Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master Thesis

Divide-and-Conquer Scheduling for Time-sensitive Networks

Bharat Bansal

Course of Study:	INFOTECH
Examiner:	Prof. Dr. Kurt Rothermal
Supervisor:	M.Sc. Naresh Nayak
Commenced:	1. September 2017
Completed:	1. March 2018
CR-Classification:	C.2.2

Abstract

The advent of the Internet of Things(IoT) and Industry 4.0 coupled with the increase in the deployment of Cyber-physical systems(CPS), which includes industrial automation systems, has increased the urgency to deploy networking technologies with real-time guarantees. The IEEE 802.1Obv standard has recently emerged as a viable alternative for the future of real-time communication over Ethernet that has stringent end-to-end latency and jitter requirements after it has been recently standardized by the Time Sensitive Network Task Group. Scheduling in Time-Sensitive Network(TSN) has not been standardized by the Task Group to carry the scheduled traffic till now.

In this paper, we address the scheduling problem for a Time-Sensitive Network(TSN) for large and complex networks. State-of-the-art scheduling algorithms take a centralized approach to compute transmission schedules for time-triggered traffic. Centralized approaches generally compute fairly optimum transmission schedules but the runtimes for the centralized approaches are high (order of days) for large and complex networks. The aim is to generate a scheduling algorithm having lower runtimes as compared to centralized approaches for large networks and having large time-triggered flows even if the computed scheduling has a worse optimum solution than the centralized algorithm.

We present a divide-and-conquer approach that might be apter in these scenarios, as it may generate a schedule with lower runtimes as compared to the centralized approaches. In particular, we present an Integer Linear Program(ILP) based formulation and a simple heuristics for the divide-and-conquer approach. Moreover, we parallelize the problem in order to further reduce the runtimes. We evaluate the optimality, utilization, scalability of the different approaches. Furthermore, we evaluate the runtimes for different network configurations and discuss the trade-off between heuristics and ILP based formulation.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

Cyber-physical systems (CPS) are at the heart of the automation technologies used in various industrial manufacturing. Over the years, progress has been made in CPS technology and CPS applications can be found in the design of next-generation planes, space rockets, electric cars and fully autonomous driving. The advancements done in CPS systems today will enable more capability, adaptability, security, scalability, resiliency than the current embedded systems. Cyber-physical systems(CPS) are the seamless integration of networking, physical and computational processes and it combines various concepts from embedded, distributed, networks, software and hardware systems. CPS connects different components such as actuators, sensors and microcontrollers and other devices to other physical systems and use distributed systems and networking to control the interlinked environment.It is used widely as it is a low cost and high-performance technology and thus emerged to play a vital role in the Internet of Things(IoT). The distribution and heterogeneity of the devices make it quite difficult to communicate in an efficient and reliable manner. So different networks and communication models are needed to solve the time stringent problems[9]. Being highly time-sensitive, CPS systems demand stringent real-time communication guarantees with low bounded network delays and very low jitter from the underlying network.

Deterministic networking is same as normal networking but with these following features:

1. Time synchronization for network nodes and hosts.
2. Resource reservation for critical data streams via management, configuration and protocol action.
3. Software and hardware needed for extremely low packet loss ratios and guaranteed end-to-end latency for a reserved flow.
4. QoS and convergence needed on a network even for a high bandwidth data streams.

Although a lot of effort and research has been put on various issues of CPS not much in the networking and communication aspect of CPS. Typically, dedicated field-bus networks such as Ethernet are used in such systems. However, these different technologies can't be operated on the same medium without loss of real-time guarantees.To cope up with differences, IEEE has set up standardized real-time extensions for Ethernet. Also with the emergence of the Internet of Things(IoT) and Industry 4.0, Ethernet has emerged as one the major transport medium for the converged network needed in CPS.

Ethernet is still widely used in the communication and networking world after more than 40 years. It was initially developed for the local-area network(LAN) but it is now used throughout in wide-area networks(WANs) which carry the world's Internet, telecoms, and wireless traffic. The demand for network capability has increased in last few decades because of large adoption of IP-based(Internet protocol) infrastructure in an accelerated manner and increase in demand for cloud-based services [11]. Ethernet is one of the most cost-effective and scalable ways to support the mobile and cloud-based services. It is based on best-effort communication and minimizing delays and maximizing the throughput are the main objectives. But Ethernet is non-deterministic in nature and doesn't provide bounded latency for the transmitted data in the network. In real-time communication, the need for the deterministic and time bounded network are the primary objectives for safety-critical purposes. These are some of the limitations that restrict the use of Ethernet as a communication medium in CPS.

Ethernet has evolved to provide the industry requirements for real-time purposes by means of technologies such as TTEthernet, PROFINET, and EtherCAT [16]. Also, the advancement in automotive domains has led to the standardization of solutions in IEEE 802 networks to provide for deterministic networks. But the investment is high to make them compatible with other proprietary technologies and are not future proof if better technologies are adopted in future. In IEEE 802.1BA, Quality of Service has been introduced to Audio and Video Bridging (AVB) as latency and jitter are important and buffers are scared at these speeds. But it still fell short of real-time applications demands and fully-deterministic demands. It needs some mechanisms to allow for low latency and strict timing guarantees to support the real-time applications on Ethernet.

The above drawbacks are mostly addressed in IEEE 802.1 Time Sensitive Networking(TSN). Since existing networks are defined by 802 standards and thus TSN was formed after various Industrial suppliers and vendors come together to ensure interoperability between devices. Time-sensitive Networking is a set of IEEE 802 Ethernet sub-standards that are defined by the IEEE TSN task group. These standards enable deterministic real-time communication over standard Ethernet and it delivers guarantees of delivery and minimized jitter. The IEEE 802 works on OSI Layer 2 and thus Ethernet frames and are not limited to Internet Protocol(IP) . Deterministic communication is important for industrial applications and thus it provides cost savings and new levels of connectivity and optimization over standard Ethernet.

The major benefits for TSN in the current industrial networking are :

1. **Convergence**: Time-triggered and best-effort traffic can be converged together in one network without any risk of the impact of the delivery of packets from each other.
2. **Security**: TSN provides top-tier security solutions, segmentation performance to IT industries and can add multiple levels of defense in a network.
3. **Interoperability**: TSN can integrate with existing applications and standard IT traffic as it is based on standard Ethernet. TSN inherits many features from Ethernet, such as HTTP interfaces and web services.
4. **Latency and Synchronization**: TSN provides low-latency synchronization and it is prioritized over fast system applications. It provides deterministic solutions in order of tens of microseconds and time synchronization in order of tens of nanoseconds. TSN provides reliable delivery of scheduled time critical-traffic and automated configurations for high - reliability data paths to provide lossless path redundancy.

TSN provides the above benefits by various aspects of clock synchronization and frame preemption to schedule traffic management. TSN is centrally managed and the determinism is achieved by clock synchronization with bounded precision, known and bounded network latencies over shared network devices. It uses time-aware traffic shaping and it helps to control logical gating on the switches.The best-effort traffic is blocked at egress ports during specific time windows when the time-sensitive traffic is expected.

Table 1.1: TSN can be categorized into below set of protocols

IEEE standard	Features / area of interest
802.1	Bridging and networking management
802.1 AS	Timing and synchronization
802.1ASbt	Timing and Synchronization: Enhancements and Performance Improvements
802.1p	Traffic class expedition and priorities
802.1Qbu	Frame Preemption, a mechanism that allows to preempt a frame in transition.
802.1Qca	Path Control and Reservation: protocols and mechanisms to set up and manage redundant communication paths in the network.
802.1Qbv	Enhancements for scheduled Traffic: a basic form of time-triggered communication.
802.1Qcc	enhancements and improvements for stream reservation.

The implementation of 802.1AS in all devices is required if there is a need to implement the synchronization between clocks. It achieves a global network clock and fault-tolerance by means of re-election mechanism of the master clock in case of failures. The communication takes place via time-sensitive flows where individual messages are defined by attaching a VLAN identifier(VID) and the data streams flow from sender to multiple receivers. Flows have different traffic classes based on priorities which are given by 3 priority bits of the priority code (PCP) defined in IEEE 802.1Q header. IEEE 802.1Qbv is a time-aware shaper enabling time-deterministic communication at egress ports.

Scheduling in Time-Sensitive Network(TSN) has not been standardized by the Time sensitive Networking Task Group to carry the scheduled traffic till now, but the hardware related to TSN enforces the scheduling such as traffic stemming from CPS. Some of the enhancements primarily consist of a programmable gating mechanism that connects/disconnect the queues of the egress ports from output link. Every port has a logical composition on egress with a number of queues buffering the respective frames till all transmission is completed. These events allow for the control over the transmission medium by enabling or disabling the transmission of frames according to a predefined schedule of flows. Some of the ongoing research going to schedule the flows transmission by the switches are:

- Scheduling Real-Time Communication in IEEE 802.1 Qbv Time Sensitive Networks developed by Craciunas et al. at TTTech Computertechnik AG. It presents a method for computing static schedules for 802.1Qbv-capable network devices using Satisfiability Modulo Theories(SMT) solvers [6].
- No-wait Packet Scheduling for IEEE Time-sensitive Networks(TSN) developed by Duerr et al. at University of Stuttgart. In it, the scheduling has been solved using No-wait Job Scheduling Algorithm [7].

The scheduling approach in paper IEEE 802.1 Qbv Time Sensitive Networks is to schedule one flow at a time by adding flow variables to the SMT context and thus solving with the added constraints. The schedule for a particular flow is fixed if a solution is found by asserting the value provided by SMT solver and it continues till the complete schedule is found. Thus it needs to know the whole topology beforehand to calculate the complete schedule.

In No-wait Packet scheduling for IEEE TSN paper, it has been discussed that generally, TSN switches use guard bands to isolate the best-effort traffic from scheduled traffic. Guard bands are created by closing the gates of best-effort traffic before the start of the scheduled traffic which leads to wastage of bandwidth for best-effort traffic. So, the guard band size can be reduced by sending lots of scheduled traffic by having lower gate openings which result in lower makespan.

The scheduling of flow events by all the above methods is basically a centralized approach as it takes into account of the entire topology to calculate the transmission schedules of the time-triggered flows in the network. While centralized approaches are effective in finding optimal solutions but they have high runtimes to calculate the flow schedules making them unsuitable for deployment in scenarios with large topologies and several time-triggered flows.

A divide-and-conquer approach might be apter in these scenarios where we can trade optimality in exchange for lower runtimes. The divide-and-conquer paradigm breaks a problem into subproblems that are similar to the original problem. As it solves the problem recursively and each subproblem should have a base and smaller than the original problem. Divide-and-Conquer solve a problem in 3 parts [13]:

1. **Divide** the problem into further subproblems that are smaller in nature of the same problem.
2. **Conquer** the subproblems recursively and solve them as a base problem if they are small enough.
3. **Combine** the solutions to the subproblems into the solution for the original problem appropriately.

Some of the advantages of divide-and-conquer are:

- Solving difficult problems: It can solve a conceptually difficult problem as it breaks the problem into sub-problems.
- Efficiency: Divide-and-Conquer paradigm helps in finding new efficient algorithms such as quicksort and mergesort algorithms.
- Parallelism: Divide-and-Conquer algorithms can be executed on multi-processor machines especially if they don't share the same memory.

Some of the standard algorithms in Divide-and-Conquer algorithms are:

1. **Binary Search** is a searching algorithm where we initially check if the search element matches the middle element in the array. If it matches, returns the index of the middle element. If the value of the element is less than the middle element, the algorithms recur to the left side other it recurs to the right side of the middle element. The time complexity of binary search is O(log n) [1], which is less than the linear search complexity of O(n).
2. **Quicksort** is a sorting algorithm. It selects a pivot element and then rearranges all elements which are smaller than pivot to the left of the pivot and all the larger elements to the right of pivot element. Then the algorithm recursively solves the

subarray on both the left and right of pivot element. The average time complexity of Quicksort is O(nlogn).

3. **MergeSort** is also a sorting algorithm. It divides the array into 2 parts and then sort it recursively and then merge the sub-arrays back to the main array. The average time complexity of MergeSort is O(nlogn).

We can use Divide-and-Conquer to schedule the flows. First, we have to divide the problem, so we basically divide the network into sub-networks by choosing a link which acts as a pivot element. Solve the scheduling problem of flows on that particular link and can also implement parallelization by solving the scheduling problem on another link which doesn't have any common flows but on a different server. Then we divide the network recursively till we have reached all the links. Thus, we can calculate the schedules in lesser time as compared to centralized approach even if we suffer in terms of optimal solutions. The aim of the thesis is to develop a scheduling algorithm using divide-and-conquer approach. Initially, we have used Integer Linear Programming(ILP) to find the optimal schedule on a link. As the runtime of ILP algorithm is quite high as with the increase in the number of flows, the time to calculate the solution increases exponentially. So, scalability of the algorithm steeply increases with the increase in topology. Thus, we further apply heuristics instead of ILP to have lower time complexity. Heuristics are basically any method or approach to problem-solving that employs an approximate method not guaranteed to be optimal, but sufficient to find the immediate goals.

The outline of my thesis follows:

1. Develop and implement a scheduling concept based on divide-and-conquer methodology for computing transmission schedules.
2. Design a heuristic approach for effectively dividing the scheduling problem with respect to parallelization of the problem, optimality of the solution etc.
3. Comprehensive evaluation of the approaches with respect to the optimality loss, runtimes, parallelization degree etc.
4. Complexity analysis of the developed scheduling approaches.

1.1 Thesis Organization

The master thesis is structured as follows

Chapter 2 – Background: presents the fundamental concepts of the thesis. It starts with an overview of Time-sensitive Networks(TSN) followed by a discussion on IEEE 802.1Qbv standard. It also gives an overview of the related scheduling methods and the heuristics algorithm used in the thesis.

Chapter 3 – System Model and Problem Statement presents the specification of the system model and the problem statement of our thesis.

Chapter 4 – Scheduling Algorithms presents the various scheduling algorithms that have been implemented in the thesis.

Chapter 5 – Results and Evaluations presents the results of the evaluations of the scheduling algorithms presented in the thesis.

Chapter 6 – Conclusion provides the conclusion with summary and possible future work.

2 Background

In this chapter, we will discuss the various terms that are necessary to better understand the concepts included in the thesis, We take a look at key Time-Sensitive Networking standards and scheduling methods.

2.1 TSN Foundation

Today, we need high precision timing systems for real-time and automation applications. "Best effort" traffic works in lightly loaded networks and the average delay is the primary metric to measure the quality. But "best effort" is not good for TCP based traffic and ignores the problem of audio and video dropouts. There is a high demand to make IP based traffic networks to 802.1 scheduled traffic for emerging markets. Reliability, redundancy, and safety are critical for the control applications and have the need for low latency, low jitter, and guaranteed bandwidth.These applications needed a converged network capable to provide efficient utilization of the bandwidth. These are driven by IEEE 802.1 Time Sensitive Networking Group, which aims for time-sensitive and deterministic low latency problems.

TSN can be better understood with the following example. Different data packets have to be distinguished from each other if they have different priorities. Each node in the network knows about the priority for a packet to be transmitted in the network. For this purpose, standards IEEE 802.1Q was introduced to tag data frames with the priority field.

TSN is not a protocol but a Layer 2 networking technology. TSN is not another form of quality of service(*QoS) and the devices have to support TSN in the hardware and it is not a software-defined technology.[4] It defines how the Ethernet devices can transmit end to end according to a schedule. Nondeterministic data(non-TSN) data will continue to flow in the system when there is no TSN data in the network.

Time-sensitive networking(TSN) was created by IEEE working group that focuses on deterministic networking over Ethernet.The TSN standard helps to deliver real-time applications, from streaming multimedia to synchronization power micro grids. The group was originally known as Audio/Visual Bridging(AVB) task group, define extensions to 802.1Q virtual LAN(VLAN) standard and it is defined in 802.1BA standard. AVB is very useful to consumer electronics, professional video, and automotive infotainment.

2.2 Different TSN protocols

2.2.1 Audio-Visual Bridging

As we have discussed above that TSN have a foundation of Audio-Visual Bridging(AVB). The main objective of AVB was to create an ad-hoc network in which users can request paths such that the bandwidth is allocated as required with guaranteed low jitter and well-bounded latency. AVB objectives are developed under these standards :

- 802.1BA: Audio Video Bridging
- 802.1AS: Timing and Synchronization for Time-Sensitive Applications.
- 802.1Qat: Stream and Reservation Protocol(SRP), and
- 802.1Qav: Forwarding and Queuing for Time-Sensitive Streams(FQTSS).

Structurally, an AVB is created as an Ethernet network with switches as the main core element to distribute the data but AVB switches are called bridges to distinguish it from Ethernet switches. It can be connected to any other Ethernet network even if it doesn't have AVB bridges. The endpoints and the bridges formed the AVB cloud. The maximum latency can be 2ms in AVB cloud such that the maximum hops are 7 between the start and end device[3].

The flow is established through these steps. First, the user will request for the bandwidth by taking into account jitter and latency requirements using the protocol SRP. Then the latency and jitter is calculated by going hop to hop on all bridges by passing the request and reserving the required bandwidth. The bridges will transfer the packets once the flow is established by using the "Credit-Based Shaper" defined in 802.1Qav.

AS more progress and development was made in AVB, it was found out that Credit Based Shaper is not as efficient as previously thought and there will be serious packet losses in case of traffic congestion. It is a serious issue in real-time applications and it has been tackled in further standards. A new shaping method is needed in place of Credit-Based Shaper to eliminate congestion loss. It was renamed as Time-sensitive Networking(TSN) in 2012 as to cope with the changes in market standards.

2.2.2 IEEE 802.1ASrev

The objective of IEEE 802.1Aserv is to achieve clock synchronization for a deterministic bounded network. It is done by calculating all the global delays from sender to receiver in a network and thus the scheduling of traffic queues can be achieved through each network component. A profile for IEEE 1588 PTP synchronization was created to enable clock synchronization between different TSN devices.

802.1Aserv also supports multiple active synchronization masters such that network supports a working clock, which is used for time-critical events and a universal clock for time-stamp events concurrently. It also has the feature for fault tolerance to have multiple replications of masters in case any one of the master clock becomes faulty.

2.2.3 IEEE 802.1CB

IEEE 802.1CB implements a redundancy mechanism which is similar to PRP(Parallel Redundancy Protocol). IT is done by sending multiple copies of the same messages in parallel over disjoint paths.At the receiver side, the redundant messages are combined to form a single stream of message. The task group is still working to standardize the method but the chances are that it will be based on sequence numbers. The sequence numbers will be inserted in a tag field similar to VLAN ID in Ethernet frame. It helps in removing the duplicate packets if the same packet reaches twice as it has the same sequence number but doesn't guarantee in-order arrival of packets.

2.2.4 IEEE 802.1Qcc

The Stream Reservation Protocol(SRP) has been improved in this protocol as to meet more stringent requirements of industrial systems. This has been achieved by supporting more streams, configurable and better description of streams characteristics, support for Layer 3 streaming, User Network Interface(UNI) for routing and reservations and convergence of deterministic stream reservations.

2.3 IEEE 802.1Qbv

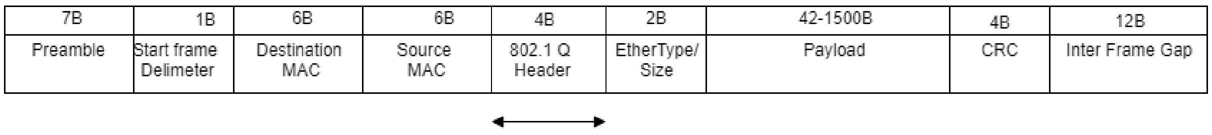
Time-sensitive Networking is basically a time-triggered principle at its core. Credit-Based Traffic Shaping (CPS) was introduced for Audio/video streams as it reduces the bursting and bunching by spacing out the frames as much as possible. CPS protects the best

-effort traffic of Audio-Visual Bridging(AVB) streams by limiting back-to-back stream bursts. But in industrial networks, there is a need for real-time applications to have a very low latency and thus there is a need to schedule the high priority traffic in the network.

Time Aware Traffic Shaping has been introduced to schedule the flows in a network. We will discuss in detail the 802.1Q data frame tagging for prioritizing data and tagging it and Time Aware Shaping protocol.

2.3.1 Data Frame tagging

The standards allow for a 16-bit header attached to an Ethernet frame consisting of 12-bits Virtual LAN(VLAN) ID, 3 bits for priority and 1 bit for drop eligible indicator. Laine [10] It has been defined in 802.1Q header version. The field defines the priority of the packet but it still doesn't fit the deterministic time requirements.



