

Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Bachelorarbeit

Joint Routing and Scheduling with SMT

Benjamin Caddell

Course of Study: Informatik
Examiner: Prof. Dr. Kurt Rothermel
Supervisor: Jonathan Falk

Commenced: December 22, 2017
Completed: June 22, 2018

Abstract

Real-time communication over Ethernet networks are of growing importance for industrial applications. With the newly developed IEEE 802.1Qbv standard, new tools for guaranteeing timely boundaries for communication are available. While this standard specifies the mechanisms in place for guaranteeing the timeliness of transmissions, it falls short of defining algorithms to compute routes and schedules. Traditionally the routing problem and scheduling problem have been tackled independently. However these problems are strongly connected and solving them independently excludes feasible solutions. We provide an SMT formulation of the joint routing and scheduling problem. We implement this formulation to solve various different scheduling problems. We showcase and compare our SMT-Based solution to a similar ILP-Based solution.

Contents

1	Introduction	15
2	Related Work	17
3	Background	19
3.1	Time Sensitive Networks	19
4	Problem Solving Frameworks	21
4.1	SMT	21
4.2	ILP	22
5	System Model	23
6	Problem Statement	25
7	Design and Implementation	27
7.1	Design	27
7.2	Constraints	28
7.3	Choice of SMT-Solver	33
7.4	Program flow	33
7.5	Implementation	34
8	Evaluation	35
8.1	Evaluation Setup	35
8.2	SMT-Evaluation	36
8.3	Comparison ILP-SMT	40
9	Conclusion	41
10	Outlook	43
	Bibliography	45

List of Figures

3.1	IEEE 802.1Qbv switch Architecture	20
8.1	Evaluation Boxplotgraph Erdos Time-Size Large	36
8.2	Evaluation Boxplotgraph Erdos Time-Flows Large	37
8.3	Evaluation Boxplotgraph Erdos Time-Flows Small	37
8.4	Evaluation Boxplotgraph Erdos Time-Size Small	38
8.5	Evaluation Boxplotgraph Line Time-Flow Small	38
8.6	Evaluation Boxplotgraph Line Time-Size Small	39
8.7	Evaluation Boxplotgraph Ring Time-Flow Small	39
8.8	Evaluation Boxplotgraph Ring Time-Size Small	40

List of Tables

List of Listings

List of Algorithms

1 Introduction

Within the transformation of the industrial landscape (Industry 4.0) a popular demand for realization for cyber-physical systems and the expansion of the Internet-Of-Things has arisen. These technologies are tied to real-time networks. To guarantee the reliability and safety of a system, communication has to ensure strict timing requirements. Originally the networks these systems want to be deployed in, are designed to perform for best-effort communication, where maximal throughput is key.

To ensure timely boundaries for the transmission of time-critical messages many different networking solutions have been deployed and developed. Traditionally field buses were used. With the boom of Ethernet technology, multiple real-time Ethernet technologies have been deployed. SERCOS III and Profinet were two of them. These technologies are mostly incompatible with each other and a standardization for real-time Ethernet technologies is required as demand is high. The Institute of Electrical and Electronics Engineers (IEEE) are developing such standards with their IEEE 802.1Q networks in mind. IEEE are developing further functionality and extensions for real-time communication within IEEE 802.1Q networks. The IEEE 802.1Qbv extension was recently standardized by the Time-Sensitive Networking (TSN) Task group, which introduced functionality for transmissions bound to real-time guarantees. This extension standardizes a gating mechanism within switches which enables transmissions to be reliably routed and scheduled across.

While this mechanism defines the tools given to route and to schedule, it falls short of actually routing and scheduling. To route and schedule transmissions many approaches are given and well formulated. However a common theme to routing and scheduling has been to simplify the problem and break it down into a routing problem and a scheduling which are solved in sequence. Consequentially some viable solutions cannot be found with such an approach, resulting in suboptimal planning of the transmissions for time-critical communication.

Work as in [SDT+17] formulate the problem as a set of constraints. These sets of constraints can be solved for within general purpose problem solving frameworks. One powerful type of general purpose solver is the SMT-Solver. A SMT-Solver uses decision procedures to solve problems formulated as first-order logic with respect to a set of background theories. Progress in the field of SMT-Solvers is incentivized and displayed by annually held competitions [BDD+13; BDS05a; BDS05b]. A solution to the closely related scheduling problem has been proposed and approached with a SMT-Solver [Ste10]. It shows capabilities of SMT and sparks hope for further development and use of general purpose solvers such as SMT-Solvers within the networking domain.

ILP-based approaches have been numerous for scheduling problems [DN16; HBS10] and the work described in [SDT+17] already showcases an ILP-Solver's capabilities for jointly routing and scheduling communication.

How a SMT-based approach to the Joint Routing and Scheduling (JRaS) problem compares to an ILP-based approach is to be determined.

The main contribution of this paper is the formal specification of the JRaS constraints for time-sensitive networks. We solve the specified constraints with the state of the art SMT-Solver z3. We repeat this for multiple different problem instances. We evaluate and analyze the SMT-based solution in comparison to an in-house ILP-Based solution.

The remainder of the paper is structured as follows, in Chapter 2 we discuss related work. Afterwards in Chapter 3 we discuss Time Sensitive Networks (TSN). In Chapter 4 we introduce the problem solving frameworks SMT and ILP. We introduce our system model in Chapter 5 and our problem statement in Chapter 6. Design insights and implementation details are given in Chapter 7. In Chapter 8 we evaluate our SMT-Based solution and compare it to an in-house ILP-Based solution. We conclude our work in Chapter 9. The final Chapter 10 provides an outlook.

2 Related Work

The problem of transmitting data under real-time guarantees has been tackled numerous times. Many approaches sequentially route and schedule these transmissions [DN16; HBS10; Ste10]. Identifying drawbacks to this approach, recent work has tried an approach which jointly routes and schedules these transmissions [SDT+17].

Hanzálek et al. [HBS10] investigate scheduling for Profinet. In this approach routes are assumed to be given. Hanzálek et al. [HBS10] provide a formulation of the scheduling problem in terms of a Resource Constrained Project Scheduling with Temporal Constraints. The authors also compare different solving techniques further analyzing solving times and the quality of the generated schedule. The approach in [HBS10] uses an optimization objective which is mainly useful for Profinet. Hanzálek et al. [HBS10] limit flows to be on the same cycle.

In [Ste10] provide a SMT formulation to the scheduling problem for TTEthernet. They propose two solving techniques, which implement the SMT-Solver Yices in different ways. In the first technique the entire problem is fed to the solver, which the solver then solves in one go. In the second technique subsets of the problem get incrementally fed to the solver, which then solves for the constraints he knows so far. This way of solving the scheduling problem yielded in lower computation time in several cases. Steiner [Ste10] then provides an analysis regarding computation times and schedule efficiency in terms of resource utilization. No optimization objective was used in [Ste10]. Opposed to the work in [HBS10], Steiner [Ste10] allowed flows of varying cycles.

In the work [durr2016], the authors introduce an abstraction of the scheduling problem called the No-wait Packet Scheduling Problem and map it to the No-wait Job-shop Scheduling Problem which they then solve with an ILP-Solver. The optimization objective is more efficient resource utilization.

In many scheduling approaches the routes are assumed to be given beforehand [durr2016; HBS10; Ste10]. This however has drawbacks as some valid routing and scheduling options are excluded.

Schweissguth et al. [SDT+17] identify the problem and provide an ILP formulation for the joint routing and scheduling problem. They implement a solution using the Gurobi Optimizer. The underlying network for this formulation is TTEthernet. The optimization objective is low end-to-end delay.

I was provided access to an in-house ILP-based solution for the JRaS problem, which makes similar abstractions to the problem mentioned in [SDT+17]. However the ILP-Solver used is IBM's CPLEX Optimizer. A commercially used optimization tool.

The work in [DN16; HBS10; SDT+17] use ILP based solvers and problem formulations. ILP-Solvers currently seem to dominate the field however due to progress in the SMT-Domain this might change. SMT-Solvers are continuously being developed and improving. This improvement is getting encouraged and showcased by SMT-Comp [BDD+13; BDS05a; BDS05b]. To my knowledge clear comparisons of ILP-Solvers and SMT-Solvers for JRaS problems have not yet been made.

