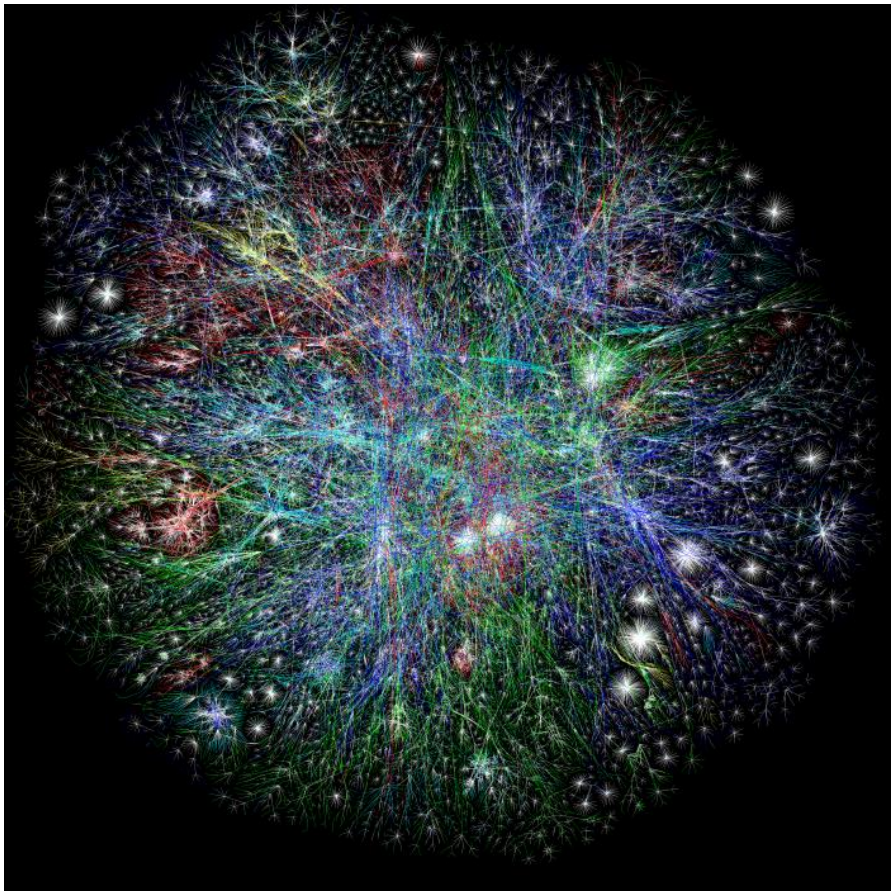

Novel Techniques for Efficient Monitoring of Connectivity Services

A thesis work at Ericsson Research in Stockholm

Master's thesis in Networks and Distributed Systems

ARMIN TIRDAD

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2020

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY AND
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden
www.chalmers.se

Master's thesis 2020

Novel Techniques for Efficient Monitoring of Connectivity Services

A thesis work at Ericsson Research in Stockholm

ARMIN TIRDAD

CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF
GOTHENBURG

Novel Techniques for Efficient Monitoring of Connectivity Services
A thesis work at Ericsson Research in Stockholm
Armin Tirdad

Supervisor: Catalin Meirosu, Ericsson Research
Examiner: Marina Papatriantafilou, Department of Computer Science and Engineering

Master's Thesis 2020
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The Internet 2003. The Opte Project, created in 2003 by Barrett Lyon, seeks to generate an accurate representation of the breadth of the Internet using visual graphics. © 2020 by LyonLabs, LLC and Barrett Lyon. http://www.opte.org/the-internet/

Gothenburg, Sweden 2020

Abstract

A network connectivity service provider commonly provides point-to-point Ethernet connectivity services over its managed-network domain to customers in a metropolitan area network. These services normally require a certain quality of service related to various connectivity parameters. As metro networks grow in size and complexity, network service providers are seeking for more automated and convenient ways of network connectivity services provisioning and monitoring of service level specifications.

This thesis work has presented an architecture and implementation (called in-network module), at the network infrastructure layer, which helps to automate the monitoring of a provisioned point-to-point Ethernet service in a distributed manner. For each connectivity service, the monitoring is done by configuring and triggering active measurements of available bandwidth, and aggregating and communicating the measurement results to a central entity in the architecture. The active measurement is handled by an existing bandwidth measurement tool called BART (Bandwidth Available in Real-Time).

Our in-network module fulfills requirements such as coordination for triggering available bandwidth measurements, aggregating the results in a distributed way, and communicating with the central management module.

Acknowledgements

This thesis would not have been possible unless I had a kind and positive support from my supervisors at Ericsson Research. I would like to especially thank Catalin Meirosu, for his continuous and constructive guidance, who utilized his concrete knowledge to help me in the areas this thesis covers while he was always accepting my sudden queries at his office. I learned valuable technology concepts from his knowledge transfer. I also would like to thank Andreas Johnsson for his kind support and help in the specific technology aspects covered in this thesis work. Hereby, I like to thank my university advisor, Marina Papatriantafilou, for her guidance and support during the thesis work.

I'd also like to thank Ericsson Research for this opportunity to work with cutting-edge concepts and technologies and learn simultaneously.

Dedication

I dedicate my thesis work to my family and many friends. I'm extremely thankful to my loving parents for their support during my studies, especially my mom who kindly encouraged me to finalize this thesis. A big thank you to my brothers especially Ali for giving me excellent consultations on academic writing. I'd like to thank my amazing friends who reminded me of finalizing this thesis every now and then.

Contents

1. Introduction

Today's networking systems are becoming more complex, delivering multi technology-enabled services to the customers and utilizing equipment from various vendors. In addition, today's metro network is becoming a converged network of fixed and wireless infrastructures and the converged network will carry both fixed and mobile traffic.

A network operator that owns or hires network infrastructure in a metropolitan territory, normally provides point-to-point and point-to-multipoint connectivity services (VPNs – Virtual Private Networks). These connectivity services can be delivered on various networking or OSI (Open Systems Interconnection) model layers, but they are normally delivered as OSI layer-2 (data link) or OSI layer-3 (network) connectivity. On the other hand, since a layer-2 connectivity service has higher flexibility for the customer to perform network address allocation, this type of service is the most favorable one. As Ethernet is one of the most common layer-2 protocols used today, providing an end-to-end Ethernet connectivity service (Ethernet VPN) over the provider's network infrastructure is highly practiced.

At the core of the network infrastructure, managed by the service provider (operator), MPLS (Multiprotocol Label Switching) is highly utilized as a protocol (mechanism) to provide routing/switching functionalities between various in-network nodes. MPLS accelerates packet forwarding by directing data between adjacent network nodes based on virtual path labels between these nodes, instead of routing based on network addresses in traditional network routing.

Service provisioning generally consists of preparing, equipping, and initializing a network for providing a service. The operator is usually willing to automate the process of provisioning different services; however this process is done almost manually today. Also they want to monitor the provisioned services status to make sure that service level agreements (SLAs) are not becoming violated. An SLA is an official assurance agreed between a service provider and a client. Specific aspects of the service such as quality, availability, and responsibilities are contracted between the two parties. The most important part of SLA is that the service provider should deliver according to the contract.

The requirements regarding monitoring of the provisioned services are addressed in this thesis work.

1.1.1 Problem statement

In telecommunications, a point-to-point (P2P) connection is a two way communication between two nodes. This definition is in contrast with a point-to-multipoint in which many nodes communicate with one node [24]. P2P connection might pass through other nodes to make a path and link the endpoints. Network provisioning is the process of delivery of the customer's services via the network elements. The network operator is required to provide quality of service according to the SLA.

The first objective of this thesis is to determine how to coordinate measuring the bandwidth of provisioned point-to-point service to make sure that the SLA is not violated. Note that bandwidth is typically one of the SLA parameters. It is also desired to have the functionality of automatic configuration for the measurements.

The secondary objective of the thesis is how to transmit the results of measurements toward a centralized entity in an efficient way. An example of the centralized entity can be a management node that provides monitoring functionality to the operator. We wanted to reduce the amount of information being presented to the user regarding network and circuits status, compared to the detailed measurement results which include every single circuit segment.

1.1.2 Contributions

The contributions have been made by this thesis were in the context of state of the art of the time of conduction (2012). In this thesis, we provide a new service validation method to facilitate available bandwidth measurements for a point-to-point Ethernet service. This validation service, integrated with the provisioning service, improves network visibility via monitoring reports, and helps to reduce the operator’s intervention. It is a part of a larger architecture project which presents a generic system for integrating and provisioning and validation of Ethernet services over transport networks with the objective of minimizing the intervention from the human operator.

During the thesis work at Ericsson Research, the operational contributions below were handled by this work which will be explained in detail in chapter 4:

- Test platform setup: host operating system installation, configuration for remote access, VM config/troubleshooting, virtual networking config/troubleshooting).

- Design of In-Network Module (INM) and its sub-modules, according to the problem statement requirements and deriving some mechanisms from existing academic work.
- Implementation of In-Network Module: developed using the Java language.
- INM functionality verification, troubleshooting and documentation.
- Execution of whole test platform setup to gather and evaluate measurement results.

2. Background, Motivation and Related Work

Traditionally, network planning, provisioning and monitoring has been handled as separate work processes for the providers of metro and long-haul network connectivity. There are basic capabilities of service validation that is integrated into the provisioning mechanisms by major manufacturers. However, these mechanisms usually require a complicated configuration of various connectivity service properties. In most cases, the service validation is done by scripts which perform basic connectivity tests. The service providers are willing to reduce the complexity of their daily operations [1].

2.1.1 Service provider challenges

The service providers primarily deal with management complexity of Ethernet connectivity services, which are defined in the Metro Ethernet Forum MEF 6.1 specification [2]. Frame Relay and ATM are being replaced by these services at the layer where the connectivity service is delivered to the customer. For underlying infrastructure to support Ethernet services, a popular choice is to use MPLS-TP [3] standardized in IETF [1]. Using Ethernet at the service layer helps customers to have more flexibility over their network protocols and addressing.