7B	1B	6B	6B	4B	2B	42-1500B	4B	12B
Preamble	Start frame Delimeter	Destination MAC	Source MAC	802.1 Q Header	EtherType/ Size	Payload	CRC	Inter Frame Gap

Figure 2.1: Insertion of 802.1Q header in Ethernet frame

The header will get deleted if there is an 802.1Q unsupported switch in the network and it is also backward compatible but will lose some extra features. In a VLAN tagged network, data from different VLANs can be distinguished even if the data streams are between 2 same endpoints. These results in multiple virtual networks within the same physical network. Even if we can define different priorities through VLAN tag but it still the protocol is not time critical and can be used for real-time applications.

2.3.2 Prioritization

The prioritization is determined by the traffic types and the data is used for in 802.1Q. There are different priority levels depending on the traffic types. It can have a maximum of eight priorities as the tag is of the 3-bit field. They are shown in the table below:

Table 2.1: Priority Levels

Priority	Traffic Types
0(lowest)	Background

1	Best Effort
2	Excellent effort
3	Critical applications
4	Video less than 100 ms latency and jitter
5	Voice less than 10 ms latency and jitter .
6	Internetwork Control
7(highest)	Network Control .

Priority can be adjusted if there are less than eight traffic classes as:

Table 2.2: My caption

	Traffic classes							
Priority	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
0	0	0	0	0	0	1	1	1
1	0	0	0	0	0	0	0	0
2	0	0	0	1	1	2	2	2
3	0	0	0	1	1	2	2	2
4	0	1	1	2	2	3	4	4
5	0	1	1	2	2	3	4	5
6	0	1	2	3	3	4	5	6
7	0	1	2	3	4	5	6	7

Prioritization of data transmission is done on the egress port. A mechanism is needed to ensure that there is coordination between different data streams. A protection window is needed to ensure that high priority data has guaranteed access to the network at a specific instant in time. It can be created by allowing the transmission of a specific traffic class over a specific period of time. There are total 8 queues one for each traffic class to accommodate different traffic classes. Transmission is decided on the basis of algorithm known as Transmission Selection Algorithm(TSA). If the data from a particular queue is selected then the corresponding gate will be opened to transmit the data. Gate opening events are determined by gate control list. If there are fewer queues than eight, than data having multiple priorities can reside in a single queue. TSA decides in which order the data has to be transmitted from the queue. It has been shown in figure 2.2. The gate following can block the transmission even though TSA has decided before the next opening gate event. This implies that we can give higher priority than the highest priority traffic type. The transmission algorithm can be selected on the basis First in First

Out(FIFO) or it can be credit-based shaper(CBS) or vendor specific algorithm. So it is up to the industry to define the gate control list and how gates are operated. Gates control list is not needed if there are no enhancements to be made for traffic scheduling.

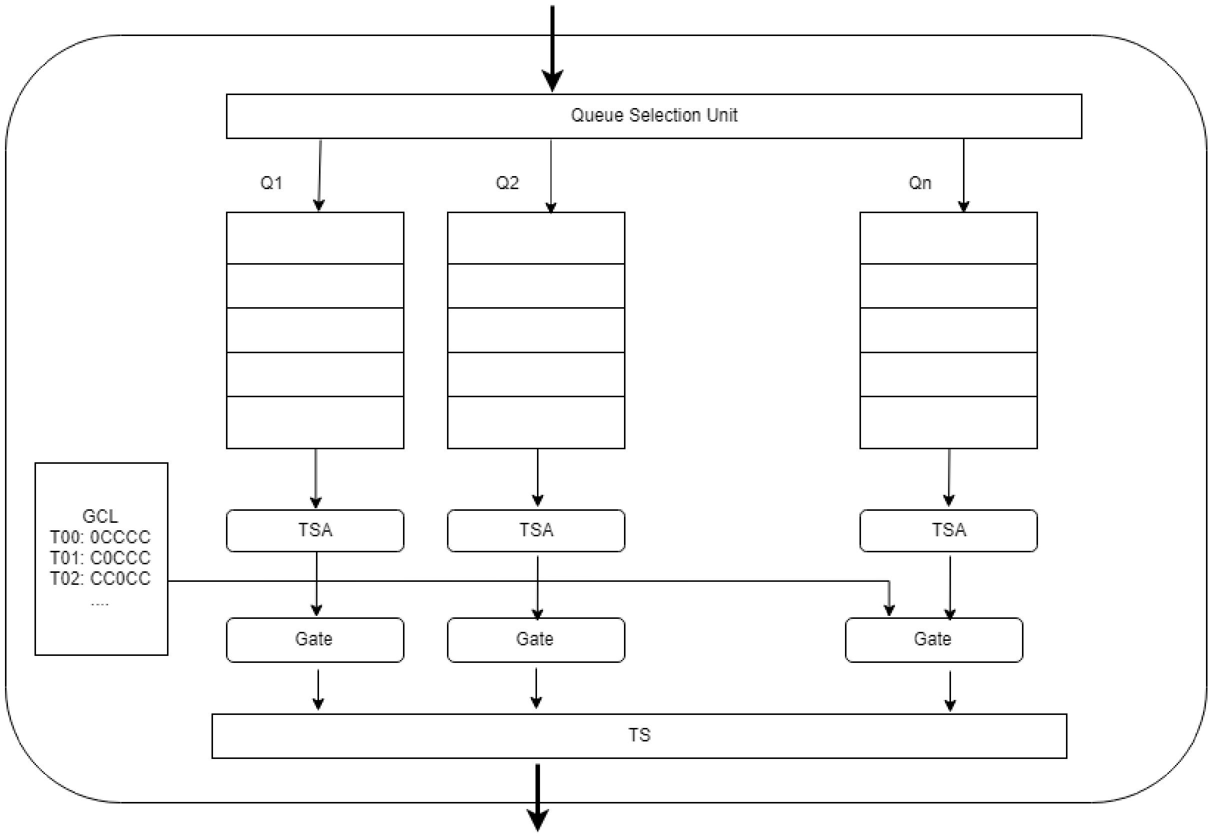


Figure 2.2: Working of TSN on outbound port

2.3.3 Time-Aware Shaper

Network Architecture consists of network switches(NS) and end systems(ES). End systems are the devices which only send or receive the data from other endpoints. Network switches are only used to transmit the data to other network switch or end systems but don't consume, create or alter the packet itself. Each switch has multiple outbound ports which are connected to other devices. Each outbound port has multiple queues and each port is only connected to only one node at the other end. There can be multiple ports connected to a node and the traffic from outbound and inbound ports doesn't necessarily have to match at each node. Each node has an inbuilt internal clock to synchronize the gate timings for TSN to work. The prioritization is done at the outbound port and according to it the packets are stored in different queues.

Time Sensitive Networking has the need for deterministic data delivery and thus employ prioritization to deliver the data with the ability to predict the latency and jitter during the transmission. In real time industrial applications, data can be transmitted in a repeated cycle. But prioritization of data can't determine the control of the scheduled traffic because if data of higher priority can't be transmitted at a scheduled time if the transmission of lower priority data is going on. Higher priority can be transmitted after only lower priority data has finished.

The gate control list needs some type of protocol to synchronize the clocks between different hardware nodes. It can be done by using precision time protocol(PTP) with very high accuracy and synchronization of time between nodes. This makes it possible to transmit low latency data with the correct opening of gates as the timing of the openings can be synchronized now. For the deterministic nature of the system, the data transmission times should be known in advanced in a time-sensitive network. Gate control list is used to determine the order for TSN and this method is known as protected windows.

In protected windows, data transmission of the previous gate has to be stopped before the new data transmission. For this purpose, guard bands are needed to be put between two gate opening events.The gates are reversed after the end of the protected window as they were before the opening of the protected window. Guard bands are essential that best-effort traffic is not transmitted between two protected windows. AS can be shown in figure 2.3, the start of guard band(T0) is before the protected window (T1). The size of the guard band is determined by the difference between T1 and T0 (T1 - T0) and for worst case, it must be equal to maximum frame size and it has to end till the end of the protected window(T2) Laine [10]. It will lead to wastage of bandwidth as some of the packets have to wait for the end of the protected window even though if they can start before the protected window.

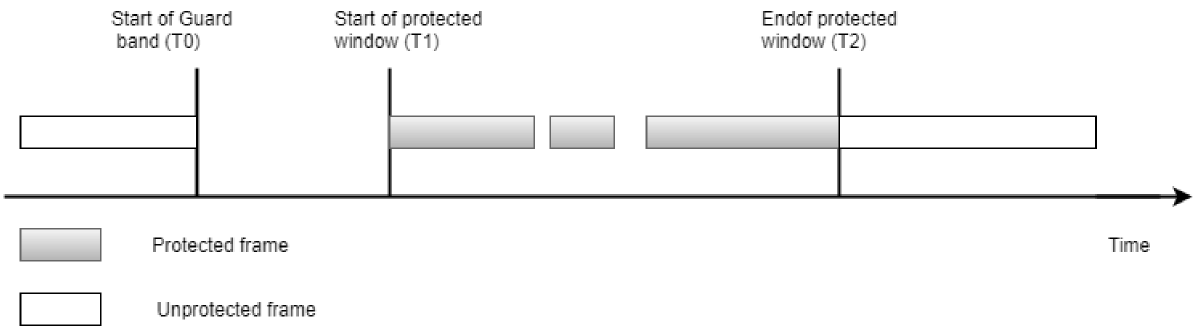


Figure 2.3: Protected window using gates

2.4 Related Work

In unsynchronized networks, it is difficult to know the scheduling as the configuration tools have to do complex calculations to know the worst-case latencies, jitter and buffer requirements in the switches. Scheduling for complex and large TSN networks doesn't suffer from complex calculations which have to be done in unsynchronized networks as the transmission times of messages in switches and nodes can be aligned to each other with respect to a network-wide synchronized clock. Computation of transmission schedule in a TSN network is calculated on the basis of the time-triggered flows and their routes. The schedule that is being calculated for each flow should guarantee minimum latency such that queuing delays are minimized. We will discuss the following two methods that solve the problem of transmission schedule in a time-triggered network.

2.4.1 Scheduling Real-Time Communication in IEEE 802.1Qbv Time Sensitive Networks

It was developed by Craciunas et al. at TTTech Computertechnik AG.They want to solve the problem of generating transmission schedule for a time-triggered network using Satisfiability Modulo Theories(SMT) [6].

They identified two key parameters which determine the timing properties of a network. First is the capability of devices, if they can schedule the flows or not. If the flows are not scheduled at the end nodes, then they arrive at the first switch in any order. The first switch then has to act as synchronization gate for that flow. It can be achieved only if there are enough queues to schedule all the flows arriving simultaneously deterministically. Second is the Queue Configuration, which determines how the queues operate. The number of queues per egress port determines different priorities a port can handle. The gates on egress port can be opened purely on the basis of the priorities or queues can be scheduled on a time-triggered(TT) paradigm where priority is overruled by the schedule. The different types of queues mean that both scheduled and non-scheduled traffic can be sent together in a network.

They define a few scheduling constraints in order to compute a schedule of the queues for the timed-gates. They divided it into two parts, one is basic constraints based on deterministic Ethernet and the second is based on 802.1Qbv specific constraints.

The Ethernet specific constraints are:

- Frame constraint: It says that the frame offset of any scheduled flow must be equal or greater than 0. Also the entire transmission window has to fit within the frame period.

- Link Constraint: No two frames can overlap if they are routed on the same physical link in the time domain. 'item Flow Transmission Constraint: The propagation of the flows must follow the same order along the routed path of the flow. The constraint is set to individual frames rather than the complete flow instance as to allow the forwarding after the first frame has been fully received without waiting for the entire flow to buffer.
- End-to-end Constraint: It specifies that the difference between the arrival and sending time of a flow has to be within the specified maximum.

The 802.1Qbv constraints are:

- **Egress Interleaving**: It is to guarantee that the end-to-end latency is always fulfilled. The order in which the frames are placed in the scheduled queue is non-deterministic as schedule controls the gate opening events not the order of frames in the queue. This is due to some amount of synchronization error between different devices which may result in frames arriving not in order during runtime. It will result in accumulation of jitter for the overall end-to-end synchronization. This is not desirable when it accumulates with each hop along the flow path in hard real-time systems. This can be solved by guaranteeing the isolation of frames in the transmitting queues by a set of constraints. It can be done either by placing flows in different queues or the constraints must enforce some deterministic order of the flows in queues.

  They assume an ideal scenario in which there is no frame loss and all flows are sent constantly in full size. It was sufficient that if any two individual frames of different flows are not scheduled to arrive at the same time to get a deterministic schedule.
- **Flow Isolation Constraint.** In reality, it is not possible to achieve the ideal scenario due to frame losses or varying size of payloads over time.In such cases, deterministic behavior can't be guaranteed as it can be seen with an example. Consider two frames are scheduled to arrive one after another from two different flows and placed accordingly in a queue at the switch. If one frame is lost than the other frame will take its place in the queue which can be transmitted in the original first frame time slot which leads to non-deterministic behavior. The constraint to enforce correct ordering of flows is that no frame is allowed to enter the queue until all the frames from the previous flows have been fully dispatched.
- **Frame Isolation Constraint.** The Flow Isolation Constraint is generally faster but it's restrictive and may decrease the solution search space for valid schedules. In order to avoid this, the researchers relax the constraints to allow frames interleaving between flows in queues while same time guaranteeing the order on the egress port is deterministic. This can be achieved by allowing only frames of one flow in

the queue at a time. If two frames from different flows have arrived than one can be allowed to enter in the shared queue if the other has already been dispatched from the queue.

The main objective of this method is to find individual frame offset values at each egress port of flows routed along the network such that the above set of constraints are met.The frame offset and queue assignment are used to generate the corresponding constraints defined in the previous section. They are inputs to an SMT solver which determines the satisfiability of the given set of constraints and generates a satisfiable solution. Their approach to scalability in scheduling problems is to introduce an incremental backtracking algorithm. If the complete schedule can't be found while adding the constraints to SMT solver than the incremental step is deemed infeasible by the solver. The SMT context is backtracked by removing the last scheduled flow and reintroducing it together with the unfeasible step in question. It is repeated till a feasible solution is found. In the worst case, all the flows are scheduled in a single step. However, in an average case, there is a significant performance improvement when network utilization is low.

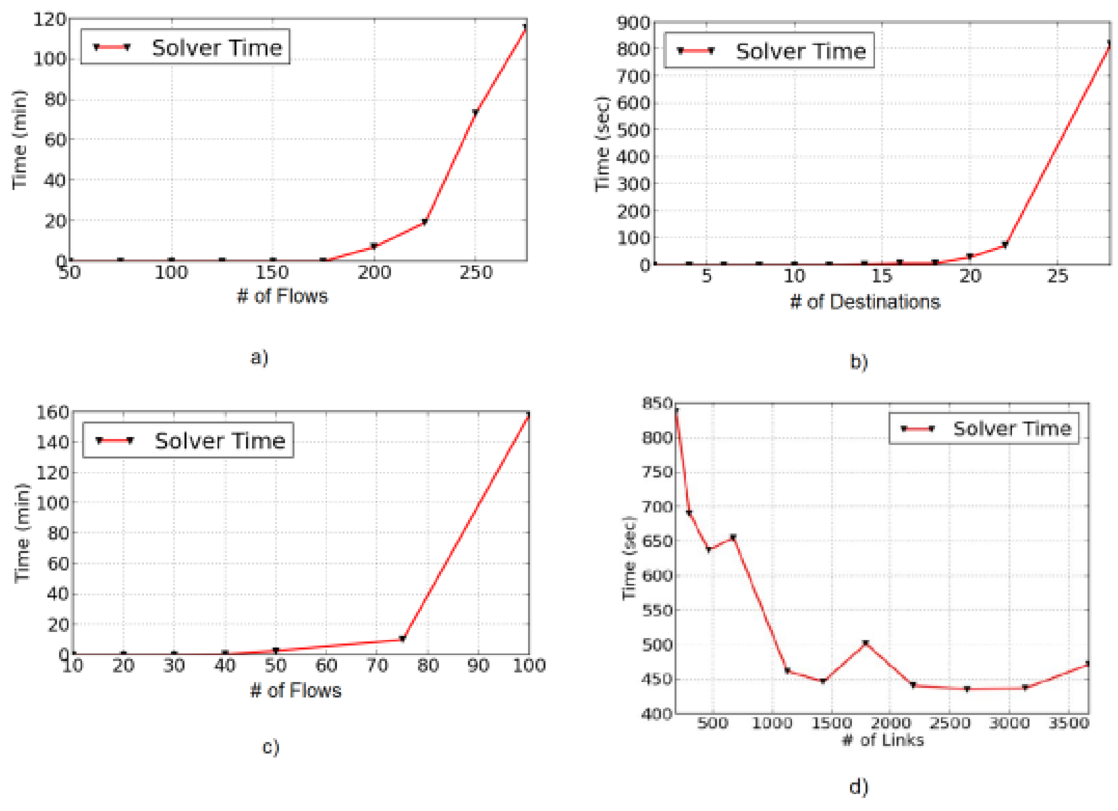


Figure 2.4: Graphs for evaluation of SMT method

The researchers performed various evaluations to solve the linear optimization objectives of the SMT solver for different scenarios. The scenarios were changed according to varying number of flows, varying number of links in the topology, varying number of destinations for a multicast and varying the period of flow. The evaluations here I am using has been done by Subarna Singh at the University of Stuttgart in her Master Thesis titled "Routing Algorithms for Time-Sensitive Networks" [12] as can be seen in Figure 2.4. Graph a) shows time required by the SMT solver to generate a schedule with respect to the number of unicast flows varying from 50-300 flows. Graph b) shows the total time required by the solver having a set of 50 flows as input with respect to a number of destinations ranging from 2-30. Graph c) shows the time required by the solver to schedule the flows with varying period size. It shows that the time complexity of SMT solver is exponential with respect to flows and thus it can't be used in large scenarios. Graph d) shows that there is no link between the time taken to calculate the schedule with respect to the total number of links in a topology.

2.4.2 No-wait Packet Scheduling for IEEE Time-sensitive Network

The method was developed by Durr et al at the University of Stuttgart [7]. The objective is to optimize the TSN scheduling for Network Interface Cards(NICs) and switch gate drivers by adapting the well-known approach Job Shop Scheduling Problem. They adopted it to a No-wait Packet Scheduling Problem for calculating TSN schedules yielding minimum network delays for time-triggered traffic.

Mostly TSN switches use "guard bands" to isolate scheduled traffic from best-effort traffic as discussed in the previous section. Increase in the number of guard band events in a schedule causes a decrease in bandwidth available for best-effort traffic and hence there is a need to reduce such events. They can not be eliminated but can be reduced such that each gate opening events lead to a large number of packets transmits belonging to the scheduled traffic. Researchers compute such TSN schedules with minimal gate opening events.

Job scheduling Problem(JSP) is a well-known scheduling problem used in operational research. A set of jobs consisting of multiple operations such as milling, drilling etc. have to be done at a set of machines in a given sequence. Each operation can be done exactly only on one machine as it takes a defined time to do one operation and one machine can process only one operation at a time. The JSP has to generate an optimized schedule such that no more than one operation is scheduled on a corresponding machine at the same time. In No-wait Job scheduling Problem(NW-JSP), there is an additional constraint that a job can't be stopped once it started. The operations can be scheduled at the starting time of job alone with this new constraint. JSP also tries to minimize the makespan, makespan is defined as the maximum time to finish all the operations for all jobs.

NW-JSP has to be mapped to packet scheduling problem, known as No-wait Packet Scheduling Problem (NW-PSP) by determining the opening and closing times of gate for schedule packets at switches and the time the packets are transmitted by NICs in the network. The time-sensitive flows correspond to the jobs and total flows on each switch correspond to a sequence of operation with no delay in the network. Network delay is basically a combination of four delays, propagation delay , processing delay, queuing delay and transmission delay. According to NW-PSP model, an operation only includes transmission and not packet processing and propagation. Thus the processing time of the job is mapped to transmission delay. It also considers propagation and processing delay which is assumed to be same for all the switches in the network. Queuing delay is assumed is assumed to be zero due to no-wait property. A packet can't be processed on a switch until the complete packet has been propagated by the previous neighbor switch. The packets are transmitted immediately after processing such that packets travel non-stop in the network. This results in minimal queue size and switches can better utilize the queues for best-effort traffic.

The main objective of the method is to minimize the flowspan which is defined as the finishing time of the flow finishing last. The minimization results in a compact schedule as it increases the chance of back-to-back schedule from different flows, such that the gate entries can be minimized by merging different gate opening times. NW-PSP can be formulated to Integer Linear Programming(ILP) with the objective to minimize the flowspan. The variables for ILP are the flow start times and the schedule of a flow is repeated after a hypercycle. Hyper-cycle is the Least Common Multiple(LCM) of all flow cycle periods to be scheduled. ILP solver tries to find a feasible and optimal solution to calculate all the schedule for time-triggered traffic in a single hyper-cycle.

ILP method doesn't scale efficiently for large networks. The method is not expected to find exact solutions efficiently due to high time complexity. So to improve the scalability, they proposed a more efficient heuristics method known as Tabu Search algorithm for NW-JSP.

