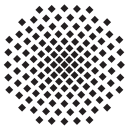


Universität Stuttgart


Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master's Thesis Nr. 0202-0005

# Packet Scheduling Algorithms for a Software-Defined Manufacturing Environment

Sujata Roy Chowdhury

| | |
|---|---|
| **Course of Study:** | INFOTECH |
| **Examiner:** | Prof. Dr. Kurt Rothermel |
| **Supervisor:** | M.Sc Naresh Nayak |
| **Commenced:** | 2015-6-01 |
| **Completed:** | 2015-12-01 |
| **CR-Classification:** | C.2.1, C.2.5 |

# Abstract

With the vision of Industry 4.0, Internet of things (IoT) and Internet of Services (IoS) are making their way to the modern manufacturing systems and industrial automation. As a consequence, modern day manufacturing systems need wider product variation and customization to meet the customer's demands and survive in the competitive markets. Traditional, dedicated systems like assembly lines cannot adapt the rapidly changing requirement of today's manufacturing industries.

A flexible and highly scalable infrastructure is needed to support such systems. However, most of the applications in manufacturing systems require strict QoS guarantees. For instance, time-sensitive networks like in industrial automation and smart factories need hard real-time guarantees. Deterministic networks with bounded delay and jitter are essential requirement for such systems. To support such systems, non-deterministic queueing delay has to be eliminated from the network. To this end, we present Time-Sensitive Software-Defined Networks (TSSDN) with a logically centralized controller which computes transmission schedules based on the global view of the network. SDN control logic computes optimized transmission schedules for the end hosts to avoid in network queueing delay. To compute transmission schedules, we present Integer Linear Programming and Routing and Scheduling Algorithms with heuristics that schedule and route unicast and multicast flows. Our evaluations show that it is possible to compute near optimal transmission schedules for TSSDN and bound network delays and jitter.

# Acknowledgements

I would like to take the opportunity to acknowledge them, without whom this thesis would never have been possible. First and Foremost, I would like to thank Prof. Dr. Kurt Rothermel for giving me an opportunity to do my thesis in the Department of Distributed Systems.

My deepest gratitude is towards my supervisor, M.Sc Naresh Nayak to give me a topic which truly interests me. I sincerely owe my gratefulness for his motivation, constant guidance, extraordinary patience and support that he has bestowed to complete this work. Without his encouragement and involvement, this thesis would not have been viable.

I would like to thank Dr.Frank Dürr for invaluable inputs on Integer Linear Programming and insightful discussions on various technical doubts.

Special thanks to my friends Buddhadeb Chakraborty and Raghavendra Lingamaneni for all their support and motivation. I am really thankful to them for creating an enjoyable work environment throughout the entire duration of this thesis.

Finally, I would like to thank my family for being there always for me. I consider myself to be fortunate to be blessed and supported by them always.

# Contents

# List of Figures

# Glossary

**MIMTMS** Multicast ILP based multiple trees multiple demands. vi, 38

**MIMTSS** Multicast ILP based multiple trees single demands. vi, viii, 36, 56–60

**MISTMS** Multicast ILP based single tree multiple demands. vi, 39

**MISTSS** Multicast ILP based single tree single demands. vi, viii, 37, 56–59

**MIUSS** Multicast ILP based with unicast flow with same slot. vi, viii, 41, 56–60

**MNIMTMS** Multicast Non ILP based with multiple trees multiple demands. vi, 43

**MNIMTSS** Multicast Non ILP based with multiple trees single demands. viii, 44, 56, 59–61

**MNISTSS** Multicast Non ILP based with single trees single demands. viii, 44, 56, 59, 60

**UIFM** Unicast ILP based fixedpath heuristics for unicast flow scheduling with multiple demands. v, 26, 48–50, 53, 54

**UIFS** Unicast ILP based fixedpath heuristics for unicast flow scheduling with single demand. v, 24

**UIPM** Unicast ILP based Path-set heuristics for unicast flow scheduling with multiple demands. v, 25, 48–50, 53, 54

**UIPS** Unicast ILP based Path-set heuristics for unicast flow scheduling with single demand . v, 22

**UNIFM** Unicast Algorithm (non-ILP) based fixed for unicast flow scheduling with multiple demands. vi, 30, 48, 51, 52

**UNIFS** Unicast Non ILP based with fixed path single demand. 31

**UNIMS** Unicast Non ILP based with path-sets single demand. 31

**UNIPM** Unicast Algorithm (non-ILP) based path-sets for unicast flow scheduling with multiple demands. vi, 29, 48, 51, 52

# Chapter 1

# Introduction

Over the decades, the automation industry has gone through three major revolutions in the form of mechanisation, electrification, and information technology(IT) (see figure 1.1). In earlier days, the variation in the manufacturing systems were limited. For instance, in case of automobile industry it was rightly described by Henry Ford's statement, "they can have any color as long as it is black." The fourth industrial revolution with Internet of Service(IoS) and Internet of Things(IoT) is now paving its way to the automation industries[39]. In this modern era of Industry 4.0, automation industry needs wider product variations and customizations to meet the customers demands and survive in the competitive market. Moreover, the manufacturing systems are expected to have a shorter product cycle and be cost effective. Existing dedicated systems such as assembly lines are not competent enough to meet these requirements.

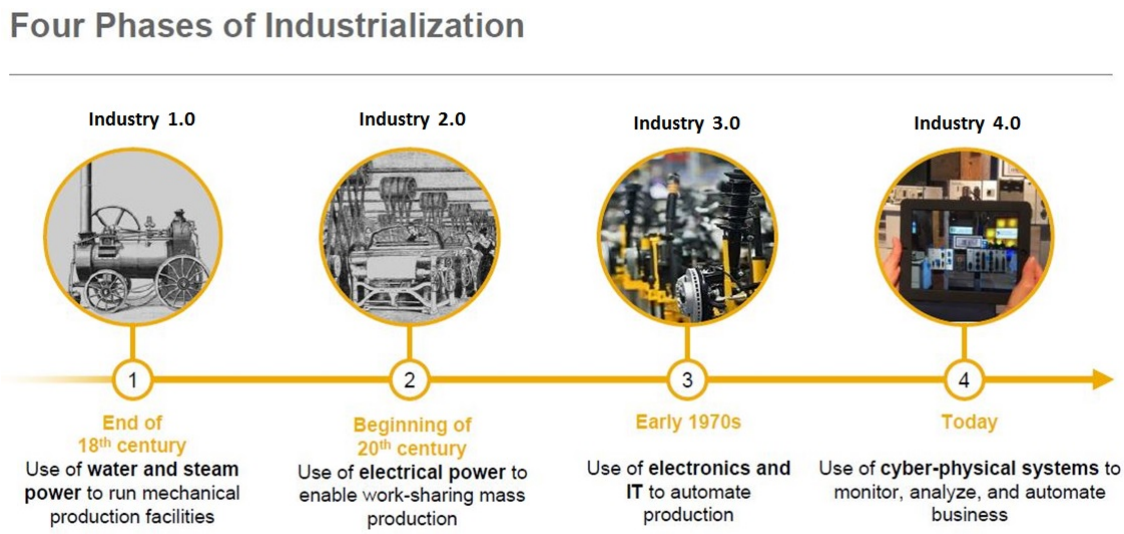



Figure 1.1: Industrial Revolution[14]

Industry 4.0 facilitates the perception of smart factories which are aimed to tailor the requirements of modern manufacturers. For this they use reconfiguring manufacturing systems

(RMS) which can be adapted quickly to produce new products and integrate new technologies. Today's manufacturing systems do not comprise of only hardware and machines but to a great extent information and communication technology(ICT) as well. A typical manufacturing system is comprised of several cyber physical systems (CPS), networked control systems (NCS) etc. Smart factories aim to connect all their devices, servers, intelligent systems, storage systems in a self organising way where they can trigger actions and control each other. Most of the this manufacturing system work on the periodic transmission of data, for instance, a set of sensors are sending data to a actuators at a constant sampling rate. These devices are often physically distributed across the shop floor and communicate with each other via communication network. Therefore, RMS must integrate a flexible communication infrastructure (ICT) as well. Software-defined Manufacturing systems could bring the essence of flexibility in the ICT infrastructure [24].

Most of these applications in industrial automation require high degree of quality of service (QoS) to function properly. They deal with time-sensitive production processes which require hard real time communication guarantees between several systems. Delay in data delivery can cause the entire process to fail. For example, applications in industrial automation requires jitter less than a 1 $\mu$s with latency between 250 $\mu$s - 1 ms [25]. Traditional bus systems like field buses meet this QoS requirement. Now, Internet of things (IoT) is being foreseen as the future for manufacturing systems. It requires connectivity to several new devices and new technologies with the existing ones. Therefore, communication networks should be highly flexible, scalable and interoperable. For example, nodes can be added to or removed from the network on the fly during operation. But the dedicated bus systems can not meet such expectations.

However, with success of Ethernet there has been a technology shift in the industry to replace the traditional bus-systems with the Ethernet in LAN topology. With Ethernet connectivity in the manufacturing devices scalability, flexibility and interoperability can be solved. The standardized IEEE 802.3 full-duplex switched Ethernet is a popular choice among industries and there is a strong demand to use Ethernet for time-sensitive network communication. As Ethernet was designed to provide "Best Effort" services, so to incorporate time-sensitivity in Ethernet based networks, additional features have to be implemented.

The network switches have finite buffer space where the packets or frames are put into a queue. This causes queuing delay and sometimes due to buffer overflow packets might be lost in the network. For instance, under heavy traffic load many devices want to transmit data to one switch, there will be non-deterministic queueing delay and buffer overflow. Both of these violate a deterministic data transmission. It is hard to eliminate queueing delay in network which has a major impact on total transmission time. IEEE Time-Sensitive Networking (TSN) Task Group is targeting [17] for time-synchronized low latency streaming services through IEEE 802 networks. IETF DetNets [7] are also working on time-sensitive communications, their work expands from Layer2 bridges to Layer3 routers as well. The current research work of the IEEE TSN task group and the IETF DetNet is prevalent in industrial automation as well. Both of these standardization bodies are aiming high time-sensitive communication. With the context of manufacturing environment described before, we can infer that it requires a time-triggered periodic communication between the devices like set of sensors, data acquisition

systems and actuators. Time-triggered communication works on the assumption that the sampling periods are constant and the control algorithm is executed with perfect periodicity.

In this thesis, we have incorporated the idea of "software-defined environments" with time-sensitive networks for reconfigurable manufacturing systems [24]. In particular, we focus on time-sensitive software-defined networking (SDN) to support real-time communications in manufacturing systems. As we already discussed that the foremost important requirement for a reconfigurable manufacturing systems is a flexible and scalable ICT infrastructure and dedicated systems like assembly line can not provide the required flexibility. Using software-defined environments (SDE) we can achieve this flexibility. Software-defined environment separates the control plane and data plane. For example, in a typical manufacturing system, control plane can handle storing, processing and analysing the data and data plane can be seen as creating the products, shifting and assembling them. By enabling SDE in manufacturing environment, all the logical computational part can be outsourced to the controller. By Implementing proper control logic algorithms (like for machine learning, detection of failure of any process) it can control the data plane. We have restricted ourselves in network resources, thus our work is based on Software defined Networking (SDN), a specific implementation of more generalised SDE.

## 1.1 Thesis Goals

To this end, we propose our work to time sensitive networks in the software defined networking(SDN) architecture (TSSDN). This work contributes in the following areas:

1. Firstly, we need suitable algorithms to calculate transmission schedules for end hosts in the network. These algorithms should allocate dedicated transmission slots to the hosts initiating a time-sensitive flow. The computed schedules should accommodate maximum number of flows and there should not be any queuing delay in network (by reserving the network links and the switches along the path of a flow for the allocated time-slot).

2. We developed our solution using a logically centralised SDN controller as basis. The main purpose of using SDN in this manufacturing environment is that, it separates control plane from the data plane, it has a global view of the network topology and a priori knowledge of the time-sensitive flows. The data plane is comprised of all the network elements whereas the control plane can compute the complex control logic like routing. We implement our scheduling algorithm as SDN's control logic. Our proposed algorithms not only computes paths for time-sensitive flows but also it calculates transmission schedules for them. Thus, it setup a flow with its path and the time when it has to be transmitted.

3. We compute schedules for unicast and multicast flows. The controller assigns time-slots to the end hosts and the end hosts transmit only on the allocated time-slots. In the

following sections, in detail we put forward our solutions using Integer Linear Programming(ILP) with some heuristics and also generic routing and scheduling algorithms to generate transmission schedules for the end hosts.

4. We also present our evaluation results and performance (quality and scalability) analysis of ILP based heuristics and generic scheduling and routing algorithms . We show that ILP based heuristics is able to compute near suboptimal solutions within few seconds for considerable size of network but solution time increases exponentially with the increases with the size of network. On the other hand, the generic algorithms compute the transmission schedules faster than ILP but it compromises on quality.

## 1.2 Thesis Organization

The remaining part of the thesis is organized as follows :

Chapter 2 gives background information for this thesis and summarizes the relevant research work.

Chapter 3 gives a detail description of the proposed system model for TSSDN and formal specification of the problem statement.

Chapter 4 is dedicated to design and implementation of scheduling algorithms for unicast communication. It includes all the ILP based heuristics to solve scheduling problem with unicast flows. We also present generic algorithms such as First fit and Random Fit to schedule unicast communication for time-sensitive flows.

In chapter 5, we present algorithms for multicast communication. It extends the design and implementation of unicast scheduling algorithms. This chapter describes different types of approaches to design a multicast group communication tree and formulation of ILP based on the tree generation. We also present Routing and Scheduling algorithms to schedule multicast communication for time-sensitive flows.

Chapter 6 provides an analysis of the implemented scheduling algorithms . It introduces the test environment used in this thesis. Also, it discusses the various experiments that were conducted and their evaluation results.

Finally, Chapter 7 concludes this thesis with a brief summary of the work done. It also proposes an outline for possible future works.

# Chapter 2

# Background and Related Work

This chapter provides background information for this thesis and summarizes related work.

## 2.1 Background

In the industrial applications like machine control, robotics, power generation, process control and transportation, require real-time communication to perform safely and securely. Timely delivery of data is a prerequisite to such systems. To support such kind of real time critical applications, different real time bus systems have been used over the decades in automation industries. Now, industrial systems are expanding out from small closed networks to an Industrial Internet of Things, where reliable and secure access to all network components is needed. Industrial control environment has more stringent requirements on reliable and real time communication. The application is considered to be failed if it does not execute a function within a confined deadline. To meet a very strict deadline, industrial applications need hard real time guarantees e.g. minimum delay in the communication between the devices, low latency and jitter in the network. Dedicated systems like field buses, fulfilled this requirements in manufacturing industries over the decades. We discuss in the following subsections about the traditional bus systems used for real time applications in different manufacturing systems.

### 2.1.1 Traditional Bus Systems in Real Time Communication

Within industrial communication systems, field buses are used for interconnection of process controllers, sensors and actuators, at the lower levels of the factory automation hierarchy. The origin of the field bus was to replace point-to-point communication link between the field devices (Field Devices are simply the Sensors and Actuators of the plant) and their controllers like PLC's by a digital single link. The main reason of favoring field buses over old point to point to communication is reduction of substantial amount of wiring and cost. They have been used in industrial real time distributed control systems and in real time communications over a decade. There are many fieldbus protocol standards such as CAN, ControlNet, DeviceNet, Foundation Fieldbus H1, HART, Modbus, and Profibus PA are the most commonly used field buses. Bus systems like CAN, Profinet, Profibus, FlexRay, TTCAN support timely delivery of data with bounded delay and latency. Most of these bus systems suffices the requirement of

real time communication in automation industrial applications. Let us have a brief overview on some of these industrial communication protocols.

#### 2.1.1.1 Control Area Network

The Controller Area Network (CAN) bus was designed to reduce the complexity of the industrial networking. In real time communication environment, the bus needs better real time performance i.e data must be delivered before a certain deadline. CAN bus provides guaranteed periodic data delivery for sending and receiving short real-time control messages at speed of up to 1Mbit/sec [6]. Though the basic CAN bus implementation doesn't provide a time-triggered scheduling mechanism, Time-Triggered Controller Area Network (TTCAN) is a time-triggered protocol that builds on the event triggered CAN protocol . It operates at the top of standard CAN hardware, provides a mechanism for scheduling messages. We can alternatively design our own schedule if the application is running on a closed network. CAN protocol handles very well the physical delay in the network, collision avoidance mechanism with CSMA/CD, fault tolerance and reliability of the network. But the inability of these CAN bus systems are the unbounded delay for the message delivery [40]. The longest time between queueing the message and the message arriving at the destination processors are non-deterministic. But for real time applications, we must provide a system with a bounded communication delay. Furthermore, CAN bus lacks flexibility and scalability, which is an essential requirement for today's industry standard.

#### 2.1.1.2 FlexRay

FlexRay is an automotive industry initiative to meet the need for real-time applications [11]. It is considered as next generation state-of-the-art for in car communication. FlexRay supports both time-triggered and event-triggered communications through the static segment and the dynamic segment respectively, which once concatenated form the communication cycle. It provides flexibility to use different bus topologies, for example: point-to-point connection, star networks, dual channel topologies, and some hybrid topologies. FlexRay uses two channels for transmitting information. Each channel is able to work independently, but can also be used to communicate the same information. Each channel should be able to carry information with a rate of 10 Mbits/s. Flexray is more flexible than the CAN bus due to its implementation of two channels which supports usage of different type of topologies. FlexRay have different time slots for messages that is scheduled by static cyclic and fixed priority [37]. Static cyclic scheduling has the advantage that it is easy to calculate response times and is very predictable. Using one static section and one dynamic section in FlexRay may be an advantage since frequent real-time messages can be sent in the static segment, while uncommon high priority messages and lower prioritized messages can use the dynamic segment. FlexRay is promising for automotive industries but for larger domain of industrial automation it does not have the scalability to connect huge number of network devices and it lacks interoperability between different devices.

#### 2.1.1.3 PROFIBUS(Process Field Bus)

PROFIBUS is a another implementation of Field Bus Protocol for industrial communication. PROFIBUS uses the token passing procedure.The master stations grant the bus access to each other, and a master-slave procedure used by master stations to communicate with slave stations. In this topology, a central line, or bus, is wired throughout the system. Devices are connected to this central bus. One bus eliminates the need for a full-length line to connect the central controller to each individual device [33]. It consists of a family of 3 protocol variations (DP, DPV1, DPV2) that can be used with different physical transmission media. The protocol is synchronous and operates in OSI layers 1, 2, 4, and 7. Despite of providing guaranteed data delivery on time, this bus systems also lack the support for flexibility. To add any new devices or nodes in the network, it requires new hardware support and it does not provide interoperability between the different devices.

In all of the bus systems discussed above, provide sufficient real-time guarantees for time-sensitive networks but they lacks scalability, interoperability between devices. Thus, we strive for new technologies to enhance those required qualities.

### 2.1.2 IEEE 802 Standards

To support time sensitivity in hard real time applications, data must be delivered before the deadline occurs otherwise the data is of no use. For example, applications in industrial automation require jitter less than 1$\mu$s with latency less (250 $\mu$ s - 1ms) [25]. Even though all of the above discussed network protocols provides hard real time gurantee but the major drawback of them is interoperability and scalability. All of theses protocols are good only for static network topology. Now a days with a higher network complexity, there is a definite need of communication between the network components and the network should be scalable. Ethernet has been the LAN of choice for business and enterprise for decades. By far it is the most successful and widely used networking technology in the world. It is affordable and reliable and is supported up by IEEE 802.3 standards [21]. It is gradually replacing the multiple field buses and proprietary networks or working with them.

## 2.2 Related Work

In this section, we discuss about the related work which are trying incorporate hard real-time guarantees in Ethernet.