In accordance with the need for Ethernet overlay services, Metro Ethernet Forum (MEF) has developed a combination of a technical and a marketing forum to accomplish its goals. The Ethernet service definition and technical specifications developed by MEF help service providers and customers utilizing these end-to-end services to have interoperability between each other [17].

For the purpose of service validation a common choice is to measure the circuit connectivity parameters mentioned in the specific SLA. There are various techniques for measuring these parameters.

2.1.2 Service level agreement and its management

According to Verma [4] service level agreement (SLA) is a formal description of the relationship between a service provider and customer. SLA can be utilized in the context of various industries and specifies what the customer should expect from the provider, the obligations of the provider and also the customer. It specifies the performance, availability and security parameters of the service as well as the procedures to be executed to make sure about compliance with SLA. When corporations outsource functions which are outside their own scope, SLAs are utilized. Many companies outsource the task of operation and maintenance of computer networks to third-party infrastructure providers, which makes SLA support an important issue in the computer networks area.

A service level agreement can contain various information like a description of the nature of the provided service, the expected performance level, the procedure for reporting problems, the time-frame for problem resolution, the process for monitoring and reporting the service level, the consequences for the provider not meeting its commitments and escape clauses and constraints. The support of proper level of performance and availability is a vital side of the operation of an enterprise. The service provider needs to monitor the performance metrics of the network as well as the operating ranges of the metrics, based on the SLA for each customer. Periodic reports on monitored performance must be available for inspection by the customer. In the case of SLA objectives not being met, the network configuration must be changed via the service provisioning process. At the last step, the operator can examine whether the agreed SLA can be met by experience and if not, he may correct the performance objectives to be feasible to meet. He can also edit the performance objective to become stricter if it can attract new revenues. [4]

Three common approaches are utilized to support and manage an SLA within IP environments: insurance, provisioning and adaptive approach. The most complete one is the adaptive approach which assumes different service levels are offered in contrast with the insurance one and employs adaptive configuration for the provisioning approach. In the case of network connectivity SLAs, all three approaches to SLA support are being employed in various providers.

In an insurance approach, the provider (ISP) assures an upper threshold on the delay and loss-rate between any pair of access routers, considering the average across all of the access points and over a sensible time period. The ISP would need to implement a monitoring plan to measure the mentioned network transport parameters. In a provisioning approach, the provider needs to supply different latencies or loss-rates to various customer services which are sharing the same network infrastructure or links. For this to be satisfied, the providers would need to employ techniques like Differentiated Services or MPLS traffic engineering to be able to

provision the networks in such a way that they comply with SLA targets. Such provisioning of services are done using tools and schemes that tend to be manual in most cases. In an adaptive approach, the network operator may use technologies like dynamic bandwidth provisioning in optical networks to dynamically supply more bandwidth on its circuits as the traffic load rises [4]. Examples of types of SLAs offered by commercial providers are available online [5]-[6].

2.1.3 Decentralized network management

The difference between traditional, centralized systems and decentralized ones is that the decentralized system runs a management operation as a distributed (instead of a centralized) process. Networks are getting larger and more dynamic, which will require more adaptive and decentralized control [7].

Lim et al. [7] call their approach pattern-based management. It is concentrated on navigation patterns based on graph traversal algorithms. Some of the advantages of this approach this is that the performance and scalability of the management programs can be analyzed formally, and that the semantics and distributed processing aspects of a management operation can be kept apart. Also, it allows fault tolerance to be programmed into a pattern, so that there is no need for a programmer to deal with faults. It also enables a finer control of code mobility.

A pattern-based management program consists of two components: a pattern and an aggregator. The execution of a pattern has two phases: expansion and contraction. An expansion can start in any node, which sends explorer messages to its neighbors, until the leaves of the spanning tree are reached. Then, the contraction phase starts from the leaf nodes, which send echo messages to their parents, until the start node of the pattern is reached. An aggregator contains the logic for performing local management options. The execution of an aggregator is controlled by the pattern. [7]

Network management systems collect status and statistical information from network devices, and process them in order to implement global management tasks aligned with management goals. As this management is a global operation, the management system normally deals with global parameters which are functions of the devices' local variables, instead of watching for individual device variables. For example, an average load of a specific IP protocol would be more interesting than the load on individual network links. Global parameters used by management units are mostly aggregate functions, like sum, min, max and average of device-specific counters. One of the most common distributed in-network aggregation methods is based on allocating a spanning tree in the management network topology [21, 22, 23]. Dam and Stadler have developed GAP (generic aggregation protocol), a novel mechanism that calculates

aggregates of device variables to be used for network management operations. GAP can continuously estimate the network aggregates while local variables and the network graph are prone to change. A key feature of this protocol is that it computes aggregates in a decentralized fashion using an aggregation tree [8]. Our solution to aggregation problem has utilized parts of the GAP protocol and mechanisms.

2.1.4 Methods for decentralized network management

The methods and techniques for designing and implementing decentralized networks have their roots in large systems decentralized control theory and computer network technology. The important factors in these methods are stability, instantaneousness, accuracy, energy efficiency, and robustness.

Prieto and Stadler [9] provide a method called A-GAP to compute aggregates of node variables in a continuous manner. A-GAP does this by implementing two primary mechanisms. Firstly, it creates and maintains a self-stabilizing spanning tree and secondly it incrementally aggregates the variables along the tree. The protocol sends aggregation updates, which include the changes in the monitored variables, towards the central management station through the aggregation tree. By filtering the updates being sent from monitored nodes toward the management entity, it also controls the management data overhead. The conditions of filtering adapt to the properties of the monitored variables and the network conditions.

According to Wuhib et al. [10], it is important to be able to detect as soon as possible when a monitored variable related to a network node has crossed a certain threshold. They present a protocol designed for detecting, in a distributed way, threshold crossings of network-wide aggregates. Their protocol is an extension of GAP [8], a tree-based aggregation protocol, and they call it TCA-GAP. It differs from GAP in that the root node raises and clears alerts. TCA-GAP also allows transition of nodes from active to passive state, and trade-off control between protocol overhead and quality of detection. The aggregation functions supported by the protocol include SUM, AVERAGE, COUNT, MAX and MIN.

To provide more stability in case of network node failures, Wuhib et al. [19] suggest a partial solution that monitors network-wide aggregates using gossip protocols. They simulated a comparative assessment of their work against a tree-based aggregation protocol, and claim that the later protocol is more accurate and robust than the former one.

To provide a higher level of energy efficiency for the network nodes, while performing tree-based in-network aggregation in sensor networks, Deligiannakis et al. [20] suggest a solution

that performs approximate evaluation of queries to decrease the number of update messages sent between the nodes, and redistributes the aggregation error thresholds.

Somers et al. [11] introduce a new way to measure the compliance of network path quality of service parameters with the target specified by SLA. Their work includes a new and improved methodology for measuring mean delay along a path. They also introduce a new and more robust methodology for measuring delay variation and an improved methodology for estimating packet loss. They unify these measurements in a tool called SLAM (SLA Monitor). The advantage of SLAM compared with previous measurement tools is that the unified probe stream consumes lower bandwidth than would be required for the separate streams.

Sofra et al. [12] consider the quality of computing aggregation functions based on incomplete measurements, used for purpose of monitoring distributed networks. Normally, the information is presented by calculating some key aggregate metrics, computed from a large amount of detailed information collected during network operations. They evaluate the accuracy of some typical aggregation functions both analytically and by simulations. The results can be used for designing optimized network management systems.

The problem of calculating global aggregate functions in large groups like large-scale sensor networks, ad-hoc networks and process groups over the Internet in a scalable and accurate way has been a challenge and several solutions has been discussed by Gupta et al. [13]. It has been argued that traditional solutions are not scalable in large groups and do not work well in faulty networks. In return, a technique has been proposed to build an abstract hierarchy over those large process groups, while it explains how this hierarchy can be constructed to mirror the network topology. They discuss other alternatives to utilize this technique and eventually they present the Hierarchical Gossiping protocol that utilizes this hierarchical technique. They show mathematical analysis and performance outputs to validate the robustness, efficiency and accuracy of the proposed algorithm.

According to Garofalakis et al. [14], malicious participants are often a problem in modern distributed systems. They present a first step towards an in-network aggregation that is both verifiable and efficiently distributed, designed to work even in non-trusted settings. Their solution is a compact verification mechanism combining cryptographic signatures and Flajolet-Martin sketches. They call this proof sketches. According to them, the solution guarantees acceptable bounds of aggregation error with high probability.

According to Applebaum et al. [15], combining and analyzing data collected at multiple locations is crucial for many applications, such as detecting threats and estimating the popularity of web sites. However, participation in collective data aggregation is often inhibited by privacy concerns. To solve this problem, they have designed, implemented and evaluated a solution for

privacy-preserving data aggregation. They use a semi-centralized architecture dividing responsibility between a proxy that blinds the input from the client, and a database that aggregates values by blinded keywords, and identifies the keywords associated to useful values. This solution gives us a new cryptographic protocol able to protect the privacy of participants and keywords.

2.1.5 Operations, administration and management tools

"Operations, administration and management" or OAM is a general term referring to the standard processes, activities, and tools for operating, administering, and maintaining systems. OAM tools are used for measuring performance of the system as well. There is a quite large number of publications on enhancing the OAM tools and we review the ones more related to our work in this section.

Császár et al. [26] explain the importance of unification of the entire cloud and carrier network made in a project called the EU FP7. This project is focused on the essential enablers and developing a dynamic service chain. One of the required elements of such design is a set of transport Software Defined Networking (SDN) controllers to serve transport programs such as tunneling and OAM functions.

Nibert et al. [27] describes the approaches in the 4WARD project presents a service-oriented architecture for embedded network self-management in a future internet. 4WARD is a generic architectural design focuses on self-properties and service orientation, however, the practical implementation appears to be fragmented and does not address issues such as the integration between control and management planes. It can be said that the role of the human operator has been too simplified by introducing knowledge supported workflows without a detailed system architecture supporting workflows.