2.5 Scheduling Algorithms

The current scheduling problem is to compute the cyclic schedules which represent the opening and closing of the gates has been found to take a centralized approach to compute the schedules. These approaches are mentioned in the 2 papers discussed in section 2.4 i.e. *No-wait Packet Scheduling for IEEE Time-sensitive Network* [] and the

Scheduling Real-Time Communication in IEEE 802.1Qbv Time Sensitive Networks[]. The runtimes for the centralized approaches are high and we want to reduce it significantly.

The paper introduces the divide-and-conquer approach, that divides a problem recursively into smaller sub-parts until it becomes simple enough to solve the problem. This has to be mapped to a complex network with various time-triggered flows routed in it. This approach looks at the lowest level of the network which is a single link connected between any 2 switches. It tries to schedule the time-triggered flows at the link level and then proceeds to combine all the computed flow schedules on a network level. To compute the schedule at the link level, we introduce the concept of flow windows which gives a time span to schedule the flow on a link. It has been discussed in algorithm 4.2 and optimum schedules are computed using an Integer Linear Programming(ILP) solver. As discussed in Introduction chapter that a divide-and-conquer approach needs a pivot link to divide the network into sub-networks. We choose the pivot link which has the maximum overlap between flow windows for all flows and it is discussed in Algorithm 4.4. It gives a direct relationship between the ILP runtimes and the overlap. The last step in the divide-and-conquer approach is to combine all the solutions which are discussed in the Algorithm 4.5. The algorithm shows how to modify the flow windows on upstream and downstream links of the pivot-link.

We further reduce the runtimes by dividing the divide-and-conquer problem by a parallel implementation. The scheduling in the divide-and-conquer algorithm happens at the link level, this gives an opportunity to compute the schedules for multiple links in parallel. The parallel implementation algorithms are discussed in Algorithm 4.6 and 4.7. The runtimes of the divide-and-conquer algorithm were computed with respect to varying number of flows and the scalability is tested with respect to the number of flows and the number of machines used in the parallel implementation. The algorithm's time complexity is exponential in nature but the complexity reduces with the increase in the number of machines. This helps us to use the parallel implementation for large topologies with a high number of time-triggered flows.

2.6 Heuristic Algorithm

The Integer Linear Programming (ILP) tries to find the most optimum solution for any given problem but the time complexity for an ILP solver is exponential in nature which is quite high. An ILP can run for a few days if the network is too complex and too many flows in the network. If the optimality of the solution is not a top priority than an approximate feasible solution is also sufficient if it can be solved in a lower amount of time as compared to ILP. This is the approach used by heuristic algorithms.

A heuristic algorithm usually finds a solution which is close to the best optimum solution and they find it fast and easily, thus it is considered as approximately and not an accurate algorithm. Sometimes, it can find out the best solution also, but it will still be called heuristic until the solution is proven to be the best. In this thesis, we present the first-fit method heuristics for a bin-packing problem [15] that has been little modified to find schedules for time-triggered flows in a network.

In the bin-packing problem, the basic concept is that a number of items of different sizes/volumes must be packed into a finite number of bins, each having volume, V, such that it minimizes the number of bins used. There are many variations for this problem, linear packing, packing by weight and so on. There are many heuristics developed to solve the bin-packing problem and one of them is the first-fit method.

The first-fit method placed is a simple and a straight-forward approach and it is also known as the greedy approach. In the algorithm, any random element is picked and it attempts to fit into any of the already open bin. If no bin is found which can fit the item, then a new bin is open and placed in it. The first-fit method algorithm for the scheduling problem is discussed in Algorithm 4.8. The time complexity of the first-fit method is an order of 1, O(1) as the evaluations show that the runtime using the heuristic algorithm remains constant with a varying number of flows in the network.

3 System Model and Problem Statement

3.1 System Model

Time Sensitive Network(TSN) needs a standardized form of configuration as to implement the various requirements for deterministic data transmission. TSN consists of different kinds of elements which are compliant with the IEEE 802.1Qbv standards and are responsible for the forwarding of packets in the network. The key elements in the architectural model of TSN consist of Centralized Network Controller(CNC), Network Elements(eg. Switches) and End nodes as shown in figure 3.1 [8]. Now we will discuss more about the elements in detail.

Centralized Network Controller(CNC): CNC is the global center for all TSN network configuration as it has a central view of the whole topology. It can discover the underlying network topology by using Discovery Protocol(Link Layer Discovery Protocol(LLDP)) from the southbound API. It also has the capability to identify the TSN-relevant features e.g. switch latency, link speeds of different network elements. It also controls the opening and closing of gate drivers in switches according to the transmission schedule by distributing the schedule to all the various switches. It also sends the timestamps of starting of the scheduled flow to hosts that when to start injecting the packets in the network. It uses standardized mechanisms such as SNMP protocol to configure the TSN devices.

Flow: A flow is defined by a set of parameters such as source host, destination host, the size of the data and the period. It is also written as:

Flow, F = (src, dst, size, period)

A flow can be time-triggered or best-effort traffic. Real-time applications use time-triggered packets as to meet the hard deadlines. Whereas best effort traffic doesn't have to guarantee any hard deadlines and it can tolerate some delay in the delivery of the packets to its destination.

Each flow has a window with starting, early and late pointers as shown in Figure 3.2. The start pointer(S) indicates the starting of the window and the time after which the egress port can start transmitting a flow in the network. The early pointer(E) indicates

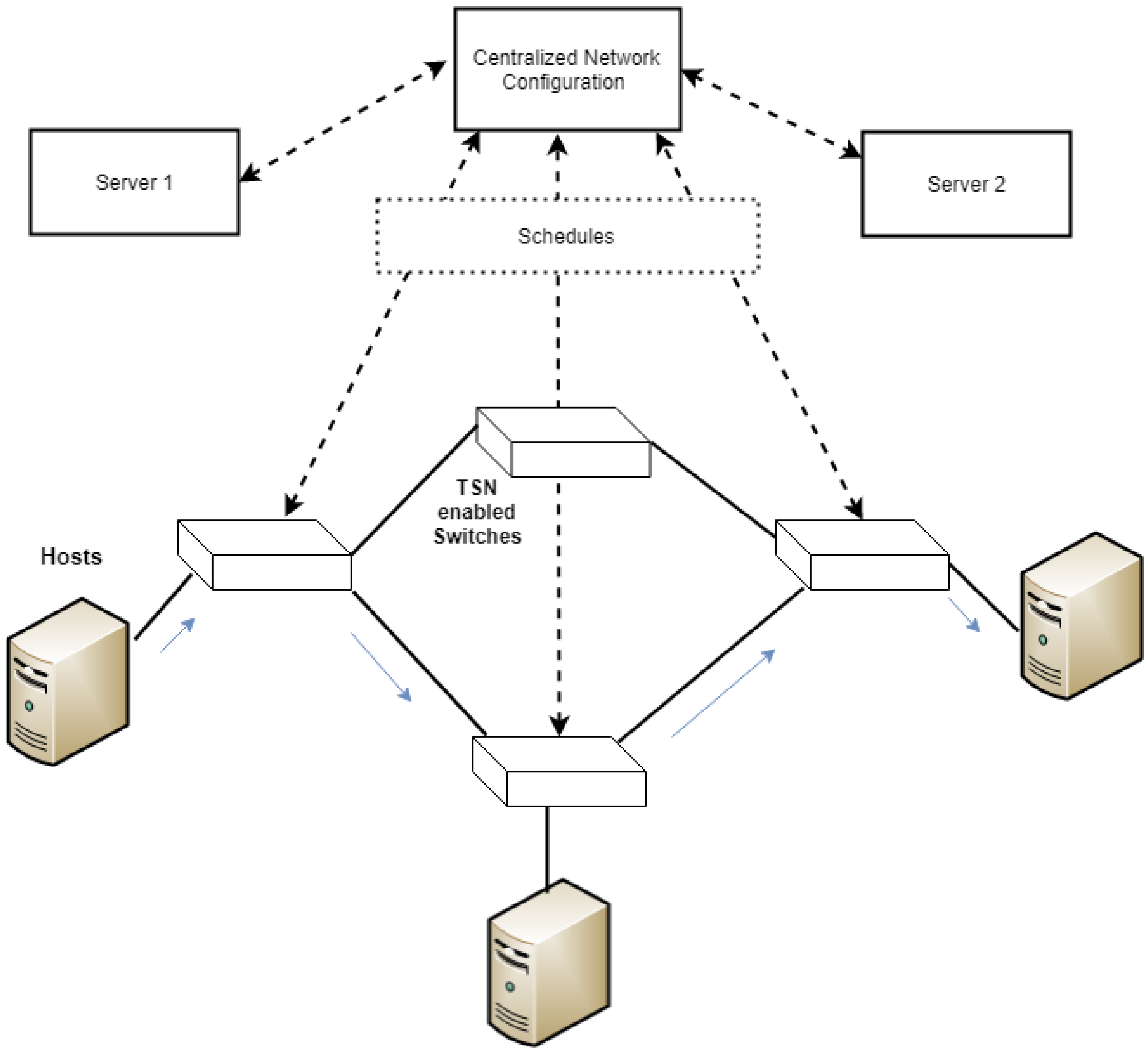


Figure 3.1: TSN Architecture

the earliest time when a flow can be transmitted so that it satisfies the minimum latency bound in the network. The late pointer(L) indicates the end of the window and the flow has to be transmitted before the late pointer time. The desired window to start the transmission of the flow is between the early and late pointer called a reception window. The reason is that the packets don't have to be queued at an egress port of any switch in its flow path as it is within the reception window. if the flow starts between the start and early pointer, the packets have to be queued and thus have to wait before it can be further forwarded from the egress port of the switch.

End Systems: The End systems or hosts work as originators and/or destinations(listeners) for a flow. They can either send or receive data but do not forward

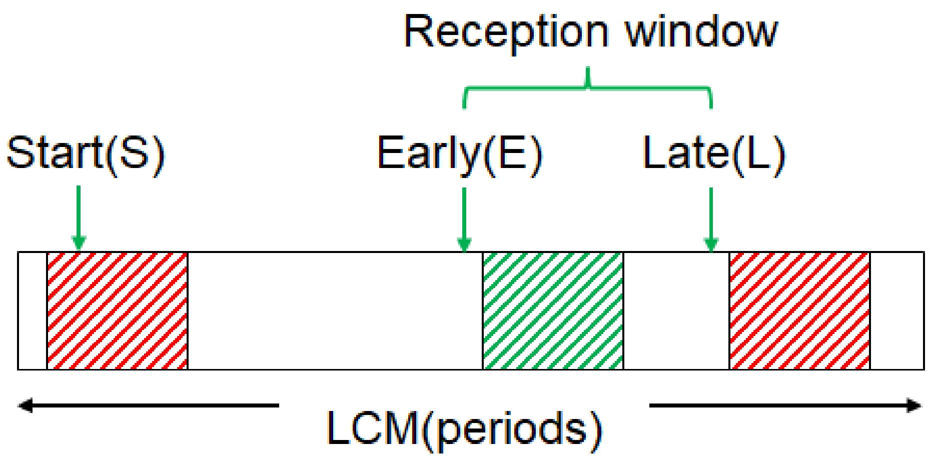


Figure 3.2: A flow with window

it. They End systems can be TSN enabled or non-TSN enabled. TSN enabled end systems can handle time-triggered traffic whereas non-TSN enabled end systems can't

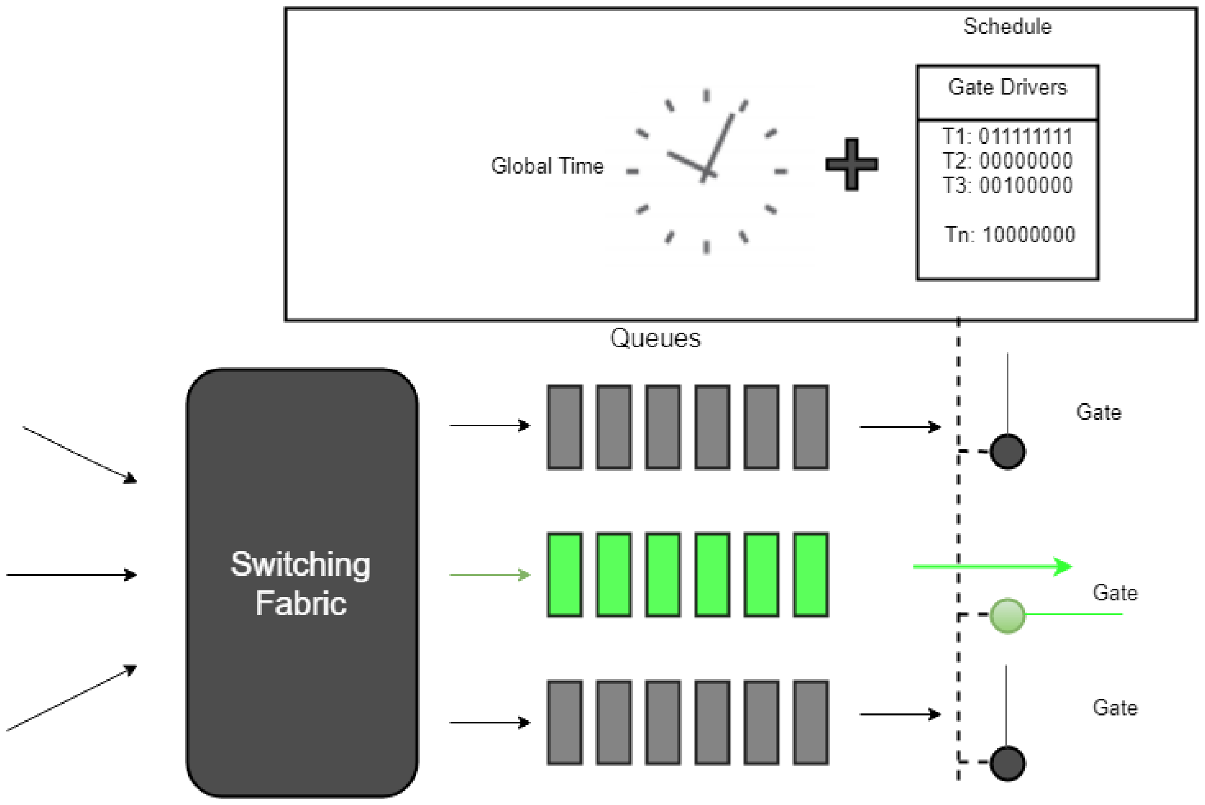


Figure 3.3: TSN complaint Switch Architecture on Single Port

handle it. In non-enabled TSN end systems, the switch connected to the host decides to buffer the packets in a queue and transmits the packets to the next switch based on the transmission schedule. We will consider only TSN enabled end systems in our thesis work. The sender host sends some information along with the data which tells the type of the data stream it wants to transmit. The information includes characteristics such as the receiver MAC address and the Class of Service(CoS) priorities. The receiver can register for the intended services and can receive the data packets matching with it's registered characteristics. We also assume that end systems send unicast packets with a constant bit rate to the destination end system.

Switches: TSN-enabled switches support the handling of packets and acts as an interface with the Centralized Network Controller. They have been standardized by the Time-Sensitive Network Group and must be compliant with the IEEE 802.1Qbv specifications. The switches are designed in a such a way to process and forward scheduled traffic along with the best-effort traffic. Mostly one queue in a switch is reserved for the scheduled traffic and the other unreserved queues are used for best-effort traffic. According to the Quality of Service(QoS), the packets are identified and placed in the appropriate queues and the transmission of the packets is executed from the egress ports during the corresponding scheduled time windows [14].

Each switch comprises of 8 queues per port and each port is mapped to a corresponding gate. Each gate has two states,'open' and 'close' and is decided by the gate control drivers. The selection process only transmits the messages from the queues at the egress when the gate is in 'open' state and other queues will be blocked during this transmission. So, the closing of the gate can protect the deterministic traffic from the non-scheduled traffic in order to guarantee the bounded latency in the network.

The selection process of the gate drivers takes into account two sets of events. The first event is the timestamp relative to the start of the program. The second event is the bitmask field which tells the state of the gates corresponding to the queues after the intended time in the first event, As it can be seen in figure 3.3, after time T2, all the gates are closed but at time T3, all the gates are closed except gate 3 (00100000). The corresponding traffic from the queue will be transmitted after the event T3. Also, all the gate events are programmed for a predefined time, T_{cycle}, after which they start repeating the same schedule.

Time Services: TSN technology is based upon a system-wide precise time system. It is assumed that all the systems have a precise synchronization of clocks based on precise time standards, IEEE 1588(PTP) and IEEE 802.1AS-REV. The TSN enabled devices are expected to pass the precise time within the system and also perform the deterministic schedule functions accordingly.

Servers: These are the optional network elements connected to the Centralized Network Configuration(CNC) to compute the transmission schedules. They are servers having enough processing power to calculate the schedules as fast as possible to minimize the total time to schedule all the flows. They compute faster as different servers execute in parallel when we apply the divide-and-conquer approach to our system. A multi-processor system helps in faster execution of a problem as different sub-problems can be executed on different machines and than combined back to get the solution. In our thesis approach, we assume that CNC acts as a master and it distributes the required information to the servers(slaves). The servers after computing the schedule will send the solution back to CNC.

3.2 Problem Statement

The goal of Time Sensitive Networks(TSN) is to address the lack of support for real-time systems. TSN provides deterministic nature of the real-time systems with bounded low latency and jitter, low delay variation, and extremely low data loss for time-critical traffic. The problem that we address in this thesis can be expressed as follows, compute a scheduling algorithm to efficiently calculate the transmission schedule in a time-sensitive network as fast as possible. We have already discussed in Section 2.2 about the other scheduling methods but they all use a centralized approach to compute the schedules.

In No-wait Packet Scheduling method, the researchers computed an optimal solution to generate the transmission schedule in terms of flowspan. They are successful in improving the quality of schedule transmission if the flowspan is as low as possible. However, the improvement in quality comes at a cost of scalability. The results show that the execution time increases polynomially with the increase in the number of flows. One of the reason is that they consider a centralized approach over the entire topology to find the feasible solution. To calculate the flowspan, it takes into account all the operations in the flow path in an order when computing the schedule for any particular flow. Moreover, the starting of the flow time is always influenced by the flows preceding to it in a totally ordered set. The dependency of an operation on the preceding operation in a flow prevents any operation to compute it's schedule independently. The calculation of schedule for all the constituent operations of a flow in a sequential order results in higher runtimes. The aim of our thesis is to find a scheduling algorithm which has lower runtimes than the centralized approach even if we have to trade-off the optimality of the solution. The problem statement can be further explained in the context of flow windows.

The time-triggered flows that are transmitted throughout the network have reception windows as mentioned in section 3.1. We have to determine a transmission schedule for a link with multiple flows that adheres the reception windows of the flows.

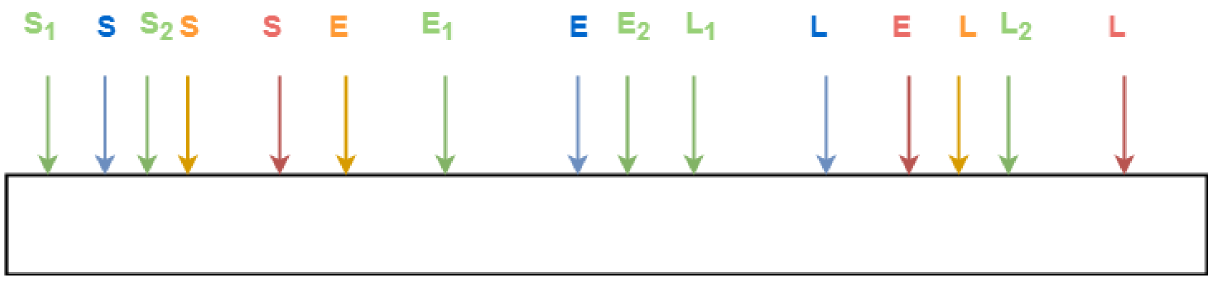


Figure 3.4: A link with multiple flow windows

Consider a link with multiple flows as shown in figure 3.4 with different window sizes. Each flow has it's own reception window as can be seen by the Start(S), Early(E) and Late(L) pointers.

The flow schedule problem is similar to a bin packing problem with additional constraints. Bin packing problem is a problem of packing a set of items into a number of bins such that the total weight, volume etc. doesn't exceed some defined maximum value. A simple solution is a first-fit method in which the items are placed in the order they come in the first bin in which they fit. In the case of the time-triggered scheduling algorithm, we have to schedule the flows in a such a way that it fits inside the reception window of the flow.