### 2.2.1 FTT-Ethernet

Ethernet was not designed to support real time communications, it was meant for best effort QoS delivery. Hence to provide real time communication support it has gone through

different types of incarnations. FFT-Ethernet[29] aims to support for dynamic QoS management with timeliness guarantees as required by automation applications, e.g., industrial multimedia streaming. FTT (Flexible Time Triggered) paradigm has a logical centralized scheduling mechanism providing flexible real-time communication, supporting rapid dynamic changes with hard real-time guarantees. FTT-enabled Ethernet switch has a strict control on the traffic which comes into a switch [30]. It also implements traffic separation mechanism based on the arrival time and patterns of the incoming traffic. Therefore, other nodes that do not have real-time traffics can be used in the network that uses FTT-enabled Ethernet switch without any problem [36]. But FTT-Ethernet needs modified switching hardware to support their protocol which is not always a feasible option for a large industry based application. Our proposed TSSDN, does not require any modification on hardware switches as it is implemented on SDN.

### 2.2.2 TDMA Ethernet

Time Division Multiple Access(TDMA) based Ethernet[42] approaches a new domain by implementing TDMA based MAC layer approach in Ethernet Datacentre network. A centralized controller is used to compute and distribute a schedule that specifies an allocation of time-slots to each individual end hosts which want to initiate a flow. When a transmission slot is assigned to a source host, it can communicate with the destination host using all the resources along the path. It can use the full link bandwidth and switch buffers through the entire network diameter. As a result the a particular flow does not share any resources like link or switch buffer in a specific time-slots. This approach reduces the queueing delay in the network. To cope up with this generated transmission schedules, TDMA Ethernet implements Ethernet's IEEE 802.3x flow control protocol. It uses PAUSE frame handling mechanism to stop the transmission for a specific time period; when a receiver host detects there is a traffic overload in the network, it sends a link-local pause frame to the sender, with a configurable pause time payload. To provide synchronization, TDMA Ethernet measures the difference in arrival timestamps of a pair of control packets at different destination hosts. But this approach can not guarantee the synchronization of the end host systems as they do not consider tight time synchronization for real time systems. In our work, we have leveraged the idea of using PTP synchronization clock in all the end systems to adhere to the transmission schedules. By means of this our proposed system can achieve very precise time-synchronization.

### 2.2.3 IEEE 802.1 AVB and Time Sensitive Networking

The IEEE 802.1 Audio/Video Bridging Task Group has designed a set of IEEE standards that specify methods to offer the required quality of service (QoS) for audio/video (A/V) streams in a heterogeneous network [17]. These standards are fully deterministic to support real time communication over Ethernet. AVB defines the IEEE 802.1AS time synchronization protocol for the synchronization of distributed end systems in Ethernet. It provides a synchronization error of less than 1 micro-sec over a maximum of seven hops using time stamping [27]. IEEE 802.1Qav defines low latency for stream reservation(SR) Class A traffic with a maximum

latency of 2ms over 7 hops and SR Class B traffic with a maximum latency of 50 ms over 7 hops.

To explain time-sensitivity in the context of the networks AVB group highlights two significant requirements. First, data must be delivered within a very stringent time constraint, i.e before a maximum latency. Secondly the network devices should be precisely synchronised. Pro audio, for example, often uses a sample rate of 48,000 samples per second, and the precise synchronization of those samples for playback is critical for digital audio to work properly. Since the scope of this AVB TG expanded beyond audio and video streaming to incorporate time-sensitive, hard real time applications such as industrial automation,vehicular communication, the TG was retitled as Time-Sensitive Network (TSN) task group reflect the new, larger focus on Industrial Control. The aim of this group is to provide real time guarantee for time sensitive applications. Industrial control has similar needs for time-sensitive networks. For instance, a factory may have large machines spread out over a facility, which also need tight synchronization[16][2].

IEEE 802.1Qav handles transmission selection and traffic shaping using priorities and credit based shaping(CBS) algorithm. At each output port of MAC layer switch traffic shaping is implemented. When different types of T-spec traffic classes arrive at the ingress port of a switch, it is queued in the switch buffer. Credit is accumulated as the number of packets are queued and credit decreases as the packets are served. CBS delays the serving of the packet as much as possible in order to reduce bursting in network. Stream Reservation Protocol(SRP) along with Credit Based Shaper (CBS) intends to provide low latency for AVB traffic. The shaper gains a credit when a packet is queued in the switch buffer. The transmission of an AVB packet can only be possible when the credit is non-negative. TSN TG address synchronization issue with IEEE 1588 [also known as the Precision Time Protocol (PTP)], which standardized the use of physical layer timestamps to compute network delays and define synchronization events. This concept influence our approach to use PTP clock on the end hosts to synchronise them precisely. They have not come up yet with the optimal solution for computing transmission schedules on the end hosts and which might be applicable for upcoming IEEE standard networks. We are motivated by their approach of using time-triggered communication for real-time communication in LAN. We have designed and implemented scheduling algorithms to schedule time-sensitive flows for deterministic networks.

### 2.2.4 Deterministic Ethernet and IETF Deterministic Networking Architecture(DetNets)

Deterministic Ethernet[9] refers to a standard networked communication technology that incorporates scheduling mechanisms to provide real-time guarantees in Ethernet. Deterministic Ethernet operates using a global transmission schedules which is shared between network components.

A deterministic system is considered to be a predictable, producing a predictable and consistent response time between two end devices [41]. Ethernet Determinism is defined by two key components: System Determinism and Network Determinism. System Determinism can

be explained as application response time or the time it takes to receive the input in an I/O device, process and write the output on the I/O device. Network Determinism refers to the ability of the network to consistently transmit data between two end hosts in the network in a guaranteed time frame.

By using Deterministic Ethernet we can handle both real-time controls traffic and regular best effort traffic on one Ethernet network simultaneously. Time-scheduled traffic is separated from all other background network traffic and is therefore immune from interruptions. This ensures guaranteed bounded latency of critical scheduled communication in a Deterministic Ethernet network. This is called Guarantee of Service. Deterministic Ethernet is used in applications where guaranteed latency is a must to function correctly. For instance, applications in autonomous driving, machine-to-machine communication and aerospace flight control.

The Deterministic Networking (DetNet) Working Group[**Deterministic Networking**] is a standardization body for deterministic Ethernet, there are working on time-sensitive communication over MAC and IP layers. We discussed how the IEEE 802.1 AVB Task Group expanded to Time-Sensitive Networking (TSN) Task Group (TG) to scatter the needs of more deterministic networks from the automation and vehicular industry. DetNet focuses on deterministic data paths that can operate over Layer 2 bridged and Layer 3 routed segments. These paths will provide bounds network delay, loss, jitter and high reliability. For example, a control systems application may expand form LAN to WAN then the path may span for instance, across a (limited) number of layer 2 802.1 bridges and then a (limited) number of layer 3 IP routers. Basically they are extending the Deterministic Ethernet queuing mechanisms to L3.

DetNet Architecture comprises of three layers: a (User) Application Plane, a Controller Plane, and a Network Plane which is similar to the Software Defined Networking (SDN) architecture. It implements a centralized SDN controller to compute the Deterministic paths which will be assigned to the Network Plane. Our approach is also in-lined to DetNet Working group. We too have a centralized SDN controller to compute paths for the data plane devices, but along with the paths we also compute transmission schedule on the end hosts systems. DetNet has not yet come up with a robust algorithm which will compute path and will schedule the data flows in the network plane in such a way that it bounds the delay and jitter. We contribute our work directly on this area by designing and implementing optimized algorithms for routing and scheduling.

## 2.3 Datacenter Networks

The unprecedented growth in industrial automation has increased the traffic load on communication networks. New types of traffic patterns and workload have to be handled by datacentre networks these days. To support the increasing demands on network, datacentre infrastructure in terms of responsiveness, flexibility and ability to roll out cutting-edge applications requires enhanced performance. A lot of research work is going on to increase the performance of datacentre network to support the workload and provide quality of service

with high throughput and low latency in associated networks. However, it is to be noted that datacentre networks deals with soft real time applications where the traffic patterns differ largely from hard real time applications. An example for soft real-time application, a video streaming on youtube. If we miss some bits or it is buffered for few seconds, still the application is not considered to be failed. But if there are too many missing bits means data is not delivered or it is delayed, the performance of the service degrades with time. However, due to huge data load in network, tackling end-to-end latency is a big concern for datacenter as well. We discuss about two prominent approaches to deal with this regarding datacentre networks.

### 2.3.1 Hedera

Today's Data-centre networks experience enormous load from different real-time applications. There are few factors which make it difficult to manage huge workload on these data-centre. First the data-centre networks workload varies in time and space, so it needs dynamic resource allocation. Secondly to run different applications it needs high bandwidth without the need of changing any protocols and lastly virtualization of the network to distribute the customer load on different physical machines. Hedera[1] gets the flow information from the switches, computes non-overlapping routing paths and routes the flow accordingly. In data centre applications, there have been use of static Equal Cost Multiple Path (ECMP,RFC 2992) routing which maps the flows to the specific paths. This static mapping results in multiple flow hashing in the same path which leads to collision in the networks and under utilization of bandwidths. Though Hedera balances the network load by re-routing the flows, but it only considers the "elephant" flows. But the load generated by the small flows left still unbalanced. Accumulation of all these small flows can result in substantial unbalanced load in the network. Moreover, Hedera does not implement scheduling of transmission slots at the end hosts, this can results in multiple flow transmissions from one host at a time. Therefore, it can not not provide any guarantee for bounded delay in data transmission. In our approach, we compute routing paths and transmission slots for the end hosts so that they do not transmit multiple flow on the same time-slots. This guarantees that there is no queuing delay for a flow.

### 2.3.2 Fastpass: A Centralized "Zero-Queue" Datacenter Network

Fastpass[31] data centre architecture uses a centralized arbiter to decide on the flow path and they allocate time-slots to the source host for transmitting the flows. In this approach avoiding queueing delay has been tackled so that time-critical flows do not have to suffer from the huge traffic in other part of the network. Fastpass implemented the idea of TDMA Ethernet with datacenter networks. In this approach they have mainly worked on two basic ideas:

1. A path selection algorithm at the arbiter to allocate path efficiently to transmit packets and to avoid collision. The path allocation algorithm routes the packets in such a way that no two packets ever traverse through colliding paths; i.e which have any common links between them get a same slot. These guarantees no collision on links. If resources

are reserved for particular time-slots to transmit one packet then there is no queueing delay in the network.

2. Time Slot Allocation algorithm at the arbiter which is basically a scheduling algorithm to decide when to send a packet. The algorithm finds out a source-destination pair which can transmit a packet to the end hosts without violating the link capacity constraint. The central arbiter has a global view of all the network traffic demands which makes it faster and efficient to compute schedule for the traffic data flow. To match the arbiter processing rate with the end host link speed, Fastpass allocates multiple time-slots simultaneously. As a result of this when the network workload is less, the time-slots are obtained quite fast but with the heavy workload time-slots allocation takes much longer time.

   We differ from Fastpass approach in our proposed solutions. In datacentre the traffic is not predictable. So, in Fastpass every time a new flow comes and even if there are no time-slots available, still it has be sent to the central arbiter to ask for a time-slot. Thus it has a round trip delay for the flow in the network. Datacenter networks can afford this delay as they do not have to necessarily provide hard real time guarantees. But we are targeting time-sensitive networks with requirement of very strict real-time guarantees. In such case, round trip delay to get a slot will surely result in deadline miss. In our approach, the time-sensitive data flows are known a priori and the traffic is periodic in nature. Hence, the end hosts do not have to ask for the transmission schedule every time. Once a transmission schedule is allocated to an end host, it will transmit on that time-slot in every cycle. Therefore, there is no such round trip delay in our proposed scheduling approach.

# Chapter 3

# System Model and Problem Statement

This chapter gives a detailed description of the network architecture, its components and the problem statement with respect to the scheduling of transmission.

## 3.1 System Model

The system model consists of a centralised logical SDN controller which separates the control and the data plane. In SDN controller's control logic application, we compute routing paths and generate global transmission schedules to allocate time-slots to the time sensitive data flows. To adhere to the generated global transmission schedules all the network components have to be precisely synchronised. Precision time protocol (PTP) may be used to maintain time synchronisation of the network elements. Furthermore, the end systems must be able to put the packets on the network as soon as the transmission time begins. This can be achieved using different frameworks for fast access to the network devices.

### 3.1.1 Network Components

TSSDN is comprised of three different types of network components: the network controller, network switches and the end hosts. The end hosts are the source and destination for a traffic data flow in any network, it can execute processes or applications. For instance, with respect to cyber-physical systems (CPS) in manufacturing which consists of sensors, actuators and CPS-controllers, any of these components could be seen as an end - host .

Network switches are the Layer 2 or Layer 3 forwarding switches which forward incoming and outgoing data frames in the network. In contrast to the conventional network switches where packet forwarding and routing takes place on the same device, software defined network separates the data and control functions of networking devices, such as routers, packet switches, and LAN switches, with a well-defined Application Programming Interface (API) between the two. We assume that, all the switches support standard OpenFlow protocol. These switches and the network controller communicate by means of the standard OpenFlow protocol implemented by southbound interfaces. We also assume that all time-sensitive network the traffic

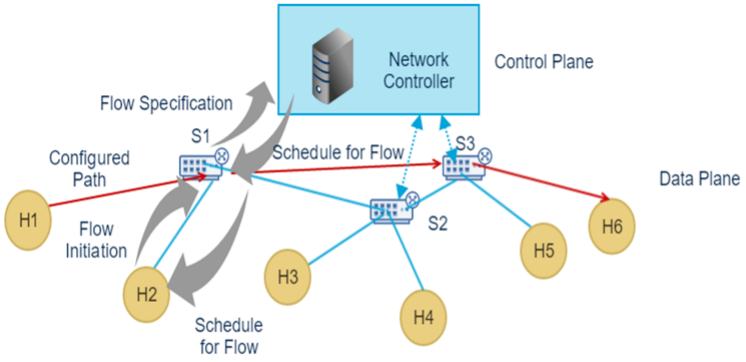



Figure 3.1: Network Components

has the same priority. It can be implemented using IEEE 802.1Q priority classes to set the highest priority for time-sensitive traffic and other non time-sensitive as lower priority. As all the time-sensitive traffic is of same priority, to avoid conflict in the network we must use additional scheduling mechanism.

The network controller is responsible for taking the forwarding decision for all the switches globally. We also assume that the centralised controller has a global view of the entire network and has a prior knowledge about the data flows that occur in the SDN data plane. We define a data flow as a sequence of packets from a source to destination. When first time a host wants to set up flow between another host or set of hosts (in case of multicast), it sends a request to the controller with source IP, destinations IP, source port, destination port and transport layer protocol. In response, controller computes the route for the flow and set up the flow tables in each of the switches along the path. This flows are periodic in nature and this periods are repeating in cycle. The controller can also be implemented as distributed controllers for more resilient architecture, which ensures no single point of failure, increases responsiveness, reliability and scalability. But this is beyond the scope of this thesis. Since the scope of this thesis is limited to IEEE 802 local area network, principally wired Ethernet, the network controller can discover the network topology with the Link Layer Discovery Protocol(LLDP, IEEE 802.1AB). In our case, the controller not only computes the path for a flow but also computes the transmission schedule for the source hosts.

In a time triggered communication, in control systems input data from sensor devices are sent to the controller at a periodic rate. The controller runs its control algorithm at a periodic rate and output results are sent to the output actuators at a periodic rate. We have leveraged the idea of periodic transmission assuming that all the hosts send packets on a fixed interval of time. We assume that the time-sensitive flows are of constant bit rate pattern, where the transmission of packets are periodic in nature. This time interval is an integral multiple of a “base period”. The transmission schedule for the end hosts is a repeating cycle with the interval of base period as shown in Figure 3.2. Each base period is again subdivided in

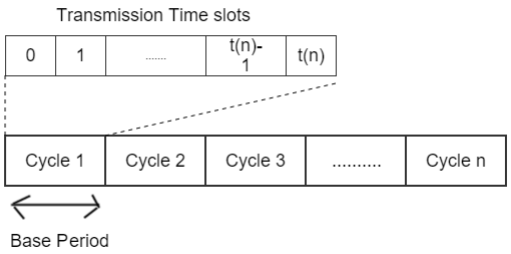



Figure 3.2: Time Triggered Time Division Multiple Access

multiple time-slots. The length of these time-slots are the maximum time a MTU (maximum transmission unit) sized packet can take to traverse the network diameter.

We consider UDP transport service for transmission of time sensitive flows. UDP is faster than TCP and it does not have any connection overhead. One flow can be short, real-time (that is, time-sensitive) where packets to be transmitted to a network peer within a certain amount of time (the "delay budget"). The packet fragments are small enough (MTU sized), so transmission time does not exceed the time required for transmitting the real-time (time-sensitive) data packet. The controller checks for the source and destination of the data flow and generates a transmission schedule for the host and sends it to the source host along with routing path. For the communication between the controller and network hosts standard OpenFlow protocol or any other signalling protocols [20]. There are different types network delay a packet experiences while traversing through the network. It is the time required to transmit a packet along its entire network diameter [12]. The total round trip time (RTT) comprised of the following:

Propagation Delay: The time required for a packet to traverse from one end to the other end of the network. For example, length of a LAN segment is l and speed of the Ethernet is c. So the propagation delay is pd = l/c. In our case we restrict our network diameter to 6 hops, so propagation delay is constant.

Transmission Delay: The time required to transmit all the bits of a packet in the network. It is dependent on the size of the packets. For instance, length of the packet is L bits and transmission speed is r bit/sec. So the transmission delay is td=L/r [23]. As we have assumed that our packets are of constant size (constant bit rate-pattern) and resources are reserved for a specific time slot, which means that transmission delay is deterministic.

Processing Delay: Time required to analyse the packet header and retrieve the routing information. In our case processing delay are almost negligible and they are deterministic in nature.

Queueing Delay: It is referred to the time when a packet is in the queue and waiting for transmission. Queueing delay occurs when multiple hosts try to send packets to one ingress port of a switch (for example in Figure: 3.3) and it depends on amount of traffic arriving at the switch.

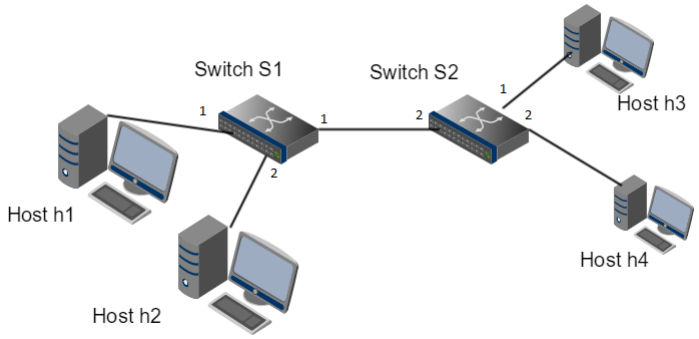



Figure 3.3: Queuing Example

So it is the queueing delay which is non-deterministic [8]. To avoid packet queueing delay and buffering time on the switch, we implement Time Division Multiple Access (TDMA) to allocate time-slots assigned by the network controller. Every time-sensitive flows has dedicated time-slots assigned by the controller. During this time period the source can transmit a flow and all the links are reserved for that flow. Therefore, there are no other packet in the path for that packet which eliminates the queueing delay completely from network.

### 3.1.2 Time Synchronisation on Networks

The end hosts are needed to be precisely time synchronised to maintain the transmission schedules. The IEEE TSN task group influences the idea of using a synchronised clock in typically Ethernet packet based networks. Clock synchronization is a vital mechanism for achieving deterministic communication with bounded message latency in TSN. The IEEE 1588 standard Precision Time Protocol (PTP) provides a solution that can be easily adopted by the industrial control industry to provide precision time for the time-based control on the factory floor. It uses physical layer timestamps to compute network delays . The accuracy of PTP synchronised clock is in the order of 100 ns [15].