The UFO system [28] exhibits a layered and resilient routing architecture that integrates with the routing infrastructure and exchanges narrowed data between multiple OSI layers. The authors focus on designing an efficient reporting system for the routing overlays. In the Cabernet system [29], the authors advise implementing probes and path-quality monitoring algorithms at the connectivity layer, however they specially are looking at delivering enduring performance. In addition, they propose that the Cabernet system can support a UFO-type reporting system to receive updates from the infrastructure. However, these two data-sources do not state how to illustrate a circuit performance in real-time and merge it with other updates initiated by the infrastructure, nor do they describe provisioning features.

Active measurements have been one of the known methods for assessing network performance by service providers. Recent improvements cover how to measure parameters such as path's available bandwidth [30], mostly at the IP/L3 layer. In [16] the "Bandwidth Available in Real-Time" (BART) method is presented. It defines how to measure path's available bandwidth employing two measurement nodes situated on each end of a path. The first node inserts probe-traffic into the network whereas the second node assesses the influences of the network's data-traffic on the probe-packets it collects. Another illustration of a related method is delivered in [31]. Active measurement techniques are necessary in assessing parameters such as round-trip time (RTT) and packet loss. The standard OAM toolsets for Carrier Ethernet and MPLS-TP contain methods for assessing these particular parameters [1].

Passive monitoring is one of the widespread performance monitoring mechanisms. Certain papers in the recent works define functionality which is applicable to the solution we exhibit in this thesis. Stadler et al. presented a series of algorithms [9] [32] which allow network nodes to interchange machine- and port-related management information base (MIB) counter data in an efficient fashion. These algorithms also compute a network-wide aggregate value, for instance the top five flows in the network. The network nodes construct a tree topology overlay which is implemented across the physical segments/links. The tree's root node triggers the call for a certain type of network-wide aggregate, and subsequently at each tree level, the nodes execute the necessary calculation and direct the calculated results from the leaves towards the root. It is illustrated that any aggregate function such as SUM can be gauged inside the network utilizing this algorithm's mechanisms and communication protocols.

Many operators provide thousands of connectivity services which all have dissimilar service level specifications (SLSs). For such operators, it can easily become challenging to keep an eye on which parameters are measureable for each circuit/service, what the satisfactory intervals are and which measurement gears must be employed to gauge the factors. The factors, OAM functions and measurement tools are reliant on equipment, whereas the SLS and satisfactory intervals are reliant on the particular connectivity service. This is a challenge that was not tackled by the state-of-the-art at the time of this project's conduction (2012).

2.1.6 More recent related literature

Since the conduction of this project in 2012, there have been significant updates to the state-of-the-art. Among those, there are two projects closely related to this project. Duvignau et al. [33] suggest adjustments to classical models for continuous distributed monitoring and establish efficient implementations in the evolved packet core. Stylianopoulos et al. [34] co-design a

general threshold monitoring framework with a wireless protocol, called Crystal, for the continuous threshold monitoring problem. Testbed deployment results show the two approaches complement each other effectively and achieve a very low duty-cycle, up to ten times less than a mainstream network stack.

The more general recent trend is to consider the network monitoring as a streaming analytics processing problem. Gupta et al. [36] provide a prototype, called Sonata, using this approach which allows network operators to identify monitoring queries and combine data streams from multiple queries. An inclusive review of the state-of-the-art in software-defined network monitoring is provided by Tsai et al. [35]. A great and recent reference for computer networking topics, software-defined networking concepts, and network management is Ross and Kurose book [37].

3. Architectural Approach

As we mentioned in Section 1.1, our primary objective is to determine how to coordinate measuring the available bandwidth of provisioned point-to-point services to make sure that the SLA is not violated. In this chapter, we explain the measuring tool, BART (Bandwidth Available in Real-Time), that we use for this purpose and its advantages compared to other OAM tools, and how we utilize this tool for our purpose.

3.1.1 Utilization of the "Bandwidth Available in Real-Time" tool

BART (Bandwidth Available in Real-Time) is an OAM tool for estimating the end-to-end available bandwidth over a network path. BART evaluates bandwidth almost continuously in real-time. It frequently samples the available bandwidth of the network path by sending sequences of randomized-rates data packages [16].

The advantage of the BART tool compared to other OAM tools lays in the little computation overhead, being light-weight in memory requirements, and a small test-traffic load. Ekelin et al. [16] compare the performance of BART tool directly with another state-of-the-art OAM tool, called pathChirp, and show the better performance of the BART. They obtain a reasonable accuracy with a small computational effort and low load in extra data-traffic using BART.

In addition to the advantages of the BART tool, the organization also had a preference to use BART. The motivation was the integration compatibility with existing source codes. The source codes for implementing BART were available to edit and adapt to our project. Moreover, some of required information for measurement was based on the circuit SLA. Therefore, we used the BART tool to measure the available bandwidth of each circuit's segment that passes data traffic.

To measure the available bandwidth of the circuit in the network infrastructure, we considered the two ends of each segment as measurement points, and utilized BART as our measurement tool. It is worth noticing that for the bandwidth measurements of a circuit, we only considered the involved nodes of that specific circuit. The involved nodes are derived from the path of provisioned circuit. In our project, these paths were given.

Furthermore, to aggregate results of measured bandwidth from several segments into one central point, called central management module, we also developed an in-network aggregation mechanism. This is required to calculate the minimum available bandwidth of all circuit's segments. In the following section, we explain the aggregation mechanism that we used for our project.

We required management modules within in-network modules for passing results to a centralized entity. To present the results to the operator, we had a central management module. The explanation of the central management module and its role will be explained in the following section. The last required element was a communication protocol to send the results to the central management module. The results are aggregated inside the network infrastructure by the aggregation mechanism. This communication was done via XML messages over a TCP session as explained in the next section.

In section 3.2, we demonstrate the schematic representation of the measured circuit, and explain each element and its functionality.

3.1.2 Architecture of the measured circuit

The architectural overview of our solution to utilize the BART tool for measuring the available bandwidth of provisioned point-to-point services is provided in Figure 1. In this section, we briefly explain each entity of this architectural design, and then in the next chapter, we will describe the integration of BART and implementation of other entities in detail.

The designed architecture has the following entities:

- User interface (web-based GUI):
  This entity, which is utilized by the operator, is the interface where a user of the system is able to interact with the system.

- Central Management Module (CMM):
  This module is a single entity that has a global knowledge about the network infrastructure. It can have a backup module in an ideal implementation. CMM maintains an abstract view of the provisioned circuits and their status to present to the user.

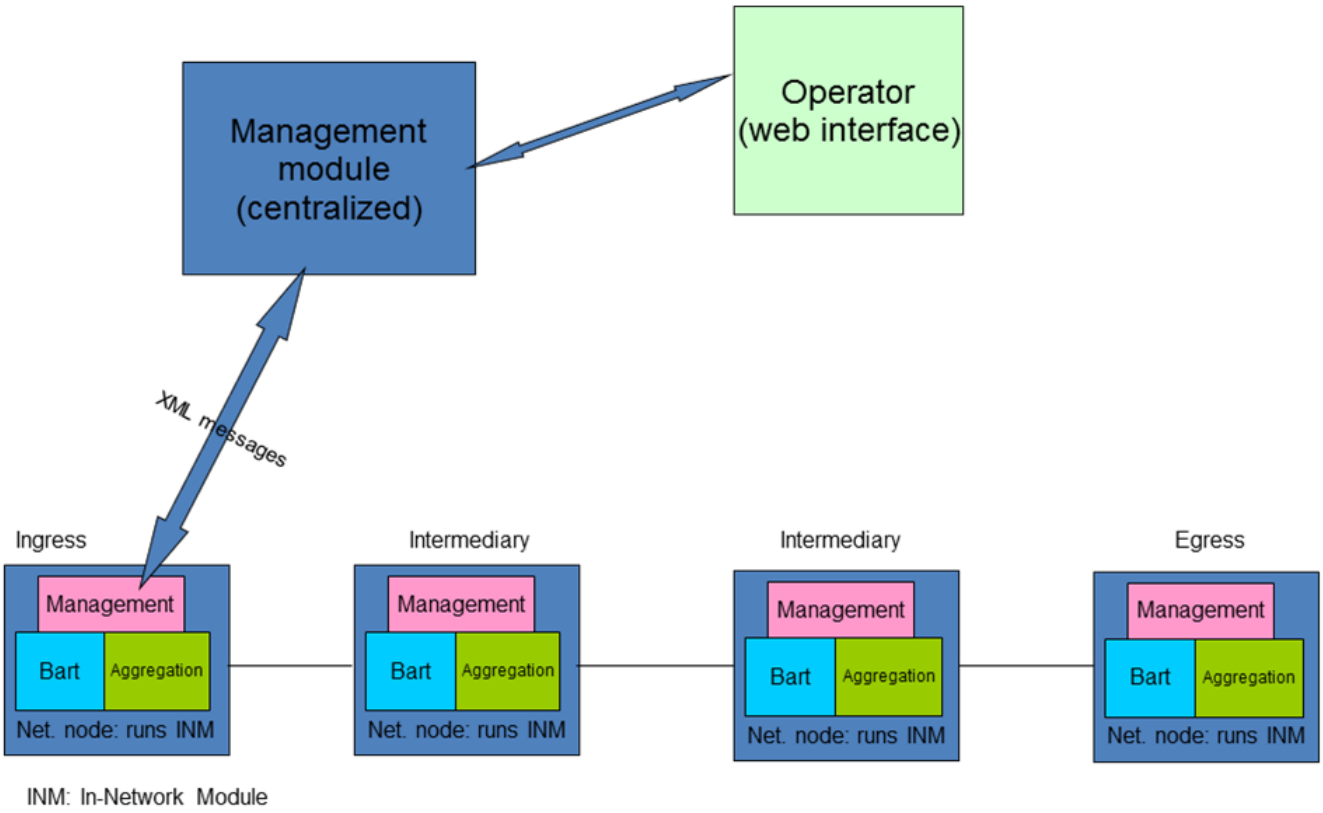

Figure 1 - Architectural representation of the measured circuit with the BART tool

- Network nodes:
  Each network node represents an infrastructure device which runs in-network module (INM). INM is the module responsible for aggregation, triggering BART tool and management functionalities in network nodes. There are three types of network nodes; ingress, intermediary and egress. The ingress node handles traffic entering the circuit. The intermediary nodes handle traffic routing towards the destination. The egress node handles traffic exiting the circuit.
  Each network node consists of three processes itself:

  - *Management process:*
    This process is responsible for communication between the INM and the central management module, and coordinating for triggering the measurements. The communication with the central management module is done via XML messages over a TCP session. This channel is used to receive measurement configuration information from management module and send aggregation results from the

INM to management module. The coordination for triggering the measurements is done by communicating with the BART process and passing measurement configuration information.

