Project Report on

Telemetry Support in IxNetwork Data plane & Dissector

By

Karan Saraf (20140372)

Group No. 102

In partial fulfillment of requirements for the award of degree in
Bachelor of Technology in Computer Science and Engineering
(2018)

SMIT SIKKIM MANIPAL UNIVERSITY

SIKKIM MANIPAL INSTITUTE OF TECHNOLOGY

Under the Project Guidance of

Mr. Bidyut Mondal
Project Manager, IXIA Solutions Group
Keysight Technologies
And
Internal Reviewer
Mr. Kiran Gautam, Assistant Professor-I, SMIT

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SIKKIM MANIPAL INSTITUTE OF TECHNOLOGY
(A constituent college of Sikkim Manipal University)
MAJITAR, RANGPO, EAST SIKKIM – 737136

PR/5682
26.9.18

Project Report on

Telemetry Support in IxNetwork Data plane & Dissector

By

Karan Saraf (20140372)

Group No. 102

In partial fulfillment of requirements for the award of degree in

Bachelor of Technology in Computer Science and Engineering

(2018)

SMIT SIKKIM MANIPAL UNIVERSITY

SIKKIM MANIPAL INSTITUTE OF TECHNOLOGY

Under the Project Guidance of

Mr. Bidyut Mondal

Project Manager, IXIA Solutions Group

Keysight Technologies

And

Internal Reviewer

Mr. Kiran Gautam, Assistant Professor-I, SMIT

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SIKKIM MANIPAL INSTITUTE OF TECHNOLOGY

(A constituent college of Sikkim Manipal University)

MAJITAR, RANGPO, EAST SIKKIM – 737136

Project Completion Certificate

This is to certify that the below mentioned student(s) of Sikkim Manipal Institute of Technology has worked under my supervision and guidance from 5th January 2018 to 18th May 2018 and has successfully completed the project entitled **"Telemetry Support in IxNetwork Data Plane & Dissector"** in partial fulfillment of the requirements for the award of Bachelor of Technology in Computer Science and Engineering.

University Registration No	Name of Student(s)	Course
20140372	Karan Saraf	B.Tech (CSE)

Bidyut Mondal

.....

Mr. Bidyut Mondal

Project Manager

IXIA Solutions Group

Keysight Technologies

Kolkata

Project Review Certificate

This is to certify that the work recorded in this project report entitled "**Telemetry Support in IxNetwork Data Plane & Dissector**" has been carried out by Mr. Karan Saraf (Reg 20140372) of Computer Science & Engineering Department of Sikkim Manipal Institute of Technology in partial fulfillment of the requirements for the award of Bachelor of Technology in Computer Science and Engineering. This report has been duly reviewed by the undersigned and recommended for final submission for Major Project Viva Examination.

.....

Mr. Kiran Gautam

Assistant Professor Gr – I

Department of Computer Science & Engineering

Sikkim Manipal Institute of Technology

Majitar, East Sikkim – 737136.

Certificate of Acceptance

This is to certify that the below mentioned student(s) of Computer Science & Engineering Department of Sikkim Manipal Institute of Technology (SMIT) has worked under the supervision of Mr.Bidyut Mondal of Keysight Technologies, Kolkata from 5th January 2018 to 18th May 2018 on the project entitled "**Telemetry Support in IxNetwork Data Plane & Dissector**"

The project is hereby accepted by the Department of Computer Science & Engineering, SMIT in partial fulfillment of the requirements for the award of Bachelor of Technology in Computer Science and Engineering.

University Registration No	Name of Student(s)	Project Venue
20140372	Karan Saraf	Ixia Office, Kolkata

31/05/2018

Kalpana Sharma
Prof. & Head
Computer Science & Engineering
Sikkim Manipal Institute of Technology
Majitar, Rangpo- 737136

.....

Dr. Kalpana Sharma

Professor & HOD

Computer Science & Engineering Department

Sikkim Manipal Institute of Technology

Majhitar, Sikkim - 737136

Signature.....

Date.....

Declaration

I, the undersigned, hereby declare that the work recorded in this project report entitled "**Telemetry Support in IxNetwork Data Plane & Dissector**" in partial fulfillment for the requirements of award of B. Tech (CSE) from Sikkim Manipal Institute of Technology (A constituent college of Sikkim Manipal University) is a faithful and bonafide project work carried out at Kolkata under the supervision and guidance of Mr.Bidyut Mondal of Keysight Technologies, Kolkata.

The results of this investigation reported in this project have so far not been reported for any other Degree / Diploma or any other Technical forum.

The assistance and help received during the investigation have been duly acknowledged.

Karan Saraf

.....

Mr. Karan Saraf (Reg 20140372)

Acknowledgement

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

I am grateful to Mr. Bidyut Mondal, Keysight Technologies for his even willingness to give me valuable advice and direction; whenever I approached him with a problem. I am thankful to him for providing immense guidance for this project.

I would like to express my humble gratitude to Mr. Biswaraj Sen, Assistant Professor, Mr. Saurav Paul, Assistant Professor-I, Mr. Vikash Kumar Singh, Assistant Professor-I and Miss Pratikshya Sharma, Assistant Professor I, Project Coordinator, Computer Science and Engineering Department, Sikkim Manipal Institute of Technology for their timely support and guidance till the completion of our project work.

I express my deepest thanks to Mr. Suvendu Mozumdar for taking part in useful decision & giving necessary advices and guidance and arranged all facilities to make life easier. I am highly indebted to Mr. Kiran Gautam for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

I am are pleased to acknowledge Dr. Kalpana Sharma, HOD CSE, for allowing me to pursue the project in this company and appreciate her support during the project period.

I would like to express my special gratitude and thanks to Mr. Supriya Sarkar, Mr. Mrinmoy Das, Mr. Satyam Kumar Singh and all others for giving me such attention and time and helping me take on every small milestone.

I would like to express my gratitude towards the Department of CSE, SMIT being the primary reason for this opportunity & all the project coordinators for their kind co-operation and encouragement which help me in completion of this project.

Karan Saraf (20140372)

DOCUMENT CONTROL SHEET

1	Report No	CSE/Major Project/External/B.Tech/102/2018
2	Title of the Report	Telemetry Support in IxNetwork Data Plane & Dissector
3	Type of Report	Technical
4	Author(s)	Karan Saraf
5	Organizing Unit	Keysight Technologies, Kolkata
6	Language of the Document	English
7	Abstract	Telemetry feature added in Geneve variable length options and IPv6 hop-by-hop Option header. Enhancement of wireshark to display the telemetry packets data.
8	Security Classification	General
9	Distribution Statement	General

Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviations	Full Form
INT	In- Band Telemetry
IPv6 HBH	Internet Protocol Version 6 Hop by Hop
IOAM	In-situ Operations, Administration, and Maintenance
GENEVE	Generic Network Virtualization Encapsulation
IPv4	Internet Protocol Version 4
UDP	User Datagram Protocol
VLAN	Virtual Local Area Network

Abstract

Telemetry is an automated communications process by which measurements and other data are collected at remote or inaccessible points and transmitted to receiving equipment for monitoring. The receiving can collect the data and use it for further computation as and when required.

Telemetry packets can be created by encapsulating it over various other packets such as Geneve, VLAN and IPv6 etc. In this project we focused on Geneve and IPv6.

These telemetry packets can be created and further used by the IxNetwork to run various test cases to test the proper working of the packets by creating a virtual traffic for the packet transmission. Once traffic is created these packets can further be captured in wireshark. Wireshark database must be updated with a program written specific to the telemetry packet so that upon capturing wireshark can decode the captured packet and make meaning out of the large stream of bits and display it. That information can then be used for various computation as required by the user.

In this project we will modify the existing protocols for Telemetry support. Geneve's variable options fields will be encapsulated with In-Band Telemetry Network and IPv6 Hop-By-Hop Options Header will be encapsulated with IOAM Data Fields. The project also focusses on modifying wireshark so that it can support the Telemetry fields.

Chapter 1

Introduction

1. Introduction

Inband Network Telemetry ("INT") is a framework designed to allow the collection and reporting of network state, by the data plane, without requiring intervention or work by the control plane. In the INT architectural model, packets contain header fields that are interpreted as "telemetry instructions" by network devices. These instructions tell an INT capable device what state to collect and write into the packet as it transits the network.

The Generic Network Virtualization Encapsulation (Geneve) protocol offers a new approach to encapsulation designed to offer control-plane independence between tunnel endpoints. The protocol specifies only a data-plane schema using a number of variable length options. Each packet contains an option length parameter, and the interpretation of different options is outside the specification.

IOAM data fields are carried by different transport protocols. Layering: If several encapsulation protocols (e.g., in case of tunneling) are stacked on top of each other, IOAM data-records could be present at every layer.

In IPv6 [RFC2460], optional Internet-layer information is encoded in extension headers that may be placed between the IPv6 header and the upper-layer header. Currently, eleven extension headers are defined. Among them is the Hop-by-Hop (HBH) Options Extension header. The HBH Extension Header contains one or more HBH Options. Each HBH option contains a type identifier. Some HBH Options contain information that is useful to a router's forwarding plane.

IxNetwork is a comprehensive network infrastructure performance testing solution. It scales to handle most powerful devices and largest networks, from routing and switching to data center ethernet and software defined networking.

Wireshark is cross-platform, using the Qt widget toolkit in current releases to implement its user interface, and using pcap to capture packets; it runs on Linux, macOS, BSD, Solaris, some other Unix-like operating systems, and Microsoft Windows. There is also a terminal-based (non-GUI) version called TShark. Wireshark,

and the other programs distributed with it such as TShark, are free software, released under the terms of the GNU General Public License.

1.1 General Overview of Telemetry

Telemetry, in general, is a term for technologies that accommodate collecting information in the form of measurements or statistical data, and forward it to IT systems in a remote location. It allows for the robust collection of data and its delivery to centralized systems where it can be used effectively. Big data technologies and big data strategies that take enormous amounts of relatively unstructured data and aggregate it in centralized systems. These types of "data-rich" data transit systems provide sophisticated and detailed reports and intelligence for any industry or public agency.

Our basic aim in this project was to encapsulate INT telemetry fields in two protocols, GENEVE and IPv6. Both the protocols have an option field where our required telemetry fields are added. IPv6 has 11 extension headers, the first one is Hop-By-Hop option header where we encapsulated our required protocol.

IxNetwork was used to create virtual traffic which provided testing aid. The traffic was then again captured in IxNetwork and wireshark is used to display the captured packet's details. Wireshark code database was updated once the packet structure was created in .xml format so that it decodes the captured stream of bits into meaningful data.

