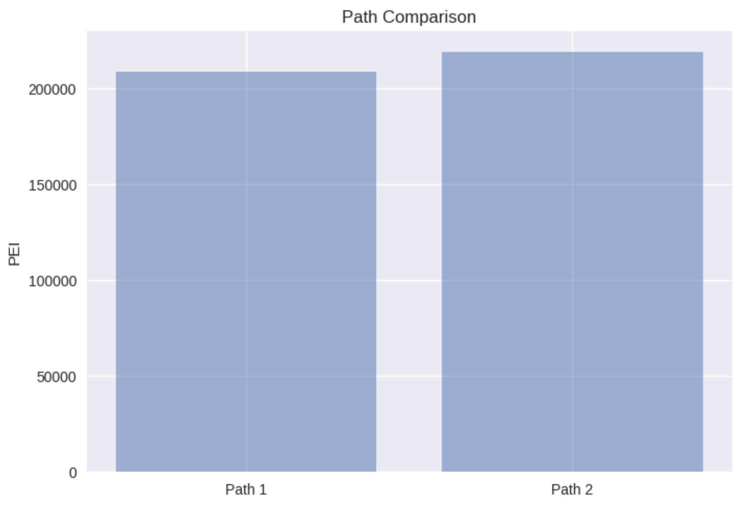


Figure 8.4.: Diagram about the Path Comparison

8.3.3. Minimum / Maximum HEI on a Path

The most efficient and most inefficient node on path 1 is visualized in the diagram below. For this demo case, packets were sent with the IOAM Aggregator set to `Min` and `Max` instead of the `Sum` aggregator. With the `Min` and `Max` aggregation option, it is possible to determine the best or worst node on a path and make further topology improvements by changing the routing path or replacing the node.

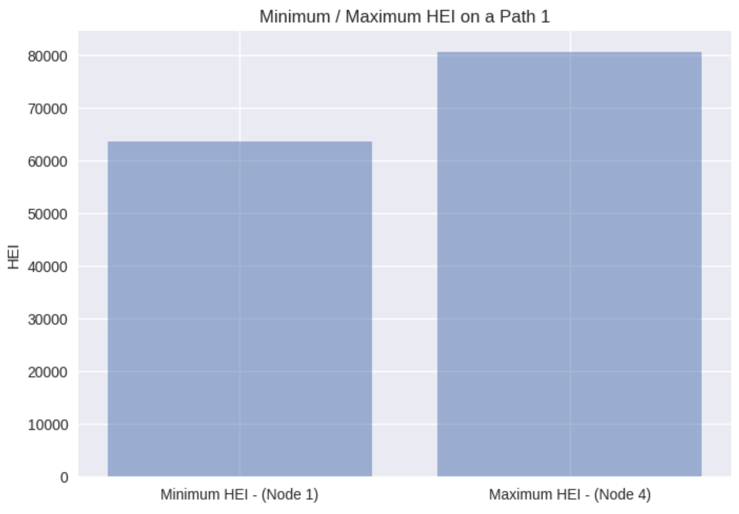


Figure 8.5.: Diagram of nodes with minimum and maximum HEI on path 1

8.3.4. PEI Composition

The PEI composition area chart shows how the individual components contribute to the resulting PEI on path 1. One can see that the two components *idle_power* and *embedded_carbon* are constant and that the *energy_mix* component varies over time. More details about this variation is given in the following description.

Information

The composition of the PEI cannot be read directly from the packet metadata. The calculations performed in the project network are simulated by the demo application and the results are displayed accordingly. The information about the invidual component and HEI values are usually lost in transit due to aggregation.

The first 25 packets in the chart represents the previously sent packets for the demo cases above. The changes on the PEI value on the last 15 packets (packet number 25 - 39) are caused by the following changes:

Packets 25 to 29 Demo with increased weight on the energy mix, set value to `50` and the weight to `2`.

Packets 30 to 34 Demo with neutral weight on the energy mix, set value to `50` and the weight to `1`.

Packets 35 to 39 Demo with negative weight on the energy mix, set value to `50` and the weight to `0`.

For each demo case listed above, five packets will be sent with a different weight on the energy mix parameter.