Common in all scheduling related papers presented here [DN16; HBS10; SDT+17] is the offline computation of the schedules. None of these schedules change during runtime. This is called offline static-scheduling and the JRaS approach presented in this paper will also follow suit.

3 Background

In this chapter we will introduce the needed background knowledge for Time Sensitive Networks.

3.1 Time Sensitive Networks

To guarantee that strict timing requirements are satisfied while transmitting data over a network, a network capable of handling scheduled traffic is necessary.

The Time Sensitive Networking (TSN) task group is in the process of standardizing such capabilities for IEEE 802.1Q networks. The TSN task group is a part of the IEEE 802.1 Working group. The recently developed IEEE 802.1Qbv standard enables the handling of scheduled traffic within IEEE 802.1Q networks.

A time-sensitive network is an IEEE 802 real-time Ethernet network, which comprises switches compliant with the IEEE 802.1Qbv standards to enable handling of scheduled traffic. The concept is to schedule the forwarding of frames through the switched network by utilizing the precise clock synchronization of switches and end-systems as achieved by the IEEE 802.1AS standard.

Networks considered in this paper are time-sensitive networks consisting of end-systems and switches along with a network controller. Connections between end-systems and switches are made via a medium for transporting electrical signals, for example copper cables or fiber cables. These connections are physical communication links which work in both directions, making the links full-duplex. End-systems inject the frames into the network periodically at precise points in time according to a schedule. End-systems inject these frames into the network via a network interface controller. Using the switch's switching fabric, the frames get routed through the network, hopping from switch to switch till the designated end-system has been reached. Switches schedule the forwarding of frames to outgoing edges over their ports utilizing a priority filter, different queues and a gating mechanism, which is called "time-aware shaper", per port. After the switching fabric selects which port the frames get forwarded to, a priority filter distributes the frames to the queues based on the frames' traffic-class which is specified in the priority code point (PCP) of the IEEE 802.1Q header. Figure 3.1 depicts how the gating mechanism regulates the frames on a single port. It is assumed that switches are Output-queued switches, containing several queues per outgoing port. We consider two types of traffic within our network, scheduled traffic and best-effort traffic. In this work all best-effort traffic is considered to be of equal importance and to simplify our model we aggregate the multiple queues used by best-effort traffic to one queue dedicated for best-effort traffic, as can be seen in figure 3.1. Since the switches are IEEE 802.1Qbv compliant the queues are regulated by a gating mechanism. This gating mechanism is used for selecting which traffic from which queue should be forwarded. Each queue is regulated by one gate respectively. A gate is either in the open state, in this case frames in the corresponding queue are available for transmission, or a gate is in the closed state, blocking frames in their corresponding queue. We assume the switches

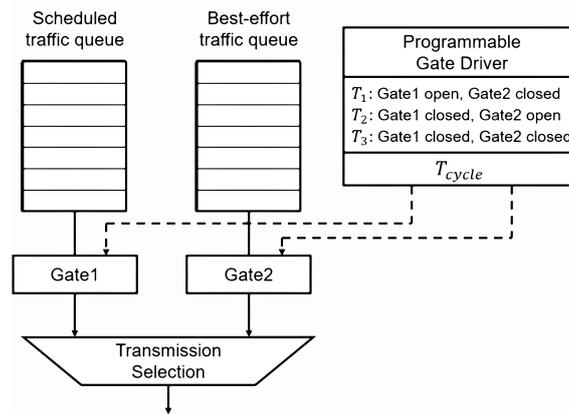


Figure 3.1: Architecture of a single port in an TSN switch (compliant with IEEE 802.1Qbv).

have enough memory to hold the periodically arriving scheduled traffic in their queues. However memory within a switch is still finite and our joint routing and scheduling has to ensure that the amount of frames in a queue doesn't increase every cycle. Gates are regulated by a programmable gate-driver. A time-sensitive network switch is programmed by programming the gate-drivers, using the network controller. The programmable gate-driver accomplishes this by specifying a finite sequence of gating events. Each event is a two-tuple containing a time the event occurs and the state of the gates. The state of the gates specify which gates to open when the event occurs. The times are specified relative to the program start at which this event occurs. When all events in the sequence have occurred the program terminates and restarts. This means that the sequence of events is repeated on a cycle with time T_{cycle} . The time-span where the system state comprised of all switch's states and end-system states repeats itself is called a hyper-cycle. A hyper-cycle has time $T_{hyper-cycle}$. Typically all switches are programmed on the same cycle time T_{cycle} which is then equal to the time $T_{hyper-cycle}$.

For example the gate-driver in the figure 3.1 at time T_1 exclusively opens $Gate_1$. At time T_2 $Gate_1$ closes and $Gate_2$ opens. All gates close at time T_4 . Afterwards the program terminates and restarts.

Forwarding a frame over a switch comes with some amount of processing time required to process the frame's header. The propagation delay of signals over the physical medium within the Local Area Networks we consider is negligible in comparison to the processing delay and will therefore be dismissed.

While the IEEE 802.1Qbv standards define the mechanisms required to schedule traffic in the network, they do not however define concrete algorithms for how to schedule the time-critical traffic in question.

4 Problem Solving Frameworks

Many problems have specialized algorithms and solvers implementing these algorithms to solve these problems. However there is no concrete intuitive algorithm to solve JRaS problems yet. We can however formulate requirements for our solution as a set of constraints and use a general-purpose solver to find a solution satisfying these constraints. Not always does solving a problem mean writing your own algorithm or using an existing fully specialized solver. Formulating the problem as a set of constraints might enable the use of general-purpose solvers such as SMT-Solvers and ILP-solvers. The following sections will introduce the SMT and ILP frameworks.

4.1 SMT

The problem of deciding whether a logical first order formula is satisfiable with respect to a combination of background theories is called the Satisfiability Modulo Theories (SMT) problem. A very closely related problem is the boolean satisfiability (SAT) problem. The boolean satisfiability (SAT) problem is that of determining if a propositional logic formula is satisfiable. SAT is one of the most well known and most fundamental NP-complete problems in computer science [Coo71]. SMT generalizes SAT by adding useful first-order theories. SMT therefore extends SAT by allowing the variables to be replaced by predicates from a variety of underlying theories. These theories include:

- The theory of integers
- The theory of reals
- The theory of arrays
- The theory of bitvectors

Problems can be categorized by their underlying logic, allowing specialized solving techniques for the specified logic. A logic evokes the use of a theory and some major restriction. For example the LIA logic states the use of linear integer arithmetic, restricting the use of integers to linear arithmetic operations only.

The theories relevant for this work comprise the theory of linear equation and the theory of integers.

A SMT-Solver is a tool for deciding the satisfiability of formulas in these theories. Many SMT-Solvers incorporate a SAT solver into their backend as a core NP-solver. These solvers are general purpose solvers as they solve a set of constraints without running a problem specific algorithm and therefore do not use structural information about the problem. We use the SMT-Solvers not only to show satisfiability of a set of constraints but also to return the *variablenbelegung* which made the set of constraints satisfiable.

Annual SMT competitions (SMT-COMP) [BDD+13; BDS05a] incentivize the continuous development and improvement of SMT-Solvers. The first SMT-COMP was held in 2005. The majority of participating SMT-Solvers in these competitions have permissive free licenses and are non-commercial software making them easily accessible. Many of the participating solvers are compatible with the SMT-LIB Language.

SMT-LIB [BST+10] is an initiative which among other things aims to provide standard rigorous descriptions of background theories and also defines standardized input and output languages for SMT-Solvers. Formulating a problem obiding by these standards therefore promises to have an up to date SMT-Solver being able to tackle it. This is an advantage since the problem formulation might outlive the development of any specific solver.

Modeling a problem as a set of SMT constraints tends to be easy due to the flexibility in background theories that are useable. SMT formulations are therefore a very powerful and expressive modelling tool. Variables and constraint formulations are very flexible and the introduction of helper-variables can mostly be avoided. However when trying to optimize a solution you leave SMT's home territory of deciding feasibility.

4.2 ILP

Linear programming (LP) is an optimization technique. Problems formulated as linear programs are optimized for a given linear objective-funktion while obiding given linear equality and linear inequality constraints. Integer Linear programming is like LP with the difference that some or all variables are restricted to be integers. ILP is NP-complete.

There are competitions for ILP-Solvers similar to the ones mentioned in Section 4.1.