Geneve header contains the following fields:
Version, O, C, Reserved, Protocol Type, VNI, Reserved and Variable Options Length. The last field (variable option length) will be used for adding telemetry.
Similarly, IPv6 Options Header will contains the following field:
Next header, Header Extension Length and Option Data. Option data will contain the IOAM data fields and its length will be automatically computed and reflected in header extension length.

rature Survey

Paper and Authors	Findings	Relevance to the project
Functionalities of wireshark functions and datatypes in Readme.doc in perforce library. Author: Mr. Supriya Sarkar Source: Informative article in perforce library IXIA domain published in 2010	• Various functions and their uses. • Function definition and their args.	How to build a wireshark code for specific packets.
In Band Network Telemetry (INT) By: Hugh Holbrook: Arista; Anoop Ghanwani: Del; Dan Daly: Intel Source: Internet published in June 2017.	• Telemetry packet format for Geneve. • Various fields and their sizes.	Creation of Telemetry packet template for geneve.
Encapsulations for In-situ OAM Data draft-brockners-inband-oam-transport-05 Source: Draft published by Internet Engineering Task Force (IETF) in July 2017.	• Packet format for IPv6 HBH • Field name and their size.	Adding fields in IPv6 hop-by-hop extension header

1.3 Problem Definition

Routers uses predefined algorithms to compute most of the routing paths. This computation is based on fixed parameters. Generally, the shortest path between nodes is used to compute the path.

For example, consider a situation where we want to watch a video online. Our first and the most important requirement would be less latency, but the shortest path might have high latency or consider another situation where you have to share a big size file. For that we would want to transfer the file whose bandwidth is high but the shortest path might have the less bandwidth.

Currently wireshark and IxNetwork does not have support for telemetry. For IxNetwork it lacks the telemetry fields in the proposed protocol packets i.e., Geneve and IPv6 HBH. For wireshark, if we open telemetry packet, it will not understand anything and display unknown data. It will consider the telemetry data as raw option data and display it accordingly. But that is not desirable.

1.4 Problem Analysis and SRS

1.4.1 Problem Analysis

Currently the route computation between source and destination is done by some predefined algorithms which computes the path based on certain parameters. One of the major parameter is the cost between the two paths. But there are certain cases where we would want to compute paths based on different parameters as well like bandwidth, latency etc.

Our aim is to collect this kind of data from all the intermediate routers.

The telemetry and IOAM data fields must be added in some protocols or a new protocol must be created for this operation. Since creating a new protocol is a bigger task, we focused on encapsulating the fields in the option part in some of the protocols which can support telemetry and

IOAM. Two of such protocol is GENEVE (Generic Network Virtualization Encapsulation) and IPv6 Hop-by-Hop Extension header respectively.

New fields must be created in order to display fields so that necessary information can be added by the router in the packet while transmission.

But IxNetwork doesn't support INT or IOAM data fields. It only contains the GENEVE and IPv6 HBH packets with its original fields.

Wireshark that we use needs to be updated with the new codes so that it can make meaning out of the captured stream of bytes. Currently wireshark doesn't support telemetry and IOAM data fields encapsulated over any protocol.

1.4.2 Software Requirements Specification (SRS)

Hardware Requirements:

- RAM : Minimum 4 GB
- Disk Space : Minimum 4 GB

Software Requirements:

- IxNetwork (Preferably latest version)
- Wireshark (Provided by the company)
- Operating System – Windows 7 or higher

1.5 Proposed Solution Strategy

IxNetwork has a library which contains all the protocol packets. Since telemetry will be a new packet which will be encapsulated over the variable length option field of GENEVE packet and IOAM data fields over option data field of Hop-by-Hop Extension Header. Once the packet is created it can be added in the IxNetwork library.

Now we need to test this packet by creating a virtual raw traffic and adding the either Geneve or IPv6 HBH (proper topology must be maintained while creating traffic).

After creating the traffic, the packet will be transferred over the network using two ports i.e. source and destination.

Once the traffic starts functioning properly, IxNetwork can be used to capture the ongoing traffic between the network ports.

We can save the captured data in .cap format which is a wireshark executable file. Wireshark helps in decoding the packet and displaying the packet information in user understandable form. A wireshark program will be written in c language which will help wireshark to understand the captured stream of bytes and dissect the packet bitwise and the convert the binary packet data into user understandable data.

Once the program is ready wireshark program can be compiled to update its database with the new telemetry packet and the captured file can be open after that.

On successful execution all the required details added in the telemetry packet can then be displayed on wireshark GUI.

1.6 Preliminary User's Manual

The users should have the basic knowledge of networking (protocols, topology etc.), IxNetwork and wireshark. The reader does not have to know anything about wireshark in advance although it would be a small advantage. This project will aid those who wants to test their networking devices for telemetry support and also for those companies which are handles various traffics and needs different path computations for different kinds of data.

i. Configuring Traffic:

1. Open advanced traffic wizard in the IxNetwork
2. Add the required topology. A
3. Add the required Protocol stack (Geneve stack or IPv6 HBH stack)
4. Expand the individual packets to fill packet fields with required data.
5. Once all the details are filled click on finish.

ii. Configure Wireshark:

1. Open the wireshark provided by the company.
2. If IPv6 HBH stack was chose by the user then add the type value provided in the "Option Type" field while editing the packet in the txt file provided in the wireshark folder.
3. Open capture for testing.

1.7 Organization of Report

CHAPTER 1: Introduction

This chapter gives a brief idea regarding inband telemetry, ioam data fields, IPv6 HBH, IxNetwork and wireshark. It gives a general idea about what this project is about and how is it providing a solution for the problem. Additional information regarding the SRS is also mentioned.

CHAPTER 2: Design and Solution Strategy

This chapter gives a visual representation of the steps taken to solve the given problem. It includes work flow diagram, flow charts and use case diagram to represent the different functionalities implemented in the project.

CHAPTER 3: Detailed Test Plan

This chapter describes the testing strategies adopted to test the entire protocol stack. Whether the packet is created at the source is the correct that is received at the user end.

CHAPTER 4: Implementation Details

This chapter gives an overview of the algorithms used to update the wireshark code database to provide support for the newly added packets.

CHAPTER 5: Results and Discussion

This chapter showcases the results obtained.

CHAPTER 6: Summary and Conclusion

This chapter summarizes the modules, their results, the difficulties which were encountered, limitations of the project and the scope of future improvement in the project.

CHAPTER 7: Final User's Manual

This chapter describes the usage of the resulting project in detail.

CHAPTER 8: References/ Bibliography

This section enlists the various sources which were referred to during the development of the project.

Chapter 2

Design Strategy

2. Design Strategy

The In-Band Telemetry Network (INT) is encapsulated over GENEVE and IOAM data fields is encapsulated over IPv6. The various fields that are to be added in the packet is mentioned with their respective description.

2.1 Packet Design Details

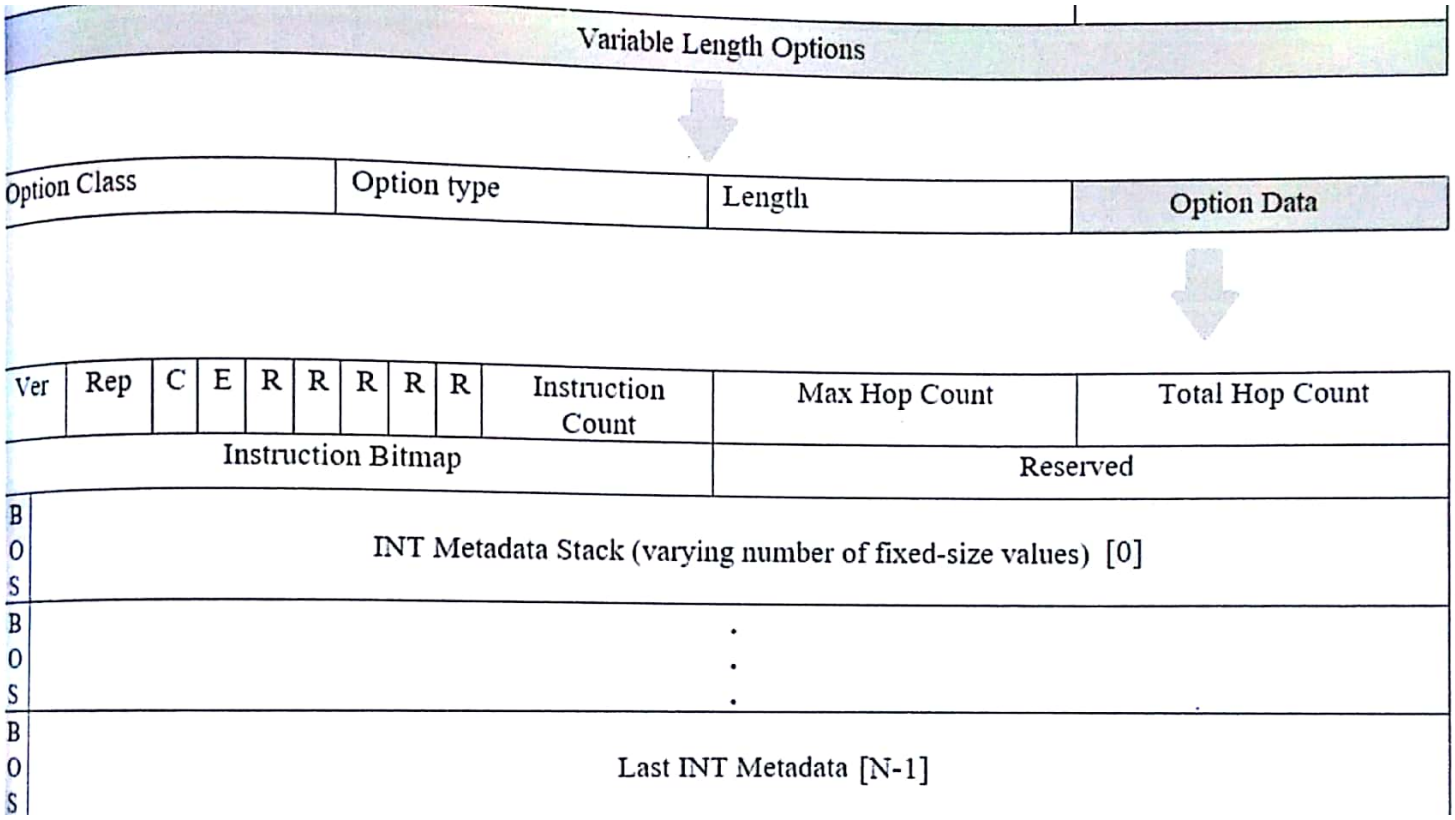
In-Band Telemetry Network Fields:

- **Version (2 bits)**: The current version number is 0.
- **Option Length (6 bits)**: The length of the options fields, expressed in four byte multiples, not including the eight byte fixed tunnel header.
- **O (1 bit)**: OAM frame. This packet contains a control message instead of a data payload.
- **C (1 bit)**: If this bit is set then tunnel endpoints MUST parse the options list to interpret any critical options.
- **Reserved (6 bits)**: Reserved field which MUST be zero on transmission and ignored on receipt.
- **Protocol Type (16 bits)**: The type of the protocol data unit appearing after the Geneve header.
- **Virtual Network Identifier (VNI) (24 bits)**: An identifier for a unique element of a virtual network.
- **Reserved (8 bits)**: Reserved field which MUST be zero on transmission and ignored on receipt.
- **Option Class (16 bits)**: Namespace for the 'Type' field. IANA will be requested to create a "Geneve Option Class" registry to allocate identifiers for organizations, technologies, and vendors that have an interest in creating types for options.
- **Type (8 bits)**: Type indicating the format of the data contained in this option.
- **R (1 bit)**: Option control flag reserved for future use. MUST be zero on transmission and ignored on receipt
- **Length (5 bits)**: Length of the option, expressed in four bytes multiples excluding the option header. The total length of each option ranges from 4 and 128 bytes.
- **Version (2bits)**: INT metadata header version. Should be zero for this version.