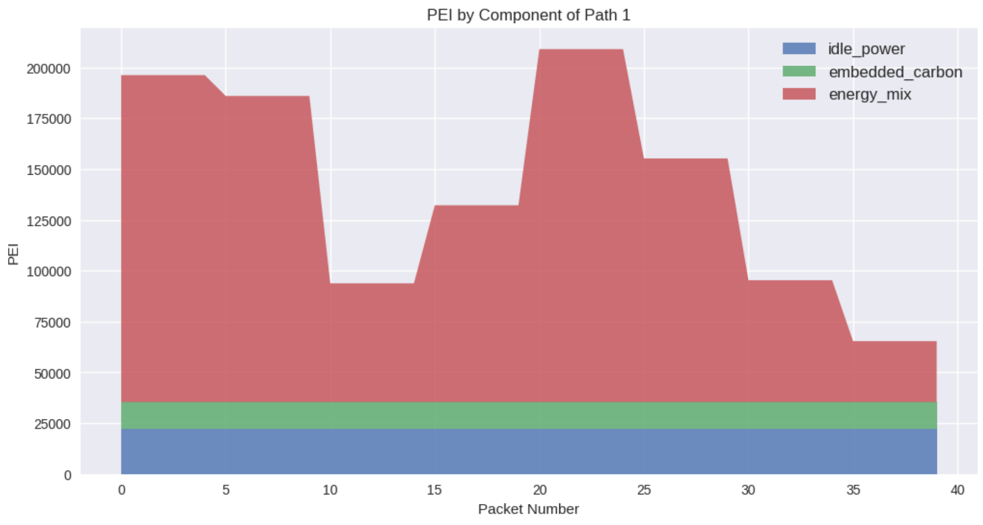


Figure 8.6.: Diagram about the PEI Composition

`Hint:` The weight influences the value of the energy mix component on each hop, that means the weight is applied once to each HEI which implies that the effect to the PEI for that specific component is directly dependent to the amount of hops on the specific path. This is why there is such a big difference in the PEI. The drop from packet 29 to 30 is caused by the energy mix component value on each HEI being 20'000 smaller than before. The drop from packet 34 to 35 is caused by the energy mix component value on each HEI being 10,000 lower than before, resulting in the drop from packet 29 to 30 being three times higher than from packet 34 to 35.

8.3.5. Path Statistics

The demo case about the path statistics which fulfills the definition of UC06 specified in chapter 3 in the inception part shows the following information about the paths:

path_id The identifier path id represents a specific path.

timestamp The timestamp shows the last information update.

node_list The node list contains all nodes of the path with their node id and the current hop limit of the packet.

num_packets The number of packets indicates the number of packets sent over this path.

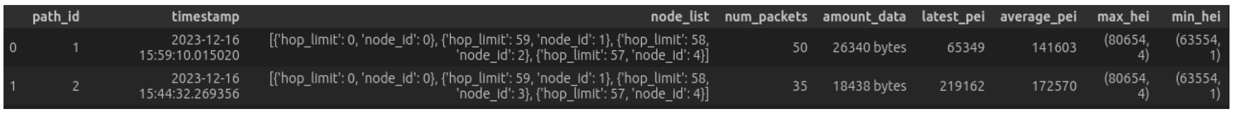
amount_data Shows how much data was sent over this path.

latest_pei Shows the latest PEI of this path.

average_pei Shows the average of the PEI over this path.

max_hei Indicates the node that is currently the most inefficient node in this path.

min_hei Indicates the node that is currently the most efficient node in this path.

