Study on Multicast Network Optimization Based on SDN

アラ, モハメド, アティア, アラカニー

https://doi.org/10.15017/1866316

出版情報：九州大学, 2017, 博士（学術）, 課程博士
バージョン：
権利関係：

KYUSHU UNIVERSITY

DOCTORAL THESIS

Study on Multicast Network Optimization Based on SDN

ALAA MOHAMMED ATTIA ALLAKANY

2017

Kyushu University, Graduate School of Information Science and Electrical Engineering, Department of Advanced Information Technology, PhD Course

Declaration of Authorship

I hereby declare that this thesis entitled “Study on Multicast Network Optimization Based on SDN” is the result of my own research except as cited in the references. This dissertation has not been accepted for any degree and is not concurrently submitted in candidature of any other degree.

Signature :

Student : ALAA MOHAMMED ATTIA ALLAKANY

Date : / /2017

Supervisor : Professor Koji OKAMURA

To my family; parents, wife, and children (LOJAYN). I would not have done this without you. Thank you for your endless love, encouragement, and support.

Abstract

Today's networking infrastructure system has been maintained almost in the same form for decades, while a lot of new services have been introduced and almost these services came with new control requirements, which in turn led to increasing network complexity that facing significant networking issues, such as Quality of Service (QoS), security, mobility and management. The network research community, have proposed many new ideas for solving these networking issues. However, these ideas often include nonstandard aspects that required change in current networks. It is difficult to incorporate these changes into current network technology since, the current technology and devices are installed at a large scale, with numerous devices and protocols, and that they are mostly based on enclosed proprietary network devices, meaning that only equipment vendors can configure and create protocols. Moreover, most current network devices have an integrated control and data plane, forcing service providers to use a repetitive process to configure each device or group of devices of the same brand in an independent way. These reasons does not help the implementation of new ideas that may arise by the research community or by new requirements of network operators. It is becoming increasingly difficult for the traditional networking infrastructure, designed decades ago, to satisfy the requirements of modern application. A solution that is able to meet the future requirements as they arise is needed. This is where the philosophy of Software Defined Networking (SDN) may play an important role. The concept of SDN emerged as a proposal to overcome these limitations. SDN is the next generation of networking architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today's applications.

Recently, there are several large scale companies interested in using and transiting to SDN technology. However for an efficient transition from current network technology to SDN various issues need to be addressed. This study focuses on optimization approaches for concurrent multicast applications namely, multimedia streams in software defined networks.

Thereby, the challenges with multimedia traffic routing should be highlighted, this study address the following three important challenges in this regards.

1) IP network uses multicast as an effective method to maximize network resources utilization. However, widespread support of IP multicast is unavailable due to technical and economical reasons, leaving the floor to application layer multicast which increased traffic load for the network, because of the responsibility for management of multicast groups is distributed among network routers, routing rules calculated based on local view and difficulties to obtain on-time network traffic. SDN provides new opportunities for re-engineering multicast protocols that can address current limitations with IP multicast.

To address this problem, the features of SDN is used and a load balance approach for multicast traffic through real-time link cost and switch load modification is presented. In this approach, the OpenFlow controller is used for network load-aware by monitoring on-time network traffic then a new concept *"available link bandwidth"* and *"available switch capacity"* is presented to be used as link and switch weights respectively. The idea is that overall performance of the network could be improved and both link and switch congestion could be avoided by considering the different capacity of each link and switch by using the concept of *"available capacity"* as weights rather than using the concept of *"current utilization"* as weight. The multicast tree was calculated using the extend Dijkstra shortest path algorithm and based on real-time measuring network traffic. The proposed approach evaluated using Mininet network emulation with POX controller. The evaluation prove that the proposed method can improve traffic distribution in network.

2) The previous proposed load balance approach for multicast traffic in SDN can shows that this method can optimize and improve traffic distribution in network and can avoid network congestion. However, using Shortest Path Tree (SPT) algorithm represented by Dijkstra algorithm for calculating multicast tree often can optimize each path in the tree but can't optimize overall multicast tree. To optimize overall multicast tree the Minimum Steiner Tree (MST) algorithm is needed. MST is NP Hard problem and always there is a negotiation

between SPT that can calculate multicast tree faster than MST and the MST that can generate solutions optimum than SPT.

To that end, a novel approach for constructing the multicast tree by combines both of Dijkstra shortest path algorithm and heuristic Tabu Search (TS) algorithm is presented. In the proposed method Dijkstra algorithm and TS work respectively for fast start-up multicast session and minimizing the size of the routing tree solution with increasing in the number of multicast group size. Proposed algorithm take the advantages of both algorithms such as fast convergence time of Dijkstra algorithm and optimum solution of TS and avoid the shortages of both algorithms. The results prove that the proposed approach can improve start-up time for initialization multicast session. Also, can minimize the constructed multicast tree.

3) Latency in a network is an important parameter that can be utilized by a variety of applications which required unicast or multicast QoS policies. Several methods for monitoring latency have been introduced. However, most of these methods monitor end-to-end path delay (delay per path) by sending probes requests along the path. These methods led to redundant work and network overhead, which resulting from monitoring multiple paths between each pair of nodes. Moreover, end-to-end probes cannot monitor the delay on path segments (delay per link) between arbitrary network devices. Monitoring delay per link is more efficient than per path delay for a lot of applications. However, measuring per link delay is challenging.

To address this challenge, the link-based delay monitoring method using OpenFlow in real-time is proposed, this method does not require any complementary support from the switching hardware and can avoid redundant work and network overhead. The key idea is to build a tree that includes all the possible paths that cover all network links from the monitoring point and eliminate redundant measurement paths to reduce the number of packets for measurement. The advantage of the proposed method is the reduction of the number of OpenFlow rules and probing costs that required for monitoring. The results prove that the proposed method can avoid redundant work and network overhead.

Acknowledgements

All praise is due to Allah almighty God for all the graces, and blessings bestowed upon my family and me during my study in Kyushu University and living at Fukuoka, Japan.

“Those who do not thank people, they do not thank God”. Words may not convey the true feelings of gratitude, and appreciation towards those who stood by my side, and supported me during my study.

First of all, I would like to express my gratitude towards my supervisor Professor Koji OKAMURA, whom without his wisdom and guidance; this work would not be possible. He was always there to show me the path and guide me through the difficulties of my study and research by his wise advice, discussions, and encouragement.

I would like to show my gratitude towards my loving wife, who has always stood by my side, took my hand and encouraged me to do my best. Also not to forget my lovely daughter LOJAYN, whose laughs are the joy of my life and have always been giving me strength to go on.

I also would like to show my deep gratitude to my loving parents, who have been my first teachers as they taught me about life, and have always been encouraging me to achieve more and do my best. And also I’m very grateful to my work to my father and mother in law, whose motivation and support has been always helping me.

In addition, I would like to express my gratitude to Professor Akihiro NAKAO, Professor Yasuo OKABE, for their kind support, guidance, and enlightening discussions throughout the meetings along my PhD study. Meetings they attended were really pleasant and fruitful, thanks to their kind comments, and advice that helped a lot to shape my work.

Also, I would like to express my gratitude to Vice-Supervisors Professor Hirofumi Amano and Professor Yoshihiro Okada, for their kind support during the meeting of thesis examination.

I would like to express my gratitude towards all members of Professor Okamura's laboratory; Mr Othman Othman, Mr Nor Masri Sahri, Mr Chengming Li, Mr Zafran, Mr. Ariel, Mr. Sanouphab Phomkeona, Mr. Kristan, and all others that have their place in memory. Thanks a lot for your kind accompany, comments, encouragement, and support.

Last but not least, I would like to thank the Egyptian Ministry of High Education for supporting my study in Japan. Also, I would like to thank Kyushu University, and the Graduate School of Information Science and Electrical Engineering, and all their Professors and Officials for their kind support.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1. Introduction

IP multicast is a technology for efficient point-to-multipoint packet delivery [1]. In the years after its introduction in 1990 IP multicast has received a lot of research. Still, applications relying on IP multicast have only been rarely deployed in the open Internet [2]. It is predicted that approximately 73% of all IP traffic will be video by 2017 [3], of which some 14% will be from Internet video to TVs. Not surprisingly, streaming of live content is increasingly prevalent on the Internet replacing the traditional means of TV broadcasting. IP multicast used as efficient method to optimize network resources and to alleviate the traffic load due to streaming multimedia. However, in today's networks, IP multicast has remained largely underplayed due to concerns on security, reliability and scalability, not to mention the requirement to have all routers in the network support the related protocols and be appropriately configured [4]. For this reason, in this study SDN is introduced as new technology with new features that can cover IP network limitation to implementing IP multicast for network resources optimizations [5]. This chapter provides a brief introduction to IP multicast, Software Defined Network (SDN) and OpenFlow protocol. It also presents the motivation and problem statement. Then it presents research goals and proposed solution.

1.2 Background

Networking principles have remained mostly unchanged over the past decade. Networks are built using more or less sophisticated switches and routers. These devices are being developed by tens of vendors usually using proprietary operating system and interfaces. Building heterogeneous networks on devices from different vendors means that organization

have to employ a specialist on every router brand. Configuration of different systems also increases the probability of configuration mistakes. This issue coupled with incompatibility of different versions of systems from one vendor make heterogeneous networks difficult or very expensive to manage. There is a need for a new technology to make networks more scalable, dynamic and to allow easier management of network devices from different vendors. These needs could be fulfilled by programmable networks, i.e., by Software Defined Networking.

1.2.1. Multicast in Traditional Networks

IP multicast allows the transmission of IP packets to a group of receivers [1]. Compared to unicast delivery, multicast can reduce transmission overhead at the sender as well as overhead in the network and decrease the latency. Furthermore, IP multicast serves as a rendezvous service as a sender is not required to know specific receiver IP addresses. Similarly, the recipient of multicast data does not necessarily need to know the address of every source.

To join or start a multicast session, hosts need to announce their membership to a multicast group. Senders do not need to be members of the multicast group they are sending traffic to. The architecture of IP multicast is fully decentralized. This means, there is no central administration of group membership and routes are established on the basis of control information exchanges between multicast routers.

Traditionally, network operators ran distributed network protocols like PIM-SM [6], IGMP [7], etc. in order to build multicast routes, maintain the constructed multicast tree and management multicast session. These protocols are implemented in a distributed fashion where each router maintains a separate local view of the network topology for routing calculations, which is updated through the dissemination of link state updates. While traffic state information can be encapsulated in link state updates. By distributed manner and local view of network presented in IP network, it is difficult to implement routing protocol that can optimize network resources and avoid network congestion. Moreover, it costly to propose

monitoring systems that can monitor real-time network traffic for implementing the routing protocol with on-time network stats.

1.2.2. Software Defined Networking

Software Defined Networking is an emerging topic that tracts attention due to its paradigm. SDN proposes cleanly separating the control plane from network switches into a centralized server e.g., controller. The switches simply forward packets in the data plane using commands sent by the controller. The switches send events to the controller regarding the arrival of specific packets, flow counters, etc. Then the controller response to these events by sending commands to switches.

By decoupling of the data plane and control plane SDN cover on of the most limitation in IP network e.g., network management. Decoupling data plane and control plane enable the operator to gets a centralized view and control of the entire network from one place e.g., the controller, instead of having to configure or poll each switch independently using different interfaces. Moreover, the functionality of the switches is abstracted into a much simpler match action data plane model instead of having to worry about the code complexity that comes with running distributed protocols. In addition, the separation and simplification of the data plane enables a simple, unified and vendor agnostic control interface like OpenFlow across a heterogeneous set of network elements from different equipment vendors.

With the flexibility and efficiency presented by SDN the researcher on routing e.g., unicast and multicast can investigate there new idea with low cost and more efficient comparing with traditional network. The network operator can use the central controller to customize the network behaviour according to its needs. This means it can flexibly route network traffic on specific paths without having to use a separate protocol for each such function. In addition, the global network visibility and a unified control interface across multiple devices makes for much more efficient decision making.

1.3. Motivation and Goals

Often most new ideas for solving these networking issues that have been proposed by research community include nonstandard aspects that required change in current networks. It is difficult to incorporate these changes into current technology since, it have numerous devices and protocols that are mostly based on enclosed proprietary network devices, this means only equipment vendors can configure and create protocols. Thus, the implementation of new ideas that arise by the research community or by new requirements of network operators are difficult with current network technology. It is becoming increasingly difficult for the traditional networking infrastructure, designed decades ago, to satisfy the requirements of modern application.

Software Defined Networking and Openflow seems to be the future Internet technology that enable innovative and creative applications development that were unachieved in current traditional Internet. By controlling the functions of every network node centrally, the management and network programmability is much effortless and practical. To enable smarter future Internet, a standard body [8] has dedicated to promote and adopt SDN through open standards development. Many researchers embarked on providing new smart applications like; a virtualized network infrastructure in [9], detection of DDoS attacks [10], virtual network migration [11], for wireless mesh networks [12], measurement-aware routing [13], supporting QoS [14], multicasting [15], load balancing [16], run-time programming for network to support big data applications [17], and many others. It is believed that large number of new applications will be proposed to enhance the operation of current technologies and to provide even new applications. Several large scale companies interested in transiting to SDN technology. For example, Google has applied an SDN architecture to its private WAN called B4 [18] for improving the network performance. Consequently, the network link utilization is driven from 30-40% to near 100% by centralized traffic engineering (TE) based on the architecture.

It seem from proposals that done by researchers and several large scale companies that SDN have more advantage than the traditional IP networking. However, for an efficient transition from current network technology to SDN various issues need to be addressed. In this regard, this study address following three important challenges.

Firstly, a load balance approach for multicast traffic through real-time link cost and switch load modification for optimizing network resources and avoid link and switch congestion is proposed. Using traditional protocol for multicasting [6] can't guarantee optimal link and switch utilization and can't avoid link and switch congestion, this because depending on distributed manner and local view of network. Even, in SDN that have the global view and centralized calculation the proposed multicast approaches that build multicast tree without considering the current state of network can't optimize the utilization of network resources [19]. To end that, the OpenFlow controller is used for network load-aware by monitoring on-time network traffic state then a new concept called *available link bandwidth (ALB)* and *available switch capacity (ASC)* is used as link and switch weights respectively. The idea is that the overall performance of the network can be improved and avoiding both link and switch congestion by considering the different capacity of each link and switch by using the concept of *ALB* and *ASC* as weights rather than using the concept of "current utilization" as weight. By this method the proposed algorithm can select the most appropriate links and switch to build the multicast tree in order to maximize the network resources.

Secondly, a novel approach for constructing the multicast tree is proposed, it is a hybrid algorithm that combines both of Dijkstra shortest path algorithm and heuristic Tabu Search (TS) algorithm. The previous presented load balance approach for multicast traffic in SDN can shows that this method can optimize and improve traffic distribution in network and can avoid network congestion. However, using Shortest Path Tree (SPT) algorithm represented by Dijkstra algorithm for calculating multicast tree often can optimize each path in the tree but can't optimize overall multicast tree. Minimum Steiner Tree (MST) algorithm can find more optimum solution. However, MST is NP Hard problem and always there is a

negotiation between SPT that can calculate multicast tree faster than MST and the MST that can generate solutions optimum than SPT. To solve this problem, in the proposed hybrid algorithm Dijkstra algorithm and TS work respectively for fast start-up multicast session and minimizing the size of the routing tree solution. Proposed algorithm take the advantages of both algorithms such as fast convergence time of Dijkstra algorithm and optimum solution of TS and avoid the shortages of both algorithms. To avoid the limitation in convergence time of TS, the advantage of the centralized OpenFlow controller to back-up a partial solutions are used.

Finally, a monitoring method based on OpenFlow in real-time for measuring the link-based delay is presented. Latency is an important parameter that can be utilized by a variety of applications which required QoS policies. Most of proposed methods monitor end-to-end path delay (delay per path) by sending probes requests along the path. These methods led to redundant work and network overhead. Moreover, end-to-end probes cannot monitor the delay on path segments (delay per link) between arbitrary network devices. Monitoring latency in network segments e.g., link is more efficient than measuring per path blatancy for a lot of applications for example, calculation SPT and MST in graph. For that limitations a mentoring approach for monitoring link-based delay is presented, this approach does not require any complementary support from the switching hardware and can avoid redundant work and network overhead. The key idea is to build a tree that includes all the possible paths that cover all network links from the monitoring point and eliminate redundant measurement paths to reduce the number of packets for measurement. By this method redundant work and network overhead can be avoided also, accuracy measurement for the delay can be achieved.