An ILP-solver referred to in this work is the CPLEX Optimizer developed by IBM [CPL09]. The CPLEX Optimizer is the ILP-solver integrated into the in-house JRaS solution which we will compare the SMT-based solution presented in this work to.

Typically there are clear optimization goals when formulating a schedule, for example minimizing average end-to-end delay of scheduled traffic. ILP solvers naturally lend themselves to these kinds of problems since they encorperate an optimization objective.

5 System Model

In this chapter the system model is outlined.

A time sensitive flow is a stream of scheduled traffic, specified by a source, a destination, a period and the amount of data sent each cycle. We will also refer to time sensitive flow as flow. The source of a flow is the end-system injecting the traffic. The destination is the traffic's targeted end-system that is to be reached. The period specifies the cycle time of the flow. The amount of data sent each cycle directly dictates how many time slots are required for the transmission of the data each cycle. We therefore model time as slots where each slot is capable of transmitting one frame of data. Every time-sensitive flow also has specified real-time guarantees associated with it that our joint routing and scheduling has to ensure. For this work we assume jitter to be within reasonable bounds and to not impact the schedules we compute, and we therefore neglect it. The real-time guarantees provided are that of ensuring frames get transmitted within a specified end-to-end delay.

Formally the topology of a network is defined by an directed graph $G(V, E)$, where the vertices V represent the hosts and switches, and the edges $E \subseteq V \times V$ represent the physical communication channels. Since the communication channels are full-duplex, every communication channel has to be represented by two edges: A channel which connects $v \in V$ to $u \in V$ is represented by $(u, v) \in E$ and $(v, u) \in E$.

Additionally we consider the processing delay $pd : V \rightarrow \mathbb{N}$, which assigns a processing time $v.pd$ to every node $v \in V$.

A flow $f_k \in F$ is defined by a 5-tuple:

$$f_k = (f_k.source, f_k.destination, f_k.period, f_k.duration, f_k.deadline) \quad (5.1)$$

$f_k.source \in V$ is the flow's source at which the frames get injected into the network.

$f_k.destination \in V$ is the flow's destination to which the frames get transmitted to.

Flow gets transmitted periodically every $f_k.period$ units of time.

$f_k.duration$ is the reservation duration. In other words $f_k.duration$ is the time required to send the flow over an edge. This means that to schedule the flow f_k for transmission over an edge $e \in E$, the schedule has to reserve the edge e for the duration $f_k.duration$.

$f_k.deadline$ is the maximal end-to-end transmission time. The deadline restricts the time the flow can take from the time it enters the network at the source to the time till it reaches the destination.

Switches forward frames from an incoming edge to an outgoing edge. The minimal time required to do this is the before mentioned processing delay. However switches have the option to hold on to the frames for a certain period of time. This is called the queueing delay. Formally $f_k^{[i,j]}.hold$ is the time a switch $i \in V$ holds frames of flow $f_k \in F$ in the scheduled traffic queue for edge $(i, j) \in E$.

To program the gate drivers for a switch $v \in V$, the opening times of gates need to be specified. Let $(v, u) \in E$ be an outgoing edge for which the gate schedule needs to be specified. The gate for scheduled traffic associated with edge (v, u) needs to open everytime flow f_k uses the outgoing edge (v, u) . The time $f_k^{[v,u]}.st$ specifies at what point in time with respect to $f_k.period$ the gate which regulates scheduled traffic and is associated with (v, u) has to open. The gate has to remain open for the duration $f_k^{[v,u]}.duration$.

6 Problem Statement

A very intuitive approach to compute transmissions of time-critical packets through a time-sensitive network is to decompose the problem into the problem of routing packets and the problem of scheduling these packets and solving the two subproblems sequentially. After decomposing the problem, the two subproblems can be solved sequentially.

However solving these problems independently from each other disregards the strong interconnectivity of the problems and can have major implications on the solutions that are found for the initial problem. For example after routing several flows through the network, it may turn out that any schedule for this problem is infeasible. This could occur when the reservation times on a single link obstruct each other in such a way that transmissions get delayed so that one flow cannot meet the required end-to-end delay anymore.

Choosing a longer route might result in lower end-to-end transmission times if the queueing delays are shorter. This showcases that choosing the shortest routes for all flows might not always be the best solution.

“In order to address this interdependency between routing and scheduling, a joint strategy is necessary” [SDT+17].

The Joint Routing and Scheduling problem is an NP-hard problem [SDT+17].

The traffic in question lends itself to be routed and scheduled statically offline. With the given complexity of the problem and time required for scheduling traffic so far a dynamic real-time approach doesn't seem viable. The work in this paper follows suit after [DN16; HBS10; SDT+17; Ste10] and implements an approach for computing solutions to the JRaS problem offline.

The joint routing and scheduling problem can be formulated as follows: Given a network and a set of flows find a path for every flow and a configuration for every switch along these paths such that the flows can be transmitted periodically.

We say a flow can be transmitted when it obeys following constraints:

- Flow conservation constraint: Flow can not appear nor disappear from the network, exceptions are to be made for the flow's source where it can appear and the flow's destination where it can disappear.
- Collision free constraint: Multiple flows can not collide on an edge, meaning they are not allowed to be transmitted on the same edge at the same time.
- Propagation constraint: Flow transmitted on an edge will be transmitted on an adjacent edge at some time after the processing delay.
- Deadline constraint: The maximum travel time of a flow through the network has to be within the flow's specified timely bounds.

- Flow initiation constraint: Flow gets periodically injected into the network at the flow's source.

To get to the core of the problem meaningful abstractions can be made. The following abstractions are made in this work.

We discretize time. This allows us to represent time with integer numbers. We do not however specify how much time one unit of time represents. For e.g. one unit of time can represent one nano second or one unit of time can represent one micro second. Enabling us to represent time with the granularity which the situation requires.

By specifying flow origin and destination we limit our routing functionality to static device to device routing. We express a flow's $f_k \in F$ route $f_k.route$ from the source $v_1 = f_k.source$ to the destination $v_r = f_k.destination$ by the sequence of edges:

$$p = [(v_1, v_2), \dots, (v_{r-1}, v_r)] \quad (6.1)$$

In this work we will provide multiple SMT formulations for the JRaS problem. One formulation does not abstract the JRaS problem any further. Other formulations consider a combination of the following two abstractions.

A further abstraction considered is limiting the routes flows can take to be cycle-free. Routing a flow over a cycle can accomplish the same as buffering but without blocking a queue. Furthermore cycles can be used to realize the act of one flow passing the other flow. Not letting two flows pass each other could lead to situation where one flow obstructs the transmission of a different flow, which then gets delayed enough to make the flow not meet the end-to-end delay requirements.

The final abstraction considered is reducing the problem to a zero-queueing JRaS problem, which disables the use of the switch-queues entirely. In many applications a zero-queueing solution is wanted. One benefit is zero-queueing frees up memory within the switch which now can be used to store more buffered best-effort traffic.

7 Design and Implementation

In this chapter we will outline design consideration, and design choices we made for formulating the SMT constraints and how they represent the JRaS problem. Furthermore we introduce the SMT-Solver Z3. We also provide an SMT formulation to the JRaS problem. We then outline a technical implementation for solving JRaS problems using the solver we introduced and the constraints we formulated.

We then give a short introduction to the SMT-Solver Z3.

We then provide an SMT formulation for the JRaS problem.

Furthermore we describe a possible technical implementation of these constraints in a Python-based framework incorporating the SMT-Solver z3 to solve these constraints.

7.1 Design

We designed the constraints in an incremental approach. We begin formulating constraints for a heavily abstracted version of the problem and then extend the constraints to encapsulate the bigger picture. This could help identify problematic constraints and show which abstractions have how much impact on the solution and the solving process. Additionally this enables our solution to be compared to the existing inhouse ILP-based solution in a more direct fashion. Having the two solution solve problems as close to each other as possible lets us draw harder more straightforward conclusions. Having both solvers view the problem instance with the same level of abstraction also makes the resulting solutions comparable and lets us generate problem instances which can be solved by both solvers.

The inhouse ILP-based solution abstracts the problem to a zero-queueing joint routing and scheduling problem. This sets the $f_k^{[i,j]}$.hold variables to *zero*, preventing scheduled traffic queues from hold frames. Furthermore the ILP-based solution generates only paths which are to be cycle free. A comparable ILP approach to the inhouse ILP solution can be found in [SDT+17].