- **Replication (2bits)**: Replication requested. Support for this request is optional. If this value is nonzero, the device may replicate the INT packet. It has 3 modes:
  - 0: No replication requested.
  - 1: Portlevel (L2level) replication requested
  - 2: Nexthoplevel (L3level) replication requested.
  - 3: PortandNexthoplevel replication requested.
- **C (1bit)**: Copy.
  - If replication is indeed requested for data packets, the INT Sink must be able to distinguish the original packet from replicas.
  - C bit must always be set to 0 by INT source.
- **E (1bit)**: Max Hop Count exceeded.
  - This flag must be set if a device cannot prepend its own metadata due to reaching the Max Hop Count.
- **R (5 bits)**: Reserved bits.
- **Instruction Count (5 bits)**: The number of instructions that are set (i.e., number of 1's) in the instruction bitmap.
- **Max Hop Count (8 bits)**: The maximum number of hops that are allowed to add their metadata to the packet.
- **Total Hop Count (8 bits)**: The total number of hops that added their metadata instance(s) to the INT packet.
- **Instruction Bitmap (16 bits):** INT instructions are encoded as a bitmap in the 16bit INT Instruction field: each bit corresponds to a specific standard metadata.
  - **Switch ID (Bit 0)**: The unique ID of a switch (generally administratively assigned). Switch IDs must be unique within a management domain.
  - **Ingress port ID (Bit 1)**: The logical port on which the INT packet was received
  - **Hop latency (Bit 2)**: Time taken for the INT packet to be switched within the device.
  - **Queue occupancy (Bit 3)**: The buildup of traffic in the queue (in bytes, cells, or packets) that the INT packet observes in the device while being forwarded.
  - **Ingress timestamp (Bit 4)**: The device local time when the INT packet was received on the ingress physical or logical port.

- **Egress port ID (Bit 5)**: The ID of the output port via which the INT packet was sent the exact meaning of this value can differ for individual devices.
- **Queue congestion status (Bit 6)**: The fraction (in percentage or decimal fraction) of current queue occupancy relative to the queue size limit.
- **Egress port tx utilization (Bit 7)**: Current utilization of the egress port via which the INT packet was sent out.
- **Reserved (Bit 8 - Bit 15)**: The remaining bits are unassigned or reserved.

- **Reserved (16 bits)**: Reserved field which MUST be zero on transmission and ignored on receipt.
- **INT Metadata Stack (32 bits)**: It contains various metadata and is append in the stack manner. Whichever flag is marked 1 in the instruction bitmap, those metadata will be added in this stack. It has two parts:
  - **BOS (1 Bit)**: It represents whether the current is the last node or not. All the BOS values is set to 0 except the last one which will be set to 1 representing the last value.
  - **Metadata (31 Bits)**: It contains the actual metadata, provided its flag is set. Refer to instruction for metadata fields.



IOAM Data Fields:

- **Next Header (8 Bits):** It represents which protocol follows the current protocol.
- **Header Extension Length (8 Bits):** The value of the Header Extension Length field is the number of 8-byte blocks in the Hop-by-Hop Options extension header, not including the first 8 bytes
- **Option Type (8 Bits):** 8-bit identifier of the type of option. It can be of 4 types:
  - Pre-Allocated Tracing Option
  - Incremental Tracing Option
  - Proof of Transit Option
  - Edge to Edge Option
- **Option Data Length (8 Bits):** 8-bit unsigned integer. Length of the Reserved and Option Data field of this option, in octets.
- **Reserved (16 Bits):** 16-bit field MUST be filled with zeroes.
- **IOAM Trace Type (16 Bits):** A 16-bit identifier which specifies which data types are used in this node data list.
  - Bit 0: (Most significant bit) When set indicates presence of Hop_Lim and node_id in the node data.
  - Bit 1: When set indicates presence of ingress_if_id and egress_if_id (short format) in the node data.
  - Bit 2: When set indicates presence of timestamp seconds in the node data.
  - Bit 3: When set indicates presence of timestamp subseconds in the node data.
  - Bit 4: When set indicates presence of transit delay in the node data.
  - Bit 5: When set indicates presence of app_data (short format) in the node data.
  - Bit 6: When set indicates presence of queue depth in the nodedatra.
  - Bit 7: When set indicates presence of variable length Opaque State Snapshot field.
  - Bit 8: When set indicates presence of Hop Lim and node id in wide format in the node data.
  - Bit 9: When set indicates presence of ingress_if_id and egress_if_id in wide format in the node data.
  - Bit 10: When set indicates presence of app_data wide in the node data.

- Bit 11: When set indicates presence of the Checksum Complement node data.
- Bit 12-15: Undefined. An IOAM encapsulating node must set the value of each of these bits to 0.

- **NodeLen (5-bit unsigned integer)**: This field specifies the length of data added by each node in multiples of 4-octets, excluding the length of the "Opaque State Snapshot" field
- **Flags (4 Bits)**: Following flags are defined:
  - **Overflow (O-bit) (1 Bit, most significant bit)**: This bit is set by the network element if there is not enough number of octets left to record node data, no field is added and the overflow "O-bit" must be set to "1" in the header.
  - **Loopback (L-bit) (1 Bit)**: Loopback mode is used to send a copy of a packet back towards the source.
  - **Reserved (2 Bits)**: Must be zero.
- **Remaining/Maximum Length (7-bit unsigned integer)**: This field specifies the data space in multiples of 4-octets remaining for recording the node data, before the node data list is considered to have overflowed for pre-allocated type and for incremental type this field is the represents the maximum length of that the nodes can be of.
- **Node data List [n]**: Variable-length field. The type of which is determined by the IOAM-Trace-Type bit representing the n-th node data in the node data list. The various data fields are:
  - **Hop_Lim and Node_Id**:
    - *Hop_Lim*: 1-octet unsigned integer. It is set to the Hop Limit value in the packet at the node that records this data.
    - *Node_Id*: 3-octet unsigned integer. Node identifier field to uniquely identify a node within in-situ OAM domain.
  - 4-octet field defined as follows:

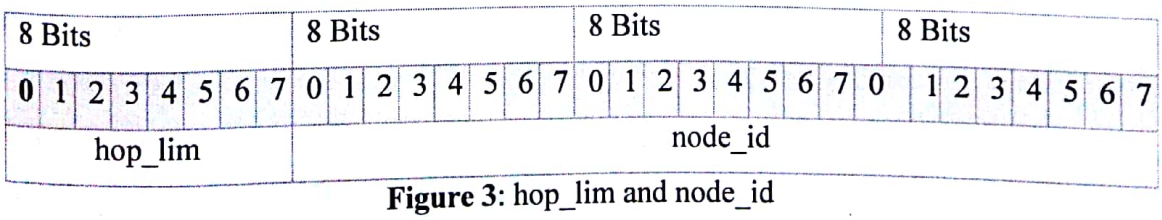


Figure 3: hop_lim and node_id

- **Ingress_if_id and Egress_if_id:**
  - *ingress_if_id*: 2-octet unsigned integer. Interface identifier to record the ingress interface the packet was received on.
  - *egress_if_id*: 2-octet unsigned integer. Interface identifier to record the egress interface the packet is forwarded out of.

4-octet field defined as follows:

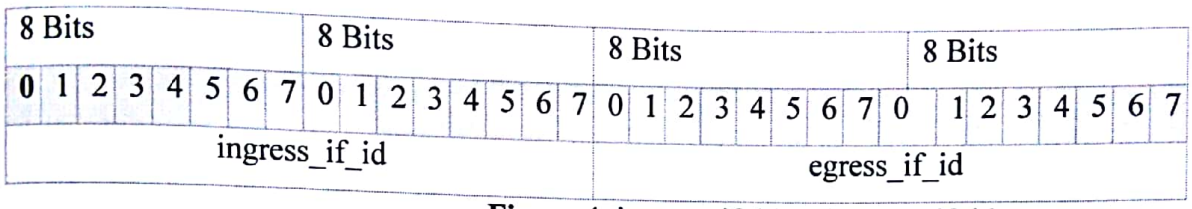


Figure 4: ingress_if_id and egress_if_id

- **Timestamp Seconds**: 4-octet unsigned integer. Absolute timestamp in seconds that specifies the time at which the packet was received by the node. This field as three possible formats; based on either PTP [IEEE1588v2], NTP [RFC5905], or POSIX [POSIX].
- **Timestamp Subseconds**: 4-octet unsigned integer. Absolute timestamp in subseconds that specifies the time at which the packet was received by the node. This field has three possible formatssimilar to timestamp seconds.

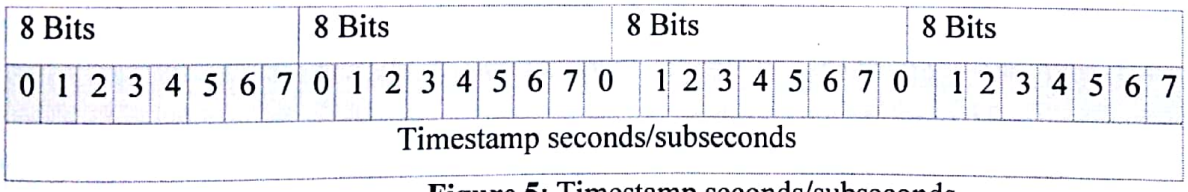


Figure 5: Timestamp seconds/subseconds

- **Transit Delay**: 4-octet unsigned integer in the range 0 to 2^31-1.It is the time in nanoseconds the packet spent in the transitnode. If the transit delay exceeds 2^31-1 nanoseconds then the top bit 'O' is set to indicate overflow and value set to 0x80000000.

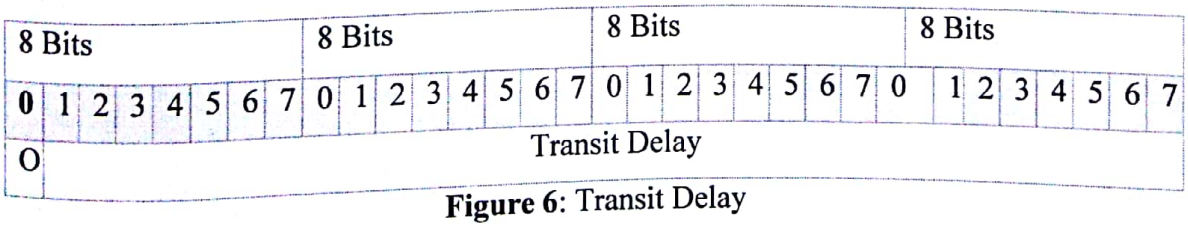


Figure 6: Transit Delay

- **App Data**: 4-octet placeholder which can be used by the node to add application specific data. App_data represents a "free-format" 4-octet bit field with its semantics defined by a specific deployment.