1.4. Research Question

Whenever network management is considered there are two main stages: monitoring and control. Thus, the first step to achieve a scheme capable of battling those problems is to design a suitable monitoring solution. Then for maximizing network utilization of the network a new applications have to propose for optimum solution based on monitored network

statistics. The problems that this study addresses is how to monitor network utilization efficiently in real time in the context of SDN and more specifically OpenFlow then algorithms in application layer for optimization network resources can be proposed. Hence, the main goals of this study are: Use Software Defined Network as a lever to meet the future networking demands, by monitoring network statistics of each link and switch and propose load balance for multicast traffic then evaluate it via an implementation using OpenFlow.

Subsequently, the research goal of this study is to provide an answer to the following questions:

- How can the new features presented by SDN for efficiently be used to maximize network resources utilization?
- How can load balance for multicast traffic be achieved with SDN?
- How can avoid the limitation of heuristic solutions for multicast with SDN?
- How can monitoring be achieved with SDN?
- What kind of improvements could SDN bring compared to present solutions?

1.5. Organization of thesis

Background review of SDN presented in Chapter 0. Next, a load balance approach for multicast traffic through real-time link cost and switch load modification in Chapter 3. Chapter 4 describes a novel approach for constructing the multicast tree. A monitoring method based on OpenFlow in real-time for measuring the link-based delay is proposed in Chapter 5. At last, Chapter 6 concludes all the works.

Chapter 2

Literature Review

2. Literature Review

The goal of this chapter is to provide insight on the philosophy of a new network paradigm, Software Defined Networking. The chapter starts with a brief summary of traditional network architectures and limitation then summaries the history of SDN and findings that led to recent advances in the field, explaining why SDN is needed and what inherited problems it should overcome. Secondly, an explanation of the SDN basics is provided. An overview of OpenFlow, the protocol that is currently considered as the SDN standard is presented. The chapter finishes with outline the major topics touched in this study that are; multicast based on SDN and network monitoring. Through which, it intends to provide bases for better understanding for the later parts of this study.

2.1. Traditional Network Architectures and Limitation

Current network technology along the past decades still remains the same for the end users with almost no major changes, the underlying infrastructure has undergone a significant change. The overall architecture turned out to be a big success due to its simplicity and effectiveness, however, the supporting network infrastructure has been becoming a problem, where both the control plane and the data forwarding plane are involved on the same device as shown in Figure1.

Networks are growing and need to support a lot of new smart applications and protocols. Currently, they have become complex to manage which results in expenses due to maintenance and operations. The reason for such stagnation, is the fact that traditional

network is a closed system, this is, only vendors of the network devices have access to device configuration, preventing the change of device characteristics. Presently, it is very difficult to attend new application needs. Network operators and large service providers are required to follow complex maintenance procedures to achieve application needs [20].

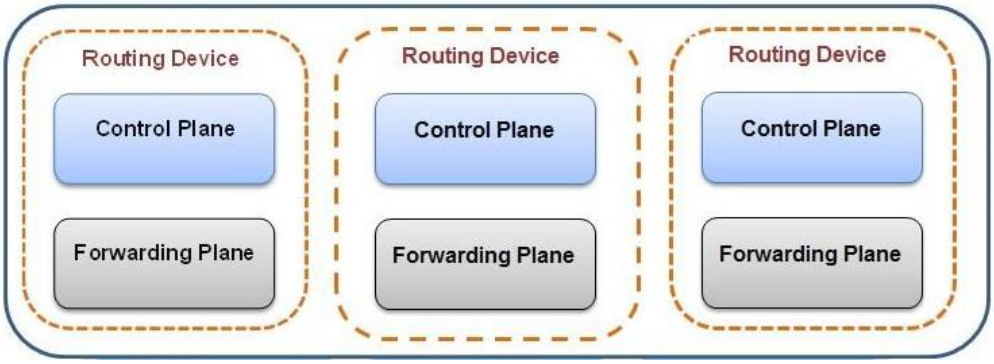

Figure 1: Traditional network Architecture.

The traditional network design has the following limitations:

- **Management Complexity:**

  Current network technologies had been built on a set of routing protocols that are engineered to connect hosts in a reliable manner over long distances with high speeds and different network designs. In order to meet the industry requirements such as high availability, security and extended connectivity, over the last decades, protocols have been designed in a lot of ways that lead to separation, where each protocol is to solve a specific kind of problems, without keeping in mind to benefit from abstractions. Such approach of design has led to one of the main problems that network administrators are facing nowadays, namely network management complexity. As a result, for any changes in the network topology or implementation of a new policy, the network operators need to configure thousands of devices and mechanisms (update ACLs, VLANs, QoS, etc.) [21]. Moreover, equipment vendor and software versions compatibility have to be considered before making any modification to the network.

As a result, network administrators keep their network rather static, in order to avoid or minimize the service downtime that can be caused by any change. Such nature of static network design is limiting the dynamic nature of server virtualization, which in turn increases the number of hosts that needs connectivity.

- **Closed Systems:**

  In Traditional network the innovation is limited by device vendors. This limitation creates a huge barrier for new ideas that may arise. With a closed system it is very difficult to have cooperation between network operators and device vendors. Operators have to know what properties and protocols have been implemented in this device, thus creating stagnation in the research of new network protocols. Companies are trying to implement new rapid-response services to the new business or user needs. However, this response capability is prevented by device vendors.

- **Scalability:**

  Every day millions of computers connect to the Internet, generating a huge amount of traffic. This process is dynamic in the sense that the amount of traffic is still increasing, changing in size and new applications are also constantly joining the flow.. Many new idea are proposed for maximize network utilization. However, having complexity problems leads to scalability problems, because their networks are no longer capable to continue growing at the same speed. Furthermore, network providers are not be able to continue investing into new equipment endlessly as they have already been heavily investing during the past decades into infrastructure.

2.2. Software Defined Network

The limitation of traditional network shown in subsection 2.1 has led the researcher community to the vision of programmable networks with Software Defined Networking (SDN) [8. The primary idea behind SDN is to move the control plane outside the switches and enable external control of data plane through a logical software entity called controller. The

controller offers northbound interfaces to network applications and southbound interfaces to communicate with data plane. OpenFlow is one of the possible southbound protocols.

2.2.1. Architecture of SDN

Normally routers, switches or any other network devices have two planes. The first plane is the forwarding plane renounceable of forwarding the data; therefore it is called the data or traffic-carrying plane. On the other hand the control plane is responsible of all the intelligence in the network and the decision making on where to route the traffic. As proposed by Open Networking Foundation (ONF) in [8], the idea of SDN represented in Figure 2 is to decouple these two planes and to transform the traditional static network into a responsive, programmable, intelligent one that can be centralized controlled.

The SDN concept consist of three planes, a short description of the planes is given below in relation with figure 2:

- **Data Plane:**

  The Data Plane is built up from Network Elements and provides connectivity. Network Elements consist of Ethernet switches, routers and firewalls, with the difference that the control logic does not make forwarding decisions autonomously on a local level. Configuration of the Network Elements is provided via the control interface with the Control Plane. To optimize network configuration, status updates from the elements are sent to a Network Controller.

- **Control Plane:**

  Network Controllers configure the Network Elements with forwarding rules based on the requested performance from the applications and the network security policy. The controllers contain the forwarding logic, normally located at switches, but can be enhanced with additional routing logic. Combined with actual status information from the Data Plane, the Control Plane can compute optimized forwarding configurations. To the application layer, an abstract view from the network is generated and shared

via a general Application Programming Interface (API). This abstract view does not contain details on individual links between elements, but enough information for the applications to request and maintain connectivity.

- **Application Plane:**

  Applications request connectivity between two end-nodes, based on delay, throughput and availability descriptors received in the abstract view from the Control Plane. The advantage over current state networks is the dynamic allocation of requests, as non-existing connectivity does not need processing at local switch level. Also applications can adapt service qualities based on received statistics. For example reduce the bandwidth for video streaming applications on high network utilization.

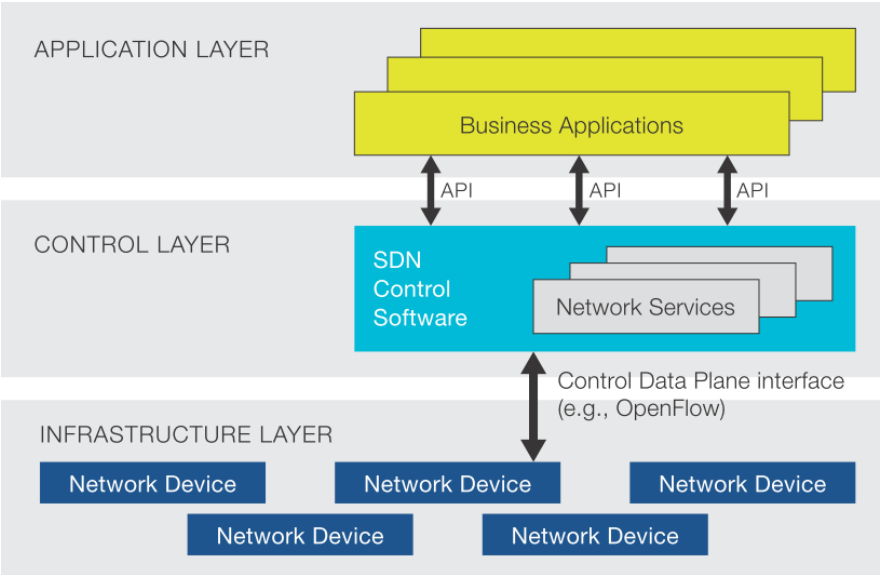

Figure 2: Software-Defined Network Architecture, as appeared in [8].

2.2.2. Introduction to OpenFlow

In order to separate control and forwarding planes, the controller needs Application Programming Interface (API) to control the forwarding plane switches. OpenFlow [22] is the first open standard communication interface defined between the control plane and the data plane in order to enable the implementation of a flexible SDN architecture by programming the flow tables within switches. The motivation behind creating an open protocol is that

switches can be developed to be vendor agnostic, greatly simplifying the task of the control plane writer. Today, OpenFlow is the most widely used SDN switch control protocol.

OpenFlow models switches as a set of one or more flow tables containing "matchaction" or "match plus action" entries. Each entry consists of a match that identifies packets and an action that specifies processing to apply to matching packets. Received packets are compared against the entries in the flow table, and the actions associated with the first match are applied to the packet. The set of available actions includes forwarding to one or more ports; dropping the packet; placing the packet in an output queue; and modifying, inserting, or deleting fields. Controllers program switches with the OpenFlow API by specifying a set of match-action entries.

- **OpenFlow Architecture**

OpenFlow defines the messaging protocol and also the semantics for changing switch states. Open- Flow networks consist of an OpenFlow Controller, OpenFlow switches (devices) and the OpenFlow Protocol, as shown in Figure 3. The OpenFlow Controller defines the rules used by the control plane. While the OpenFlow switch has the function of forwarding traffic in the network. Communication between the switch and the Controller is done through a secure, Transport Layer Security (TLS)/ Secure Sockets Layer (SSL) based, channel. Both the Controller and the switch interface implement the OpenFlow Protocol.

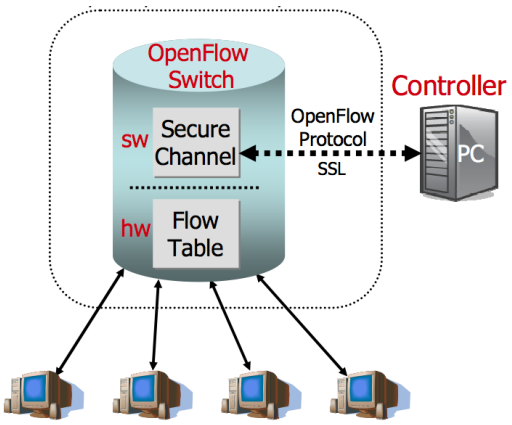

Figure 3: The OpenFlow architecture, as appeared in [22].

Any switch or router that supports OpenFlow must have a flow table, a secure channel to connect the switch or router to the OpenFlow controller and an OpenFlow protocol which is used as a protocol of communication between the switch or router and the controller over the secure channel. The flow table, which consists of flow entries see Figure 4.

Rule	Match	Action	Priority
R1	11*	Fwd 1	3
R2	1*0	Fwd 2	2
R3	10*	Fwd 3	1

Figure 4: OpenFlow table entry.

Where each flow entry consists of header fields to which the header of the incoming packet is matched against (see Figure 5), counters to provide statistics about the flow entry and actions to be performed to the matched incoming packet. The actions can be either forwarding the packet to physical port or ports, enqueue the packet in a queue attached to a physical port, dropping the packet or modifying incoming packet's header fields, which include modifying fields shown in Figure 5.

Ingress Port	Ether source	Ether dst	Ether type	VLAN id	VLAN priority	IP src	IP dst	IP protocol	IP ToS	TCP/UDP src	TCP/UDP dst

Figure 5: Header fields.

Several versions of the OpenFlow specification have been published. OpenFlow 1.0 [23] presents a switch model with a single flow table and a fixed set of fields for matching. OpenFlow 1.1 [24] extends the switch model to support multiple flow tables, adds support for MPLS matching, provides multipath support, improves tagging support, and enables virtual

ports for tunnel endpoints. OpenFlow 1.2 [25] adds support for IPv6 and extensible matching. OpenFlow 1.3 [26] adds tunneling and logical port abstractions, support for provider backbone bridging (PBB), and new quality of service mechanisms. Finally, OpenFlow 1.4 [27] adds support for optical ports, extends status monitoring, and enhances extensibility of the protocol.

2.2.3. Benefits of SDN

With the introducing of SDN, networks have become open standards, nonproprietary, and easy to program and manage. SDN will give enterprises and carriers more control of their networks, allow them to tailor and to optimize their networks to reduce the overall cost of keeping the network. Some of the main SDN benefits can be summarized below:

- **Network management Simplicity:** With SDN the network can be viewed and managed as a single node which will transfer complicated default network management tasks to be abstracted in a rather easy to manage interfaces.
- **Fast service deployment:** New features and applications can be deployed in a fast manner within hours instead of many days.
- **Automated configuration:** Manually configuration tasks such as assigning VLAN and configuring QoS can be provisioned automatically.
- **Network Virtualization:** Since servers and storage virtualization has become deployed more than before networks can benefit from SDN to be virtualized as well.
- **Reducing the operational expense:** By befitting from the automation of network deployment, a change on the network has never been easier, as a result reducing the cost of the network operation

2.3. Load Balance for Multicast Traffic

2.3.1. Optimization of Network Resources

A wide range of applications have emerged in today's Internet which require the real-time transmission of multimedia data from one or more traffic sources to a group of receivers.

Examples of these applications include Internet Protocol TV (IPTV), video and audio conferencing, multi-player games, and Virtual Local Area Networks (VLANs). While unicast delivery can be used for these applications, this results in unnecessary duplication of packets at the traffic source, and inefficient usage of network resources as these duplicate packets are carried through the network. Multicast delivery improves the efficiency of these applications by allowing the network forwarding elements to optimize delivery such that packets are only duplicated within the network when strictly necessary to reach all receivers.

Due to this fast grow of the network traffic optimizing network resources i.e., link and switch, became one of more important research topic. For efficient network resources optimization the load balancer technique are required.

Load balancing in computer networks is a technique used to spread workload across multiple network links and switches or computers. This helps improve performance by optimally using available resources and helps in minimizing latency and response time, maximizing throughput and avoiding congestion in network. Load balancing is achieved by using multiple resources i.e. multiple servers that are able to fulfill a request or by having multiple paths to a resource. Having multiple paths i.e., a combine of links and switches, with load spread out evenly across them avoids congestion at a link and switch and improves network performance.

2.3.2. Load balance in IP network

Current load balance technique or Traffic Engendering (TE) techniques in IP network mostly focuses on unicast. By contrast, compared with individual unicast, multicast can effectively optimize network resources. Existing load balance technique in IP network are based routing mechanism such as ECMP or existing routing protocols such as IS-IS or MPLS [28-31]. The Open Shortest Path First (OSPF) and IS-IS .Routing protocols do not adapt to the changes in the network condition because the link weights are static and these protocols lack any performance objectives while selecting the paths. The traffic engineering extensions

to IS-IS and OSPF standard, extends these protocols by incorporating the traffic load while selecting a path. In these approaches during link state advertisements, routers advertise the traffic load along with link costs. After routers exchange link costs and traffic loads, then they calculate the shortest path for each destination. These standards require the routers to be modified to collect and exchange traffic statistics [30], [31].