We generate different versions of the same constraints to allow analysis of how these abstractions alter the solution process and to simultaneously have a version that is one to one comparable with the ILP-based solution.

The first most abstracted version of the constraints implement JRaS with zero queueing and cycle free routing. These constraints will then be incrementally extended to account for queueing, multiple queues and cycles.

Constraints relating to the forwarding of frames over a network switch can easily be formulated for zero queue packet scheduling while being easily extend to incorporate waiting. In many applications for the joint routing and scheduling problem an optimization towards fast transmissions and short

routes is wanted. Limiting the routes to shortest routes only defeats the purpose of joint routing and scheduling. Faster transmissions however for best cases include no wait packet scheduling. Scheduling with zero queuing forwards scheduled traffic through switches as soon as possible. The total end-to-end transmission time then is a sum of processing delays caused by switches without additional queuing times caused by switches holding frames in their queues.

Scheduling with no wait results in the neglect of the queues meaning that queues are not used. Formulating queue related constraints for this iteration is therefore unnecessary we will however formulate queue related constraints for this work to work towards a solution to fully cover the JRS problem.

Further abstractions made are the formulation of no cycles. Flow conservation constraints can easily be modified and built upon the no cycles version to also allow cycling of flows.

Designing the first iteration of the SMT-based JRS solution with the same abstractions made as in the ILP-solution makes sense. It makes the comparison easier. The constraints for the simplified problem can easily be changed to cover a more general problem making the simplified problem easily extendable.

To evaluate the effectiveness of the SMT-based solution that will be further described in this work, we compare it to an existing solution utilizing a different problem solving engine. The existing ILP-Based jrs solution [SDT+17] does things. One of the goals is to compare my SMT-based solution to an existing ILP-solution. This ILP-solution models the JRS problem as Job-Shop Scheduling Problem. This simplifies the JRS Problem to No-Wait Packet Scheduling which differs from JRS by not allowing the Network switches to hold scheduled traffic in their queues. So our first iteration of an SMT-based solution will also utilize this assumption. Furthermore it saves data for solvertime and saves data for transmission paths, which we can also track to evaluate our solution for solve-speed and solve-quality compared to this existing ILP-base implementation.

7.2 Constraints

In this section we will provide a SMT formulation of the JRaS problem. We introduce the variables needed for every degree of abstraction specified in Section 7.1. Afterwards we will specify concrete mathematical formulations for the constraints introduced in Chapter 6. Multiple versions of a constraint can be given as specified in Section 7.1. This formulation of the JRaS problem already expresses the constraints in SMT formulation.

To formulate routing related constraints and to easily identify used routes we introduce a new binary variable $f_k^{[i,j]}.u$ to determine if flow $f_k \in F$ uses edge $(i, j) \in E$:

$$f_k^{[i,j]}.u = \begin{cases} 1, & \text{if flow } f_k \text{ uses edge } (i, j) \\ 0, & \text{otherwise} \end{cases}$$

If flow $f_k \in F$ uses edge $(i, j) \in E$ then the edge gets reserved for the duration $f_k.duration$ periodically every $f_k.period$ units of time.

Additionally we introduce an integer variable $f_k^{[i,j]}.ast$ which specifies the absolute start time of the flow $f_k \in F$ on edge $(i, j) \in E$. Absolute start time is the time which passed from beginning of the system start to the first transmission of a flow. This differs from $f_k^{[i,j]}.st$ in such a way that $f_k^{[i,j]}.st$ is the flow's reservation start time with respect to the period, while $f_k^{[i,j]}.ast$ is the flow's reservation start time with respect to the system start.

The following constraints set boundaries for the variables used throughout this section:

Version1 (no queueing):

$\forall f_k \in F, \forall (i, j) \in E :$

$$(f_k^{[i,j]}.u = 1) \implies 0 \leq f_k^{[i,j]}.st < f_k.period \quad (7.1)$$

$$f_k^{[i,j]}.st = f_k^{[i,j]}.ast \bmod f_k.period \quad (7.2)$$

Since the start time $f_k^{[i,j]}.st$ is given with respect to the flows period, the start time cannot exceed this period.

The start time $f_k^{[i,j]}.st$ and the $f_k^{[i,j]}.ast$ are inherently connected. Dividing $f_k^{[i,j]}.ast$ by $f_k.period$ yields how many cycles have passed since system start. The remainder of this division represents the offset or the start time $f_k^{[i,j]}.st$ with respect to the cycle period.

Version2 (with queueing):

$\forall f_k \in F, \forall (i, j) \in E :$

$$(f_k^{[i,j]}.u = 1) \implies 0 \leq f_k^{[i,j]}.st < f_k.period \quad (7.3)$$

$$f_k^{[i,j]}.st = f_k^{[i,j]}.ast \bmod f_k.period \quad (7.4)$$

$$f_k^{[i,j]}.h \geq 0 \quad (7.5)$$

For any flow $f_k \in F$ on any edge $(i, j) \in E$ the time $f_k^{[i,j]}.h$ the flow f_k buffers in switch i can't be negative.

We will now begin to formally formulate the constraints mentioned in ???. To ensure that flow does neither appear nor disappear within the network we formulate the flow conservation constraint as follows:

$\forall f_k \in F, \forall i \in V \setminus \{f_k.source, f_k.destination\} :$

$$\sum_{j \in V: (i,j) \in E} f_k^{[i,j]}.u = \sum_{j \in V: (j,i) \in E} f_k^{[j,i]}.u \quad (7.6)$$

We exclude source and destination from these constraints as flow gets injected into the network at the source and this flow leaves the network at the destination.

To realize the periodic flow injection into the network, the flow initiation constraint has to be formalized.

Flow initiation constraint Version1 (no cycling):

$\forall f_k \in F :$

$$\sum_{i \in V: (f_k.source, i) \in E} f_k^{[f_k.source, i]}.u = 1 \quad (7.7)$$

Equation (7.7) forces every flow $f_k \in F$ to use exactly one of the edges leaving the flow's origin $f_k.source$. This is where the flows' routes begin.

Flow initiation constraint Version2 (with cycling):

$\forall f_k \in F :$

$$\sum_{j \in V: (f_k.source, j) \in E} f_k^{[f_k.source, j]}.u = \sum_{i \in V: (i, f_k.source) \in E} f_k^{[i, f_k.source]}.u + 1 \quad (7.8)$$

$$\sum_{i \in V: (i, f_k.destination) \in E} f_k^{[i, f_k.destination]}.u = \sum_{j \in V: (f_k.destination, j) \in E} f_k^{[f_k.destination, j]}.u + 1 \quad (7.9)$$

Equation (7.8) and Equation (7.9) are formulated with the consideration of cycles through source and destination. Equation (7.8) specifies that the flow conservation constraint holds in the source with the exception that one more outgoing edge has to be used. Equation (7.9) specifies that the flow conservation constraint holds in the destination with the exception that one more incoming edge has to be used.

To ensure that flow gets propagated along the network, we formulate the propagation constraint (no queueing):

$$\forall i \in V : \quad (7.10)$$

$$(f_k^{[v_1, i]}.u = 1) \wedge (f_k^{[i, v_3]}.u = 1) \implies \quad (7.11)$$

$$f_k^{[v_1, i]}.ast + i.pd = f_k^{[i, v_3]}.ast \quad (7.12)$$

The time required to propagate a flow from an edge through a switch $i \in V$ on to an adjacent edge is exactly the switch's processing delay $i.pd$.

propagation constraint (with queueing):

$$\forall i \in V : \quad (7.13)$$

$$(f_k^{[v_1, i]}.u = 1) \wedge (f_k^{[i, v_3]}.u = 1) \implies \quad (7.14)$$

$$f_k^{[v_1, i]}.ast + i.pd + f_k^{[i, v_3]}.h = f_k^{[i, v_3]}.ast \quad (7.15)$$

The minimal time required to propagate a flow f_k from an edge through a switch $i \in V$ on to an adjacent edge $(i, j) \in E$ is the switch's processing delay $i.pd$ additionally the gate drivers can be scheduled to hold the flow in the switch i 's queue for time $f_k^{[i, j]}.h$.

The set of constraints Equations (7.6) and (7.7) would be enough to implement routing. However there might be some artefacts I call "ghost-cycles" in which some amount of traffic gets routed along a cycle. These artefacts however disappear with the use of propagation constraints. This happens because the ast variables have to increase with each propagation (assuming there is any processing delay).

