Universität Rostock

Traditio et Innovatio

Robuste Schedules für zeitsensitive Netze

Name: Behrens, Nico

Matrikelnummer: 219202809

Abgabedatum: 06.06.2023

Gutachter: Dr.-Ing. Helge Parzyjeglas
Universität Rostock
Fakultät für Elektrotechnik und Informatik

PD Dr.-Ing. habil. Peter Danielis
Universität Rostock
Fakultät für Elektrotechnik und Informatik

Bachelorarbeit
Fakultät für Informatik und Elektrotechnik

Danksagungen

Ich möchte mich bei meinen beiden Betreuern und Gutachtern **Dr.-Ing. Helge Parzyjeglas** und **PD Dr.-Ing. habil. Peter Danielis** bedanken. Ohne ihre zielorientierten Anweisungen und ihr regelmäßiges Feedback wäre die Anfertigung dieser Arbeit nicht möglich gewesen. Beide nahmen sich auch außerhalb der Betreuungszeiten Zeit, um meine Fragen zu beantworten und mir bei Problemen zu helfen.

Mein Dank geht auch an meine Familie und Freunde, die mich besonders während meines Schreibprozesses unterstützt haben.

Hier erwähnt seien noch Cong und Alan, die sich Zeit genommen haben, Teile meiner Arbeit Korrektur zu lesen und mir wertvolle Hinweise gegeben haben.

Kurzfassung

Time-Sensitive Networking (TSN) ist ein Standard zur Ermöglichung von zeitkritischem Netzwerkverkehr. TSN baut auf regulärem IEEE 802.1 Ethernet auf und erweitert dieses um ein Zeitverständnis zur Ermöglichung von maximalen Latenzen, geringem Jitter und minimalem Stau des Netzwerkverkehrs. Ein Schedule definiert, wann Pakete von den jeweiligen Netzwerkteilnehmern gesendet werden. Dieser Schedule funktioniert nur, wenn die Uhrzeiten auf allen Netzwerkteilnehmern identisch sind.

Für diese Arbeit wird die Simulationssoftware OMNeT++ mit ihrer Erweiterung INET benutzt. Letztere stellt Funktionalitäten des TSN-Standards zur Verfügung. In dieser Software werden zuerst ein Vergleichsexperiment im Optimalfall und eines einem fehlerbehafteten Netzwerk durchgeführt. Mit optimalen Bedingungen zeigt der Schedule keine Schwächen. Ohne Gegenmaßnahmen stellen sich in realistischen Netzwerken ungeeignete Vergleichsmetriken ein. Das ist Motivation für weitere Untersuchungen. Als erste Lösung wird eine graduelle Vergrößerung der Zeitfenster von 25%, 50% und 100% in den Schedules untersucht. Größere Zeitfenster führen zwar zu einer Verbesserung der Vergleichsmetriken, reichen allerdings wegen ihres hohen Jitters alleine nicht aus, um die Anforderungen an zeitkritischen Netzwerkverkehr zu erfüllen. Als zweite Lösung wird das im TSN-Standard definierte generalized Time-Precision-Protokoll zur Synchronisation der Netzwerkteilnehmer verwendet und produziert in zwei Konfigurationsvarianten bessere Ergebnisse, bleibt aber hinter dem Optimum.

Abschließend ist die Verwendung eines Synchronisationsprotokolls unerlässlich, allerdings ist dieses nur in richtiger Konfiguration des gPTP effektiv und erfordert zur Verbesserung der Ergebnisse Restrukturierung der Netzwerktopologie. Eine Kombination beider Maßnahmen erscheint sinnvoll.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

In den nachfolgenden Abschnitten wird eine grundlegende Übersicht über die Motivation und Ziele dieser Arbeit gezeichnet. Dazu wird zunächst der Begriff Time-Sensitive Networking (TSN) definiert und die Motivation für die Verwendung von TSN erläutert. Zuletzt wird die Struktur der Arbeit beschrieben.

1.1. Motivation

Time-Sensitive Networking (TSN) hat in den letzten Jahren an Bedeutung gewonnen. Vor allem in der Industrie kommt TSN vermehrt zur Anwendung, um zeitkritischen Netzwerkverkehr zu ermöglichen. In industriellen Produktionsprozessen kann es zu erheblichen Schäden kommen, wenn Nachrichten selbst durch Verzögerungen im Nanosekundenbereich nicht rechtzeitig im Netzwerk an ihr Ziel gesendet werden können.

TSN baut auf regulärem IEEE 802.1 Ethernet auf. Ethernet besitzt kein Verständnis von Zeit und kann deshalb keinen präzisen Netzwerkverkehr ermöglichen. Durch das Senden von zeitgesteuertem Netzwerkverkehr lassen sich obere Schranken für die Latenz in Netzwerken einhalten und oben genannte Probleme vermeiden.

Jeder TSN-Switch verfügt über bis zu acht FIFO-Warteschlangen mit jeweils einem Gate. Bei diesen Gates wird über eine Gate-Control-List (GCL) angegeben, auf welchem Gate Netzwerkverkehr gesendet werden darf. Sogenannte Schedules bestimmen für jeden Netzwerkteilnehmer individuell wann Netzwerkverkehr gesendet werden darf, das ermöglicht zeitkritische Kommunikation.

Die Uhren der verschiedenen Netzwerkteilnehmer müssen dabei einheitlich sein, andernfalls kann es zu verheerenden Auswirkungen bei der Einhaltung des Schedules kommen.

1.2. Ziel

Es gilt verschiedene Auswirkungen von fehlender Zeitsynchronisation im Netzwerk zu testen. Die Teilziele dieser Arbeit lassen sich wie folgt auflisten:

Szenarien und Metriken. Es muss verschiedene Szenarien geben, in denen man mit Fehlern behaftete als auch richtige Konfigurationen untersuchen kann. Dabei sind verschiedene Metriken wie Latenz, Jitter und Queue-Länge während der Durchführung der Experimente aufzuzeichnen. Unterschiedliche Netzwerktopologien können verschiedene Ergebnisse liefern und müssen Beachtung finden.

Uhrensynchronisation. Jeder Netzwerkteilnehmer muss eine einheitliche Uhrzeit besitzen, um zeitgesteuerten Netzwerkverkehr zu realisieren. In der echten Welt entste-

hen durch kleinste Ungenauigkeiten der sich auf den Netzwerkteilnehmern befindlichen Uhren Abweichungen. Diese Fehler gilt es durch verschiedene Synchronisationsmodelle auszugleichen. Im Standard ist dabei das generalized Precision-Time-Protocol (gPTP) definiert. Dieses generiert selber wieder Netzwerkverkehr, der in den Schedules oder der Strukturierung der Topologie Beachtung finden muss.

Gegenmaßnahmen. TSN besitzt im Standard definierte Mechanismen zur Reduzierung von auftretenden Ungenauigkeiten oder Fehlern. Neben der Synchronisation von Uhren steht auch die Vergrößerung der Zeitfenster im Schedule zur Auswahl. Eine Vergrößerung der Zeitfenster könnte potentiell Netzwerkverkehr zulassen. Durch sinnvolle Anwendung dieser Strategien ist eine Begrenzung von Fehlern möglich.

Evaluation. Im Netzwerksimulator gilt es verschiedene Strategien zu testen und deren Eignung zur Bekämpfung von Ungenauigkeiten und Fehlern miteinander zu vergleichen. Uhren besitzen in der Realität immer einen Drift und gegebenenfalls ein Offset. Als Referenz gilt es ein Netzwerk ohne Ungenauigkeiten der Uhren und eines mit Ungenauigkeiten ohne Gegenmaßnahmen heranzuziehen.

Durch die Lösung dieser Probleme wäre präziserer Netzwerkverkehr möglich.

1.3. Aufbau der Arbeit

In Kapitel 2 werden sowohl TSN und gPTP als wichtigste Bestandteile dieser Arbeit erklärt als auch das Werkzeug OMNeT++ mit seiner TSN-fähigen Erweiterung INET beschrieben.

Um die grundlegende Funktionsweise von OMNeT++ und dem Schedule zu demonstrieren und später gewonnen Daten als Referenz nutzen zu können, werden Ausgangsexperimente in Kapitel 3 durchgeführt, um aufzuzeigen, welche Probleme entstehen können und wie sich das Netzwerk und der Schedule im Optimalfall verhalten. Es werden Überlegungen zur Lösung dieser Probleme angestellt.

Die zuvor ermittelten Strategien werden in Kapitel 4 anhand von Simulationsergebnissen auf ihre Effizienz untersucht. Eine Unterteilung in zwei unterschiedliche Lösungsstrategien wird vorgenommen. Zuerst wird die Vergrößerung der Zeitfenster des Schedules erwogen. In mehreren Versuchsdurchläufen werden die Zeitfenster vergrößert und die Ergebnisse mit vorherigen Erkenntnissen verglichen. Als Zweites werden zwei unterschiedliche Netzwerkkonfigurationen zur Synchronisation der Uhren auf den Switches mit gPTP verwendet. Die Ergebnisse werden miteinander verglichen.

Kapitel 5 zieht einen Schlussstrich und gibt dem Leser abschließende Handlungsanweisungen für die Wahl von geeigneten Gegenmaßnahmen im Kontext von ungenauen Uhren.

2. Grundlagen

Nachfolgend werden TSN und gPTP als Standards um zeitkritischen Netzwerkverkehr zu ermöglichen in ihrer Funktionsweise näher beschrieben. Die Durchführung von allen Experimenten in dieser Arbeit wird mit OMNeT++ und INET durchgeführt und müssen deshalb eingeführt werden.

2.1. Time-Sensitive Networking

Time-Sensitive Networking (TSN) ist ein Standard, entwickelt vom Institute of Electrical and Electronics Engineers (IEEE) zur Bereitstellung von zeitkritischem Netzwerkverkehr [9] [2]. Der TSN-Netzwerkverkehr geschieht periodisch. Unter Kenntnis der einzelnen Netzwerkströme lässt sich der Netzwerkverkehr planen. Der Netzwerkverkehr kann je nach Verkehrsklasse unterschiedlich behandelt werden. Jeder Switch besitzt für jeden Port mehrere Queues die mit jeweils einem Gate verbunden sind. In jeder Queue wird der ausgehende Netzwerkverkehr einer Verkehrsklasse gespeichert, das erlaubt es die einzelnen Netzwerkklassen durch variable Priorisierung unterschiedlich zu steuern. Es gibt bis zu acht verschiedene Verkehrsklassen, denen unterschiedliche Prioritäten zugewiesen werden können. Pro Port existieren zwischen 1 und 8 Queues. Somit existiert eine eins zu eins Abbildung von Verkehrsklassen zu Queues. Eine Gate-Control-List (GCL) sagt, wann welches Gate offen ist. Die Breite der GCL stimmt mit der Anzahl der Gates überein und jeder Eintrag besteht aus einer Bit-Sequenz, die angibt, wann welches Gate zur Übertragung offen ist. Die Einträge der GCL wechseln zur unterschiedlichen Zeitpunkten. Die Anzahl der Einträge gilt als Länge der GCL. Ein Transmission-Selection-Algorithmus wählt dann das zu übertragende Paket aus.

Als Visualisierung dient Abbildung 2.1. In dieser Abbildung gibt es 8 Queues beschriftet von 0 bis 7. Jede Queue ist mit einem Traffic-Selection-Algorithmus verbunden. Darunter befinden sich die individuellen Transmission-Gates. Diese sind entweder offen (o) oder geschlossen (C). Die Steuerung wird über die GCL rechts an der Seite bestimmt. Je nach Zeitpunkt T ist ein anderer Eintrag der GCL aktiv. Die GCL besitzt in diesem Fall eine Breite von 8 und eine Länge von 80. Für die endgültige Selektion des Paketes ist dann der Selektionsalgorithmus zuständig.

Zur Bestimmung des zu übertragenden Paketes, wenn zwei oder mehr Gates offen sind und sich in den zugehörigen Queues Pakete befinden, werden im Standard verschiedene Traffic-Selection-Algorithmen definiert. Die zwei wichtigsten Algorithmen wären:

1. der Strict Priority Algorithmus nimmt das Paket aus der Queue mit der höchsten Priorität und

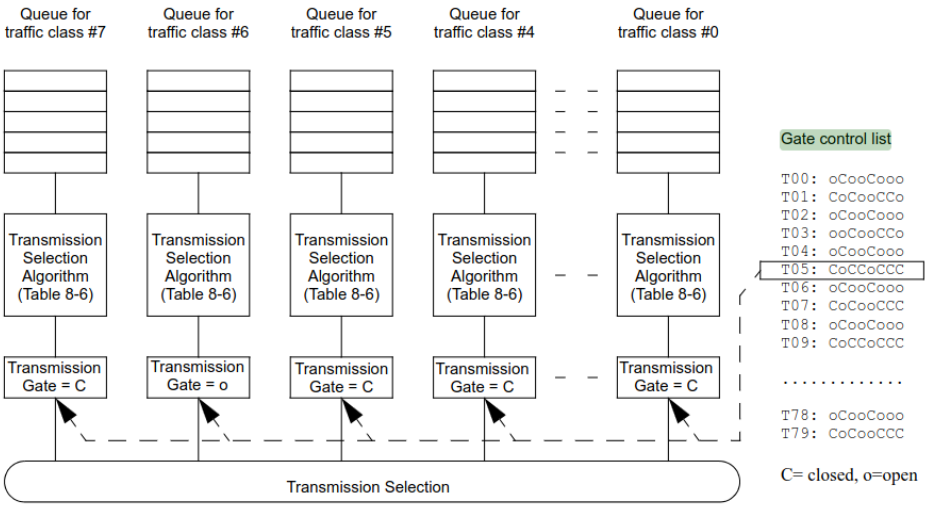


Abbildung 2.1.: Aufbau der Transmission Selection mit welcher die zu übertragenden Pakete bestimmt werden. Entnommen aus [9].

2. der Credit-based Shaper Algorithmus, bei dem eine Credit-Variable mit der Einheit Bits für jede Queue existiert und die Übertragung nur dann stattfinden kann, wenn neben einem offenen Gate auch Credits vorhanden sind. Eine Größe *portTransmitRate* gibt die Übertragungsgeschwindigkeit vom MAC-Dienst dieses Ports an. Die Anzahl an Credits wird mit einer konstanten Rate *idleSlope* erhöht, wenn zwar keine Übertragung stattfindet aber das zugehörige Gate trotzdem offen ist. Bei der Übertragung werden die Credits dann mit der Rate $sendSlope = idleSlope - portTransmitRate$ reduziert.

Eine Übertragung kann nur stattfinden, wenn eine der Queues ein Paket enthält und das dazugehörige Gate offen ist.

TSN erlaubt es neben zeitsensitiven Netzwerkverkehr auch Best-Effort-Verkehr zu übertragen. Dieser erhält dann als regulärer Ethernet-Netzwerkverkehr keine deterministische Latenz und Jitter, benötigt aber kein eigenes Netzwerk. Das lässt sich mit zwei verschiedenen Queues realisieren, die den Netzwerkverkehr aufteilen. In die eine Queue gelangt nur zeitsensitiver Netzwerkverkehr und in die andere der Best-Effort-Verkehr. Die GCL schaltet das Gate, welches zur Queue mit zeitsensitiven Netzwerkverkehr gehört, genau dann ein, wenn Pakete in der Queue vorliegen. Andernfalls wird der Best-Effort-Verkehr übertragen [5].

Innerhalb von zeitsensitiven Netzwerken ist es möglich mithilfe von Network-Calculus (NC) obere Schranken für Latenz und Jitter zu berechnen [6].

2.2. Generalized Precision Time Protocol

Beim generalized Precision-Time-Protocol (gPTP) handelt es sich wie in [10] beschrieben um ein Protokoll zur Synchronisierung von Uhren in TSN-Netzen.