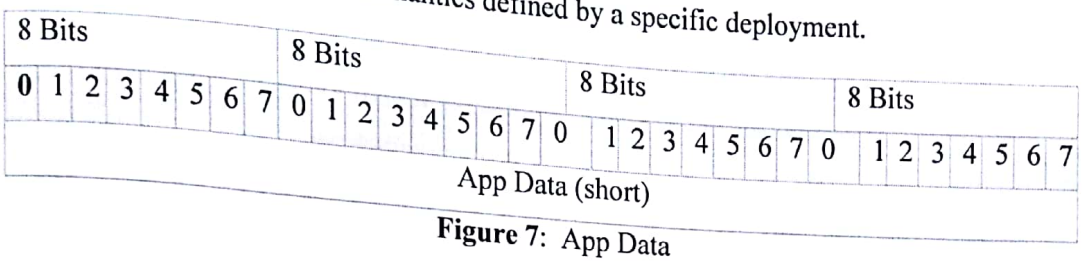


Figure 7: App Data

- **Queue Depth**: 4-octet unsigned integer field. This field indicates the current length of the egress interface queue of the interface from where the packet is forwarded out.

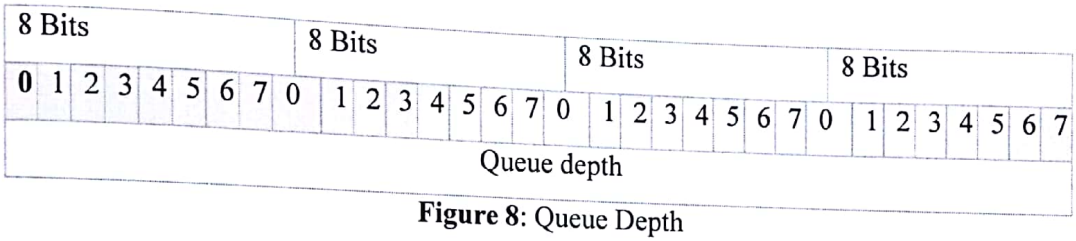


Figure 8: Queue Depth

- **Opaque State Snapshot**: Variable length field. It allows the network element to store an arbitrary state in the node data field, without a pre-defined schema. It has 3 fields:
  - *Length*: 1-octet unsigned integer. It is the length in multiples of 4-octets of the Opaque data field that follows Schema Id.
  - *Schema ID*: 3-octet unsigned integer identifying the schema of Opaque data.
  - *Opaque data*: Variable length field. This field is interpreted as specified by the schema identified by the Schema ID.

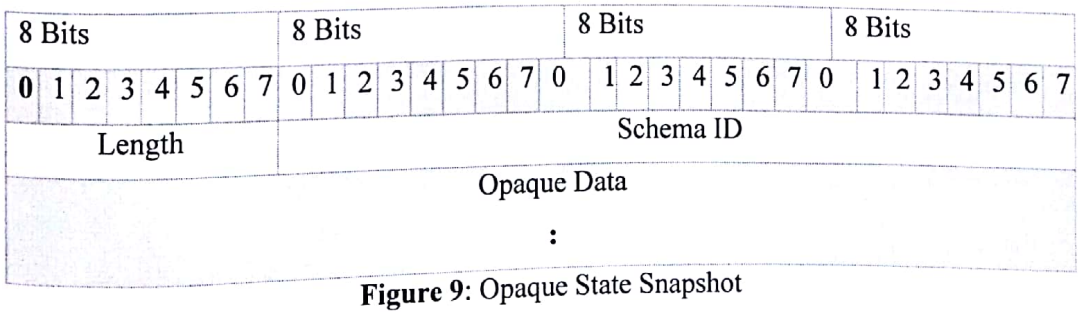


Figure 9: Opaque State Snapshot

- **Hop_Lim and node_id wide:**
  - *Hop_Lim*: 1-octet unsigned integer. It is set to the Hop Limit value in the packet at the node that records this data. Hop Limit information is used to identify the location of the node in the communication path.
  - *Node_Id*: 7-octet unsigned integer. Node identifier field to uniquely identify a node within in-situ OAM domain. The procedure to allocate, manage and map the node_ids.

  8-octet field defined as follows:

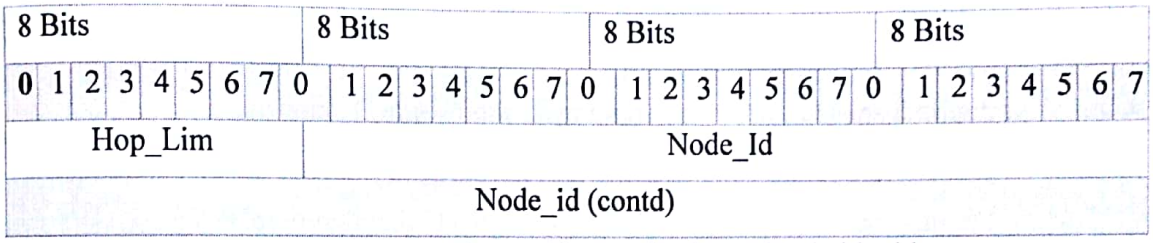


Figure 10: Hop limit and Node id wide

- **Ingress_if_id and Egress_if_id wide:**
  - **Ingress_if_id:** 4-octet unsigned integer. Interface identifier to record the ingress interface the packet was received on.
  - **Egress_if_id:** 4-octet unsigned integer. Interface identifier to record the egress interface the packet is forwarded out of.

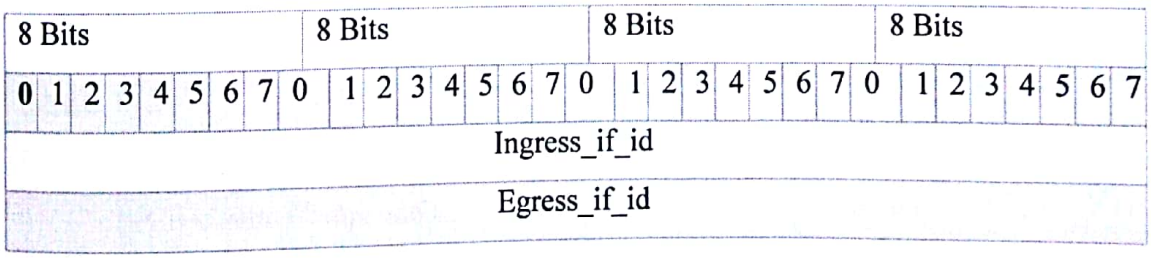


Figure 11: Ingress if id and Egress if id wide

- **App_Data wide**: 8-octet placeholder which can be used by the node to add application specific data. App data represents a "free-format" 8-octed bit field with its semantics defined by a specific deployment.

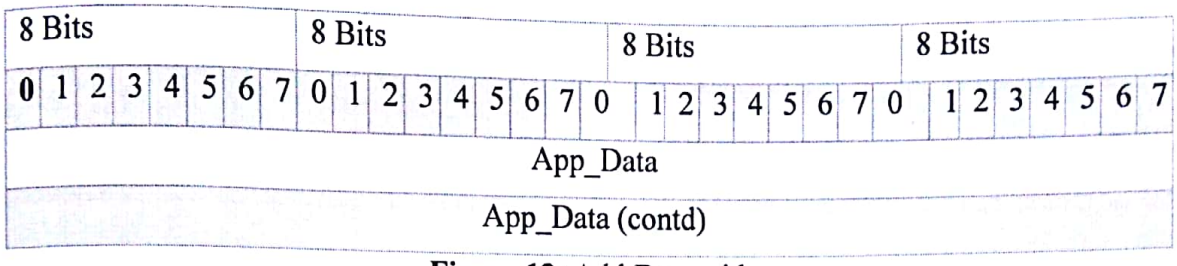


Figure 12: Add Data wide

- **Checksum Complement**: 4-octet node data which contains a two-octet Checksum Complement field, and a 2-octet reserved field. When the Checksum Complement is present, an IOAM encapsulating/transit node adding node data MUST carry out one of the following two alternatives in order to maintain the correctness of the UDP Checksum value:
  1. Re-compute the UDP Checksum field.
  2. Use the Checksum Complement to make a checksum-neutral update in the UDP payload; the Checksum Complement is assigned a value that complements the rest of the node data fields that were added by the current node, causing the existing UDP Checksum field to remain correct.