- *BART process:*
  This process triggers the BART measurement tool which consists of sender-receiver pairs. It is worth noticing that BART process is a wrapper for the BART measurement tool. The BART receiver process waits for measurement results from the tool and BART sender process triggers the tool according to the measurement configuration information received from the management process.

- *Aggregation process:*
  The aggregation mechanism is responsible for aggregating the measurement results and delivering them to the central management module. The aggregation process waits for a measurement result from the BART receiver process and upon receiving it, performs aggregation actions. If the result of aggregation at a node changes, an update is generated and sent to the neighbor node towards the ingress node. The ingress node will inform the management process of the aggregated result.

- Communication types:
  There are three types of communication in the architecture:
  - between user interface and management module: web services
  - between management module and network nodes: XML over TCP and HTTP post
  - between network nodes themselves: raw data over UDP

  The communication between the management processes of the nodes located in the network infrastructure can be inbound or outbound (over dedicated management links).

In this chapter, we provide a brief explanation of the entities of the architecture. The next chapter provides the solution implementation in detail. First, we will explain how to trigger BART for measuring available bandwidth. Subsequently, we will describe the aggregation of BART measurements, and communication process of transferring aggregated results from INM to CMM. Finally, we will summarize INM interactions and mechanisms.

4. Solution Implementation of In-Network and Central Management Modules

We described the solution architecture and entities in the previous chapter. We will explain the implementation of this architecture for in-network modules (INM) and central management module. The distributed/decentralized entity is called in-network module (INM) and is residing on network nodes in the infrastructure. The central management module also triggers service level specification (SLS) validation for the service. In return, in-network module will initiate the active monitoring of circuit SLS parameters which monitors the circuit/service status periodically.

The tool that has been utilized for active measurements is an extended version of BART [16] called BART_Ethernet which measures the available bandwidth. BART_Ethernet is the BART version that works at OSI layer-2 or so called Ethernet layer. This version deals with source and destination MAC addresses of the measured segment, instead of IP addresses that are employed by BART tool.

The measurement results are aggregated in a harmonic and distributed architecture by INM agents, and presented to the Central Management Module (CMM) to report a summary to the user.

The layout of this chapter is as follows. Section 4.1 discusses our approach for coordination of triggering the BART tool. This is needed to be done through an automatic configuration of the BART tool, and then triggering the measurements. In section 4.2, we show how we integrate the BART measurements using an aggregation engine. The aggregation engine is responsible for receiving the measurement results from the local BART receiver and the neighbor node, calculating the aggregation value at the node and updating the neighbor node. We describe how the aggregation mechanism works at a functional level. Section 4.3 explains how INM and CMM communicate. There are two types of communication between these two entities, on-demand and periodic. Section 4.4 provides a brief picture of the INM interactions with other INMs on separate network nodes and also with the central management module.

4.1.1 Coordination for triggering available bandwidth measurements

Here, we explain coordination of triggering the BART tool. We have done this through an automatic configuration of the BART tool, and then triggering the measurements. We start by explaining the automatic configuration of operations, administration and maintenance (OAM) tool. This automatic configuration is initiated from the circuit's ingress node and is propagated through the circuit all the way towards the egress. We will also explain the measurement configuration data format. Upon receiving the measurement configuration, each node starts the active measurements.

4.1.2 Automatic configuration for Operations, Administration and Maintenance

We describe the steps of automatic configuration for OAM. This process is performed by the INM management process and initiated from the ingress towards the egress node. After the ingress node receives the provisioning message for a specific circuit from the central management module, it initiates the automatic configuration mechanism by loading the appropriate circuit measurement configuration from a locally stored file. The ingress takes away the part/block of the configuration message that is relevant to itself. It processes that block and sends the measurement configuration message/signal to the next node, which is normally an intermediary node, along the circuit path.

The intermediary node processes the configuration message after receiving it. It takes away part of the message relevant to itself, processes that part and sends the remainder of the message to the next node along the circuit path towards the egress. The next node can be another intermediary or the egress node.

In each node processing the signal, if there is no remainder block from the configuration message, then the current node is the egress of the circuit and automatic configuration for OAM is completed.

In our implementation, there is no interaction between the INM (in-network module) entity and a GMPLS control plane. Therefore, we obtain circuit measurement configuration from a locally stored file. The GMPLS control plane has the capability to determine the best path between every two points in the infrastructure/backbone topology. This is accomplished by contacting the Path Computation Engine (PCE). PCE has the complete topology map with details about the interfaces involved in the infrastructure. Upon requesting a path between every two points in the infrastructure topology, the request is passed to the PCE to get a primary and a backup

path. The primary path then becomes active for the circuit.

The locally stored measurement configuration file at the ingress node is a text with information blocks separated by special characters, ‘,’ as value delimiter and ‘$’ as block delimiter. Each block has the following information:

- Circuit ID: the circuit this block belongs to.
- Next-hop service MAC address: the address to send OAM frames to. This will be empty for circuit egress node.
- Next-hop management IP address: the address to forward the remainder of the measurement configuration message to. This will be empty for circuit egress node.
- Previous-hop service MAC address: the address which incoming OAM frames are sourced with. This is required by the BART_Ethernet receiver implementation to be able to recognize the correct sender.
- Parent management IP address: the address of the parent in the aggregation tree with ingress node as the root. The tree is used for aggregating the measurement results toward the ingress.
- Measurement interval: is used as the base interval for triggering measurements.

In the measurement configuration message propagation mechanism, the message traverses the participating nodes in a circuit in the order of data flow from ingress to egress. We assume that either there is the possibility to reach all nodes from every single node via management links; or the data links have the capability to carry management traffic as well which means IP addresses are assigned to them.

4.1.3 Triggering measurements

Triggering the measurements is performed by the INM management process and through wrapper process for BART_Ethernet sender. The measurement is done by triggering BART_Ethernet tool periodically according to the received measurement configuration. Each node, participating in the circuit, starts the active measurements after processing the configuration message while delaying a small amount of time. The delay helps to make sure that the configuration phase is complete along the circuit path before sending OAM frames. The appropriate parameters are passed to the BART_Ethernet executable code via command line execution. The command line parameters for BART_Ethernet and the base interval of executions are retrieved from the configuration message.

The pre-developed BART_Ethernet measurement tool we have used was implemented in C language and also modified to work at Ethernet layer. This tool had two C-code pairs which were called BART_Ethernet sender and BART_Ethernet receiver. In the rest of this document, the simpler names, BART sender and BART receiver, have been used.

At the triggering step, the BART sender C code is called and the code returns after completing the execution. The BART sender code sends a trail of OAM frames at the data link layer, e.g. Ethernet in our environment, over the defined segment/link.

At each BART sender's call, the OAM frames are configured to be sourced as the MAC address of the node's outgoing interface towards the next node/hop. The OAM's destination address is configured to be the MAC address of the receiving node's interface on this segment/link. The outgoing interface should also be defined before calling BART sender. This is because BART_Ethernet works at Ethernet level and does not have IP level information of the target node, to be able to query the local system routing table for outgoing interface resolution. On the other hand, when the segment connectivity between the nodes is just configured to have layer-2 information, we need a mechanism other than IP routing to determine the outgoing interface. This scenario can be the case for a circuit established over a layer-2 Ethernet backbone or a Multiprotocol Label Switching (MPLS) backbone with out-of-band IP management links.

At the other end of the link, BART receiver is listening for incoming OAM frames on the Ethernet link. BART receiver only picks the OAM frames BART sender has generated and calculates the segment's available bandwidth.

We have used non-equal triggering intervals from Poisson distribution. BART sender is being triggered according to random sampling from the Poisson distribution of the base interval. This results in triggering intervals within a confidence interval of the given base. We utilized this method to make the triggering moments on the circuit path desynchronized to help scattering the load of processing aggregation messages at ingress.

4.1.4 Aggregating the results in a distributed way

Here we explain how the measurement results are collected, aggregated and updated at the INM node level. This includes the integration between BART_Ethernet and aggregation engine,

and also the aggregation function that works in a distributed manner.

4.1.5 Integration between "Bandwidth Available in Real-Time" tool and aggregation engine

The aggregation engine is responsible for receiving the measurement results from the local BART receiver and neighbor node, calculating the aggregation value at the node and updating the neighbor node if the aggregation value has changed. The delivery of measurement results to aggregation engine occurs immediately. The integrations between BART Ethernet tools and the rest of INM are handled by the BART Sender and BART Receiver wrapper processes as shown in Figure 2.