### 3.1.3 Framework for fast access to the network devices

The increasing speed on the network enhances the requirement of using high performance network application devices applications such as software switches, routers, firewalls, traffic monitors and generators etc.

We generate schedules and use high speed Ethernet to transmit the packets from one end to the other end of the network. The end hosts should adhere to the generated transmission schedules and put the packets on the network links as soon as possible. The socket API offered by the operating systems are not precise enough to comply to the transmission schedule and that introduce non-deterministic delay in network. There are different frameworks such as DPDK, Netmap which enable us to eliminate special-purpose hardware such as network

processors (NPUs), co-processors, application specific integrated circuits (ASICs), and field-programmable gate arrays (FPGAs). With the implementation of fast packet processing framework we can eliminate the potential jitter which are caused by network stack.

Netmap: It is a novel framework that enables normal operating systems to handle the millions of packets per seconds traversing with high speed, without changing any hardware[34]. Netmap provides user processes access to NIC hardware with a high-performance userspace packet processing speed. The device drivers remain in the host kernel, but a lower overhead access to hardware has been offered to userspace applications. Instead of reducing the latency and jitter between the sending process and network interface, it bypasses the network stack of operating systems. In addition to the memory-mapping of buffers, it has other performance optimization methods such as batch processing and buffer reallocation, and can easily saturate a 10 Gbps link with minimum-size frames.

DPDK: Like Netmap, DPDK is also meant for fast network processing and multiprocessor-aware networking. DPDK offers network access by attaching to hardware and providing a hardware-independent API for sending and receiving packets [38][28]. It reduces the packet processing time by eliminating sharing buffers and metadata between kernel and userspace.

As a prerequisite to this thesis, this framework was already implemented on our hardware setup. Out of all of the implemented frameworks, DPDK enhances the network performance immensely. With DPDK it was possible to exploit the link capacity on both 1 Gbit and 10 Gbit on the network adapters.

By the virtue of this network performance, we can ensure that the end hosts can cope with the schedules, which is a critical requirement for this thesis.

## 3.2 Problem Statement

The main objective of this thesis is to design and implement efficient scheduling algorithms for unicast and multicast communication to compute transmission schedules to support time-sensitive networking in real-time applications. Our approach is to schedule the flows so that there is no queueing, the source host will only transmit when the resources are reserved along the entire path of the flow. Now, we develop our algorithms to compute the routing paths for the time-sensitive flows, generate the optimized transmission schedules for them. It should also maximize the number of allocation of flows over the network, so that demands of large number of the real-time applications can be served.
Let us look at an example of flow allocation with a naive approach and a good approach. In figure 3.4 shows a small topology with 12 hosts (H11, H12, ..,H34) and 4 switches (S1, L1, L2, L3). The example in table (3.1), we consider flow as a pair of source and destination hosts, number of packets are the demand for a flow and slot allocation is the time-slots the flow can be transmitted. We assign 5 slots on each link and 5 set of flows (in col 1) have to transmitted with respective number of packets. For each flow we compute the path first and assign the time-slot. We can see that in a naive approach, we can allocate four flows even though there are free slots in the network. The flows can not be allocated, whereas with a good approach

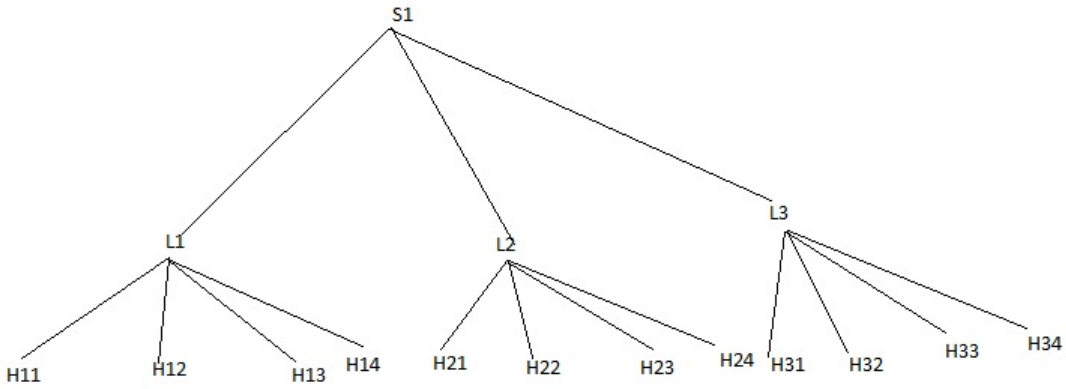



Figure 3.4: Example of a simple network topology

| Flows to be scheduled | No of Packets | Slot Allocation(Naive) | Slot Allocation(Good) | Path |
|---|---|---|---|---|
| (H11,H21) | (2) | (1,2) | (1,2) | (H11 - L1 - S1 -L2 - H21) |
| (H31,H22) | (2) | (3,4) | (3,4) | (H31 - L3 -S1 -L2- H22) |
| (H21,H32) | (2) | (1,2) | (1,2) | (H21 -L2 -S1 -L3 - H32) |
| (H22,H33) | (2) | (3,4) | (4,5) | (H22 - L2 - S1 -L3 - H33) |
| (H14,H24) | (1) | ? | (5) | (H14 - L1 - S2 - L2 - H24) |

Table 3.1: Time-slot allocation

we can allocate all the five flows. For a larger topology and large number of flows, we need an algorithmic approach to generate schedules for time sensitive flow.

### 3.2.1 NP Optimization Problem

We discussed in the previous section that our problem is to generate transmission schedules for the time-sensitive flows for both unicast and multicast communication. Furthermore, we address the time-sensitive flows where it can have single demand which means it can be transmitted over a single packet or the flows having multiple demands which has to be spread over multiple packets per cycle. In the formulation of all our problems either we want to maximize the allocation of flows or want to maximize the demands. Hence, all of the problems are maximization problems. This category of problem can be classified as optimization problem in computer networks. An optimization problem can be defined as finding an optimum solution from a set of all feasible solutions. It can be theoretically formulated as follows [3] :

Input: a function f :$A \rightarrow R$ from some set A to the real numbers
Output: an element x0 $\in A$ such that $f(x0) \leq f(x) \forall x \in A$("minimization") or such that $f(x0) \geq f(x) \forall x \in A$("$maximization$").

To compute the schedule, first we approach the problem with Integer Linear Programming(ILP) with different heuristics. The scheduling problem is a typical maximization NP hard problem with different constraints. It can be mapped from the famous wavelength continuity problem in optical networks. It is known as Wavelength Routing Assignment problem(RWA)[44]. Similar to the RWA problem, we also divide our solution in two phases, namely routing and scheduling.

We approached the scheduling problem in two different ways. We implemented different heuristics with Integer Linear Programming (ILP) and also with routing and scheduling algorithms. The approach with ILP we discuss here.
First we find out the paths for the requested flows. We use shortest path algorithm to assign the paths to flows. These paths are static and are used as an input to the Integer Linear Programming.

Once the routing is completed for all the flows, the controller generates the transmission schedules using ILPs. ILP computes the optimal solution to assign time-slots to all the paths, so that maximum number of flows can transmit within one cycle.

For the Multicast scheduling problem also we have formulated ILP. Similar to unicast, multicast has also been developed on the concept of static traffic routing. In optical network multicast routing and wavelength assignment problems are generally solved by creating light-trees [35]. A light-tree is typically a Steiner tree which originates at the source node of a multicast session and connects all the destinations nodes and the tree splits at the intermediate nodes where the paths diverse. This problem is termed as MCAR problem in [4]. Where the problem has been defined as follows:
Given a network $G = (V, E);\ \forall V \in \{1, 2, ..NumVertices\},\ \forall E \in \{1, 2, ..NumEdges\}$
A set of multicast advance reservation requests $\Delta = \{R_1, R_2, ...R_n\}$
The objective of the problem is to provide at-most one wavelength to the multicast light-trees without violating the wavelength collision on the links. And this problem is proved to be NP hard. In our approach also we built multicast tree to connect the source host and all the receiver hosts of a multicast request and assign time-slots to the trees and implement constraints on the links to avoid collisions. Hence, infer that we can map our multicast single demand problem to an existing NP hard optimization problem. We have designed and implemented the scheduling algorithms using different heuristics with Integer Linear Programming, furthermore we also extended the existing scheduling algorithms and implemented them with different time-slots assignment mechanisms for Unicast and Multicast Scheduling.

# Chapter 4

# Scheduling Unicast Flows

In this chapter we discuss about the algorithms to solve the scheduling problem for unicast flows in time-sensitive applications. We propose two different implementations. Firstly, we present Integer Linear Programming(ILP) to compute the optimal solution using some heuristics. Secondly, we also present routing and scheduling algorithms to address the scheduling problem.

## 4.1 Terminologies and Notations

We have considered the network topology as a directed graph. We denote the directed graph as $G \equiv \{V, E\}$; where V is the set of network nodes, E is set of all the edges in the network which connects the nodes in the graph. $E \equiv \{i, j\};\ where\ i,\ j \in V$; where i, j are connected via a network link. We represent network links L as L $\equiv e_l$ where l $\in \{1, 2, ...E\}$.

We define the time-sensitive network data flow with three tuple. The flow is denoted as, $fs_i \equiv \{s_i, d_i, p_i\},\ where\ s_i\ and\ d_i$ belongs to the end hosts for the time-sensitive flows. $s_i\ and\ d_i$ denotes the source and destination of the time-sensitive flows respectively and $p_i$ indicates the flow period. The period repeats cyclically and it is divided in smaller time-slots as explained in chapter 3 subsection 3.1.1. These time-slots are available on the network links and they are denoted by S, where S$\equiv t_k;\ \ k \in \{1, 2, 3, ..S\}$.

## 4.2 Overview of scheduling approaches

We have two different approaches to solve the scheduling problems. They are: Integer Linear Programming (ILP) and non ILP based routing and scheduling algorithms. Figure 4.1 shows the different types of scheduling approaches we have developed in the following sections. Depending on the routing of the flows, we have two different heuristics, namely fixedpath and path-sets. Moreover, We have considered that the flows can be transmitted over one single packet or multiple packets. We present different formulations to handle single and multiple packets, a flow which is transmitted over one single packet named as single slot or single demand and the flows which would be transmitted over multiple packets named as multiple slots or multiple demands.

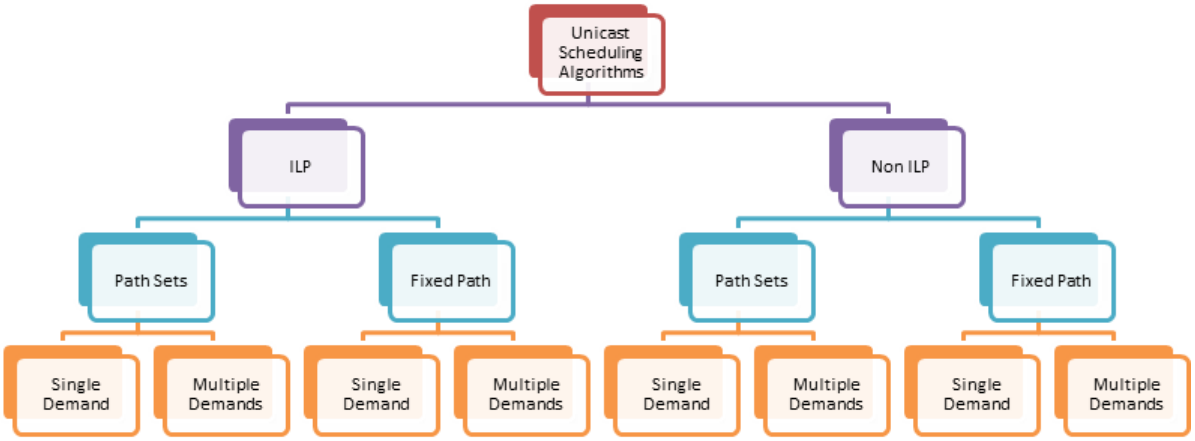



Figure 4.1: Classification of Scheduling approaches for Unicast Flows

## 4.3 Scheduling with ILP based heuristics

As shown in the figure 4.1, we have different types of ILP based heuristics to address the unicast flow scheduling problem. It is be noted that, for ILP based approaches, we have considered multi-flow scheduling problem. Here the set of flows to scheduled is known a priori and static. The multi-flow scheduling problem deals with maximization of number of flows that can be scheduled on the network from the total set of given flows. The following section contains the detailed formulation for all the implemented ILP based scheduling approaches.

### 4.3.1 Unicast - PathSets Single Slot (UIPS)

In order to route the time-sensitive flows, we have used multi-path routing to discover multiple candidate paths between a source and destination hosts. Routing over the shortest paths not only reduces the time needed to transmit the packets but also reduces the resource usage as it uses less number of links to traverse. In terms of ILP, giving a fixed set of paths to the flows lessens the search space over the whole available paths which leads to lesser computing time. In our first heuristic approach, the flows are allowed to take any one of its candidate paths and we are taking into consideration that all the flows will be transmitted using one single time-slots.

**ILP Inputs**: The required inputs for the formulation of ILP are as follows:

1. Set of time-sensitive flows which has to scheduled, DF.
   $DF \equiv df_i;\ i \in \{1, 2, ...No\ of Flows\}$

2. Set of all possible shortest paths through which the flows can traverse, P:
   $P \equiv p_j;\ j \in \{1, 2, 3..., No\ of Paths\}$

3. Mapping of flows to the paths, DP:
   $DP \equiv fp_{i,j};\ \forall i \in DF, \forall j \in P.\ fp_{i,j} = 1,$ if flow i is routed over the path j.
   One flow can have multiple candidate paths, one path will be used for routing of one flow, i.e., the candidate paths identifies the respective flow.

4. Mapping of paths and links, PL.
   $PL \equiv pl_{j,l};\ \forall j \in P,\ \forall l \in L.\ pl_{j,l} = 1$, if path j has link l, else 0.

**ILP variables:** We assign time-slots to a path. As every path is used for the routing of a flow, so path indirectly indicates a flow here.
$PS \equiv ps_{j,k};\ \forall j \in DP, k \in S.\ s_{j,k} = 1$, if path j is allocated to time-slot k, else 0.

**Objective Function:** The objective function of this ILP is to allocate time-slots to maximum number of paths.

$$\text{Maximize} \sum_{\forall k \in S} \sum_{\forall j \in DP} s_{j,k}$$

**Constraints:** This ILP is subjected to the following constraints:

1. Path Constraint:
   Each Path can be allocated at-most one slot.
   $$\sum_{\forall k \in S} s_{j,k} \leq 1;\ \forall j \in DP$$
2. Link Constraint: There should not be any collision on the links. Any two paths having overlapping links will not be assigned the same time-slot.
   $$\sum_{\forall j \in DP} s_{j,k} \times p_{j,l};\ \leq 1;\ \forall k \in S, \forall l \in DL$$
3. Flow Constraint:
   Every flow should be allocated to one time-slot as in this formulation we have assumed that they carry one MTU sized packet in one period.
   $$\sum_{\forall j \in DP} s_{j,k} \times df_{i,j} \leq 1;\ \forall i \in DF, \forall k \in S$$

### 4.3.2 Unicast Fixed Paths Single Slot (UIFS)

In fixed path heuristic, the path over which a flow is routed is chosen randomly from the set of all shortest candidate paths between the source and destination, it is given as an input to the ILP. Since every flow has only one path to be routed over, so the ILP deals with assigning time-slot to that path. Hence this particular formulation has one degree of freedom that is time-slot. This approach enhances the computation speed, however this solution can not provide the optimal solution as we are restricting the path for a flow.

**ILP Inputs :** The inputs for this ILP formulation are as follows:

1. Set of data flow which has to scheduled, DF.
   $DF \equiv df_i;\ \ i \in \{1, 2, ...No\ OfFlows\}$

2. Set of all the links i.e. network edges of the graph G
   $DL \equiv dl_j;\ \ j \in \{1, 2, ...E\}$

3. Mapping between the flows and the links, FL. It maps a flow to all the links that it uses to be routed over the randomly chosen path. $FL \equiv fl_{i,j};\quad \forall i \in DF, \forall j \in E :$
   $fl_{i,j} = 1, if\ a\ flow\ i\ traverses\ through\ the\ link\ j,\ else\ 0.$

**ILP Variables:** In this formulation only one decision variable is required to map a flow to the time-slots FS. FS specifies which time-slot is allocated to which flow.
$FS \equiv s_{i,k};\ \ \forall i \in DF, k \in S :$
$s_{i,k} = 1,\ if\ a\ flow\ is\ using\ the\ link\ k,\ else\ 0.$

**Objective Function:** The objective function is to maximise the allocation of time-slots to the flows. The formulation is as follow :

$$Maximize \sum_{\forall k \in S} \sum_{\forall i \in DF} s_{i,k}$$

**Constraints:** This ILP is subjected to following constraints:

1. Flow Constraint: Every Flow gets at-most one time-slot allocated to it, as in this formulation we have assumed one flow can be transmitted using one MTU sized packet.

   $$\sum_{\forall k \in S} s_{i,k} \leq 1 \quad \forall i \in DF$$

2. Link Constraint: No two flow can be allocated to same time-slot, if they have common links.

$$\sum_{\forall i \in DF} s_{i,k} \times fl_{i,j} <= 1; \quad \forall k \in S; \forall j \in DL$$

### 4.3.3 Unicast - PathSets with Multiple Slots (UIPM)

This ILP formulation is based on the following assumptions:

1. The flows can be routed over multiple available candidate paths.
2. Here we address those flows where data has to be transmitted over multiple packets, as flow data exceeds the size of MTU. In this ILP, one flow can get multiple time-slots, number of allocated time-slots will be number of packets are required to be transmitted for that flow.

**ILP Inputs :** The inputs for this ILP formulation are the time-sensitive flows which have be scheduled and the set of candidate paths on which flows can be routed.

1. Set of flows which has be scheduled, DS.
   $DS \equiv ds_j; \quad j \in \{1, 2, 3, ....No\ of\ flows\}$

2. Set of candidate paths for all the flows, DP.
   $DP \equiv dp_i; \quad i \in \{1, 2, 3, ...No\ of\ paths\}$
   Here No of paths means all the paths exist in the network.
3. Set of all the demands for the set of all the flows which have to be scheduled, D.
   $D \equiv d_n;\ n \in \{1, 2, ...No\ of\ demands\}$
4. The links in the network, DL.
   $DL \equiv dL_l;\ \forall l \in \{1, 2, ...E\}$
   Here E represents all the links in the graph G.
5. Mapping of the flows and the paths, FP.
   $FP \equiv f_{i,j};\ \forall i \in DP,\ \forall j \in DS:$
   $f_{ij} = 1$, if flow j traverses through the path i, else 0.
6. Mapping of paths and links, PL.
   $PL \equiv pl_{i,l};\ \forall i \in DP, \forall j \in DS:$
   $p_{i,l} = 1,\ if\ path\ i\ uses\ the\ link\ l,\ else\ 0.$

**ILP Variables:**

1. In this formulation we assign time-slots to the paths, each path corresponds to a flow. Since one flow can consists of multiple packets, hence one path can have multiple time-slots so that one flow is routed over only one path with multiple time-slots or the packets can be routed using different candidate paths with different time-slots.
   $PS \equiv s_{i,k};\ \forall i \in DP,\ \forall k \in S:$
   $ps_{i,k} = 1,\ if\ path\ i\ gets\ allocated\ time - slot\ k,\ else\ 0.$

2. We have used a set of binary decision variables which are mapped to the set of demands, D:
   $M \equiv d_n;\ \forall n \in \{1, 2, ..D\}$

**Objective Function:** The objective function for this ILP can be formulated to satisfy two different goals. They are namely Maximize demands, Maximize flows.