The solutions presented in IP network that consider the network load are unlike other solutions that use global network information This means that the proposed technique focuses on local information in each node. The routers exchange information about links only to their immediate neighbours. So the nodes only have the information regarding their neighbours. During multi-path routing any neighbouring node which is closer to the destination has a smaller cost than the current node. This neighbouring node is considered as a viable candidate for the next hop. The advantage of taking routing decision based on local information is that it can reduce the signalling and memory overhead. The downside to these approach is, since the nodes do not have the global knowledge of the network state, it may not result in optimum routing of the traffic. Also due to the inherent limitation of the traditional network architecture it cannot adapt to the rapid changes in the traffic pattern and it can cause oscillation in the network.

To work any proposed load balance technique in an efficient way a network load aware approach is required. When any changes happen in traffic volume the load balancer should quickly decide on how to route the traffic to different paths to balance link utilization. However, the technique that have the above characteristics are difficult to implement in the IP network architecture since the access to global information in real-time is needed, which is difficult in this paradigm.

2.3.3. Load balance in SDN

In SDN-based networks the controller can dynamically change the network state, for example, in traditional networks the link cost for routing protocols such as IS-IS are kept

static for a long period. If congestion happens in the network it may lead to poor delivery of data till the link costs are changed or the problem is resolved. However, in SDN these values can be changed more dynamically to adapt to the changes. More innovative routing mechanism can be implemented, or the existing routing protocols can be modified, so that they can change dynamically as per network state to enhance resource utilization, avoid failure and congestion, and improve QoS. With the advances in SDN several traffic engineering techniques have been introduced by the research community. Table 1 summarizes some of the TE techniques in SDN.

The author in [32] address the Load balancing in SDN for unicast routing, it introduced a multipath based forwarding traffic engineering mechanism called MSDN-TE. The goal of this mechanism is to forward the traffic in such a way that it avoids congestion on any link in the network. MSDN-TE dynamically selects the best available shortest paths and forwards the incoming traffic. This TE mechanism gathers network state information and considers the actual path's load to forward the flows on multiple paths. In [33] the author address the same problem in SDN for multicast traffic. This paper used the feature of SDN to monitor real link state on-time and assign the weight of the link based on current utilization of the link. Then, the multicast tree was constructed using dijkstra algorithm. This paper shows that modification link-cost based on current link utilization is efficient way to propose load balance approach for multicast traffic.

However, this technique used link utilization as link Wight and don't consider the different capacity of each link and assume that links that have same percent of utilization per second are equal in weight, for example, if there are two links L_1 and L_2, and the percent of utilization on both are 50%, but bandwidth of L_1 is 100 Mbps and L_2 is 10 Mbps, then the existing methods assume that both links have same weight, which increases the chance of congestion on L_2. Moreover, the switch load not considering for constructing the path or tree in case of multicast.

2.3.4. Proposed Research Objective

In order to overcome the limitations listed above, an approach for applying traffic load balancing to multicast traffic through real-time link cost and switch load modification in SDN is proposed. In this approach, a concept of *available link bandwidth (ALB)* and *available switch capacity (ASC)* is presented to be used as link and switch weights.

By this new concept ALB and ASC the proposed system can choice the best appropriate links and switch to construct multicast that optimize network resources i.e., links and switches and can also avoid congestion in network. This because this concept of using the remaining bandwidth in each link can differentiate between the links that have different bandwidth capacity. Also, in switches it is possible to differentiate between switches that have different capacities.

2.4. Heuristic algorithm for multicasting in SDN

2.4.1. Shortest Path Tree (SPT)

Multicast technology effectively reduce overall bandwidth consumption in backbone networks by around 50% compared to unicast routing [35]. To implement multicast technology in the network, calculation of multicast tree is required. A multicast tree that can minimize the total number of links of the tree i.e., Steiner Tree can optimize network resource than the shortest path tree. The current Internet multicast standard, i.e., PIM-SM, employs a shortest path tree to connect the source and destinations, and traffic engineering is difficult for PIM-SM since the path from the source to each destination is the shortest one. A shortest-path tree tends to lose many good opportunities to reduce the bandwidth consumption by sharing more common edges among the paths to different destinations. SPT is better in case of optimization each path individual. Moreover, it can reduce the calculation time required to calculate the tree comparing with Minimum Steiner Tree (MST). However, SPT is not designed to support traffic engineering.

2.4.2. The Minimum Steiner Tree (MST)

Given an undirected, weighted graph $G=(V, E)$ and a set R of nodes, called *terminals*, where $R \subseteq V$, the minimum Steiner tree problem is to find a minimum-weight tree, called the minimum Steiner tree, to span all terminals in R. When $R=S$, the minimum Steiner tree is actually the minimum spanning tree [34]. Furthermore, when each edge cost is 1, the minimum Steiner tree is the tree with the minimum number of edges to span all terminals. The minimum Steiner tree problem has been proven to be NP-hard. Thus, there probably exists no deterministic algorithm running in polynomial time complexity to solve such a problem. However, many polynomial-time complexity heuristic algorithms have been proposed to solve the problem. Although the Steiner tree minimizes the tree cost and the volume of traffic in a network, ST is computationally intensive and is not adopted in the current Internet standard.

Overall, both SPT and MST have its advantages and shortages, always there is a negotiation between SPT that can calculate multicast tree faster and optimize individual paths than MST and the MST that can generate solutions optimum than SPT for supporting traffic engineering.

2.4.3. Proposed Research Objective

To end the negating between SPT and MST, a novel approach for constructing the multicast tree by combines both of Dijkstra shortest path algorithm and heuristic Tabu Search (TS) algorithm is presented. In the proposed method Dijkstra algorithm and TS work respectively, Dijkstra algorithm for fast start-up multicast session and TS for minimizing the size of the routing tree solution when there are increasing in the number of multicast group size. Proposed algorithm take the advantages of both algorithms such as fast convergence

time of Dijkstra algorithm and optimum solution of TS and avoid the shortages of both algorithms.

To avoid the shortage of heuristic Tabu Search algorithm (computationally intensive), the features present by SDN i.e., centralized controller are used to calculate pack-up solution form the source to every destination, then this solution can be used by TS to great neighbour solution from current solution to optimize the multicast tree, chapter 4 shows more details. By this way the time required by TS to calculate the multicast tree can be reduced. In this work SDN can be help to avoid the shortage of heuristic algorithm to calculated multicast tree. Finally, this work take the advantage of the two mechanism for calculating multicast tree and avoiding its shortages.

2.5. Latency Monitoring per link in SDN

Accurate traffic monitoring is one of important issue as a key requirement for network management in order to reach QoS agreements and traffic engineering. Due to the fast grow on the applications that required End-to-End delay constraints the network monitoring has been an active research topic, particularly because it is difficult to retrieve online and accurate measurements. Most of existing latency monitoring mechanism can measure delay of all path i.e., End-to-End delay from source to destination, and can’t monitoring delay per path segment i.e., delay per link. However, measuring delay of all links in a network is required by many application. For example, if it is considered that a multicast application often it required calculation of a multicast tree. One of algorithms that calculate multicast tree is to solve Minimum Steiner problem. Finding MST required driving a weighted graph that shows each link delay in the network. To derive a weighted graph per link delay have to be calculated. Therefore, the existing monitoring method (path-based delay) lacks for supporting QoS multicast routing with end-to-end delay constraints. Generally, knowledge about link delay over the network would benefit many users and operators of network applications. Moreover, Unicast applications that required end-to-end delay or least cost delay path from source to

destination can be calculated using existing monitor methods by measuring per path delay [36,37], but these methods cannot avoid network overhead (bandwidth, CPU). The next sub-sections discusses briefly the latency monitoring in IP network and SDN and outline the limitation of monitoring latency on both technologies.

2.5.1. Delay measurements in the Internet

Because Traffic Engineering (TE) in turn, needs granular real-time monitoring information to compute the most efficient routing decisions. Latency is one of parameter that needed in TE, Many researcher proposed mechanisms to measure latency in IP network.

In [38] a project that analyzed the Internet topology and performance using active probing, used geographically distributed beacons to perform trace routes at a large scale. Its probe packets contain timestamps to compute RTT and estimate delays between measurement beacons. This method introduces additional inaccuracy due to the addition and subtraction of previously existing uncertainty margins. The author in [39] presents a solution that captures the header of each TCP/IP packet, timestamps then sends to a central server for further analysis. Multiple monitoring units need to be installed to retrieve network-wide statistics. Where the technique is very accurate (in the order of microseconds), additional network overhead is generated due to the necessary communication with the central server.

Overall, it is difficult to retrieve online and accurate measurements in IP networks due to the large number and volume of traffic flows and the complexity of deploying a measurement infrastructure.

2.5.2. Time measurements in SDN

With OpenFlow, it becomes easy to pick up switch and per-flow statistics into a centralized point. There are several proposals for monitoring QoS parameters in SDN, they mostly solve the problems of e.g. bandwidth utilization [40–42], packet loss ratio [36], packet delay per path [36,37], and route tracing [43].

In SDN, most existing solutions to estimate latency on a path generates probe packets that traverse the path and trigger PacketIn messages at the first and last switches on the path. To guide a probe along an arbitrary path, this mechanism pre-install forwarding rules at switches along the path, whose action field instructs the switch to send matched packets to the next hop switch. In addition, to generate PacketIn's, the rules at the first and last switch on the path contain send to controller as part of their action. By calculating the sending time and deportation time from the first and last switch along the path the controller can calculate path delay.

The latency is measured in these methods by end-to-end delay of path between two individual devices, often, these methods cannot calculate delay on path segments (per link) between arbitrary network devices. However, per link delay measurement can have significant importance for both service provider and application perspectives. Furthermore, the network overhead resulting of measuring per path delay.

2.5.3. Proposed Research Objective

This work proposed a method to measure per link delay in real-time to efficiently apply QoS policies, the proposed method does not require any complementary support from the switching hardware and can avoid redundant work and network overhead.

The idea in the proposed method, firstly, this method will derive a tree that covers all links in the network and minimizes the total links in each path of this tree by modifying the Dijkstra's shortest path algorithm. The source of this tree is the monitoring point MP. Secondly, this tree will be divided into different levels according to the number of hops from MP, between each two levels there is only one hop as shown in Figure 3. Finally, the advantage of OpenFlow will be used to measure the delay for individual paths of this tree at each level in order to calculate link delay as it is shown in chapter 5.

Chapter 3

Load Balance for Multicast Traffic in SDN using On-Time traffic Monitoring

3. Load Balance for Multicast Traffic in SDN using On-Time traffic Monitoring

3.1. Introduction

Recently, there is a fast growth in the applications that require transmission data from one source to a group of receivers. Multicasting technologies are a good solution for this kind of communication, they can save network resources utilization and improve network performance by distributing the packets from source to multiple receivers by duplicating packets at routers along a multicast tree. Live video streaming, video conferencing, and on-line multiplayer games are examples of these applications.

Multimedia applications such as video streaming is one of applications that have a large amount of bandwidth consumption. It is predicted that approximately 73% of all IP traffic will be video by 2017 [43], of which some 14% will be from Internet video to TVs. Not surprisingly, streaming of live content is increasingly prevalent on the Internet replacing the traditional means of TV broadcasting. For these reasons load balance mechanisms are required for optimizing network resources.

In order to optimize network resources an effective mechanisms have to propose for management these applications. However, management these applications requires accurate and timely monitoring statistics of network resources. Researchers have proposed many new ideas for managing of these applications and monitoring the network statistics, however, these

ideas often include nonstandard aspects that required change in current networks. It is difficult to incorporate these changes into IP networks since the devices in these networks do not allow changes to be made in their software systems.

Software Defined Networking (SDN) is introduced as a new technology that provides network operators more control of the network infrastructure by the following features: 1) Control and data planes are separated from each other. Therefore, network devices no longer have control functionalities. 2) Control plane is moved to an external entity called controller, and 3) data plane, is used to forward coming data based on pre-install flow in flow table. In SDN the controller and switch can communicate over the OpenFlow protocol [8]. By taking the advantages of OpenFlow that enables controllers to query for statistics and inject packets into the network, the network statics can be efficiently monitored to support the proposed load balancing mechanism for multicast traffic in SDN.

Recently, there have been several approaches for implementing multicast routing in SDN [44,45]. They mostly show that using the advantage of the SDN can optimize and improve network performance comparing with IP Network. Most of these approaches consider, static link weight [8], link utilization as link weight only without considering weight of switch [45], or link and switch *utilization* as weights to link and switch [44]. These methods led to network congestion, which is resulting from using static link weight for building multicast tree or using link utilization as link weights and don't consider the different capacity of each link and assume that links that have same percent of utilization per second are equal in weight, for example, if there are two links L_1 and L_2, and the percent of utilization on both are 50%, but bandwidth of L_1 is 100 Mbps and L_2 is 10 Mbps, then the existing methods assume that both links have same weight, which increases the chance of congestion on L_2.. However, if the *available bandwidth* of link is considered in this case, the first link has 500Mbps and the second has 5Mbps .Therefor, it is unreasonable to assign same weight to both links. Moreover,

assigning switch weight based in the current load of the switch gives same results "i.e., switches with different capacities, but have same percentage of load, will have same weights".

In contrast to other researches, an approach for applying traffic load balancing to multicast traffic through real-time link cost and switch load modification in SDN based on *available capacity* of each link and switch is proposed.

The proposed application can track the topology of the network and collect on-time statistics over the network switches, and thus is able to calculate the available bandwidth between any two points in the network and available capacity over any switch. Then, the Extending Dijkstra Algorithm is modified for calculating shortest path multicast tree based on real-time available links bandwidth and available switches capacities in the network. By introducing the concept of *"available bandwidth"* of link and *"available capacity"* of switch, the proposed approach can decide efficient weight that considers different links bandwidth and switches capacities and hence, this approach can maximize network resources utilization and avoid link and switch congestion.

3.2. Related work

In IP network, the multicast service usually requires the coordination of a set of protocols such as Protocol Independent Multicast Spare Mode (PIM-SM) and Multicast Open Shortest Path First (MOSPF) for calculating the multicast tree, and Internet Group Management Protocol (IGMP) to management the multicast sections. When, any host wants to join existing multicast group, or initialize a new multicast section, the host sends IGMP messages to its multicast router. The router listens to IGMP messages, and periodically sends out queries to discover which groups are active. Then, routing protocol calculates the routing paths to construct a source-based tree or group-based tree for sending multicast packets.

In SDN architecture, many researches handling multicast traffic, but limited researches have been proposed load balance multicast traffic in SDN. In [46] the authors proposed a new multicast algorithm, called Avlanche Routing Algorithm (AvRA), the objective of this algorithm is to minimize the multicast tree created for each multicast section by tries to find the shortest path to the nearest node in current multicast tree. However, this algorithm is designed for special topologies used in data center network "FatTree". Moreover, it builds the tree based one the shortest number of hops and didn't assign any kinds of weights to links or edges.

The author in [33] proposed a Load balance for multicast traffic based in SDN, this paper used the feature of SDN to monitor real link state on-time and assign the weight of the link based on current utilization of the link. Then, the multicast tree was constructed using dijkstra algorithm. This paper shows that modification link-cost based on current link utilization is efficient way to propose load balance approach for multicast traffic. The proposed approach is similar to this paper, the features of SDN have been used for monitoring network statistics in order to propose load balance approach for multicast traffic for maximizing the network resources utilization. But, in the proposed approach, the concept of *"available link bandwidth"* and *"available switch capacity"* have been introduced to calculate link weight and switch load respectively. On contrast of this method that uses the current utilization as link weight.

Multicast shortest path tree with minimizing the end-to-end delay have been proposed in [47]. In this paper, the author calculated the multicast tree based on Dijkstra's the shortest path algorithm [48] that considered not only the edge weights, but also the node weights for a graph derived from the underlying SDN topology. This paper shows that extended Dijkstra algorithm is more efficient comparing with others two algorithms. This is because the extended Dijkstra's algorithm takes edge weights as transmission delays over edges and takes node weights as process delays over nodes, while the other algorithms consider only edge weights or no weights. However, this paper focused in optimizing end-to-end delay and

calculated the weight of link and switch based on the current utilization of link and switch, respectively.