Dieses Protokoll basiert auf dem Precision-Time-Protocol (PTP) und erweitert dieses beispielsweise durch die Verbindung von PTP-fähigen Netzwerkgeräten (PTP-Instanzen) zu einem gPTP-Netzwerk. Zur Synchronisierung sendet initial immer eine Grandmaster-Instanz die aktuellen Zeitinformationen an alle verbundenen PTP-Instanzen. Jede PTP-Instanz verfügt über eine eigene Uhr. Eine PTP-Instanz kann folgendes sein und sich unterschiedlich verhalten:

1. eine PTP-Endinstanz, die nur Zeitinformationen erhält und sie zur Korrektur nutzt oder

2. eine PTP-Relay-Instanz, die Zeitinformationen im Netz erhält, sie mit ihren eigenen Zeitinformationen korrigiert und an verbundene PTP-Instanzen weitersendet.

Jeder Port an einer PTP-Instanz ist entweder ein Slave-Port, der PTP-Netzwerkverkehr empfängt oder ein Master-Port, der PTP-Netzwerkverkehr sendet. Ein Port kann nicht gleichzeitig Master- und Slave-Port sein. Eine PTP-Endinstanz besitzt folglich nur einen Slave-Port, eine PTP-Relay-Instanz besitzt genau einen Slave-Port und mindestens einen Master-Port und eine Grandmaster-Instanz besitzt mindestens einen Master-Port. Diese Wahl erzeugt aus den Netzwerkteilnehmern einen Spannbaum, in denen der PTP-Netzwerkverkehr fließt.

Ein PTP-Netzwerk ist in Abbildung 2.2 zu sehen. Dort ist eine Grandmaster-Instanz abgebildet. Diese ist mit zwei Relay-Instanzen verbunden. Eine Relay-Instanz verbindet sich direkt mit einer Endinstanz, die andere verbindet sich davor noch mit einer anderen Relay-Instanz. Die Endinstanzen stehen immer am Ende einer Verbindung. Mit M und S sind die Ports in Master- und Slave-Ports aufgeteilt.

Zur Synchronisation der Uhren werden verschiedene Arten von Nachrichten zwischen den verschiedenen PTP-Instanzen ausgetauscht. Die Grandmaster-Instanz sendet Synchronisationsnachrichten an alle anderen PTP-Instanzen. In dieser befindet sich ein Zeitstempel mit denen die Uhren ihre Abweichung korrigieren können. Nach dem Sender der Synchronisationsnachricht wird eine Follow-Up-Nachricht mit dem genauen Sendezeitpunkt der Synchronisationsnachricht an die PTP-Instanzen gesendet um eine präzisere Zeitsynchronisation zu ermöglichen. Die PTP-Instanzen senden periodisch Delay-Request-Nachrichten an die Grandmaster-Instanz. In diesen befindet sich ein Zeitstempel des Sendezeitpunktes. Diese werden von der Master-Instanz mit Delay-Response-Nachrichten beantwortet. In diesen befindet sich wieder ein Zeitstempel des Sendezeitpunktes und der Sendezeitpunkt der Synchronisationsnachricht. Damit kann die Übertragungszeit der Synchronisationsnachrichten gemessen werden und für die Synchronisation genutzt werden. Die Intervalle, in denen die Nachrichten von gPTP gesendet werden, können je nach Bedarfsfall unterschiedlich konfiguriert werden.

Im Gegensatz zum weitverbreiteten Network-Time-Protocol (NTP) [7][3], das eine Synchronisierung im Millisekundenbereich erlaubt, erreichen gPTP und PTP eine viel höhere Genauigkeit im Nanosekundenbereich. NTP nutzt eine hierarchische Master-Slave-Ordnung, während PTP auf Peer-to-Peer-Verbindungen basiert. An der Spitze des NTP-Netzwerks steht eine präzise Uhr, beispielsweise Satelliten- oder Atomuhr aus dem Internet. PTP macht sich Hardware Timestamping zunutze, dadurch können durch Software ausgelöste Verzögerungen verhindert werden. Durch das Senden von

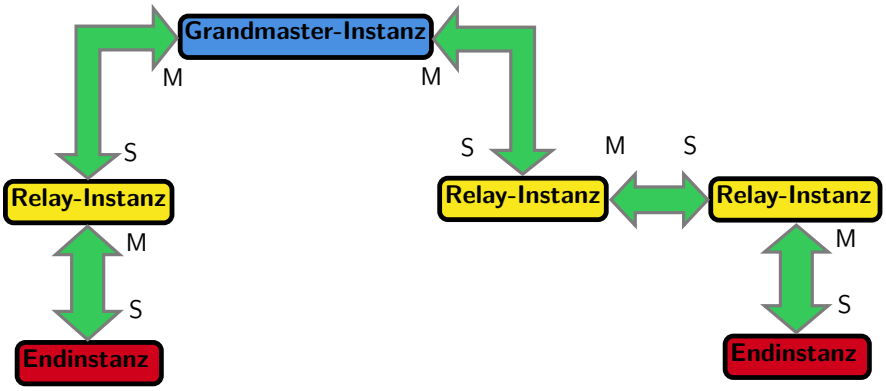


Abbildung 2.2.: Aufbau eines PTP-Netzwerkes

Synchronisationsnachrichten bei PTP zwischen den PTP-fähigen Geräten können genaue Abweichungen der einzelnen Uhren korrigiert werden. NTP verlässt sich auf das periodische Abfragen von Zeitinformationen der Master-Instanz.

Eine PTP-Instanz kann in mehreren gPTP-Domänen existieren. Eine gPTP-Domäne ist eine Ansammlung von PTP-Instanzen die gPTP realisieren. Folglich kann ein gPTP-Netzwerk aus mehreren PTP-Domänen bestehen.

PTP und damit auch gPTP sind sinnvoller in lokalen Netzen und letzteres liefert damit in Anbetracht unserer Anforderungen in zeitsensitiven Netzen ein passendes und bereits im TSN-Standard beschriebenes Protokoll zur Uhrensynchronisation.

2.3. OMNeT++

Bei OMNeT++ [12] handelt es sich um eine quelloffene Software zur ereignisorientierten Simulation. OMNeT++ selbst bietet eine eingebaute Entwicklungsumgebung an. OMNeT++ ist in C++ geschrieben, während die Entwicklungsumgebung auf Eclipse basiert und damit Java benötigt. Innerhalb der Entwicklungsumgebung kann man Simulationsmodelle aufbauen und konfigurieren. Das Programmieren der Simulationselemente zur Erweiterung ihrer Funktionalität ist über C++ möglich. In OMNeT++ lassen sich die Simulationen über die Kommandozeile oder über eine Benutzeroberfläche anschauen. Ersteres ist besonders dann sinnvoll, wenn man viele Simulationen durchführen möchte, ohne an einem einzelnen Simulationsdurchlauf interessiert zu sein.

Die Benutzeroberfläche wird über die plattformunabhängige Software Qt bereitgestellt [8]. Ein OMNeT++-Projekt erhält immer eine oder mehrere ini-Dateien - oft heißt diese omnet.ini - mit Konfigurationsoptionen. OMNeT++ verfügt mit NED (Network Description) über eine eigene Spezifikationssprache von Simulationsmodellen. In dieser lassen sich Parameter definieren, die dann entweder direkt oder in der ini-Datei definiert werden können. OMNeT++ erlaubt das Aufzeichnen von Log-Dateien zur späteren Analyse der Ergebnisse. Nach der Installation stellt OMNeT++ eine umfassende Sammlung von Werkzeugen für die Kommandozeilen bereit mit denen Simulationen durchgeführt und ausgewertet werden können.

2.3.1. Network Description

Mit Network Description (NED) verfügt man innerhalb von OMNeT++ über eine Sprache um Simulationskomponenten zu definieren. Komponenten können hierarchisch organisiert werden, damit einzelne Module aus mehreren kleineren zusammengesetzt werden können. Komponenten sind in Paketen sortiert, die sich nach Bedarf importieren lassen. Namenskonflikte werden so vermieden. Komponenten können nach Bedarf auch von anderen Komponenten erben, dabei übernehmen sie alle Eigenschaften der Elternkomponente. Mehrere Komponenten können als ein Netzwerk zusammengesetzt werden. Dieses bildet dann ein Simulationsmodell.

Die einfachsten Module werden mit dem Keyword *simple* definiert. Ihre Programmlogik befindet sich dann in C++-Dateien.

Zur Verwendung eines Moduls ist keine Kenntnis der zu Grunde liegenden Module notwendig. Mithilfe der Strukturierung dieser Module vermeidet man das Definieren redundanter Module. Die Verwaltung der Module wird für den Benutzer zugänglicher, weil er sich durch die zusätzliche Abstraktion nicht mit den Details der Module auseinandersetzen muss.

In der omnet.ini hat man die Möglichkeit, die in der NED-Datei definierten Netzwerke und Module zur Simulation zu nutzen. Die Parameter einzelner Module können gesetzt werden.

2.3.2. INET

INET stellt eine in C++ geschriebene, quelloffene Erweiterung von OMNeT++ dar, die OMNeT++ eine Reihe an Internetprotokollen und spezialisierten Netzwerkgeräten zur Verfügung stellt. INET ist modular aufgebaut und gliedert sich in verschiedene Bereiche nach dem OSI-Schichtenmodell. Je nach Bedarf können einzelne Projektbestandteile ausgeschaltet werden, um die Kompilationszeit zu reduzieren [1].

Um TSN mit INET zu realisieren, benötigt man verschiedene Module zum Erstellen von Netzwerken. Hierbei sind für diese Arbeit vor allem relevant:

1. TSN-Devices, die als Client und Server dienen können.
2. TSN-Switches erweitern die normalen Ethernet-Switches.
3. TSN-Clock, die als gPTP-Instanz dienen kann, aber keinen anderen Netzwerkverkehr zulässt.

Auf alle genannten TSN-Komponenten lassen sich Uhren konfigurieren. Je nach Anwendungsgebiet kann man 3 Grundtypen unterscheiden:

1. Eine ideale Uhr, deren Zeit mit der Simulationszeit übereinstimmt. Ohne eine andere Uhr einzurichten, wäre das die Ausgangslage für den Switch.
2. Eine oszillierende Uhr, die eine Drift-Rate besitzt und damit von der Simulationszeit abweichen kann. Der eigentliche Drift der Uhr ist dabei entweder
   a) konstant, die Drift-Rate ändert sich damit nicht oder

   b) zufällig, die Drift-Rate wandert in einem festgelegten Bereich, mit einer Änderungsrate des Drifts.

3. Die in 1. und 2. genannten Uhren lassen sich nicht verstellen, dafür benötigt man die verstellbare Uhr. Sie bildet eine Subklasse der oszillierenden Uhr und erweitert sie lediglich um die Fähigkeit sie im C++-Code oder durch ein Ereignis im Szenario anzupassen. Das ist nötig für Uhrensynchronisationen, beispielsweise mit Hilfe von gPTP.

Das gPTP-Modul lässt sich auf allen TSN-fähigen Netzwerkgeräten aktivieren und individuell konfigurieren. Genaue Konfigurationsoptionen erlauben es, die Intervallen, in denen gPTP-Nachrichten gesendet werden, festzulegen.

Mit der TSN-Funktionalität von INET lassen sich realistische Simulationen in zeitsensitiven Netzen durchführen [4] [11].

3. Konzept

Eine einfache Gegenüberstellung eines Netzwerks mit und ohne Uhrendrift werden aufzeigen, ob

1. der selbst erstellte Schedule akkurat ist und
2. ob Gegenmaßnahmen gegen den Uhrendrift notwendig sind.

Das Netzwerk wird dabei mit OMNeT++ modelliert. Zusätzlich wird die INET-Erweiterung verwendet, um TSN-Funktionalität bereitzustellen. Die Simulation ist deterministisch, produziert deshalb bei gleichen Startparametern dieselben Ergebnisse und muss deshalb nicht wiederholt werden.

Die Robustheit eines Schedules ist dabei die Fähigkeit des Schedules auf Störungen im Netzwerk zu reagieren. Eine höhere Robustheit führt also zu geringen Latenzen, Jitter und Queue-Längen.

3.1. Erwartungen

Das Senden von zeitgesteuertem Netzwerkverkehr sollte nur mit einheitlichen Uhren funktionieren. Mit der zusätzlichen Bedingung eines korrekten Schedules, sollte es in diesem Fall keinerlei Verzögerungen bei der Übertragung von Nachrichten im Netzwerk geben. Hier dürfte die Latenz für jedes Paket konstant verlaufen und damit sind Jitter und Queue-Länge sehr klein.

Wenn die Uhren der Geräte im Netzwerk durch einen Drift nicht synchron sind, kann es zu Verzögerungen kommen. Diese Verzögerungen können sich durch eine höhere Latenz, einen höheren Jitter oder durch vollere Queues bemerkbar machen.

Bei einem fixen Offset dürften diese Effekte vergleichsweise weniger stark auftreten, weil jeder Offset nach einiger Zeit durch den Drift erreicht wird.

3.2. Konfiguration

Das Netzwerk besteht aus mehreren Switches, um die Auswirkungen von Ungenauigkeiten bei der Uhrensynchronisation zu verstärken, weil jeder Switch eine eigene Uhr mit variablem Drift und Offset besitzen kann.

Vor diesem Hintergrund bildet das Netzwerk eine Linientopologie mit 6 TSN-Switches, 6 Clients und einem Server. Jeder Switch ist mit einem Client verbunden. Und am sechsten Switch befindet sich neben einem Client auch ein Server. Die Netzwerktopologie ist in Abbildung 3.1 abgebildet. Die Switches sind dabei TSN-Switches und die Clients

und der Server TSN-Geräte aus dem INET-Paket. Bei den Verbindungen handelt es sich um Ethernet-Verbindungen. Alle Ethernet-Verbindungen sind auf eine Bandbreite von 100 Mbps festgelegt. Eine Linientopologie findet sich häufig in der realen Welt wieder.

Die Länge der Linie wurde so gewählt, um die Visualisierung der Ergebnisse zu vereinfachen, bei gleichzeitiger Ermöglichung von Fehlern.

Die Uhren auf den Switches sollten nur im suboptimalen Experiment einen Drift oder einen Offset erhalten.

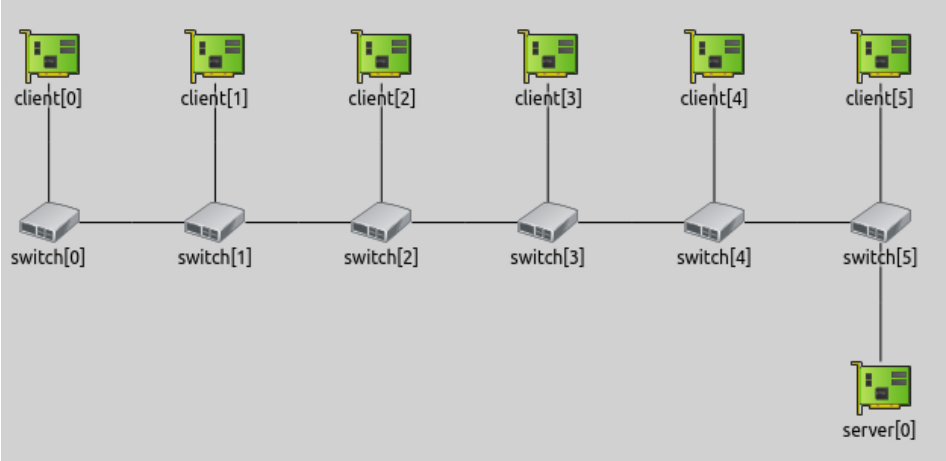


Abbildung 3.1.: Linientopologie mit 6 Switches, 6 Clients und einem Server. Verbunden über Ethernet.