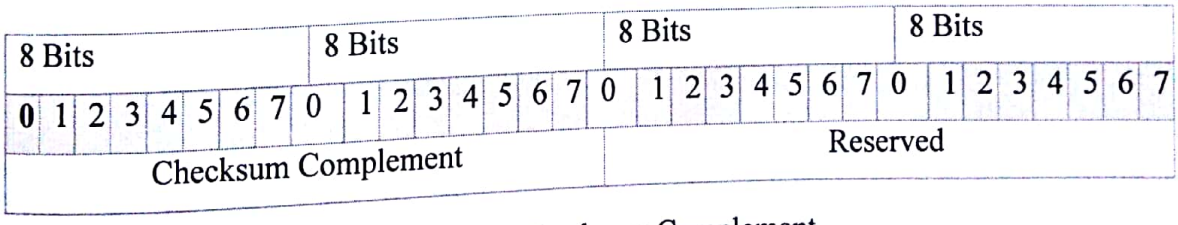


Figure 13: Checksum Complement

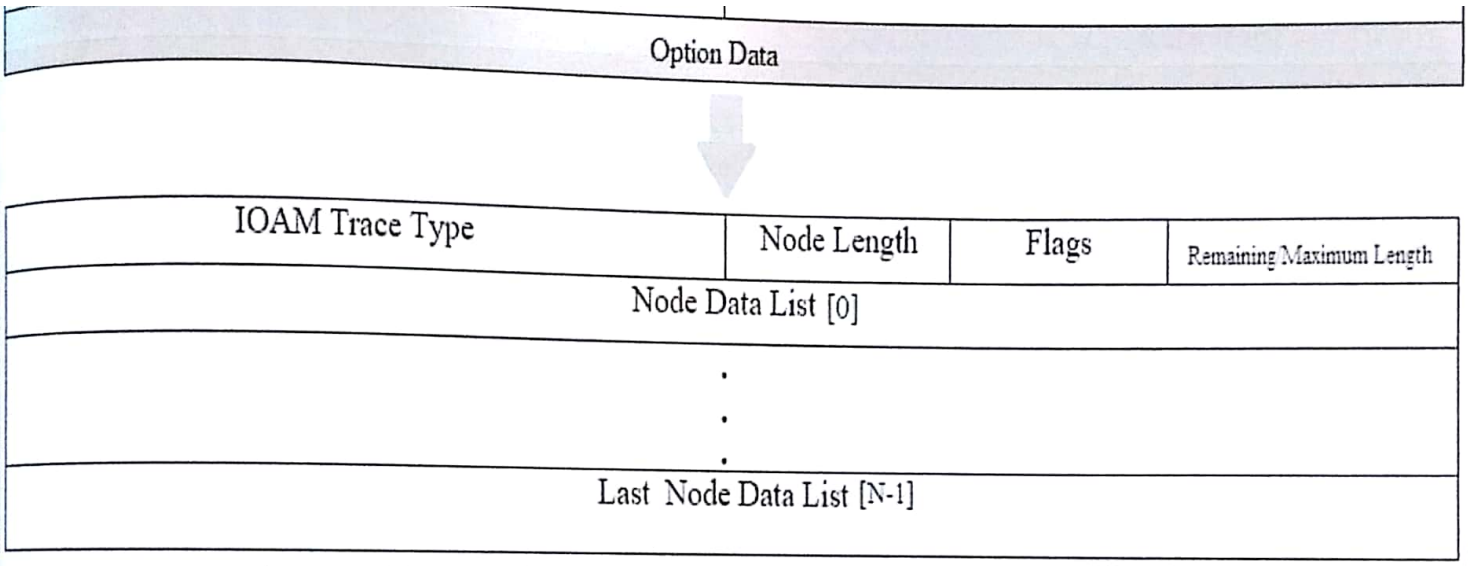
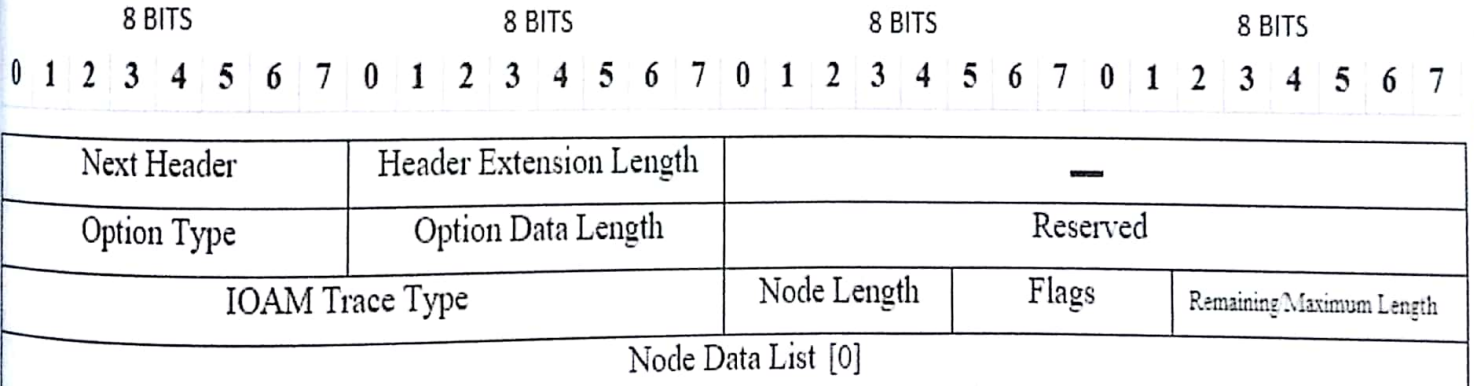


Figure 14: Packet Format of IPv6 Hop-By-Hop Extension Header and IOAM Data fields