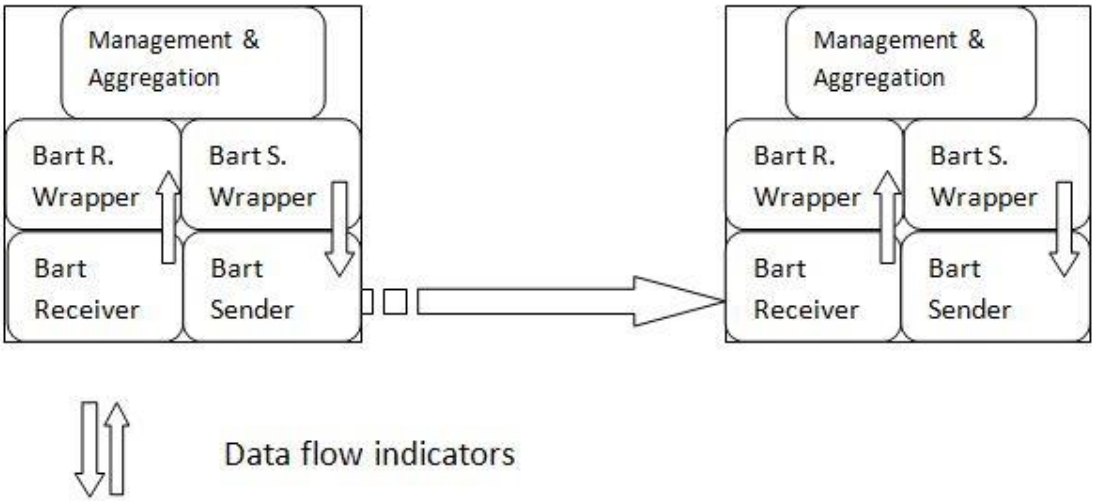

Figure 2 - Integration of the BART tool using wrappers

Upon receipt of the OAM frames by the BART_Ethernet receiver, it generates the result on the standard output which is monitored and immediately captured by the BART receiver wrapper. Then the wrapper parses and delivers the result to the local aggregation engine. The communication between BART receiver wrapper and local aggregation engine is performed via an update message. The update message utilizes a format similar to the format of aggregation update messages sent between nodes. The aggregation update messages will be described in the aggregation section. The message is passed as a parameter by calling a *shared method* in the aggregation engine.

The BART_Ethernet code is in C, however, the aggregation code and the whole implemented INM is written in java programming language. Therefore we need to handle the multi-language

integration. The C and Java integration is handled by BART receiver wrapper. At the BART sender side there was a need to call the BART_Ethernet sender code periodically. This is handled by utilizing the process library of Java and creating a command line process. The required arguments are passed via making them sorted as a command line string and calling the BART_Ethernet sender executable supplying the command line string. As a result, the C code will be run in a separate thread in the operating system and will terminate and return after completing execution.

At the BART receiver side we needed to first create the command line process of BART_Ethernet receiver while passing the required arguments that are extracted from the received measurement configuration message. The BART_Ethernet receiver executable will stay running in memory and listen for incoming OAM frames at the Ethernet layer. In the BART receiver wrapper thread, we defined a listener method on the standard output of the BART_Ethernet receiver. The listener method will get activated whenever a new line is generated by the receiver process. Upon receipt of an OAM frame the listener method is called and in return it will parse the BART_Ethernet receiver output and generate the update message to be delivered to the local aggregation engine.

As the aggregation engine is running in its own thread and separate from the BART receiver wrapper thread, we need to address the inter-process communication issue. The mutual exclusion situation is handled by implementing a lock variable in the *shared method* of the aggregation engine. Before entering the critical section (shared method), the lock prevents a thread from entering the section if another thread is still running the body of that method. Upon exiting the method by the already entered thread, the lock is released and the waiting thread will enter the body of the shared method. This functionality is handled by aggregation process.

4.1.6 Aggregating over a tree overlay topology

Here we describe the aggregation mechanism at a functional level. The aggregation engine works over a tree overlay topology. The connections/branches between nodes on this tree can be maintained over data links or management links. If the data links are utilized, the aggregation mechanism resembles in-band management. If there are separate management links implemented for the infrastructure, the aggregation tree can be maintained over management links and the aggregation mechanism resembles out-of-band management. In this prototype, we implemented the in-band method for aggregation mechanism. Our aggregation has the following characteristics:

- Tree topology:
  The topology of our aggregation tree can be input from an external process which is aware of the infrastructure data/management links. In that case, the external process might perform spanning tree (or another protocol) calculation for deriving the tree for the specific nodes participating in a circuit.
  In the solution that we have implemented for this prototype, the aggregation tree should be available as a locally stored file. We can manually create the XML file to represent our aggregation topology.

- Representing the tree on file:
  The input format for the aggregation tree topology is XML. The current implementation reads the XML tree from a locally stored file, available for all of the nodes. The root of the tree is the ingress node for the specific circuit.

- Measurements direction and results location:
  As mentioned earlier a segment, being part of the network infrastructure and involved in carrying the data traffic of a specific circuit, is measured by sending OAM (operation, administration and maintenance) frames over that segment. In our solution, the probing OAM frames always traverse toward the egress of the circuit on each segment. If the OAM frames are sent from node A to B then the BART receiver at node B will receive and calculate the available bandwidth of the segment between A-B and in A to B direction. The available bandwidth measurement result for A->B segment, is delivered to the aggregation engine of node B. Therefore we always have the segment measurement result at the node closer to the circuit egress.

- Aggregate calculation over the tree overlay:
  As we are willing to have the aggregated result at the ingress of the circuit, we define the aggregation tree in such a way that the ingress node of the circuit becomes the tree's root node. Therefore, every node in a circuit needs to calculate the partial aggregate at its position in the aggregation tree and send it to its parent in the tree.
  Since our circuit tree is a path graph [18], we have only one path between any two nodes. Therefore, we will receive unique node updates every time and there are no duplicate update messages received at a node. If our circuit topology was not a tree, then there could be more than one path between two nodes and duplicate update messages could be received.

- Accommodates the aggregation of *bandwidth* and *delay* measurement results:
  The implemented aggregation engine is capable of handling *minimum* and *sum*

aggregation actions.
In the case of bandwidth measurements, we utilize the *minimum* aggregation action to derive the end-to-end available bandwidth of the circuit. In the case of delay measurements, we utilize the *sum* aggregation action to derive the end-to-end delay along the circuit.
The aggregation engine executes the relevant action (minimum/sum) based on the *measurement type* defined in an update message.

- Aggregation engine filters updates for its parent:
  Updates to parent are sent only when aggregated value changes at a node. This filtering helps for decreasing the amount of update messages sent over the links, which saves bandwidth and processing cycles.
  The detail of the operation is described in the *shared method* section.

Table 1 – Aggregation fields

Field	Description
ID	a unique identifier for a network node
Relation {Self, Parent, Child}	Each row represents a piece of information received from the network node itself (self), a child or info related to the parent which is learned via the topology file.
Aggregation Action {Min, Sum}	This field in the current implementation does not play any role. It can be used manually determine the aggregation action to use on a particular record, without considering the *measurement type*.
Measurement Type	This field stores the type of measurement for this record's value (*Measurement Result*).
Measurement Result	This field stores the aggregated measurement value received from the neighbor node or the measurement result from the local node (self).

Aggregation table is an array of aggregation rows. An aggregation row is composed of the certain fields. Table 1 describes the utilized aggregation rows. There are aggregation sub-functions that work with aggregation table values. The following sub-functions, which are shared methods, are executed upon receipt of a message:

- Parsing the update message:
  In this sub-function the supplied update message is parsed to firstly figure out the update type and then interpreting it accordingly. There are two update message types, local update and neighbor update. The one that is passed from BART receiver wrapper to the local aggregation engine is of type local update. The update message that is sent from a neighbor to its parent is of type neighbor update and is used in the aggregation mechanism. If the message type is neither of those above, an error is logged. In the implementation, we used white space as the actual delimiter.

  - The local update fields are as the following: message type, measurement type, and value.
  - The neighbor update fields are as the following: message type, neighbor ID, parent ID, measurement type, and value.

- Updating the aggregation table based on the message contents:
  In this sub-function, based on the update type and measurement type, the corresponding record for the message source node in the aggregation table will get updated. If the message type is neighbor update and the record with source ID of the neighbor node does not already exist in the table, it will be added; otherwise the existing record's old value gets overwritten by the received value.
  If the message type is local update, the corresponding record with local node's ID will get updated in case it exists, and will get created otherwise.
  Note: The update sub-function also keeps track of the measurement type and it should also match for the record update action in the aggregation table.

- Calculating new aggregate value:
  In this sub-function the new aggregate value (for bandwidth or delay) will be calculated every time after executing the previous sub-functions. This calculated value represents the partial aggregate value at the local node. This value is calculated considering the local node as the root for the sub-tree where the other nodes in the sub-tree are all its children. The old aggregate value is stored in a variable and the newly calculated one is stored in another variable. Having the last and new aggregate values helps for detecting if the aggregate at this node has changed after receiving the update.
- Aggregation action:
  In our aggregate method we will utilize the *minimum* action if we are dealing with bandwidth of the segments; and will utilize the *sum* action if we are dealing with delay of the segments. The choice of aggregation action to use is retrieved from the latest processed message type field.

As a result, the segment with *minimum* available bandwidth will be the bottleneck for the provisioned circuit and represents the available bandwidth of that circuit, if we measure the bandwidth.

- Sending update message to parent in case the aggregate value has changed:
  There is a check to see if the new aggregate value (bandwidth/delay) has changed since the last calculation. If yes, then a sub-function will be called that sends a *neighbor update* message with the new aggregate value of the local node to parent.

4.1.7 Communication between Central Management Module and In-Network Module

There are two types of communication between CMM and INM, on-demand and periodic. On-demand communication occurs when a request is made by the CMM as well as when a response is produced by INM. Periodic communication is the automated status update from INM to CMM. All of the message contents, in the both communication types, are XML formatted.

4.1.8 On-demand communication

The first is over pure TCP socket communication which is initiated from CMM toward INM. This connection on a specific server port, defined in options.ini of INM, can send a request via two message types and there is one defined reply message type associated with each request type.