1. Maximize Demand $\sum_{\forall q \in M} \sum_{\forall q \in D} m_q \times d_q$
2. Maximize Flow $\sum_{\forall q \in M} m_q$

**Constraints:** The constraints for this ILP are described below :

1. Path Constraints: Every path can have maximum number of time-slots which is equal to the number of demands for the corresponding flow.

$$\sum_{\forall k \in s} s_{i,k} = m_q \times d_q;\ \forall q \in M,\ \forall q \in D,\ \forall k \in S$$

2. Link Constraint : Any two paths having common links between them can not be allocated same time-slot:

$$\sum_{\forall i \in DP} s_{i,k} \times p_{i,l} \leq 1;\ \forall k \in S, \forall l \in DL$$

3. Flow Constraint: All the flows should get number of time-slots which is equal to the number of demands it has or number of packets is transmit the flow.

$$\sum_{\forall i \in DP} s_{i,k} \times f_{i,j}\ = m_q \times d_n;\ \ \forall q \in M, \forall n \in D, \forall k \in S$$

### 4.3.4 Unicast - Fixed Path Multiple Slots (UIFM)

Fixed Path heuristics is an extended approach of Path Sets heuristics. Instead of giving the ILP to choose from all the set of shortest paths, we choose one shortest path from the available equidistant candidate paths like ECMP and feed that path as in input to the ILP. Thus the ILP has to only assign time-slots to the static paths. Here, we have considered that each of the flow can be transmitted as multiple units of packets, therefore each flow has multiple demands.
The inputs to the ILP formulation as as follows:
**ILP Inputs**

1. Set of unicast flows which have be scheduled, DF,
   $DF \equiv df_i;\ \forall i \in \{1, 2, ..SetofFlows\}$

2. The links in the network, DL.
   $DL \equiv dL_j;\ \forall j \in \{1, 2, ...E\}$
   Here E represents all the links in the graph G.

3. Mapping between the links and the flows. FL.
   $FL \equiv fl_{ij};\ \forall i \in DF, \forall l \in DL:$
   $dl_{i,j} = 1$, if flow j uses the link l to traverse from source to destination else 0.

4. Set of all the demands for the set of all the flows which have to be scheduled, D.
   $D \equiv d_n;\ \in \{1, 2, ...No\ of\ demands\}$

**ILP Variables** The required decision variables for this ILP formulation are as follows:

1. We assign time-slots to each flow. Decision variables are required to map between the flows and the time-slots, FS.
   $FS \equiv s_{i,k};\ \forall i \in DF,\ \forall k \in S.$
   $s_{i,k} = 1,$ if flow i is allocated time-slot k, else 0.

2. Another binary decision variable is required check whether a demands of a flow is satisfied or not, M.
   $M \equiv m_q;\ \forall q \in D.$

**Objective Function :** The objective function for this ILP could be formulated in two ways.

1. Maximize Demand $\sum\limits_{\forall q \in M}\ \sum\limits_{\forall q \in D} m_q \times d_q$
2. Maximize Flow $\sum\limits_{\forall q \in M} m_q$

**ILP Constraints** The required constraints for this ILP are as follows:

1. Flow Constraints : Every flow can be transmitted using multiple packets as the flow data size could be larger than the Maximum Transmission Units. Every flow must be allocated number of time-slots which are equal to the number of demands required to transmit that flow.

$$\sum_{\forall k \in S} s_{i,k} = m_q \times d_q;\ \forall q \in M,\ \forall q \in D, \forall k \in S$$

2. Link Constraint: No two flow can be allocated to same time-slot if they have common links.

$$\sum_{\forall i \in DF} s_{i,k} \times fl_{i,j} <= 1; \quad \forall k \in S, \ \forall j \in DL$$

## 4.4 Routing and Scheduling Algorithms for Unicast Scheduling

In the previous section, we explained in detail about the Integer Linear Programming approaches with heuristics. In this section, we present another version of scheduling algorithms. We implement routing and scheduling algorithms to find the paths for flows and assign the time slots to the them. We have implemented k-shortest path algorithm to find all the shortest paths between a source and destination pair and First Fit and Random Fit algorithms to allocate transmission schedules for the end hosts. However, this approach differs from ILP based multi-flow scheduling problem. In contrast to multi-flow scheduling problem, here we have used single-flow scheduling problem. In this approach, the flows are scheduled incrementally without changing the allocated time-slots and routed paths for the already scheduled flows. However, multi-flow scheduling problem has the advantage of having prior knowledge of network flows which may increase the better network utilization and optimization of scheduled flows.

The network topology has been modeled as a bi-directed graph $G \equiv (V, E)$; where V is the set of nodes and E is the network links. And we also assume that S is the number of time-slots available on each network links bi-directionally.

Similar to ILP, the scheduling problem is again sub-divided. First the routing of a flow is computed and then we schedule the flow along its candidate paths. All the flows could be spread across multiple packets in the networks, we consider this as multiple demands per flow. Each flow can have multiple candidate paths. However, these candidate paths are the all shortest path available between a source and destination pair. To avoid collision on the links, we have designed the algorithm such a way that no two overlapping paths can have the same time-slots. Every flow uses the same time-slot on every links to transverse through the selected routing paths.

We describe the routing and scheduling algorithms in the following subsections.

### 4.4.1 First Fit Multiple Paths with Multiple Demands (UNIPM)

**Algorithm 1:** Scheduling and Routing Algorithm with First Fit approach

**Input** : *Graph* $G \equiv (V, E)$, Set of time-slots, Set of flows
**Output**: Transmission Schedules for time-sensitive flows

```
1  BEGIN
2    scheduledFlow ← [ ]
3   for all flow in setOfFlows do
4     listshortestPath ← [ ]
5     listshortestPath ← computeAllShortestesPath(flow.source, flow.destination)
6     for all path in listshortestPath do
7       availableTimeSlotlist ← [ ]
8       remainingSlots ← [ ]
9      for all links in path do
10       availableTimeSlotlist ← getTimeSlotOnLinks(link)
11     setOfTimeSlots ← set.intersection(availableTimeSlotlist)
12     if setOfTimeSlots ≥ flow.demands then
13       assignSlots ← setofTimeSlots.popFirstElements(flow.demands)
14       scheduledFlow.add(flow, assignSlots)
15       remainingSlots ← setOfTimeSlots − set(assignSlots)
16     else
17       Reject flow
18     end if
19       remainingSlots ← updateTimeSlotOnLinks(links)
20   return scheduledFlow
21   END
```

In the described algorithm 1, to begin with, we iterate over all set of flows (line 3), which are set of data streams between an unique pair of source and destination with multiple demands. For this algorithm, we use the set of all shortest paths between a source and destination pair of a given flow as its candidate paths. These paths are computed using k-shortest paths algorithm (line 5). A set of time-slots are allocated to each link bidirectionally. To schedule a flow, we search for required number of time-slots, which are equal to the number of demands of a flow. To assign time-slot to a given flow, we iterate over all the available candidate paths. We check for the available time-slots on each link of path. To find out the set of available time-slots over a path, we take an intersection of all the available time-slots of each links (line 11). Next, we check if this intersection list contains more or equal number of time-slots required to satisfy the demand of the particular flow. If a path satisfies the demand, we schedule the flow over that path with first available time-slots from the intersection list (line 13). If none of the candidate paths of a flow have the required number of time-slots, we reject that flow (line 17). To avoid allocating same time-slots to multiple overlapping paths, once a time-slot on a link is given to a schedule a flow, we remove that time-slot from that link (line 15). The algorithm returns the list of scheduled flows with allocated time-slots.
This algorithm considers set of all shortest paths for a flow. Furthermore, we extend this

approach to a more specific approach. We choose one shortest path randomly from a set of given paths for each flow. The design of the algorithm does not change much except in line 4 instead of computing all shortest paths, we just take one path among them. This reduces the search space of the algorithm and it only deals with assigning time-slots to the chosen path. Though this approach reduces the complexity of the scheduling algorithm but it comprises on the quality on scheduling the number of flows.

### 4.4.2 Random Fit with Fixed Path Multiple demands (UNIFM)

**Algorithm 2:** Routing and scheduling algorithm with fixxed-path heuristics and random fit

**Input** : Set of time-sensitive flows, Transmission Slots, Network Topology
**Output**: Transmission Schedules for time-sensitive flows

```
1  BEGIN
2    scheduledFlow ← [ ]
3   for all flow in setOfFlows do
4      allshortestPath ← getallShortestesPath(flow.source, flow.destination)
5      randomPath ← selectRandom(allshortestPath)
6      availableTimeSlotlist ← [ ]
7      remainingSlots ← [ ]
8     for all link in randomPath do
9       availableTimeSlotlist ← getTimeSlotOnLinks(link)
10      setOfTimeSlots ← set.intersection(availableTimeSlotlist)
11    if setOfTimeSlots ≥ flow.demands then
12        assignSlots ← setofTimeSlots.popRandomElements(flow.demands)
13        scheduledFlow.add(flow, assignSlots)
14        remainingSlots ← availableTimeSlotlist − assignSlots
15     else
16       Reject flow
17     end if
18       remainingSlots ← updateTimeSlotOnLinks(links)
19   return scheduledFlow
20   END
```

This algorithm is also designed in a similar fashion to compute transmission schedules for time-sensitive flows. This scheme searches for all the available time-slots on all the links of a path between a source and destination. From all the available time-slots, the scheduling algorithm assigns random time-slots to the time-sensitive flows (line 12). These two are the only differences between the algorithm 1 and 2. In the algorithm 2, we have shown fixed-path scheduling problem. Where instead of using all shortest path for a flow, we just choose one random path in line 5. Thus, we need not to iterate over the set of all candidate paths for a given flow. Rest of algorithm works pretty much similar to the algorithm 1. However, we have also used both of these path-sets and fixedpath heuristics for single demand, the only

difference is that every flow has just one demand. So they get at most one time-slot. We named it as, UNIMS and UNIFS.

### 4.4.3 Time Complexity of the Algorithms

We designed the algorithms to schedule the time-sensitive flows to have a better scalability in terms of reduced run-time in compare to the ILP based solutions. Therefore, it is important to analyse the time-complexity of the proposed algorithms. It helps us to have a better understanding on how the problem size impacts the run-time of a particular scheduling algorithm.

```
for flow in listOfFlows:
    #get all the candidate paths for flow.
    listOfpaths= getPaths(flow)
    for path in listOfpaths:
        for link in path:
            #check if slots are available #
```

In the described algorithm 1 in line 3, we begin with iterating over the set of flows. Next, in line 6, we iterate over nested loop of the list of paths for a flow. Again, we iterate over the number of links of a path in line 8. This can be mapped to the above sample code.
So, the first outermost loop executes n times for set of flows to be scheduled. For each iteration over flows, the number of execution of second inner loop is the number of paths available for a flow. Now, it depends on the network topology and the source and destination of the flow that how many candidate paths a flow has. A flow can have just one path, if the source and destination are connected to the same switch and if they are from different switch they can have path diversity (number of candidate paths) depending on the topology. For instance, in a leaf-spine topology, a flow having source and destination connected to different leaf switches, then the number of candidate paths is equal to the number of spine-switches in the network. In our algorithm, this candidate paths are shortest paths between a pair of host and destination, so the number of paths for a flow is much lesser than the total number of flows to be scheduled (i.e. n) which is denoted by k. Hence, the body of the second iteration executes $n * k$ (where k $<<$ n). The third iteration is over the number of links in a path. Our approach is restricted to local area networks with network diameter of maximum of 7 switches between any pair of hosts which is consistent with IEEE 802.1D standard. Hence, the total number of links between any pair of hosts would be maximum of 8, comprised of 2 edge links and 6 core links between the switches. So, in worst case, the third loop executes 8 times for each path. The total number of execution is $n * k * 8$. The order of the algorithm is $O(kn)$.

In case of the algorithm 2, we selected a random path out of all the available candidate paths in line 5. It reduces the complexity further. We iterate over flows (line 3) and then over the links in the selected random paths (line 8). The body of the second inner loop executes 8 times in worst case. Thus, the complexity is $O(n)$.

# Chapter 5

# Scheduling Multicast Flows

In this section, we will discuss about the multicast communication for time-sensitive networks. Traditionally, communication in network is unicast where one source communicates with one single destination. Multicast communication allows the sender host to send packets to a subset of a group of hosts simultaneously. IP multicast is a widely used technology these days. It facilitates the idea of using one single data stream traffic from the sender host to the multiple destination hosts. Thus, it reduces the bandwidth utilization in the network. The data packet gets replicated in the network where the distribution path for the multicast group diverse [26]. Multicast applications examples are distributed data processing, storage area networks, e-Science, Video and Audio conferencing, distance e-learning, distribution of software, stock quotes, and news. Multicast uses a group concept. Group is typically consists of an arbitrary number of receivers who are interested to receive a particular data stream from a source. The size of these groups can vary from small to larger one. In optical communication also multicast communication are widely used. In optical communication wave length division multiplexing(WDM) network uses multicast-capable wavelength-routing switches at the network switches who are capable of replicating the data stream from the incoming source port to multiple destination ports. ILP is commonly used to solve the multicast routing problem in optical networks.

## 5.1 Overview of Scheduling Multicast Flows

In this section, we introduce different approaches to address scheduling of multicast flows in networks (figure 5.2). The most widely suggested approach to model a multicast problem in literature for communication networks are canonical approach[43]. In this particular approach a Steiner tree is used for the transmission of multicast flows analyzing all the possible cuts of the network graph [19]. But the main problem with this kind of approach is that all the possible cuts increases almost exponentially as the number of nodes increases in the network. Therefore, canonical approach is mostly not used in practise for solving multicast communication problem. The other approach is flow formulation [32] . In this approach, the path from source host to each of the destination host is computed as unicast path with node-link flow conservation constraints. Then all the links which are part of at least one unicast path are connected to the tree.

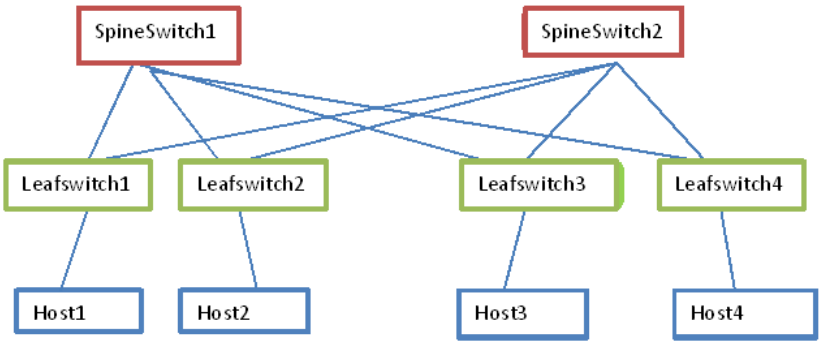



Figure 5.1: An example of leaf-spine topology

In our work, the first approach is based on building candidate trees formulation which can be compared as link-path formulation of unicast flow. To be more precise, each multicast group has multiple candidate trees that originate at the source hosts and connect all the receiver hosts of that group. For a multicast group communication to be established, one of the tree is selected and use for setting up the flows. The algorithm selects one of the candidate paths and assign time-slot to it. We call this approach as multiple trees and this has been implemented with non and non-ILP based solutions. In detail, we have used leaf-spine topology[22]. As shown in the figure 5.1, in a leaf-spine topology, the spine-switches are connected to all the leaf-switches in the network. As we can see in the figure 5.1, there is a mesh topology which provides multiple routing paths between different hosts. The mesh insures redundancy (useful for backup paths) and all hosts are exactly the same number of segments away and contain a predictable and consistent amount of delay or latency [10].

We also implement another method to address multicast communication without forming any multicast tree. We treat all the flows in one multicast group as unicast flows and they can choose any of the available candidate paths but they have to be transmitted using the same time-slot.

In a nutshell, all the multicast formulations in our work address the following sub-problems[18] :

1. construction of trees for routing of multicast flows.
2. assign time-slots to the constructed trees.
3. routing of unicast flows using same time-slot .

However, we have avoided designing multicast scheduling algorithm with dynamically forming multicast trees with ILP, it takes more time which is not suitable for our purpose of calculating transmission schedules for time-sensitive flows. We have formulated different approaches of multicast scheduling as shown the figure 5.2.

In the following section we present our formulation for different problem with multicast

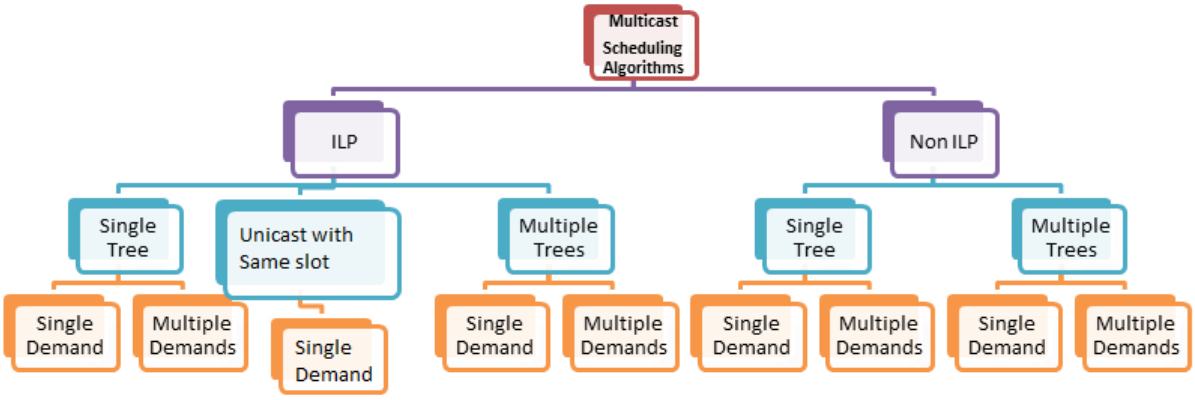



Figure 5.2: Classification of scheduling algorithms for multicast flows

scheduling.

## 5.2 Terminologies and Notations

Given a network, $G \equiv (V, E)$; where G is a digraph, V is the set network nodes and E represents all the links in the network. One multicast group is defined as $M \equiv (s, D, m)$; *where* $s \in V$ source node of the multicast request, $D \subset V - \{s\}$, is the set of destination nodes of a multicast group, m is the demand for the multicast group. One flow in a multicast group is defined similar to an unicast flow refer to section 4.1. It is defined as $DF \equiv (s_i, d_i, m_i, p_i)$ where $s_i, d_i \in V$, d is the number of demands for each flow and $p_i$ refers to the flow period.

## 5.3 Scheduling with ILP based heuristics

It is to be noted that like Unicast scheduling, here also all the multicast flows and the demands are static and are known a priori. The optimization problem is mainly concerned about maximization of the number of connection set up between the sender and group of receivers.

### 5.3.1 Multiple Trees with Single Time-slot (MIMTSS)

In this ILP formulation we have taken into consideration that that every multicast group has multiple trees which connects all the hosts of the group who are participating in the group communication. The ILP inputs are as described below:

**ILP Inputs:**

1. Set of multicast groups requesting time-slots, G.
   $G \equiv g_i;\ \forall i \in \{1, 2, 3...No\ of\ MulticastGroups\}$
2. Set of trees available for all the groups, T.
   $T \equiv t_j;\ \forall j \in \{1, 2..No\ of\ trees\}$
3. Mapping between links and trees, TL.
   $TL \equiv f_{j,k};\ \forall j \in T,\ \forall k \in L;\ f_{i,k} = 1$, if tree j traverses the link k, else 0.
4. Mapping between the groups and trees, GT.
   $GT \equiv p_{i,j}; \forall j \in T, \forall i \in G;\ p_{i,j} = 1;$ if group i traverses through the tree j, else 0.