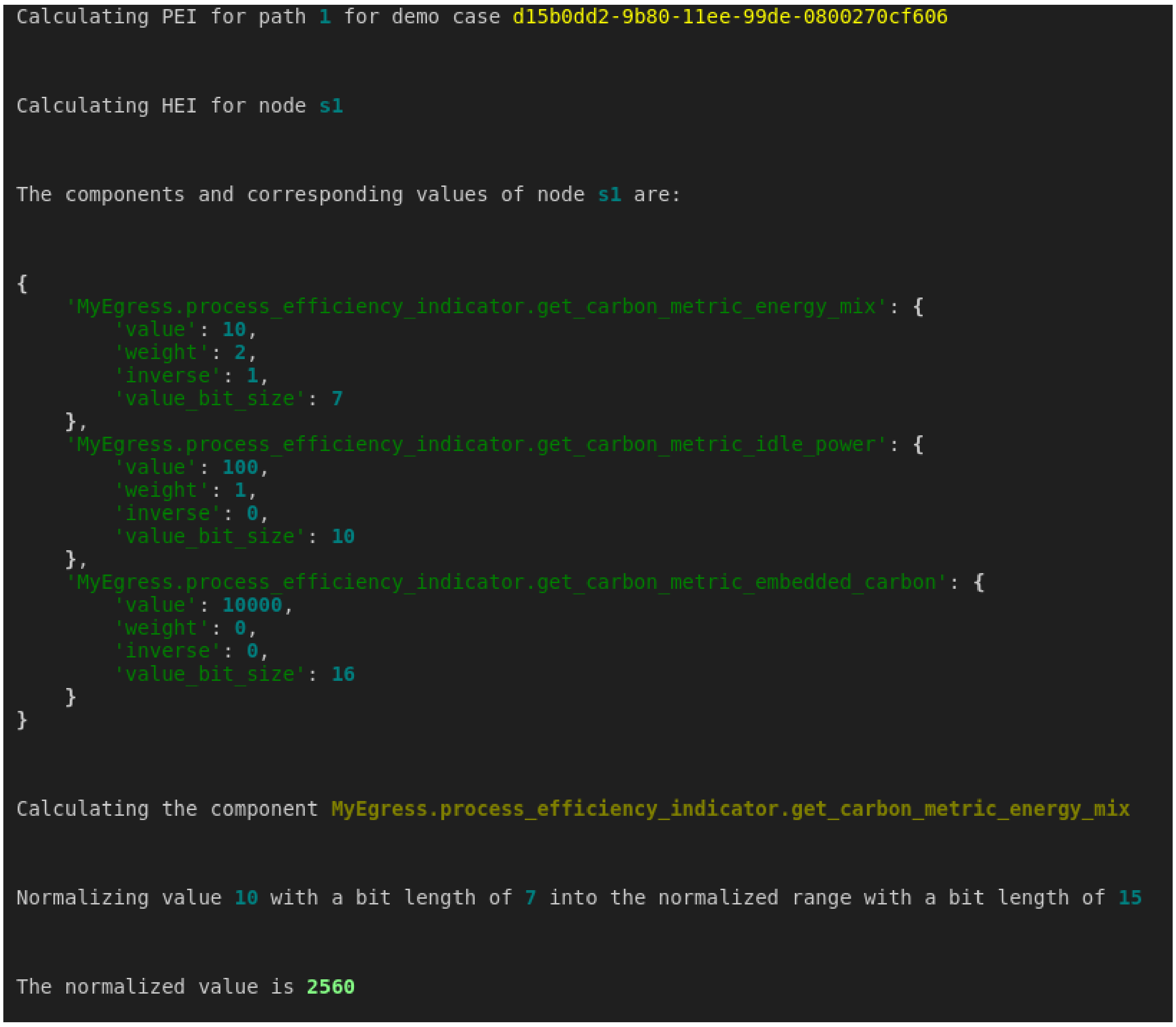


	path_id	timestamp	node_list	num_packets	amount_data	latest_pei	average_pei	max_hei	min_hei
0	1	2023-12-16 15:59:10.015020	[{'hop_limit': 0, 'node_id': 0}, {'hop_limit': 59, 'node_id': 1}, {'hop_limit': 58, 'node_id': 2}, {'hop_limit': 57, 'node_id': 4}]	50	26340 bytes	65349	141603	(80654, 4)	(63554, 1)
1	2	2023-12-16 15:44:32.269356	[{'hop_limit': 0, 'node_id': 0}, {'hop_limit': 59, 'node_id': 1}, {'hop_limit': 58, 'node_id': 3}, {'hop_limit': 57, 'node_id': 4}]	35	18438 bytes	219162	172570	(80654, 4)	(63554, 1)