3.3. Load Balance for Multicast Traffic in SDN using On-Time traffic Monitoring

3.3.1. System design

In order to maximize the utilization of network resource and lighten the congestion of links and switches in SDN, the proposed architecture monitors on-time available capacity of each link and switch in the network at a periodically time set by the administrator. Then, the proposed controller using Dijistra shortest path immediately calculates the best load condition of multicast tree when receiving any request for multicast session. This proposed architecture is built based on POX controller, the component of this architecture are shown in Figure 6. The following discuss, the possibilities that OpenFlow introduces for implementing load balance for multicast traffic and present in details the modules of proposed method namely, Topology discover module, load monitoring module and multicast tree construction module.

A. *Topology discover module*

This module uses Link Layer Discovery Protocol (LLDP) to discover network topology. Most existing OpenFlow controllers support this protocol. By using the information resulting from this module, the network topology graph G (V, E) can be built, where the node set V corresponds to the switches and the edge set E corresponds to the links. Then, the data relative to the topology graph G can be send to tree construction module to build up the multicast tree.

B. *Load Monitoring Module*

In this module, the advantages of OpenFlow protocol to query port statistics from every OpenFlow switch via OFPT_STATS_REQUESTS message is used to calculate parameters weights. The controller periodically query all the switches inside the

network and the links between them. Then, the proposed application uses this information to measure the available capacity of each link and switch in the network. By, measuring the available capacity then link weight and switch weight can be assigned as in Equation 3 and Equation 6 respectively.

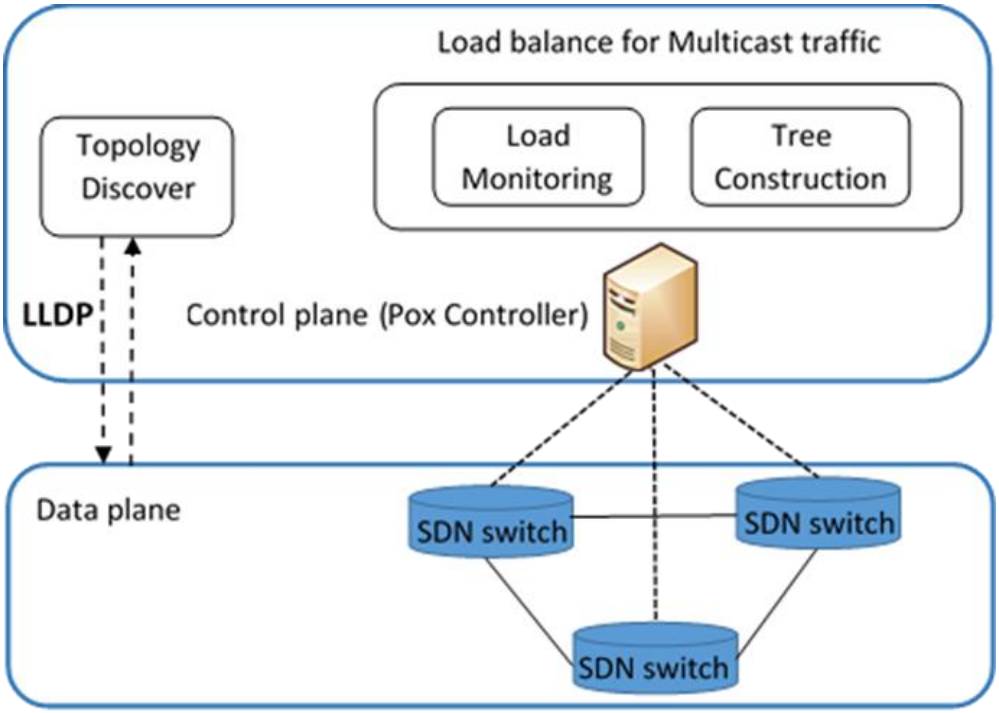

Figure 6: Proposed Architecture.

- **Available Bandwidth of link:**

Firstly, the proposed mechanism measure the available bandwidth (*AB*) for each link in the network, then *AB* is used to decide link weight as in Equation 3 .To calculate *AB*, first the proposed method measure the current bandwidth utilization (BU_i) of every link *i* in the network topology. If assumed that B_t represents the transmitted bytes at link *i* at time *t* and the interval time used for this measurement is T, this means that previous measurement was at time $(t\text{-}T)$ is $B_{t\text{-}T}$. Then, the current bandwidth utilization (BU_i) of link i is calculated according to Equation 1.

$$BU_i = \frac{B_t - B_{t-T}}{T} \tag{1}$$

The available bandwidth (AB_i) can be calculated by Equation 2, where B_i is the capacity of link *i*

$$AB_i = B_i - BU_i \tag{2}$$

The link weighted is calculated using Equation 3, where MAX_B is the maximum bandwidth capacity over the network.

$$LW_i = 1 \Big/ \frac{AB_i}{MAX_B} \tag{3}$$

- **Available capacity of switch:**

Also, to calculate the switch weight, the proposed method measure the current switch utilization *(SU_i) for every switch i in the network* according to the following equation.

$$SU_i = \sum_{j=1\ to\ n} PU_j \tag{4}$$

Where PU_j represents the utilization of port j in interval time, T and n represent the total number of ports in switch i. then, the *available switch capacity* ASC_i can be calculated using equation (5). Where, SC_i is the capacity of switch *i*.

$$ASC_i = SC_i - SU_i \tag{5}$$

Then, switch weight can be calculated using Equation [6]. Where, MAX_c is the maximum capacity of any switch in the network.

$$SU_i = \sum_{j=1\ to\ n} PU_j \tag{6}$$

- **Available capacity of switch:**

  In this module Extended Dijkstra Shortest Path Algorithm [47] is used to derive multicast tree using the link weight calculated in Equation [3] and switch weight calculated in equation [6]. In the original Dijkstra's algorithm, nodes are associated with no weight, but the Extended Dijkstra shortest path algorithm considers both the edge weights and the node weights for End-to-End routing. For a weighted, directed graph *G= (V, E),* a single source node (s) and a set of destination node (D), the Dijkstra's algorithm can return a multicast tree with shortest path on the tree from source to every destination. Paper [44] shows how extended Dijkstra algorithm can be used for deriving multicast tree. The steps of the proposed load balance for multicast traffic are shown in Figure 7.

Steps of the proposed method
1: Using Topolgy Discover module drive Graph *G= (V, E)*
2: Set the periodically time for monitoring links and switches statistics.
3: Using Load Monitoring module Calculate link weight using Eq. 3
4: Using Load Monitoring module Calculate switch weight using Eq. 6
5: For each source node S and destination group D
Use Dijikstra algorithm to calculate multicast tree
6: Install flows to OpenFlow switches

Figure 7: Steps of load balance for multicast traffic.

3.4. Performance Evaluation

To show the ability of the proposed load balance mechanism, the proposed mechanism will be evaluated using two parameters switch load and link utilization. The objective is to optimize network resources i.e., switch and link by avoid congestion. , the proposed method will be tested compared to shortest path tree proposed in [44], that don't considering any weights for link or switch. The proposed method is implemented as OpenFlow controller

modules, POX controller [49] and Mininet [50] will be used to emulate the network. A random connected topology generated using the waxman generator provided by BRITE will be used.

The proposed method assume that all link in the network have link capacity between 10Mb/s and 20Mb/s. the topology of 20 switch and 60 link is greeted. With each switch there are only one host is connected. This method use 720p video in variable bit-rate MPEG4 format for multicast session using (*VLC application*). A machine with core i3 processor and 8G or Ram used for this emulation. The network overhead resulting of both method have been tested by measuring the switch and link load.

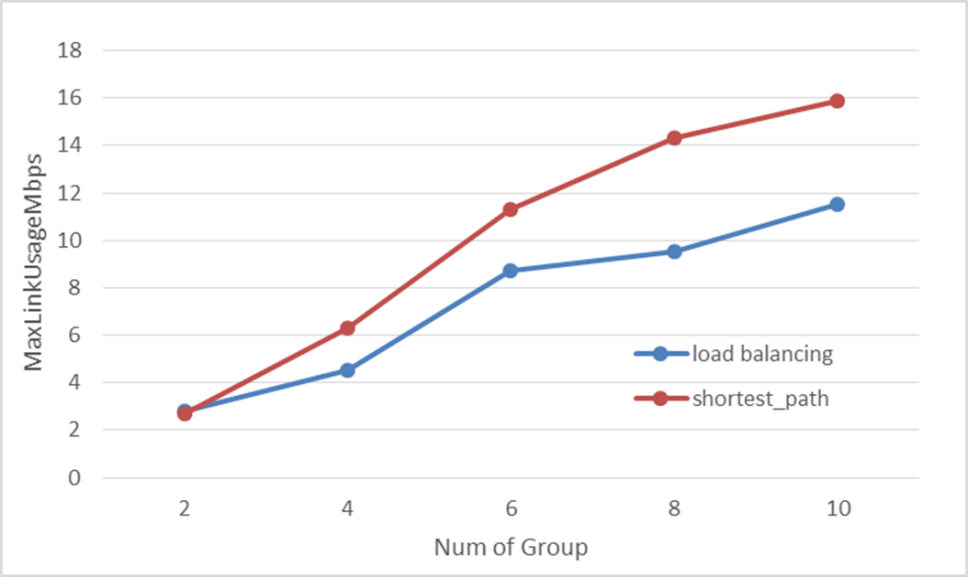

Figure. 8 The max link utilization over the network Mbps

Figure 8 and 9 show a number of multicast session, the number of multicast sessions are: 2, 4, 6, 8 and 10, with fixed group size of one source and 6 receivers. Figure 8 shows the maximum bandwidth utilization over the network with different group's number, it shows that the proposed load balancing approach is better than the other approach and can avoid the congestion over the network links. Also, Figure 9, shows the maximum switch load over the network. It shows that the result of the proposed approach is better the other approach.

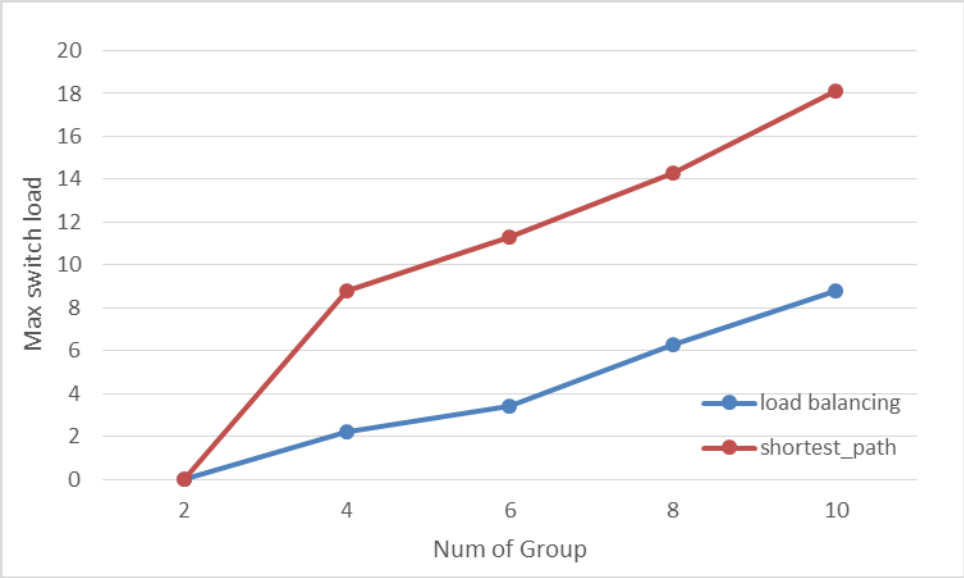

Figure. 9 The max switch load over the network Mbps

The average switch load is shown in Figure 10. It shows that the proposed approach can maintain the average switch load at a value better than the shortest path tree method. Overall, the three figure 8, 9 and 10, show that the proposed mechanism can take the advantages presented in SDN to effectively monitor network statistics and calculate link and switch weight based on the proposed concept available bandwidth and available switch load.

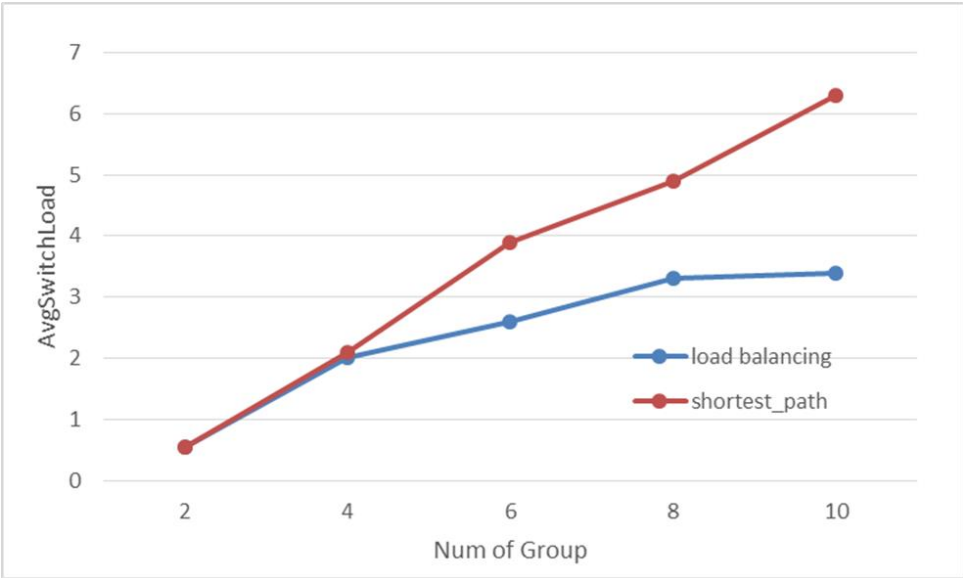

Figure. 10 The average switch load

The total numbers of flows installed over the network are shown in Figure 11. It shows that shortest path tree algorithm is better than the proposed approach, because the shortest path algorithm considers the shortest path from source to every destination on the multicast tree, but the proposed approach calculates the path with minimum links throughput and switch load, therefore, this path almost longer than the shortest path.

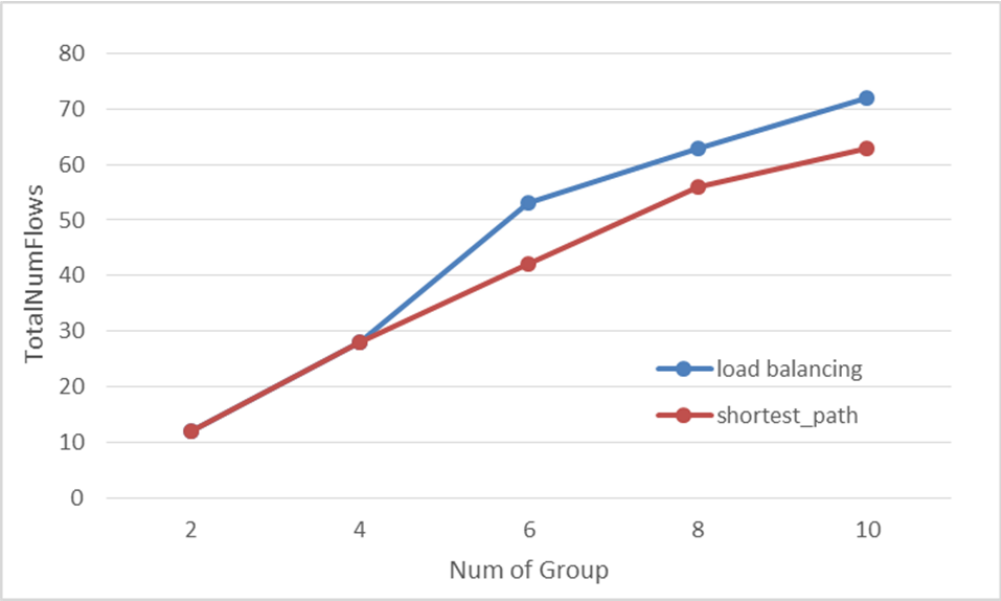

Figure. 11 The maximum number of installed flow over the network

3.5. Summary

In this work, a load balance approach for multicast traffic in SDN is proposed. The advantage of the proposed mechanism, it can calculate the routing based on calculating on-time link weigh and switch weight. Also, for efficiently optimizing network resources a new concept *"available link bandwidth"* and *"available switch capacity"* to calculate both link weight and switch weight respectively is presented. By using available link bandwidth and available switch capacity the proposed mechanism can cover the limitation in many proposed method that can't differentiate between links with different bandwidth and switches with different capacities.