**ILP Variables:** The required ILP variables are enumerated below:

1. Mapping between trees and time-slots, TS
   $TS \equiv y_{i,x};\ \forall j \in T,\ \forall x \in S;\ y_{i,x} = 1;$ if tree j allocated time-slot x, else 0.

**Objective Function:** The objective of this ILP formulation is to maximise the number of time-slots allocation to the multicast trees.

$$\text{Maximize} \sum_{\forall j \in T} \sum_{\forall x \in S} y_{i,x}$$

**ILP Constrainst:** The required constraints for this ILP are as follows:

1. Tree Constraint: Each tree can be allocated at most one time-slot as we are considering all the flows can be transmitted over a single packet.
   $\sum_{\forall x \in S} y_{j,x} \leq 1;\ \forall j \in T$
2. Link Constraint: No two trees uses the same time-slot on a common link.
   $\sum_{\forall j \in T} y_{j,x} \times f_{j,k} \leq 1;\ \forall x \in S, \forall k \in L$
3. Every multicast group gets at-most one time-slot.
   $\sum_{\forall j \in T} y_{j,x} \times p_{i,j} \leq 1;\ \forall x \in S, \forall i \in G$

### 5.3.2 Single tree Single Time-slot (MISTSS)

In this ILP formulation we randomly choose one path for all the unicast communication in one multicast group. We built one tree for each multicast group which connects all the hosts participating in the multicast communication. This is static routing where the routing paths are already computed and assign to the multicast groups. For this formulation we assume that every flow data can be send with one packet. Hence each flow needs only one transmission slot to transmit data.
The inputs to the ILP are as follow:
**ILP Inputs**

1. The multicast group which are sending data within one period, G.
   $G \equiv g_i;\ \forall i \in \{1, 2, ..No\ of\ MulticastGroup\}$

2. In order to route the multicast traffic,we construct one single tree for each group. These trees are constructed using the links between the source and all the destinations of one group. All the trees which are available to the flows in the network, T.
   $T \equiv t_j;\ \forall j \in \{1, 2, ..No\ of\ Trees\}$

3. Mapping between the groups and trees, GT:
   $GT \equiv f_{i,j};\ \forall i \in G, \forall j \in T$
   $f_{i,j} = 1$; if the group i uses tree j for routing, else 0.

4. No of time-slots available for one period, S.
   $S \equiv s_k;\ \forall k \in \{1, 2.., No\ of\ TimeSlots\}$

**ILP Variables:** In this ILP formulation we require only one decision variable to map between the multicast group and time-slot, GS.
$GS \equiv s_{i,k};\ \forall i \in G, \forall k \in S$
$s_{i,k} = 1$, if group i gets time-slot k else 0.

**Objective:** The objective for this formulation is to maximize the number of time-slots allocation to the multicast groups so that maximum multicast groups can transmit data within the period.

$$Maximize \sum_{\forall i \in G} \sum_{\forall k \in S} s_{i,k}$$

**Constraints:** This ILP is formulated based on the following constraints:

1. Group Constraint: Every Multicast group gets at most one single time-slot to transmit data as we have assumed all the flows can be transmitted using one packet.

$$\sum_{\forall k \in S} s_{i,k} \leq 1;\ \forall i \in G$$

2. Link Constraint : Two overlapping trees can not have same time-slot, means if any link can not have same time slot for two different multicast tree.

$$\sum_{\forall i \in G} s_{i,k} \times f_{i,j} \leq 1;\ \forall k \in S;\ \forall j \in T$$

### 5.3.3 Multiple Trees with Multiple Time-slots (MIMTMS)

In this particular ILP formulation, we consider the multicast flows which are can be transmitted with multiple same size of packets means demand for each flow could be more than one. We built all the possible multicast trees for each group and provide them as an input to the ILP. The ILP finds out one tree among all possible trees on which it can assign as many as time-slots are required to transmit the flow. This heuristic approach have two degrees of freedom, one is path and another one is time-slot. So the ILP first chooses the multicast trees and then allocate time-slots to it. It gives near optimal solution where maximum number of multicast groups gets assigned time-slot but it takes longer computation time.

**ILP Inputs:** The inputs to this ILP are enumerated below:

1. Set of Multicast Groups participating in the time-sensitive data transmission, G :
   $G \equiv g_j;\ \forall j \in \{1, 2, ..No\ of\ multicast\ groups\}$
2. Set of No of multicast trees available for routing, T.
   $T \equiv t_k;\ \forall k \in \{1, 2, ..No\ of\ multicast\ trees\}$
3. Set of demands for each multicast group, D.
   $D \equiv d_n;\ \forall n \in \{1, 2, ..No\ of\ multicast\ groups\}$
4. Mapping between multicast group and trees, GL.
   $GL \equiv gl_{j,i};\ \forall j \in G,\ \forall i \in L$; Here L is number of links in the network
   $gl_{j,i} = 1$; if group g uses the link i for its multicast tree, else 0.
5. Set of demands for the multicast groups, GD.
   $GD \equiv gd_q;\ \forall q \in D$

**ILP Variables:** The decision variables required for this ILP formulations are described as follows:

1. Decision variable is required to map between the multicast trees and the time-slots, TS.
   $TS \equiv y_{i,x};\ \forall j \in T,\ \forall x \in S.$
   $y_{i,x} = 1$; if tree j assigned time-slot x, else 0.

2. Binary decision variable is required to verify whether a group has allocated to the required number of time-slots or not, M :
   $M \equiv m_q;\ \forall q \in G$

**Objective Function:** The objective function of this ILP can be formulated to maximize demands or maximize allocation of flows. In the first approach we prioritize the flows which has more numbers of demands and in the second case we give priority to the flows which could be transmitted with lesser number of packets. The formulation for the objective function are given below:

1. Maximize Flows : $\sum_{\forall q \in M} m_q$
2. Maximize Demands : $\sum_{\forall q \in M} \sum_{\forall q \in D} m_q \times d_q$

**ILP Constraints:** This ILP is also a constraint based ILP, the required constraints for this ILP are as follows:

1. Tree Constraints: Each tree will be assigned the number of time-slots required to transmit the set of packets for a multicast group. In other words, we can say that each group will have number of time-slots equals to the demands of the group which traverses through it.

$$\sum_{\forall j \in T} y_{j,x} = m_q \times d_q;\ \forall x \in S,\ \forall q \in GD$$

2. Link Constraint: Any two overlapping trees should not be allocated same time-slot.

$$\sum_{\forall j \in T} y_{j,x} \times f_{j,k} \leq 1;\ \forall x \in S,\ \forall k \in L$$

3. Group Constraints: Every group should be assigned the number of time-slots which is equal to the number of demands for that group.

$$\sum_{\forall j \in T} y_{j,x} \times p_{j,i} = m_q \times d_q;\ \forall q \in M,\ \forall q \in GD,\ \forall x \in S$$

### 5.3.4 Single Tree with Multiple Time-slots (MISTMS)

In this formulation we consider that every flow can be spread across multiple data packets, hence each flow needs multiple time-slots to be transmitted in the network. In this formulation also we have chosen a single random path among all the available candidate paths for every flow. Every multicast group has only one tree to route all the data packets for multicast communication.

The input for the ILP are as following :

**ILP Inputs :**

1. Set of Multicast Groups participating in the time-sensitive data transmission, G :
   $G \equiv g_j;\ \forall j \in \{1, 2, ..No\ of\ multicast\ groups\}$
2. Set of No of multicast trees available for routing, T.
   $T \equiv t_k;\ \forall k \in \{1, 2, ..No\ of\ multicast\ trees\}$
3. Set of demands for each multicast group, D.
   $D \equiv d_n;\ \forall n \in \{1, 2, ..No\ of\ multicast\ groups\}$
4. Mapping between multicast group and trees, GL.
   $GL \equiv gl_{j,k};\ \forall j \in G,\ \forall k \in T.$
   $gl_{j,k} = 1$, if group g uses the tree k, else 0.

**ILP Variables :** This ILP has only one degree of freedom i.e. the time-slot which will be assigned to a multicast tree. Moreover this ILP should satisfy the demands of multicast groups. Thus we assign time-slots to the trees so that it can carry on the data packets required for transmission of a flow. The required ILP variables are as follows :

1. Mapping between the multicast groups and time-slot, GT.
   $GT \equiv t_{j,k};\ \forall k \in T,\ \forall j \in G,$
   $t_{j,k} = 1$ if group j gets slot k, else 0.
2. Set of binary decision variable mapped to the demands of the multicast flows, M :
   $M \equiv m_q;\ \forall q \in \{1, 2, ...D\}$

**Objective Function :** The objective function of this ILP is to maximize allocation of time-slots to the multicast groups. In order to do so, we can have two different objective functions. First, we prioritize the flows having less number of demands by maximizing number of flows that are allocated time-slots. Second, we can maximize the number of demands that could be allocated time-slots. The formulations are as follows :

1. Maximize Flows : $\sum_{\forall q \in M} m_q$
2. Maximize Demands : $\sum_{\forall q \in M} \sum_{\forall q \in D} m_q \times d_q$

**Constraints :** The constraints for this ILP formulation are listed as below :

1. Group Constraint : Each multicast group can be allocated as many as time-slots are required to transmit all the data packets.

$$\sum_{\forall k \in T} t_{j,k} \leq m_q \times d_q;\ \forall q \in M,\ \forall q \in D$$

2. Link Constraint : In order to avoid collision on the links, no two overlapping multicast tress with common links between them will be assigned same time-slot.

$$\sum_{\forall j \in G} t_{j,k} \times gl_{j,i} \leq 1;\ \forall k \in T,\ \forall i \in L$$

### 5.3.5 Multicast with Unicast flows (MIUSS)

Till now we have discussed about the ILPs which are based on using multicast trees formation. In this formulation we are tackling the multicast communication in a different way. We do not form any multicast tree for the group communication rather we compute path for each unicast flows in a multicast group and then we assign one time-slot for the selected candidate paths. We impose a constraint on the unicast flows that they have to be transmitted using the same time-slot. In this way all the unicast flow acts like a multicast group communication. For simplicity we have considered that each flow can be transmitted using one single packet over the network. So the ILP has to assign at-most one time-slot to the selected path. The detailed formulation of the ILP are as follows:

**ILP Inputs:** The input to ILP are described as follows :

1. The set of multicast flows, F.
   $F \equiv \{df_i\};\ \forall i \in \{1, 2, 3...No\ of\ Flow\}$
2. Set of computed all possible candidate paths using Equal distant shortest path(ECMP) algorithm to setup connections between the sender host to all of the receiver hosts in a multicast group, P.
   $P \equiv \{p_k\};\ \forall k \in \{1, 2, ...No\ of\ Paths\}$
   This set of paths contains all the shortest paths from a source host to all the destination hosts for each flow in multicast flows $df \in F$.
3. Mapping between the multicast groups and the flows, GF.
   $GF \equiv \{gf_{i,j}\}$ :
   $\forall i \in F,\ \forall j \in G;\ gf_{i,j} = 1$, if the group i has the flow j, else 0.
4. Mapping between the paths and the links, PL .
   $PL \equiv \{pl_{k,m}\};\ \forall k \in P,\ \forall m \in L$ :
   $pl_{k,m} = 1$, if path k uses link m, else 0.
5. Mapping between all flows and the paths, FP.
   $FP \equiv \{fp_{i,k}\};\ \forall k \in P,\ \forall i \in F$ :
   $fp_{i,k} = 1$, if the flow i uses the path k, else 0.
6. Mapping between groups and paths, GP.
   $GP = \{gf_{j,i} \times fp_{i,k}\} \equiv \{gp_{j,k}\};\ \forall i \in F, \forall j \in G, \forall k \in P$ :
   $gp_{j,k} = 1$, if group j uses path k, else 0.

7. Mapping between the groups and the links, GL.
   $GL = \{gp_{j.k} \times pl_{k,m}\} \equiv gl_{j,m};\ \forall j \in G,\ \forall k \in P,\ \forall m \in L :$
   $gp_{j,m} = 1$, if the group j uses the link m, else 0.

**ILP Variables:** In this ILP formulation, we allocate time-slots to a multicast group so every flow in that group gets the same time-slot. One variable is required to map between the time-slots to the multicast groups, GS.
$GS_{\equiv}\{s_{j,x}\};\ \forall j \in G,\ \forall x \in S\ :$
$s_{j,x} = 1$, if group j gets slot x, else 0.

**Objective Function:** The objective for this ILP formulation is to maximize the number of multicast groups with the assigned time-slots.

$$\text{Maximize} \sum_{\forall j \in G} \sum_{\forall x \in S} s_{j,x}$$

**ILP Constraints:** The required constraints for this ILP formulation are listed as below :

1. Group Constraint: Each group should get at-most one time-slot so that all flows in a given group transmit at the same time.

$$\sum_{\forall x \in S} s_{j,x} \leq 1;\quad \forall j \in G$$

2. Link Constraint: No two multicast group should be allocated same time-slot if they have a common link to traverse between the source host to set of destination hosts.

$$\sum_{\forall j \in G} s_{j,x} \times gl_{j,m} \leq 1;\quad \forall m \in L,\ \forall x \in S$$

## 5.4 Routing and Scheduling Algorithms for Multicast Scheduling

ILP based heuristics give near optimal solutions, but it suffers from scalability (run-time for computation) issue. The run-time to create the ILP increases exponentially as the number of multicast trees and available number of time-slots increase. To address this scalability problem, we propose non-ILP based routing and scheduling algorithms to compute transmission schedules. This approach provides better scalability in terms of lesser computation time to calculate transmission schedules but it compromises on quality.
To implement the algorithms, we have used similar directed network digraph $G \equiv (V, E)$, where V is the set of network nodes and E is the set of bidirectional network links which connects the network nodes. Furthermore, $V \equiv S \cup H$,where S is the set of network switches and H is the set of end hosts.
Unlike multi-flow scheduling in ILP based approach, here we have used single flow scheduling for multicast flows. We built a set of possible candidate trees for each multicast group. As

we have used leaf-spine topology, the multicast trees are splitted at the spine switches and number of multicast trees for a given group is equal to the number of spine switches in the network. Each of these trees originate at the source node of a multicast group and connects all the destination nodes for that group. We also assign time-slots on the bidirectional links.

### 5.4.1 First Fit with Multiple Trees Multiple Time-slots (MNIMTMS)

**Algorithm 3:** First Fit Scheduling Algorithm for Multicast

**Input** : *Graph* $G \equiv (V, E)$, Set of Multicast Group $M$, Set of time-slots , Set of multicast flows DF.
**Output**: Transmission Schedules for time-sensitive flows

```
1  BEGIN
2  dictMulticastTree ← {}
3    for (group,setOfDestination) in M do
4      multicastTreelist ← [ ]
5      dictsourceTree ← [ ]
6      multicastTree ← computeTree(source, setOfdestination)
7      multicastTreelist.add(multicastTree)
8      dictMulticastTree[group] = multicastTreelist
9      dictMulticastTree.update(multicastTreelist)
10    for all (group,multicastTreelist) in dictMulticastTree do
11       for all tree in multicastTreelist do
12         availableTimeSlotlist ← [ ]
13         remainingSlots ← [ ]
14          for all link in tree do
15            availableTimeSlotlist ← getTimeSlotOnLinks(link)
16          setOfTimeSlots ← set.intersection(availableTimeSlotlist)
17       if setOfTimeSlots ≥ group.demands then
18            assignSlot ← setofTimeSlots.popFirstElement(group.demands)
19            allocatedGroup ← assignSlot
20            remainingSlots ← (availableTimeSlotlist − assignSlot)
21       else
22         GO TO NEXT TREE
23       end if
24         remainingSlots ← updateTimeSlotOnLinks(links)
25   return (allocated group,assignSlot)
26   END
```

In the algorithm 3, we compute a set of available multicast trees for each group in line 7. Each of these trees connects the source and set of all destination hosts for a multicast group. We iterate over each link of a tree and from each link, we get the available time-slots and make an intersection list of all these available time-slot (line 16). Next, we compute the available time-slots in that multicast tree (line 15). If it satisfies the number of demands of that particular group, we allocate the first available slots from the intersection list (line

18), else we go to the next multicast tree. If none of the multicast trees have the required number of time-slots, the multicast request is rejected and we go to the next group. After each iteration of a group, we remove the allocated time-slots from the traversed links (line 24) to avoid collision on links. At the end, the algorithm return a list of schedule multicast group with allocated time-slots.

We have also implemented another algorithm which extends the idea of multiple multicast trees. We select a random multicast tree from the available multicast trees for a group. Then the algorithm works only to assign a time-slot for a single multicast tree. It reduces the complexity of the algorithm by minimizing the search space of the available trees. Though this single tree multicast algorithm is faster than the multiple tree multicast approach, the number of scheduled flows is much less than the previous approach. It is to be noted that, multicast flows with single demand, we have used the same algorithms as Multiple trees and single tree heuristics (only the demand value in line 17 is always one). They are denoted as MNISTSS and MNIMTSS.

### 5.4.2 Complexity Of the Algorithm

We have designed and implemented the algorithms to reduce the run-time time to compute schedules compared to ILP based heuristics. Therefore, it is important to analyse the time complexity of these designed algorithms. Our ILP based heuristics solutions are NP hard optimization problems, whereas scheduling algorithms are whose which can be solved in polynomial time.

In the algorithm 3, we have a list of available multicast trees for each group. We iterate over each group with the list of multicast trees (line 10). Then, in line 11 and 14, we again iterate over each tree and each link in a tree respectively. In every iteration over a tree, it checks if that tree can satisfy the required number of time-slots for that group, otherwise it goes to the next iteration. Now, for the worst cast, a multicast group has to iterate over all the available multicast trees to search for required number of time-slots. This can be mapped to the following sample code.

```
for mcast_Flows in list_Mflows:
    #get all the candidate trees for a multicast group.
    listOfTrees= computeTrees(mcast_Flows)
    for tree in listOfTrees:
        for link in tree:
            #check if slots are available #
```

So, the first outermost loop executes n times for set of flows to be scheduled. For each iteration over a multicast flow, the second loop executes as many times as the number of multicast trees it has. Now, it depends on the network topology and the source and destinations of the multicast flows that how many multicast trees it has. For instance, in a leaf-spine topology 5.1, maximum number of trees a multicast flow can have is equal to the number of spine-switches in the network, which is denoted by k. Hence, the body of the second iteration

executes $n * k$(where k $<<$ n). The third iteration is over the number of links in a tree. It can vary depending on the destination hosts of a multicast flow and can be denoted as m. So the order of the algorithm is $O(n * k * m)$.

However, if we use single tree multicast algorithm, it reduces the time-complexity. As we only give one multicast tree as an input to each group, hence it eliminates the iteration over all possible multicast trees (line 11). Thus the time-complexity of single tree is $O(n * m)$.

# Chapter 6

# Evaluation

This chapter analyzes the performance of implemented algorithms. The analysis of the algorithms were conducted on a simulated network with varying size of the network topology, number of simulated data flows and available number of time-slots on the network links. We evaluated the algorithms to measure quality and scalability. The results of evaluation widely varied between two different implementation approaches. The performance of the ILP based approach in terms of quality is far better than the non-ILP based algorithms. However, ILPs take considerable amount of time to give near optimal solutions whereas routing and scheduling algorithms are faster, but they compromise on quality. In the following sections we will analyze our experiments results in detail.

## 6.1 Experiments Setup

The experiments with ILP based heuristics were performed on a multi-processor machine with 2×8 cores, while routing and scheduling algorithms were evaluated on a machine with 3 cores and 2.5 GHz processor. To compute the transmission schedules using ILP based heuristics, we have used the commercial solver CPLEX [5]. All the evaluations have been performed in a simulated network. We created our network using python supported networkx 1.9.1 [13], where the network switches and the end hosts are networkx nodes, the links between the nodes are networkx edges. We simulated set of random flows between the network nodes, which acts as an input to the system. To route these set of flows, we used networkx provided shortest path algorithms. It implements k-shortest path routing algorithm to find out all the possible shortest paths between a source and destination pair.