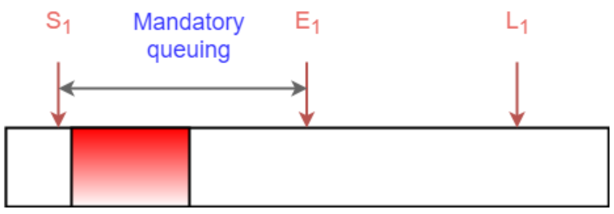


Figure 3.5: Mandatory queuing in a link

The second task is the optimization of the scheduling algorithm. In order to reduce mandatory in-network queuing, schedule flows as close to the "Early" pointer as possible. Mandatory queuing happens when the flow is scheduled between the start pointer and early pointer as shown in figure 3.5. The packets that are sent before Early pointer have to be queued at the egress port of a switch as they cannot be transmitted before the Early pointer for that particular link. This results in the blocking of network ports and the subsequent packets from other flows have to wait in the queue till the queue is empty, thus reduces the overall utilization of the network.

The runtimes of the divide-and-conquer approach can be reduced further if it can be implemented on multiple machines in parallel. Note that the reduction in number of machines doesn't guarantee that the execution time will decrease substantially as not all machines will work at their 100 percent utilization. Finally, we reduce the schedules further using a simple heuristics approach based on the first-fit method.

4 Scheduling Algorithms

In this section, we discuss the scheduling algorithm based on the divide-and-conquer methodology for the time-sensitive network. The algorithms for the divide-and-conquer approach discussed is the upstream and downstream reception windows modification. The solution for scheduling is formulated as Integer Linear Program(ILP) and a heuristic approach based on first fit method. Here, we now define some of the terms used in the algorithms.

4.1 Terminology

- **Graph**: A Graph is a collection of nodes together with a collection of edges, which are pairs of nodes. A graph is called a directed graph if the edges are directed i.e. there is an order of the edge pairs(u,v). If no order of edges is defined, it is known as an undirected graph.
- **Flow**: A flow in network is generally represented by $f \equiv (src, dst, period, size)$ where src is the source host, dst is destination host, period is the time within which the packets from source has to be reached at destination node and size represents the packet size in bytes transmitted in the flow by the source.
- **Flow Window**: It is defined as the window between which a flow can be scheduled anywhere. It is represented as $flowWindow \equiv (start, early, late)$. It has 3 states, start pointer, early pointer and late pointer. Start represents the start time of the window of the flow, early represents the earliest time at which the flow should ideally start to avoid in-networking delay and late represents the latest time the flow must be scheduled.
- **Path of a Flow**: It is the sequence of the links over which the flow is transmitted from source to destination in the network.
- **Reception Window**: It is defined as the window between early and late pointer which is ideal to schedule a flow without any mandatory in-network queuing.

Algorithmus 4.1 Adapting Flow Windows

```
 1: procedure CREATESCHEDULE
 2:     $route \leftarrow \{\}$
 3:     $cumTime \leftarrow \{\}$
 4:     for all $flow$ in $F$ do
 5:         $(src,\ dst,\ period,\ size) \leftarrow flow$
 6:         $(start,\ early,\ late) \leftarrow flowWindow[flow]$
 7:         $route[flow] \leftarrow$ DIJKSTRAPATH($topology$, $src$, $dst$, $weight$)
 8:         for all $edge$ in $reverse(route[flow])$ do
 9:             $cumTime[edge] \leftarrow ((srTime * size)/MTUSIZE + prTime) * linksTraversed$
10:             $flowWindow[edge][flow][start] \leftarrow flowWindow[edge][flow][start] - cumTime[edge]$
11:             $flowWindow[edge][flow][early] \leftarrow flowWindow[edge][flow][early] - cumTime[edge]$
12:             $flowWindow[edge][flow][late] \leftarrow flowWindow[edge][flow][late] - cumTime[edge]$
13:         end for
14:     end for
15:     return $flowWindows$
16: end procedure
```

4.2 Adapting Reception Windows

In a time-sensitive flow, the packets encounter some network delay when they are transmitted in a network. A network delay consists of four delays which are propagation delay of signals along the link, the processing delay for deciding on which port to forward the packets, the queuing delay is the time spent by packets in the queue on an outgoing port and the transmission delay to serialize the packets on the line. We assume processing delay and propagation delay to be same for all the links in the network. The transmission delay is defined by the packet size of the flow and data rate of the switch. The queuing delay is assumed to be zero due to the no-wait property of the time-sensitive network.

In figure 4.1, a flow,*F* flows through L_1, L_2, L_3, L_4 links. The flow reception windows have to be adjusted on the whole path as to account for the transmission, processing and propagation delays. In figure 4.2, the windows on each switch have to be moved with respect to the corresponding delays.

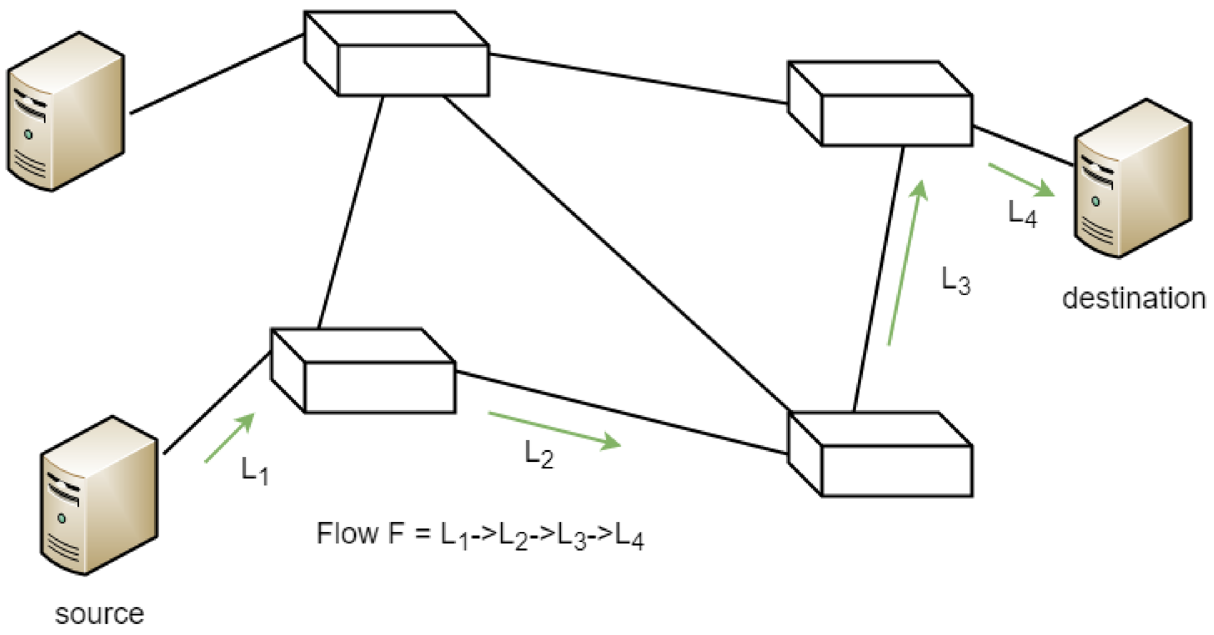


Figure 4.1: A flow in a network

In the algorithm, 4.1 Adapting Link Windows, the data structure to compute the cumulative delay per link is $cumTime$ based on serializing delay, $srTime$ and propagating delay, $prTime$ that are fixed and same for all the links. The transmission delay is based on the packet $size$ and $MTUSIZE$, which is the Maximum Transmission Unit(MTU), the largest size of a single data unit that can be transmitted over the network. For Ethernet packet, MTU size is 1500 bytes.

The data structure, $flowWindows$ contains the start, early and late pointers for each flow corresponding to all the links in its flow path. Initially, start, early and late pointers for each flow window are given for the last link. The start, early and late pointers for a link is calculated by subtracting the cumulative delay of the link from the next in path link corresponding start, early and late pointers, for eg. $Start_{f_1} \leftarrow Start_{f_2} - cumDelay$. The timing The path has been calculated using Dijkstra shortest path algorithm.

4.3 Integer Linear Program for Divide-and-Conquer approach

Here, we define the divide-and-conquer approach model using an Integer Linear Program(ILP) formulation. First, we investigated how to implement the divide-and-conquer approach to our TSN scheduling problem. The divide-and-conquer approach has 3 stages as discussed in Introduction chapter.

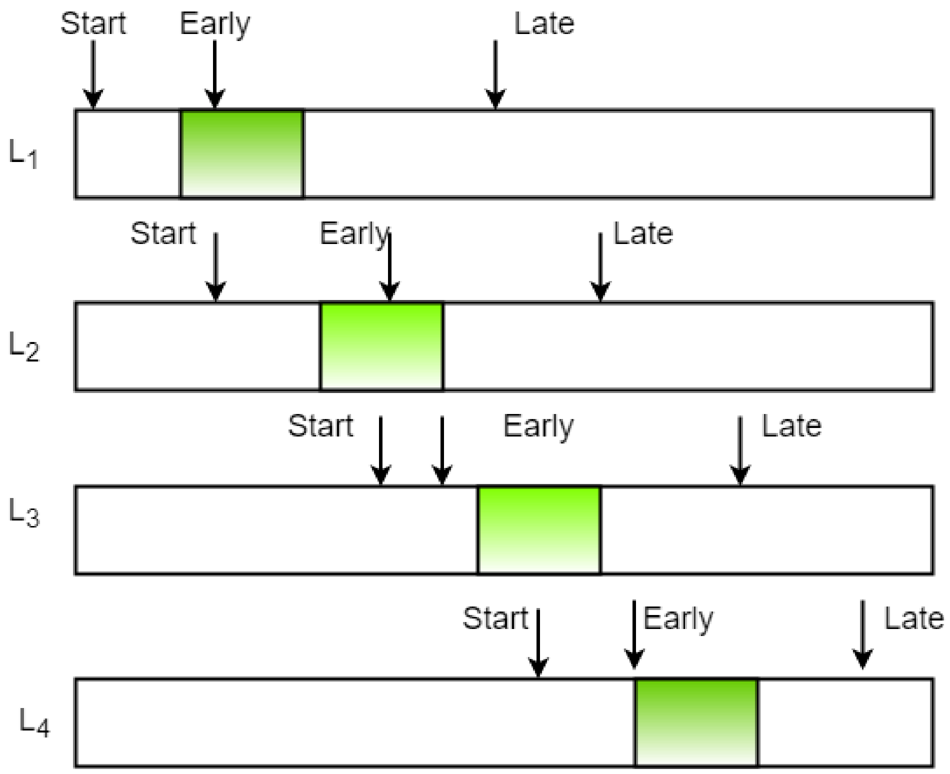


Figure 4.2: Adapting Windows due to delays

1. **Divide** the problem into sub-problems. To divide a TSN network, a pivot link is required to divide into sub-networks. The link choosing strategy has been discussed in next section, 4.4.
2. **Conquer** the sub-problems by solving them recursively. The objective is to reduce the search space by a constant factor and if they are small enough, solve the subproblems as base problems. In a TSN network, the smallest sub-problem is a single link on which we can compute the scheduling using an ILP solver. In this section we have formulated the ILP inputs, variables and constraints.
3. **Combine** the solution of sub-problems into one main solution. In a TSN network, it corresponds to modifying the upstream and downstream reception windows which we will discuss later in section 4.5.

   The initial ILP based solution approach is devised by Naresh Nayak, M.Sc from University of Stuttgart. To find a schedule on a link, first adapt all the reception windows for all flows on each link as in Algorithm 4.1. This gives us a starting point on any link to compute the schedule.

   The ILP inputs corresponding to the problem are:

Algorithmus 4.2 compute the schedule on any link

```
 1: procedure COMPUTESCHEDULE($flows$, $flowWindows$ , $T_{max}$)
 2:     $flowPairs \leftarrow\ Selectf1, f2$ from $flow$
 3:     $ilpProblem \leftarrow \min(\sum_{\forall f \in F} |offset_f - early_f| - \frac{\sum_{\forall f \in F}(offset_f - start_f)}{Normalizationfactor}$
 4:     for all $flow$ in $F$ do
 5:         $ilpProblem \leftarrow offsets[flow] <= flowWindow[f][late]$
 6:         $ilpProblem \leftarrow offsets[flow] >= flowWindow[f][start]$
 7:     end for
 8:     for all $f_1$, $f_2$ in $flowPairs$ do
 9:         $ilpProb \leftarrow Offsets[f_1] - Offsets[f_2] + flows[f_1][size] <= Ordering(f_1, f_2) * T_{max}$
10:         $ilpProb \leftarrow Offsets[f_1] - Offsets[f_2] + flows[f_1][size] <= (1 - Ordering(f_1, f_2)) * T_{max}$
11:     end for
12:     ILPSOLVER($ilpProblem$)
13:     return ($Offsets$, $flowOrder$)
14: end procedure
```

- The set of all flows on a link with the corresponding flow windows i.e. start, early and late pointers.

$$F \equiv\ f_1, f_2, ..., f_n$$

$$flowWindow \equiv (start_i, early_i, late_i)$$

- The maximum schedule period length, T_max under which we have to schedule all the flows in a time-sensitive network. This is the upper bound period as to maintain the latency bound for the whole network.

The main objective of our problem is to find the optimum offset value for a flow such that it flows start as near to the early pointer as possible. The delay in the whole network should be as minimum as possible to avoid the traffic queuing at the egress ports. If it starts before early pointer, the packets have mandatory in-network delays due to queuing at egress ports of the switch. The ILP objective in the Algorithm 4.2 is given on line 3 :

$$ilpProblem \leftarrow \min(\sum_{\forall f \in F} |offset_f - early_f| - \frac{\sum_{\forall f \in F}(offset_f - start_f)}{Normalizationfactor}$$

The above equation can be divided into 2 parts. The first part states that the total difference between the offset and the early pointer irrespective of the sign for all flows on a link should be as minimum as possible. The second part states that the offset should be on the right side of the early pointer as possible to avoid the in-network delays, so we

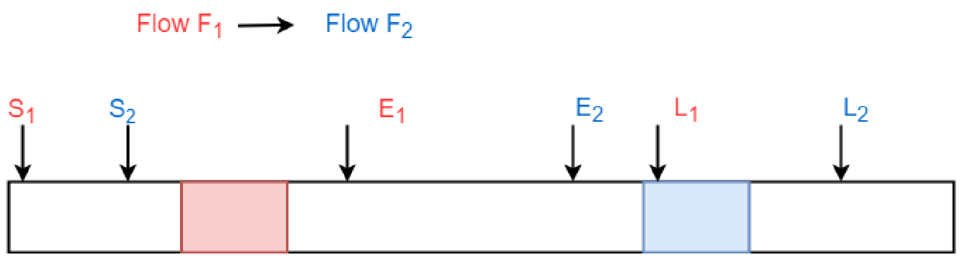


Figure 4.3: Order between 2 flows

subtract the difference between offset and early pointer from the first part. We divide the second part by a normalization factor, which is given by :

$$Normalizationfactor \leftarrow \sum_{\forall f \in F} (late_f - start_f) + 1$$

The constraints for the ILP problem are:

- Offset Bounds: A flow must be scheduled within the start and late pointers of the window, otherwise the schedule of the flow is infeasible i.e.

  $$\in finF : start_f \leq offset_f \leq late_f$$

- Collision Constraints: Due to physical limitations, two packets from different flows can't be transmitted over a link at the same time. There is an order between any two flows on the same link and it must be maintained on all links in both flow paths. It is given by:

  $$\in (f_1, f_2) \forall F$$

  - Either $f_1 \rightarrow f_2$
  - OR $f_2 \rightarrow f_1$

  If the order is not maintained on all the links, then one flow, f_1 might reach before the other flow, f_2 on a link if the order is already established as $f_2 \rightarrow f_1$. The packets from the flow, f_1 has to wait in the queue of the particular switch as to maintain the flow order, which results in mandatory network delays which we want to avoid at all cost. It has been shown in figure 4.3.

Depending on all the constraints and input variables, ILP will give an optimum solution to the problem. The solution is the offset values for all the flows on a link and the order between the flows satisfying the constraints. Scheduling the flows as close to the early pointer as possible reduce the whole network delay. The network delay is the summation of difference between offset value and early pointer for all the flows in the network.

$$offsetDelay \leftarrow \sum_{\forall f \in F} |offset_f - early_f|$$

Algorithmus 4.3 Choosing Link: w.r.t. number of flows

```
1: $orderLinks \leftarrow$ {}
2: for all $edges$ in $network$ do
3:     $numberFlows \leftarrow$ total flows on $edge$
4:     $orderLinks \leftarrow$ SORT($numberFlows$)
5: end for
```

Algorithmus 4.4 Link Choose strategy: slack time w.r.t. reception window

```
1: $totalOverlap \leftarrow$ {}
2: for all $flow$ in $F$ do
3:     $(start,\ early,\ late,\ size) \leftarrow flowWindow[flow]$
4:     $flowPairs \leftarrow\ Select f1, f2$ from $flow$ inOrder
5: end for
6: for all $f1,\ f2$ in $flowPairs$ do
7:     if $flowWindow[f1][early] + flowWindow[f1][size] < flowWindow[f2][early]$ then
8:         $overlap \leftarrow flowWindow[f1][early] + flowWindow[f1][size] - flowWindow[f2][early]$
9:     else
10:        if $flowWindow[f2][early] + flowWindow[f2][size] < flowWindow[f1][early]$ then
11:            $overlap \leftarrow flowWindow[f2][early] + flowWindow[f2][size] - flowWindow[f1][early]$
12:        else
13:            $overlap \leftarrow 0$
14:        end if
15:    end if
16:    $totalOverlap \leftarrow \sum overlap$
17: end for
18: return $totalOverlap$
```

4.4 Choosing Link

One of the major problems is to choose the pivot element for our divide-and-conquer approach. The link in the network which is chosen to be pivot element should make the scheduling much more convenient and fast. The pivot element divides the network into smaller sub-networks that makes it easier to calculate the schedule as now we have to only consider a smaller network. The criteria to chose the link is based on these approaches:

- **Number of flows on a link**: The pivot link is selected with the highest number of flows that are scheduled to flow over the link in one transmission period. In most of the network topologies such as mesh network or tree topology, it is assumed that the link with the highest flows will be somewhere in the middle of the network. So, if we choose the link with the highest flows, the network is divided equally into 2 sub-networks.
- **The cumulative slack w.r.t the reception windows**: The cumulative slack w.r.t reception window is defined as the total overlap between the reception windows of all the flows on a link. The link with the highest overlap is selected as the pivot element for the first iteration and so on for the next iterations. It is assumed that the link with the highest overlap will be the trickiest and most complicated to schedule as there is not much flexibility to schedule the flows in the cyclic period.

In figure 4.2, one of the links is chosen as the pivot element which divides the network into two sub-networks based on the divide-and-conquer approach.

In the results, we see that the overlap approach to choose a link has better results as compared to number of flows approach. Even a link with the highest number of flows can have an effective overlap of 0, the schedule will be easier to compute and faster. Thus, there is no correlation between the execution time and the number of flows.

In algorithm 4.3, all the links in the network are sorted in reverse order w.r.t number of flows on each link. The data structure $orderLinks$ stores the links in the decreasing order of the number of flows.

The next algorithm 4.4, first it is checked if there is any overlap between any 2 flows on a link. If yes, the overlap is calculated between 2 flows reception windows which is computed as:

$$overlap \leftarrow flowWindow[f1][early] + flowWindow[f1][size] - flowWindow[f2][early]$$

It can be shown with the help of an example shown in figure 4.4. To calculate the overlap, we assume an ideal scenario when all flows are scheduled just after the early pointer in the reception windows. The overlap between F_1 and F_2 is given by O_{F_1,F_2} and between F_2 and F_3 is given by O_{F_2,F_3}. The overlap between F_1 and F_3 is 0 as the early pointer of F_3 is starting after F_1 is scheduled after it's early pointer.