Figure 8.7.: Diagram about the path statistics

8.3.6. Sample Step-by-Step PEI Calculation

The step-by-step PEI calculation is based on a previously calculated demo case. Each calculation step is listed and described, providing a clear and concise explanation of the resulting value. For more detailed information, please run the demo application.



```
Calculating PEI for path 1 for demo case d15b0dd2-9b80-11ee-99de-0800270cf606


Calculating HEI for node s1


The components and corresponding values of node s1 are:


{
    'MyEgress.process_efficiency_indicator.get_carbon_metric_energy_mix': {
        'value': 10,
        'weight': 2,
        'inverse': 1,
        'value_bit_size': 7
    },
    'MyEgress.process_efficiency_indicator.get_carbon_metric_idle_power': {
        'value': 100,
        'weight': 1,
        'inverse': 0,
        'value_bit_size': 10
    },
    'MyEgress.process_efficiency_indicator.get_carbon_metric_embedded_carbon': {
        'value': 10000,
        'weight': 0,
        'inverse': 0,
        'value_bit_size': 16
    }
}


Calculating the component MyEgress.process_efficiency_indicator.get_carbon_metric_energy_mix


Normalizing value 10 with a bit length of 7 into the normalized range with a bit length of 15


The normalized value is 2560
```

Figure 8.8.: Cutout of the step-by-step PEI calculation

Part IV.

Transition

1. Conclusion and Discussion

The primary goal of this research project was to propose a practical solution for aggregating, transporting, and interpreting green metrics within a simulated computer network. The study focused on developing efficiency indicators at both the hop and path levels, utilizing a robust mathematical framework to calculate these indicators. The findings, implications, and recommendations for future research are discussed below.

1.1. Review of Findings

The study successfully defined and implemented meaningful indicators at both the hop and path levels, contributing to the understanding of network efficiency. The Hop Efficiency Indicator (HEI) was introduced as a measure of each node's efficiency, with a flexible mathematical formula allowing for future adjustments. Importantly, the calculation algorithm for HEI maximized efficiency by avoiding division operations, ensuring optimal performance.

To compare different path efficiencies, a Path Efficiency Indicator (PEI) was proposed, summarizing the hop efficiency indicators of nodes along a path. The study explored alternatives for aggregation, considering the trade-off between minimizing additional data in packet headers and preserving information about individual hops. The implementation also allows the discovery of minimum and maximum HEI values and the corresponding node IDs within a path, providing valuable insights into efficient and inefficient nodes.

The integration of PEI into user packets using the In-situ Operations, Administration, and Maintenance (IOAM) protocol, specifically the hop-by-hop extension header, showcased a practical approach to transport efficiency metrics. The study also addressed the limitation of the IOAM Aggregation Type Option by recommending the use of the IOAM Trace Option Type in conjunction with the IOAM Aggregation Option Type for path assignment.

The demo application demonstrated the successful extraction and display of both the path traveled and the PEI for each packet. This innovative approach to incorporating efficiency indicators into user packets offers a tangible solution for optimizing network energy efficiency.

1.2. Implications of the Study

The study's implications extend to the optimization of network energy efficiency based on the elaborated efficiency indicators. By incorporating these indicators into data forwarding and including them as part of user packets, the network can be dynamically scaled up or down. For instance, traffic can be routed through the most efficient paths, and inefficient routes can remain deactivated until needed, leading to more sustainable networking practices. The study lays the foundation for increased visibility into the energy efficiency of network paths, marking a significant step toward sustainable networking practices.

1.3. Recommendations for Future Research

While the study successfully introduced efficiency indicators and demonstrated their application, there are potential limitations associated with mapping arbitrarily large values and restricting them to numerical ranges based on 2. Future research should delve deeper into analyzing these limitations and explore alternative approaches to address potential constraints. Additionally, investigating the scalability of the proposed solution and its performance in larger, more complex network scenarios could further enhance the practicality and applicability of the proposed methodology.

In addition to the considerations mentioned earlier, future research endeavors can focus on the development and implementation of complementary features to enhance the proposed solution's overall functionality and applicability.

Export of Network Energy Efficiency Data Using IPFIX into a Time Series Database Exploring the integration of the proposed energy efficiency indicators with the IP Flow Information Export (IPFIX) protocol RFC 7011 could offer a standardized and efficient means of exporting network energy efficiency data. By leveraging IPFIX, which is designed for collecting and exporting flow information, researchers can establish a structured approach to feed energy efficiency metrics into a time series database. This integration could facilitate comprehensive analysis, trending, and historical comparisons of network energy efficiency over time.