In detail, we used leaf-spine topology (see figure 5.1). The sizes of implemented leaf-spine topologies, number of flows as well as available number of time-slots have been taken into consideration in the separate evaluations. To calculate transmission schedules for unicast and multicast flows using ILP based solutions, first we created the ILP and then invoked CPLEX solver for optimum transmission schedules. For algorithm based approach, we allocated time-slots based on the first come first serve basis meaning we assign time-slot to one flow at a time.

## 6.2 Evaluation of Unicast Scheduling

To begin with, we evaluated the quality and scalability of all the ILP and non-ILP based scheduling algorithms for unicast scheduling. All the evaluations for unicast scheduling, we used algorithms UIPM and UIFM for ILP based approaches and UNIPM and UNIFM for non-ILP based solutions.

### 6.2.1 Quality Of Solutions

To evaluate the quality of the ILP and non-ILP based heuristics (fixed-path, path-sets), we computed transmission schedules iteratively with different scenarios. To measure the quality of the solutions, we have chosen a smaller topology of 6 switches (3 spines, 2 leaf switches per spine) and 24 hosts with less number of available time-slots on the links. We focus more to have a near optimal solution where a large set of flows are competing for a small number of time-slots. In our evaluation, for unicast communication, each of the scenario comprised of 20 - 100 random flows with unique source and destination hosts to be scheduled with 3 - 10 available number of time-slots. In order to simulate a highly challenging scenario, we deliberately choose a smaller number of available time-slots in the network. All the requested flows have multiple demands between 2 - 5.

#### 6.2.1.1 Evaluation of ILP based solutions

To compare the quality of the solution, we measured a the number of scheduled flows against the number flows requested for both fixed-path and path-sets multiple slots. We observe in figure 6.1, that path-sets is able to schedule 90 % of the requested flows, whereas fixed-path can only schedule 60 % of the flows on an average. With same topology and set of flows, path-sets heuristics gives 30 % better quality.

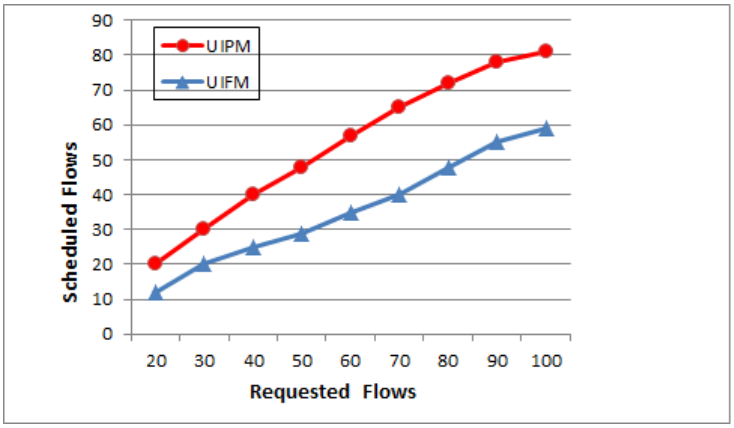



Figure 6.1: Quality evaluation of Unicast Scheduling with ILP based heuristics (UIPM and UIFM)

#### 6.2.1.2 Evaluation of Routing and Scheduling algorithmic solutions

In non-ILP based heuristics approaches also we used same topology with same set of flows and number of slots. We did our experiments with first fit and random fit scheduling algorithm and have plotted the average scheduled flows by them in figure 6.2. Path-sets and Fixed-path schedules 70% and 45% of the requested flows respectively.
In both of our approaches we see that, quality improvement achieved by using path-sets heuristics. Increasing number of candidate paths for a flow has a significant effect on the results.

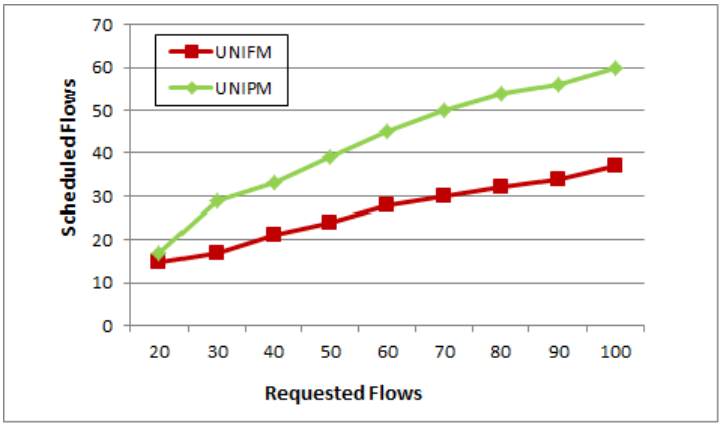



Figure 6.2: Quality evaluation of Unicast Scheduling with algorithm based heuristics (UNIPM and UNIFM)

### 6.2.2 Scalability Of Solutions

After investigating the quality of ILP and non-ILP based heuristics, next we evaluated their scalability. To evaluate scalability, we measured the required time to compute the solutions for different scenarios. The factors that effect the run-time for computing transmission schedules are: the number of flows to be scheduled, size of the network topology and the available number of time-slots. To begin with, we measured the scalability of ILP based heuristics first followed by algorithm based heuristics .

#### 6.2.2.1 Evaluation of ILP based solutions

1. First, we evaluated the scalability of the path-sets (UIPM) and fixed-path (UIFM) in ILP based solutions. Figure 6.3 shows that UIFM has a linear increment in run-time with increasing number of links. It is able to schedule each flow in 2.5 sec on average. We observe a sharp increase in run-time for UIPM in figure 6.3. To increase the path diversity in path-set, we increased the number of spine switches in the topology from 2 - 6 with varying number of 36 to 248 links. The worst average case to schedule a flow

for path-set heuristics is 15 s, which might not be acceptable in real time scenarios to schedule a time-sensitive flow .

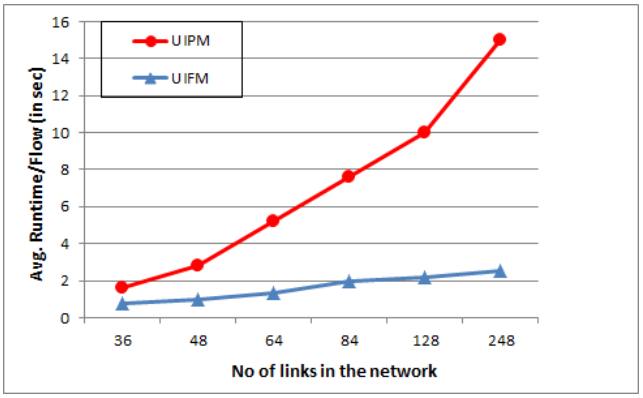
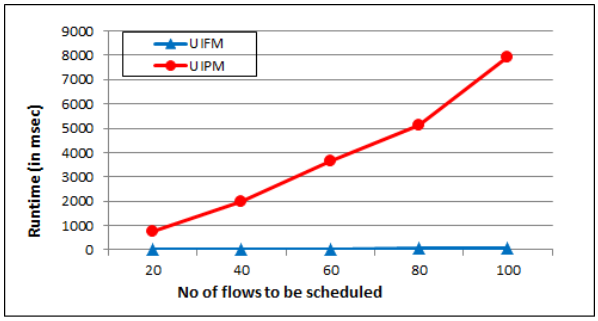
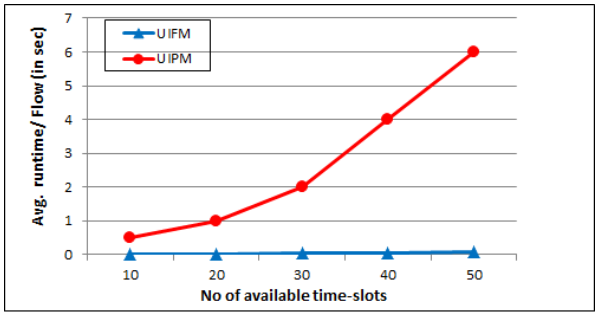



Figure 6.3: Runtime vs Topology Size Unicast Scheduling with ILP based heuristics (UIPM and UIFM)




(a) Runtime vs No of flows Unicast Scheduling with ILP based heuristics (UIPM and UIFM)




(b) Runtime vs No of time-slots Unicast Scheduling with ILP based heuristics (UIPM and UIFM)

2. We evaluated the impact of increasing number of flows on run-time. To measure the run-time, we used a topology with 24 hosts and 6 switches, we assigned 5 slots on each links and varied the number of flows between 20 - 100. Moreover, each flow has multiple demands. Figure 6.4a shows that, fixed-path heuristics (UIFM) takes less than 100 msec to schedule 100 flows. In our evaluation, it took maximum 76 msec to schedule 100 flows. Path-sets heuristics (UIPM) shows an exponential increment in run-time with the increasing number of flows. It takes on average 8 s to schedule 100 flows.

3. Next, we evaluated the impact of increasing number of slots on run-time. We used a topology of 120 links and placed over 100 flows on them. We varied the number of slots ranging from 10 to 50. Figure 6.4b again shows that fixed-path heuristics takes very less time to schedule one flow (it took 900 msec to schedule 100 flows with 50 time-slots), whereas path-sets takes 6 s to place one flow on the same topology. From figure 6.4b, it is clear that for path-sets, run-time increases rapidly with increasing number of

slots. Thus, if we have huge number of slots available and flows have to be scheduled in a very short time, this heuristics will not be feasible to implement in such real time applications.

#### 6.2.2.2 Scalability evaluation of Routing and algorithms

1. At first, we evaluated the run-time for a varying number of flows. We measured the run-time for these scheduling algorithms in Leaf-spine topology with 20 switches and 120 hosts ( 5 spines, 4 leaf switches per spine and 6 hosts per leaf switch), 50 time-slots and varying number of (100 - 300) flows. As shown the figure 6.5, run-time for algorithm UNIFM increases linearly and it scheduled 300 flows in 2 s . Run-time for Path-sets (UNIPM) increased rapidly with the increasing number of flows. Nonetheless, it took on average just 11 s to schedule 300 flows. Thus we can infer, that both of these heuristics based approaches provide sufficient scalability to support a time-sensitive networks of 120 hosts and quite a large number of flows.

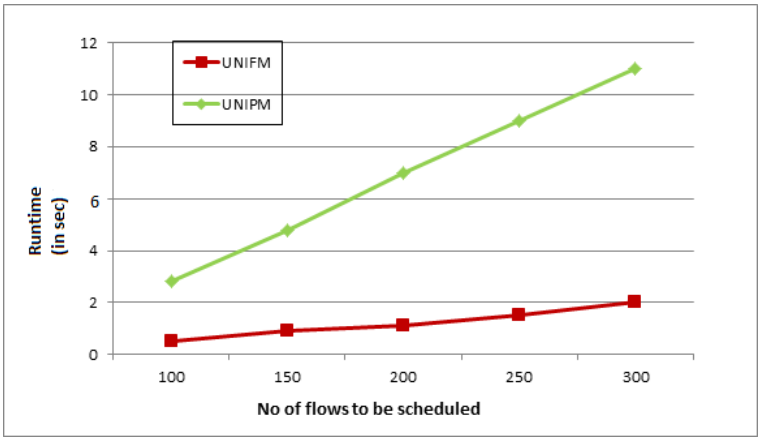



Figure 6.5: Runtime vs No. of flows for Unicast Scheduling with routing and scheduling algorithm based heuristics (UNIPM and UNIFM)

2. We also evaluated the effect of the size of the topology on the run-time of the algorithms using path-sets and fixed-path heuristics. Each of the scenarios have been evaluated with 36 - 248 number of available network links and 7 available time-slots and 200 flows with multiple demands. We have measured the average run-time per flow. The experiments results shows that run-time in case of fixed-path increases linearly with varying number of network links. Figure 6.6 shows that it takes on an average 2.2 msec to schedule a flow in a topology with 248 links. The run-time for path-sets are not directly proportional to the size of the topology. Rather, it depends on number of candidate paths for a flow. In our experiments in particular, we have varied the number of spine switches in the topology to have more path diversity of the network. We observed that with increasing number of available shortest paths for a given flow, run-time time increases. The average run-time for each flow with 248 links in the network is 15 msec.

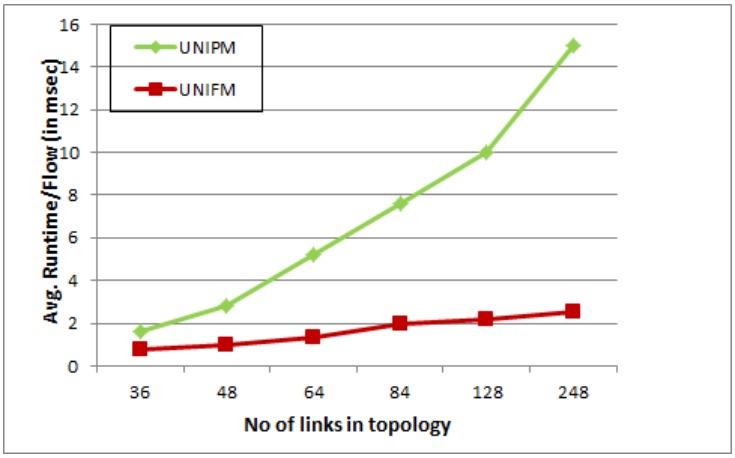



Figure 6.6: Runtime vs Topology Size for Unicast Scheduling with algorithm based heuristics (UNIPM and UNIFM)

3. Further, we evaluated the impact of increasing number of time-slots on the run-time. We evaluated with a leaf-spine topology of 40 switches and 240 hosts, 120 links and we placed 800 flows on them. We varied the number of time-slots from between 100 and 500. We scheduled 800 flows using the algorithms with UNIPM and UNIFM. Fixed-path schedules each flow on an average of 10 msec and path-set needs 60 - 70 msec to schedule one flow. Figure 6.7 shows, the run-time of path-sets is not proportional with increasing number of slots. As it found more number of slots available on each path for a flow, which reduced the number of iteration over the candidate paths and the computation time also decreased to schedule a flow. For fixedpath the computation time is almost linear.
   We can infer from these evaluation results that, non-ILP based solutions can be used in network with huge number of time-slots and larger topology for its better performance in terms of scalability.

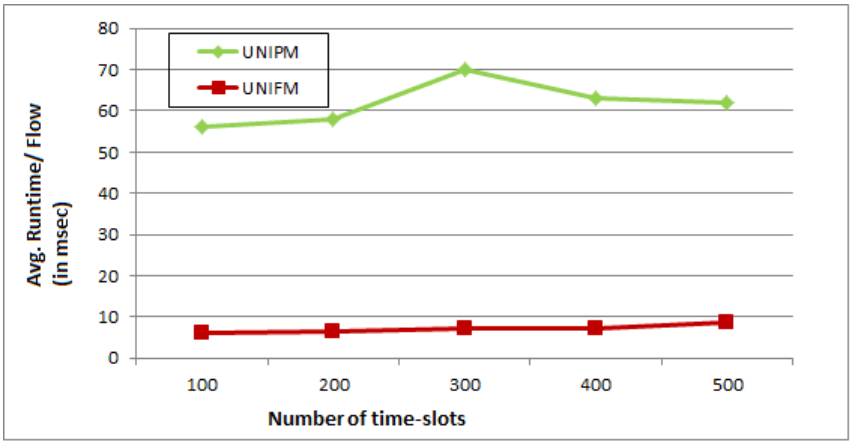



Figure 6.7: Runtime vs time-slots for Unicast Scheduling with algorithm based heuristics (UNIPM and UNIFM)

## 6.3 Analysis of Unicast Evaluation

### 6.3.1 Quality Analysis

We evaluated Intra-ILP and Intra-Algorithmic solutions for fixed-paths and path-sets. Furthermore, to have a quality comparison of ILP based and non-ILP based solutions, we compared both of these approaches. Comparing the results of the solutions, it is clear that, increased number of scheduled flows achieved by using multiple paths for each flow. Figure 6.8 shows the quality of different solutions achieved by different heuristics and algorithms. We achieved the best near optimal solution by path-sets heuristics. For example, ILP based path-sets scheduled 80 % of the requested set flows whereas algorithm based path-set scheduled 60% of the flows. The results of ILP based fixedpath and algorithm based path-sets are very close.

### 6.3.2 Scalability Analysis

After analysing the quality of ILP and routing and scheduling algorithms of Unicast, next we analyse the scalability of these approaches. We have compared the impact of increasing number of slots, number of links and number of flows on run-time separately. The scalability of routing and scheduling algorithms is better than the ILPs with respect to increasing number of links, slots and flows. Figure 6.9, 6.10, 6.11 show that ILP with path-sets is effected most with the change in network parameters. The run-time for ILP based path-set increases rapidly for all three of the scenarios. The other approaches showed almost linear behavior. For instance, in figure 6.10 path-sets (UIPM) took 25 s to schedule a flow with 248 links, while fixedpath (UIFM) took less than 5 s. The run-time for all the approaches with routing and scheduling algorithm is in milliseconds. Figure 6.9 shows that UIPM takes 6 s to place one flow with 50 slots while the other approaches take less than a second to schedule a flow. Thus, ILP with

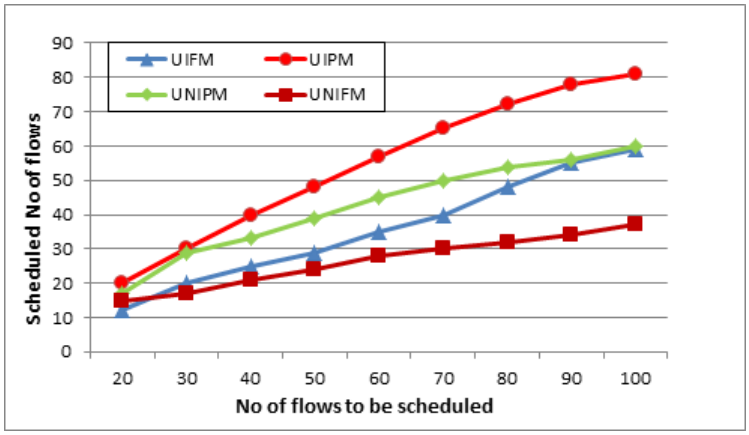



Figure 6.8: Quality Analysis of Unicast Scheduling approaches (ILP and non-ILP based heuristics)

path-sets can not be used in larger topology with huge number of flows to be scheduled and number of slots more than 50 or so.

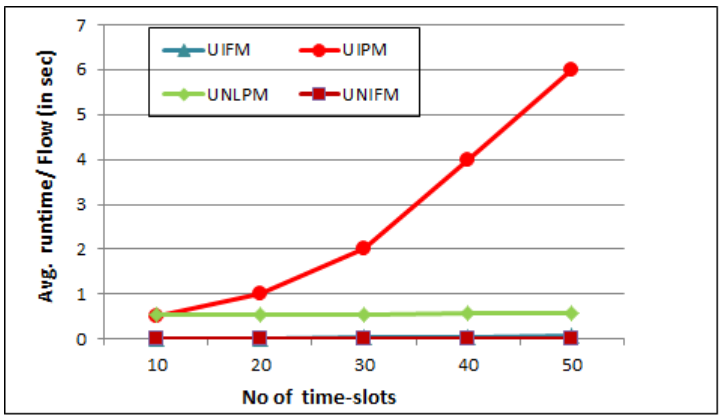



Figure 6.9: Runtime vs No of time-slots for Unicast Scheduling approaches (ILP and non-ILP based heuristics)

So far, we analyzed all the approaches for unicast scheduling. ILP based UIPM gives near optimal solution for computing transmission schedules but it does not scale as much as the other heuristic UIFM. Non-ILP based heuristics scales much better than the ILPs but they deviates from near optimal solutions largely. Thus, it is a trade off between quality and scalability. Depending on the network size, number of flows, available time-slots and expected quality of solutions the scheduling approaches could be selected for a particular network.