The total overlap should ideally be the sum of the individual overlap between all the pair of flows on a link. But consider a case as shown in figure 4.5, the total overlap for the 4 flows, F_4, F_5, F_6, F_7 is equal to the total overlap for the 3 flows, F_1, F_2, F_3 as in figure 4.4.

$$O_{F_1,F_2} + O_{F_2,F_3} = O_{F_4,F_5} + O_{F_6,F_7}$$

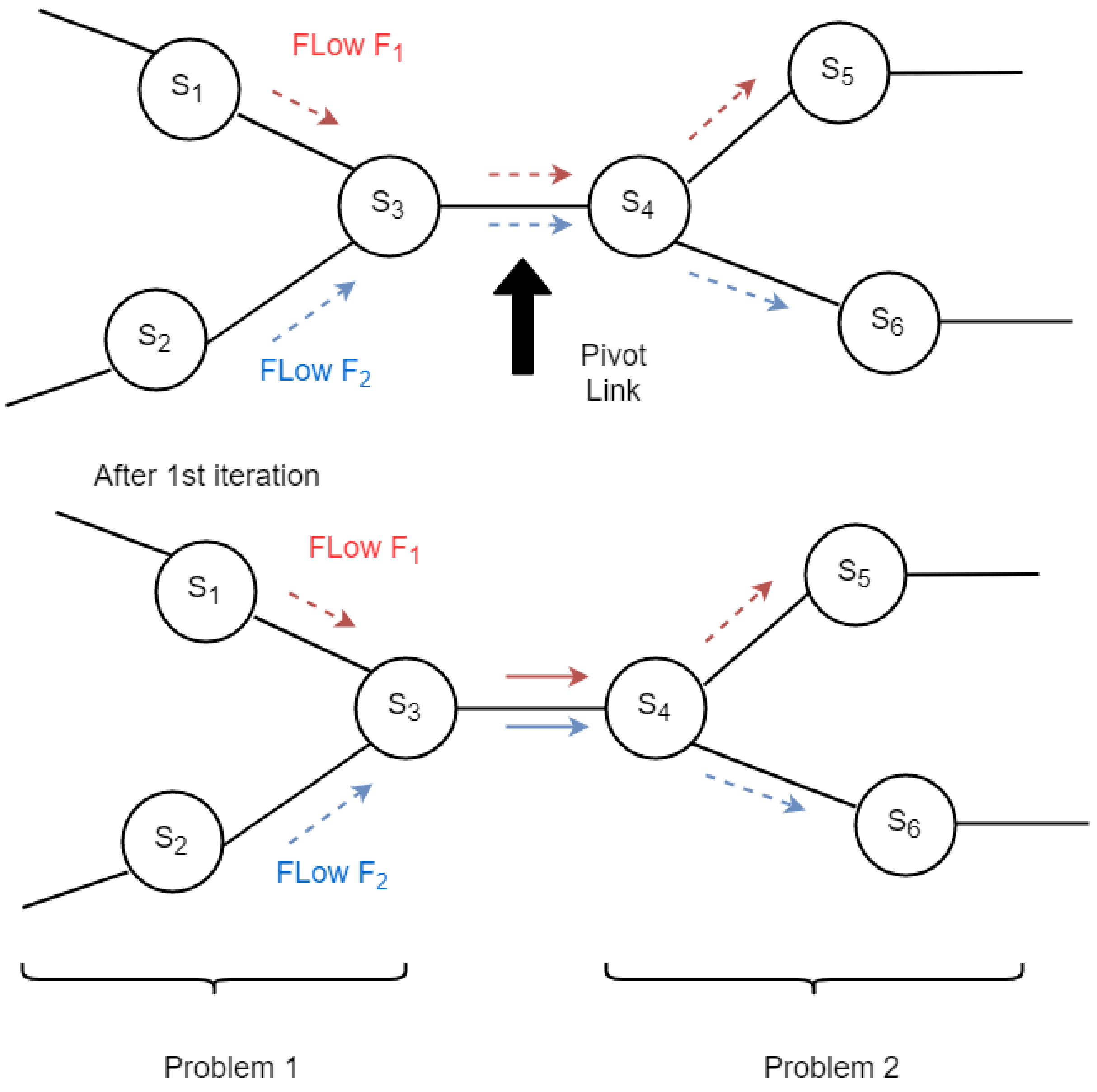


Figure 4.4: Divide-and-conquer approach

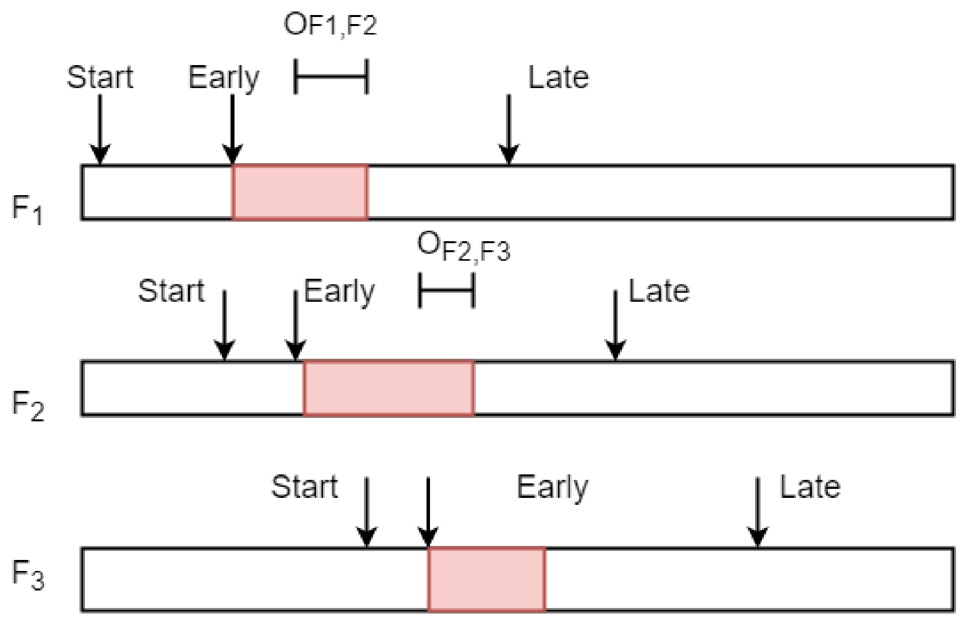


Figure 4.5: Overlap between 3 flows

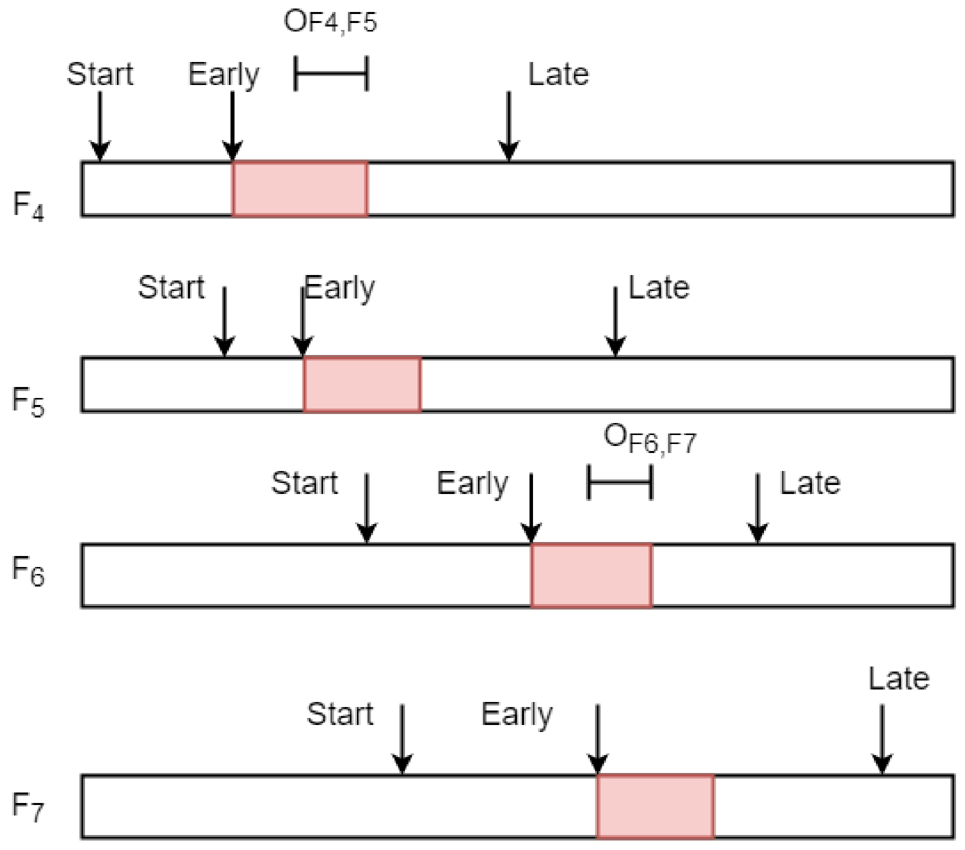


Figure 4.6: Overlap between 4 flows

With some evaluations it has been found that the execution time to compute the schedules on the 3 flows is greater than the 4 flows. The reason for this abnormality is due to the cascading effect of the overlap O_{F_1,F_2} and O_{F_2,F_3}. Flow F_2 has overlap with both F_1 and F_3. The overall overlap for flow, F_3 will be much greater than O_{F_2,F_3} as it will be summation of O_{F_1,F_2} and O_{F_2,F_3}. The flow F_2 is common in both overlaps and the shift in the schedule due to overlap,O_{F_1,F_2} will have a domino effect on all the next subsequent overlaps. So ,we tweak the overlap formula to include the cascading overlap effect, thus it becomes:

$$totalOverlap \leftarrow \sum overlap + \sum cascadeOverlap$$

.

This formula helps to better select the pivot element for our schedule problem.

Note that we implemented only the above mentioned 2 approaches for selection of the pivot element due to the subjectivity of the problem. We also did some evaluations with the pivot link having the least amount of overlap or the minimum number of flows. In both cases, the results were inconclusive as the solution found was infeasible in some of the test scenarios.

4.5 Modify Reception Windows

As already mentioned that after a schedule has been found, all the upstream and downstream reception windows have to be modified. This is the last step for our divide-and-conquer approach in which we combine all the sub-problems solutions to a single solution.

For our TSN network, the approach is to modify the upstream and downstream reception windows for all the links in the flow path. Once an offset value has been found for a particular flow, it has an effect on the whole flow path as we have to satisfy the constraint, $Start_f \leq Offset_f \leq Late_f$

The offset values for all flows remain fixed on a particular link once a schedule has been computed. The windows for other links in the path are change according to the following criteria:

- Upstream links: The links corresponding to the upstream of the flow have to be modified as (note, that the subscripts denote the link number):
  - If $Offset_{f_1} > Early_{f_1}$, then $Late_{f_2} = Offset_{f_1}$.

Algorithmus 4.5 Modify Reception Windows

```
1: procedure CHANGERECEPTIONWINDOW($link$, $offset$)
2:     for all $flow$ in $flowWindows[link]$ do
3:         for all $edge$ in $route[flow]$ do
4:             if $link == edge$ then
5:                 $upStreamLinks \leftarrow$ all links to the right of link in $route[flow]$
6:                 $downStreamLinks \leftarrow$ all links to the left of link in $route[flow]$
7:             end if
8:         end for
9:         for all $edge$ in $upStreamLinks$ do
10:            if $offset > flowWindow[link][flow][early]$ then
11:                $flowWindow[edge][flow][late] \leftarrow$ Move it by $offset - flowWindow[link][flow][late]$
12:            else
13:                $flowWindow[edge][flow][early] \leftarrow$ Move it by $offset - flowWindow[link][flow][early]$
14:                $flowWindow[edge][flow][late] \leftarrow flowWindow[edge][flow][early]$
15:            end if
16:        end for
17:        for all $edge$ in $downStreamLinks$ do
18:            if $offset > flowWindow[link][flow][early]$ then
19:                $flowWindow[edge][flow][start] \leftarrow$ Move it by $offset - flowWindow[link][flow][start]$
20:                $flowWindow[edge][flow][early] \leftarrow flowWindow[edge][flow][start]$
21:            else
22:                $flowWindow[edge][flow][start] \leftarrow$ Move it by $offset - flowWindow[link][flow][start]$
23:            end if
24:        end for
25:    end for
26: end procedure
```

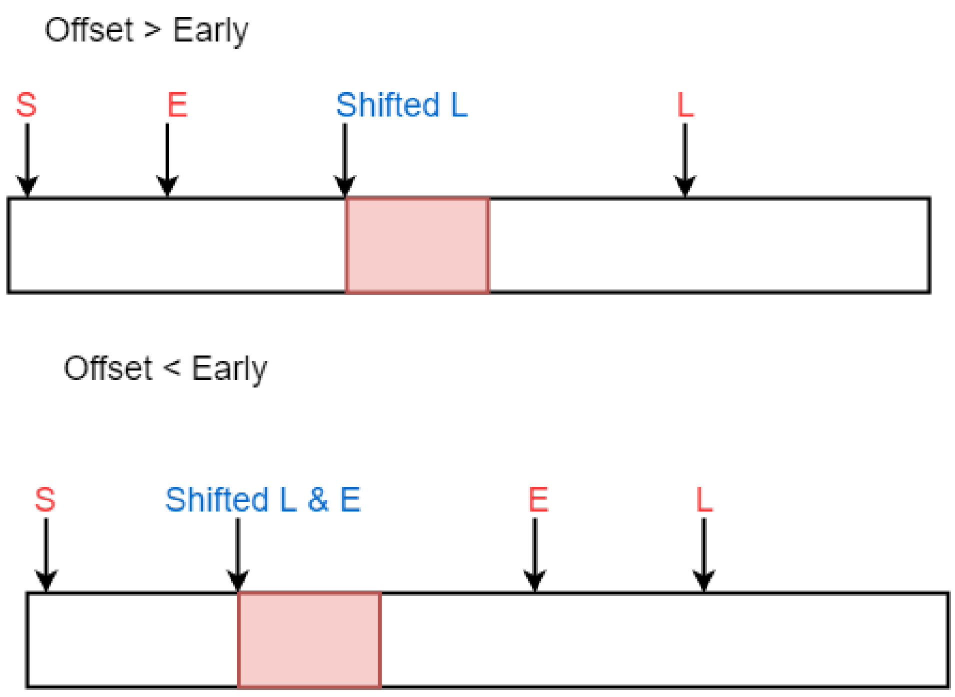


Figure 4.7: Modification of Reception Windows of upstream links

- If $Offset_{f_1} < Early_{f_1}$, then $Late_{f_2}$ and $Early_{f_2} = Offset_{f_1}$.

The flow on upstream links has to start latest by the offset in both cases. If flow, f starts later than offset then the packets may reach later than the offset value calculated for link 1. The early pointer for flow, f is moved to the offset value in the 2nd condition as to avoid further in-networking delay.

It has been depicted in figure 4.7.

- Downstream links: The links corresponding to the downstream of the flow have to be modified.
  - If $Offset_{f_1} > Early_{f_1}$, than $Start_{f_2}$ and $Early_{f_2} = Offset_{f_1}$.
  - If $Offset_{f_1} < Early_{f_1}$, than $Start_{f_2} = Offset_{f_1}$

Start pointer is moved to the offset in both cases as the flow, f can not be scheduled on the link, 2 before it has not been dispatched from the link,1. The early pointer is also moved to the offset when offset is greater than the early pointer to avoid the mandatory in-networking delay. It has been depicted in figure 4.8.

In the case where the offset value is equal to the early pointer, there is no need to modify the upstream or downstream reception windows. Also if the schedule for any link in the downstream or upstream path has already been computed, the modification for that

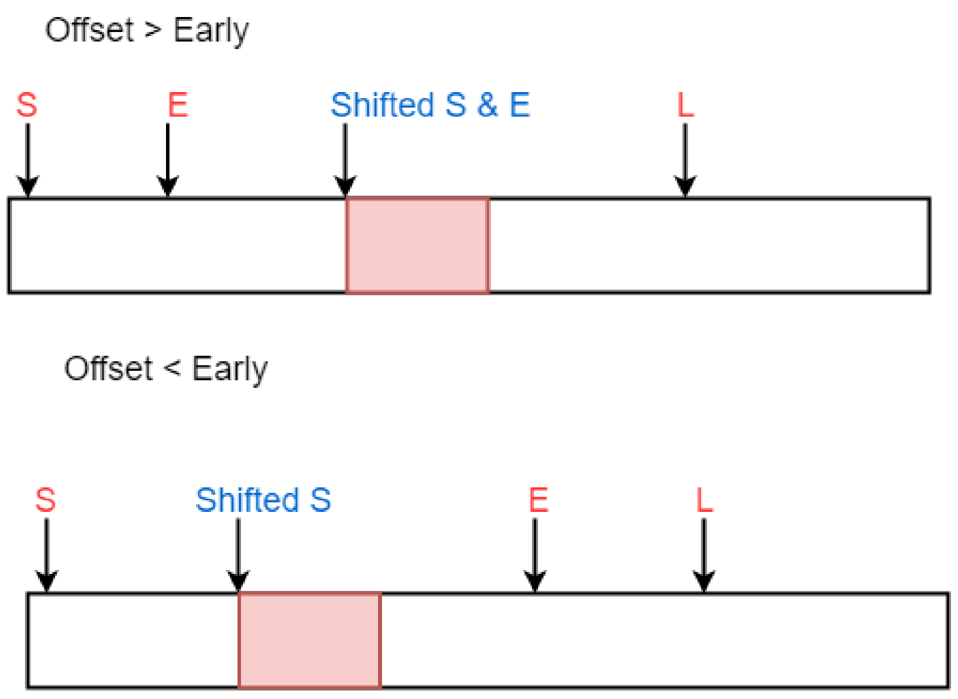


Figure 4.8: Modification of Reception Windows of Downstream links

link is skipped as it will interfere with the already calculated schedule. The schedule for the next link can start once all the reception windows have been modified in the flow path. When the schedule for all the links in topology has been computed, we get the total execution time for our divide-and-conquer scheduling algorithm.

4.6 Parallel Implementation Approach

The divide-and-conquer algorithm runtime can be further reduced using a parallel implementation approach. The basic idea is to run the ILP solver on multiple machines such that on each machine it computes a schedule for a different link in topology. It will reduce the overall runtime as we solve the problem on multiple processes.

We took a distributed cluster approach based on master and slave for our parallel implementation. One machine will act as master and all the other available machines will act as slaves. The master and slave communicate with each other using Transmission Control Protocol(TCP) protocol. We selected the TCP protocol as it is reliable, data integrity and the packets reach in-order from source to destination in large complex networks.

Algorithmus 4.6 Choose link for parallel implementation

```
1: $CurrentScheduledLinks \leftarrow [\,]$
2: $doneLinks \leftarrow [\,]$
3: $orderLinks \leftarrow$ Overlap or NumberFlows Links list
4: for all $link$ in $orderLinks$ do
5:     if $link$ in $CurrentScheduledLinks$ and in $doneLinks$ then
6:         continue
7:     else
8:         if check any $flow$ matches in $flowWindows[link]$ then
9:             $edge = link$
10:        end if
11:    end if
12: end for
13: return $(edge)$
```

Algorithmus 4.7 Slave Thread

```
1: $flowOrder \leftarrow [\,]$
2: while True do
3:     $link \leftarrow$ CHOOSE LINK
4:     if $link == edge$ from $CurrentScheduledLinks$ then
5:         continue
6:     else
7:         add $link$ in $CurrentScheduledLinks$
8:         SEND DATA TO MASTER($flowWindows[link]$, $flowOrder[link]$)
9:         $Offset$ ,$flowOrder \leftarrow$ RECEIVE DATA FROM MASTER
10:        remove $link$ from $CurrentScheduledLinks$
11:        add $link$ in $doneLinks$
12:        CHANGERECEPTIONWINDOW($link$ , $offset$)
13:    end if
14: end while
```

The information that is sent from master to slave includes the link, which is selected from algorithm 4.4 along with the associated flow windows (start, early and late pointers) of all flows. Master also send the flow order list as to maintain the flow order between any pair of flows. The slave calculates the schedule for that particular link and sent back the offset solution to the master with the updated flow order list. A distributed network system is shown in figure 4.9.

The duties of the master are

- Choose a link.

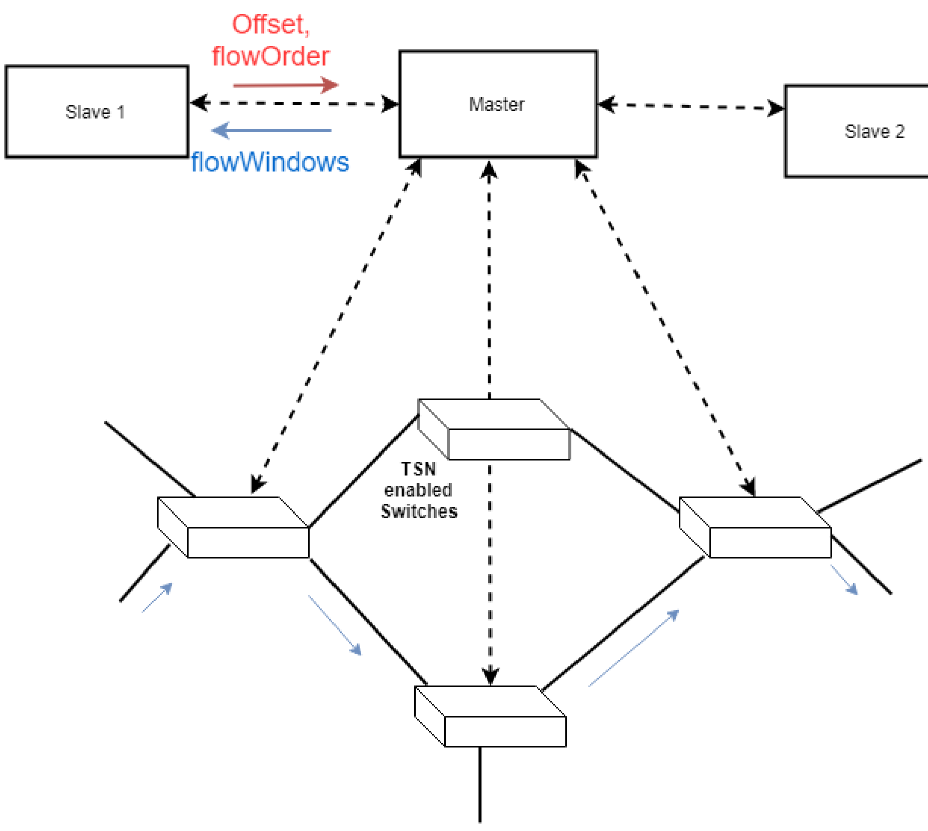


Figure 4.9: Distributed system with master and slaves

- Maintain a list of all the links currently in execution on other machines(slaves).
- Modify the upstream and downstream reception windows based on the offset solution received from slaves.
- Can compute a schedule of any other link itself on a separate thread.