Das Netzwerk wird in der omnet.ini konfiguriert. Auf allen Clients wird eine UDP-Applikation konfiguriert, die in fixen Intervallen von 200µs Pakete zum Server sendet. Die Pakete haben eine Größe von 179 Byte bzw. 1432 Bit, die Übertragungsdauer eines Pakets von einem Netzwerkgerät zu einem anderen ist damit

$$\frac{1432\text{bit}}{100\text{Mbps}} = 14{,}32\text{µs}.$$

Der Server hat 6 verschiedene Applikationen, die auf unterschiedlichen Ports laufen und Pakete empfangen. Jeder Client n sendet die Pakete so, dass sie erwartungsgemäß zum gleichen Zeitpunkt am Ingress des Switches n ankommen, wie der Netzwerkverkehr von Switch $n-1$. Auf diesem Wege entsteht ein potentieller Konflikt zwischen dem von Client n gesendeten Paket und dem ersten von Switch $n-1$ gesendeten Paket. Die genaue Konfiguration befindet sich in Tabelle A.1.

Bei gleichzeitigem Ankommen zweier Pakete wird jenes zuerst in die Queue aufgenommen, welches vom angrenzenden Client stammt. Das wird durch eine selbstgewählte Konfigurationsoption in OMNeT++ gewährleistet. Diese werden dann zuerst an den Switch $n+1$, bzw. für Switch 5 an den Server gesendet. Andernfalls entstünden schwer nachzuvollziehende Situationen, in denen kontraintuitiv der Netzwerkverkehr von Client 0 vor den Paketen der anderen Clients am Server ankommt. Selbst wenn die Uhren der Switches synchron laufen und der Schedule optimal gewählt wurde.

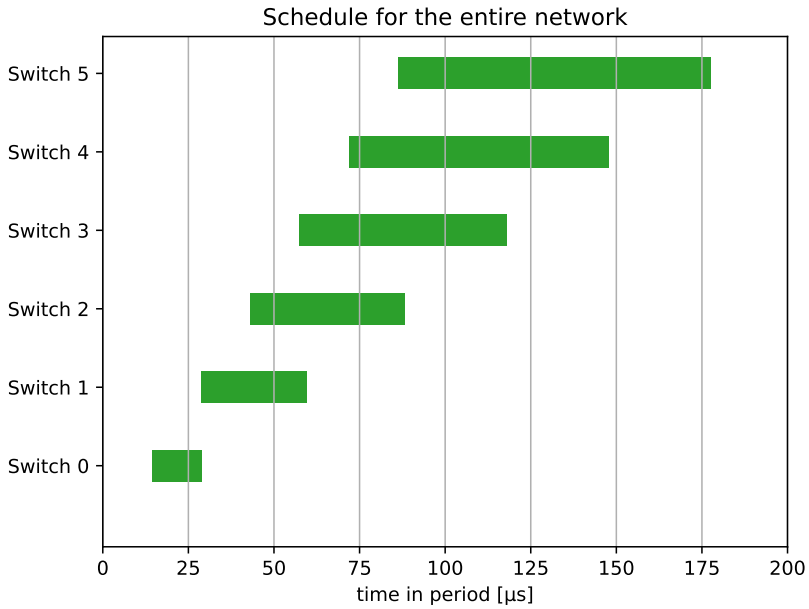


Abbildung 3.2.: Diagramm, das einen optimalen Schedule für das Netzwerk darstellt.

Jeder Switch besitzt ein einziges Gate mit einer Queue. Jede Queue hat eine unbegrenzte Kapazität an Paketen.

Als Periode für die zeitgesteuerten Gates dienen 200 µs. Die Gates haben am Anfang immer einen Zeitslot der Länge x, in denen sie offen sind. Im darauffolgenden Zeitfenster sind die Gates dann wieder für die Periodendauer $- x$ geschlossen, für das aktuelle Experiment also 200µs $- x$. Der Offset bestimmt, ab welchem Punkt das offene Zeitfenster in der aktuellen Periode anfängt. Alle Zeitfenster sind minimal bemessen und geplant, d.h. eine Verkleinerung der Zeitfenster würde dazu führen, dass die Pakete nicht mehr in der aktuellen Periode an ihr Ziel kommen, weil sie an einem Switch nicht mehr rechtzeitig übertragen werden können. Die offenen Zeitfenster der Switches mit höherer ID werden größer gewählt, weil durch den Aufbau als Linientopologie durch sie mehr Netzwerkverkehr fließt. Die Offsets müssen gleichermaßen graduell größer gewählt werden, weil die ersten Pakete sie später in der Periode erreichen.

Zur Visualisierung des Schedules dient Abbildung 3.2. Auf der x-Achse befindet sich die Zeit innerhalb einer Periode, also von 0µs bis 200µs. Auf der y-Achse sind die einzelnen Switches abgebildet. Die grünen Balken zeigen für jeden Switch an, wann deren einziges zeitgesteuertes Gate offen ist. Man sieht deutlich, dass die Gates der Switches mit höherer ID länger offen sind und wie die Offsets für die einzelnen Gates wachsen.

Die genauen Zeitfenster und Offsets der einzelnen Gates sind im Anhang in Tabelle A.2 aufgeführt.

3.3. Netzwerk im Optimalfall

Im Optimalfall verfügen die Switches über perfekte Uhren, die mit der Simulationszeit übereinstimmen. Sie besitzen weder einen Offset noch einen Drift, der die Übertragungsmetriken negativ beeinflussen könnte.

Die Simulation wird für 5 Sekunden ausgeführt. Wichtige Daten wie Latenz, Jitter und Queue-Länge werden aufgezeichnet. Die Gates sind wie in Abschnitt 3.2 konfiguriert.

In Abbildung 3.3a befindet sich auf der x-Achse der Zeitpunkt in Millisekunden, wann das Paket abgesendet wurde und auf der y-Achse befindet sich die Latenz des Paketes in Mikrosekunden. Jede Applikation hat eine eigene gefärbte Linie, dabei gehört Applikation n zu Client n, siehe dazu Abbildung 3.1. Die Latenz der Übertragungen bei allen Paketströmen konstant. Dabei bestehen feste Abstände zwischen den Applikationen. Zwischen der Latenz zwei benachbarter Applikationen besteht eine konstante Differenz von 31,65µs. Dieser Abstand ergibt sich auch aus den längeren Übertragungswegen, die die Pakete einer weiter entfernteren Applikation durchlaufen müssen und aus den Wartezeiten, die während der Übertragung entstehen. Der Anteil des Übertragungsweges an der Latenz l_{Weg} einer Applikation n ergibt sich aus der Anzahl der Hops zwischen Client und Server. Ohne die Interferenz anderer Applikation zu betrachten, ist

$$l_{\text{Weg}} = 2 \cdot (5 - n) \cdot 14{,}32\text{µs} + 14{,}32\text{µs}.$$

Das gilt insbesondere für Applikation 5, weil diese den Netzwerkverkehr nur über den Switch 5 an den Server sendet. Der Netzwerkverkehr dieser Applikation muss wegen der Priorisierung bei gleichzeitigem Ankommen zweier Pakete nicht warten und wird direkt an den Server übertragen. Die Latenz der Applikation 5 beträgt $2 \cdot 14{,}32\text{µs} = 28{,}64\text{µs}$.

Die Pakete aller anderen Applikation müssen vor ihrer vollständigen Übertragung an den Server in einer Queue auf die Übertragung des Netzwerkverkehrs anderer Applikationen auf den Switches warten. Die totale Wartezeit während der Übertragung eines Paketes der Applikation n von Client nach Server l_{Warten} ist für alle Applikationen außer 5 gleich, weil die Pakete der Applikation n an Switch $n + 1$ auf die Übertragung des Netzwerkverkehrs der Applikation $n + 1$ einmalig warten müssen. Somit ist

$$l_{\text{Warten}} = 14{,}32\text{µs}.$$

Für die Gesamtlatenz l gilt dann

$$l = l_{\text{Weg}} + l_{\text{Warten}}.$$

Der Jitter der einzelnen Paketströme hängt von der Varianz der Latenzen ab. Die Latenz weist für einen Paketstrom im Graph gut erkenntlich keine Varianz auf und damit ist der Jitter vernachlässigbar klein. Die Versuchsdaten geben gleichermaßen null zurück.

In Abbildung 3.5c sind die durchschnittliche und die maximale Queue-Länge für die einzelnen Switches abgebildet. Auf der x-Achse befinden sich die einzelnen Switches, nummeriert von 0 bis 5 und auf der y-Achse die Anzahl an Paketen. Die grünen Balken

stehen für die durchschnittliche Paketanzahl und die roten für die maximale Paketanzahl während der Simulation. Die Balken sind für alle Switches ≤ 1, weil sich in einem perfekten Schedule in den Queues der Switches maximal ein Paket in den Queues befinden kann. Die Balken sind bei Switch 0 überhaupt nicht vorhanden, weil alle Pakete in Switch 0 von Client 0 gesendet werden und damit keine Konflikte mit anderen Paketen entstehen können.

Hiermit ist die Korrektheit des Schedules für das Netzwerk in Abbildung 3.1 hinreichend gezeigt.

3.4. Netzwerk mit fehlerhaften Uhren

Die Uhren des Netzwerks verhalten sich jetzt wie im echten Leben und erfahren einen Uhrendrift und erhalten einen Offset. Jeder Offset stellt sich nach gewisser Zeit automatisch ein, wenn man davon ausgeht, dass der Uhrendrift konstant bleibt und in dieselbe Richtung verläuft. Um das zu zeigen, wird das Netzwerk zuerst nur mit einem Offset betrieben.

3.4.1. Uhren mit Offset

Die einzelnen Offsets der Uhren werden in Tabelle A.3 aufgelistet. Die Uhren erfahren keinen Drift, deshalb ist der Offset für die gesamte Simulationsdauer konstant. Der Offset ist groß genug gewählt, um die Schedules der Switches zu stören.

In Abbildung 3.3b ist die Auswirkung des Uhren-Offset auf die Latenz zu sehen. Auf der x-Achse befindet sich der Simulationszeitpunkt, wann das Paket abgesendet wurde in µs und auf der y-Asche die Latenz des Pakets in µs. Jede Applikation bekommt ihre eigene Kurve. Die Latenz ist bei allen Applikationen entweder gleich oder größer als im Vergleichsexperiment in Abschnitt 3.3. Sie bleibt allerdings für alle Applikationen konstant. Die höheren Latenzen sind durch die Verschiebung der Zeitfenster zu erklären, weil einige Pakete nicht mehr in der aktuellen Periode gesendet werden können. Die unterschiedlichen Abstände zwischen den Latenzen sind auf die Tatsache zurückzuführen, dass die Pakete der Applikationen 0 und 1 einen längeren Weg zurücklegen müssen und deshalb mehr als ein Zyklus benötigen um am Server anzukommen.

Die Latenzen sind für die Dauer des Experiments konstant und erfahren keine Schwankungen. Der Jitter als Variation der Latenz ist somit null.

Aufschluss darüber gibt Abbildung 3.5b, auf der x-Achse sind die verschiedenen Switches abgebildet und auf der y-Achse befindet sich die Queue-Länge. Die beiden Balken stehen für die durchschnittliche und maximale Queue-Länge. Anders als im Optimalfall befinden sich in den Queues jetzt im Maximum ein Paket mehr in jeder Queue.

In Kombination dieser Fakten lässt sich sagen, dass der Uhr-Offset nur eine konstante Verschlechterung der Latenz und der Queue-Länge für die gesamte Dauer des Experiments bewirkt, weil die Verschiebung der Schedules für die Dauer des Experiments gleich bleibt. Ein Paket, was in der aktuellen Periode nicht mehr gesendet werden kann, wird erst in der nächsten Periode gesendet. Dabei sollte ein anderes Paket dieses Zeitfenster nutzen. Dieses muss dann warten.

Jeder Uhren-Offset stellt sich bei Uhren mit konstantem Drift nach einer gewissen Zeit ein. Für weitere Betrachtung ist die Hinzunahme des Offset als Startkonfiguration nicht nützlich für den Erkenntnisgewinn und wird deshalb vernachlässigt.

3.4.2. Uhren mit Drift

Die Uhren der einzelnen Switches erfahren einen realistischen Uhrendrift und stimmen am Anfang der Simulation mit der Simulationszeit überein. Die Datenwerte sind in Tabelle A.4 zu sehen. Das Alternieren der Vorzeichen bei den Uhrendrifts ist ein plausibler Extremfall, weil die Differenz der Uhrendrifts zwischen zwei benachbarten Switches maximiert wird, wenn der Betrag jedes Uhrendrift begrenzt ist. Eine Wahl eines variablen Drift würde die Durchführung von Experimenten durch die nichtdeterministische Natur dieser erschweren.

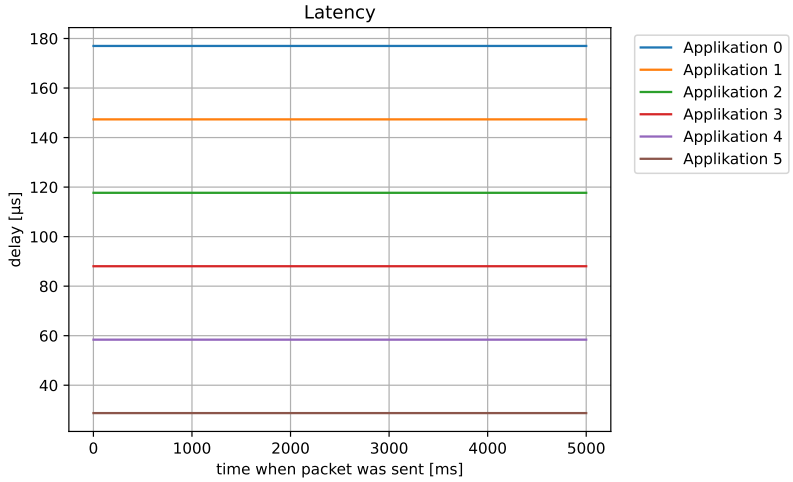
In Abbildung 3.3c befindet sich auf der x-Achse die Zeit, wann das Paket gesendet wurde in Millisekunden. Auf der y-Achse ist die Lebensdauer bzw. die Latenz des Pakets in Mikrosekunden dargestellt. Die Latenzen der Applikationen sind weiterhin aufsteigend größer. Das Szenario führt mit Uhrendrift zu einem unbegrenzten Wachstum der Latenz bei den Applikationen von 0 bis 4. Applikation 0 steigt in 5 Sekunden zum Ende der Simulation zu einem Wert von 3,5ms. Die Kurven der Applikationen 1 und 2 liegen nah beieinander, beide enden bei 2,0ms. Die Latenz der Applikationen 3 und 4 verlaufen weniger steil und enden bei 0,75ms. Die Latenzen verlaufen bei diesen Applikationen ähnlich, weil sie sich gleichlang in den Queues der Switches stauen. Der Netzwerkverkehr von Applikation 5 legt einen deutlich kürzeren Weg zurück und muss nur über einen Switch an den Server gelangen, deshalb steigt dessen Latenz nicht ungehindert weiter und bleibt unter 0,2ms. Die Latenzen steigen, weil sich für die Dauer der Simulation mehr Pakete in den Queues ansammeln und damit benötigen die Pakete länger, um am Server anzukommen. Generell lässt sich sagen, dass je mehr Hops zwischen Client und Server liegen, desto schneller steigt die Latenz und desto höher ist sie nach Simulationsende.