2.2 Block Diagram

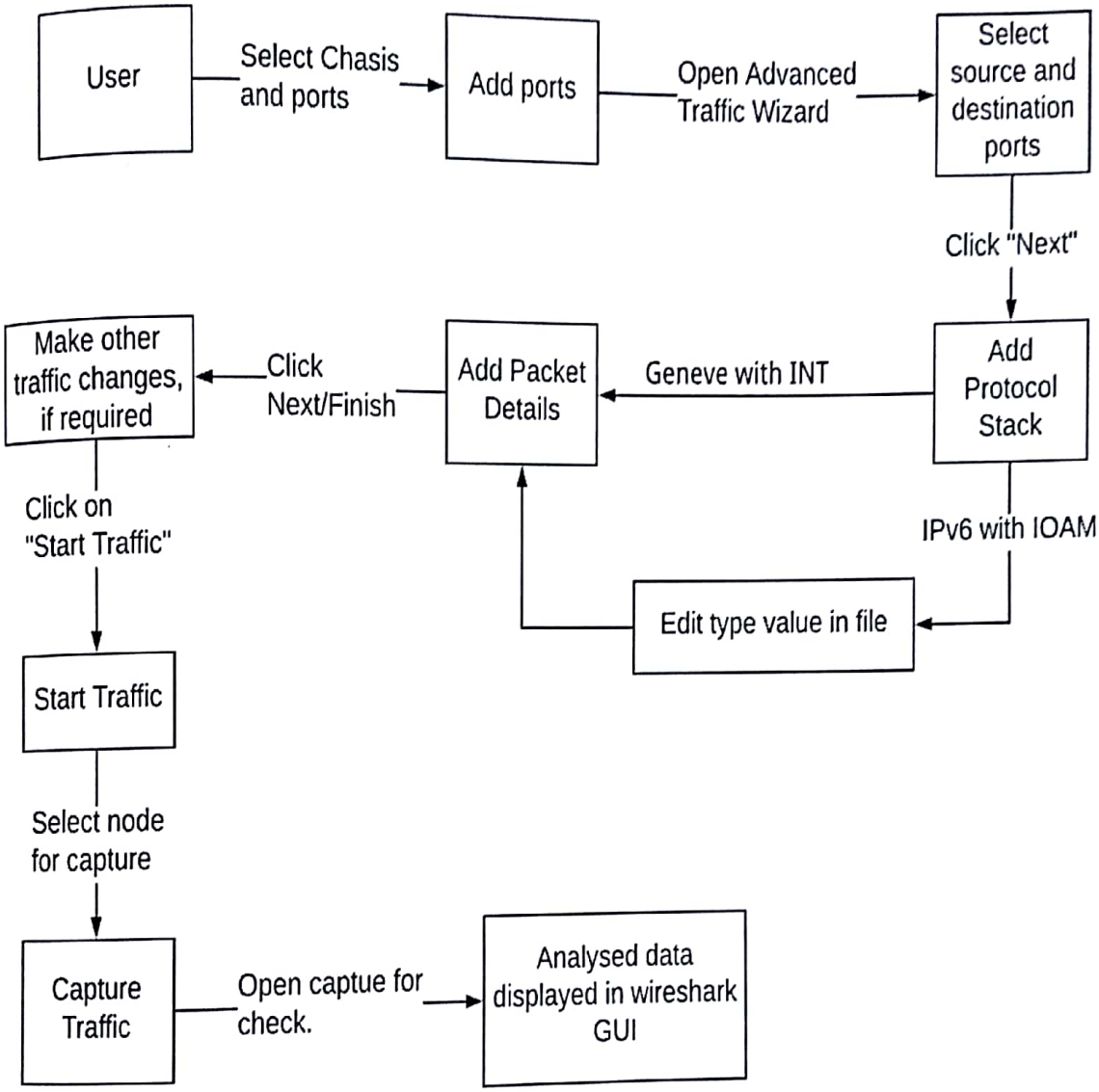


Figure 16: Block Diagram of IxNetwork

2.3 Flowcharts

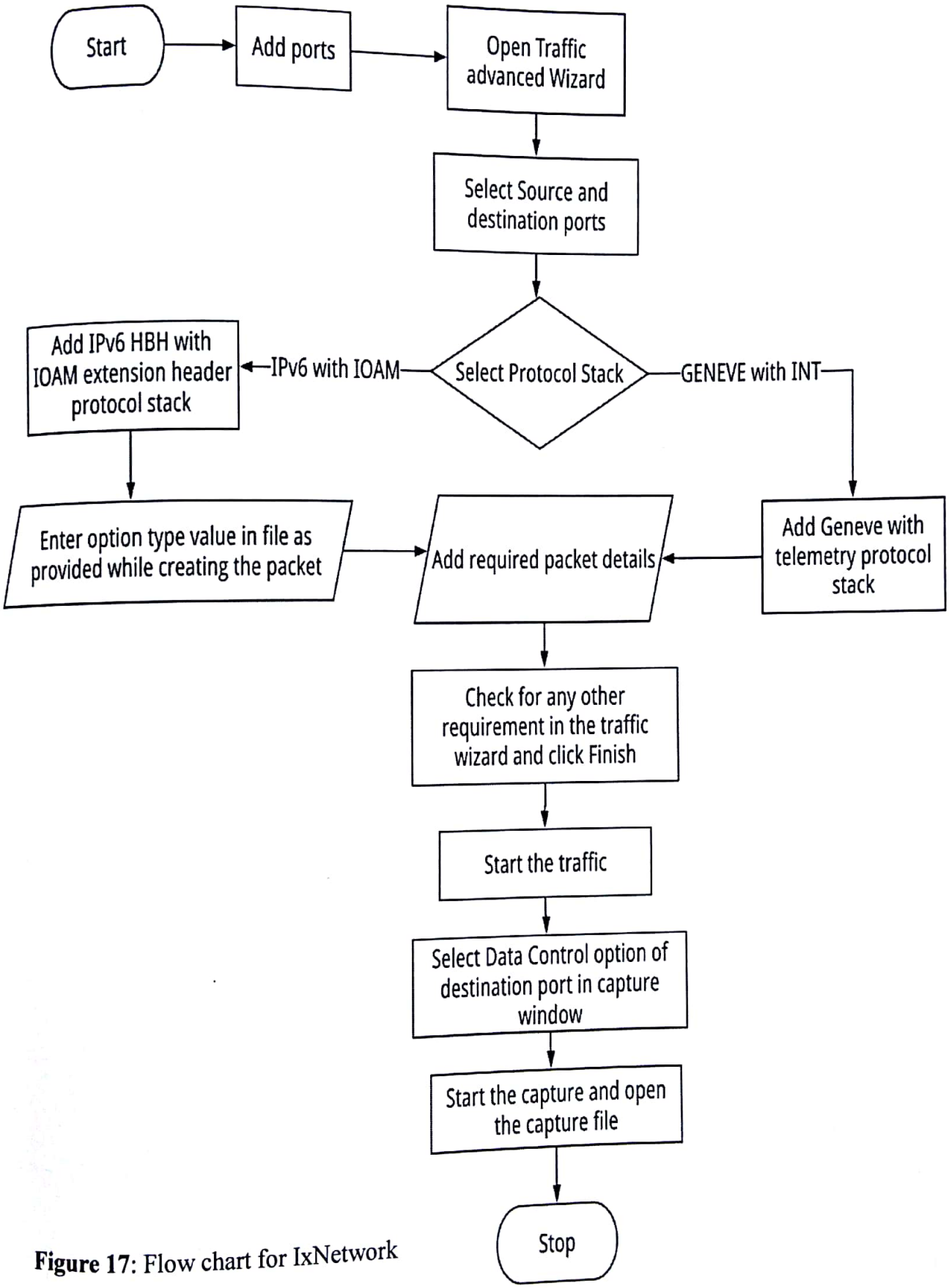


Figure 17: Flow chart for IxNetwork

2.4 Work Flow Diagram

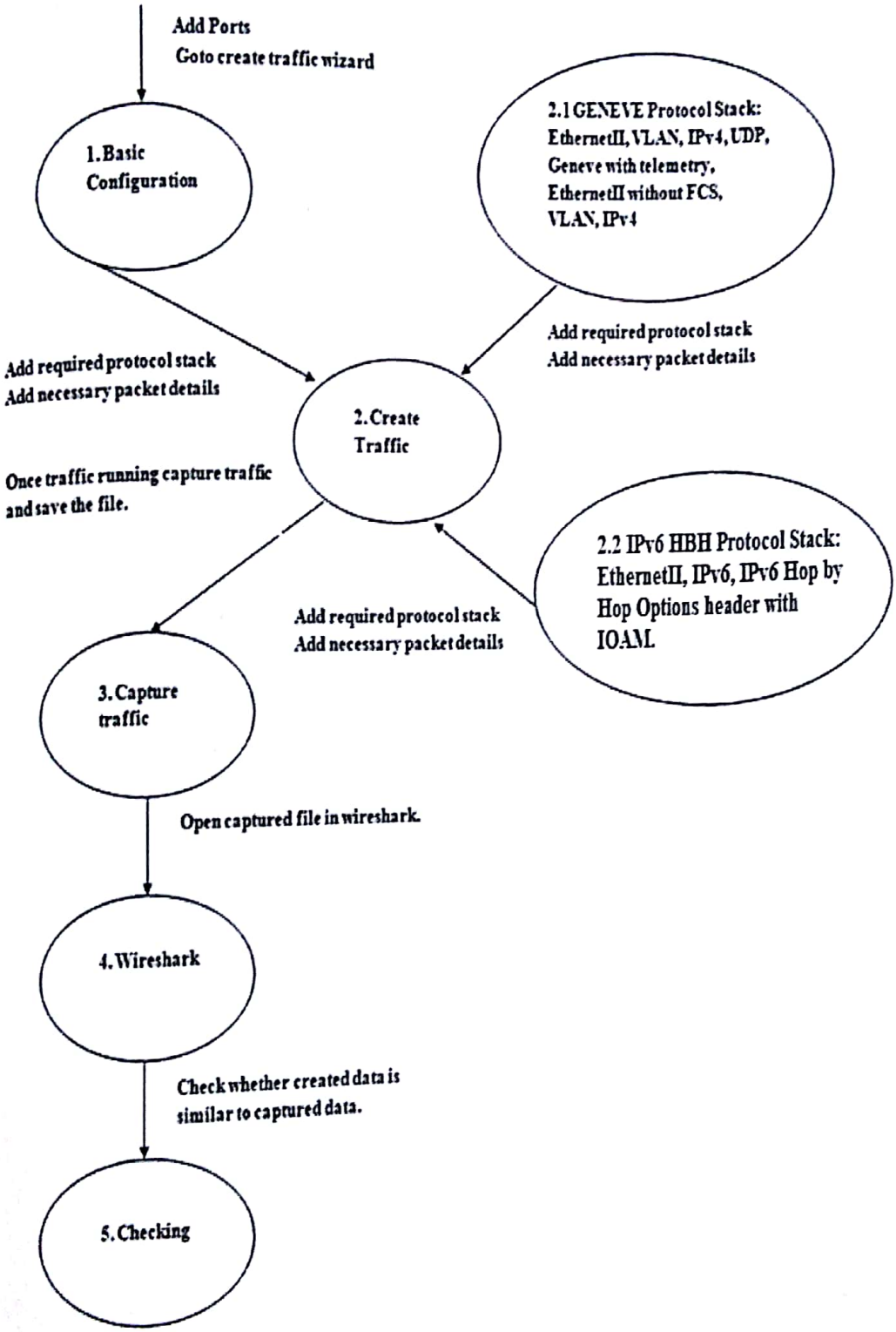


Figure 18: Work flow diagram for IxNetwork

Chapter 3

Detailed Test Plan

3. Detailed Test Plan

3.1 Unit Testing

All the components were tested rigorously with various protocol stacks and different captures were created. Traffic was also created in a variety of ways to get satisfactory output.

Serial No.	Components	Test Case	Output
1.	Ports	Simulate up/down	Ports on/off
2.	Traffic wizard	Create a random traffic	Traffic created and ran successfully
3.	Capture Traffic	Capture any created traffic and save it	Traffic captured successfully.
4.	Wireshark	Open any capture	Packet captured is similar to packet created

Table 2: Unit Testing

3.2 System Testing

The complete system was tested with a lot of scenarios i.e., the traffic was created in a lot of ways to check various cases of packet creation and also to check whether all the changes are being reflected in wireshark.

Serial No.	Components	Test Case	Output
1.	IxNetwork	Create a traffic for INT over GENVE and capture it.	Traffic created successfully and the packets were captured.
2.	IxNetwork	Create a traffic for IOAM data fields over IPv6 and capture it.	Traffic created successfully and the packets were captured.
3.	Wireshark	Open the captured packets	The captured packet details were compared to packet created to see whether its correct or not.

Table 3: System Testing

Chapter 4

Implementation Details

4. Implementation Details

As mentioned earlier, the main objective of this project is to provide telemetry support over the existing protocols, so that the customers can test their respective devices for this feature. This application includes XML files, 2 each for INT and IOAM data fields and wireshark code for dissecting each protocol. Once the packets are created and the wireshark is updated, next step would be traffic creation. While creating the traffic the protocols must be added in a sequential manner. The protocol stack in mention below

INT over GENEVE	**IOAM Data Fields over IPv6 HBH**
EthernetII without FCS	EthernetII
VLAN	Ipv6
Ipv4	Ipv6 HBH Options header with IOAM.
UDP	Ethernet Trailer
Geneve with telemetry	
EthernetII without FCS	
VLAN	
Ipv4	

Table 4: Protocol Stack

7.3 Algorithm for Traffic Generation

Algorithm: Traffic Simulation

Input: Traffic created using new protocols and input values in the field

Output: Captured packets consisting of the protocols added.

Step 1: Start

Step 2: Open IxNetwork

Step 3: Add ports

Step 4: Open Advanced Traffic wizard

Step 5: Select source and destination ports, click next

Step 6: Add the required protocol stack and click next

6.1 If INT over 28eneve is required then the respective stack as mention earlier

6.2 Similarly if IOAM over Ipv6 HBH is required then the respective stack.

Step 7: Make any other change if required, otherwise click finish.

7.1 Select the type of flow group

7.2 Edit the frame and rate setup

7.3 Change the protocol behavior

7.4 Preview the traffic created and then validate

7.5 Click finish or after step 6 you can directly go to step 8 by clicking on finish.

Step 8: Start the L2-L3 traffic

Step 9: Goto captures from the menu

Step 10: Select "Data-Enable" of destination port.

Step 11: Start Capture while the traffic is running.

Step 12: Save the capture file.

Step 13: Open the captured file in wireshark and check whether the captured details matches with the details entered while creating the traffic.

Step 14: Stop

4.2 Algorithm for INT over GENEVE for wireshark support:

Algorithm: Wireshark Dissection

Input: Captured stream of bytes containing required telemetry (INT) traffic.

Output: Structured output reflecting packet details.

Step 1: Start

Step 2: Open the existing 29eneve.c program

Step 3: In function dissect_geneve(agrs):

3.1 Call function dissect_geneve_options(args)

//dissect_geneve(args) ends

Step 4: In function dissect_geneve_options(args):

4.1 Add the sub trees required to display data.

4.2 while (len>0)

4.2.1 Read values: Option class, type and length

4.2.2 Call function dissect_unknown_option(args)

//dissect_geneve_options(args) ends

Step 5: In function dissect_unknown_options(args)

5.1 Call the various functions and pass the required values to represent the data.

5.2 Add the sub trees required to display data.

5.3 Call function dissect_geneve_telemetry(args)

//dissect_unknown_options ends

Step 6: In function dissect_geneve_telemetry(args)

6.1 Call the various functions to display the different packet data.

6.2 check if(totalhopcount>maxhopcount)

6.2.1 display "Total hop count should not be greater than max hop count"

6.3 Add the sub trees required to display data.

6.4 Call function dissect_geneve_instructionbitmap(args)

// dissect_geneve_telemetry ends

Step 7: In function dissect_geneve_instructionbitmap(args)

7.1 Read all the flag bits and store in 30eneve30 format.

7.2 Call the various functions to display the packet data in a tree structure.

7.3 Add the subtrees required to display data.

7.4 for(i = 0; i<totalhopcount; i++)

7.4.1 Add the subtrees required to display data.

7.4.2 If Instrruction bitmap flag bit == SET

7.4.2.1 offset += call dissect_geneve_bitmap_metadata(args)

7.4.3 Repeat step 7.14.2 for all the flags.

//for loop ends

// dissect_geneve_instruction_bitmap ends

Step 8: In function dissect_geneve_bitmap_metadata(args)

8.1 Read the 4-byte value.

8.2 Add required subtree to display the values.

8.3 Call functions to display the packet data in a tree structure.

8.4 Define a switch case to display the respective metadata.

8.5 return 4

// dissect_geneve_bitmap_metadata ends

Step 9: Stop

4.3 Algorithm for Ipv6 Hop-by-Hop Extension Header over IOAM data fields for wireshark support:

Algorithm: Wireshark Dissection

Input: Captured stream of bytes containing required telemetry (IOAM Data Field) traffic.

Output: Structured output reflecting packet details.

Step 1: Start

Step 2: Open the existing ipv6.c program

Step 3: In function dissect_opts(args)

3.1 Call the various functions to represent the various data in a tree structure.

3.2 when defining "option type" field call the following functions

3.2.1 Call function read_incremental_type_value_from_file(args)

3.2.2 Call function dissect_option_type(args)

3.3 In switch case. Add two cases for pre-allocated and incremental type option

3.3.1 for each case call function dissect_incre_tracing_option(args)

3.3.2 break;

// dissect_opts ends

Step 4: In function read_incremental_type_value_from_file(args)

4.1 Read the two values from the file.

4.2 Store the values in 2 global variables (incremental and pre-allocated).

Read_incremental_type_value_from_file(args) ends

Step 5: In function dissect_option_type(args)

5.1 Compare captured option type value with predefined option type values and also with the two-global variable.

5.2 return the respective type value defined.

5.3 If none matches, return -1;

// dissect_option_type(args) ends

Step 6: In function dissect_incre_tracing_option(args)

6.1 Call the various functions to represent the various data in a tree structure.

6.2 Add the required sub trees to display the data.

6.3 Store all the IOAM Trace Type flag values.

6.4 If the Option Type is "preallocated"

6.4.1 After displaying Remaining Length, display the pre-allocated empty stack.

6.5 Calculate the node_len and no of hops.

6.6 while (node_offset> offset)

6.6.1 Add the subtrees required to display data.

6.6.2 If IOAM Trace Type flag bit == SET

7.4.2.1 offset += call ioam_node_data(args)

6.6.3 Repeat step 6.5.2 for all the flags.

//while loop ends

6.7 return offset

Step 7: In function ioam_node_data(args)

7.1 Add the required subtrees to display the required data.

7.2 switch (node_type)

7.2.1 Display the meta data depending on the IOAM Trace Type.

7.2.2 Store the length in bytes displayed in say, "return_val"

7.2.3 Repeat step 7.2.1 till all the types are covered.

7.3 return return_val;

// ioam_node_data(args) ends

Chapter 5
Results and Discussions

5. Results and Discussions

This project has been successfully completed with a few features to be added that will be incorporated in it as an update for the release. The user can create traffic using the newly added in the existing packets. IxNetwork is also able to capture this traffic and wireshark is successfully displaying all the updated fields properly as required.