In algorithm 4.6, the link is chosen such that it does not share any common flow with any other links which are currently in execution on other machines. If there are any common flows between 2 links, the reception windows for all the links in the flow path have to be modified and this results in double modification of windows without incorporating the previous changes. So, we must avoid the parallel execution of 2 links having any common flow.

In algorithm 4.7, a thread is run on the master known as slave thread and it connects to the slave.The first step is to choose the link from Alg 4.6. The link is added to the current executing links list, $CurrentScheduledLinks$. The thread sends the flow order and the flow windows list to slave. The slave algorithm is same as given in section 4.3.

It returns back the updated flow order and the offset values to the master. Finally, it modifies the reception windows using algorithm 4.5.

The total execution time is less than the single machine execution time as the problem is solved on multiple machines. The best case scenario is when no machine remains idle i.e. the utilization is 100 percent for all the machines. This is an ideal case and will result in an extremely fast runtime as compared to a single machine. The worst case scenario is when all the flows are common on all the links. This results in all the machines lying idle except for one machine which computes all the schedules for all the links. The runtime, in this case, will be same as that of a single machine. The general case will lie between best and worst case for any physical real-time networks. Also, in almost all our testing scenarios, we found that the increase in the number of slave machines will reduce the overall execution runtimes.

4.7 Heuristics Approach

We tried to further reduce the execution time of our scheduling algorithm. An ILP solver, such as CPLEX takes a lot of time to find an optimum solution for the divide-and-conquer algorithm in large and complex networks. We investigated that a heuristic approach is much better as compared to an ILP solver. On the one hand heuristics computes a faster solution than an ILP. On the other hand, the ILP gives better and optimum solution than heuristics.

The first-fit method algorithm is used as the heuristics algorithm as discussed in section 2.6. In order to come with the scheduling algorithm formulation, we have to map it to the first-fit method. The corresponding items to bins in a TSN network are the reception windows and we have to fit the flow schedule within the corresponding flow window. We have to tweak the first-fit method as the flow windows are not of equal size and we can't schedule the flow in any of the other flow windows.

In algorithm 4.8, we select the first flow with the minimum early pointer time in it's corresponding flow window. First, we try to schedule the flow at the early pointer. If it is not possible to schedule due to some overlap with any other flow windows, then it is scheduled as close to the right of early pointer but before it's late pointer i.e. in reception window. If further, there is no space left in the reception window. we try to schedule the flow as close to the left of early pointer but after the start pointer. We discard the flow if it can't be scheduled due to no space left in its flow window. The reason could be that the complete flow window space is occupied by the already scheduled flows.

It should be noted that once a schedule for a particular flow has been calculated, generally it is not adjusted due to the complexity of the algorithm. It is adjusted only

Algorithmus 4.8 Heuristics Approach

```
1: $offset \leftarrow \{\}$
2: for all $flow$ in $F$ do
3:     $(start,\ early,\ late,\ size) \leftarrow flowWindow[flow]$
4:     $flowPairs \leftarrow\ Select f1, f2$ from $flow$ inOrder
5: end for
6: $offset[$ first flow in flowPairs, $f] \leftarrow flowWindow[f][early]$
7: for all $f1$, $f2$ in $flowPairs$ do
8:     if $flowWindow[f1][early] + flowWindow[f1][size] < flowWindow[f2][early]$ then
9:         $offset[f2] \leftarrow flowWindow[f2][early]$
10:    else
11:        $offset[f2] \leftarrow offset[f1] + flowWindow[f1][size]$
12:        if $offset[f2] > flowWindow[f2][late]$ then
13:            if $offset[f1] \leq flowWindow[f2][late]$ and $offset[f1] + flowWindow[f2][size] \leq flowWindow[f1][late]$ then
14:                $offset[f2] \leftarrow offset[f1]$
15:                $offset[f1] \leftarrow offset[f2] + flowWindow[f2][size]$
16:            else
17:                try to schedule the flow,$f2$ between $flowWindow[f2][start]$ and $flowWindow[f2][early]$
18:            end if
19:        end if
20:    end if
21: end for
```

when the next corresponding flow i.e. the flow with the nearest early pointer can't be schedule to the right of it's early pointer. The objective is to avoid the schedule of a flow between the start and early pointer which reduces the mandatory in-networking delay. For example, 2 flows, f_1 and f_2 are lined up to schedule next to each other such that, $Early_{f_1} < Early_{f_2}$. The schedule for the flow, f_1 has already been computed and the schedule for flow, f_2 can't be scheduled to the right of the early pointer as it overlaps with the schedule of flow, $f_!$. The approach is to schedule the flow,f_2 first and then try to see if it is possible to schedule the flow, f_1 after f_1 such that it satisfies it's boundary constraints i.e. the flow offset is before it's late pointer. If it is possible to schedule f_2 before f_1, the new offsets of the two flows are:

$$offset_{f_2} \leftarrow offset_{f_1}$$

$$offset_{f_1} \leftarrow offset_{f_2} + Size_{f_2}$$

The advantage of this method as compared to ILP is that it allows very fast runtimes. Our approach to heuristics is a simple one and it takes only one iteration to compute all the flow schedules on a single link. The runtime for heuristics algorithm doesn't increase much and almost remains constant with the increase in the number of flows.

5 Results and Evaluations

In this chapter, we present the results of the evaluations done for the scheduling algorithms presented in chapter 4. We measured it against the following criteria : the impact of the divide-and-conquer algorithm on the runtime and the optimality of the solution, the impact of the link choosing algorithm on the runtime, the impact of the parallel implementation approach on the runtime, the scalability of the divide-and-conquer algorithm, the impact of the heuristic algorithm on the runtime and compare the ILP and heuristics scheduling algorithms on the basis of runtimes and optimality of solution. We evaluated our solution in various scenarios having randomized topologies with a set of time-sensitive network flows randomly generated.

5.1 Link Choosing Evaluations

As discussed in section 4.4, a pivot link is needed for our divide-and-conquer algorithm. We have analyzed two methods to choose a link. The methods are overlap between reception windows and the total number of flows scheduled on a flow. It forms the base on which we evaluate our divide-and-conquer algorithm. As choosing a link can determine the runtimes for the algorithm and whether a solution is feasible or not for all the flows in the network.

To evaluate the effectiveness of the overlap method, we analyze the total effective overlap on a link with respect to the ILP execution time. We executed our evaluations for different scenarios with a varying number of flows and varying size of the effective overlap.

We solved the scheduling problem using CPLEX [5], which is an ILP solver. It is a commercial optimization software package developed by IBM. It is a flexible, high-performance mathematical programming solver for linear programming problems. CPLEX provides some parameters to terminate an optimization problem early and obtain the best possible solution computed till that time based on different parameters. The parameters can be deterministic time limit which is to set time limit bounds to our problem, relative MIP gap tolerance is to set the relative tolerance on the gap between the best integer objective and the objective of the best node remaining. We evaluate some of the high complexity problems which take days to compute to use the above-mentioned parameters to reduce the time.

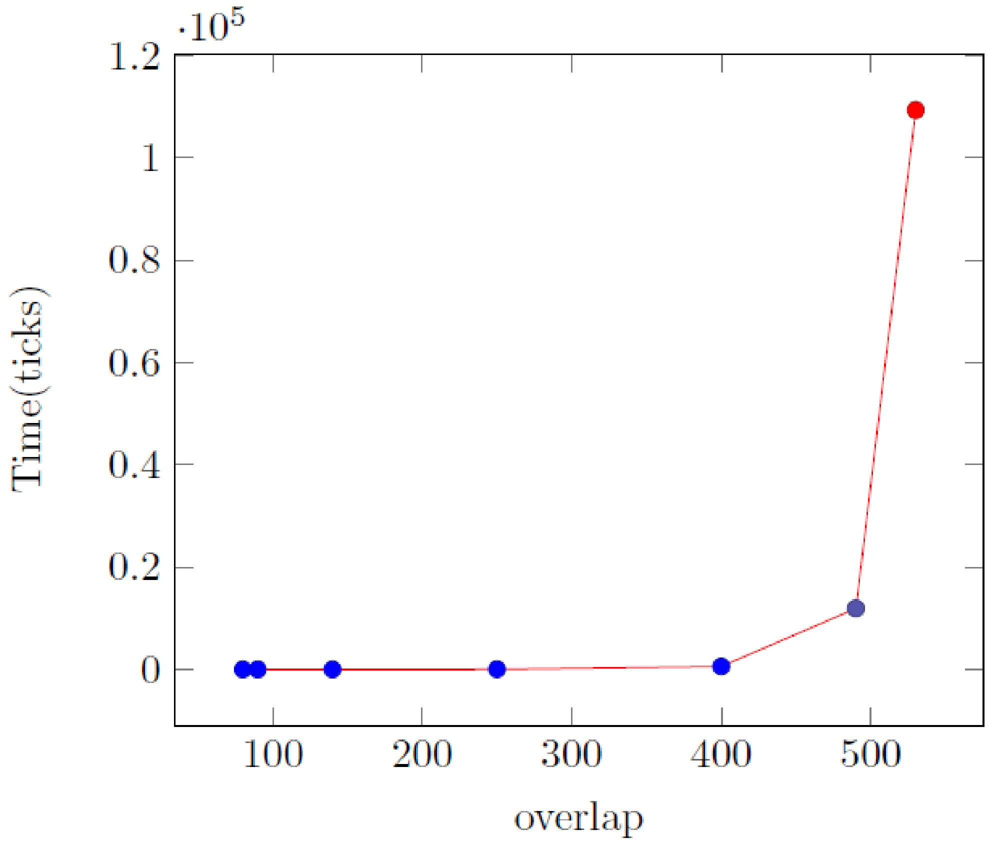


Figure 5.1: Total Time(mSecs) to schedule all flows on a link vs effective overlap

Evaluations of the effective overlap with respect to execution time is shown in figure 5.1. For our testing, we increase the overlap from around 100 to 500. This implies that the scheduling time for small overlap remains almost constant but increases rapidly after it reaches a certain amount. The scheduling runtime increases exponentially with the increase of the overlap. The overlap algorithm mentioned in 4.4 doesn't specify how to compute a particular effective overlap value. We change the effective overlap size by changing the flow windows of the flows. It can be concluded from the graph that there is a direct relationship between the overlap and the execution time. It gives us a starting point for the selection of the pivot link which is crucial for further evaluations.

Evaluation of the change in the number of flows with respect to execution time is shown in figure 5.2. It has to be noted that the overlap remains constant around 200, the evaluations suggest that the execution times almost remains the same. This implies that the change in the number of flows doesn't necessarily mean that the execution time

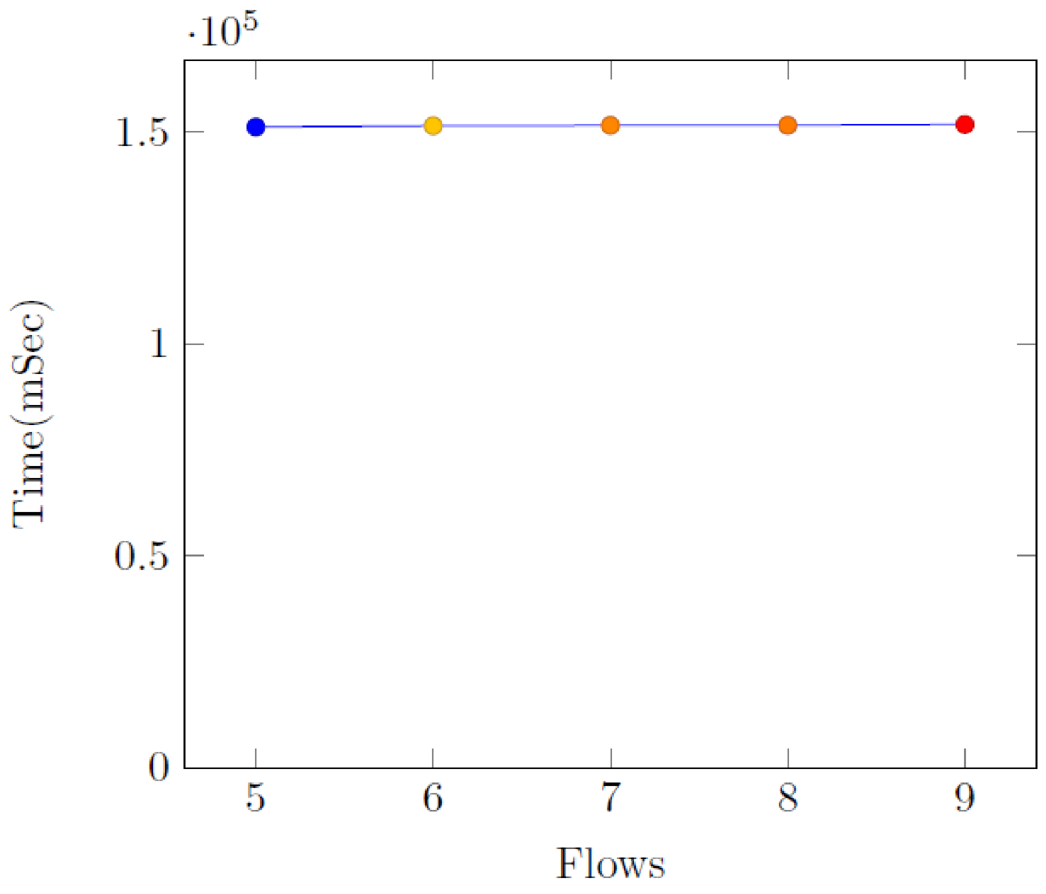


Figure 5.2: Total Time(mSecs) to schedule all flows on a link w.r.t number of flows with a fixed overlap of 200

will also change. It can be concluded that there is no effective relationship between the number of flows with respect to the execution time of the ILP solver.

It might create an impression that there is a relationship between the number of flows and runtimes as shown in figure 5.3. All the flows in the graph have same windows span as shown in table 5.1 i.e. it creates the same overlap between any two adjacent flows. If we observed carefully, the increase in runtimes is due to increase in overlap and not due to the increase in the number of flows.

The effective overlap method is a more effective and reliable method than the number of flows to choose the pivot link. In our multiple evaluations with a different number of flows and different effective overlap, the results indicate that there is a correlation between the overlap between reception windows and the ILP runtimes. It might vary slightly in some random scenarios, a one we will discuss in the next section 5.2.

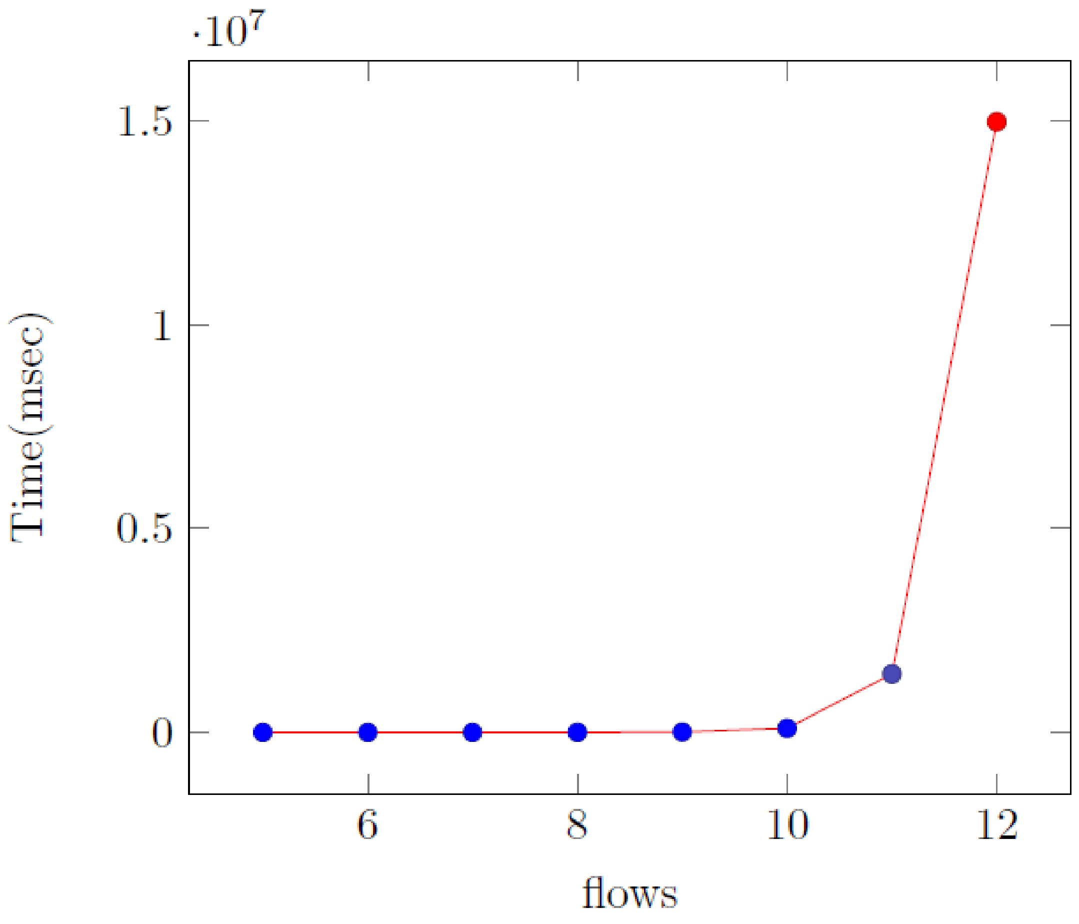


Figure 5.3: Total Time(mSecs) to schedule all flows w.r.t number of flows with same window span

5.2 Impact of Scheduling Algorithm on Runtime

The scheduling algorithm discussed in Chapter 4 is to compute the transmission schedules in a time-sensitive network using a divide-and-conquer approach. In this section, we evaluate and discuss the impact of the algorithm with respect to the number of flows transmitted in the network and the graph connectivity. We used the Erdos- Rényi(ER) graph [2] methods to generate different network models. The ER graph model generate a graph with the help of some parameters. The parameters are the number of nodes (n), a random number generator (seed), a probability to create an edge(p). The graph connectivity of a network is represented by the parameter probability for edge creation (p) . It indicates the probability of a connection existing between a pair of nodes. Higher

Flows	**Start**	**Early**	**Late**	**Size**
1	100	200	5000	30
2	120	220	5020	30
3	140	240	5040	30
4	160	260	5060	30
5	180	280	5080	30
6	200	300	5100	30
7	220	320	5120	30
8	240	340	5140	30
9	260	360	5160	30
10	280	380	5180	30
11	300	400	5200	30
12	320	420	5220	30

Table 5.1: Flow Windows for different links

the value of graph connectivity, higher is the number of links in the graph. We used different connectivity, number of switches and hosts for our evaluations.

In our initial testing, we found that in complex network, CPLEX runs for few days to compute a schedule on a single link. So, we set time-bounds and mip tolerance gap on CPLEX to terminate after a reasonable amount of time that provides the best solution available at that time for our evaluations.

5.2.1 ILP solver vs Runtime

We executed our evaluations using ILP solver in multiple scenarios with different number of flows in a network. The topology we have used for our evaluations contains 5 switches, 50 hosts with p = 0.98 which generates 120 links in network.