Um die Variation der Latenz besser darzustellen, gibt Abbildung 3.4 den Jitter wieder. Auf der x-Achse befindet sich der Zeitpunkt des Sendens in Millisekunden und auf der y-Achse der Jitter j, als Varianz der Latenz mit Einheit μs^2, berechnet mit

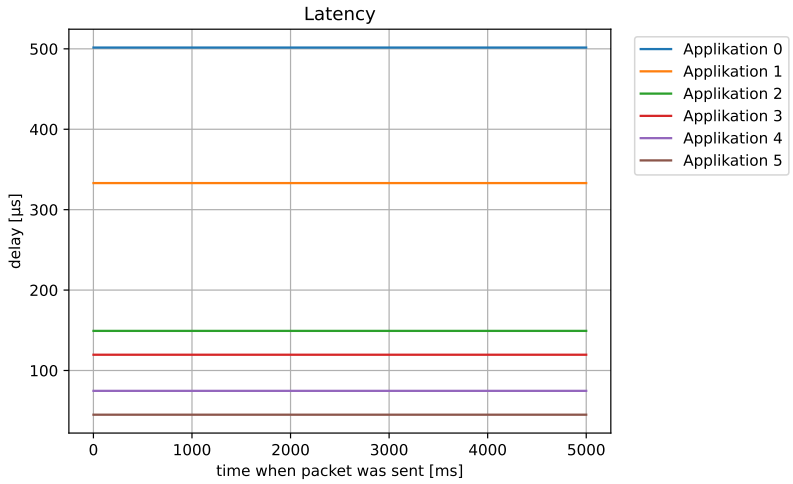
$$j = \frac{\sum_{i=1}^{n}(l_i - \bar{l})^2}{n}.$$

Dabei ist n die Anzahl der bisher gesendeten Pakete, l_i die Latenz von Paket i und $\bar{l}$ die durchschnittliche Latenz aller bereits gesendeten Pakete. Der Datenwert zu einem Zeitpunkt gibt also die Varianz der Latenz der bisher gesendeten Pakete an. Der Jitter wächst bei den Applikationen von 0 bis 4 ungehindert bis zum Simulationsende. Bei Applikation 0 wächst der Jitter nicht ungehindert und bleibt unter $50\mu s^2$. Das ist auf die nahezu konstante Latenz des Netzwerkverkehrs der Applikation 5 zurückzuführen. Der Jitter der Applikationen 1 und 2 verläuft fast identisch, genauso wie der Jitter der Applikationen 3 und 4. Der Jitter wächst in beiden Fällen bis zum Simulationsende gleich, weil sich auch die Latenz dieser Applikationen gleich verhält.

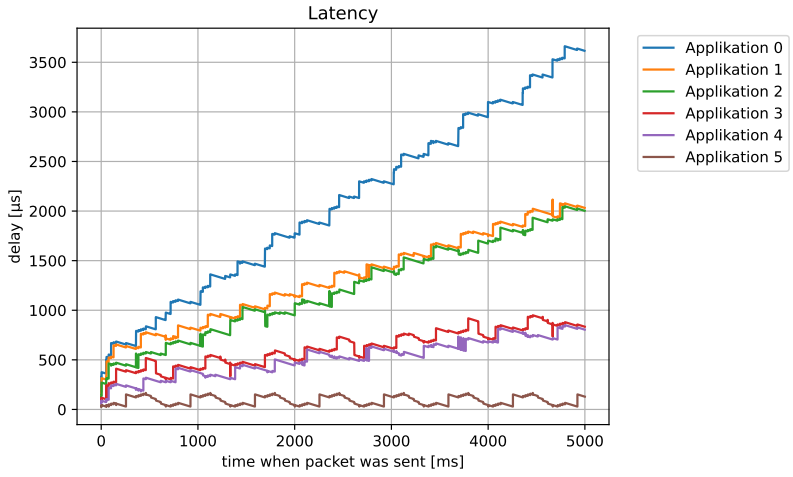
Die Queue-Länge des aktuellen Experiments ist in Abbildung 3.5c dem Experiment im Optimalfall gegenübergestellt. Die x-Achse bildet wieder die 6 verschiedenen Switches von 0 bis 5 ab. Auf der y-Achse ist die Anzahl der Pakete in der Queue abgebildet. Es sammeln sich in allen Queues des Switches deutlich mehr Pakete an. Besonders in den Queues der Switches 0, 2 und 4 sind durchschnittlich und im Maximum deutlich mehr Pakete vorhanden als im Optimalfall vorhanden. Dieser Kontrast ist auf den unterschiedlichen Drift, den die Uhren auf den Switches erfahren, zurückzuführen. Der Uhrendrift auf den Switches 0, 2 und 4 ist mit -300ppm negativ und führt bei ungehindertem Uhrendrift auf Dauer zu einem Stau von Paketen. Diese Effekte lassen sich nur bei ungehindertem Uhrendrift beobachten, weil der angesammelte Drift die Periodendauer übersteigen muss. Die Queues der Switches 1, 3 und 5 beinhalten weniger Pakete als die anderen Switches, allerdings noch deutlich mehr als im Optimalfall. Die durchschnittliche Paketmenge ist ungefähr halb so groß wie die maximale Paketanzahl, was darauf hindeutet, dass die Pakete in den Queues sich für die meiste Zeit der Simulation graduell ansammeln und nicht nur am Ende in den Queues eintreffen. In erster Linie erreichen die Pakete nicht am Anfang der Simulation ihr Maximum in den Queues.



(a) Das Diagramm zeigt die Latenz unter optimalen Bedingungen; ohne Uhrendrift und Offset.



(b) Das Diagramm zeigt die Latenz bei fehlerhaften Uhren, die lediglich einen konstanten Offset besitzen.



(c) Das Diagramm zeigt die Latenz bei fehlerhaften Uhren, die einen realistischen Uhrendrift besitzen.

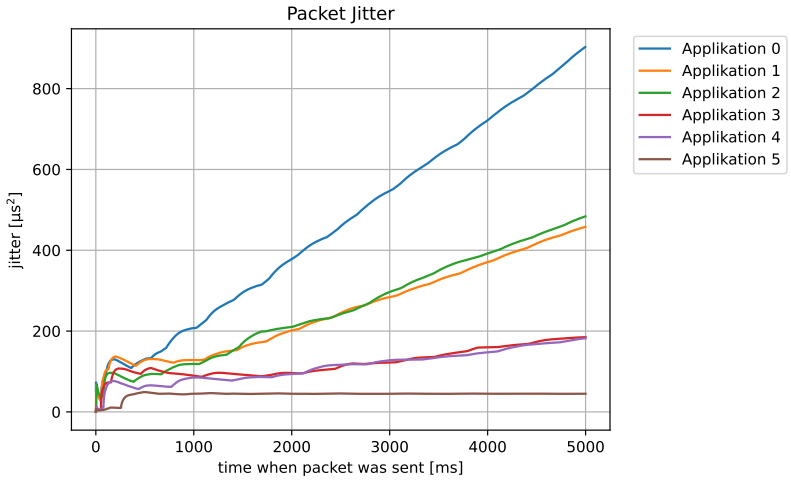
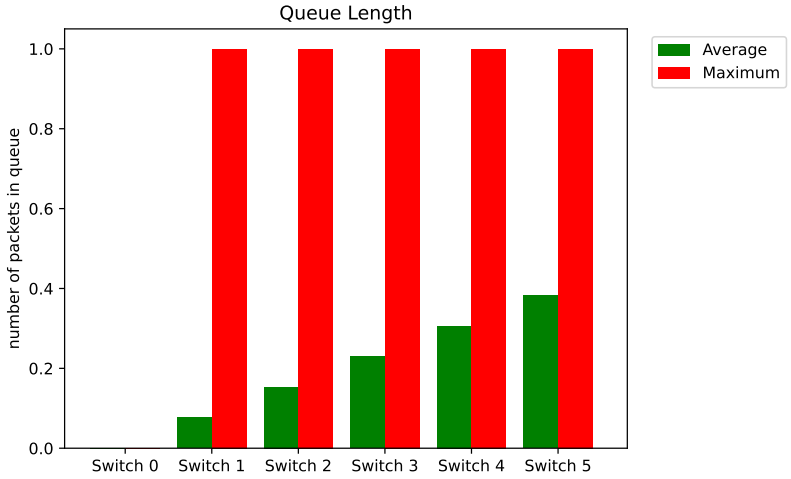
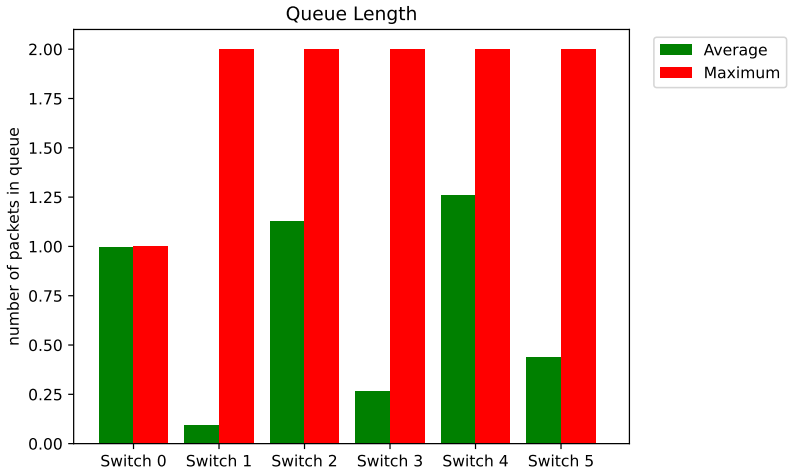


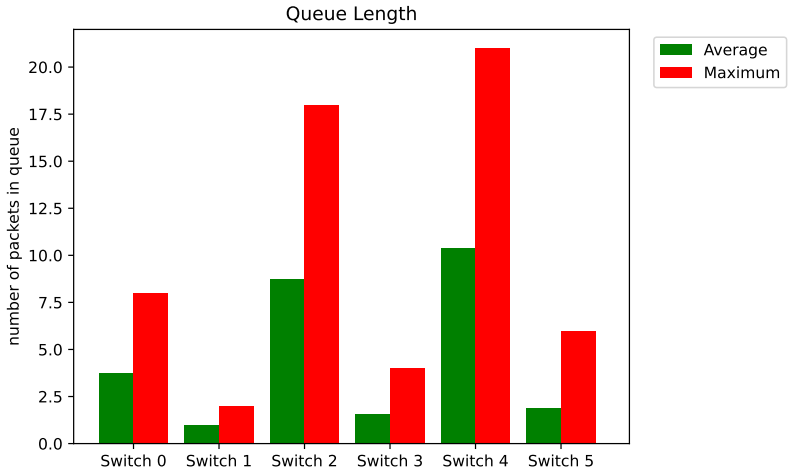
Abbildung 3.4.: Das Diagramm zeigt den Jitter im Experiment mit Uhrendrift.



(a) Das Diagramm zeigt die Queue-Länge unter optimalen Bedingungen; ohne Uhrendrift und Offset.



(b) Das Diagramm zeigt die Queue-Länge bei fehlerhaften Uhren, die lediglich einen konstanten Offset besitzen.



(c) Das Diagramm zeigt die Queue-Länge bei fehlerhaften Uhren, die einen realistischen Uhrendrift besitzen

4. Lösungen

Zur Lösung der oben genannten Probleme kommen folgende Gegenmaßnahmen in Erwägung:

Größere Zeitfenster. Die Zeitfenster der Gates werden vergrößert, denn bei größeren Zeitfenstern ist es unwahrscheinlicher, dass die Pakete ihren Zeitslot verpassen. Es lassen sich unterschiedliche Vergrößerungen der Zeitfenster gegenüberstellen, um zu sehen, wie stark man die Zeitfenster für eine Verbesserung der Vergleichsmetriken anpassen muss.

Uhrensynchronisation. Dem Uhrendrift kann mit einer Zeitsynchronisation entgegengewirkt werden. Wenn alle Switches die gleiche Zeit haben, sollte eine reibungslose Übertragung möglich sein. Für die Synchronisation kommt gPTP in Frage. Mit gPTP fällt allerdings auch neuer Netzwerkverkehr an. Dieser muss in der Topologie Beachtung finden.

Diese Lösungsansätze werden in diesem Kapitel näher beleuchtet.

4.1. Größe der Zeitfenster

Die Überlegung ist es, durch eine Vergrößerung der einzelnen Zeitfenster der Schedules eine Verbesserung der Vergleichsmetriken wie Latenz, Jitter und Queue-Länge zu erzielen. Dabei wird ein Zusammenhang zwischen Zeitfenstergröße und Robustheit des Schedules hergestellt.

4.1.1. Konfiguration

Das Netzwerk unterscheidet sich in seiner Topologie nicht von dem Netzwerk in Kapitel 3. Lediglich die Konfiguration der Switches variiert. Insbesondere die Größe der Zeitfenster der Gates wird verändert.

Die Applikationen und Uhren im Netzwerk sind wie in Kapitel 3 konfiguriert.

Die Vergrößerung der Zeitfenster wird in 3 aufsteigend größeren Varianten getestet:

1. 25% Vergrößerung
2. 50% Vergrößerung
3. 100% Vergrößerung

Zur Visualisierung der Konfiguration dienen Abbildung 4.1a, Abbildung 4.1b und Abbildung 4.1c. Auf der x-Achse ist die Zeit innerhalb einer Periode abgebildet. Auf der y-Achse befinden sich die 6 Switches von 0 bis 5. Die grünen Balken zeigen, wann

die Gates der einzelnen Switches offen sind. Wenn ein Zeitfenster die aktuelle Periode übersteigt, wird dieses auch in der nächsten Periode für die noch verbleibende Zeit offen gelassen. Das sieht man im Schedule durch zwei getrennte grüne Regionen.

Die genauen Konfigurationen sind in Tabelle A.5, Tabelle A.6 und Tabelle A.7 zu sehen.

Die 3 unterschiedlichen Schedules erlauben es einen Zusammenhang zwischen Zeitfenstergröße und Robustheit der Schedules herzustellen. Für die Dauer des fünfsekündigen Experiments werden Metriken wie Latenz, Jitter und Queue-Länge aufgezeichnet.

4.1.2. Latenz

In Abbildung 4.2a, Abbildung 4.2b, Abbildung 4.2c sieht man auf der x-Achse die Zeit in Millisekunden, wann das Paket gesendet wurde. Auf der y-Achse befindet sich die Latenz in Mikrosekunden. Jede Applikation enthält einen in der Legende beschriebenen und unterschiedlich gefärbten Funktionsgraph.

Für alle Experimente lässt sich sagen, dass die Latenz von Applikation n für einen beliebig gewählten Zeitpunkt stets kleiner ist als die Latenz der Applikation $n+1$. Das liegt an der größeren Distanz, die die Applikationen mit geringerer Nummer zurücklegen müssen.

Im Fall von 25% größeren Zeitfenstern wächst die Latenz der Applikationen 0 für die Dauer der Simulation. Die maximale Latenz der Applikation ist wenige Millisekunden vor Simulationsende mit 3050µs erreicht. Das ist eine geringfügige Verbesserung gegenüber dem Experiment in Unterabschnitt 3.4.2. Die Latenzen der Applikationen 1 und 2 wachsen ebenso unbegrenzt, erreichen allerdings nur eine Endwert von 1250µs. Die Latenzen der Applikationen 3 bis 5 wachsen nicht unbegrenzt, sondern schwanken zwischen 0µs und 400 µs.