By utilizing the proposed load balance multicasting controller and getting real-time calculation of available link utilization and available switch capacity, the evaluation indicates

that it is effective in reducing link and switch congestion of multicast traffic and can support Traffic Engendering TE mechanism to optimize the network resources.

Chapter 4

Heuristic algorithm for multicasting in SDN

4. Heuristic algorithm for multicasting in SDN

4.1. Introduction

IP multicasting still faces some problems: Firstly, Traditional multicast routing algorithms require routers to participate in data forwarding and control management. Then multicast routers need to maintain each group state, which arouses a lot of control overhead and add substantial complexity to routers. Secondly, Routers construct and update the multicast tree in a distributed manner; each router has only local or partial information on the network topology and group membership and there are high number of communication messages that neighboring routers have to exchange in order to update their multicast trees at every time a client joins or leaves a multicast group. These cause more latency time and difficulty to build an efficient multicast tree due to the lack of global information.

SDN is presented as a networking approach that facilitates the decoupling of the control plane in a network using a remote controller from the data plane. OpenFlow protocol [8], defines the communication between OpenFlow switches and the controller of the network. With the centralized network, OpenFlow controller has a global view of the current status of the network and can interact with its network devices. All the multicast management, such as multicast tree computing, group management are handled by this controller, and the controller has complete knowledge of the topology and the members of each group, then it can create more efficient multicast trees than the distributed approach [51]. Recently, several researchers

have been proposed multicast routing algorithms for solving the problem of shortest path tree (SPT) and Minimum Steiner tree (MST) in SDN.

A shortest-path tree, construct multicast tree by calculating the shortest path between the source and every destination node in the network, by this manner SPT tends to lose many good opportunities to reduce the bandwidth consumption by sharing more common edges among the paths to different destinations. SPT is better in case of optimization each path individual. Moreover, it can reduce the calculation time required to calculate the tree comparing with Minimum Steiner Tree (MST). However, SPT is not designed to support traffic engineering.

Steiner tree problem is to find a minimum-weight tree, called the minimum Steiner tree, to span all destination node [34]. When each edge cost is 1 in the graph, the minimum Steiner tree is the tree with the minimum number of edges to span all destination node in multicast group. The minimum Steiner tree problem has been proven to be NP-hard. Thus, there probably exists no deterministic algorithm running in polynomial time complexity to solve such a problem. However, many polynomial-time complexity heuristic algorithms have been proposed to solve the problem. Although the Steiner tree minimizes the tree cost and the volume of traffic in a network, ST is computationally intensive and is not adopted in the current Internet standard.

Both SPT and MST have its advantages and shortages, and researcher often choice the one that satisfy the required of it proposed mechanism. SPT can calculate multicast tree faster and optimize individual paths than MST and the MST that can generate solutions optimum than SPT for supporting traffic engineering.

In this work, a novel approach for constructing the multicast tree by combines both of Dijkstra shortest path algorithm and heuristic Tabu Search (TS) algorithm is presented. In the proposed method Dijkstra algorithm and TS work respectively, Dijkstra algorithm for fast

start-up multicast session and TS for minimizing the size of the routing tree solution when there are increasing in the number of multicast group size. Proposed algorithm take the advantages of both algorithms such as fast convergence time of Dijkstra algorithm and optimum solution of TS and avoid the shortages of both algorithms.

To avoid the shortage of heuristic Tabu Search algorithm (computationally intensive), this approach use the features present by SDN i.e., centralized controller to calculate pack-up solution from the source to every destination, then this solution can be used by TS to great neighbour solution by this way this approach can reduce the time required by TS to calculate the multicast tree.

4.2. Related work

Various multicast mechanism and algorithm are proposed for solving the problem of multicast routing in IP network and SDN. The author in [52] provides a mechanism to compute multicast trees centrally by flooding group membership information to all multicast routers. This mechanism (MOSPF) has a scalability problem that all routers have to compute a multicast tree per multicast group when the new multicast group appears or receivers join in or leave from multicast groups.

The Protocol Independent Multicast - Sparse Mode (PIM-SM) is the most common for IP multicasting, the routing algorithms of this protocol are not designed to build optimal routing trees [6]. PIM-SM builds trees rooted at either the source of the multicast group or at a pre-determined rendezvous point (RP) for the group.

In [53] the author has suggested high-level primitives (API) based in Open-Flow to provide a more friendly development of multicasting networks. These primitives have a simplified implementation of the OpenFlow multipoint protocol but does not consider questions such as changes in multicast groups.

The author in [54] proposed a multicast clean-slate approach logically centralized based on SDN and anticipated processing for all routes from each possible source. The author of this paper aiming to reduce event delays from source to each destination and don't consider minimizing the total edges in construction multicast tree.

Finally, in [55] this paper proposed a novel multicast mechanism based on OpenFlow to separate the data and control plane by shifting the multicast management to a remote centralized controller. The Dijkstra algorithm is used to construct spanning tree in the network and after that drive the multicast tree from existing spanning tree. This method can't construct optimum multicast tree because MST using Dijkstra algorithm can construct a tree with shortest path from source to every destination in the network but it can't minimize the multicast tree.

The objective of this work is proposing multicasting OpenFlow controller's modules for optimizing multicasting based on SDN. This work proposed a new algorithm based on two individual algorithms, Dijkstra algorithm and heuristic Tabu Search algorithm. These two algorithm work respectively to solve some multicast routing problems. In this method, the proposed multicast routing algorithm take the advantage of Shortest Path Tree algorithms represented by Dijkstra algorithm and Minimum Sterner Tree algorithms represented by Tabu Search algorithm and avoid the shortages of both. This work design SDN controller based on POX controller to achieve the following goals:

- Implementing multicasting in SDN network with using the features of OpenFlow protocol.
- Proposed a module in this controller to constructing the multicast tree based on the proposed routing algorithm.
- Reduce latency time required for initialization multicast tree.
- Construction near an optimum multicast tree (minimizing the size of the multicast tree).

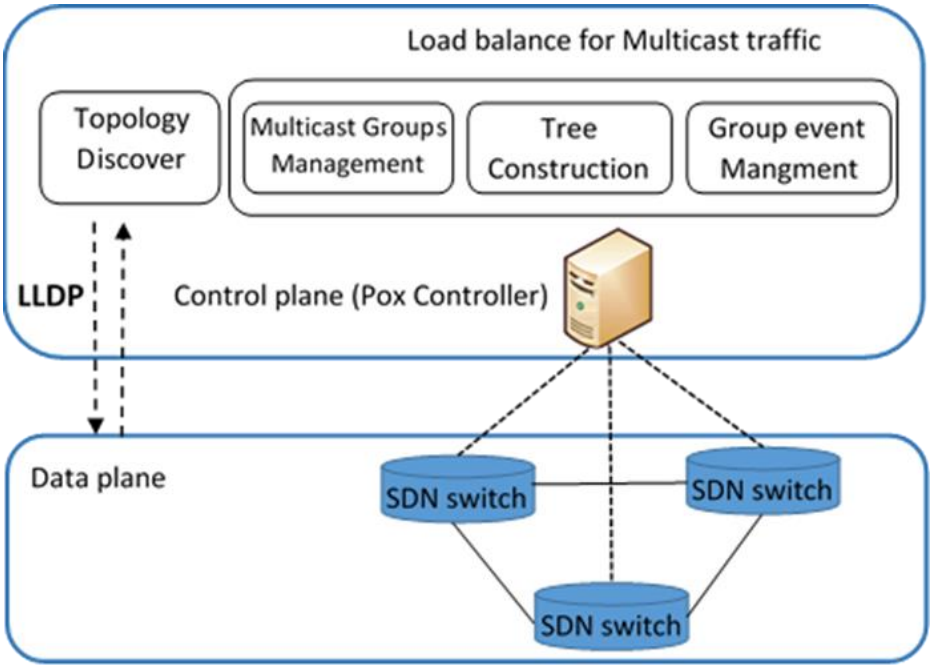

Figure 12. Proposed Architecture

4.3. Design of Multicasting controller

This section present the design of the proposed multicasting OpenFlow controller. Figure 12 show the architecture of the proposed scheme, there are two main components in this architecture, controller and OpenFlow switches (forwarder). The functions of the controller are to manage the multicast group state, construct the multicast tree and handle the host requests sent from the forwarders, then set up flow entries that are required to deliver multicast packets into the switches. While forwarders only need to receive instructions from the controller and forward data. The proposed controller consists of four modules to implement the proposed algorithm, the details and the function of each module in this controller as follow.

- **Topology discover module:**

  This module uses Link Layer Discovery Protocol (LLDP) to discover network topology. The information resulting from this module are used to build up the network topology graph G(V, E), where the node set V corresponds to the switches and the edge set E corresponds to the links. Then, the data relative to the topology graph G are send to a tree construction module to build up the tree.

- **Multicast groups management module:**

  This module is responsible for maintain multicast group state by storing sender information including locations of devices and watching IGMP packets from devices and stores the receivers' locations. Then provide this information for others openflow controller modules to construct multicast tree and management multicast group events (join in or leave any member form current multicast group).

- **Tree construction module:**

  When controller received new request to initialize a multicasting session, firstly multicast group management module process these messages to obtain sender and receiver information, then notify this module for construction the multicast tree. This module use Dijkstra shortest path algorithm for constructing initial multicast tree from source to receivers as described in next sub-section 4.4.1.

  The advantage of using this algorithm it can construct the multicast tree in short time comparing with heuristic algorithms that required a long time, moreover, the multicast tree that cover only the current receivers at initialization multicast session will be build and will not build MST that required more time and can't present an optimum solution to this problem.

- ***Group events Management module:***

  Whenever controller receives join in or leave message from current multicast sessions this module is used for updating multicast tree and install new rule to the forwarders. This module use Tabu Search algorithm to update the current multicast tree by generating neighbor's solutions of the current tree with the new receiver and choice the best solution for updating the multicast tree. The feature of SDN controller are used to calculate K-shortest path from source to every destination on offline mode and then using the pre-cached backup paths by TS algorithm to update multicast tree in

short time so the proposed method can reduce the latency time required to join new members and find more optimum solution using this heuristic algorithm compared to Shortest Multicast tree. The details of this module are described in section 4.4.2.

4.4. The purposed algorithm

These subsections described in details the implementations of algorithms proposed in *tree construction module* and *group events Management module*. In the proposed scheme a three different algorithms is used 1- Dijkstra algorithm to construct shortest path tree. 2- Tabu Search algorithm to update multicast tree. 3- K-shortest paths algorithm [56] to find a K of paths between the source and each receiver in the network and caching these calculated paths in the controller so the proposed method can use it for fast update multicast tree based on Tabu Search algorithm, by this way the time required by TS to update multicast tree can be reduced.

4.4.1. Multicast tree construction module

The main function of this module is to construct multicast tree when the controller needs to start a new multicast session. The Dijkstra shortest path algorithm is used to construct the multicast tree. This algorithm used only one time to the initialize multicast tree and don’t use for updating the multicast tree for the following reasons: 1- when the controller start to initialize multicast tree the number of the receivers almost small number, so this algorithm can construct the tree in short time. 2- This algorithm construct shortest path tree that has the least cost from source to every receiver but it can’t minimize the total number of links on the tree, so the proposed mechanism use it only in the first stage of multicasting (initialization). By this way, the proposed method take the advantage of this algorithm (i.e., fast constructing multicast tree) and avoid the shortage of this algorithm (i.e., can’t minimize multicast tree). The details of this algorithm are shown in Figure 13.

Dijkstra's Algorithm
Input: G = (V,E), Source and multicast group M Output: a tree cover all receivers in this group

1: T={S};d[S]←0; d[u]←∞ and pred[i]←nil for each u≠S, u€V
2: insert u with key d[u] into the priority queue Q, for each u€V
while (Q ≠ ∅)
3: j ← Extract-Min(Q)
4: for every node i, i €T and i is adjacent to j
alt = d[j] + ew(j, i) // ew: is edge weight =1for all links
if alt < d[i] then
d[i] ← alt
pred[i] ← j // set i as a child node of j
if node (i) not inclue in T, add the node into T
if all nodes in M join to T stop.
5: return T.

Figure 13. Dijkstra for multicast tree

4.4.2. Group events Management module

The main function of this module is to update multicast tree with the near optimum tree by minimizing the total number of links in the tree. The tabu search algorithm [57] and K-shortest path algorithm with the global information available by the centralized controller (i.e., network information) are used to update multicast tree with near optimum solutions.

TS is a higher level heuristic procedure for solving the optimization problem, designed to guide other methods or their component processes to escape the trap of local optimality. TS has obtained optimal and near optimal solutions to a wide variety of classical and practical problems in applications ranging from scheduling to telecommunications and from character recognition to neural networks. It uses flexible structures memory (to permit search information to be exploited more thoroughly than by rigid memory systems or memory less systems), conditions for strategically constraining and freeing the search process embodied in tabu restrictions and aspiration criteria. Figure 14 show TS algorithm flowchart for the optimization problem.

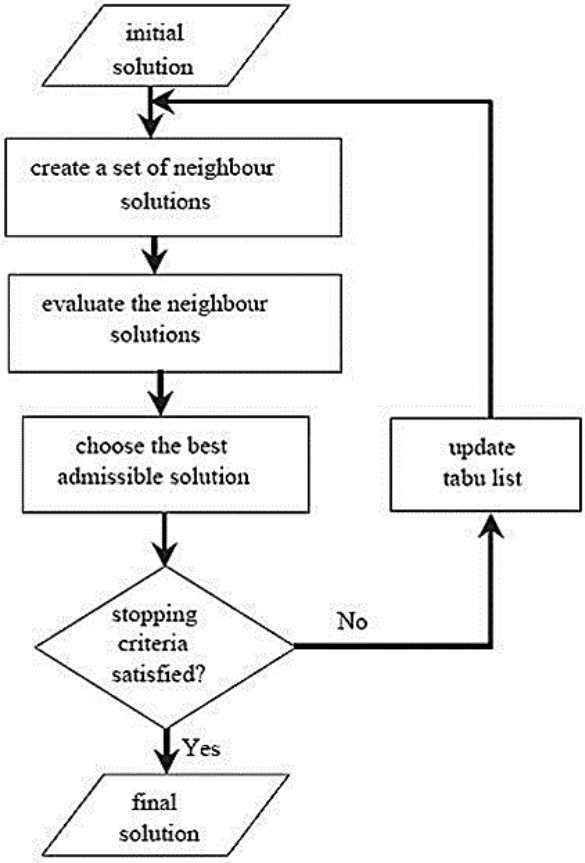

Figure 14. Tabu Search flowchart

Our proposed procedure to update multicast tree shown in Figure 15, the description of this algorithm as follow.

The algorithm first encourages the move to worth solution if there is no any improvement in the current solution. The tabu list introduced to discourage the search from coming back to previously-visited solutions and sure scape from local solution to global solutions. The following fitness function is used to evaluate each solution to select the best one for next iteration and final can find the near optimum solution.

$$F(T)=[C_T]^{-1}=\left[\sum_{i,j\forall e_{ij}\in T} C_{ij}\right]^{-1} \quad (1)$$

Here C_T is the summation of the total link cost in the multicast tree and C_{ij} represents the cost of each link and it is a predefined value by the network administrator. The next

section will give an example show how TS find near optimum multicast tree when a multicast tree need to be updated for any new join request.

```
Procedure: management multicast events (leave and join)
X      : current solution
M      : The new distention that will be add to the current tree
K      : Number of generated neighbors of X (current solution)
L (K)  : backup list of paths from source to destinations
X'     : Best neighbor solution
N(X)   : Neighborhood of solution X
Fit    : Fitness for solution X
Fit'   : Fitness for solution X'
Begin
    For I= 2 to K
        Get from L (K) the backup list for destination M
        Generate K of neighbor solution N(X) by attach the K backup path to current solution.
        Get Fit' for each new neighbor solution.
    X' = Find best solution in N(X)
    Select X' as new multicast tree
End
```

Figure 15. Steps of Tabu Search for updating multicast tree

The first step in this algorithm, it uses the current multicast tree as an initial solution. Then the algorithm used pre-cached backup list shown in table 1 to great more than one neighbor to the current multicast tree and select the best solution for next iteration. In each iteration, the TS randomly selected path from source to any destination and replaced this path by another backup path. If there is no farther improvement on the best solution then this solution will be used for updating multicast tree. Figure 16 is an example shows how this module works for updating multicast tree.