INM at the ingress listens on TCP port 10300 for incoming "Provisioning request" and "SLA parameter validation request". It keeps the connection open and in case of "Provisioning request" it replies with "Provisioning reply", which have the requested “service-ID” and “ok” as response, and closes the connection. In case of "SLA parameter validation request" it fetches the latest aggregate result for the requested measurement-parameter(s) {Bandwidth, Delay}, replies with "SLA param validation reply" and closes the connection.

4.1.9 Periodic communication

The INM on the ingress also sends periodic updates to the CMM. When the aggregate value for a specific measurement-parameter changes, it sends an HTTP post containing an "SLA

parameter validation reply" to a defined URL on the CMM, indicating the measurement-parameter and value.

4.1.10 In-network module interactions and mechanisms

As we discussed in Chapter 3, there are three main entities in the architecture: user interface, central management module and network nodes. In-Network Module (INM) runs on the network nodes, which in most cases they would run INM on their management processor or supervisor engine. In this section, the aim is to give a brief picture of the INM interactions with other INMs on separate network nodes and also with the central management module. Figure 3 gives an overall picture of these interactions.

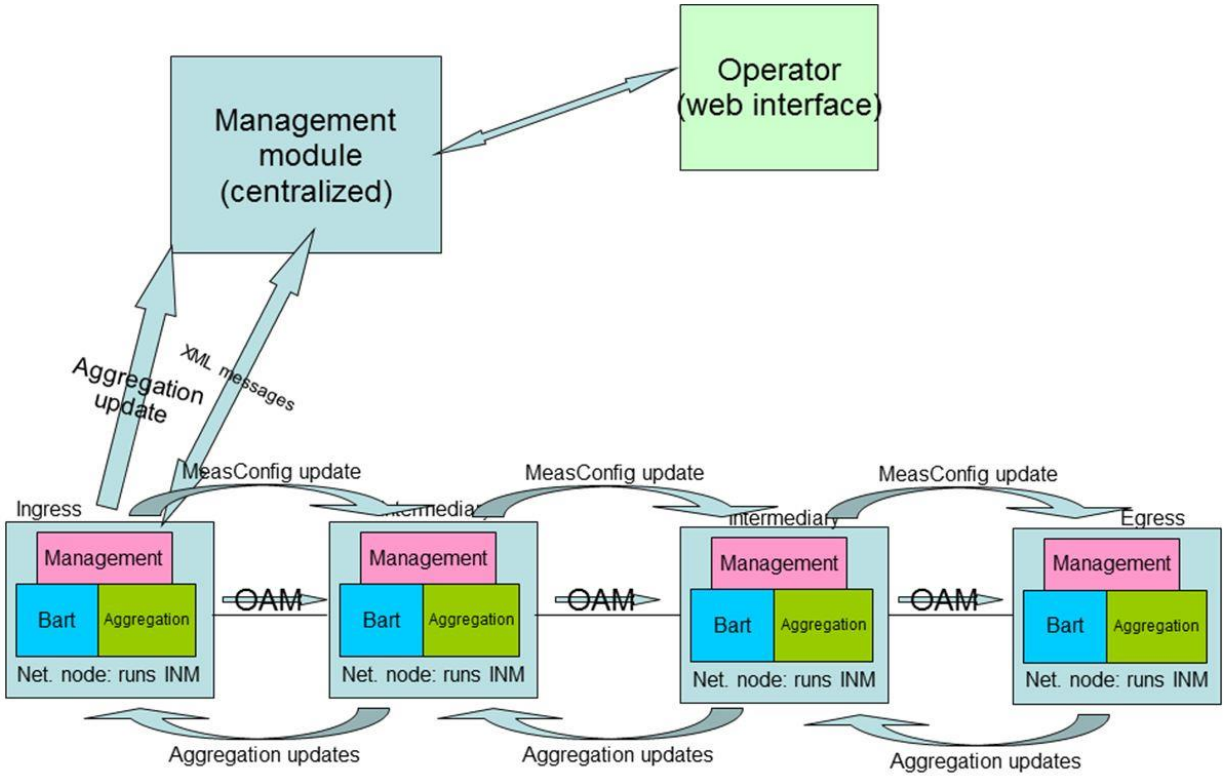

Figure 3 - INM interactions

4.1.11 In-network module interactions

In Figure 3 we have four network nodes which are participating in a circuit between the ingress and egress nodes. In this scenario the aggregation tree is established as a chain between the

ingress and egress nodes, with ingress being the root of the tree. We also have the signaling and monitoring traffic as “in band” over the data links. The following describes the INM interactions:

- At the provisioning stage, INM management process on the ingress receives the circuit provisioning signal together with the circuit measurement configuration information from the central management module. In our scenario the measurement configuration is stored on a local file with its values being assigned manually with the prior knowledge of the link properties. After receiving the provisioning signal, ingress’s management will initiate the automatic measurement configuration by applying the part of the configuration related to itself and sending the MeasConfig update message to the next participating node, which is an intermediate one, in the circuit.

- INM management process on the intermediate node will receive the MeasConfig update message and after configuring the local values, sends the MeasConfig update to the next node in the circuit. This continues till the message reaches the egress node. The egress node’s management figures out that this is the end of the MeasConfig signaling mechanism, as the message content indicates it, and automatic measurement configuration stops at this point.

- With a bit delay after receiving the MeasConfig update message at each node, the INM on the node will start the active measurements by triggering the BART tool which sends OAM frames over the data link that is in the path towards the egress of the circuit.

- Right after each node receives an OAM-frames block, generated by one measurement attempt, the INM BART process will calculate and report the available bandwidth to the INM aggregation process. As this result arrives to the aggregation process, it will recalculate the aggregate value for this location, which is local node, on the aggregation tree.

- If the aggregate value at the local node has changed, the INM aggregation process will send an aggregation update, a neighbor update in this case, including the new aggregate value of the local node to its parent in the aggregation tree.

- The INM aggregation process that receives the aggregation update, will recalculate the aggregate value at its location of the aggregation tree, and if it had changed, will send an aggregation update to its parent.

- When the aggregation update reaches the ingress node, which is the root of aggregation tree, after recalculating the circuit's complete aggregation value, the INM aggregation process will send an aggregation update to the central management module in form of an XML message. This XML message represents the latest available bandwidth value of the measured circuit over its various links. Afterwards, the latest result will be available for fetching by the user interface or operator.

4.1.12 In-network module mechanisms

For the implementation of In-Network Module, we chose the Java language for developing in-network functionalities. In this section we will cover main mechanisms and technologies which we have utilized.

There are some main threads/processes in INM, called management, BART wrappers and aggregation. These threads are running all the time while doing the following tasks:

- Management: listens and reacts to incoming provisioning request, latest measurement value request and automatic measurement configuration; handles automatic configuration of OAM; initiates/reconfigures active measurements; starts up INM on the network node.

- BART Sender Wrapper: triggers BART Sender tool for active measurements based on the measurement configuration received from Management process

- BART Receiver Wrapper: loads BART Receiver tool based on the measurement configuration received from Management process, waits for measurement results from the tool and delivers the results to the Aggregation process.

- Aggregation: maintains an aggregation table with multi measurement type capability; handles locally delivered measurement results; listens for incoming aggregation updates from neighbor nodes; calculates the aggregate at the occurrence of any new aggregation event; sends aggregation updates to its parent in the aggregation tree.

The above threads need to communicate with each other and passing data. As they are running as separate threads in the operating system, there need to be a mechanism for mutual

exclusion handling to facilitate this. To implement this inter-process communication, we have utilized lock mechanism to handle mutual exclusion in a shared method.

We chose User Datagram Protocol (UDP) as the transport protocol for MeasConfig message passing and aggregation update messages. The reason is that the mentioned messages are going to be transferred over management links which are reliable and lossless.

INM, running on the ingress node of a circuit, is responsible for communicating with central management module. INM utilizes the XML message generation and parsing techniques to send and receive information. INM on each network node reads the aggregation tree topology from an XML file. This gives us the flexibility to input various aggregation topologies, by either manual configuration or having an external daemon/process, such as Spanning Tree Protocol implementation. This helps us to provide the INM aggregation engine with the aggregation tree. The current implementation configures the tree only once at INM startup time.

We needed to have access to the forwarding information base (FIB) table of the underlying operating system. This is required to read the interface index and MAC address of the local Ethernet interface by having the MAC address of the remote Ethernet interface from MeasConfig message. This MAC address pair is associated with the data link that is going to be measured for bandwidth and is needed to be supplied to the BART tool at each end of the link. As the INM implementation has been done on the Linux platform, we could receive those values by accessing the forwarding table of the Linux, acting as our network node, but we chose to have a static FIB for our measurement purposes for the ease of implementation.

We have used a configuration file, formatted as Java options file, to receive the main parameters at INM startup time, including node ID and management IP address. As the INM code was decided to be developed in Java programming language and the BART tool that we had was developed utilizing C language, there was a need to integrate these two. We utilized the command line process creation in Java to call the BART tool in C.

In this chapter, we provided the detailed description and operational processes of each entity of the architecture. The development and function of BART_Ethernet, the aggregation of the results, and communication and interaction between entities were explained. In the next chapter, we will represent the experimental results of operating the architecture on virtual machines.

5. Implementation Outcomes of the Architecture

Here, we explain the implementation of the architecture for available bandwidth measurements described in Chapter 4. Section 5.1 represents the test platform for the implementation. We implemented the prototype on a test platform with four virtual machines as shown in Figure 4. Each virtual machine acts as one of our network nodes in the circuit. Section 5.2 provides the evaluation and analysis of the measured data. We recorded all of the measurement and aggregation results for evaluation of the INM functionality and accuracy. Section 5.3 explains the benefits of using our architecture as an automated OAM functionality.