Evaluations of the resulting ILP runtimes with 150 flows routed in the network as shown in figure 5.4. We generated flows with random windows with a fair amount of overlap on most of the links. All the 120 links in the network are plotted against the runtimes(in milliseconds) in the graph. The links are selected in the decreasing order of the effective overlap as discussed in the previous section. The leftmost point on the x-axis indicates the link with the highest overlap. It can be made more clear as we have plotted another graph in figure 5.5 of the effective overlap with respect to ILP runtimes for all the 120 links. The decrease in the effective overlap indicates almost a consistent decrease in the runtimes except for the execution time of the first link, which is slightly less than

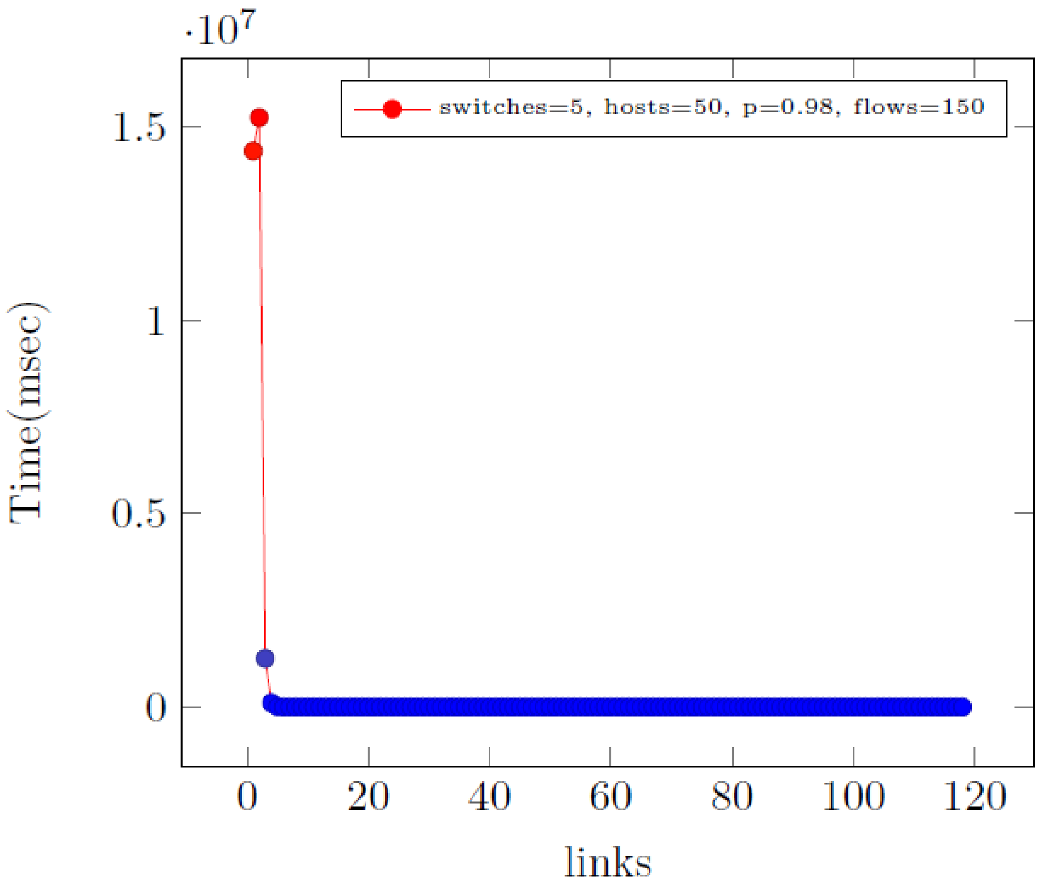


Figure 5.4: Total Time(mSecs) to schedule 150 flows in a network

the 2nd link. It might be that the algorithm discussed in section 4.4 is not full proof but the method to compute the effective overlap is the best approach we can found to choose the pivot link. The maximum ILP runtime to schedule 150 flows in the given scenario on a particular link is around $1.5 * 10^7$ mSecs which is approximately 4 hours. The corresponding overlap is $8 * 10^4$ bytes.

Similarly, we schedule 180 flows in the same network as shown in figure 5.6. It is comparable to the evaluations for 150 flows as it also shows a exponential decrease in the ILP runtimes. The link with the highest effective overlap is selected first and is on the leftmost side of the graph. We didn't see any anomaly in 180 flows as compared to 150 flows. The maximum ILP runtime to schedule 180 flows in the given scenario on a particular link is around $7 * 10^7$ mSecs which is approximately 19.5 hours.

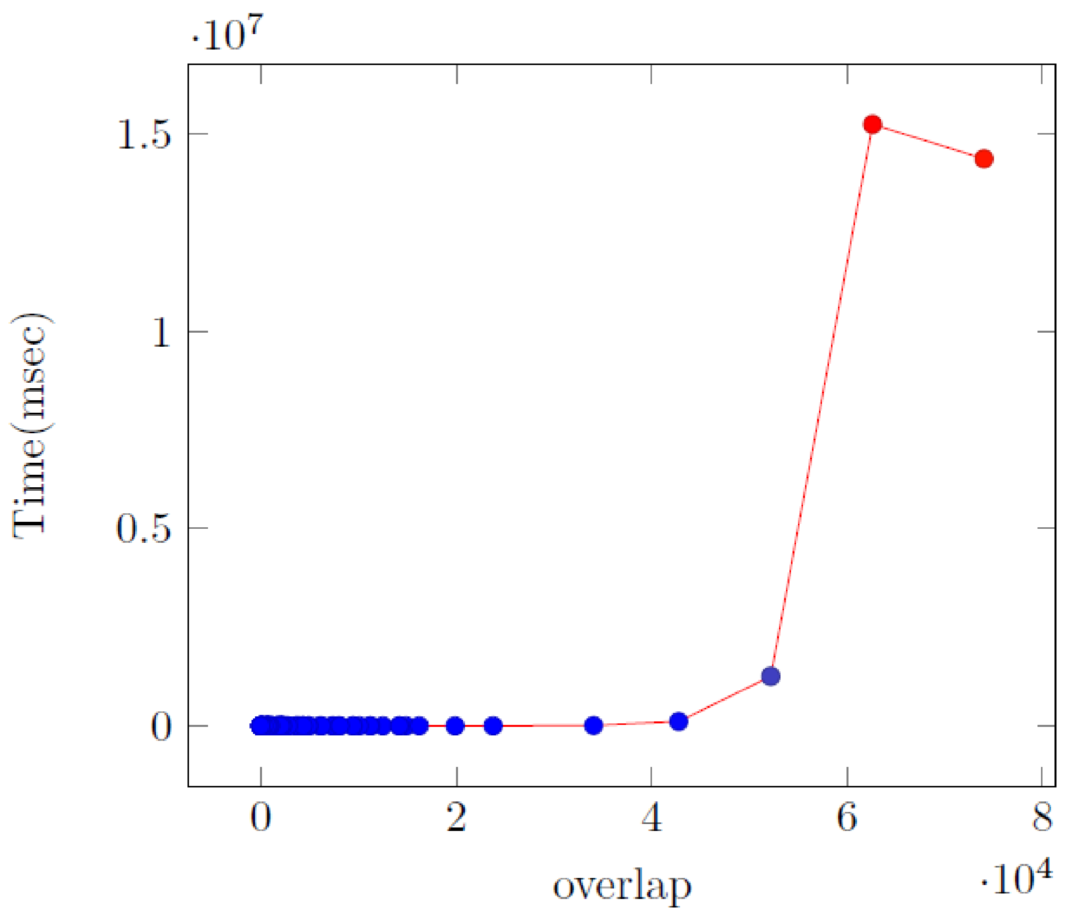


Figure 5.5: Total Time(mSecs) to schedule 150 flows in a network

It is observed that in both cases with 150 and 180 flows, the ILP execution time for the links having low overlap almost remains constant as it can schedule the flows at the early pointer or as close to the early pointer.

5.3 Parallel Implementation Evaluations

To evaluate the parallel implementation with respect to overall ILP runtimes and utilization, we execute it on multiple machines. The distributed approach is used where one machine acts as a master and other machines as slaves. For our evaluations, we used multiprocessor machines (Intel(R) Xeon(R) CPU E5-1650 v3 @ 3.50GHz) with 2 X 6 cores having 16 GB of memory. It has to be noted that the evaluations were done on maximum 3 machines due to the limited amount of resources available. The topology we

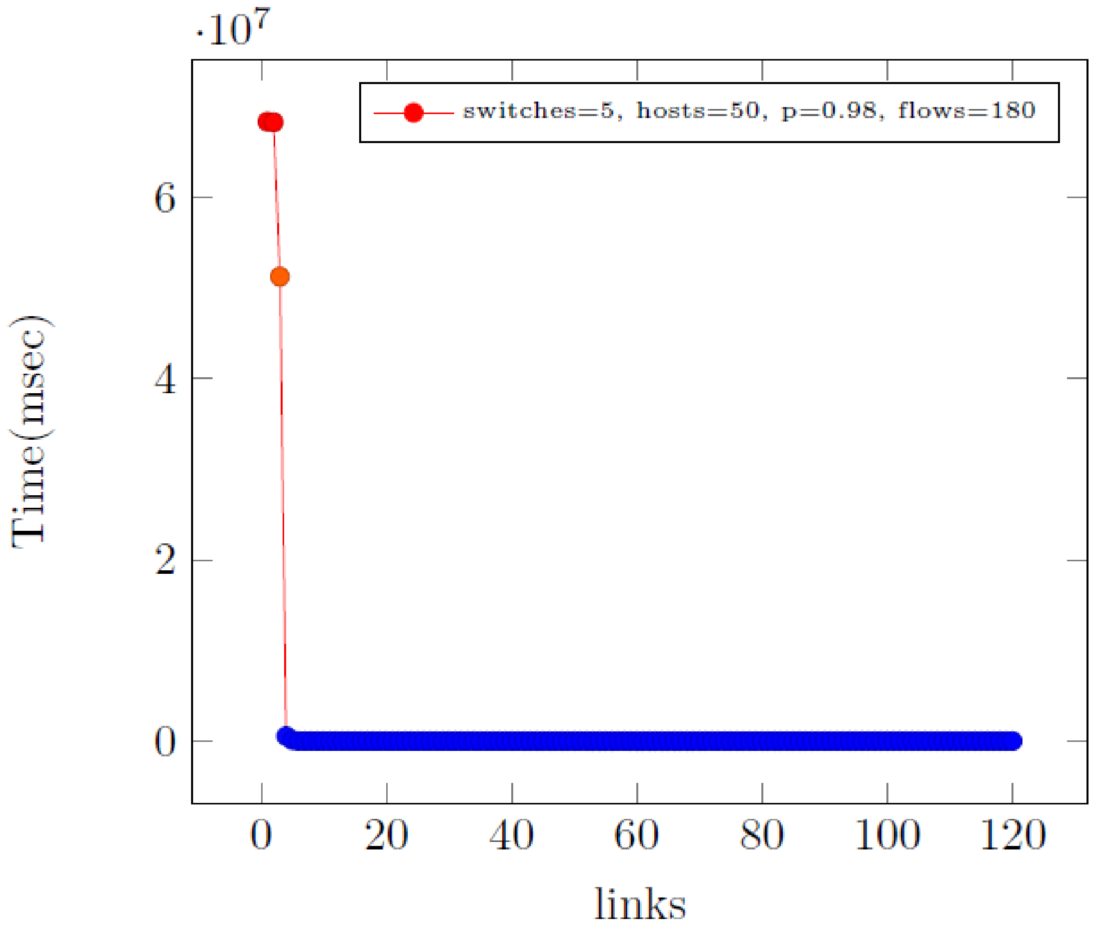


Figure 5.6: Total Time(mSecs) to schedule 180 flows in a network

have used for our evaluations contains 6 switches, 50 hosts with p = 0.98. Evaluations are done for different scenarios with 100,150 and 180 flows routed in the network.

Evaluation of the total execution time to compute the schedule for 100 flows on all the links with respect to the number of machines is shown in figure 5.7. The total runtime decreases with the increase in machines as shown in the graph. The decrease of runtimes from using 1 machine to 2 machines is significantly higher from using 2 machines to 3 machines. The overall reduction in runtimes from 1 machine to 2 machines is approx. 45 %, whereas the reduction in runtimes from 2 to 3 machines is approx. 12 %. There is not much incentive to evaluate it on 3 machines as the decrease in runtimes from 2 machines to 3 machines is not significantly large.

The utilization of a machine is calculated by dividing the total time a machine remains in use (when the ILP is running), required for the whole topology. A machine remains idle

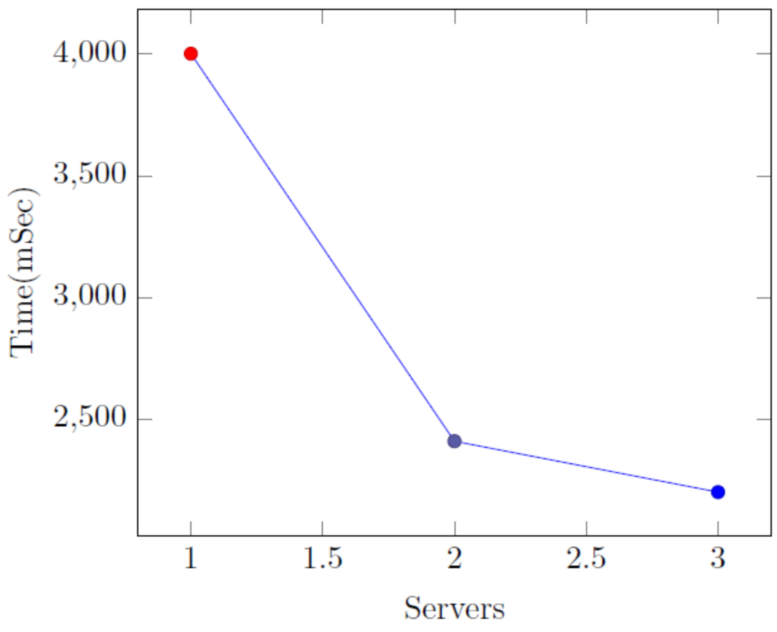


Figure 5.7: Total Time(mSecs) to schedule 100 flows on different number of machines

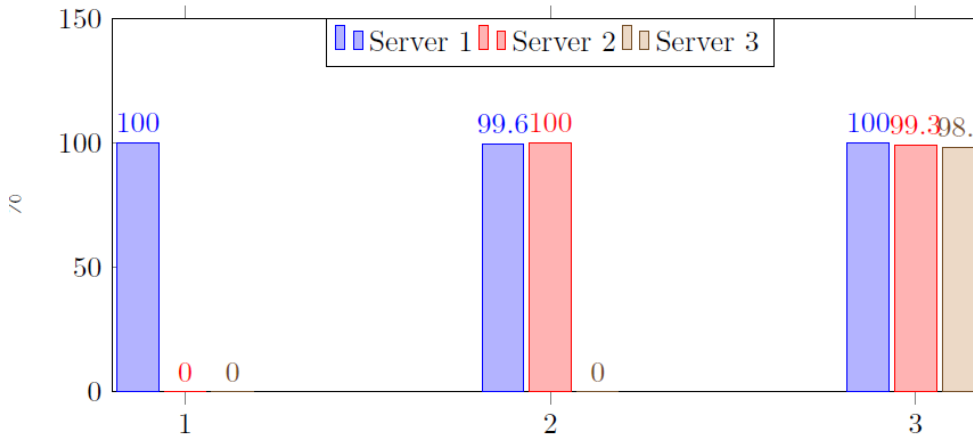


Figure 5.8: The utilization percentage of the 3 machines while scheduling 100 flows on 1 ,2 and 3 machines respectively

when the slave is waiting for the next link but the master can't find any link to choose for the scheduling. It is because all the remaining unscheduled links have some common flows with the links that are executed in parallel on other slaves. The master constantly searches for a link to schedule until it finds a link which doesn't have any common flow with any other currently scheduled links. Analyzing the utilization of machines helps us to decide whether an extra machine is required or not. If utilization of any machine is quite low, it is hard to justify to deploy an extra machine.

The utilization percentage of the 3 machines while scheduling 100 flows is shown in figure 5.8. The utilization when only 1 machine is used to schedule is always 100 % as a single machine has to execute all the links. It was observed that the utilization remains around 100% for both machines 2 and 3 when using all the 3 machines. The runtime is in order of few milliseconds for 100 flows, so the idle time is almost negligible for all the machines and it results in 100% utilization.

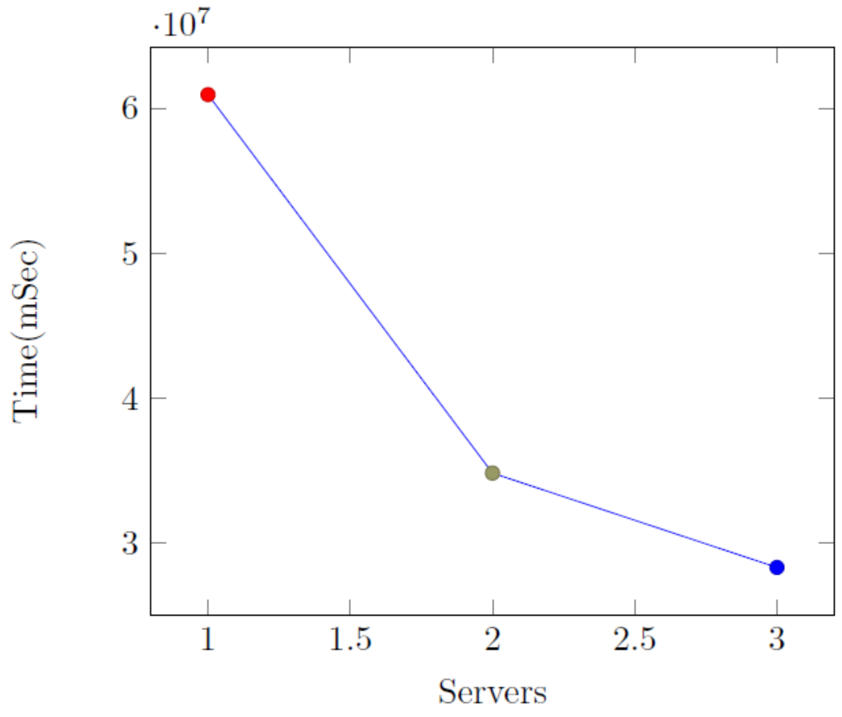


Figure 5.9: Total Time(mSecs) to schedule 180 flows on different number of machines

Similarly, the evaluations are run for 150 and 180 flows as shown in figures 5.9 and 5.11 respectively. The evaluations for 180 flows is as expected in real-world scenarios and is similar to the evaluations of the 100 flows. The increase in the number of machines results in the decrease of the ILP execution runtimes. The overall reduction in runtimes from 1 machine to 2 machines is approx. 44 %, whereas the reduction in runtimes from 2 to 3 machines is approx. 20 %. In terms of actual runtime, it decreases from 9.2 hours to 7.7 hours, saving almost 1.5 hours.

The utilization percentage of the 3 machines for scheduling 180 flows is shown in figure 5.10. The utilization percentage for one of the machines is significantly higher as compared to other machines when utilizing 2 or 3 machines. It is expected as the

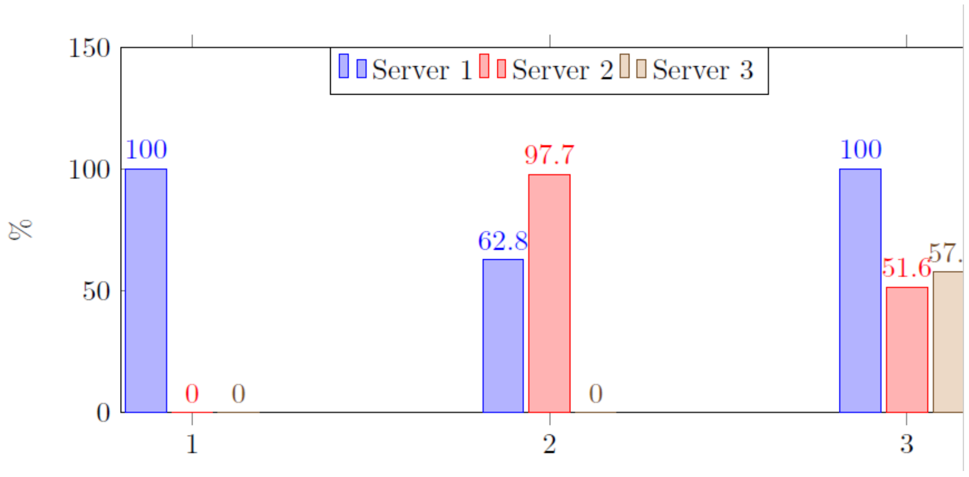


Figure 5.10: The utilization percentage of the 3 machines while scheduling 180 flows on 1 ,2 and 3 machines respectively

runtime to compute a schedule for a single link is in order of few hours, and therefore the other machines might remain idle for long period of time.