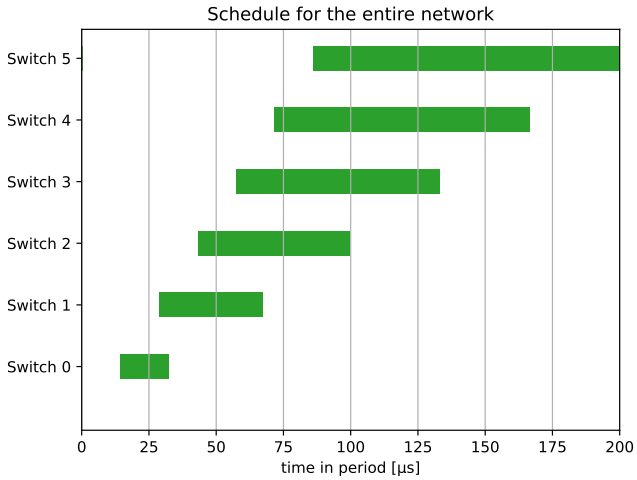
Bei einer Fenstergröße von 50% wächst nur die Latenz der Applikation 0 unbegrenzt weiter und erreicht am kurz vor Ende der Simulation eine maximale Latenz von 1900µs. Die anderen Latenzen bewegen sich unterhalb von 510µs.

Bei Fenstergrößen von 100% sind alle Latenzen beschränkt und wachsen nicht über 550µs hinaus. Die Latenzen verlaufen periodisch. Sie sinken auf ihr Minimum, welches am Anfang der Simulation bestand. Danach wiederholt sich der Verlauf der Latenzen. Die genaue Zeit ergibt sich aus dem Drift der Uhren mit 300ppm beziehungsweise -300ppm. Die genaue Zeit, die die Uhren brauchen um wieder bei ihrer Ausgangssituation anzukommen und den Schedule temporär zu erfüllen, bestimmt sich mit

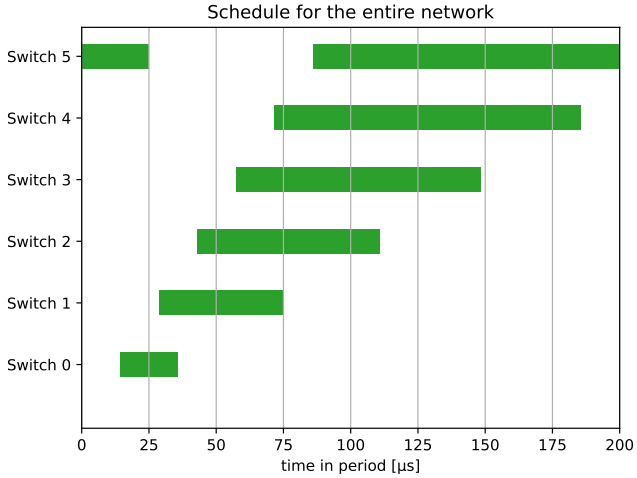
$$\frac{300\text{ppm}}{200\text{µs}} = \frac{2}{3}\text{s} \approx 667\text{ms}.$$

Das gleiche Verhalten lässt sich auch bei den anderen Zeitfenstergrößen beobachten. Die Periodendauer der Graphen bleibt wegen des festgesetzten Uhrendrifts für alle Experimente unverändert.

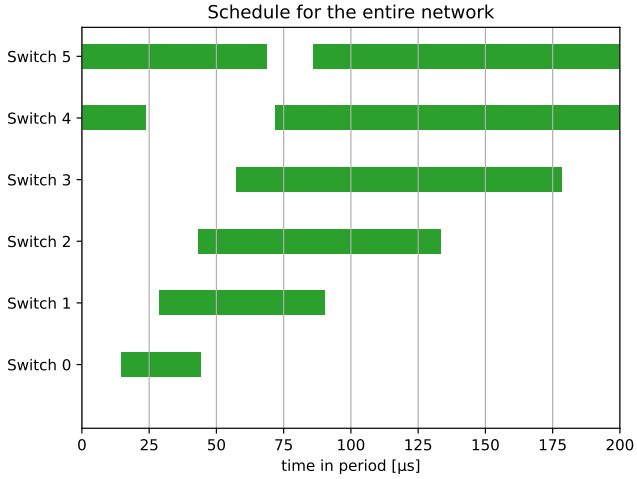
Die Zeitfenstergröße hat einen signifikanten Einfluss auf die Latenz im Experiment mit Uhrendrift. Bereits mit geringer Vergrößerung der Zeitfenster wachsen nicht mehr alle Latenzen unbegrenzt weiter. Die Applikationen, deren Latenzen trotz größerer Zeit-



(a) Schedule des Experiments mit 25% größeren Zeitfenstern.



(b) Schedule des Experiments mit 50% größeren Zeitfenstern.



(c) Schedule des Experiments mit 100% größeren Zeitfenstern.

Abbildung 4.1.: Schedules der verschiedenen Zeitfenstergrößen

fenster ungehindert wachsen, sind immer diejenigen, die weiter vom Server entfernt sind.

4.1.3. Jitter

In Abbildung 4.3a, Abbildung 4.3b und Abbildung 4.3c befindet sich auf der x-Achse die Zeit in Millisekunden und auf der y-Achse der Jitter in Mikrosekunden. Jede Applikation enthält einen in der Legende beschriebenen und unterschiedlich gefärbten Funktionsgraphen.

Der Jitter beginnt bei allen Applikationen bei 0 und wächst in den ersten 0,5s stark an. Danach stabilisiert er sich für alle Applikationen.

In Abbildung 4.3a wächst der Jitter der Applikation 0 ungehindert und endet bei einem Wert von 710µs^2. Der Jitter der Applikationen 1 und 2 wächst gemeinsam und erreicht nach 5 Sekunden 300µs^2. Bei den Applikationen 3 bis 5 wächst der Jitter nicht weiter und bleibt bei diesen unter 100µs^2.

In Abbildung 4.3b wächst nur der Jitter der Applikation 0 ungehindert weiter und endet bei 450µs^2. Die anderen Jitter bewegen sich unterhalb von 140µs und verändern sich nach einer Sekunde kaum. Bei der Applikation 3 verbessert sich der Jitter von 85µs^2 auf 60µs^2. Die Jitter der Applikationen 4 und 5 bleiben unverändert.

In Abbildung 4.3c wächst kein Jitter mehr unbegrenzt. Alle Jitter weisen einen festen Wert auf. Der größte Jitter ist bei Applikation 0 mit 110µs^2 zu finden, das stellt eine Verbesserung dar. Die anderen Applikationen erfahren auch eine Reduktion ihres Jitters. Für die Applikation 1 bedeutet das einen Jitter von 80µs^2. Applikation 2 endet bei 40µs^2 und die Applikation 3 bei 30µs^2. Die Applikation 4 bei 15µs^2 und Applikation 5 bei 8µs^2.

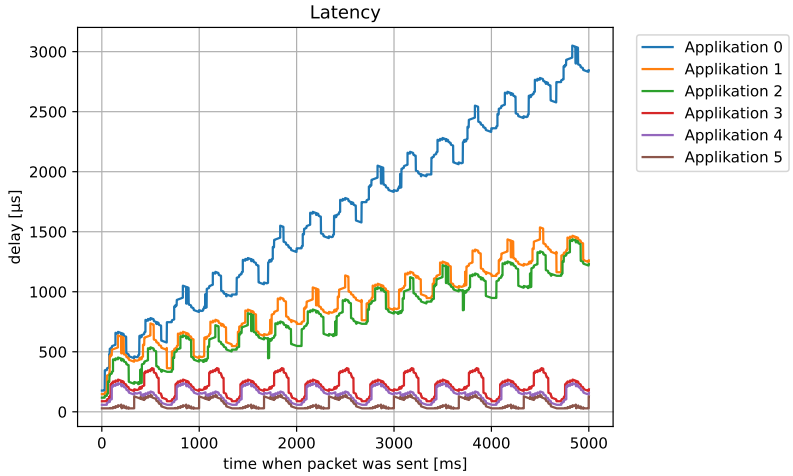
Generell ist der Jitter der Applikation n größer als der Jitter der Applikation n+1. Der Verlauf des Jitters verhält sich ähnlich zur Latenz. Wenn die Latenz in eine Richtung ungehindert wächst, dann wächst auch der Jitter ohne Begrenzung. Mit den Ergebnissen lässt sich sagen, dass eine Vergrößerung der Zeitfenster den Jitter reduziert.

4.1.4. Queue-Länge

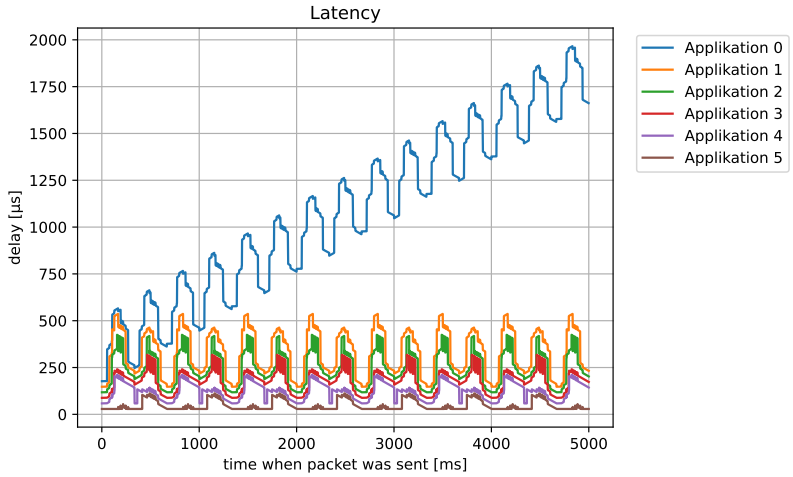
Die Queue-Länge für die einzelnen Experimente ist in Abbildung 4.4a, Abbildung 4.4b und Abbildung 4.4c dargestellt. Auf der x-Achse sind die verschiedenen Queues der Switches abgebildet und auf der y-Achse befindet sich die Anzahl der Pakete in der jeweiligen Queue. Die roten Balken stehen für die maximale Queue-Länge und die grünen Balken für die durchschnittliche Queue-Länge.

In allen Versuchsexperimenten sind in den Queues im Durchschnitt und im Maximum weniger Pakete in den Queues als im Experiment ohne Gegenmaßnahmen, dabei sind alle Ergebnisse aber schlechter als die Queue-Länge im Optimalfall mit synchronisierten und idealen Uhren.

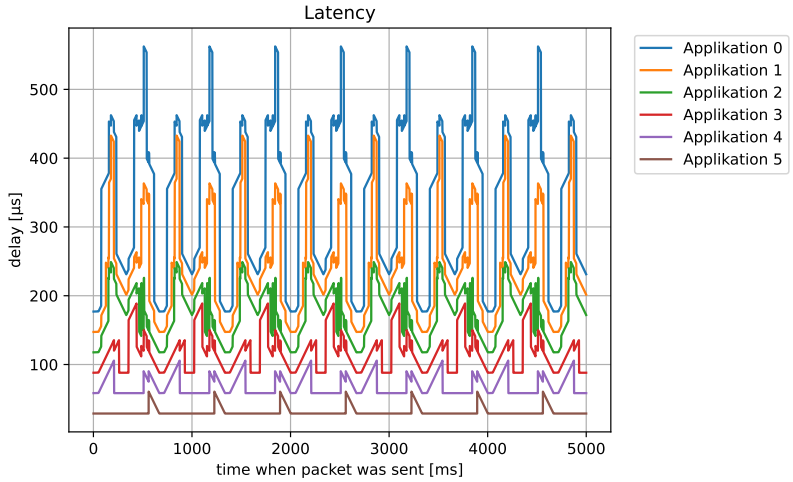
In Abbildung 4.4a tritt bei Switch 2 das Maximum aller Queues mit 18 Paketen auf. Das stellt bereits eine Verbesserung gegenüber dem Experiment ohne Gegenmaßnahmen dar. Dort sind zu einem Zeitpunkt 21 Pakete in der Queue des Switches 4. Switch 0 erfährt als einziger Switch keine Verbesserung, weil die Vergrößerung des Zeitfensters noch nicht für eine Übertragung von zwei Paketen pro Periode ausreicht. In jeder



(a) Latenz des Experiments mit 25% größeren Zeitfenstern.

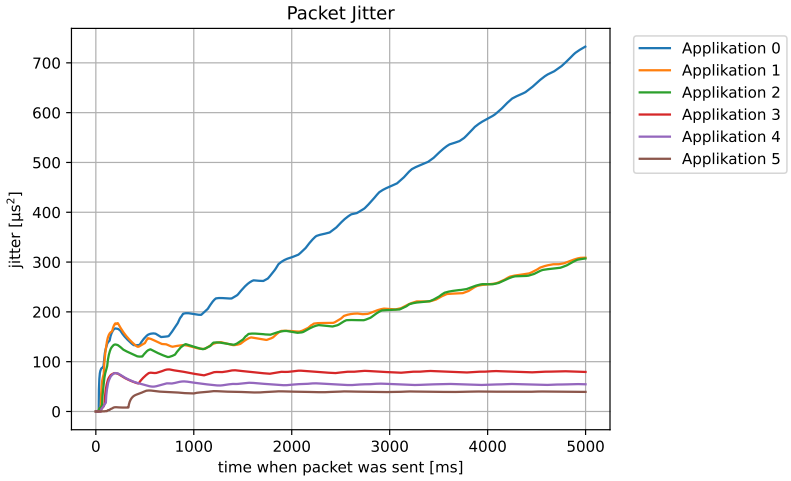


(b) Latenz des Experiments mit 50% größeren Zeitfenstern.

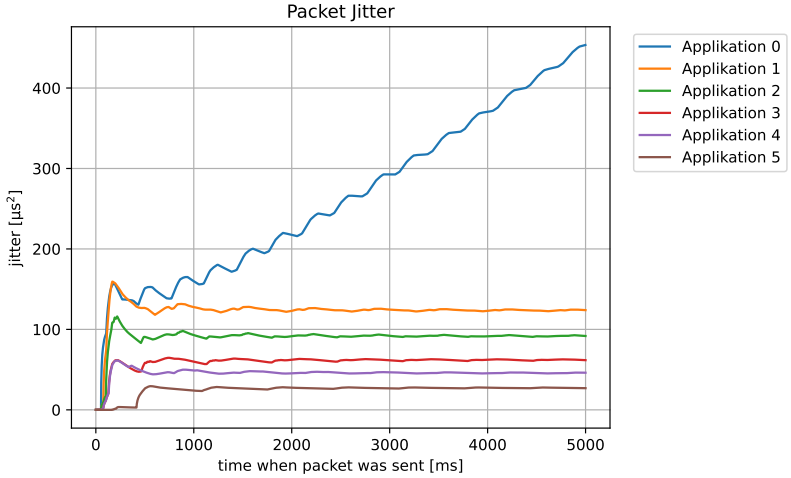


(c) Latenz des Experiments mit 100% größeren Zeitfenstern.

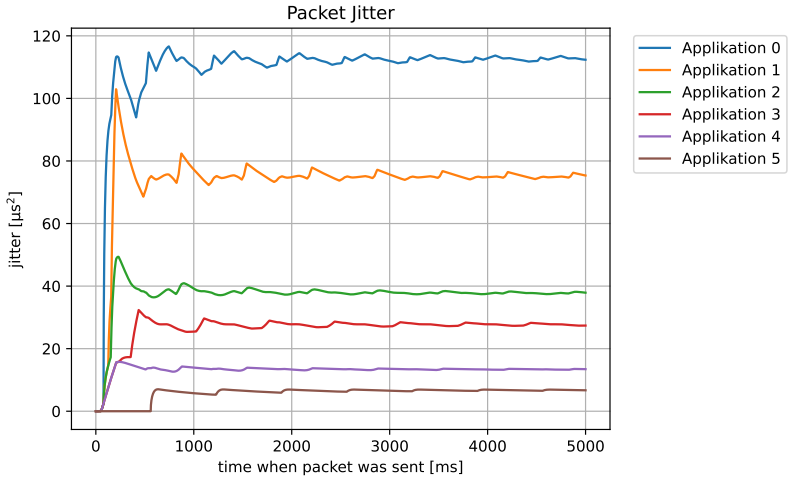
Abbildung 4.2.: Latenz der verschiedenen Zeitfenstergrößen



(a) Jitter des Experiments mit 25% größeren Zeitfenstern.