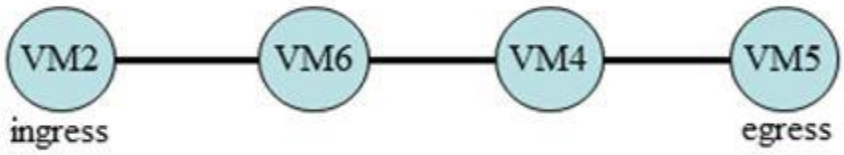

Figure 4 – Network nodes connectivity topology in the testbed

5.1.1 Test platform characteristics

We had a test platform with four virtual machines acting as our network nodes for a circuit, see Figure 4. The testbed had the following main features:

- A virtualized environment:
  The network nodes, that we were utilizing to run the INM code on top of, were a set of virtual machines. This has been done utilizing QEMU virtualization engine which is open source software.

- Host machine:
  The physical machine, which we had utilized as our hardware platform, was having four

Intel Xeon cores. We had installed a 64-bit Linux as the host operating system and configured a couple of system and network services to be able to access the physical machine remotely over IP network.

- Virtual machines as guest operating systems:
  The virtual machines were running 32-bit Linux as the guest operating system (OS) and these four had similar configurations. The virtual platform for each guest OS was provided by QEMU engine. We had a separate QEMU process on the host machine for each guest OS. We could access each guest OS by either SSH or VNC graphical connection.

- Virtual network connectivity:
  We had pre-configured virtual connections between these virtual machines to have a static connectivity which resembles the circuit that is already established/provisioned. We had configured and utilized Linux bridge interfaces on the host machine. These interfaces implement virtual switches which can provide layer-2/Ethernet connectivity between all of the Ethernet interfaces, physical or virtual. The interfaces are assigned to each virtual switch on the host.

5.1.2 Evaluation of in-network module functionality

For evaluation of the INM functionality and correctness, we recorded all of the measurement and aggregation results. The nodes and links of our testbed are setup using virtual machines and Linux bridges. However, the measurement results are generated by the existing tool, BART_Ethernet.

5.1.3 Testing the in-network module functionality

We wanted to verify the functionality and correctness of the aggregation mechanism by testing the prototype. For this purpose, we utilized the testbed and executed the INM on all the mentioned four virtual machines. The INM code triggered the BART_Ethernet measurement tool, and gathered and aggregated the measurement results.

For recording the measurement results, we utilized an additional functionality on each node that recorded every measurement result, immediately after receipt of the OAM frame, together with the timestamp on a local file. This helped us to import locally measured results offline.

For recording the final aggregate, we also utilized the recording functionality at the ingress node of the circuit to record every aggregation update, sent from ingress to management module, together with the timestamp on a local file. Therefore, we could import the aggregation update values offline.

We considered three separate sets of measurement results, each representing the recorded results for a specific segment, together with the aggregation update values from the ingress node. We imported those four sets into Excel-file, while having the timestamp values for each set as a separate column. The timestamps were recorded by the network nodes in the Epoch time encoding format.

On the other hand, to have the machine-times synced between all of the four nodes and the host machine, we utilized and configured network time protocol (NTP) client and server. This synchronization allowed us to generate diagrams out of the offline data that had been gathered from individual virtual machines or our network nodes, with a good accuracy.

5.1.4 Analysis of measurement results

Figure 5 represents segment measurement results at each non-ingress node and final aggregate update values at ingress over time. There are two interesting observations in this figure.

The first observation, marked by pointer A, shows a sudden change of the aggregate update value after an absence of a value for a period of time. This happened because the BART_Ethernet tool reported zero when the measured available bandwidth is below 1Mbps. For this period of time, the available bandwidths in all three segments were below 1Mbps, and therefore, the aggregate value, which is the minimum of the three measurement results, was zero as well. At the last moment of this period, we have the minimum equal to the bandwidth of the segment reported by VM4, shown by yellow points. We see that the aggregation update has only reported a value when the minimum value has changed. This shows the correctness of the aggregation mechanism and also demonstrates the filtering functionality in the aggregation update message generation.

The second observation, marked by pointer B, shows a delayed follow of the aggregation value at several different points. One reason for this result is that as we were utilizing virtual machines and Linux bridge interfaces, which were emulated on top of the physical machine's operating system. Hence, we were experiencing a semi-random delay for virtual machine processing times and on the virtual Ethernet links. These conditions can result in the aggregation update messages being generated, processed and transmitted with a delay. A

similar delay is also observed in one of experimental evaluation of GAP [8] where only local values change randomly over time.

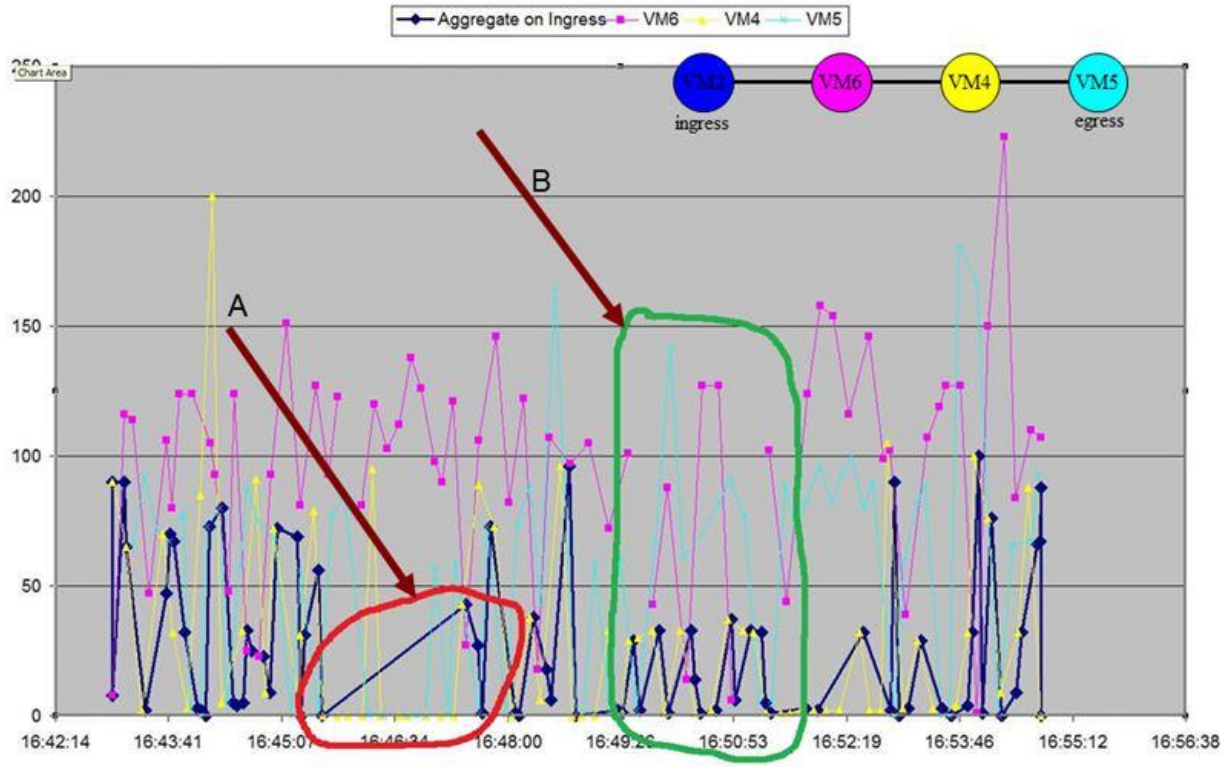

Figure 5 – Measurement results and final aggregate update values (bandwidth in Mbps) sketched vs. time

Figure 5 also shows the existence of filtering functionality as part of the INM code on all nodes. This functionality causes a node to send aggregation update only when the aggregate value, partial or final, changes at that node. This behavior is similar to the GAP algorithm by Dam and Stadler [8] which sends an update vector if the newly calculated aggregate value has changed since last calculation, but the difference is that GAP also checks if a certain time has passed since the transmit of last update vector and utilizes broadcast mechanism to send the update vector to all known neighbors.

A high fluctuation is observed in the available bandwidth measurement results. This could be result of virtualization as all nodes' network adapters and hardware platform utilized for our experiments were virtualized. We were experiencing an unstable measured bandwidth from one node's Ethernet interface to a neighbor's Ethernet interface.

5.1.5 Advantages of our in-network module implementation

There are two main advantages of our INM implementation; having the functionality of configuration of OAM in an automatic fashion, and reducing the measurement results for the monitoring operator.

5.1.6 Configuration of operations, administration and management tool in an automatic fashion

The implemented automatic configuration of OAM provides the functionality of having all of the participating nodes in a circuit to be configured and ready for active measurements at the provisioning time. This is handled utilizing the MeasConfig update messages.

5.1.7 Reducing the measurement-results data for the monitoring operator

The implemented in-network aggregation of measurement results, reduces the data gathered from several measurements at multiple distributed nodes and transfers only the data which is considered valuable for central monitoring purposes.

On the other hand, utilizing in-network aggregation, which is a distributed solution for processing data, helps for balancing the processing load between network nodes participating in a circuit, in contrast to a centralized solution.

6. Conclusions

This thesis has covered part of a major architecture and was functionally implemented at the layer of network infrastructure devices forming a metro network. The work has been implemented and verified in a virtualized environment which resembles the metro network infrastructure consisting of network nodes and links. We wanted to monitor the SLA parameters, mainly bandwidth, of a provisioned point-to-point Ethernet service/circuit in an efficient way and deliver the results to a central entity in the architecture. The following functionalities, forming the In-Network Module (INM), have been designed and implemented to be executed on the network nodes at the data transmission layer to fulfill the mentioned requirements:

- Coordination for triggering available bandwidth measurements:
  The first set of functions in this category was designed and implemented to automatically configure the network nodes participating in a provisioned circuit for bandwidth measurements at Ethernet layer. The second set of functions was implemented to periodically trigger the BART Ethernet tool which would send OAM frames over the desired Ethernet segment of the circuit toward the egress node for available bandwidth measurements.