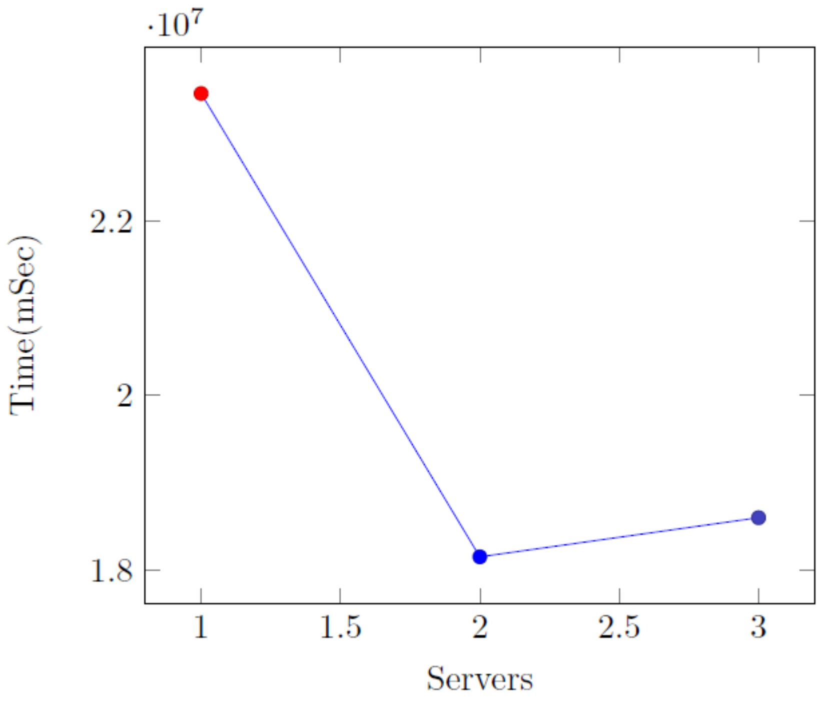


Figure 5.11: Total Time(mSecs) to schedule 150 flows on different number of machines

We observed that there is a deviation from expected behavior while scheduling 150 flows on 3 machines as can be seen in figure 5.11. The time to schedule 150 flows

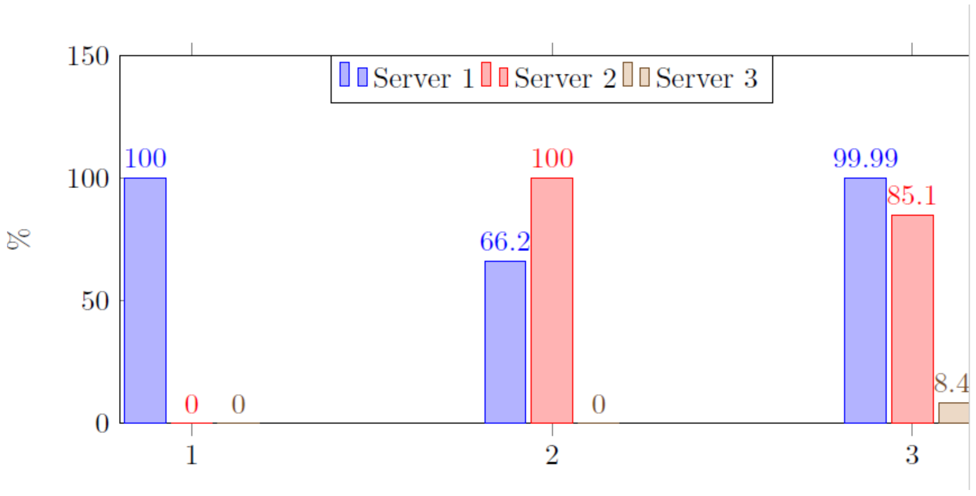


Figure 5.12: The utilization percentage of the 3 machines while scheduling 150 flows on 1 ,2 and 3 machines respectively

increases while going from 2 to 3 machines instead of the decreasing behavior we expected. The increase is only around 3% but still the overall runtime increases. The reasons for the above anomaly can't be properly analyzed. One of the expected reason is that the sequence of links which is selected by the master to compute the schedule is quite different in both scenarios and it might somehow lead to higher runtimes in case of 2 machines. This is just a hypothesis, we have to do more extensive research work to determine the reason for this unexpected behavior.

The utilization percentage of the 3 machines for scheduling 150 flows is shown in figure 5.12. The utilization percentage for one of the machines is 8.4% which is significantly low as compared to other machines when using 3 machines. It can be concluded that adding a 3rd machine will not reduce the runtime significantly but it might have a negative impact as we observed while scheduling 150 flows on 3 machines.

5.4 Scalability Evaluations

To evaluate the scalability, we measured the runtimes for a various number of flows in 2 scenarios. The first is to schedule on a single machine and the second is to schedule on 2 machines using parallel implementation. The evaluations of runtimes with respect to the different number of flows can be summarized in a single graph as shown in figure 5.13. It has to be noted that for this particular scenario, we didn't use any CPLEX termination parameters and the flow windows generated are different from the section 5.3. It compares the runtimes when running the algorithm on 1 machine or 2 machines

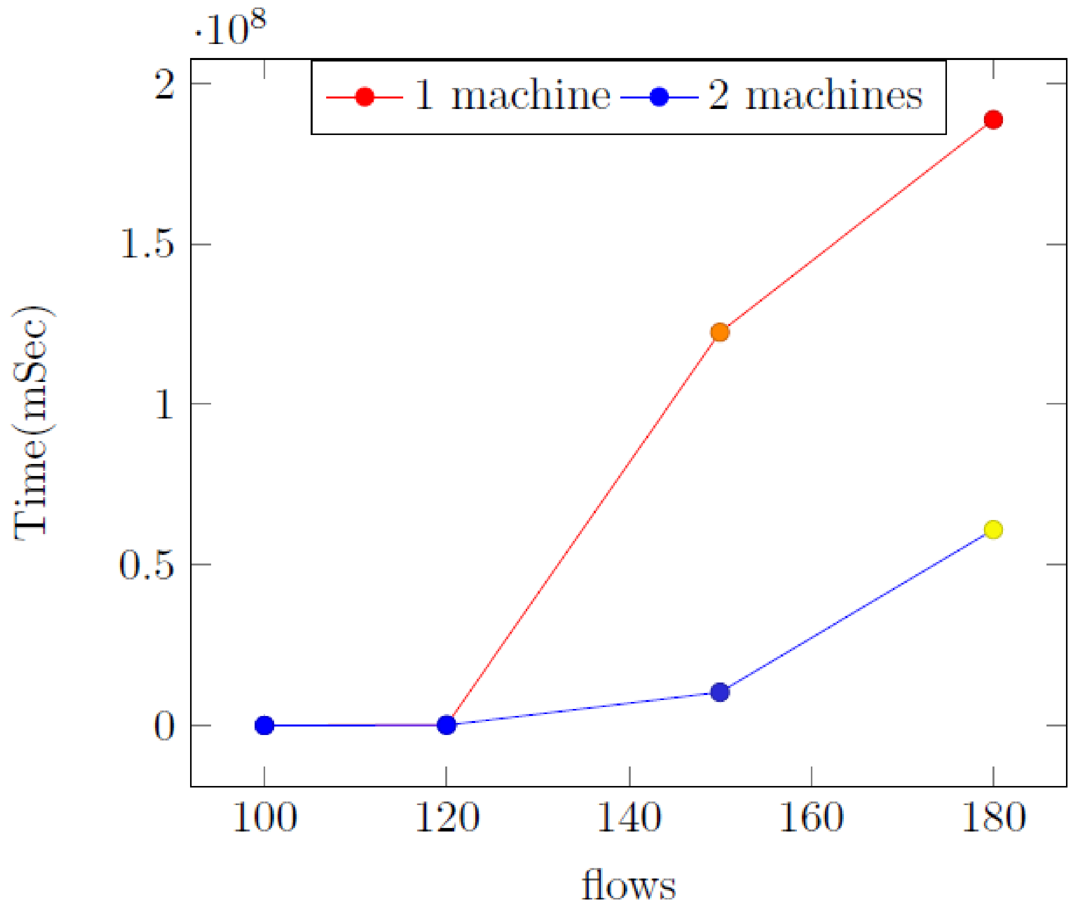


Figure 5.13: The runtimes w.r.t to flows

while changing the total number of flows routed in the network. The runtimes are almost identical for 100 and 120 flows when using 1 machine or 2 machines. The runtime reduces significantly for 150 and 180 flows in case of 180 flows as much as 75% in case of 180 flows when using 2 machines as compared to 1 machine. The reduction The evaluations shows that the higher the number of flows to be scheduled in the network, the chances of the overall runtime to decrease increases when using multiple machines. Thus, scalability improves when using more than a single machine for large networks.

5.5 Heuristic Algorithm Evaluations

The evaluations for the heuristic algorithm as compared to ILP for a single link is shown in figure 5.14. It has to be noted that in this case, with the increase in the number of

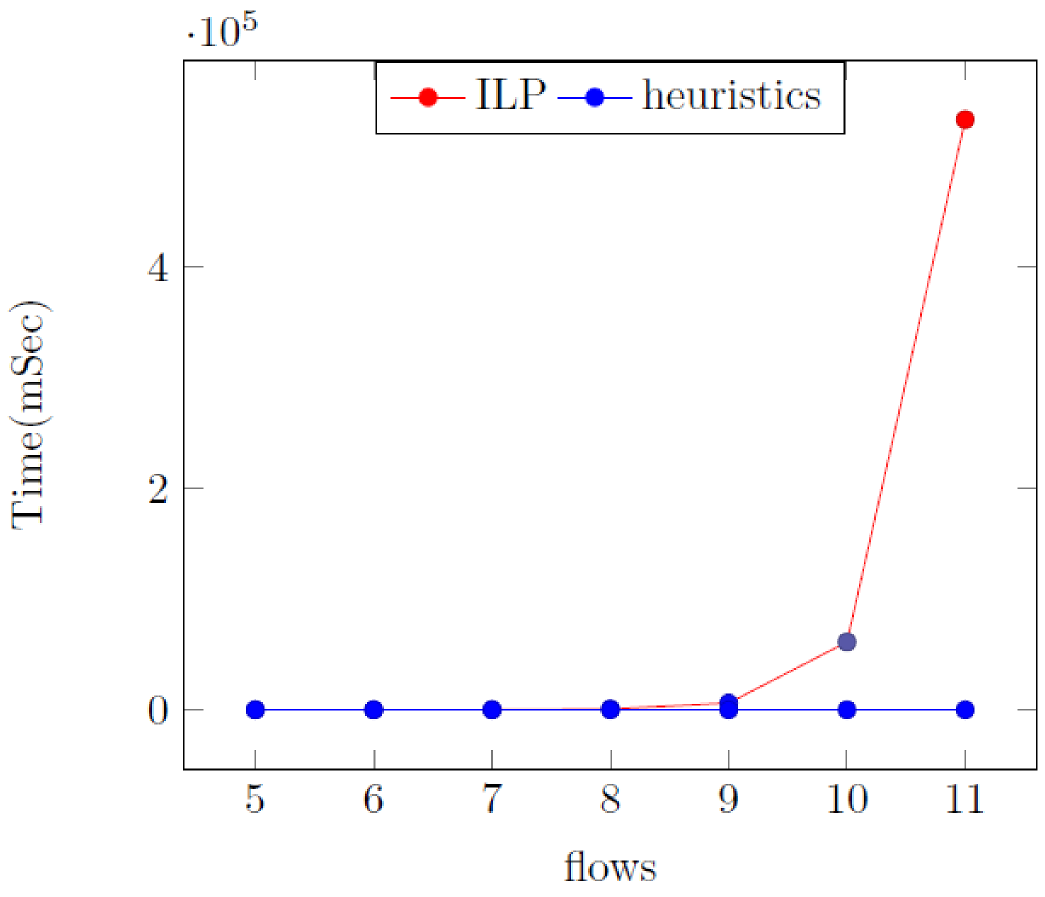


Figure 5.14: Runtime vs number of flows with respect to ILP and heuristics

flows the effective overlap also increases. For smaller overlap (< 9 flows), the scheduling time for both ILP and heuristics is same and almost negligible. The scheduling for ILP takes a longer time as compared to heuristics when overlap increases significantly, whereas runtime for heuristics is in few milliseconds and remains constant.

5.6 Optimality Evaluations for ILP and Heuristics

The optimality of the heuristics and ILP approach is compared in figure 5.15. The optimality is defined by the summation of the difference between the computed offset and the respective early pointer for all the flows, $\sum_{\forall f \in F} |offset_f - early_f|$. For smaller flows (smaller overlap), the difference between the optimality of the two approaches is not too significant, but still, ILP gives better optimum results. With the increase in flows (overlap increases), the gap between ILP and heuristics widens. The heuristics approach

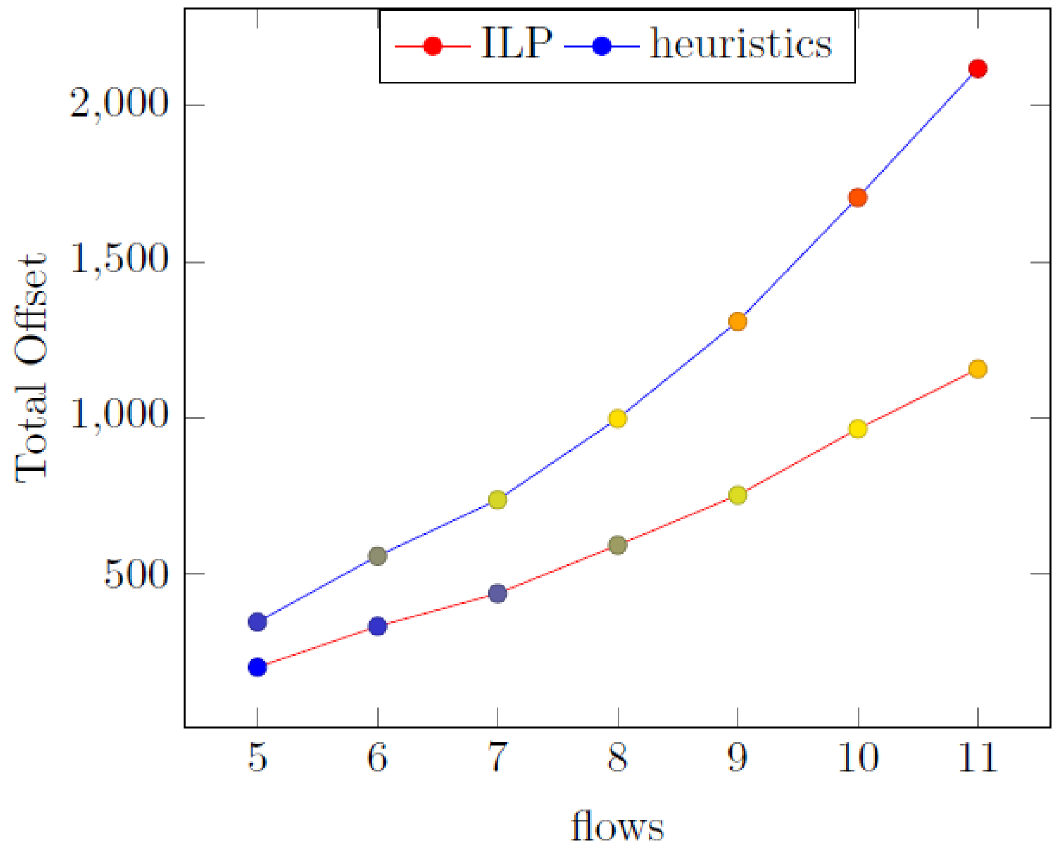


Figure 5.15: Offset vs number of flows with respect to ILP and heuristics

that has been implemented can't find optimum solutions as close to ILP and the gap widens when the overlap increases. Therefore, it is a trade-off between the scheduling execution time and the optimality for the two algorithms. For small overlap, one can use the heuristics approach as it gives a solution as near to ILP but with a significant reduction in computation time. When the overlaps start to increase and one can't justify the gap between the two solutions due to the requirement of lower latency bounds in a time-sensitive network, it is better to use an ILP approach to compute the scheduling problem.

6 Conclusion

In the thesis, we presented the problem of scheduling in a Time-Sensitive Network(TSN) and motivated the need to present a scheduling algorithm to transport time-triggered traffic based on a divide-and-conquer approach in order to reduce the runtimes. Our objective is to reduce the high runtimes required for centralized approaches in large networks even if we have to compromise on the optimality of the solution. We also presented a simple and naive heuristics approach as an alternative to ILP to formulate the scheduling problem.

Our contributions include a divide-and-conquer approach with the modification of the reception windows. The evaluations done in the thesis indicates that the total runtime to compute a schedule on a link depends on the total effective overlap but not on the number of flows. This behavior was mostly consistent in multiple scenarios. The scalability we further reduced the runtimes using a parallel implementation and executed the divide-and-conquer approach on multiple machines simultaneously. The evaluations show that the runtime reduces significantly when using more than a single machine and in some scenarios, it reduces as much as 75%. It can be concluded that the parallel implementation approach is highly scalable for a network having a large number of flows. One of the limitations of parallel implementation is that if we keep on adding a machine for computation, the decrease in runtimes will not reduce as much as it reduced before. It is a trade-off between the cost of running an extra machine and the time one can save while scheduling a large number of flows in a complex network.

The evaluations of our heuristic algorithm indicate that it is highly scalable in any large network due to negligible runtimes(in order of few milliseconds). The runtime is significantly less as compared to ILP based approach(in order of few hours or days) for large networks. A drawback of heuristics approach is that the optimality decreases with the increase in the effective overlap, whereas an ILP computes a better optimum solution in all scenarios.

In our future work, we can further extend the parallel implementation of our divide-and-conquer based algorithm on more than 3 machines and see if there is any significant decrease in scheduling runtimes. The heuristic approach we have presented is a basic approach and the evaluations suggest that it has the scope to improve it's optimality while retaining the advantage of low runtimes. Hence, we could expand the heuristics algorithm further to reduce the optimality gap with the ILP based solution.

Bibliography

[1] K. Academy. *Divide and conquer algorithms*. URL: https://www.khanacademy.org/computing/computer-science/algorithms/merge-sort/a/divide-and-conquer-algorithms (cit. on p. 17).

[2] P. S. Aric Hagberg Dan Schult. *NetworkX Reference*. URL: https://networkx.github.io/documentation/latest/_downloads/networkx_reference.pdf (cit. on p. 66).

[3] Christiane Bangert. *Ethernet AVB - Audio Video Bridgings*. 2016. URL: https://www.professional-system.de/basics/avb-audio-video-bridging/ (cit. on p. 22).

[4] Cisco Public. *Cisco Supports Time-Sensitive Networking for Critical Industrial Applications*. URL: https://www.cisco.com/c/dam/en/us/solutions/collateral/industry-solutions/at-a-glance-c45-738823.pdf (cit. on p. 21).

[5] *CPLEX optimizer*. URL: https://www.ibm.com/analytics/data-science/prescriptive-analytics/cplex-optimizer (cit. on p. 63).

[6] S. S. Craciunas, R. S. Oliver, M. Chmelik, W. Steiner. *Scheduling real-time communication in IEEE 802.1 Qbv time sensitive networks*. ACM, 2016 (cit. on pp. 16, 28).

[7] F. Dürr, N. G. Nayak. *No-wait packet scheduling for IEEE time-sensitive networks (TSN)*. ACM, 2016 (cit. on pp. 16, 31).

[8] P. D. George A. Ditzel III. *Time Sensitive Network (TSN) Protocols and use in EtherNet/IP Systems*. 2015. URL: https://www.odva.org/Portals/0/Library/Conference/2015_ODVA_Conference_Ditzel-Didier_TSN.pdf (cit. on p. 35).

[9] R. Hummen. *What is TSN? A Look at Its Role in Future Ethernet Networks*. Feb. 2017. URL: http://www.belden.com/blog/industrialethernet/what-is-tsn-a-look-at-its-role-in-future-ethernet-networks.cfm (cit. on p. 13).

[10] H. Laine. *Simulating Time-Sensitive Networking*. 2015. URL: http://www.imm.dtu.dk/~paupo/publications/Laine2015aa-Simulating%20Time-Sensitive%20Netw-.pdf (cit. on pp. 24, 27).

[11] U. Modai. *Ethernet Evolves Again To Meet The Internet Of Things*. URL: http://www.electronicdesign.com/communications/ethernet-evolves-again-meet-internet-things (cit. on p. 14).

[12] S. Singh. *Routing Algorithms for Time Sensitive Networks*. 2017 (cit. on p. 31).

[13] Thomas Cormen and Devid Balkcom. *Divide and conquer algorithms*. 2017. URL: https://www.khanacademy.org/computing/computer-science/algorithms/merge-sort/a/divide-and-conquer-algorithms (cit. on p. 17).

[14] TTTech. *IEEE TSN (Time-Sensitive Networking):A Deterministic Ethernet Standard*. URL: http://www.hit.bme.hu/~jakab/edu/litr/TimeSensNet/TTTech_Time-Sensitive_Networking_Whitepaper.pdf (cit. on p. 38).

[15] E. W. Weisstein. *Bin-Packing Problem*. URL: http://mathworld.wolfram.com/Bin-PackingProblem.html (cit. on p. 34).

[16] P. S. Zdenek Hanzelek Pavel Burger. *Profinet IO IRT Message Scheduling* (cit. on p. 14).

All links were last followed on March 17, 2008.

Declaration