To ensure that flows can't collide on an edge we have to accommodate for the periodic transmissions of the flows. To identify how many temporal cycles for each flow we have to account for we compute the least common multiple LCM of the flows we want to ensure freedom of collision for. $LCM(a, b)$ denotes the least common multiple of a and b . We ensure freedom of collision for each pair of flows.

Collision free constraint:

$$\begin{aligned}
& \forall (i, j) \in E, \forall f_k, f_l \in F, f_k \neq f_l, \\
& \forall c_k \in [0, \dots, \frac{LCM(f_k.period, f_l.period)}{f_k.period}], \\
& \forall c_l \in [0, \dots, \frac{LCM(f_k.period, f_l.period)}{f_l.period}]: \\
& ((f_k^{[i,j]}.u = 1) \wedge (f_l^{[i,j]}.u = 1)) \implies \\
& (((c_k \times f_k.period + f_k^{[i,j]}.st) + f_k.duration \leq \\
& \quad (c_l \times f_l.period + f_l^{[i,j]}.st)) \vee \\
& ((c_l \times f_l.period + f_l^{[i,j]}.st) + f_l.duration \leq \\
& \quad (c_k \times f_k.period + f_k^{[i,j]}.st)))
\end{aligned}$$

This ensures that when comparing every cycle of flow f_k with every cycle of flow f_l one of the two flows is finished transmitting before the other starts its transmission. $(c_k \times f_k.period + f_k^{[i,j]}.st)$ is the flow's start time of cycle c_k . We have to compare enough cycles to accommodate for the "overshoot". The "overshoot" is part of the reservation which crosses the periods border. This could potentially generate a huge amount of constraints. A slight optimization to limit the amount of constraints we generate in this procedure is instead of probing every cycle of flow f_k with every cycle of flow f_l for collision, we only compare cycles which are within one period of reach.

Deadline constraint:

$$\begin{aligned}
& \forall (f_k.source, j) \in E, \forall (i, f_k.destination) \in E: \\
& (f_k^{[f_k.source, j]}.u = 1) \wedge (f_k^{[i, f_k.destination]}.u = 1) \implies \\
& f_k^{[i, f_k.destination]}.ast - f_k^{[f_k.source, j]}.ast \leq f_k.deadline
\end{aligned}$$

We ensure that the end-to-end delays are within the timely bounds given by the deadline. For the no-cycle abstraction this constraint is straightforward as there can only be one pair of edges for which $(f_k^{[f_k.source, j]}.u = 1) \wedge (f_k^{[i, f_k.destination]}.u = 1)$ hold true. Assuming the constraint is fulfilled then the path's first edge $f_k^{[f_k.source, v_2]}.ast$ and the path's last edge $f_k^{[v_{(r-1)}, f_k.destination]}.ast$ must have a difference smaller than the deadline. The term $f_k^{[i, f_k.destination]}.ast - f_k^{[f_k.source, j]}.ast$ only gets smaller if other edges leaving the source or entering the destination are used, implying that these edges also satisfy the inequality.

To prevent flows within a queue from passing each other we formulate a queueing constraint:

$$\forall (i, j) \in E, \forall f_k, f_l \in F, f_k \neq f_l, \quad (7.16)$$

$$\forall c_k \in [0, \dots, \frac{LCM(f_k.period, f_l.period)}{f_k.period}], \quad (7.17)$$

$$\forall c_l \in [0, \dots, \frac{LCM(f_k.period, f_l.period)}{f_l.period}]: \quad (7.18)$$

$$((f_k^{[i,j]}.u = 1) \wedge (f_l^{[i,j]}.u = 1)) \wedge \quad (7.19)$$

$$(f_k^{[i,j]}.qid = f_l^{[i,j]}.qid) \implies \quad (7.20)$$

$$(((c_k \times f_k.period + f_k^{[i,j]}.st) < (c_l \times f_l.period + f_l^{[i,j]}.st)) \wedge \quad (7.21)$$

$$((c_k \times f_k.period + f_k^{[i,j]}.st - f_k^{[i,j]}.h) \leq (c_l \times f_l.period + f_l^{[i,j]}.st - f_l^{[i,j]}.h))) \vee \quad (7.22)$$

$$(((c_l \times f_l.period + f_l^{[i,j]}.st) < (c_k \times f_k.period + f_k^{[i,j]}.st)) \wedge \quad (7.23)$$

$$((c_l \times f_l.period + f_l^{[i,j]}.st - f_l^{[i,j]}.h) \leq (c_k \times f_k.period + f_k^{[i,j]}.st - f_k^{[i,j]}.h))) \quad (7.24)$$

$c_k \times f_k.period + f_k^{[i,j]}.st$ is the start time of flow f_k on edge (i, j) . $c_k \times f_k.period + f_k^{[i,j]}.st - f_k^{[i,j]}.h$ is the arrival time of flow f_k in queue for transmission on edge (i, j) . The set of constraints then states the flow that is first scheduled on this edge has to also arrive earlier or at the same time as the other flow. This constraint only applies for flows which use the same edge $(f_k^{[i,j]}.u = 1) \wedge (f_l^{[i,j]}.u = 1)$ and use the same queue $f_k^{[i,j]}.qid = f_l^{[i,j]}.qid$.

To allow the use of only a single queue as is common in IEEE 802.1Qbv networks we can limit the queue variable as follows:

$$\forall f_k \in F, \forall (i, j) \in E :$$

$$f_k^{[i,j]}.qid = 0$$

We can extend the problem by allowing multiple queues. This is realized by limiting the queue constraint as follows:

$$\forall f_k \in F, \forall (i, j) \in E :$$

$$f_k^{[i,j]}.qid \geq 0$$

$$f_k^{[i,j]}.qid \leq \text{number of queues}$$

The variable qid then specifies which scheduled traffic queue is used.

7.3 Choice of SMT-Solver

Z3 is the SMT-Solver integrated used in this work. Z3 is a state-of-the-art theorem prover from Microsoft Research. It is a popular and well tested software boasting 3,146 stars and 559 forks on github, while other well performing competitors like cvc4 have less than 1/10th of z3's github popularity. In the annually held SMT-COMP [BDD+13; BDS05a] although not an official competitor has the best overall results [BDD+13]. Z3 was a clear choice especially since I chose an SMT solver early on in development when I wasn't sure which logics I'm going to use.

7.4 Program flow

In this section we will be discussing the workflow of our implemented solution.

We will invoice our project passing it problem parameters. For our solution we allow both the generation and the import of problem instances. A problem instance spans a network and multiple flows. In the case of problem generation we have implemented some different but yet limited customization options. Allowing the importation of a problem instance let's the user run a specific handwritten problem instance. Problem generation works in two steps. In the first step a network is generated incorporating the network specific problem parameters. There are five network topologies support at the moment.

1. waxman
2. erdos
3. line
4. ring
5. price

A graph is then generated of a size specified in the problem parameters. In the second step we generate a number of flows which can also be customized. Source and destination nodes for every flow get randomly chosen. Through the parameters a list of possible reservations and cycle periods can be passed which the flows randomly choose from. Every flow gets assigned a deadline which is also customizable.

After the problem generation, the problem instance gets saved to files. We pass file location of the problem to a SMT-Solver-Wrapper. The Wrapper loads the problem instance from the files and generates an SMT formulation of the JRaS problem specified within the previously generated files.

The project passes the SMT constraints to the Z3 Theorem solver, via Z3's python API. Z3 shapes the problem to be of true SMT-LIB 2 syntax. We save the SMT formulation of the problem into an ".smt2" file, which can then be used to solve the JRaS problem with a seperate SMT-LIB 2 compliant solver. Z3 then solves the problem. If Z3 deems to problem to be satisfiable it returns a problem model which satisfied all the conditions. The SMT-Wrapper generates a solution to the JRaS problem by reading and interpreting the model. The solution to the problem is saved. Then statistics regarding solve time and solve quality get saved.

7.5 Implementation

The SMT-based approach to solve JRaS problems is implemented in a Python-based framework. The python version used is 3.6.5. The tool can generate a variety of problem instances. The pseudorandomnes used in the generation of the problem instances is provided by the random-module. Graphs for the problem instances are generated using the generation subpackage of the graph-tools module (v2.26). The SMT-Solver implemented is Z3 v4.7.1. SMT constraints are generated and added to Z3 via the Z3 API. The SMT constraints are generated by iterating over loops avoiding the use of quantifiers in the constraint formulations. An SMT-LIB 2 compliant version of the constraints is generated by the Z3 API. Optional console parameters give enough control to the generate SMT-Based solutions for the JRaS problem using the different abstractions outlined in this Chapter 7.