(b) Jitter des Experiments mit 50% größeren Zeitfenstern.



(c) Jitter des Experiments mit 100% größeren Zeitfenstern.

Abbildung 4.3.: Jitter der verschiedenen Zeitfenstergrößen

Periode kommt ein Paket an Switch 0 an und wird weiter gesendet. Durch den Drift sammeln sich allerdings langsam mehr Pakete in der Queue an, weil Pakete in einem offenen Zeitbereich am Switch ankommen, aber nicht mehr übertragen werden können. Das geschieht so oft, wie die Abweichung der Uhr die Periodendauer erreicht. Der Drift von 300ppm erreicht alle $\frac{2}{3}$s einen Wert von 200µs, was einer Periode entspricht. Bei einer Simulationsdauer von 5 Sekunden wären das 7,5 Perioden. Bei jeder Periode sammelt sich dauerhaft ein weiteres Paket im Switch an. Das sind im Maximum 8 Pakete in Switch 0.

In Abbildung 4.4b ist der Balken des Switches 2 drastisch reduziert. Auch die Paketanzahl der Switches von 3 bis 5 sind nun kleiner. Die Pakete bleiben in den Switches 0 und 1 gleich, weil der Anstieg der Zeitfenstergröße um 50% bei ihnen nicht ausreicht, um mehrere Pakete in einer Periode zu senden. Die durchschnittliche Queue-Länge ist bei den Switches 3, 4 und 5 bedeutend kleiner als die maximale Queue-Länge. Das bedeutet,dass die Pakete in Schüben ankommen, aber nicht lange in der Queue verweilen.

In Abbildung 4.4c ist das globale Maximum mit 4 Paketen in Switch 3. Auffällig ist Switch 0, der jetzt im Maximum nur noch ein Paket aufweist, weil bei einer um 100% vergrößerten Zeitfenstergröße genau zwei Pakete im Zeitfenster an den nächsten Switch gesendet werden können. Seine durchschnittliche Queue-Länge hat sich auf 0,5 reduziert. Die anderen Durchschnitte der Queue-Längen bleiben bei 0,5 Paketen. Die maximalen Queue-Längen der anderen Switches sinken leicht gegenüber dem vorherigen Experiment.

Insgesamt lässt sich sagen, dass die Vergrößerung der Zeitfenstergröße bis zu einem gewissen Punkt einen relevanten Einfluss auf die Queue-Länge der Switches hat. Die Verbesserungen nehmen ab, wenn die Zeitfenstergröße eines Switches die Übertragungsdauer aller vorherigen Pakete mehr als die Übertragungslänge eines Paketes übersteigt, weil selbst durch die Verschiebung der Zeitfenster sich keine Pakete mehr ansammeln können. Es werden in einer Periode dann mehr Pakete gesendet als sich ansammeln können.

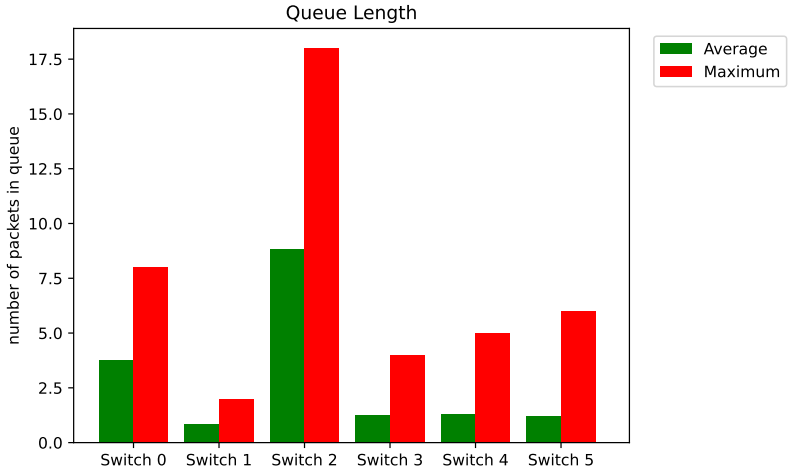
4.2. Uhrensynchronisation

Die Überlegung ist, den Uhrendrift der einzelnen Switches mithilfe des gPTP-Protokolls zu synchronisieren. Dafür sollen zwei Fälle untersucht werden. Anhand der einzelnen Vergleichsmetriken lässt sich so eine Verbesserung messen.

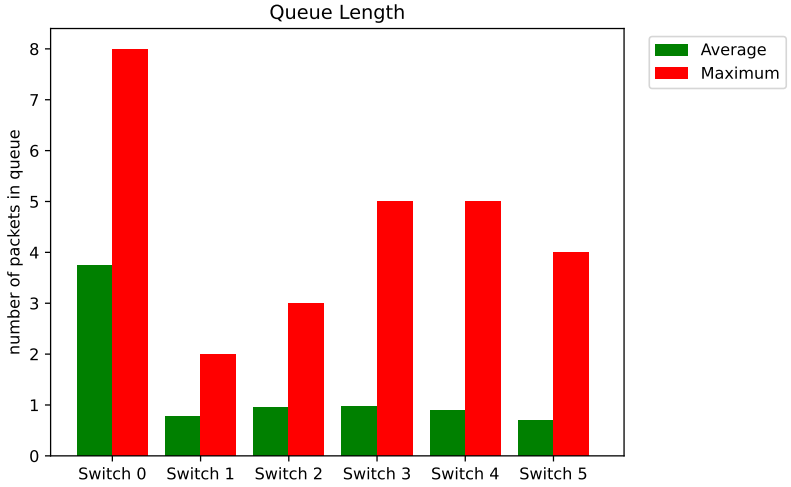
4.2.1. Konfiguration

Es werden zwei Möglichkeiten untersucht, eine TSN-Uhr zur Linientopologie hinzuzufügen:

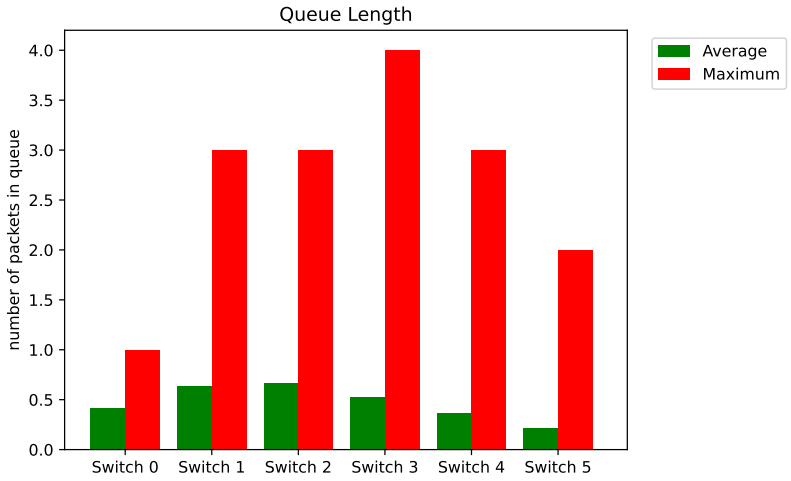
1. Die TSN-Uhr ist lediglich mit Switch 0 verbunden. Das Netzwerk ist in Abbildung 4.5 zu sehen. Die Switches 0 bis 4 sind gPTP-Relay-Instanzen. Switch 5 ist eine gPTP-Endinstanz.



(a) Queue-Länge des Experiments mit 25% größeren Zeitfenstern.



(b) Queue-Länge des Experiments mit 50% größeren Zeitfenstern.



(c) Queue-Länge des Experiments mit 100% größeren Zeitfenstern.

Abbildung 4.4.: Queue-Länge der verschiedenen Zeitfenstergrößen

2. Die TSN-Uhr ist mit allen Switches verbunden. Die einzelnen Switches dienen als gPTP-Endinstanzen. Das Netzwerk ist in Abbildung 4.6 zu sehen.

Die zweite Konfiguration ist der ersten in der Synchronisierung aller Switches überlegen, weil der Netzwerkverkehr im Gegensatz zum ersten Ansatz nicht über die einzelnen Switches fließen muss, sondern direkt von der TSN-Uhr an die Switches gesendet wird. Auf diese Weise entstehen keine Kollisionen mit anderen Paketen, die über die Switches an den Server gesendet werden. Die Synchronisierung der Uhren ist somit effizienter. Mit der zweiten Konfiguration geht allerdings auch eine größere Veränderung des Netzwerks einher. Es werden 5 weitere Ethernet-Verbindungen benötigt.

Die TSN-Uhr und die Switches unterstützen gPTP. Das ermöglicht eine periodische Synchronisation aller Netzwerkteilnehmer. In beiden Konfiguration entsteht eine Baumstruktur im Netzwerk mit der TSN-Uhr als Grandmaster-Instanz.

In der ersten Konfiguration erhalten die Switches zwei Queues. Mithilfe eines Content-Based-Classifiers werden die Pakete der Clients in die erste Queue und die gPTP-Pakete in der zweiten Queue gespeichert. Die erste Queue erhält ein zeitgesteuertes Gate, das wie im Schedule in Abbildung 3.2 schaltet. Die genauen Datenwerte sind in Tabelle A.2 einzusehen. In der zweiten Konfiguration erhalten alle Switches nur eine Queue mit einem Gate, weil die gPTP-Pakete zu keinem Zeitpunkt zwischengespeichert werden müssen. Die Pakete kommen ohne eventuelle Konflikte auf den Switches an. Die Gates verhalten sich nach dem gleichen Schedule.

Die Uhren auf allen Switches erfahren einen konstanten Zeit-Drift, der in Tabelle A.4 zu sehen ist.

Die Applikationen auf den Switches verhalten sich weiterhin wie in den vorherigen Experimenten. Ihre Konfiguration lässt sich aus Tabelle A.1 entnehmen.

Die TSN-Uhr dient als Grandmaster-Instanz, die mit der aktuellen Simulationszeit übereinstimmt. Die Grandmaster-Instanz sendet alle 100ms Synchronisationsnachrichten an alle Switches. Die Switches dienen als Slave-Instanzen und senden alle 500ms Delay-Request-Nachrichten an die Grandmaster-Instanz, damit eine Berechnung der Zeitdifferenz möglich wird. Die Synchronisationsnachrichten und der Delay-Request-Nachrichten werden mit einem Offset von 100µs an die Switches und die Clock gesendet, weil die Nachrichten der Applikationen bereits zum Start der Simulation gesendet werden, werden Konflikte gemieden, können aber ohne eine Veränderung der Topologie nicht gänzlich verhindert werden. Auch das Einrichten von separaten Zeitslots kann dieses Problem nicht vollständig lösen, denn bei der Übertragung eines Paketes auf einer Verbindung kann kein zweites zur selben Zeit gesendet werden. Unweigerlich würde so auch Konflikte entstehen. Der Offset stellt einen einfachen Kompromiss für die erste Konfiguration dar, der die Funktionsweise von gPTP nicht negativ beeinflusst.

4.2.2. Uhrenabweichung

In Abbildung 4.7 ist die Abweichung der Uhren auf den Switches zur Simulationszeit zu sehen. Auf der x-Achse befindet die Simulationszeit in ms. Um die Uhrenabweichung besser zu visualisieren, wurde die x-Achse auf einen Bereich von 1s bis 2s beschränkt. Andernfalls wären die individuellen Synchronisationen aufgrund der vergleichsweise

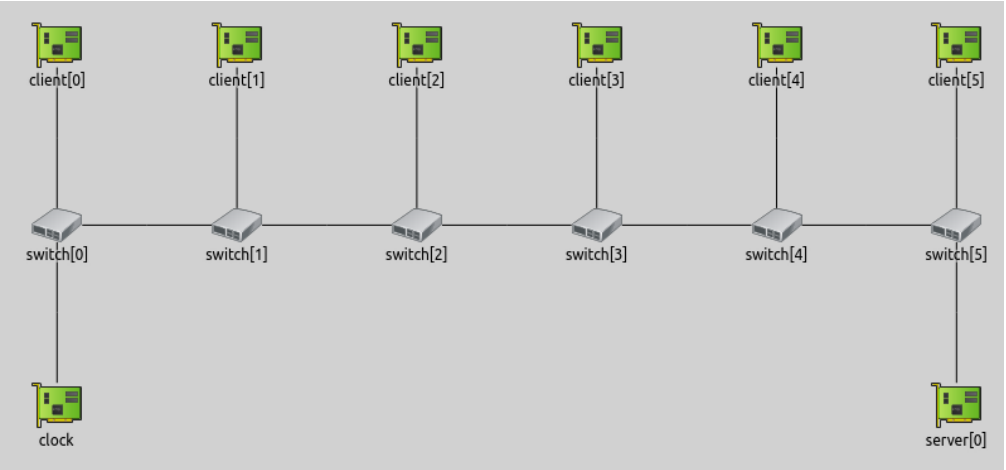


Abbildung 4.5.: Der Linientopologie, bestehend aus 6 Switches, 6 Clients und einem Server, wurde eine Uhr hinzugefügt.

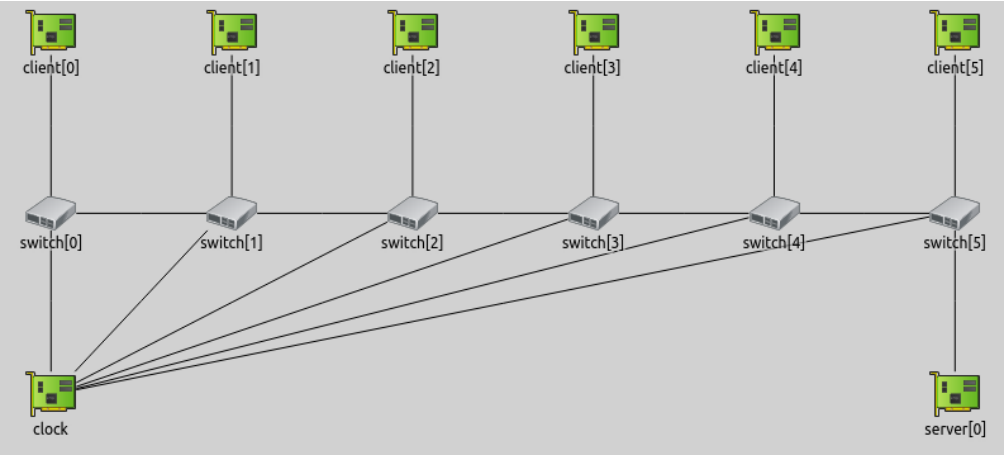


Abbildung 4.6.: Der Linientopologie, bestehend aus 6 Switches, 6 Clients und einem Server, wurde eine Uhr hinzugefügt und mit allen Switches verbunden.

kleinen Synchronisationsintervallen nicht zu erkennen. Auf der y-Achse befindet sich die Differenz der Uhrzeit zur Simulationszeit.

Die Switches 0, 2, und 4 erfahren einen negativen Drift, deshalb sinkt die Differenz permanent und bleibt stets unterhalb der Nulllinie. Die Switches 1, 3, und 5 erfahren einen positiven Drift und deren Differenz steigt somit. Die Uhren der Switches steigen bis zu einem Betrag von 30µs. Die maximale Abweichung zur Simulationszeit ergibt sich aus dem Drift von 300ppm, nach 100ms ergeben sich $300 \cdot 10^{-6} \cdot 100\text{ms} = 30\mu\text{s}$. Danach werden alle Switches wie in der Konfiguration beschrieben mit einer Periode von 100ms bis zum Ende der Simulationszeit synchronisiert. Deren Differenz zur Simulationszeit wechselt dann zu 0µs. Die Master-Instanz weist keine Differenz zur Simulationszeit auf und verläuft deckungsgleich mit der x-Achse.