In this figure, TS algorithm generates three neighbor solutions to the current solution and the best one will be selected to join the new member to the current multicast session. The neighbors solution generated based on pre-defined backup paths. Then, for more optimization

the algorithm will repeat a specified number of iterations in each iteration it randomly selects one destination and repeats the same method to generate neighbor solution using backup paths and select the best solution for next generation. In the case of leave member from the current multicast group, only the algorithm will burn it path from the tree.

Table 1. List of backup paths from source (1) to destination (4, 7 and 8)

Destination	Path(1)	Path(2)	Path(3)
4	1,2,4	1,5,4	**1,3,5,4**
7	1,5, 7	1,5,6,7	**1,2,3,7**
8	**1,3,6,8**	**1,3,4,8**	**1,3,7,8**

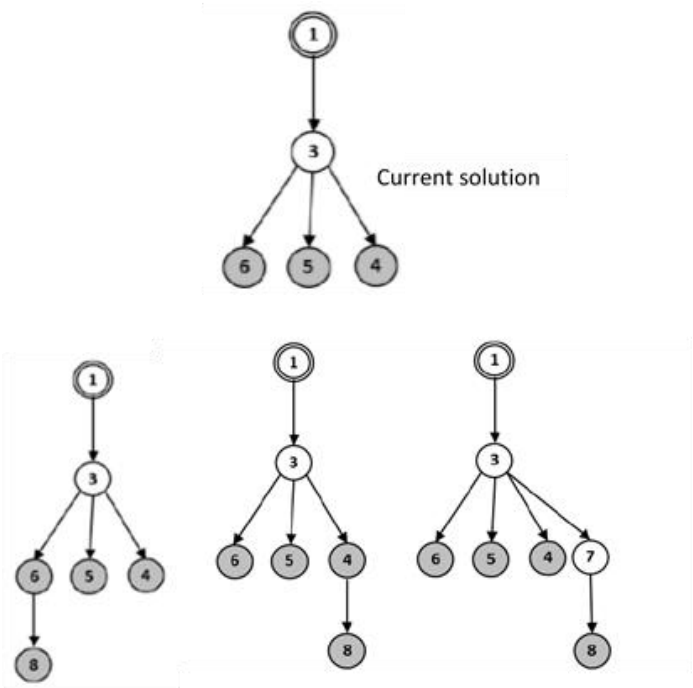

Figure 16. Generating neighbour solutions from the current tree

4.5. Experimental Results

This section test the proposed method by comparing it with Dijkstra shortest path tree algorithm in [47]. The proposed method is implemented as OpenFlow controller modules, The POX controller [9] and Mininet (10) are used to emulate the network. A random connected topology generated using the Waxman generator provided by BRITE is used in this emulation.

It is assumed that all link in the network have link capacity 20Mb/s. This emulation use 720p video in a variable bit-rate MPEG4 format for the multicast session using (VLC application). A machine with core i3 processor and 8G of Ram are used for this emulation.

Figure 17, shows the results of measuring the delay time required by the controller to construct the multicast tree at initialization of the multicast secession. A different topology size of 20 nodes to 80 nodes are used. This emulation uses one host to be a source for video streaming (h0), and the number of receivers (i.e., group size) start by four receivers with the network of size 20 nodes and increased each time by 2 receivers with increasing the network size.

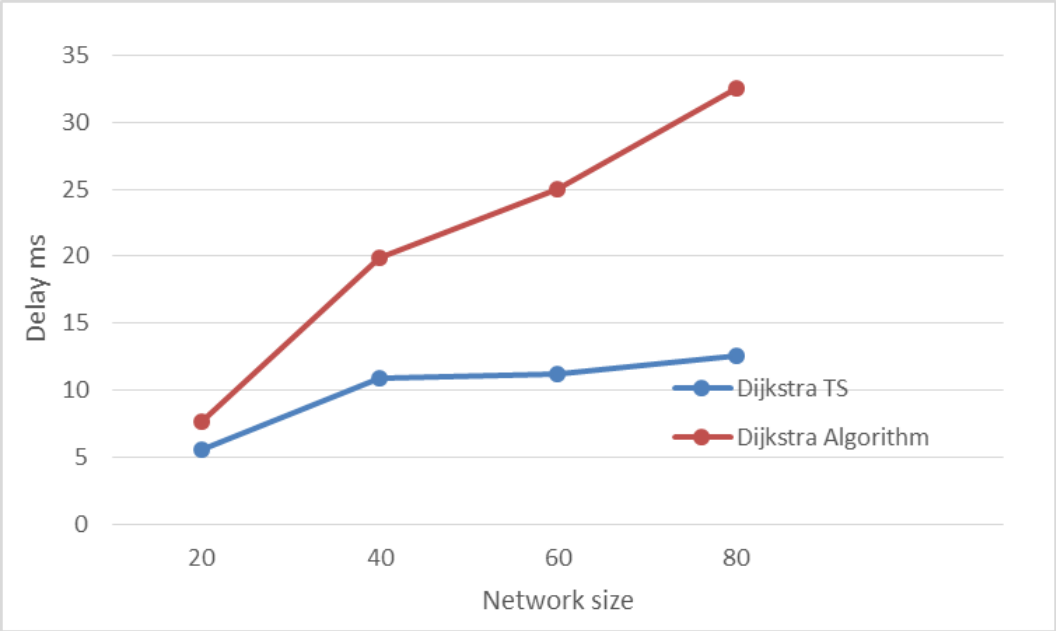

Figure 17. Delay time for initializing multicast tree.

Figure 17, shows that the proposed methods can minimize the delay time required by the controller to start the multicast session compared to the other method. This because the proposed method uses Dijkstra shortest path algorithm to construct the multicast tree from source to receivers only and then use Tabu Search algorithm to update the multicast tree when any new receiver want to join the multicast session. But in the other method purposed in [55], it constructs Minimum Spanning Tree that covers all switches in the network and then installs

only the flows that represent the active receivers from the MST, so, it required more delay to construct MST.

Table 2: Total number of flows installed per each method

	Group 1	***Group 2***	***Group 3***
Dijkstra	9 flows	10 flows	10 flows
Dijkstra &TS	7 flows	6 flows	7 flows

Table 2. Shows the installed flows in OpenFlow switches to forward the multicast data based on the constructed multicast tree. A large number of flows means the constructed multicast tree can't minimize the total hops in the tree.

This emulation start all multicast session with one source and 3 receivers, then 2 other receivers will request the controller to join the multicast session. In this case, the controller will update multicast tree to cover the new 2 receivers. The proposed method uses the backup paths to each destination and using TS algorithm will find the more optimum path to update each receiver. But, in the other method will use the back-up paths in the constructed MST to update the multicast tree.

Table 2, shows that the proposed method can reduce the total number of flows required to update multicast tree compared to another method. This means that the proposed method is able to minimize the multicast tree with any update in the tree to join new receiver to multicast session.

The reason that make the proposed algorithm able to minimize multicast tree is that, Tabu search is kind of heuristic algorithm that calculate Minimize Steiner Tree, and it is known that MST can minimize the total number of hops in the multicast tree, but it take a long time to construct multicast tree, so the proposed method use it to just update multicast tree based on pre-calculated back-up paths and used Dijkstra algorithm to initialize multicast tree because it take short time. But the other method use Dijkstra shortest path algorithm for both initialize

and update multicast tree and it know that this algorithm can find the least cost of each path but can't minimize to total links in the multicast tree.

4.6. Summery

This work presented an efficient multicasting approach based on SDN. The proposed approach can avoids the shortages resulting from building multicast tree based on shortest path algorithms or using minimum Steiner tree algorithms. Due to shortest path algorithm ex. Dijkstra algorithm can construct the multicast tree with short time but can't minimize the resulting multicast tree and on the other hand minimum stonier tree ex. Tabu Search (TS) can minimize the resulting tree but it take a long time, the proposed approach combine both algorithm to avoid the shortages of both. For constructing multicast tree the proposed hybrid algorithm takes advantage of Shortest Path Tree algorithms represented by Dijkstra algorithm and Minimum Sterner Tree algorithms represented by Tabu Search algorithm and avoid the shortages of both. By using Dijkstra to construct multicast tree at initialization of multicast session and using TS to update multicast tree in case of joining any new receivers the proposed method get better performance compared to using the only shortest path tree algorithm. In this method, the feature of SDN by using the centralized controller to construct the back-up paths for TS algorithm for fast update the multicast tree is used for more efficient solution.

Chapter 5

Latency Monitoring in Software-Defined Networks

5. Latency Monitoring in Software-Defined Networks

5.1. Introduction

Each application running on a network infrastructure may has different requirements. End-to-End delay is one of these requirements. In traditional network, it is difficult to retrieve online and accurate measurements. Most of existing latency monitoring mechanism can measure delay of all path i.e., End-to-End delay from source to destination, and can't monitoring delay per path segment i.e., delay per link. However, measuring delay of all links in a network is required by many application. For example, a multicast application often required calculation of a multicast tree. On of algorithm that calculate multicast tree is to solve Minimum Steiner problem. Finding MST required driving a weighted graph that shows each link delay in the network. To derive a weighted graph a per link delay have to be calculated. Therefore, the existing monitoring method (per path delay) lacks for supporting QoS multicast routing with end-to-end delay constraints. Generally, knowledge about link delay over the network would benefit many users and operators of network applications.

Because Traffic Engineering (TE) in turn, needs granular real-time monitoring information to compute the most efficient routing decisions. Latency is one of parameter that needed in TE, Many researcher proposed mechanisms to measure latency in IP network. In [38] a mechanism to compute RTT and estimate delays are presented in traditional netowrk. This method introduces additional inaccuracy due to the addition and subtraction of previously existing uncertainty margins. Overall, it is difficult to retrieve online and accurate

measurements in IP networks due to the large number and volume of traffic flows and the complexity of deploying a measurement infrastructure.

In SDN, there are several proposals for monitoring QoS parameters in SDN, they mostly solve the problems of e.g. bandwidth utilization [40–42], packet loss ratio [36], packet delay per path [36,37], and route tracing [43]. These method and most existing solutions to estimate latency on a path generates probe packets that traverse the path and trigger PacketIn messages at the first and last switches on the path. To guide a probe along an arbitrary path, this mechanism pre-install forwarding rules at switches along the path, whose action field instructs the switch to send matched packets to the next hop switch. In addition, to generate PacketIn's, the rules at the first and last switch on the path contain send to controller as part of their action. By calculating the sending time and deportation time from the first and last switch along the path the controller can calculate path delay.

The latency is measured in these methods by end-to-end delay of path between two individual devices. These methods cannot calculate delay on path segments (per link) between arbitrary network devices. However, per link delay measurement can have significant importance for both service provider and application perspectives. Furthermore, the network overhead resulting of measuring per path delay.

By taking the advantages of OpenFlow that enables controllers to query for statistics and inject packets into the network the monitoring system can measure QoS parameters in SDN. In this work, a method to measure link-based delay in real-time to efficiently apply QoS policies is proposed, this method does not require any complementary support from the switching hardware and can avoid redundant work and network overhead.

5.2. Related work

Due to the fast growth in the applications that have strict Quality of Service (QoS) requirements. The researcher proposed many idea to monitor network parmater for optimizing network resource based on Traffic Engineering mechanism. End-to-End delay is one of these

parameters that very important for a lot of application running on a network infrastructure and sensitive to delay. This section illustrate some related work to latency monitor in IP network and SDN.

In [38] a project that analyzed the Internet topology and performance using active probing, used geographically distributed beacons to perform trace routes at a large scale. Its probe packets contain timestamps to compute RTT and estimate delays between measurement beacons. This method introduces additional inaccuracy due to the addition and subtraction of previously existing uncertainty margins.

The author in [39] presents a solution that captures the header of each TCP/IP packet, timestamps and sends it to a central server for further analysis. Multiple monitoring units need to be installed to retrieve network-wide statistics. Where the technique is very accurate (in the order of microseconds), additional network overhead is generated due to the necessary communication with the central server.

With OpenFlow, it becomes easy to pick up switch and per-flow statistics into a centralized point. The researches use this advantage to monitor network traffic. In [40] the authors propose a monitoring method to use only the mandatory OpenFlow messages to monitor the bandwidth utilization in the network.

The authors in [41] proposed an algorithm (MonSamp) that use the Flow-Stats-Req message in OpenFlow to poll the interface and flow counters in the switches for bandwidth measurement, but this method suggests decreasing the sampling rate when the traffic load is high and that can reduce the accuracy of measurement.

In [42] the authors proposed a novel mechanism called OpenNetMon, this method offers a solution for packet lost and per path delay monitoring. To measure path delay, the controller estimates the complete path delay by calculating the difference between the packet's departure and arrival times, subtracting with the estimated latency from the switch-to-controller delays. But this method still uses per path delay measure that led to redundant

work and network overhead which resulting from monitoring multiple paths between each pairs of nodes.

This work presented to reduce network overhead for monitoring link delay to support QoS unicast and multicast applications with end-to-end delay constraints. The main contributions are to propose a new monitoring method for measuring delay per each link in OpenFlows networks.

5.3. Design

In order to measure link delay on real-time, a monitoring method based on OpenFlow is presented. In the following the details of the proposed method and discussion of the possibilities that OpenFlow introduces in the latency measurement is introduced. Figure 18, shows the modules of proposed architecture for monitoring link delay, mainly, *Topology Discover, Tree Construction* and *Latency Mentoring*:

1- ***Topology discover module:*** this module uses Link Layer Discovery Protocol (LLDP) to discover network topology.

2- ***Tree construction module:*** this module is proposed to derive a tree that covers all links in the network and minimizes the total number of hops in each path.

3- ***Latency monitoring module:*** this module is used for monitoring path delay for all paths in each level in this tree.

Tree construction module and *Latency monitoring module* are the two basic modules in this method. Using the results of these two modules, the proposed method can calculate link delay and then use the resulting data for QoS polices. The next sections explain more details about each module and how calculate the delay per link.

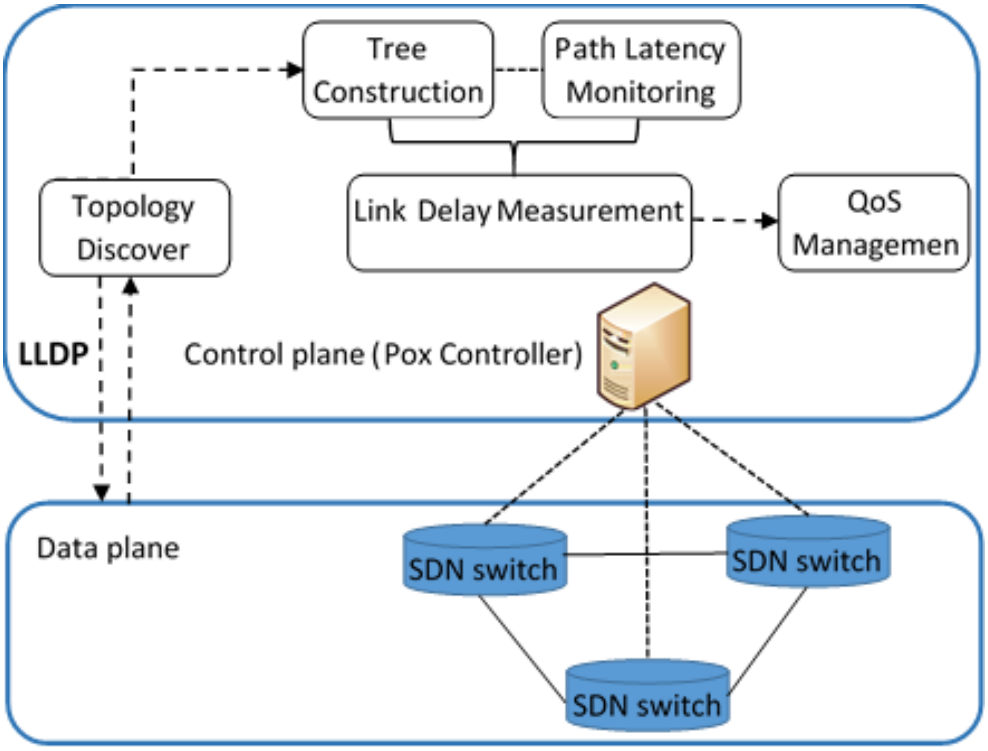

Figure 18. Proposed Architecture.