8 Evaluation

The evaluation of the SMT-based JRaS approach for time-sensitive networks is presented in this section. The evaluation setup is outlined and explained before we evaluate the SMT-based solution's performance and finally evaluate the solution's performance in contrast to the ILP-based solution's performance.

8.1 Evaluation Setup

All experiments were run on a 64bit CentOS 7.4.1708 machine. It is an 8-core 2000MHz Intel Xeon E312xx (Sandy Bridge) machine with 64000MB memory. The comparison is implemented in a Python-based framework. It uses Python version 3.6.5. We compare the performance of Z3 v4.7.1 to CPLEX v12.8 for different problem instances. CPLEX is a commercially used optimization tool developed by IBM.

The experiments are run in a three part setup.

In part one we generate a problem instance. The problem instances vary in graph topology, graph size and number of flows used. Topologies used are erdos, line and ring topologies. Graph sizes vary from 6 to 20 nodes with edges connected depending on the topology. Graphs are generated using graph-tool's "generation" subpackage. All switches within the network are considered to be equal and therefore have the same processing delay. We assign a processing delay of 8 to every node. The number of flows used in the problem instances range from 1 to 20 flows used. A flow's source and destination is randomly selected from the available graph nodes. A flow's cycle period and reservation length is randomly selected from a list of given possibilities. Cycle periods used are 100, 200 and 400. Reservation lengths used are 3, 6 and 9. End-to-end deadlines used are dependant on graph size and processing delay. We assign an end-to-end deadline of number of nodes \times processing delay to every flow. Furthermore we specify a maximum computation time of 30 Minutes after which the solvers timeout. In the following parts the SMT solution and the ILP solution will both try to solve this generated problem instance.

In part 2 we solve the problem using SMT. This is done in two steps. In the first step we generate a SMT formulation of the JRaS problem. In the second step we use z3 to solve the problem.

In part 3 we solve the problem using the inhouse ILP-based solution. This is done in similar two steps. An ILP formulation of the JRaS problem is generated. And afterwards CPLEX solves the generated formulation of the problem.

Since we did not specify how z3 utilizes multiple cores and we did not concern ourselves with changing the SMT formulation in a way such that z3 could natively make use of multiple cores, we leveled the playingfield by restricting both z3 and cplex to solve a given problem instance on a single core only.

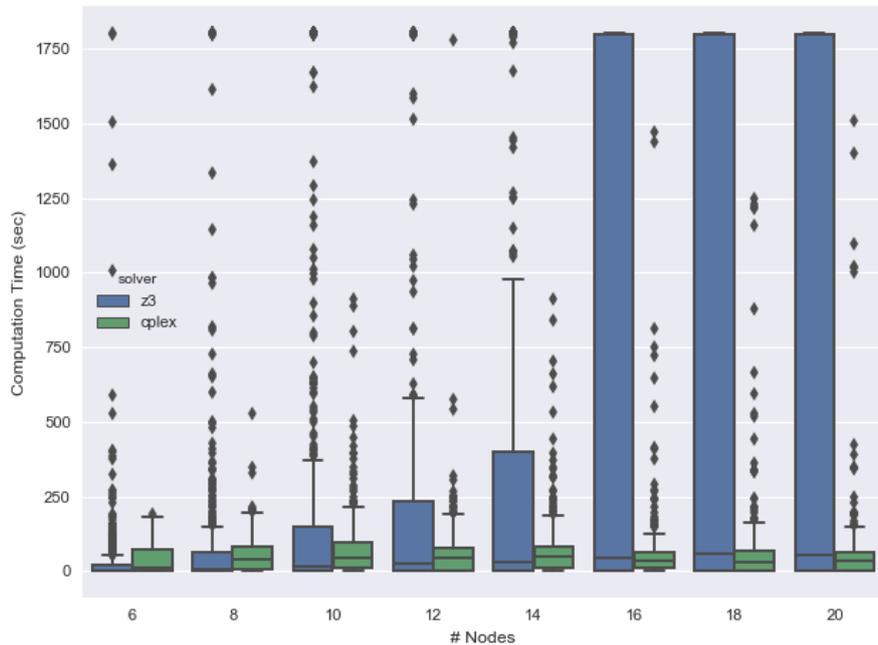


Figure 8.1: Box plot of JRaS solver runtimes with varying graph size for 3200 tests on erdos topology

Furthermore it should be noted that cplex is a commercially used optimization tool, while z3 uses a permissive free software license and is available for private use to any individual.

8.2 SMT-Evaluation

Figure 8.1 displays a box plot of solver runtimes for 3200 problem instances of with varying number of vertices, flows $\in [1-20]$, transmission periods $\in \{ 100, 200, 400 \}$ and transmission reservations $\in \{ 3, 6, 9 \}$. Figure 8.2 displays the same set of data but instead of plotting computation time against graphs size, we plot computation time against the number of flows.

As we can easily see the proposed z3-solution is impacted both by the size of the network and the number of flows. Computation time for JRaS of three flows can be consistently computed within 200 seconds. However at four flows the worst case times explode. The sudden dip at 12 flows in Figure 8.2 can be caused by timeouts of the z3 solution. 500 timeouts of the z3 solution have been registered. These timeouts are missing datapoints within the graph.

Far less timeouts (50 z3-timeouts) and maybe better samples can be found within smaller networks. Figure 8.3 displays a boxplot solver runtimes for 1600 problem instances for varying number of vertices $\in \{ 6, 8, 10 \}$. As we can see z3 can solve JRaS problems for up to 18 flows quite efficiently with only some outliers going above 700 seconds.

The median values in Figure 8.2 are quite low. However the third quartile spans across a very wide spread range of data points.

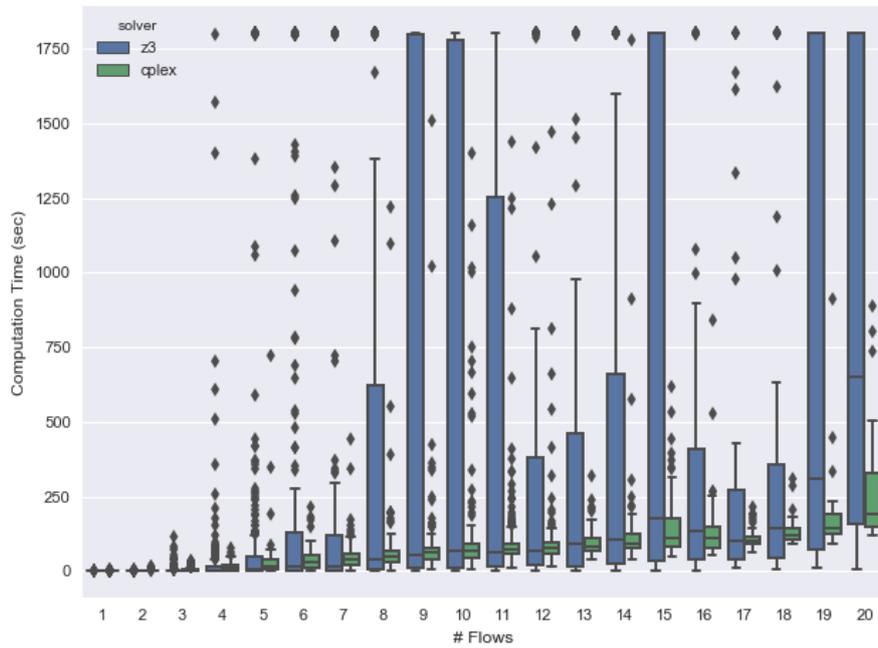


Figure 8.2: Box plot of JRaS solver runtimes with varying flow number for 3200 tests on erdos topology