The main aim of this project is to provide a customer support for the telemetry. Since telemetry is not there in the current market, so with this project, the networking companies can test their networking devices for telemetry support. This project involves creation of virtual traffic which contains our telemetry fields encapsulated over different protocols. This virtual traffic is ran across two ethernet ports which acts as a source and destination port (Fig 7).

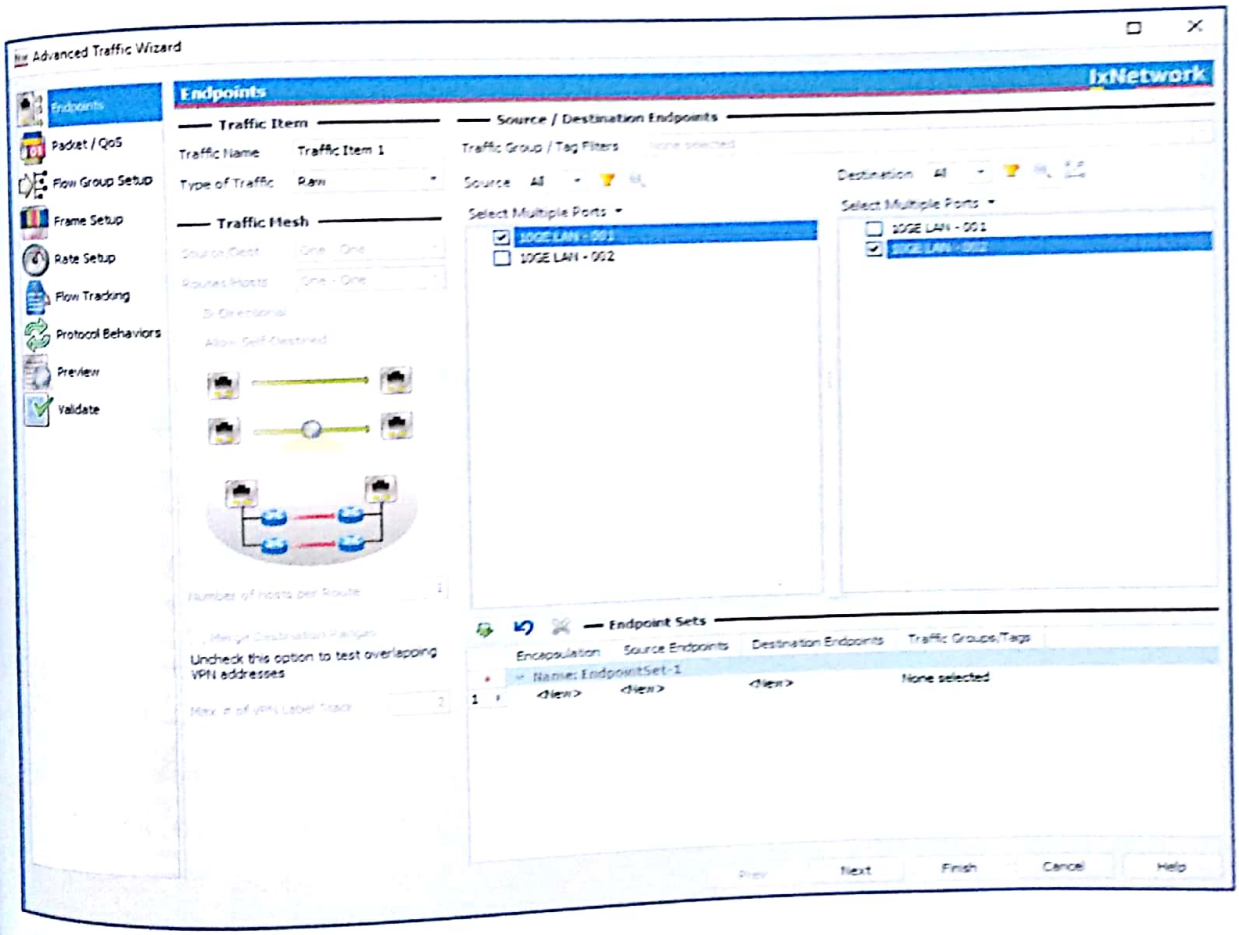


Figure 19: Advanced Traffic wizard (Endpoints)

Once the traffic is created, user can capture the packets that were transmitting as traffic in the selected ports. Once captured these capture files can be open and checked for output.

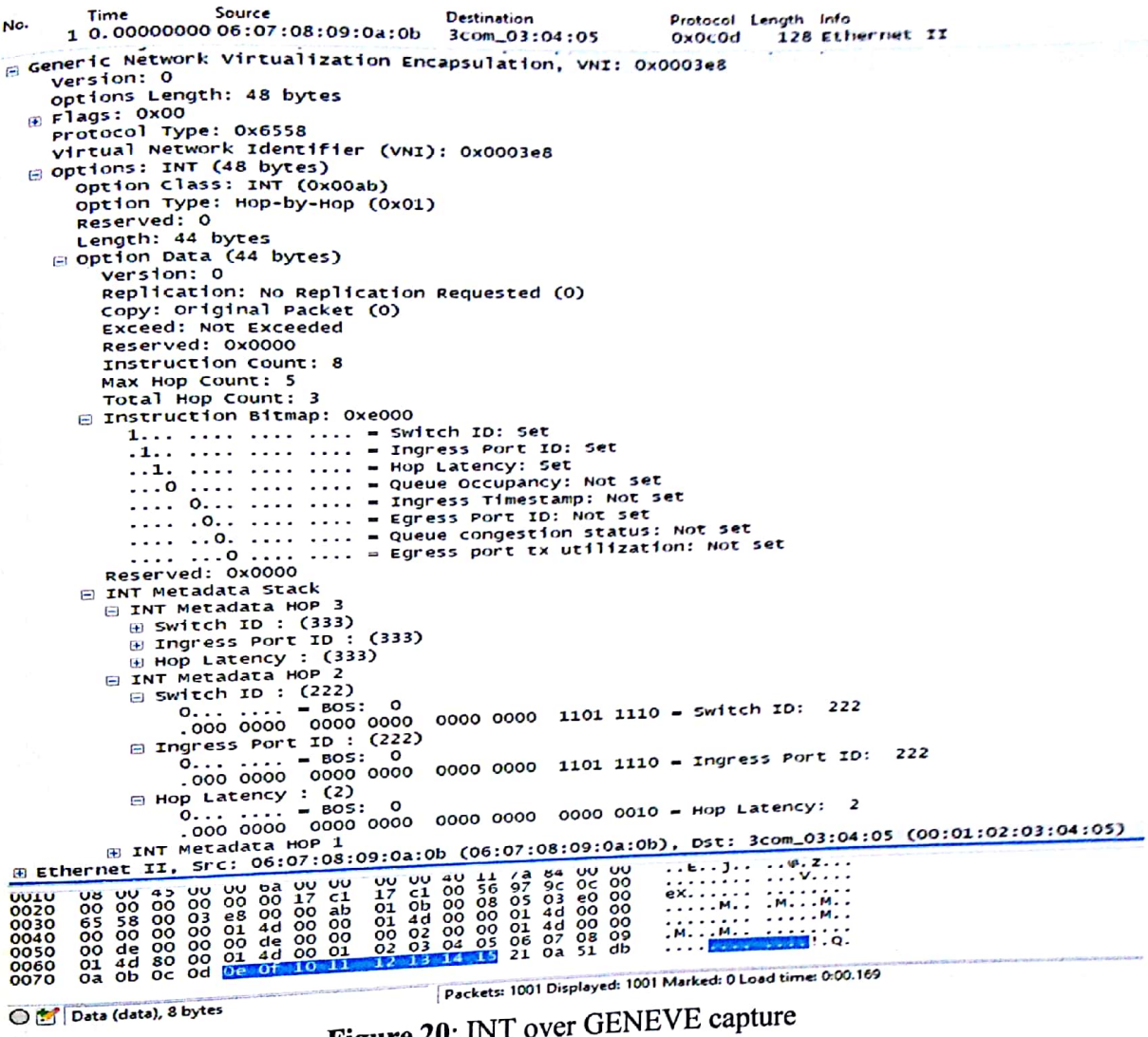


Figure 20: INT over GENEVE capture

The above figure (fig 8) shows the captured details of an individual packet from the traffic. In the pic we can see a tree named "Option Data", from this field our telemetry fields start. One thing that is to be taken care of is the protocol stack that is mentioned in table 14, if proper stack is not imported, then after capturing the packet, wireshark will display all the details correctly but it will also give an error of malformed packet. An figure displaying what happens incorrect stack is added is reflected in figure 9. We see that all the fields that we added while creating the traffic are reflected in the capture but at the end it displays an error.

```
Frame 1: 166 bytes on wire (1328 bits), 166 bytes captured (1328 bits)
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
802.1Q Virtual LAN
Internet Protocol Version 4, Src: 0.0.0.0 (0.0.0.0), Dst: 0.0.0.0 (0.0.0.0)
User Datagram Protocol, Src Port: 6081 (6081), Dst Port: 6081 (6081)
Generic Network Virtualization Encapsulation, VNI: 0x0003e8
  Version: 0
  Options Length: 108 bytes
  Flags: 0x00
  Protocol Type: 0x6558
  Virtual Network Identifier (VNI): 0x0003e8
  Options: INT (108 bytes)
    Option Class: INT (0x00ab)
    Option Type: Hop-by-Hop (0x01)
    Reserved: 0
    Length: 104 bytes
    Option Data (104 bytes)
      Version: 0
      Replication: No Replication Requested (0)
      Copy: Original Packet (0)
      Exceed: Not Exceeded
      Reserved: 0x0000
      Instruction Count: 8
      Max Hop Count: 5
      Total Hop Count: 3
      Instruction Bitmap: 0xff00
      Reserved: 0x0000
      INT Metadata Stack
        INT Metadata HOP 3
        INT Metadata HOP 2
        INT Metadata HOP 1
[Malformed Packet: Ethernet]
  [Expert Info (Error/Malformed): Malformed Packet (Exception occurred)]
    [Message: Malformed Packet (Exception occurred)]
    [Severity level: Error]
    [Group: Malformed]
```

Figure 21: INT over GENEVE capture (incorrect protocol stack)

Next, we captured the traffic consisting of IPv6 HBH protocol stack. The fields are not similar to INT though their purpose is same. Similar to instruction bitmap, IOAM consists of IOAM trace type, also IOAM has 12 metadata fields defined, whereas INT has 8 fields defined. One major difference between the two is that both their metadata are of different sizes. In case of INT all the metadata are of 4 bytes whereas IOAM's metadata can be of 4 or 8 bytes and one of the metadata can be varied length. IOAM is of two types:

1. Incremental Type Tracing: In incremental, the metadata is appended while creating the traffic and no free space is left (Refer fig 10).
2. Pre-Allocated Type Tracing: In pre-allocated, the size is fixed and there might be free space in the packet which is reflected by Remaining length field, depending on this field pre-allocated stack is included in the packet while creation. (Refer fig 11)

```
Hop-by-Hop Option
    Next header: IPv6 no next header (59)
    Header Extension Length: 9 (80 bytes)
  IPv6 Option (Incremental Tracing Option)
    Option Type : Incremental Tracing Option
    Option Data Length: 70 bytes
    Reserved: 0x0000
  Option Data (68 bytes)
    IOAM Trace Type
      1... .... .... .... = Hop Limit - Node Id (Short): Set
      .1.. .... .... .... = Ingress If Id - Egress If Id (Short): Set
      ..1. .... .... .... = Timestamp Seconds: Set
      ...1 .... .... .... = Timestamp Subseconds: Set
      .... 1... .... .... = Transit Delay: Set
      .... .1.. .... .... = App Data (Short): Set
      .... ..1. .... .... = Queue Depth: Set
      .... ...0 .... .... = Opaque State Snapshot: Not set
      .... .... 0... .... = Hop Lim - Node Id (Wide): Not set
      .... .... .0.. .... = Hop Lim - Node Id (Wide): Not set
      .... .... ..0. .... = App Data (Wide): Not set
      .... .... ...1 .... = Checksum Complement: Set
    Node Length: 8 (32 bytes)
  Flags
    Maximum Length: 0 (0 bytes)
  IOAM Node Data 2
    Hop Limit - Node Data (Short)
      Hop Limit: 2
      Node ID: 2
    Ingress If Id - Egress If Id (Short)
      Ingress If Id: 2
      Egress If Id: 2
    Timestamp Seconds
      Timestamp Seconds: 2
    Timestamp Subseconds
      Timestamp Subseconds: 2
    Transit Delay
      0... .... .... .... .... .... .... .... = Overflow: Not set
      Transit Delay: 2
    App Data (Short)
      App data: 2
    Queue Depth
      Queue Depth: 2
    Checksum Complement
      Checksum Complement: 2
      Reserved: 0x0000
  IOAM Node Data 1
IPv6 Option (Pad1)
IPv6 Option (Pad1)

0000  00 00 00 00 00 00 00 00  00 00 00 00 86 dd 60 00   ........ ......`.
0010  00 00 00 50 00 40 00 00  00 00 00 00 00 00 00 00   ...P.@.. ........
0020  00 00 00 00 00 00 00 00  22 46 00 00 fe 10 40 00   ........ "F....@.
0030  00 00 00 00 00 00 3b 09  00 00 00 02 00 00 00 02   ......;. ........
0040  02 00 00 02 00 02 00 02  00 00 00 02 00 02 00 00   ........ ........
0050  00 00 00 02 00 00 00 02  00 00 00 02 00 00 00 02   ........ ........
0060  02 00 00 02 00 02 00 02  00 00 00 02 00 02 00 00   ........ ........
0070  00 00 00 02 00 00 00 02  02 00                     ........ ..
0080  00 00 00 00 00 00 82 2c

File: "K:\Project\IOAM\Test Case\Case 1 (inc...  Packets: 14014 Displayed: 14014 Marked: 0 Load time: 0:00.043
```

Figure 22: IOAM over IPv6 HBH capture (Incremental type)

```
Frame 1: 114 bytes on wire (912 bits), 114 bytes captured (912 bits)
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 6, Src: :: (::), Dst: :: (::)
  0110 .... = Version: 6
  .... 0000 0000 .... .... .... .... .... = Traffic class: 0x00000000
  .... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
  Payload length: 56
  Next header: IPv6 hop-by-hop option (0)
  Hop limit: 64
  Source: :: (::)
  Destination: :: (::)
  [Source GeoIP: Unknown]
  [Destination GeoIP: Unknown]
Hop-by-Hop Option
    Next header: IPv6 no next header (59)
    Header Extension Length: 6 (56 bytes)
  IPv6 Option (Pre-Allocated Tracing Option)
    Option Type : Pre-Allocated Tracing Option
    Option Data Length: 50 bytes
    Reserved: 0x0000
    Option Data (48 bytes)
      IOAM Trace Type
        Node Length: 3 (12 bytes)
      Flags
        Remaining Length: 2 (8 bytes)
        Pre-Allocated Empty Stack: 0
        Pre-Allocated Empty Stack: 0
      IOAM Node Data 3
        Hop Limit - Node Data (Short)
          Hop Limit: 3
          Node ID: 3
        Ingress If Id - Egress If Id (Short)
          Ingress If Id: 3
          Egress If Id: 3
        Timestamp Seconds
          Timestamp Seconds: 3
      IOAM Node Data 2
      IOAM Node Data 1
  IPv6 Option (Pad1)
    Option Type : Pad1
    Pad1

0000  00 00 00 00 00 00 00 00  00 00 00 00 86 dd 60 00   ........ ......`.
0010  00 00 00 38 00 40 00 00  00 00 00 00 00 00 00 00   ...8.@.. ........
0020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 3b 06  2f 32 00 00 03 00 03 00   ......;. /2......
0040  00 00 00 00 00 00 00 02  03 00 00 02 00 00 18 02   ........ ........
0050  00 00 00 03 02 00 00 01  00 02 00 02 00 00 03 00   ........ ........
0060  01 00 00 01 00 01 00 01  00 00 00 01 [00 00] 7a f9   ........ ....[..]z.
0070  de 42
```

Figure 23: IOAM over IPv6 HBH capture (Pre-Allocated type)

Chapter 6

Summary and Conclusion

6. Summary and Conclusion

6.1 Summary of Achievements

After the successful creation of the new packets, integrating them with the IxNetwork and updating the wireshark, following were the achievements:

- Successfully build the environment for wireshark development for editing the existing dissectors.
- Successfully updated the dissectors for INT and IOAM support.
- Registered the modified dissectors in wireshark library.
- Using this newly added packet we can now store the intermediate devices information and use it for different purposes.
- We have two different protocols that are updated to provide support for this feature.
- Testing phases was successful for almost all the test cases i.e., the output was as expected.

6.2 Difficulties encountered

The major difficulties that were encountered during this project is mentioned below:

- Building the wireshark environment. Wireshark does not let anyone update its library. Proper environment must be setup, a lot of softwares is to be installed. User may encounter of problem while installing.
  All the major problems were tackled using Cygwin which provided a lot of features for the environment setup and also by referring an installation manual provided by manager.
- While updating the wireshark library, the existing programs were modified, in doing so, the programmer had to understand the previous code written by someone else and then modify it to meet its own requirement. There were a few bugs while integrating it with the new code. Those were finally removed by proper debugging of the whole program line by line.
- There were some fields which were dynamic as in the number of fields is not defined. But while creating a packet format every field has to be predefined. If a field occurs 10 times, then in the xml code, the tag adding that particular field needs to be written 10 times. This was solved much later by adding a "recurse" tag.

6.3 Limitation of the project

The limitations of this project:

- User must have knowledge of IxNetwork for traffic creation and wireshark for traffic analysis.
- There is no checking on the metadata whether correct metadata is used to add the values, like there was a like cases were suppose we have last 3 flags in instruction bitmap and while adding the value in the metadata, we added the values in the first 3 fields. Wireshark will still display the output with respect to the last 3 fields.
- The value of option type in case of IOAM data field is not yet defined. Both the incremental and pre-allocated type can have value from 0x20 to 0x3f. So, for proper working, user need to update the type value in a file provided by the company. This file contains the current value the option type that is being provided while creating the traffic.
- A lot of fields requires automation, and there are areas which needs to be optimized for better user interaction.

6.4 Future scope of the project

This project has the following future scope:

- There are some flags which are unassigned in both the packets (INT and IOAM). Once the test phase of this packet is successful those fields can be updated so that more metadata can be stored from the intermediate devices.
- Some of the fields can to be automated for IxNetwork to provide a better UI.

Chapter 7

Gantt Chart

7. Gantt Chart

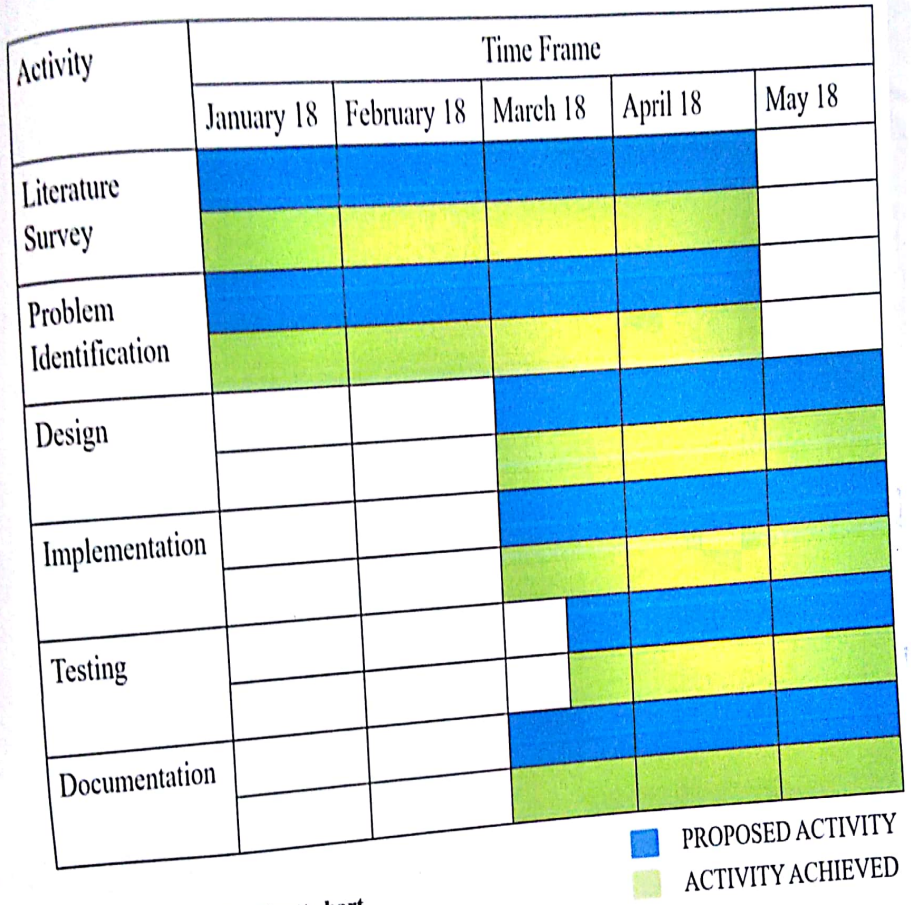


Figure 24: Gantt chart

References

7. References

[1] In-band Network Telemetry (INT) by Hugh Holbrook: Arista; Anoop Ghanwani: Dell; Dan Daly: Intel
[Online, Published in June, 2017]

[2] Encapsulations for In-situ OAM Data [RFC Draft], http://datatracker.ietf.org/drafts/current/
[Online, Published in January, 2018]

[3] Packet format for IOAM Data fields [RFC Draft], http://datatracker.ietf.org/drafts/current/
[Online, Published in January, 2018]

[4] Readme.docx from the perforce library (ssl: ixin: vmp4proxy) [IXIA server since 2010]