5.3.1. Topology discover module

This module is used to discover the topology of the network. The resulting information of this module is used to build up the network topology graph G (V, E), where the node set V corresponds to the switches and the edge set E corresponds to the links. Then, the data relative to the topology graph G is send to tree construction module to build up the tree.

5.3.2. Tree construction module

In order to measure link delay, this module will construct the tree that covers all links in the network. Dijikstra shortest path algorithm [58] can be used to derive Minimum Spanning Tree (MST) that cover every ***node*** in the network. In the proposed method, the Dijikstra shortest path algorithm is modified to derive the tree that cover every ***link*** in the network and minimize the total links in each path. Figure 20, shows the tree derived by this algorithm from the network topology of Figure 19.

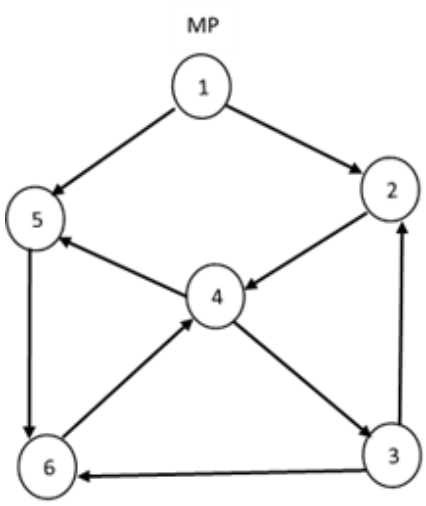

Figure 19. Network topology.

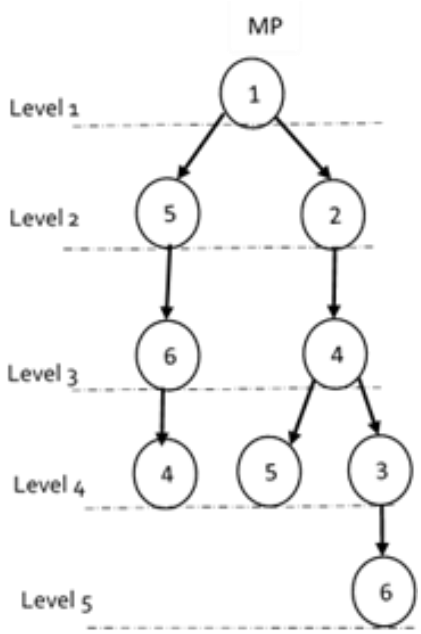

Figure 20. Tree covering all networks links.

The modified Dijikstra shortest path algorithm avoid repeating any node in the same path, but, the nodes may be repeated in different paths of the tree. This, because this tree is used to derive individual paths at each level to measure path delay by sending probes request along these paths. Figure 20, shows that node number 4 is repeated twice in the tree, but in different paths.

Table 3, shows the individual sub-paths at each level of the tree on Figure 20. This tree have three main paths ($path_1$, $path_2$ and $path_3$) and each path has it's sub-paths at different level. In this table P_{ij} represents a sub-path, i represents the path number and j represents level number, this means that P_{13} represents sub-path derived from path1 at level 3: (1 → 5 → 6).

If there are sub-paths repeated more than one time, this method will avoid monitoring repeated sub-paths in order to avoid network overhead, for example, sub-paths (P_{32} and P_{33}) and sub-paths (P_{22} and P_{23}) are the same, in this case the total sub-paths that will be monitored are 8 sub-paths according to table 1.

Table 3. Sub-paths for each path at different levels.

Paths in the tree		**Path1**	**Path2**	**Path3**
		(1 → 5 → 6 → 4)	**(1 → 2 → 4 → 5)**	**(1 → 2 → 4 → 3 → 6)**
Sub path at each level	Level 2	P_{12} (1 → 5)	P_{22} (1 → 2)	P_{32} (1 → 2)
	Level 3	P_{13} (1 → 5 → 6)	P_{23} (1 → 2 → 4)	P_{33} (1 → 2 → 4)
	Level 4	P_{14} (1 → 5 → 6 → 4)	P_{24} (1 → 2 → 4 → 5)	P_{34} (1 → 2 → 4 → 3)
	Level 5			P_{35} (1 → 2 → 4 → 3 → 6)

The advantage of using this algorithm is that, it can minimize total number of paths and links in the tree that will be used for monitoring the network. Therefore, this method can reduce network overhead. The steps of this module are summarized as the following:

Step1: constructing the tree that covers network links based on modified Dijikstra shortest path algorithm as shown in Figure 20. The steps of this algorithm are shown in Figure 22.

Step 2: dividing this tree into different levels based on the distance from the monitoring point MP, between each two levels there is one hop, as shown in Figure 20.

Step 3: for each level the individual sub-paths is derived, the data of these paths will be send to next module (Path latency monitoring) for measuring the delay of these paths.

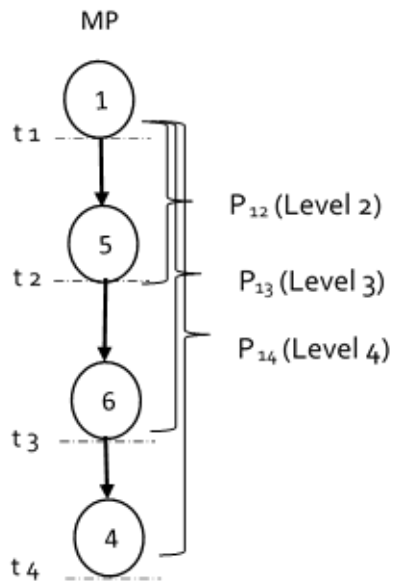

Figure 21. Example of sub-paths from MP to node 4 shows arrival time at different levels

Modified Dijkstra's Algorithm
Input: G = (V,E), MP (monitoring point) Output: a tree covering all link in the network

6: T={MP};d[MP]←0; d[u]←∞ and pred[i]←nil for each u≠MP, u€V
7: insert u with key d[u] into the priority queue Q, for each u€V
while (Q ≠ ∅)
8: j ← Extract-Min(Q)
9: for every node i, i €T and i is adjacent to j
10: alt = d[j] + ew(j, i) // ew(j, i) is edge weight =1 for all links
11: if alt < d[i] then
12: d[i] ← alt
13: pred[i] ← j // set i as a child node of j
14: if link (i,j) € T, add the link into T
15: return T

Figure 22. Dijikstra Algorithm

5.3.3. Path latency module

This module use the advantages of OpenFlow's to inject packets into the network for measuring path delay. The output data of previous module (*Tree construction module)* will be used as inputs for this module.

Regularly, at every sub-path derived from the tree a packets at the first switch will be injected, such that probe packet travels exactly the same path, and have the last switch send it back to the controller. To calculate the departure time at first switch and arrival time at last switch in the path, then the controller to first switch delay is calculated by calculating $RTT_{C\rightarrow S}$ from controller to switch, also from last switch to controller the delay is measured by calculating $RTT_{S\rightarrow C}$. Then Equation 1 and Equation 2 are used to calculate departure time at first switch and arrival time at last switch respectively. The controller estimates the complete path delay by calculating the difference between the packet's departure time at first switch and arrival time at last switch.

$$T_{First-switch} = T_{send} + (RTT_{C\rightarrow S})/2 \quad (1)$$

$$T_{Last-switch} = (RTT_{S\rightarrow C})/2 - T_{arrival} \quad (2)$$

Here T_{Send} represent the time of sending probe packet form the controller and $T_{arrival}$ represent the arrival time of the packet to the controller.

5.3.4. Link Delay Measurement

The output of both *Tree Construction module* and *Path Latency module* will be used to calculate link delay. How the link delay is calculated is discussed using example shown in Figure 4. This Figure represent path 1 from the constructed tree Figure 20, and it has three sub-paths (P_{12}, P_{13} and P_{14}), at levels ($level_2$, $level_3$ and $level_4$) respectively. In this method t_1 represents the departure time at MP, (t_2, t_3 and t_4) represent arrival times of probe packet at last switch of (P_{12}, P_{13} and P_{14}) respectively.

Calculating the link delay between switch 1 and 5, by subtract t_2 and t_1 (t_2-t_1) and to calculate the link delay between switch 5 and 6, subtract t_3 and t_2 (t_3-t_2) and so on. After calculating all links delay in the network weighted direct graph of the network topology can be derived. Then, this method store this weighted graph into the controller for applying QoS polices.

The advantages of the proposed monitoring method are: 1) Avoiding redundant work and network overhead resulting from measuring multiple paths for each pair of switch, this because the proposed method uses algorithm that can minimize the total paths and links in each path to derive the link delay. 2) Measuring latency in real time for each individual link (per link) not per path delay, then, a weighted network graph can be derived. 3) By deriving a weighted graph of the network this method can support most of optimization algorithm in unicast and multicast routing. On the other hand, per path delay can't support most of multicast application QoS polices. This method is summarized in Figure 23

Steps of the proposed method	
7: Using Dijikstra algorithm calculate the minimum tree that covering all network links	//Figure 5
8: Divide this tree into different level starting at the monitoring point MP.	// Figure 3
9: For each level on the tree derive sub-paths of this level.	// Table 1
10: Calculate delay for all sub-paths using method describe in *Path Latency module*.	
11: By subtracting the time between each two level the links delay between these two levels can be calculated.	
12: Derive the weighted graph of the network for QoS polices.	

Figure 23. Steps of the proposed method

5.4. Experimental results

This section, test the proposed method with OPENNETMON [42] mechanism. The proposed method is implemented as OpenFlow controller modules, a POX controller and Mininet are used to emulate the network. This emulation uses random partial connected

topology generated using the waxman generator provided by BRITE. It is assumed that all link in the network have same link capacity of 20Mb/s. the topology of 10 switch and 24 link is greeted. With each switch it is supposed that there are only one host is connected. For emulation multicast session this method uses 720p video in variable bit-rate MPEG4 format for unicast generated using (VLC application), the hosts H_1 connected to switch S_1 and H_2 connected to S_2 will be used for video streaming (sources). It is supposed that each host in the network will send request to each source for watching the video, then the controller calculated Least Delay Path between the source and destination. In this case OPENNETMON mechanism have to measure all available paths between source and destination and install relative flows for monitoring these paths to decide the least delay path. The network overhead resulting of both method is tested.

Table 4. Network overhead in the proposed method and OPENNETMON

	Number of probe packet To monitor network	Number of links in motoring paths
LINK-MON	15	43
OPEN-NET-MON	36	116

Table 4 shows the network overhead resulting from sending Probe packet to measure delay in the network. The total number of installed paths (i.e., the total number of probe packet that will send from controller to monitor the network) to forward probe packets and the total number of flows installed on OpenFlow switches to forwarded these packets (i.e., number of links along the paths that will be used to forwarded probe packet in each method) is compared.

This table shows that the proposed method install 15 paths and thus it need only 15 probe packets to measure all links delay in the network, moreover, the total number of links in these paths are 43 links, this means smaller number of packet will travel along the network to discover the all links delay. In the other side OPENNETMON install 36 path (i.e., the controllore need to 36 probe packets along these paths) and the total number of links on these paths are 116 links. This because, 1) the proposed method uses an algorithm to minimize the

total number of paths and links in each path, that used for monitoring all links in the network. But, OPENNETMON monitor all available paths for QoS to calculate least delay path. 2) This method measure delay per link and store the result in the controller for any new request from destination to different sources, then the path delay is a summation of number of segments by very sample calculation. But OPENNETMON have to calculate and install the flows for any new source and destinations, because it measure End-to-End delay, this mean with changing the source the controller start monitoring the ne`w paths for QoS polices.

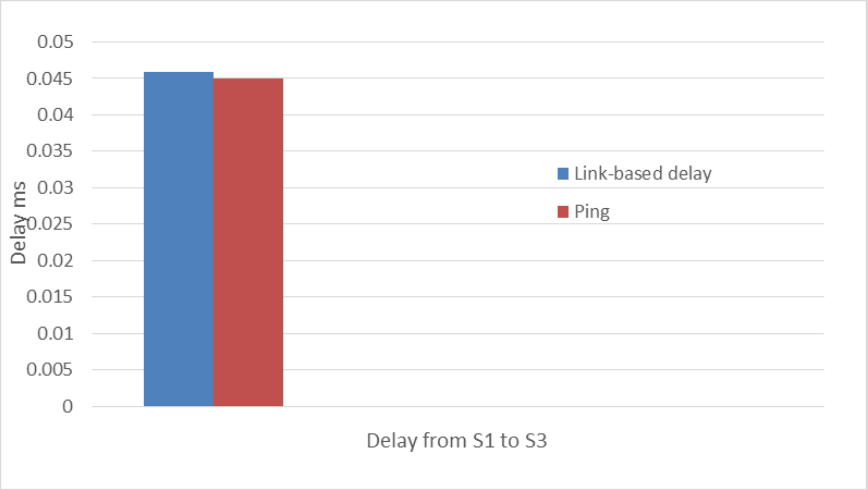

Figure 24. Accuracy of delay measurement

Fig. 7, shows the accuracy of delay measurement between the proposed method and Ping tools. For this experiment a linear topology of 5 switch is used and then measure delay between switch S_1 and S_3. In the proposed method the delay is the summation of segments (i.e., links) between the two switch. It can be seen from this figure that the proposed method have percent of 99% of accuracy comparing with measuring delay using Ping tools.

5.5. Summary

There are many proposed approaches for monitoring latency in SDN, However these methods lead to redundant work resulting from measuring same link many times by different paths, this because it is measure delay per path (path-based). This redundant measurement cause network overhead. In order to reduce network overhead for monitoring link delay to

support a verity of application that required QoS polices, this work presented a new method for monitoring latency in SDN. The proposed method have two advantages comparing with existing method. First, it can measure link-based delay rather than measuring path-based delay. Link-based delay is more efficient than path-based delay for various applications. Secondly, this method can reduce network overhead resulting from redundant work of measuring path-based delay. The idea of the proposed method is modifying the Dijkstra's shortest path algorithm to derive minimum tree that covering all link in the network, then the individual paths at deferent levels of this tree can be derived. Finally this method measure delay for each path. At this point the link delay based on the purposed idea can be calculated. The result shows that Link-based measurement can avoid network overhead and give accuracy measurement of latency.

Chapter 6

Conclusion

6. Conclusion

Software Defined Networking (SDN) and Openflow seems to be the future Internet technology that enable innovative and creative applications development that were unachieved in current traditional Internet. The core feature of SDN is to decouple the controlling function of network from the forwarding function and the network is managed from a centralized network controller. Through dynamic, cost-effective and easily adaptable, making it supreme for the dynamic nature and high bandwidth of today's network applications. With the flexibility and efficiency presented by SDN the researchers on routing e.g., unicast and multicast can investigate there new idea with low cost and more efficient comparing with traditional network. The network operator can use the central controller to customize the network behaviour according to its needs. This means it can flexibly route network traffic on specific paths without having to use a separate protocol for each such function. In addition, the global network visibility and a unified control interface across multiple devices makes for much more efficient decision making.

It seem from proposals that done by researchers and several large scale companies that SDN have more advantage than the traditional IP networking. However, for an efficient transition from current network technology to SDN various issues need to be addressed. Thus, this study focused on three significant contributions of SDN: 1) support Load balancing techniques to be able to optimize network resources and avoid both link and switch congestion. 2) Design OpenFlow controller that implement a hybrid heuristic algorithm i.e.,

Tabu Search and Dijistra Shortest path. 3) Proposed a new monitoring mechanism that can efficiently monitor latency per link i.e., path segment delay.