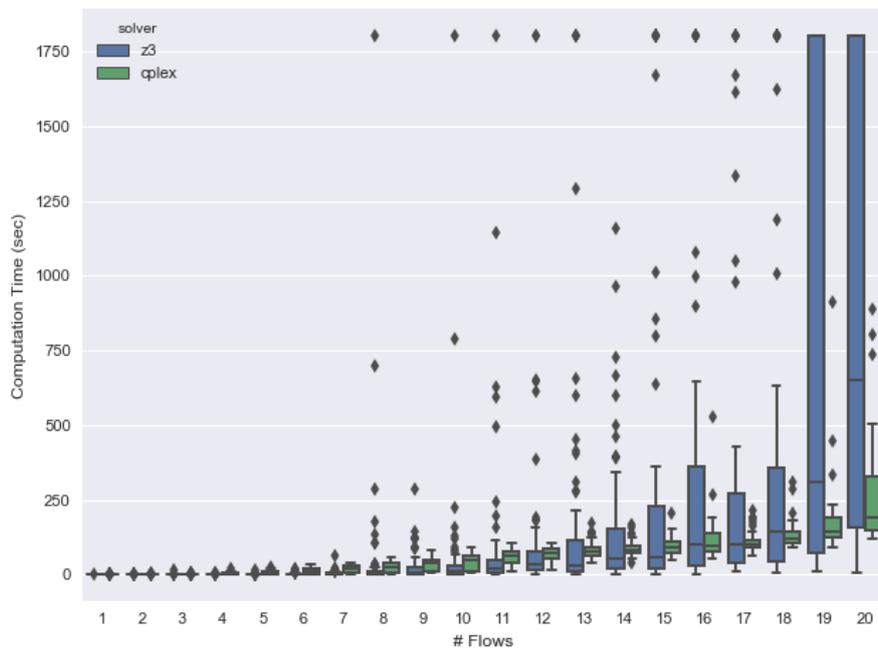


Figure 8.3: Box plot of JRaS solver runtimes with varying graph size for 1600 tests on erdos topology

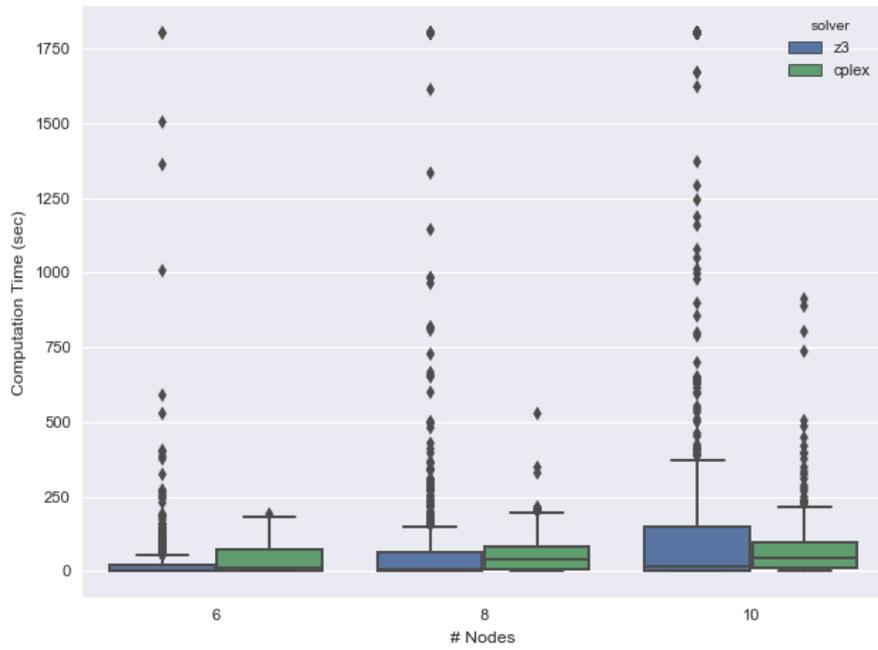


Figure 8.4: Box plot of JRaS solver runtimes with varying graph size for 1600 tests on erdos topology

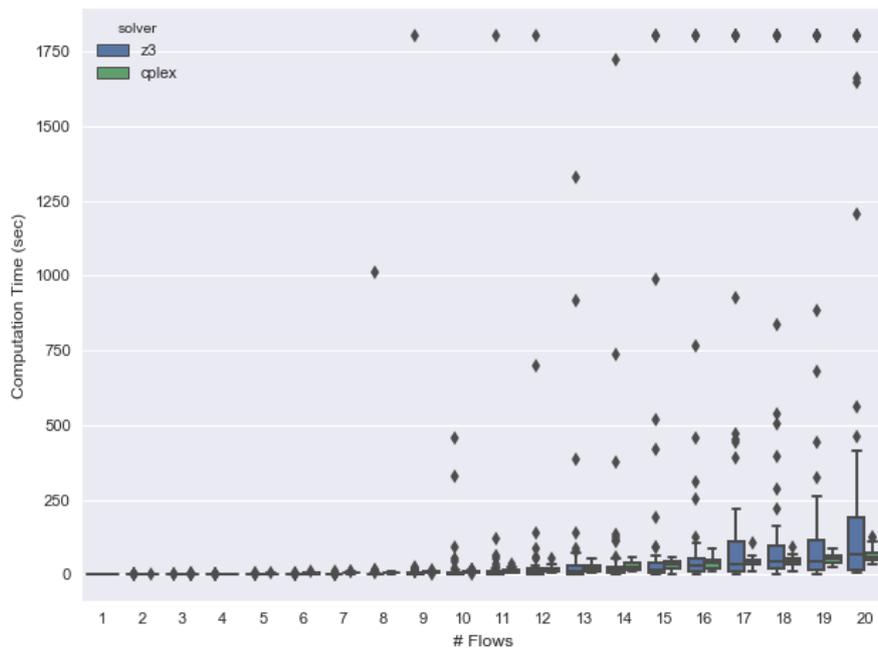


Figure 8.5: Box plot of JRaS solver runtimes with varying graph size for 1200 tests on line topology

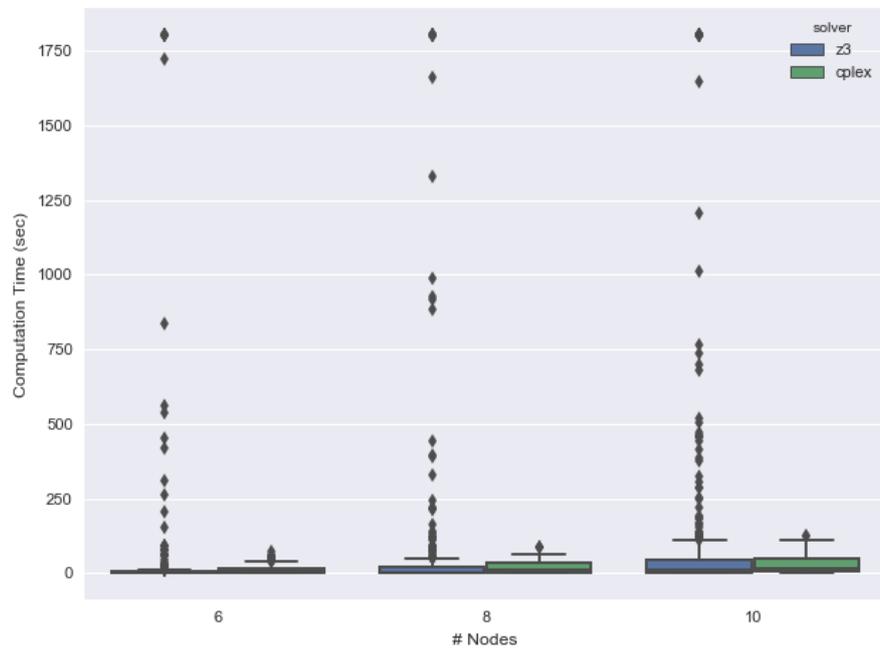


Figure 8.6: Box plot of JRaS solver runtimes with varying graph size for 1200 tests on line topology