Central Monitoring Platform Accessing Time Series Data A critical aspect of advancing network energy efficiency research is the establishment of a central monitoring platform. This platform would interface with the time series database, providing a consolidated view of energy efficiency metrics across the network. Researchers can develop sophisticated visualization tools and analytics within the central monitoring platform, enabling real-time monitoring, trend analysis, and the identification of potential optimizations. Such a platform would serve as a valuable resource for network administrators and researchers seeking deeper insights into the dynamic nature of energy efficiency within the network.

Central Management Platform for Configuration To further enhance the practicality of an energy efficiency indicator-enabled network, the development of a central management platform is recommended. This platform would empower network administrators to configure parameters essential for running and customizing energy efficiency indicators. Researchers can explore user-friendly interfaces that allow for the adjustment of weighting components in the Hop Efficiency Indicator (HEI) calculation, fine-tuning of aggregation methods, and customization of alert thresholds. A central management platform would streamline the deployment and management of energy efficiency metrics within diverse network environments.

These additional research avenues not only extend the scope of the proposed solution but also contribute to the establishment of a comprehensive framework for managing and optimizing network energy efficiency. By integrating IPFIX, developing central monitoring platforms, and creating user-friendly management interfaces, future research can address the practical challenges associated with deploying and maintaining energy efficiency indicators within complex network infrastructures.

In conclusion, this research project has made substantial strides in proposing and implementing a practical solution for aggregating, transporting, and interpreting green metrics in a simulated computer network. The findings provide valuable insights into network efficiency, and the implications pave the way for more sustainable networking practices. Future research can build upon these foundations to address potential limitations and enhance the scalability of the proposed solution.

Bibliography

[1] Frank Brockners, Shwetha Bhandari, and Tal Mizrahi. *Data Fields for In Situ Operations, Administration, and Maintenance (IOAM)*. Request for Comments RFC 9197. Num Pages: 40. Internet Engineering Task Force, May 2022. DOI: 10.17487/RFC9197. URL: https://datatracker.ietf.org/doc/rfc9197 (visited on 12/13/2023).

[2] Alexander Clemm and Laurent Metzger. *Aggregation Trace Option for In-situ Operations, Administration, and Maintenance (IOAM)*. Internet Draft draft-cxx-ippm-ioamaggr-00. Num Pages: 9. Internet Engineering Task Force, Oct. 23, 2023. URL: https://datatracker.ietf.org/doc/draft-cxx-ippm-ioamaggr (visited on 12/09/2023).

[3] Alexander Clemm et al. *Green Networking Metrics*. Internet Draft draft-cx-green-metrics-02. Num Pages: 19. Internet Engineering Task Force, Mar. 8, 2023. URL: https://datatracker.ietf.org/doc/draft-cx-green-metrics (visited on 12/13/2023).

[4] *FURPS*. In: *Wikipedia*. Page Version ID: 1110147882. Sept. 13, 2022. URL: https://en.wikipedia.org/w/index.php?title=FURPS&oldid=1110147882 (visited on 11/01/2023).

[5] IEEEComSoc. *IEEE ICC 2018 // Keynote: Nick McKeown, Programmable Forwarding Planes Are Here To Stay*. June 14, 2018. URL: https://www.youtube.com/watch?v=8ie0FcsN07U (visited on 11/28/2023).

[6] *In Situ OAM (IOAM)*. URL: https://www.iana.org/assignments/ioam/ioam.xhtml (visited on 12/10/2023).

[7] *Internet Protocol Version 6 (IPv6) Parameters*. URL: https://www.iana.org/assignments/ipv6-parameters/ipv6-parameters.xhtml (visited on 12/10/2023).

[8] *P4 – Language Consortium*. URL: https://p4.org/ (visited on 11/29/2023).

[9] *P4~16~ Language Specification*. URL: https://staging.p4.org/p4-spec/docs/P4-16-v1.2.4.html (visited on 12/07/2023).

[10] *Scapy*. URL: https://scapy.net/ (visited on 12/14/2023).

Part V.

Appendix

A. IETF Draft

As already mentioned throughout this document the IETF draft *draft-cxx-ippm-ioamaggr* authored by Alexander Clemm served as the foundation for the research attempted during this semester thesis. The following pages enclose the current version of the draft.