Die Synchronisation funktioniert bei beiden Varianten optimal. Das wird bei der ersten Konfiguration durch den Offset der gPTP-Pakete realisiert. Die Pakete werden gesendet, wenn gerade keine anderen Pakete von den Clients an den Server gesendet werden. Das reduziert Konflikte und die Synchronisation findet ohne Probleme statt.

OMNeT++ erlaubt es nicht, den Drift der Uhren zu korrigieren. Die Synchronisation muss also manuell in kleinen Intervallen durchgeführt werden, um ein Einhalten der Schedules zu gewährleisten.

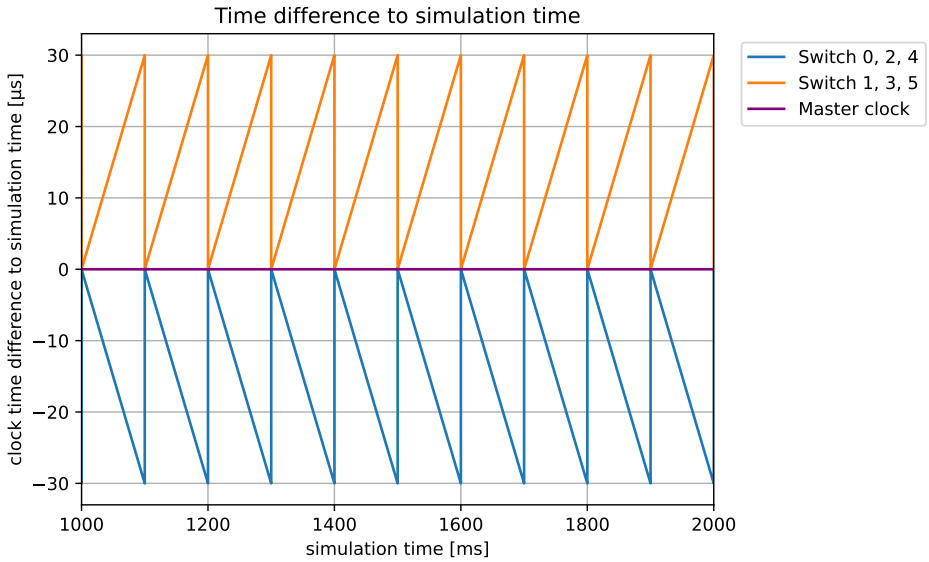


Abbildung 4.7.: Das Diagramm zeigt einen Ausschnitt der Simulationszeit von 1000ms bis 2000ms in welcher die Uhren periodisch synchronisiert werden.

4.2.3. Latenz

In Abbildung 4.8a ist die Latenz im Experiment mit der ersten Konfiguration zu sehen. Auf der x-Achse befindet sich die Simulationszeit in Millisekunden von 0 bis 5000. Auf der y-Achse ist die Latenz in Mikrosekunden zu sehen. Jede Applikation erhält ihre eigene gefärbte Linie.

Die durchschnittliche Latenz der Applikation n ist größer als die Latenz der Applikation $n+1$, weil die Clients weiter vom Server entfernt sind. Aufgrund der Strukturierung des Netzwerks und der Priorisierung des Netzwerkverkehrs, welcher von den Clients kommt, muss sogar zu jedem Zeitpunkt die Latenz eines Paketes der Applikation n größer sein, als die Latenz eines Paketes der Applikation $n+1$, welches in der gleichen Periode gesendet wurde.

Die maximale Latenz erreicht Applikation 0 mit 705µs. Die Latenz der Applikation 1 beträgt im Maximum 680µs. Danach folgt Applikation 2 mit 495µs. Applikation 3 besitzt eine maximale Latenz von 295µs. Darunter befindet sich Applikation 4 mit 250µs und Applikation 5 mit 80µs. Die Latenzen sind bereits deutlich kleiner als die Latenz im Experiment ohne Gegenmaßnahmen und weisen eine obere Schranke auf, bleiben aber dennoch weit über den Werten, die im Optimalfall erzielt wurden.

Die Latenzen der Applikationen schwanken stark um einen konstanten Wert und wiederholen sich mit einer Periode von 100ms. Dies entspricht dem Interval der gPTP-Synchronisationsnachrichten. Zu diesen Zeitpunkten kann der Schedule erfüllt werden, weil die Uhren korrekt laufen. Das führt bei den Latenzen der Applikationen immer zum Erreichen eines lokalen Minimums. Die Schwankungen entstehen, wenn sich viele Pakete zur gleichen Zeit in einer Queue befinden und dann weitergesendet werden. Die Queue-Länge schwankt dann erheblich für die Dauer des Experiments.

In Abbildung 4.8b ist die Latenz für die zweite Konfiguration zu sehen. Die Latenzen sind alle deutlich kleiner als im vorherigen Experiment. Die höchste Latenz erreicht Applikation 0 mit 550µs. Die Latenz der Applikation 1 beträgt im Maximum 510µs. Danach folgt Applikation 2 mit 350µs. Applikation 3 besitzt eine maximale Latenz von 150µs. Darunter befindet sich Applikation 4 mit 105µs und Applikation 5 mit 25µs. Die Latenz der Applikation 5 ist sogar äquivalent zum optimalen Ergebnis.

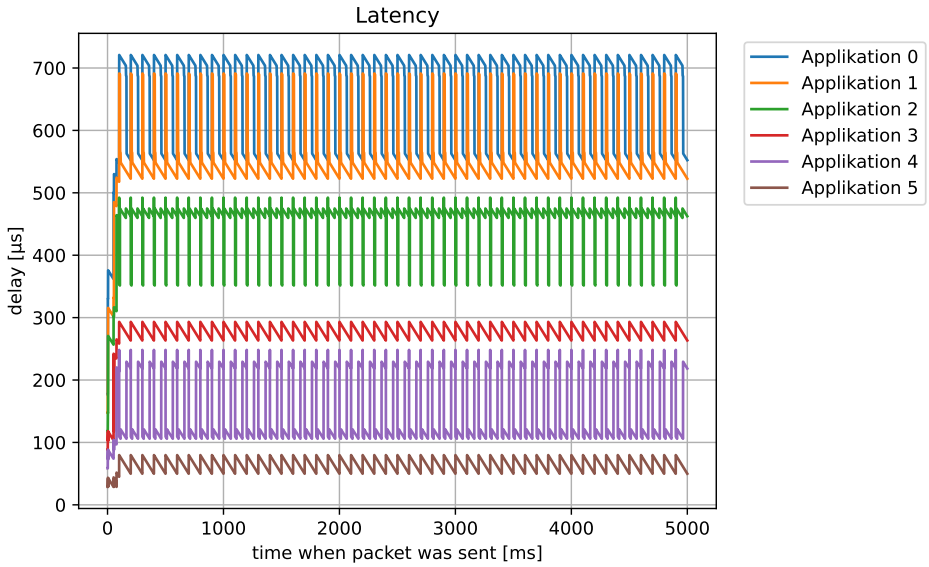
Die periodischen Schwankungen der Latenzen sind deutlich geringer, weil gPTP nicht mit der Übertragung der eigentlichen Nachrichten interferieren kann. Die Schwankungen nehmen nach ein wenig mehr als einer Sekunde ab, weil die ersten Delay-Request-Nachrichten von den Switches gesendet werden und so eine optimale Synchronisation ermöglicht wird.

Die Schwankungen der Latenzen in beiden Experimenten ist aus der Minimalität der Zeitfenster erwartbar. Denn bei jedem beliebigen Uhrendrift führt bereits eine kleine Verschiebung der Zeitfenster zu einer Verletzung der Schedules, weil Pakete nicht mehr übertragen werden können.

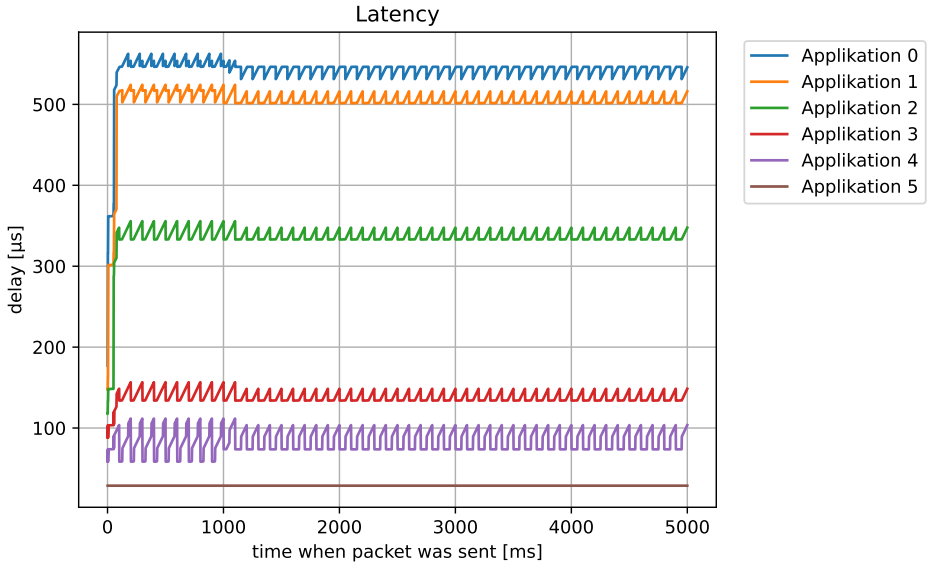
4.2.4. Jitter

In Abbildung 4.9a ist der Jitter für die erste Konfiguration abgebildet. Auf der x-Achse befindet sich die Simulationszeit in Millisekunden von 0 bis 5 und auf der y-Achse der Jitter mit der Einheit μs^2

Der Jitter beginnt bei allen Applikationen bei 0. Der Jitter der Applikationen 0, 1, 2, 3, und 5 fällt nach 100ms ab. Der Jitter der Applikation 4 fällt nach einer Sekunde nicht mehr ab. Nach 4 Sekunden erreicht der Jitter bei allen anderen Applikationen einen relativ konstanten Wert. Das ergibt sich aus dem Abhängigkeitsverhältnis den Applikation 4 aufweist; alle Pakete dieser Applikation müssen nur über die Switches 4



(a) Latenz der ersten Konfiguration



(b) Latenz der zweiten Konfiguration

Abbildung 4.8.: Latenz der zwei Netzwerkkonfigurationen mit gPTP

und 5 an ihr Ziel gelangen. Dies führt potentiell zu einer größeren Variation der Latenz, weil die Übertragung bei einigen Übertragungen ohne Verzögerung stattfinden kann, bei anderen allerdings durch ein einziges Paket im Switch verzögert wird.

Den größten finalen Jitter hat dabei Applikation 0 mit 80µs^2 Darunter liegt Applikation 4 mit 55µs^2. Applikation 1 besitzt einen Jitter von 50µs^2. Applikation 2 liegt bei 40µs^2 und Applikation bei 3 bei 20µs^2. Den niedrigsten Jitter hat Applikation 5 mit 10µs^2.

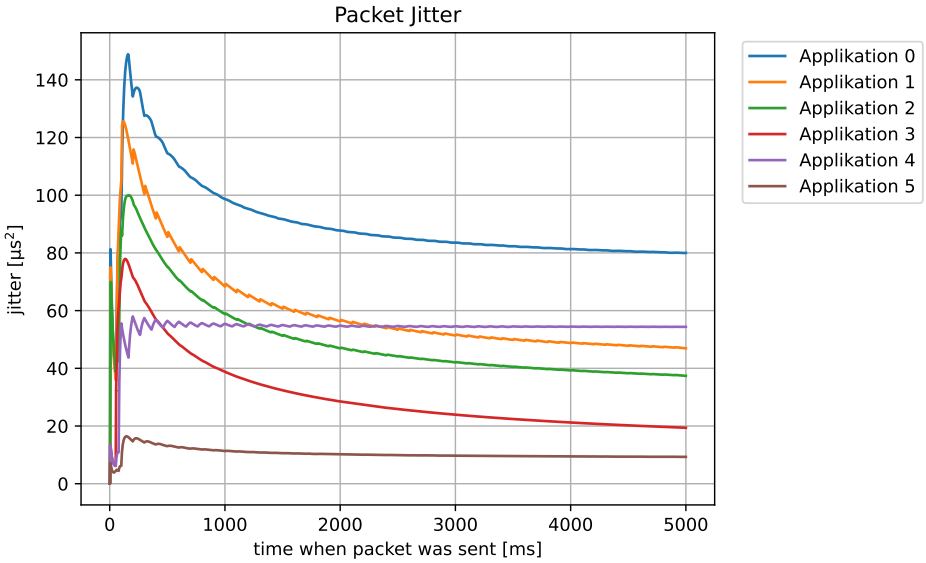
Demgegenüber steht der Jitter der zweiten Konfiguration in Abbildung 4.9b. Der Jitter beginnt wieder bei allen Applikationen bei 0. Bei den Applikation 0, 1 und 2 steigt der Jitter zuerst und fällt nach 250ms rapide ab und sinkt dann für die Dauer des Experiments weiter. Den größten Jitter hat Applikation 1 mit 25µs^2. Der Jitter der Applikationen 0 und 2 ist 20µs^2. Darunter folgt Applikation 4 mit einem Jitter von 15µs^2 und Applikation 3 mit einem Jitter von 10µs^2. Der Jitter der Applikation 5 ist erwartungsgemäß 0, denn die Latenz ist konstant.

4.2.5. Queue-Länge

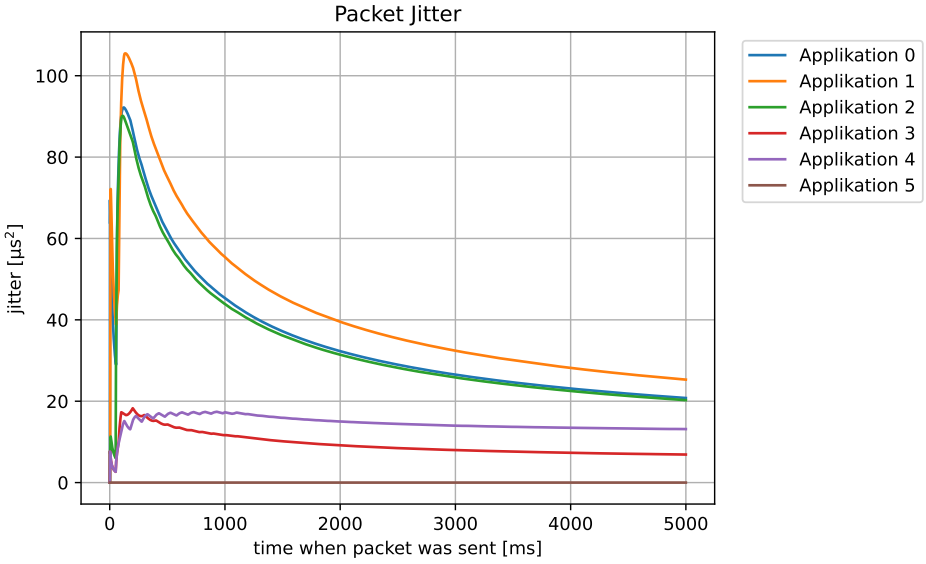
In Abbildung 4.10a ist die Queue-Länge für die erste Konfiguration zu sehen. Auf der x-Achse befinden sich die 6 verschiedenen Switches und auf der y-Achse befinden sich die Anzahl der Pakete in den Queues. Die Balken repräsentieren die maximale und durchschnittliche Queue-Länge.