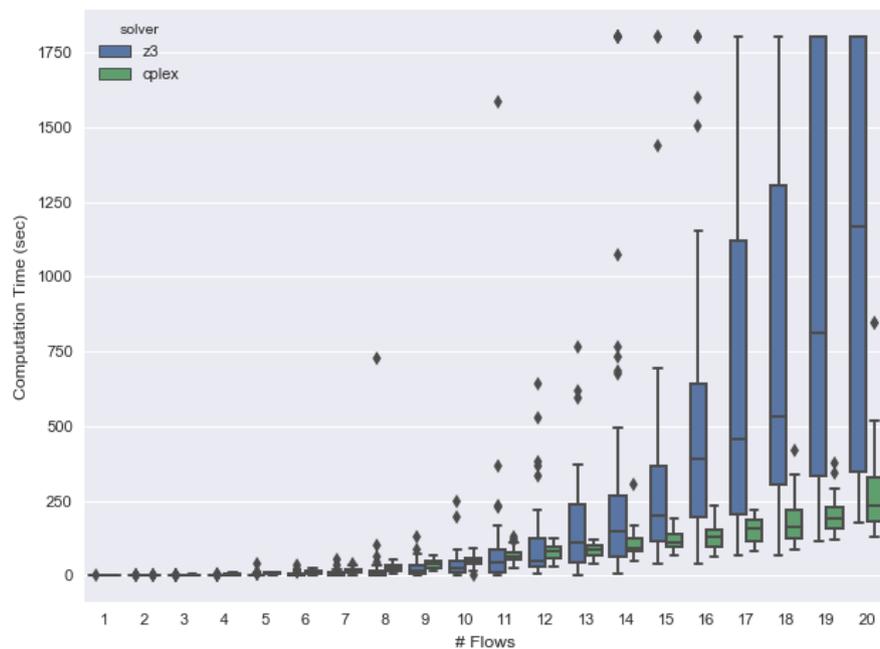


Figure 8.7: Box plot of JRaS solver runtimes with varying graph size for 1200 tests on ring topology

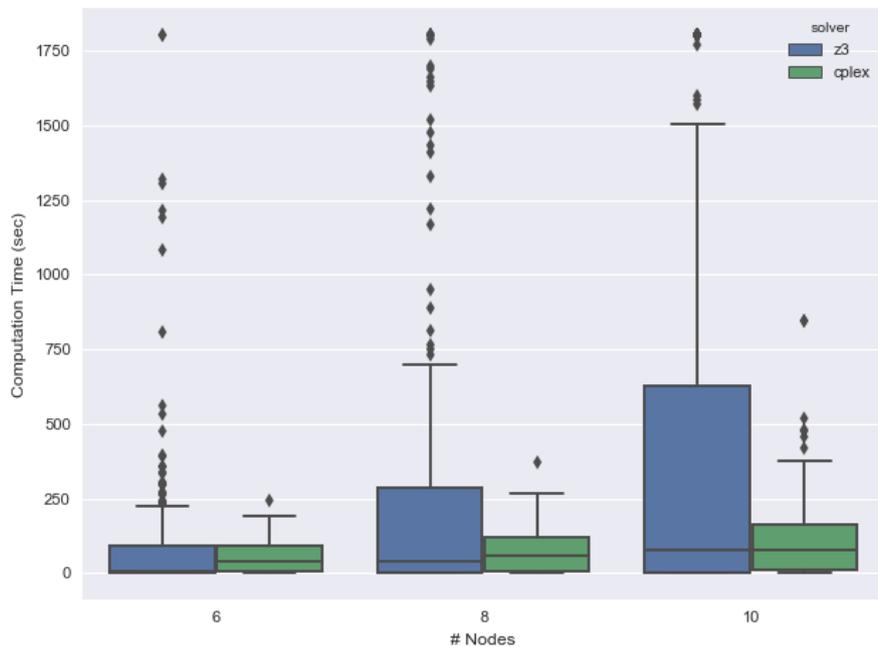


Figure 8.8: Box plot of JRaS solver runtimes with varying graph size for 1200 tests on ring topology

We now compare the performance of the z3-solution for the three different topologies to each other. Important mentions are that z3 timed out 50 times on erdos topology graphs, 40 times on ring topology graphs and 20 times on line topology graphs. As we can see in Figure 8.4, Figure 8.6 for graphs comprised of 10 nodes, the first three quartile of the tests can be computed in far below 250 seconds. In Figure 8.8 we can see that for ring graphs some test cases of the third quartile took more than twice this amount. This could mean that ring topologies pose some sort of greater difficulty for z3. When focusing on the average cases in Figure 8.3, Figure 8.5, Figure 8.7 we see a splendid performance of the z3 solution for line graphs Figure 8.5 a rather good performance for erdos graphs Figure 8.3 but horrendous performance for the ring topology.

8.3 Comparison ILP-SMT

We now compare the z3-solution to the cplex-solution used. Overall in all graphs we notice that cplex has far less outliers and that the outliers are far less drastic when compared to the z3-solution. Furthermore the datapoints seem far more compact for the cplex solution. The z3-solution displays huge sizes for the third quartile, this gets especially highlighted in the erdos topologies Figure 8.2, Figure 8.1, Figure 8.3, Figure 8.4 and appears to be prominent for high flow counts in combination with large graphs. On erdos and line topologies the average case of z3 often lies a few seconds below the cplex solution. cplex convincingly beats z3 however in ring topology graphs.

9 Conclusion

In this work we addressed the joint routing and scheduling problem for time-sensitive multi-hop networks. We did this while considering the new tools the IEEE 802.1Qbv standard provided. We approached the problem using a state-of-the-art general purpose SMT-Solver. We provided a SMT formulation for the JRaS problem. We've also showcased small tweaks to the formulation to further abstract the problem in a meaningful way. We deployed my formulation and generated test instances for it. A comparison of the SMT-based solution to an inhouse ILP-based solution was displayed. We analyzed and evaluated the two approaches based on their computation time. Compared to other options the proposed SMT-based solution yields promising average results, is however prone to frequent and severe outliers in performance.

10 Outlook

We could not definitively say if SMT-Solvers outclass other solution options for the Routing and Scheduling problem for time-critical communication. However with our promising average results and the continuous development of SMT-Solvers paired with new time-sensitive networking technology it is difficult to predict what the future may hold. We provide a timeless very flexibly modeled SMT formulaion of the Joint Routing and Scheduling problem which can easily be adapted to adjust for newly developed networking technologies.

Bibliography

- [BDD+13] C. Barrett, M. Deters, L. De Moura, A. Oliveras, A. Stump. “6 years of SMT-COMP”. In: *Journal of Automated Reasoning* 50.3 (2013), pp. 243–277 (cit. on pp. 15, 18, 22, 33).
- [BDS05a] C. Barrett, L. De Moura, A. Stump. “Design and results of the first satisfiability modulo theories competition (SMT-COMP 2005)”. In: *Journal of Automated Reasoning* 35.4 (2005), pp. 373–390 (cit. on pp. 15, 18, 22, 33).
- [BDS05b] C. Barrett, L. De Moura, A. Stump. “SMT-COMP: Satisfiability modulo theories competition”. In: *International Conference on Computer Aided Verification*. Springer. 2005, pp. 20–23 (cit. on pp. 15, 18).
- [BST+10] C. Barrett, A. Stump, C. Tinelli, et al. “The smt-lib standard: Version 2.0”. In: *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, England)*. Vol. 13. 2010, p. 14 (cit. on p. 22).
- [Coo71] S. A. Cook. “The complexity of theorem-proving procedures”. In: *Proceedings of the third annual ACM symposium on Theory of computing*. ACM. 1971, pp. 151–158 (cit. on p. 21).
- [CPL09] I. I. CPLEX. “V12. 1: User’s Manual for CPLEX”. In: *International Business Machines Corporation* 46.53 (2009), p. 157 (cit. on p. 22).
- [DN16] F. Dürr, N. G. Nayak. “No-wait packet scheduling for IEEE time-sensitive networks (TSN)”. In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. ACM. 2016, pp. 203–212 (cit. on pp. 15, 17, 18, 25).
- [HBS10] Z. Hanzálek, P. Burget, P. Sucha. “Profinet IO IRT message scheduling with temporal constraints”. In: *IEEE Transactions on Industrial Informatics* 6.3 (2010), pp. 369–380 (cit. on pp. 15, 17, 18, 25).
- [SDT+17] E. Schweissguth, P. Danielis, D. Timmermann, H. Parzyjegla, G. Mühl. “ILP-based joint routing and scheduling for time-triggered networks”. In: *Proceedings of the 25th International Conference on Real-Time Networks and Systems*. ACM. 2017, pp. 8–17 (cit. on pp. 15, 17, 18, 25, 27, 28).
- [Ste10] W. Steiner. “An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks”. In: *Real-Time Systems Symposium (RTSS), 2010 IEEE 31st*. IEEE. 2010, pp. 375–384 (cit. on pp. 15, 17, 25).

All links were last followed on March 17, 2018.

Declaration

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

place, date, signature