Workgroup: IPPM
Internet-Draft: draft-cxx-ippm-ioamaggr-00
Published: 23 October 2023
Intended Status: Standards Track
Expires: 25 April 2024
Authors: A. Clemm
Futurewei Technologies, Inc.
L. Metzger
Ostschweizer Fachhochschule - OST

Aggregation Trace Option for In-situ Operations, Administration, and Maintenance (IOAM)

Abstract

The purpose of this memo is to describe a new option type for In-Situ Operations, Administration, and Maintenance (IOAM). This option type allows to aggregate IOAM data along a network path. Aggregates include functions such as the sum, average, minimum, or maximum of a given data parameter.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This memo proposes a new option type for In-Situ Operations, Administration, and Maintenance (IOAM) [RFC9197]. The IOAM Aggregate option type allows to aggregate IOAM data along a network path. Aggregates include functions such as the sum, average, minimum, or maximum of a given data parameter.

Many applications interested in telemetry data across a path are not so much interested in each individual node's telemetry, but an aggregate to paint a more holistic picture. An example of an aggregate could be a sum (for example, the sum of packet dwell times experienced across a path), an average (for example, the average dwell time experienced across a path), or a minimum or maximum (for example, of the dwell time experienced on any hop across the path, along with the node ID where the extreme was experienced). Other applications include sustainable networking, where (for example) the carbon-intensity of a path as a whole needs to be determined as an input to applications that attempt to minimize pollution attributable to specific networking traffic.

The aggregation option type proposed in this memo addresses the needs of those applications. Rather than collecting individual IOAM data parameters at each node and exporting them for further processing, IOAM Aggregate allows to preprocess telemetry data into an aggregate as a packet traverses a path. Aggregating parameters along the path, instead of merely collecting them, offers the following advantages:

* It keeps the packet size constant. This avoids problems such as the possibility of packets exceeding their MTU and need for fragmentation and reassembly in case of longer data paths, or deteriorating packet delays as packets grow in size along a path.

* It reduces the volume of data to be exported.

* It obviates the need to correlate data exported from individual nodes as belonging to the same flow, when compared with processing of postcard telemetry data [RFC9326].

* It significantly reduces the amount of processing that needs to be done by applications, simplifying their development and deployment.

* It enables greater network intelligence, such as taking actions on aggregates when certain thresholds are exceeded.

Aggregating parameters does require a small amount of processing (such as, an arithmetic operations to add to a sum, or a comparison operation to determine a minimum) at each hop, requiring some additional processing cycles. This is a small tradeoff to be aware of when choosing this option. We believe that this tradeoff will be acceptable in many implementations and deployment scenarios.

2. Background

[RFC9197] defines the scope of IOAM as well as the different IOAM nodes. The following section reiterates those roles and explains how they are applied in the context of IOAM Aggregation.

IOAM is focused on "limited domains", as defined in [RFC8799]. IOAM is not targeted for a deployment on the global Internet, which would incur additional considerations such as the crossing of Trust Boundaries, authentication of IOAM data, or the desire to obfuscate domain internals to outside parties. The part of the network that employs IOAM is referred to as the "IOAM-Domain".

An IOAM-Domain consists of "IOAM encapsulating nodes", "IOAM decapsulating nodes", and "IOAM transit nodes", as depicted in Figure 1.

```
                                                Export of
                                            IOAM data (opt.)
                                                    ^
                                                    |
User      +--------+     +--------+     +--------+     +---+----+
packets  |Encapsu-|     | Transit|     | Transit|     |Decapsu-|
-------->|lating   |====>| Node    |====>| Node    |====>|lating   |-->
         |Node     |     | A       |     | B       |     |Node     |
         +--------+     +--------+     +--------+     +--------+
```

Figure 1: Roles of IOAM nodes

The role of these nodes is as follows:

*The Encapsulating Node originates the IOAM aggregation. It adds the IOAM Aggregation Option to the packet for which telemetry data is to be aggregated across the path and populates the fields with their initial values.

*The Transit Node is an IOAM-enabled node that aggregates the value of its own telemetry with the aggregate in the packet, updating the aggregation data as needed.

*The Decapsulating Node terminates the IOAM aggregation. It aggregates the value of its own telemetry with the aggregate in the packet and updates the aggregation data as needed. It subsequently exports the aggregated data, specifically, including the value of the aggregate itself as well as auxiliary data as applicable (e.g. node ID for min, max, and in case of errors).