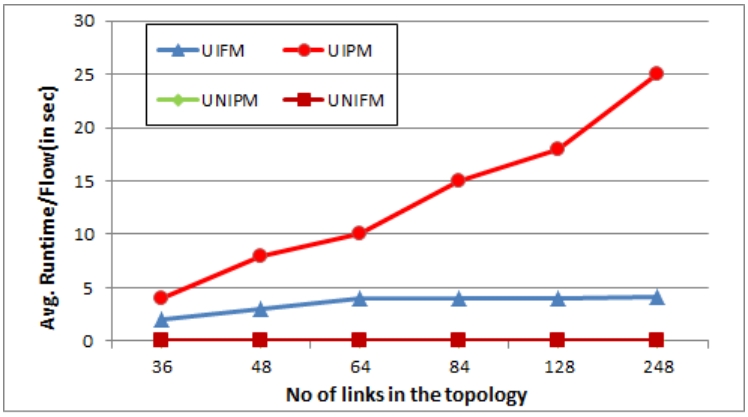



Figure 6.10: Runtime vs Topology size for Unicast Scheduling approaches (ILP and non-ILP based heuristics)

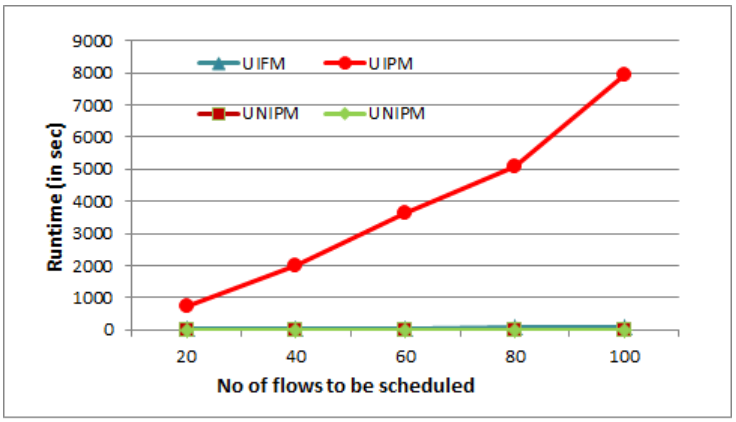



Figure 6.11: Runtime vs No of flows for Unicast Scheduling approaches (ILP and non-ILP based heuristics)

## 6.4 Evaluation of Multicast Scheduling

After evaluating the quality and scalability of scheduling algorithms for unicast, next we evaluated the multicat scheduling algorithms. Similar to unicast, we have evaluated the implemented multicast problems using ILP based heuristics and routing and scheduling algorithms with the same experiments setup.

### 6.4.1 Quality of solutions

To begin with the experiments of multicast solutions, we have considered all the multicast flows have single demand meaning each of the flow needs one transmission-slot to transmit

all the data packets. We measured the multicast solutions with a topology of 36 hosts and 6 switches, 5 time-slots and varying number of (5 - 35) multicast flows.

#### 6.4.1.1 Quality evaluation of ILP solutions

In ILP based solution, we observed that single tree (MISTSS) schedules quite a less number of multicast flows in compared to multiple trees (MIMTSS) and unicast with same time-slot (MIUSS) approaches. However, multiple trees heuristic schedules almost same number of multicast flows as unicast with same time-slots heuristic. Figure 6.12 shows MISTSS schedules maximum number of 12 multicast flows out of 35 requested flows and the solution does not scale with the increasing number of flow requests. MIMTSS and MIUSS can schedule maximum of 25 multicast flows out of 35 flows. The quality improvement is achieved by using multiple options to route the multicast flows over multiple available paths rather than selecting one random tree.

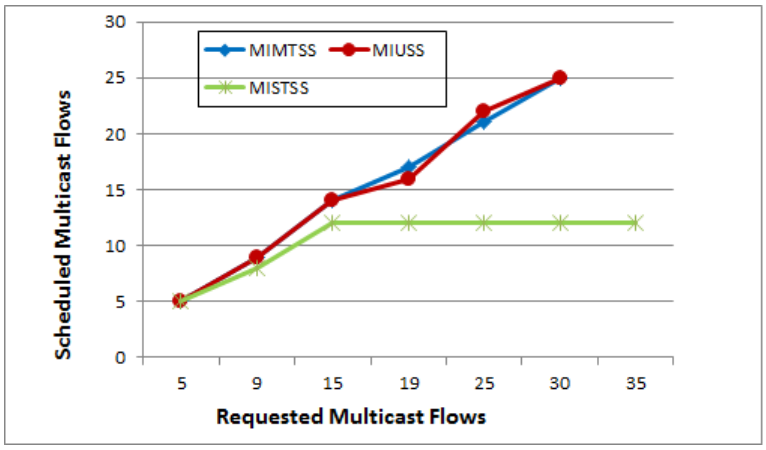



Figure 6.12: Requested vs Scheduled Multicast Flows for ILP based heuristics(MIMTSS, MISTSS, MIUSS)

#### 6.4.1.2 Quality evaluation of Routing and Scheduling Algorithms

Further, we measured the quality of implemented non-ILP based algorithms for multicast scheduling. We evaluated the quality of solution provided by single tree (MNISTSS) and multiple trees (MNIMTSS) heuristics. We have used the same set of parameters for this experiment as described in section 6.4.1.1 except we have used 10 time-slots instead of 5. It is quite obvious that multiple trees can schedule more number of multicast flows than single tree. Figure 6.13 shows that multiple trees approach schedules up to 19 multicast flows where as the single tree can only schedules maximum of 10 flows out of 35 requested flows. In single tree approach, the quality of the solution varies on the random selection of the multicast tree. Multiple trees heuristics exploits the path diversity of leaf-spine topology. Every multicast group has multicast trees as many as number of spine-switches available in the topology. This

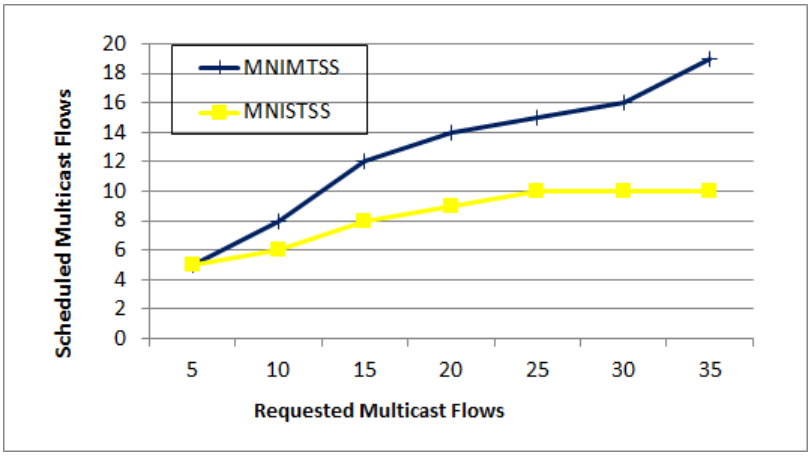



Figure 6.13: Requested vs Scheduled Flows for Multicast Routing and Scheduling algorithms

is also applicable to any other topologies as well as long as multiple trees exist for a multicast group.

### 6.4.2 Scalability of solutions

After measuring quality of scheduling algorithms, next we evaluated the scalability of the implemented algorithms. We discussed in section 6.2.2, the parameters which impacts on the run-time of the scheduling algorithms.

#### 6.4.2.1 Scalability evaluation of ILP solutions

1. Figure 6.14 shows the impact of varying number of flows on run-time. We measured the run-time of multicast scheduling problems with ILP based heuristics with leaf-spine topology of 6 switches (3 spine-switches, 2 leaf-switches per spine-switch) and 48 hosts, 5 time-slots and a varying number (5 - 40) of multicast flows. We observed that to schedule 40 multicast flows, MISTSS took 2.2 s. MIMTSS and MIUSS took on average 9 s and 5 s respectively to schedule 40 multicast flows.
2. To find out the impact of increasing number of available time-slots on run-time, we used a leaf-spine topology of 64 hosts and 8 switches (4 spine-switches, 2 leaf-switches per spine). We scheduled 50 multicast groups on this topology with a varying number time-slots between 20 and 50. In figure 6.15, we observe that single tree scales linearly with increasing number of time-slots. The run-time for multiple trees and unicast with same time-slots increases rapidly with the increasing number of time-slots.
3. Next, we evaluated the impact of varying topology size on run-time. We performed our experiments with 24 hosts and have placed 50 multicast flows. We varied our network size with varying number of network links from 100 - 400. Figure 6.16 shows that, single tree heuristics only takes on average 1.2 s to schedule one multicast flow. The

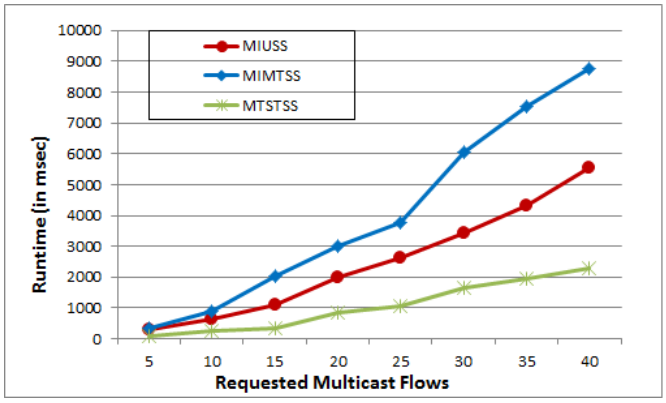



Figure 6.14: Runtime vs no of Multicast Flows for Multicast Scheduling with ILP based heuristics(MIMTSS, MISTSS, MIUSS)

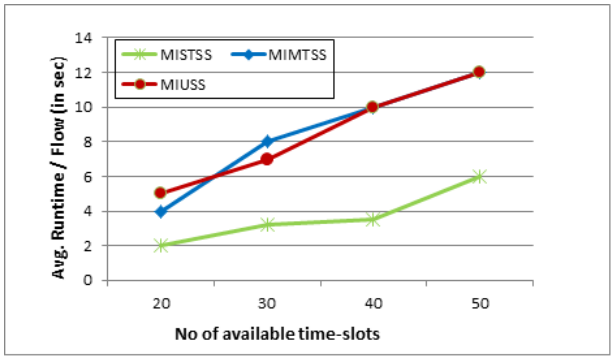



Figure 6.15: Runtime vs no of time-slots for Multicast Scheduling with ILP based heuristics(MIMTSS, MISTSS, MIUSS)

run-time for mutiple trees (MIMTSS) and unicast with same slot (MIUSS) increases almost rapidly and reaches to 7 s to schedule one multicast flow.

#### 6.4.2.2 Scalability evaluation of Routing and Scheduling algorithmic solutions

1. To evaluate the impact of number of time-slots on run-time, we extended our experiments with a topology of 240 hosts, 24 switches. We performed the experiments with 400 flows and 120 multicast flows. Figure 6.17 shows that, run-time increases linearly for the single tree heuristics with the increasing number of available time-slots in contrast to multiple trees. Run-time for multiple tree increase rapidly with increasing number of time-slots. For instance, with 500 time-slots single tree can schedule 1 flow in 800 ms and multiple trees needs 1.4 s. A LAN 10 Gbps links, 7 hops network diameter and 1500 bytes of MTU and with 1 ms base period can provide around 500 slots . Thus, to provide 500 slots with 1ms base period on a network requires all the links to be 10 Gbps.

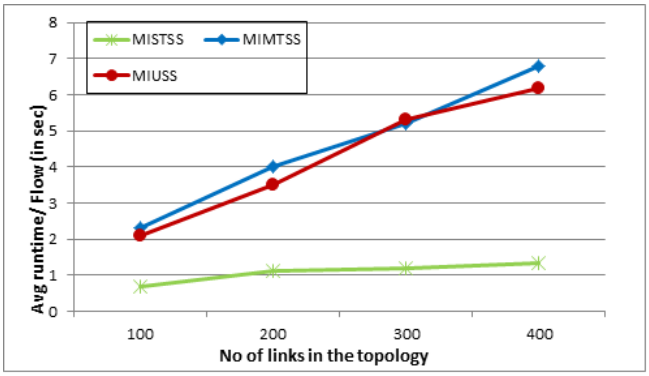



Figure 6.16: Runtime vs Topology size for Multicast Scheduling with ILP based heuristics(MIMTSS, MISTSS, MIUSS)

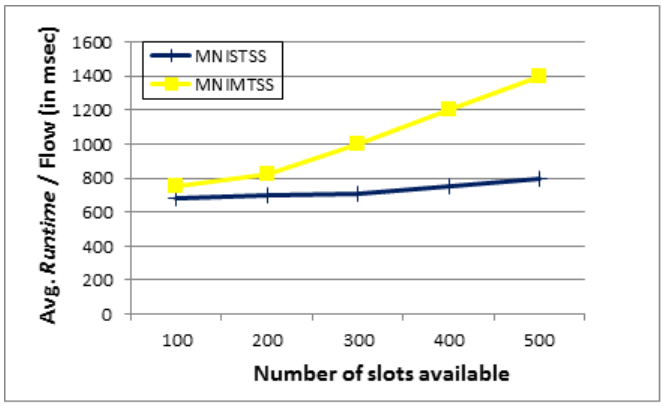



Figure 6.17: Runtime vs no of slots for Multicast Scheduling with Routing and Scheduling Algorithms (MNIMTSS,MNISTSS)

2. Figure 6.18 shows the run-time for varying number of links. We used topologies ranging from 200 - 500 links and scheduled 200 multicast flows. We observe that runtime for single tree increases linearly and it takes 2s to schedule a multicast flow. The run-time for multiple tree increases exponentially with the increasing number of links and takes maximum of 13 s to schedule a flow. To exploit the path diversity, we deliberately increased number of spine-switches in our topology. Our evaluation showed that, increasing number of multicast trees for a multicast group impacts on run-time rapidly.

3. To measure the impact of increasing number of flows on run-time, we used a topology of 32 switches and 160 hosts, 150 time-slots on the links. We deliberately placed large number of multicast flows to measure the scalability of non-ILP based approach. We varied the number of group communications between 20 - 100. Figure 6.19 shows that single tree can schedule 100 multicast flows in just 4 s where as multiple trees need around 7 s to schedule the same.

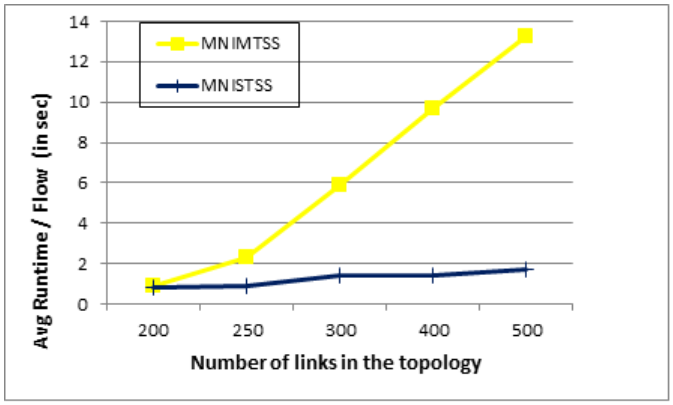



Figure 6.18: Runtime vs Topology size for Multicast Scheduling with Routing and Scheduling Algorithms (MNIMTSS,MNISTSS)

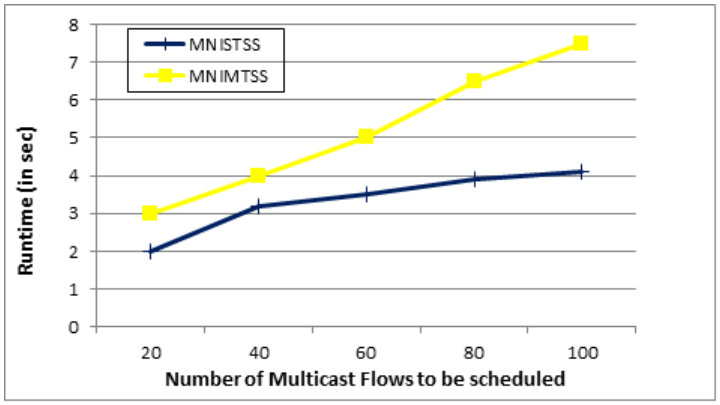



Figure 6.19: Runtime vs no of Multicast Flows for Multicast Scheduling with Routing and Scheduling Algorithms (MNIMTSS,MNISTSS)

## 6.5 Analysis of Multicast Evaluation

After evaluating all the algorithms for quality and scalability, we analysed the results of different approaches.

### 6.5.1 Quality Analysis

All of our evaluations showed that, ILPs with heuristics provide near optimal solutions and routing and scheduling algorithms compromise on quality of solutions. With a same topology and same number of time-slots, this algorithms schedule less number of flows than ILPs. Figure 6.20 shows that, ILP with MIMTSS and MIUSS can schedule 44 and 41 multicast flows

respectively, in compared with MNIMTSS algorithm which can schedule only 13 multicast flows.

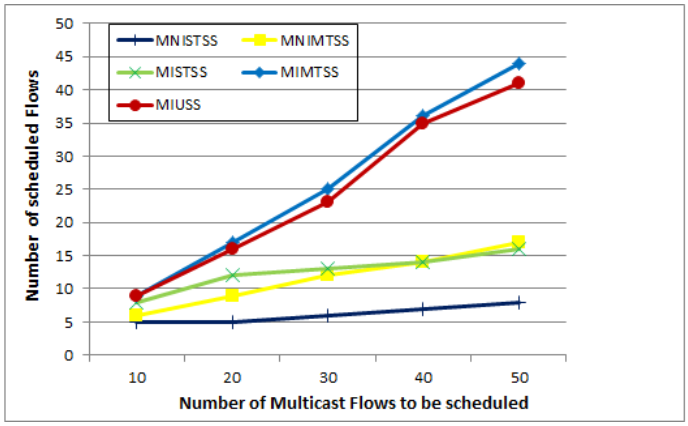



Figure 6.20: Quality Analysis of Multicast scheduling with ILP and non-ILP based heuristics

### 6.5.2 Scalability Analysis

1. After analysing the quality of ILP based heuristics and routing and scheduling algorithms for multicast, we next analyse their scalability. In the previous sections we measured the scalability of ILP based heuristics and routing and scheduling algorithms separately. Here, we analyse the results of those approaches. The figure in 6.21 shows that, scalability of routing and scheduling algorithms (with single tree and multiple trees approaches) scale much better than the ILPs. With a leaf-spine topology of 48 hosts and 40 multicast group communications, Multiple tree heuristic with ILP takes almost 9 s to schedule, whereas routing and scheduling algorithms with multiple tree takes on an average 1.2 s. ILP based unicast with same slot takes 5.5 s to schedule a multicast flow. Single tree with ILP and routing and scheduling algorithm takes 2.2 s and 1 s respectively.

2. Next, we analysed the impacts of increasing number of slots on run-time with respect to different approaches. We evaluated with a topology of 64 hosts and 8 switches. We placed 200 multicast flows and varied the number of time-slots between 20 - 50. Figure 6.22 shows that, routing and scheduling algorithms with both single tree and multiple tree approach, takes on average 1s to schedule one multicast flow. Heuristic multiple trees and unicast with same slot with ILP approach takes 12 s to schedule one multicast flow, single tree heuristics with ILP approach lies between the range of other two ILPs and algorithms. It takes around 6 s to schedule a multicast flow.