- Aggregating the results in a distributed way:
  A set of functions in this category was designed and implemented to receive the available bandwidth measurement results from the BART_Ethernet tool, aggregate the local's and neighbor's, which is its child in the aggregation tree, results and send aggregation update messages to the parent. Aggregation mechanism is evaluated via diagrams and its correctness/features are demonstrated.

- Communicating with the central management module:
  A set of functions for this purpose was designed and implemented to firstly pass the updated aggregation results from the ingress node to the Central Management Module

(CMM) on a periodic basis upon a change in those results. The second set of functions is responsible for handling on-demand bandwidth measurement requests, received from the central Management Module, to reply with the latest aggregation result.

6.1.1 Future work

This work has developed a mechanism for automatic configuration of measurement probe parameters, which is being handled by INM on each node and initiated by the INM at the ingress node. This automatic configuration of OAM function can be integrated into the GMPLS control plane signaling which is responsible for provisioning the circuits in many service provider scenarios. That way, by utilizing the GMPLS control plane, the nodes on a circuit data-path can be configured to be ready to perform in-band measurements at the circuit provisioning step.

It is worth mentioning here that the contributions made by this thesis were in the context of the state-of-the-art of the time of conduction (2012). In the light of recent developments, design ideas and methods are moving more towards integrating network monitoring functionalities with control plane (or even management plane) of a software-defined network. Developing more efficient monitoring, investigating potential utilization scenarios, and exploiting multilayer security monitoring are suggested directions for future works.

7. Acronyms

Acronym	Description
MeSON	Metro Self-Organizing Network
BART	Bandwidth Available in Real-Time
OSI	Open Systems Interconnection
MPLS	Multiprotocol Label Switching
MPLS-TP	Multiprotocol Label Switching - Transport Profile
RTT	Round-Trip Time
SLA	Service Level Agreements
SLS	Service Level Specification
SLAM	SLA Monitor
OAM	Operations, Administration and Management (/Maintenance)
GMPLS	Generalized Multi-Protocol Label Switching
PCE	Path Computation Engine
MEF	Metro Ethernet Forum
IETF	Internet Engineering Task Force
ATM	Asynchronous Transfer Mode
IP	Internet Protocol
VPN	Virtual Private Network
ISP	Internet Service Provider
GAP	Generic Aggregation Protocol
A-GAP	Adaptive GAP
TCA-GAP	Threshold Crossings of Aggregates – GAP
INM	In-Network Module
CMM	Central Management Module
GUI	Graphic User Interface
MeasConfig	Measurement Configuration
MIB	Management Information Base
FIB	Forwarding Information Base
RIB	Routing Information Base
VM	Virtual Machine

QEMU	Quick Emulator
KVM	Kernel Virtual Machine
OS	Operating System
XML	Extensible Markup Language
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
HTTP	Hypertext Transfer Protocol
MAC	Media Access Control
SSH	Secure Shell
VNC	Virtual Network Computing
NTP	Network Time Protocol

8. Bibliography

[1] A. Johnsson, C. Meirosu, N. Anjum, A. Tirdad, “Towards Automatic Provisioning and Validation of Ethernet Services over Transport Networks”, in 4th IFIP/IEEE Workshop on Distributed Autonomous Network Management Systems, 2011.

[2] “MEF Technical Specification MEF 6.1 -- Ethernet Service Definitions”, Phase 2., 2008.

[3] Bocci, E., Bryand, E., Frost, D., Levrau, L., Berger, L., “A Framework for MPLS in Transport Networks”, IETF RFC 5921, July 2010.

[4] D. C. Verma, “Service Level Agreements on IP Networks”, 2004.

[5] “Sprint NEXTEL service level agreements”, https://business.sprint.com/terms-and-conditions/#service-level-agreements, 2010.

[6] “NTT Communications Global IP Network Service Level Agreement (SLA)”, https://www.ntt.com/en/services/network/gin/sla.html, 2010.

[7] K. Lim, C. Adam, R. Stadler, “Decentralizing Network Management”, in IEEE electronic Transactions on Network and Service Management (eTNSM), 2005.

[8] M. Dam and R. Stadler, “A generic protocol for network state aggregation”, 2005.

[9] A. G. Prieto, R. Stadler: A-GAP, “An Adaptive Protocol for Continuous Network Monitoring with Accuracy Objectives”, in IEEE Transactions on Network and Service Management, 2007.

[10] F. Wuhib, M. Dam, R. Stadler, “Decentralized detection of global threshold crossings using aggregation trees”, 2008.

[11] J. Sommers, P. Barford, N. Duffield, A. Ron, “Multi-objective Monitoring for SLA Compliance”, in IEEE/ACM Transactions on Networking, Vol. 18, No. 2, April 2010.

[12] Sofra, N. , He, T. , Zerfos, P. , Ko, B.J. , Lee, K.-W. , Leung, K.K., “Accuracy analysis of data aggregation for network monitoring”, IBM research report, in Military Communications Conference, 2008.

[13] Gupta, I.; Van Renesse, R.; Birman, K.P., “Scalable fault-tolerant aggregation in large process groups”, in Dependable Systems and Networks International Conference, 2001.

[14] Garofalakis, M.; Intel Res. Berkeley, CA; Hellerstein, J.M.; Maniatis, P., “Proof sketches: verifiable in-network aggregation”, IEEE 23rd International Conference on Data Engineering, April 2007.

[15] B. Applebaum, H. Ringberg, M. Freedman, M. Caesar, J. Rexford, “Collaborative, privacy-preserving data aggregation at scale”, Privacy Enhancing Technologies. Springer Berlin Heidelberg, 2010.

[16] Ekelin, Svante, Martin Nilsson, Erik Hartikainen, Andreas Johnsson, J-E. Mangs, Bob Melander, and Mats Bjorkman, "Real-time measurement of end-to-end available bandwidth using kalman filtering", In 2006 IEEE/IFIP Network Operations and Management Symposium 2006, pp. 73-84, IEEE, 2006.

[17] Santitoro, Ralph, "Metro Ethernet Services–A Technical Overview", In Metro Ethernet Forum, vol. 2006, 2003.

[18] Bondy, J. A.; Murty, U. S. R., “Graph Theory with Applications”, North Holland, 1976.

[19] Wuhib, F., Dam, M., Stadler, R., & Clem, A., “Robust monitoring of network-wide aggregates through gossiping”, Network and Service Management, IEEE Transactions, 6.2, 2009.

[20] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos, ”Hierarchical in-network data aggregation with quality guarantees”, in Proc. 9th International Conf. Extending Database Technology (EDBT), Mar. 2004.

[21] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, ”TAG: A tiny aggregation service for ad-hoc sensor networks”, in Proc. 5th Symposium Operating Systems Design Implementation (USENIX - OSDI), Dec. 2002.

[22] M. A. Sharaf, J. Beaver, A. Labrinidis, and P. K. Chrysanthis, “Balancing energy efficiency and quality of aggregate data in sensor networks”, International J. Very Large Data Bases, vol. 13, no. 4, pp. 384–403, Dec. 2004.

[23] P. C. Roth, D. C. Arnold, and B. P. Miller, “MRNet: A software-based multicast/reduction network for scalable tools”, Proc ACM/IEEE Conf. Supercomputing (SC), Nov. 2003.

[24] Flood, John Edward, ed., “Telecommunication networks,” The International of Electrical Engineers, 1997.

[25] Glossbrenner, K., “Internet protocol data communication service-IP packet transfer and availability performance parameters.”, ITU-T Recommendation I 380 ,1999.

[26] Császár, A., John, W., Kind, M., Meirosu, C., Pongrácz, G., Staessens, D., Takács, A. and Westphal, F.J., "Unifying cloud and carrier network: EU FP7 project unify", Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing. IEEE Computer Society, 2013.

[27] Niebert, Norbert, Stephan Baucke, Ibtissam El-Khayat, Martin Johnsson, Borje Ohlman, Henrik Abramowicz, Klaus Wuenstel, Hagen Woesner, Jurgen Quittek, and Luis M. Correia, "The way 4ward to the creation of a future internet", IEEE 19th International Symposium on Personal, Indoor and Mobile Radio Communications, pp. 1-5. IEEE, 2008.

[28] Zhu, Y., Rexford, J., Bavier, A., and Feamster, N., UFO, “A resilient layered routing architecture”, Editorial Zone, ACM SIGCOMM Computer Communications Review, October 2008.

[29] Zhu, Y., Zhang-Shen, R., Rangarajan, S., and Rexford, J., Cabernet, “Connectivity architecture for better network services”, Proc. Workshop on Rearchitecting the Internet, December 2008.

[30] “ITU-T Y.1540 Amendment 1”, IP-layer Capacity Framework, 2009.

[31] Ribeiro, V., Riedi, R., Baraniuk, R., Navratil, J., Cottrel, L., “Efficient available bandwidth estimation for network paths”, Passive and Active Measurement Workshop, 2003.

[32] Prieto, A. G., Stadler, R., “Monitoring Flow Aggregates with Controllable Accuracy”, Lecture Notes in Computer Science, 2007.

[33] Romaric Duvignau, Marina Papatriantafilou, Konstantinos Peratinos, Eric Nordström, “Continuous Distributed Monitoring in the Evolved Packet Core”, Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems, 2019.

[34] Charalampos Stylianopoulos, Magnus Almgren, Olaf Landsiedel, Marina Papatriantafilou, “Continuous Monitoring meets Synchronous Transmissions and In-Network Aggregation”, 15th International Conference on Distributed Computing in Sensor Systems (DCOSS), IEEE, 2019.

[35] Pang-Wei Tsai, Chun-Wei Tsai, Chia-Wei Hsu, Chu-Sing Yang, “Network monitoring in software-defined networking: A review”, IEEE Systems Journal, 12.4, pp. 3958-3969, December 2018.

[36] Gupta, Arpit, et al., “Network monitoring as a streaming analytics problem”, Proceedings of 15th ACM Workshop on Hot Topics in Networks, 2016.

[37] Keith W. Ross, James F. Kurose., "Computer networking", Pearson, 7th Edition, 2019.