3. IOAM Aggregation Option-Type

3.1. Overview

This section defines the data fields for the IOAM Aggregation Option Type format. Like other IOAM Aggregation Option Types, these data fields can be mapped into a number of transport protocols [RFC9378]. For example, a transport over IPv6 [RFC8200] has been defined in [RFC9486].

The format of the IOAM Aggregation Option Type data fields is depicted in Figure 2.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Namespace-ID           | Flags |       Reserved        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               IOAM Data Param                 |  Aggregator   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Aggregate                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Auxil-data Node-ID                |   Hop Count   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2: IOAM Aggregation Option Type Format

The total length of the IOAM Aggregation Option Type data fields is fixed at 16 octets (word-aligned). These 16 octets hold header information as well as aggregation data in the following fields:

*Namespace-ID: 16-bit identifier of an IOAM-Namespace, as defined in [RFC9197]. The Namespace-ID is populated by the encapsulating node and MUST NOT be changed by any of the intermediate nodes. The Namespace-ID value of 0x0000 is defined as the "Default-Namespace-ID" and MUST be known to all the nodes implementing IOAM. For any other Namespace-ID value that does not match any Namespace-ID the node is configured to operate on, the node MUST NOT change the contents of the IOAM-Data-Fields except for the Namespace Flag (see below).

*Flags: 4-bit field to indicate errors that were encountered when attempting to process the IOAM Aggregation Option along the path. Once a flag is set, no further aggregation occurs along the path. An intermediate node that encounters an error during processing of the IOAM Aggregation that prevents it from updating the aggregate as requested MUST set the corresponding flag to 1. In order to facilitate troubleshooting, it also MUST set the value of the Auxil-data Node-ID field to its own Node-ID. The encapsulating node MUST set the value of Flags to zero upon transmission. When an intermediate node encounters receives a packet in which any of the Flags are non-zero, the node MUST NOT perform further IOAM operations on that packet; instead, the IOAM data MUST be forwarded as-is unchanged. The following flags are defined:

-Flag 1: Aggregator not supported

-Flag 2: Unsupported IOAM data parameter

-Flag 3: Unsupported Namespace

-Flag 4: Any other error

*Reserved: An IOAM encapsulating node MUST set the value to zero upon transmission. IOAM transit nodes MUST ignore the received value.

*IOAM Data Param: This field identifies the data parameter that is to be aggregated across the nodes. It MUST be set by the IOAM encapsulating node. IOAM transit nodes MUST NOT change it. Contrary to IOAM Trace-Type in the pre-allocated and incremental trace option types, only a single parameter is aggregated at a time. Accordingly, the data parameter to be aggregated is not identified by a a particular bit, but by a value.

*Aggregator: This 8-bit field identifies the aggregation function that is to be applied. Its value MUST be set by the IOAM encapsulating node. IOAM transit nodes MUST not change it. The following aggregators are defined:

  -Sum (value: 0b1)

  -Min (value: 0b10)

  -Max (value: 0b100)

  -Average (value: 0b1000)

*Aggregate: This 32-bit field contains the aggregated value. Its value is initialized by the encapsulating node,in general by simply recording the value of its data parameter that is to be aggregated. The field is updated by each subsequent node pre the requested aggregation, including IOAM transit nodes as well as the IOAM decapsulating node (prior to performing decapsulation).

*Auxil-data Node-ID: This 24-bit field contains a Node-ID. It MUST be set by the encapsulating node to its own Node ID. Subsequent nodes (IOAM transit nodes, as well as the IOAM decapsulating node prior to performing decapsulation) MUST update the value to their own Node-ID IF AND ONLY IF one of the following conditions hold, otherwise they MUST NOT change its value:

  -When a flag is set by the node (i.e., the first time any type of error is encounted along the path)

  -When the aggregator is one of Min or Max, and a new minimum respectively maximum is encountered. The value of the Auxil-data Node-ID field is hence used to record where the minimum respectively maximum value was first encountered. (When a node matches an existing minimum or maximum but does not beat it, the Node-ID is not updated.)

* Hop Count. This 8-bit fields contains a hop count to record the number of nodes along the path that successfully processed the IOAM Aggregation. The encapsulating node MUST set the value to 1, and each subsequent node (transit nodes, as well as the decapsulating node prior to performing decapsulation) MUST increment its value by 1. If the Hop Count at a node exceeds 255, that node MUST set the Hop Count to 0 and set Flag 4 ("any other error") to prevent further processing of the IOAM Aggregation.