3. Lastly, we analysed the performance of heuristics with respect to the effect of varying number of network links on run-time. We measured the run-time of the solution with a topology ranging from 100 - 400 links and placed 50 multicast flows on them. Results of this evaluation is also at par with the previous evaluations. Figure 6.23 shows that

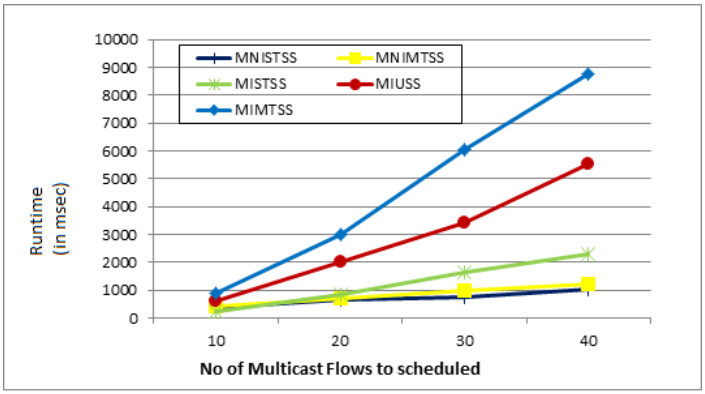



Figure 6.21: Runtime vs no of Multicast Flows for Multicast Scheduling with ILP and non-ILP based heuristics

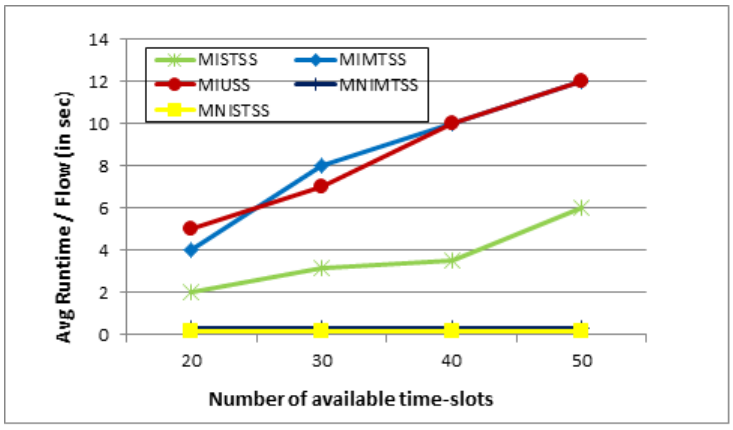



Figure 6.22: Runtime vs no of time-slots for Multicast Scheduling with ILP and non-ILP based heuristics

routing and scheduling algorithms with multiple trees takes only 2.5 s and with ILP based approach it takes almost 7 s.
In our experiments, we observed that in ILP based approaches, ILP creation time overshadows ILP solution time and ILP creation time increases rapidly with increasing number of flows, links and slots.

We can infer from the analysis of all the heuristics with ILP and routing and scheduling algorithms that ILP gives a better quality of solutions than routing and scheduling algorithms. Path-sets in unicast and multiple trees in multicast gave near optimal solutions and were able to schedule maximum number of flows on the network. But it took considerable amount of time to compute transmission schedules. On the other-hand, fixed path heuristics in unicast took much less time to generate near optimal solutions, but the number of flows it scheduled is considerably less. Routing and scheduling algorithms scaled much better than the ILP based heuristics approaches. They took on an average 2 s to schedule a multicast flows with 400

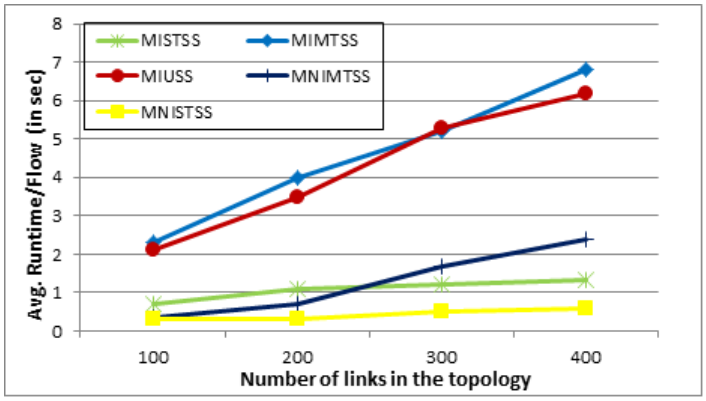



Figure 6.23: Runtime vs Tpology size for Multicast Scheduling with ILP and non-ILP based heuristics

links on the network, but could only place 16 multicast flows out of 50. Multiple trees with ILP approach scheduled 45 multicast flows in 9 s. Figure 6.22 showed that with 50 slots ILP with path-sets and unicast with same slot, it took 12 s to place one flow, in contrast to other approaches which scales much better. Thus, if the network has huge number of time-slots, for instance 500 slots, then non-ILP based approaches are better choice. It clearly showed that, ILP based heuristics give near optimal solution but it require more time to schedule, whereas algorithms gives solution in a very less time but it compromises on quality.In figure 6.24, we show the performance of all the implemented ILPs and algorithms with respect to quality and scalability. Now, it depends on the requirement of network administrators to choose the type of service they want to provide. If the network uses 1Gbps links and have large number of flows to be placed on it with a strict time constraint to meet, they can choose highly optimized Path-sets or Multiple trees approach using ILP. If the network has all 10 Gbps links or it has very less number of flows to scheduled, ILP with fixed path or algorithmic solutions could be chosen.

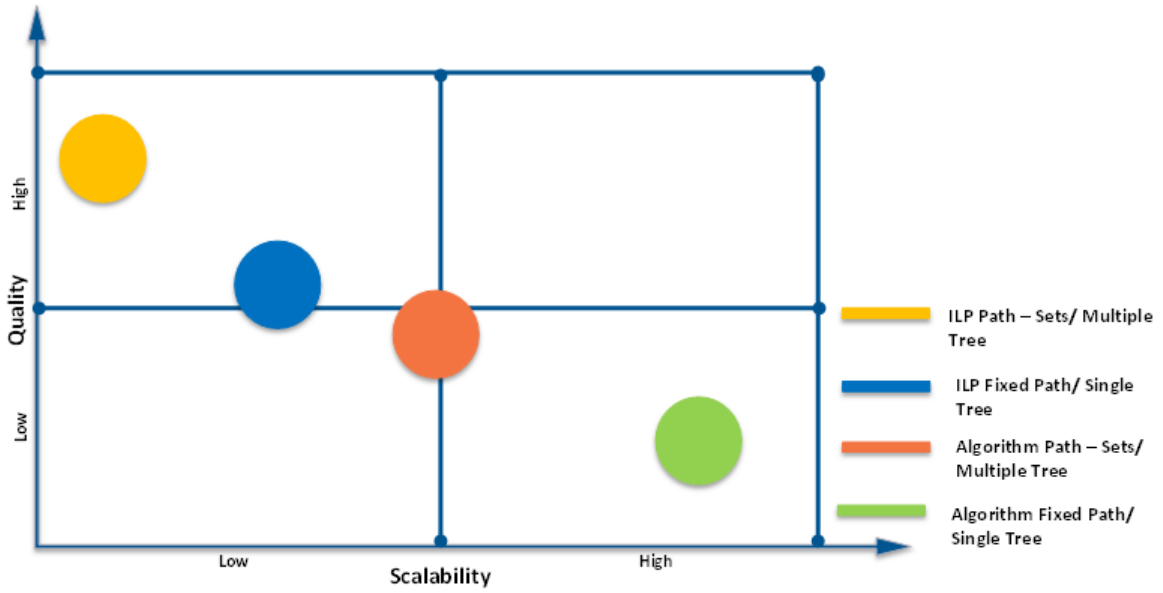



Figure 6.24: Quality vs Scalability

## 6.6 Evaluation Summary

In a nutshell, our evaluation showed :

1. The ILP based heuristics finds near optimal solution but exhibits higher computational time.
2. The second approach with routing and scheduling algorithms compromises on quality of solution and deviates from near optimal solution but it reduces the computational time.
3. It is a trade off between the scalability and quality as we have depicted in figure 6.24. Depending on the network parameters and requirements a suitable approach could be chosen.

# Chapter 7

# Conclusion and Future Work

To support real-time applications for example cyber-physical systems and applications from automation industries (Industry 4.0), deterministic networks with bounded delay and jitter are essential requirements. Furthermore, with the invasion of industry 4.0 in the automation and manufacturing industries, flexibility and scalability are highly required to support customization and variation of the products. To provide the implementation of such systems, we proposed time-sensitive software defined networks which uses a logically centralized SDN controller to compute optimized transmission schedules and routing paths for the time-sensitive flows. In this thesis, we designed and implemented set of routing and scheduling algorithms and integer linear programming with different heuristics for unicast and multicast communication. In our evaluations, we showed that it is possible to calculate near optimal solutions with ILPs using suitable heuristics. Routing and scheduling algorithms with heuristics are scalable enough to support large networks but with less optimal solutions. Moreover, with the available results from the experiments with DPDK, we inferred that it is possible to adhere to the transmission schedule with high precision. Thus, we can conclude that with efficient algorithms it is possible to eliminate the queueing delay in network and calculate near optimal transmission schedules.

In future work, we further improve the scalability of control plane with a distributed controller comprising of several physically distributed network hosts. Another, approach we are looking for is to partitioned scheduling, where we can divide the networks into smaller sizes. Thus, we can re-use the same slots in different network partitions at the same time for scheduling. But, new concept is required to develop this idea for handling flows which traverse across different partitions. Another interesting approach would be exploiting the single flow scheduling with different ILP based heuristics to achieve near optimal solutions.

# Bibliography


[1] Mohammad Al-Fares, Sivasankar Radhakrishnan, and Barath Raghavan. "Hedera: Dynamic Flow Scheduling for Data Center Networks." In: *Nsdi* (2010), p. 19. ISSN: 0723-8045. URL: http://dl.acm.org/citation.cfm?id=1855730$\backslash$nhttps://www.usenix.org/legacy/event/nsdi10/tech/full_papers/al-fares.pdf.

[2] *AVB*. http://controlgeek.net/blog/2015/1/9/a-new-direction-for-avb. [Online; accessed 23-November-2015]. 2015.

[3] Stephen Boyd and Lieven Vandenberghe. "Mathematical optimization techniques Slides". In: (). ISSN: 0160-5682. DOI: 10.1057/jors.1964.8. arXiv: 1111.6189v1.

[4] Neal Charbonneau and Vinod M. Vokkarane. "Static routing and wavelength assignment for multicast advance reservation in all-optical wavelength-routed WDM networks". In: *IEEE/ACM Transactions on Networking* 20.1 (2012), pp. 1–14. ISSN: 10636692. DOI: 10.1109/TNET.2011.2175007.

[5] *CPLEX OPTIMIZER*. http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/. [Online; accessed 10-November-2015]. 2015.

[6] Marco Di Natale et al. "Understanding and Using the Controller Area Network Communication Protocol: Theory and Practice". In: *inst. eecs. berkeley. edu/˜ ee249/fa08/Lectures/ ...* (2012). DOI: 10.1007/978-1-4614-0314-2. URL: http://www6.in.tum.de/pub/Main/TeachingWs2013MSE/CANbus.pdf.

[7] *draft-finn-detnet-architecture-01 - Deterministic Networking Architecture*.

[8] Frank Dürr and Thomas Kohler. "Comparing the Forwarding Latency of OpenFlow Hardware and Software Switches". In: 2014/04 (2014), p. 18. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=TR-2014-04\&engl=1.

[9] Real-time Ethernet. "IEEE TSN ( Time-Sensitive Networking ): A Deterministic Ethernet Standard". In: (2015), pp. 1–9.

[10] Network Fabric et al. "Cisco ' s Massively Scalable Data Center". In: (2010), pp. 1–6.

[11] José A Fonseca. "Real-Time Communications : from Fieldbuses and Industrial Automation to Wireless and Vehicular Applications Real-Time Communications R-T Comm : Role in Control & Autom . The communications infrastructure must be able to R-T Communications : Fieldbuses". In: (2012), pp. 1–24.

[12] B.A. Forouzan and S.C. Fegan. *Data Communications and Networking*. McGraw-Hill Forouzan networking series. McGraw-Hill Higher Education, 2007. ISBN: 9780072967753. URL: https://books.google.de/books?id=bwUNZvJbEeQC.

[13] Aric a. Hagberg, Daniel a. Schult, and Pieter J. Swart. "Exploring network structure, dynamics, and function using NetworkX". In: *Proceedings of the 7th Python in Science Conference (SciPy 2008)* SciPy (2008), pp. 11–15. ISSN: 1540-9295.

[14] Helbig (Deutsche Post Ag). Henning, Kagermann(National Academy of Science and Engineering). Wolfgang, Wahlster (German Research Center for Artificial Intelligence). Johannes. "Recommendations for implementing the strategic initiative INDUSTRIE 4.01". In: April (2013), p. 82.

[15] Hirschmann. "Precision Clock Synchronization". In: (), pp. 1–20.

[16] J. Huntington. *Show Networks and Control Systems: Formerly Control Systems for Live Entertainment*. Zircon Designs Press, 2012. ISBN: 9780615655901. URL: https://books.google.com/books?id=zLXLNT9oK5UC.

[17] M.D. Johas Teener et al. "Heterogeneous Networks for Audio and Video: Using IEEE 802.1 Audio Video Bridging". In: *Proceedings of the IEEE* 101.11 (2013), pp. 2339–2354. ISSN: 0018-9219. DOI: 10.1109/JPROC.2013.2275160.

[18] Ahmed E. Kamal and Raza Ul-Mustafa. "Multicast Traffic Grooming in WDM Networks". In: *SPIE 5285, Optical Networking and Communications* (2003), pp. 25–36. DOI: 10.1117/12.533163. URL: http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=836000.

[19] T. Koch and a. Martin. "Solving Steiner tree problems in graphs to optimality". In: *Networks* 32.3 (1998), pp. 207–232. ISSN: 0028-3045. DOI: 10.1002/(SICI)1097-0037(199810)32:3<207::AID-NET5>3.0.CO;2-O. URL: http://doi.wiley.com/10.1002/(SICI)1097-0037(199810)32:3<207::AID-NET5>3.0.CO;2-O.

[20] Diego Kreutz et al. "Software-Defined Networking: A Comprehensive Survey". In: (), pp. 1–61. ISSN: 0018-9219. DOI: 10.1109/JPROC.2014.2371999. arXiv: 1406.0440.

[21] Provides T H E Lifeline and O F Industrial Control. "Provides the lifeline of industrial control". In: ().

[22] Yang Liu et al. "Data Center Network Topologies: Current State-of-the-Art". In: *Data Center Networks*. Springer, 2013, pp. 7–14.

[23] Network Monitoring. "End-to-end Delay". In: ().

[24] Naresh Ganesh Nayak, D Frank, and Kurt Rothermel. "Software-defined Environment for Reconfigurable Manufacturing Systems". In: IoT (2015).

[25] Peter Neumann. "Communication in industrial automation—What is going on?" In: *Control Engineering Practice* 15.11 (2007). Special Issue on Manufacturing Plant Control: Challenges and IssuesINCOM 200411th {IFAC} INCOM'04 Symposium on Information Control Problems in Manufacturing, pp. 1332 –1347. ISSN: 0967-0661. DOI: http://dx.doi.org/10.1016/j.conengprac.2006.10.004. URL: http://www.sciencedirect.com/science/article/pii/S0967066106001742.

[26] Version Number. "IP Multicast Technology Overview". In: *Source* 1 (2002), pp. 3–26.

[27] "Overview and timing performance of IEEE 802.1AS". In: *2008 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication, ISPCS 2008, Proceedings* (2008), pp. 49–53. DOI: 10.1109/ISPCS.2008.4659212.

[28] *Packet Processing Framework with Intel DPDK*. http://dpdk.readthedocs.org/en/latest/sample_app_ug/netmap_compatibility.html. [Online; accessed 18-October-2015]. 2015.

[29] Paulo Pedreiras, Luís Almeida, and Paolo Gai. "The FTT-Ethernet Protocol: Merging Flexibility,Timeliness and Efficiency". In: *Proceedings of the 14th Euromicro Conference on Real-Time Systems*. ECRTS '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 152–. ISBN: 0-7695-1665-3. URL: http://dl.acm.org/citation.cfm?id=787256.787357.

[30] Paulo Pedreiras et al. "FTT-Ethernet: A flexible real-time communication protocol that supports dynamic QoS management on Ethernet-based systems". In: *IEEE Transactions on Industrial Informatics* 1.3 (2005), pp. 162–172. ISSN: 15513203. DOI: 10.1109/TII.2005.852068.

[31] Jonathan Perry et al. "Fastpass: A Centralized "Zero-queue" Datacenter Network". In: *SIGCOMM Comput. Commun. Rev.* 44.4 (Aug. 2014), pp. 307–318. ISSN: 0146-4833. DOI: 10.1145/2740070.2626309. URL: http://doi.acm.org/10.1145/2740070.2626309.

[32] Michal Pióro and Deepankar Medhi. *Routing, Flow, and Capacity Design in Communication and Computer Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004. ISBN: 0125571895.

[33] *PROFIBUS*. http://www.rtaautomation.com/technologies/profibus//. [Online; accessed 20-November-2015]. 2015.

[34] Luigi Rizzo. "Netmap: A Novel Framework for Fast Packet I/O". In: *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*. USENIX ATC'12. Boston, MA: USENIX Association, 2012, pp. 9–9. URL: http://dl.acm.org/citation.cfm?id=2342821.2342830.

[35] L.H. Sahasrabuddhe and B. Mukherjee. "Light trees: optical multicasting for improved performance in wavelength routed networks". In: *Communications Magazine, IEEE* 37.2 (1999), pp. 67–73. ISSN: 0163-6804. DOI: 10.1109/35.747251.

[36] R Santos et al. "On hierarchical server-based communication with switched Ethernet". In: *2010 IEEE 15th Conference on Emerging Technologies & Factory Automation (ETFA 2010)* (2010), pp. 1–4. DOI: 10.1109/ETFA.2010.5641073. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5641073.

[37] Till Steinbach, Franz Korf, and Thomas C. Schmidt. "Comparing time-triggered Ethernet with FlexRay: An evaluation of competing approaches to real-time for in-vehicle networks". In: *IEEE International Workshop on Factory Communication Systems - Proceedings, WFCS* (2010), pp. 199–202. DOI: 10.1109/WFCS.2010.5548606.

[38] *Survey of rump kernel network interfaces*. https://blog.netbsd.org/tnf/entry/survey_of_rump_kernel_network. [Online; accessed 20-October-2015]. 2015.

[39] "Survey on real-time communication via ethernet in industrial automation environments". In: *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)* (2014), pp. 1–8. DOI: 10.1109/ETFA.2014.7005074. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7005074.

[40] K. Tindell, a. Burns, and a.J. Wellings. "Calculating controller area network (can) message response times". In: *Control Engineering Practice* 3.8 (1995), pp. 1163–1169. ISSN: 09670661. DOI: 10.1016/0967-0661(95)00112-8.

[41] tttech. *Time-Scheduled Ethernet Standards*. https://www.tttech.com/technologies/deterministic-ethernet/. [Online; accessed 19-September-2015]. 2015.

[42] Bhanu Chandra Vattikonda et al. "Practical TDMA for Datacenter Ethernet". In: *Proceedings of the 7th ACM European Conference on Computer Systems*. EuroSys '12. Bern, Switzerland: ACM, 2012, pp. 225–238. ISBN: 978-1-4503-1223-3. DOI: 10.1145/2168836.2168859. URL: http://doi.acm.org/10.1145/2168836.2168859.

[43] Krzysztof Walkowiak, Róża Goścień, and Michał Woźniak. "Joint Optimization of Multicast and Unicast Flows in Elastic Optical Networks". In: (2015), pp. 6808–6813.

[44] Hui Zang, Janson P. Jue, and Biswanath Mukherjee. "A Review of Routing and Wavelenght Assignment Approaches for Wavelength Optical WDM Networks". In: August 13 (1999), pp. 1–25.

# Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Signature

Date, Place