In chapter 3, a load balance approach for multicast traffic through real-time link cost and switch load modification for optimizing network resources and avoid link and switch congestion is proposed. Using traditional protocol for multicasting can't guarantee optimal link and switch utilization and can't avoid link and switch congestion, this because depending on distributed manner and local view of network. Even, in SDN with the global view and centralized calculation the proposed multicast approaches that build multicast tree without considering the current state of network can't optimize the utilization of network resources. Thus, this study use OpenFlow controller for network load-aware by monitoring on-time network traffic state then a new concept called *available link bandwidth (ALB)* and *available switch capacity (ASC)* to be used as link and switch weights respectively is used. The idea is that possible to improve overall performance of the network and avoid both link and switch congestion by considering the different capacity of each link and switch by using the *ALB* and *ASC* as weights rather than using of "current utilization" as weight. By this method the proposed algorithm can select the most appropriate links and switch to build the multicast tree in order to maximize the network resources. The main objectives of the proposed load balance for multicast traffic in SDN include 1) Using the features presented by SDN as centralized controller and global view of the network for efficiently monitoring and management multicast network. 2) Implementing multicasting in SDN by proposing the modules required for constructing and management multicast tree. 3) Introduce a new method for calculating the network parameters weight (i.e., link weight and switch weight) to differentiate between different capacity of links and switches in the network then the network resources can be efficiently optimized. Simulation shows that the proposed load balancing approach is better than the other approach for the maximum bandwidth and switch utilization over the network and can avoid the congestion.

Chapter 4 investigated a novel approach for constructing optimum multicast tree by combines both of Dijkstra shortest path algorithm and heuristic Tabu Search (TS) algorithm. The proposed load balance approach for multicast traffic in chapter 4 can optimize and improve traffic distribution in network and can avoid network congestion. However, using Shortest Path Tree (SPT) algorithm represented by Dijkstra algorithm for calculating multicast tree often can optimize individual paths form source to each destinations but can't optimize overall multicast tree. To solve this problem, this study presented a new approached for finding the optimum solution for multicast traffic in SDN. In the proposed method Dijkstra algorithm and TS work respectively for fast start-up multicast session and minimizing the size of the routing tree solution with increasing in the number of multicast group size. Proposed algorithm take the advantages of both algorithms such as fast convergence time of Dijkstra algorithm and optimum solution of TS and avoid the shortages of both algorithms. Dijkstra algorithm can construct the multicast tree with short time but can't minimize the resulting multicast tree, in the other hand TS can minimize the resulting tree but it take a long time. By using Dijkstra to construct multicast tree at initialization of session and using TS to update multicast tree in case of joining any new receivers a better performance can be got compared to using the only shortest path tree algorithm. This method uses the good feature on SDN by using the centralized controller to construct the back-up paths for TS algorithm for fast update the multicast tree. The results prove that the proposed approach can improve start-up time for initialization multicast session. Also, can minimize the constructed multicast tree.

In chapter 5, a latency monitoring approach for measure link-based delay is proposed. There are several methods for monitoring latency have been introduced. However, most of these methods monitor end-to-end path delay (path-based dely) by sending probes requests along the path. These methods led to some shortages include 1) Redundant work resulting from measuring same link many times by different paths. 2) Network overhead resulting from sending many probes packet to network for measuring all available paths in the network. 3)

Inaccuracy measurement of delay because of many packets that send from controller to measure the delay make the traffic in the network unreal traffic. 4) End-to-end probes cannot monitor the delay on path segments (link-based delay) between arbitrary network devices. Monitoring delay per link is more efficient than per path delay for a lot of applications. To end these limitations, the proposed method modified the Dijkstra's shortest path algorithm to derive minimum tree that covering all link in the network, then this method derive individual paths at deferent levels of this tree and measure delay for each path. At this point the link delay based on the proposed idea can be calculated. The objective of the proposed idea includes 1) Avoiding redundant work and network overhead by finding the minimum number of paths that can monitor all links in the network. 2) Achieve accuracy Measurement of latency by minimizing the number of traffic (i.e., probes packets) that send form the controller to network then the proposed method can measure delay in the network with real traffic. Simulation shows that the proposed method can achieve accuracy for measuring link-based delay and can avoid network overhead.

References

[1] Deering, Stephen E., and David R. Cheriton. "Multicast routing in datagram internetworks and extended LANs." ACM Transactions on Computer Systems (TOCS) 8.2 (1990): 85-110..

[2] Diot, Christophe, et al. "Deployment issues for the IP multicast service and architecture." Network, IEEE 14.1 (2000): 78-88.

[3] Cisco Systems Inc., “The zettabyte era – trends and analysis,” White Paper, May 2013.

[4] C. Diot, B. Levine, B. Lyles, H. Kassem, and D. Balensiefen, “Deployment issues for the IP multicast service and architecture,” IEEE Network, vol. 14, no. 1, pp. 78–88, Jan. 2000

[5] Open Networking Foundation, “Software-defined networking: The new norm for networks,” ONF White Paper, Apr. 2012.

[6] Farinacci, Dino, et al ,"Protocol independent multicast-sparse mode (PIM-SM): Protocol specification." (1998).

[7] Cain, Brad, et al. "Internet group management protocol, version 3." , (2002).

[8] Open Networking Foundation, “Software-Defined Networking: The New Norm for Networks,” Open Networking Foundation, 2012.

[9] H. Shimonishi and S. Ishii, “Virtualized network infrastructure using OpenFlow,” in *Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP*, 2010.

[10] R. Braga, E. Mota and A. Passito, “Lightweight DDoS flooding attack detection using NOX/OpenFlow,” in *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*, 2010.

[11] P. Pisa, N. Fernandes, H. Carvalho, M. Moreira, M. Campista, L. Costa and O. Duarte, “Openflow and Xen-based virtual network migration,” *Communications: Wireless in Developing Countries and Networks of the Future*, pp. 170-181, 2010.

[12] P. Dely, A. Kassler and N. Bayer, “Openflow for wireless mesh networks,” in *Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on*, 2011.

[13] G. Huang, C. Chuah, S. Raza and S. Seetharaman, “Dynamic measurement-aware routing in practice,” *Network, IEEE*, Vols. 25,3, 29-34.

[14] B. Sonkoly, A. Gulyas, F. Nemeth, J. Czentye, K. Kurucz, B. Novak and G. Vaszkun, “On QoS Support to Ofelia and OpenFlow,” in *Software Defined Networking (EWSDN), 2012 European Workshop on*, 2012.

[15] Y. Nakagawa, K. Hyoudou and T. Shimizu, “A management method of IP multicast in overlay networks using openflow,” in *Proceedings of the first workshop on Hot topics in software defined networks*, 2012.

[16] R. Wang, D. Butnariu and J. Rexford, “OpenFlow-based server load balancing gone wild,” in *Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services*, 2011.

[17] G. Wang, T. Ng and A. Shaikh, “Programming your network at run-time for big data applications,” in *Proceedings of the first workshop on Hot topics in software defined networks*, 2012.

[18] Jain, S., et al. 2013. B4: Experience with a globally-deployed software defined WAN. ACM SIGCOMM Computer Communication Review. 43, 4, 3-14.

[19] J. Jiang, W. Yahya, and M. Ananta. “Load Balancing and Multicasting Using the Extended Dijkstra’s Algorithm in Software Defined Networking” ADFA, Springer-Verlag Berlin Heidelberg 2011.

[20] ONF: Software-defined networking: The new norm for networks (April 2012)

[21] ONF. Software-defined Networking: The New Norm for Networks. White Paper, April 2012.

[22] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S.,Turner, J.: Openflow: enabling innovation in campus networks. SIGCOMM Comput. Commun. Rev. 38(2), 69–74 (Mar 2008)

[23] OpenFlow Switch Specification, Dec. 2009. Version 1.0.0.

[24] OpenFlow Switch Specification, Feb. 2011. Version 1.1.0.

[25] Open Networking Foundation. OpenFlow Switch Specification, Dec. 2011. Version 1.2.0.

[26] Open Networking Foundation. OpenFlow Switch Specification, Apr. 2013. Version 1.3.2.

[27] Open Networking Foundation. OpenFlow Switch Specification, Aug. 2014. Version 1.4.0.

[28] B. Fortz and M. Thorup, “Internet traffic engineering by optimizing OSPF weights”, in Proc. 19th IEEE Ann. Joint Conf. of the IEEE Comp. & Commun. Soc. INFOCOM

2000, Tel Aviv, Israel, 2000, vol. 2, pp. 519–528.

[29] X. Xiao, A. Hannan, B. Bailey, and L. M. Ni, “Traffic engineering with MPLS in the Internet”, Network, vol. 14, no. 2, pp. 28–33, 2000.

[30] K. Ishiguro, A. Lindem, A. Davey, and V. Manral, “Traffic engineering extensions to OSPF Version 3”, RFC 5329, IETF Trust, 2008 [Online]. Available: https://tools.ietf.org/html/rfc5329

[31] T. Li and H. Smit, “IS-IS extensions for Traffic Engineering”, RFC 5305, IETF Trust, 2008 [Online]. Available: https://tools.ietf.org/ html/rfc5305

[32] K. T. Dinh, S. Kukliński, W. Kujawa, and M. Ulaski, “MSDNTE: Multipath Based Traffic Engineering for SDN”, in Intelligent Information and Database Systems. Asian Conference on Intelligent Information and Database Systems, N. T. Nguyen, B. Trawiński, and R. Kosala, Eds. Springer, 2016, pp. 630–639.

[33] A. Craig, B. Nandy, I. Lambadaris, and P. Ashwood-Smith. “Load Balancing for Multicast Traffic in SDN using Real-Time Link Cost Modification” ICC, IEEE, 2015, page 5789-5795.

[34] M. Imase and B. M. Waxman, 1991, "Dynamic Steiner tree problem," SIAM lournal on Discrete Mathematics, vol. 4, no. 3, pp. 369-3S4

[35] R. Malli, X. Zhang, and C. Qiao, “Benefit of multicasting in all-optical networks,” Proceedings of the SPIE, vol. 3531, pp. 209–220, Nov. 1998.

[36] Adrichem N. van, Doerr C., and Kuipers F. 2014. Opennetmon: Network monitoring in openflow software-defined networks. In Network Operations and Management Symposium (NOMS), 2014 IEEE, pages 1–8, May 2014.

[37] Phemius K. and Bouet M. 2013. Monitoring latency with openflow. In 9th International Conference on Network and Service Management (CNSM), pages 122–125, 2013.

[38] B. Huffaker, D. Plummer, D. Moore, and K. Claffy, “Topology discovery by active probing,” in Applications and the Internet (SAINT) Workshops, 2002. Proceedings. 2002 Symposium on. IEEE, 2002, pp. 90–96.

[39] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot, “Packet-level traffic measurements from the sprint ip backbone,” Network, IEEE, vol. 17, no. 6, pp. 6–16, 2003.

[40] Yu C., Lumezanu C., Zhang Y., Singh V., Jiang G., and Madhyastha H. 2013. Flowsense: Monitoring network utilization with zero measurement cost. In Passive and Active Measurement, volume 7799 of Lecture Notes in Computer Science, pages 31–41. 2013.

[41] Raumer D., Schwaighofer L., and Carle G. 2014. MonSamp: A distributed SDN application for QoS monitoring. In Federated Conference on Computer Science and Information Systems (FedCSIS), Sept. 2014.

[42] Adrichem N. van, Doerr C., and Kuipers F. 2014. Opennetmon: Network monitoring in openflow software-defined networks. In Network Operations and Management Symposium (NOMS), 2014 IEEE, pages 1–8, May 2014.

[43] Agarwal K., Rozner E., Dixon C., and Carter J. 2014. SDN traceroute: Tracing sdn forwarding without changing network behavior. In Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, pages 145–150, 2014.

[44] J. Jiang, W. Yahya, and M. Ananta. “Load Balancing and Multicasting Using the Extended Dijkstra’s Algorithm in Software Defined Networking” ADFA, Springer-

Verlag Berlin Heidelberg 2011.

[45] C. A. C. Marcondes ; T. P. C. Santos ; A. P. Godoy ; C. C. Viel ; C. A. C. Teixeira. “CastFlow: Clean-slate multicast approach using in-advance path processing in programmable networks” Computers and Communications (ISCC), July 2012 IEEE Symposium on.

[46] A. Iyer, P. Kumar, V. Mann, “Avalanche: Data center Multicast using Soft-ware Defined Networking”, IEEE Communication Systems and Networks (COMSNETS), Sixth International Conference, 2014

[47] J. Jiang, H. Huang, J. Liao, and S. Chen, "Extending Dijkstra’s Shortest Path Algorithm for Software Defined Networking," Technical Report, National Central University, 2014

[48] E. Dijkstra, “A note on two problems in connexion with graphs,” Numerische mathematik, vol. 1, no.1, 1959, pp. 269-271.

[49] NOX and POX SDN Controllers. retrieved: Sept.,2015.

[50] Mininet, An Instant Virtual Network on your Laptop (or other PC), http://mininet.org/, last accessed on June 2014.

[51] C. Marcondes, T. Santos, A. Godoy, C. Viel, and C. Teixeira , Jul. 2012 "CastFlow: Clean-slate Multicast Approach using In-advance Path Processing in Programmable Networks", IEEE Symposium on Computers and Communications (ISCC).

[52] J. Moy, Mar. 1994 “Multicast Extensions to OSPF,” RFC 1584 (Historic), Internet Engineering Task Force. [Online]. Available: http://www.ietf.org/rfc/rfc1584.txt.

[53] K.-K YAP; T.-Y. HUANG; DODSON, B.; LAM, M.S.; MCKEOWN, N. 2010. Towards Software-Friendly Networks. In: ACM ASIA-PACIFIC WORKSHOP ON SYSTEMS,

1, New York, 2010. Proceedings… New York, p. 49-54.

[54] A. C. Cesar Teixeira, 2012 "CastFlow: Clean-slate multicast approach using in-advance path processing in programmable networks", ISCC, 2012, 2013 IEEE Symposium on Computers and Communications (ISCC), 2013 IEEE Symposium on Computers and Communications (ISCC) 2012, pp. 000094-000101, doi:10.1109/ISCC.6249274.

[55] J. -Ruey Jiang1, W. Yahya1,2, and M. Tri Ananta, 2011, “Load Balancing and Multicasting Using the Extended Dijkstra’s Algorithm in Software Defined Networking” , © Springer-Verlag Berlin Heidelberg.

[56] D. Eppstein. 1994. Finding the k shortest paths. 35th IEEE Symp. Foundations of Comp. Sci., Santa Fe, 1994, pp. 154-165. Tech. Rep. 94-26, ICS, UCI, 1994

[57] F. Glover. 1989. "Tabu Search - PART 1". ORSA Journal on COMPUTING 1 (2): 190–206. doi:10.1287/ijoc.1.3.190.

[58] Dijkstra E. 1959. “A note on two problems in connexion with graphs,” Numerische mathematik, vol. 1, no.1, 1959, pp. 269-2

Published Papers

(1) Alaa Allakany and Koji Okamura, Distributed GA for Popularity Based Partial Cache Management in ICN, Proceedings of the 9th International Conference on Future Internet Technologies, ser. CFI ' 14. New York, NY, USA: ACM, pp. 21-34, 2014.

(2) Alaa Allakany and Koji Okamura, Multiple constraints QoS multicast routing optimization algorithm based on Genetic Tabu Search Algorithm ACSIJ Advances in Computer Science: International Journal, Vol. 4, Issue 3, No.15, pp. 118-125, 2015.

(3) Alaa Allakany and Koji Okamura, Multicasting Tabu Search Mechanism with near optimum multicast tree on OpenFlow, Proceedings of the 40th Asia Pacific Advanced Network (APAN) on Research Network Workshop, pp. 29-33. 2015.

(4) Alaa Allakany and Koji Okamura, Fast Switching Mechanism for Multicasting Failure in OpenFlow Networks, World Academy of Science, Engineering and Technology. Proceedings of the 18th International Conference on Computer and Information Technology, ICCIT 2016. Roma, Italy, pp. 21-22. 2015.

(5) Alaa Allakany and Koji Okamura. Efficient Multicasting Algorithm Using SDN. International Journal of Computer Science and Network Security. Vol. 17 No. 4 pp. 292-297. 2017.

(6) Alaa Allakany and Koji Okamura, Load Balance for Multicast Traffic in SDN using OnTime traffic Monitoring. International Journal of Computer Science and Information Security. Vol. 15 No. 4. pp. 372-377. 2017.

(7) Alaa Allakany and Koji Okamura, Latency Monitoring in Software-Defined Networks, Proceedings of the 12th International Conference on Future Internet Technologies, ser. CFI. Japan. 2017.