3.2. Discussion

This section explains some of the design choices and points out items that may be subjected to further discussion.

* Single versus multiple parameters. The Aggregation Option Type allows to only aggregate one data parameter at a time. This allows to keep the format of the data structure simple and of fixed size. This facilitates processing. It also limits the number of processing cycles that need to be spent for aggregation at each node. An application seeking to perform multiple types of aggregation at a time will need to apply different types of aggregation for different packets.

* IOAM data parameter identification. Unlike other IOAM Option Types, data parameters are not represented by a bit position in a field but by a 24-bit identifier. This allows to support a greater number of parameters. In order to facilitate compatibility, initially only identifiers SHOULD be used that utilize bits 12 through 22, with other bits set to 0. The assignment of IOAM data parameter identifiers is at this point up to the network operator, with IOAM data parameters being specific to an IOAM name space. It is conceivable that a global namespace and a corresponding IANA registry for IOAM data parameters would be introduced at a later point in time.

* Average aggregator. An average can be easily derived from dividing a sum obtained across all nodes by the hopcount. Avoiding division operations along the path can save considerable processing cycles. It is FFS if the average aggregator is really required.

* Simultaneous use with other IOAM Option Types. There are use cases conceivable that would benefit from also adding a trace of which nodes were actually traversed on the path. The possibility to do so is already provided with other IOAM Option Types and does not need to be added here. In order to use multiple IOAM Option Types simultaneously, applications can use one of several alternatives. In one alternative, multiple IOAM Option Types with their corresponding data structures are simultaneously used in

the same packet. In another alternative, different packets of the same flow are each send with a different IOAM Option Type, a form of sampling which however provides no absolute guarantees of path congruency (i.e., different packets traversing the exact same path).

4. Security Considerations

A malicious node along the path could attempt to forge the aggregate, resulting in a wrong aggregate to be reported. This might mislead applications. Likewise, a malicious node along the path could set a flag to trick other nodes not to process the aggregate any further, or clear a flag to make an errored result appear legitimate. To avoid this, network operators need to ensure that their network nodes can be trusted and are not compromised.

5. IANA Considerations

IANA requests are TBD. Future versions of this document may request the establishment of a registry for Aggregators as well as for IOAM Data Parameters.

6. Contributors

*Ramon Bister, OST

*Severin Dellsperger, OST

*Reto Furrer, OST

7. References

7.1. Normative References

[RFC9197] Brockners, F., Ed., Bhandari, S., Ed., and T. Mizrahi, Ed., "Data Fields for In Situ Operations, Administration, and Maintenance (IOAM)", RFC 9197, DOI 10.17487/RFC9197, May 2022, <https://www.rfc-editor.org/rfc/rfc9197>.

7.2. Informative References

[RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/rfc/rfc8200>.

[RFC8799] Carpenter, B. and B. Liu, "Limited Domains and Internet Protocols", RFC 8799, DOI 10.17487/RFC8799, July 2020, <https://www.rfc-editor.org/rfc/rfc8799>.

[RFC9326]
Song, H., Gafni, B., Brockners, F., Bhandari, S., and T. Mizrahi, "In Situ Operations, Administration, and Maintenance (IOAM) Direct Exporting", RFC 9326, DOI 10.17487/RFC9326, November 2022, <https://www.rfc-editor.org/rfc/rfc9326>.

[RFC9378] Brockners, F., Ed., Bhandari, S., Ed., Bernier, D., and T. Mizrahi, Ed., "In Situ Operations, Administration, and Maintenance (IOAM) Deployment", RFC 9378, DOI 10.17487/RFC9378, April 2023, <https://www.rfc-editor.org/rfc/rfc9378>.

[RFC9486] Bhandari, S., Ed. and F. Brockners, Ed., "IPv6 Options for In Situ Operations, Administration, and Maintenance (IOAM)", RFC 9486, DOI 10.17487/RFC9486, September 2023, <https://www.rfc-editor.org/rfc/rfc9486>.

Authors' Addresses

Alexander Clemm
Futurewei Technologies, Inc.

Email: ludwig@clemm.org

Laurent Metzger
Ostschweizer Fachhochschule - OST

Email: laurent.metzger@ost.ch