Die maximale Queue-Länge des Switches 0 beträgt 1 Paket und stimmt mit dem Optimalfall überein. Switch 1 enthält 4 Pakete. Switch 2 liegt bei 3 Paketen. Switch 3, 4 und 5 liegen bei 5 Paketen. Die durchschnittliche Queue-Länge ist bei Switch 0 fast 0. Das bedeutet, dass die Pakete direkt von Switch 0 weitergesendet werden, wie im Optimalfall. Bei Switch 1 ist die durchschnittliche Queue-Länge etwas kleiner als die halbe maximale Queue-Länge. Hier sind von Anfang an keine Pakete in der Queue und sammeln sich dort für die Dauer der Simulation an. Switch 2 hat eine durchschnittliche Queue-Länge die unterhalb von 1 liegt, weil die Queue für den größten Teil der Simulation leer ist. Die durchschnittliche Queue-Länge der Switches 3 und 5 ist fast identisch mit 2,5 Paketen. Bei Switch 4 befinden sich ähnlich zum Switch 2 die meiste Zeit wenig Pakete in den Queues und sammeln sich zwar für die eine Periode an, werden dann allerdings direkt weitergesendet. Die Pakete stauen sich damit nur für die Dauer einer Periode an, während die Queue nach dem Schließen der Zeitfenster wenig gefüllt ist. Die Queues der Switches die näher an dem Server liegen, sind tendenziell voller, weil durch sie mehr Netzwerkverkehr fließen muss.

In Abbildung 4.10b ist die Queue-Länge der zweiten Konfiguration zu sehen. Die maximalen Queue-Längen der Switches 0 und 2 und 4 sind identisch zum vorherigen Experiment. Alle anderen Queue-Längen haben sich deutlich verbessert. Switch 1 liegt bei 2 Paketen, Switch 3 liegt bei 3 und Switch 5 fällt sogar auf das Optimum von einem Paket. Die Synchronisation sorgt dafür, dass die Pakete von Applikation 5 direkt an den Server gesendet werden. Selbst wenn die Synchronisation der Uhr auf Switch 5 nicht häufig genug passiert um die Übertragung aller Pakete in einer Periode zu ermöglichen, werden zumindest immer die Pakete der Applikation 5 rechtzeitig an den Server gesendet. Die durchschnittliche Queue-Länge ist bei den Switches 1, 2, und 5

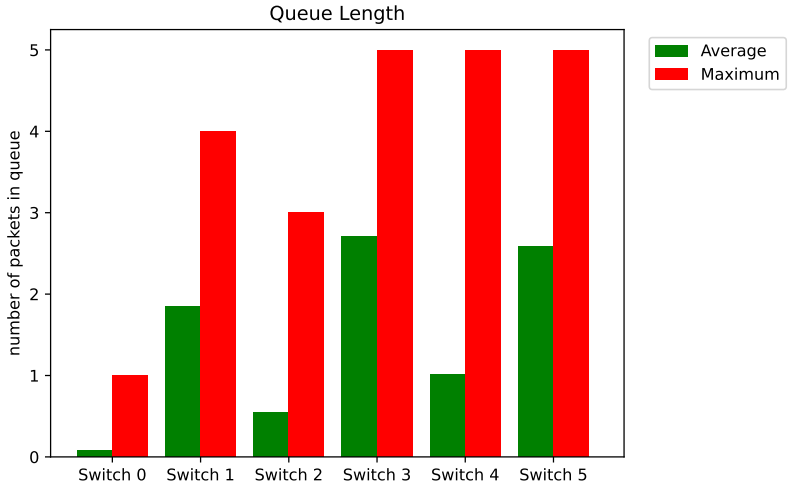


(a) Jitter der ersten Konfiguration

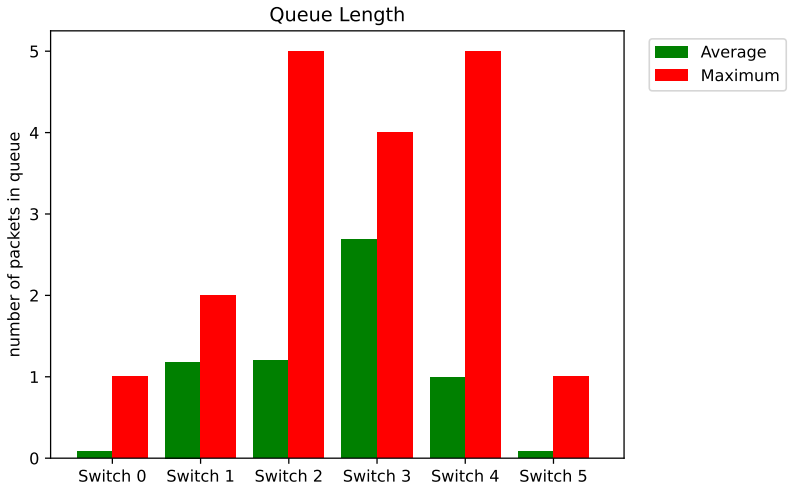


(b) Jitter der zweiten Konfiguration

Abbildung 4.9.: Jitter der zwei Netzwerkkonfigurationen mit gPTP



(a) Queue-Länge der ersten Konfiguration



(b) Queue-Länge der zweiten Konfiguration

Abbildung 4.10.: Queue-Länge der zwei Netzwerkkonfigurationen mit gPTP

deutlich geringer als im vorherigen Experiment. Bei den anderen Switches lässt sich keine Verbesserung feststellen.

Insgesamt verbessert das Hinzufügen von Ethernet-Verbindungen, welche für gPTP reserviert werden, die maximale und durchschnittliche Queue-Länge.

5. Resümee

Für TSN-Schedules sind präzise Uhren notwendig. In der Realität besitzen Uhren jedoch einen Uhrendrift, der zu einem Verschieben der Zeitfenster führt. Ohne Synchronisation aller Netzwerkteilnehmer kann kein optimaler zeitsensitiver Netzwerkverkehr ermöglicht werden. Mit Hilfe von gPTP lassen sich die Uhren auf den Switches synchronisieren.

Die Etablierung der Vergleichsmetriken Latenz, Jitter und Queue-Länge in allen Experimenten erlaubt ein Vergleich zwischen den verschiedenen Lösungsansätzen.

Alle Experimente wurden mit OMNeT++ durchgeführt. Dieses ermöglicht eine deterministische Simulation von zeitsensitiven Netzwerken. Die Experimente wurden so aufgebaut, um einen Zusammenhang zwischen den verschiedenen Metriken und den Gegenmaßnahmen herzustellen. Insbesondere bei der Untersuchung der Zeitfenster wurden schrittweise die Startparameter in Form ihrer Größe angepasst. Das Beispiel der Linientopologie dient als eine häufige Netzwerktopologie im echten Leben. Die zwei gPTP-Konfigurationen dienen dazu, eine abschließende Aussage über die Strukturierung des Netzwerks und der Effizienz der Zeitsynchronisation zu ermöglichen.

Zuerst wurde der Schedule auf seine Richtigkeit überprüft. Dieser produziert unter optimalen Bedingungen minimale Ergebnisse, die mathematisch nicht kleiner sein könnten.

Das Experiment mit Offset zeigt, dass nur konstante Auswirkungen für die Vergleichsmetriken entstehen. Jeder Offset stellt sich automatisch durch einen Drift ein und realistische Uhren besitzen nie nur einen fixen Offset. Ein Offset ist daher für die Durchführung und Auswertung von realitätsnahen Experimenten weniger relevant als ein Uhrendrift.

Das Experiment mit Drift dient als Negativbeispiel und führt in allen Bereichen zu schlechten Ergebnissen. Das dient als Motivation Gegenmaßnahmen zu untersuchen.

Die Experimenten zur Lösung der Probleme zeigen, dass die Vergrößerung der Zeitfenster und die Uhrensynchronisation unterschiedlich brauchbare Maßnahmen zur Behebung von Ungenauigkeiten in zeitsensitiven Netzwerken sind.

Bei einem konstanten Offset der Uhren ist die Vergrößerung der Zeitfenster in Bezug auf Jitter und Queue-Länge ausreichend. Auch aus diesem Grund ist die Betrachtung des Uhren-Offset nicht zielführend.

Eine Vergrößerung der Zeitfenster um 25% begrenzt die Latenz der Applikationen 4 und 5, während eine Vergrößerung um 50% die Latenzen der Applikationen von 1 bis 5 begrenzt. In beiden Fällen ist mindestens die Latenz einer Applikation unbegrenzt. Erst ab eine Vergrößerung der Zeitfenster um 100% stellt sich eine Begrenzung aller Latenzen für die Dauer des Experiments ein. Allerdings ist die Latenz instabil und verläuft periodisch für die Dauer des Experiments.

Die schwankende Latenz schlägt sich in einem hohen Jitter für alle Applikationen nieder, die ungeeignet für zeitsensitiven Netzwerkverkehr ist.

Die Queue-Länge ist erst für die Vergrößerung der Zeitfenster um 100% ausreichend klein. Bei kleineren Zeitfenstern stellt sich für einige Switches eine Queue-Länge ein, die zeitsensitiven Netzwerkverkehr unmöglich macht.

Mit Hilfe der Vergrößerung der einzelnen Zeitfenster ist es möglich, die Vergleichsmetriken Latenz, Jitter und Queue-Länge positiv zu beeinflussen.

Die Gegenüberstellung der zwei unterschiedlichen gPTP-Konfigurationen, zeigt, dass mit beiden eine reibungslose Synchronisation aller Netzwerkteilnehmer möglich ist. Bei der zweiten Konfiguration ist es wichtig, dass die gPTP-Pakete in einem Offset zum regulären Netzwerkverkehr gesendet werden. Eine separate Queue und ein permanent geöffnetes Gate für gPTP-Pakete sind ohne weitere Planung eine gute Lösung für die Bereitstellung von gPTP-Paketen ohne exklusive Ethernet-Verbindungen.

Die Latenz ist in beiden Experimenten mit gPTP begrenzt. Diese ist zwar nicht optimal, aber deutlich niedriger als im Experiment ohne Gegenmaßnahmen und erlaubt das Ermitteln einer oberen Schranke.

Der Jitter fällt für die Dauer beider Experimente und nähert sich einem Wert an.

Die Queue-Länge liegt über dem Optimum, verbessert sich aber dennoch.

Die zweite Konfiguration ist in allen Bereichen besser als die erste. Dies ist nur auf die Kollisionsgefahr der gPTP-Pakete mit dem regulären Netzwerkverkehr zurückzuführen.

Im Vergleich zur Wahl größere Zeitfenster erzielt die Uhrensynchronisation mit gPTP keine wirkliche Verbesserung der Latenz. Allerdings können massiv Schwankungen in der Latenz reduziert werden. Das wirkt sich auf eine massive Verbesserung des Jitters aus. Selbst beim Experiment mit 100% größeren Zeitfenstern ist der maximale Jitter noch 250% größer als im Experiment mit gPTP. Die Queue-Länge ist bei beiden Experimenten vergleichbar.

Die Ergebnisse verdeutlichen, dass nur eine Verwendung von größeren Zeitfenstern nicht ausreicht. Eine Form der Uhrensynchronisation ist folglich unerlässlich.

Bei Synchronisationsintervallen, die zu groß sind, um den Drift zu jedem Zeitpunkt auszugleichen, ist ein Kompromiss die Wahl größerer Zeitfenster, die die Übertragung aller Pakete auch bei minimalen Abweichungen erlauben, denn bei knapp gewählten Zeitfenstern führen bereits geringe Abweichung zur Verspätung des Netzwerkverkehrs. In jedem Fall ist die Wahl größerer Zeitfenster, die die Abweichung der Uhren berücksichtigen, eine plausible Möglichkeit, die Vergleichsmetriken zu verbessern. Eine Kombination aus gering vergrößerten Zeitfenstern und einer der beiden Netzwerkkonfigurationen mit gPTP erscheint am besten geeignet. Die Wahl der Netzwerkkonfiguration im Kontext von gPTP ist abhängig von der Umsetzbarkeit und der Striktheit der Anforderungen.

Literatur

[1] Zoltan Bojthe u. a. *INET Framework*. https://inet.omnetpp.org/. (Besucht am 22. 03. 2023).

[2] Janos Farkas, Lucia Lo Bello und Craig Gunther. „Time-Sensitive Networking Standards“. In: *IEEE Communications Standards Magazine* 2.2 (2018), S. 20–21. DOI: 10.1109/MCOMSTD.2018.8412457.

[3] Rhys Haden. *Network Time Protocol (NTP)*. https://www.rhyshaden.com/ntp.htm. (Besucht am 01. 06. 2023).

[4] Junhui Jiang u. a. „A Time-sensitive Networking (TSN) Simulation Model Based on OMNET++“. In: *2018 IEEE International Conference on Mechatronics and Automation (ICMA)*. 2018, S. 643–648. DOI: 10.1109/ICMA.2018.8484302.

[5] Angelos Mimidis Kentis, Michael Stubert Berger und Jose Soler. „Effects of port congestion in the gate control list scheduling of time sensitive networks“. In: *2017 8th International Conference on the Network of the Future (NOF)*. 2017, S. 138–140. DOI: 10.1109/NOF.2017.8251236.

[6] Lisa Maile, Kai-Steffen Hielscher und Reinhard German. „Network Calculus Results for TSN: An Introduction“. In: *2020 Information Communication Technologies Conference (ICTC)*. 2020, S. 131–140. DOI: 10.1109/ICTC49638.2020.9123308.

[7] Jim Martin u. a. *Network Time Protocol Version 4: Protocol and Algorithms Specification*. RFC 5905. Juni 2010. DOI: 10.17487/RFC5905.

[8] Haavard Nord u. a. *Qt - Cross Platform Software*. https://www.qt.io/. (Besucht am 23. 03. 2023).

[9] Glenn Parsons u. a. „IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks“. In: *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)* (2018), S. 1–1993. DOI: 10.1109/IEEESTD.2018.8403927.

[10] Glenn Parsons u. a. „IEEE Standard for Local and Metropolitan Area Networks–Timing and Synchronization for Time-Sensitive Applications“. In: *IEEE Std 802.1AS-2020 (Revision of IEEE Std 802.1AS-2011)* (2020), S. 1–421. DOI: 10.1109/IEEESTD.2020.9121845.

[11] Henning Puttnies u. a. „A Simulation Model of IEEE 802.1AS gPTP for Clock Synchronization in OMNeT++“. In: *Proceedings of the 5th International OMNeT++ Community Summit*. Hrsg. von Anna Förster u. a. Bd. 56. EPiC Series in Computing. EasyChair, 2018, S. 63–72. DOI: 10.29007/wfl9.

[12] Andras Varga, Rudolf Hornig, Török Attila u. a. *OMNeT++ Discrete Event Simulator*. https://omnetpp.org. (Besucht am 22. 03. 2023).

A. Anhang

Applikationen Konfiguration		
Client-ID	Sende-Offset (µs)	Zielport
0	0	1000
1	14.37	1001
2	28.74	1002
3	43.11	1003
4	57.48	1004
5	71.85	1005

Tabelle A.1.: Auf jedem Client existiert eine Applikation, die UDP Netzwerkverkehr in Intervallen von 200 µs an den Server sendet. Die Offsets maximieren das Konfliktpotential.

Gate Konfiguration			
Switch-ID	Zeit offen (µs)	Zeit geschlossen (µs)	Offset (µs)
0	14.37	185.63	14.37
1	30.74	169.26	28.74
2	45.11	154.89	43.11
3	60.48	139.52	57.48
4	75.85	124.15	71.85
5	91.22	108.78	86.22

Tabelle A.2.: Alle 6 Switches besitzen ein zeitgesteuertes Gate. Die Gates haben größere Zeitfenster mit dem Aufsteigen der Switch-ID, um den höheren Netzwerkverkehr auszugleichen.

Offsets	
Switch-ID	Offset (µs)
0	-1
1	1
2	-1
3	1
4	-1
5	1

Tabelle A.3.: Auf jedem Switch befindet sich eine Uhr mit konstantem Offset. Die Uhren erfahren keinen Drift.

Uhrendrifts	
Switch-ID	Uhrendrift (ppm)
0	-300
1	300
2	-300
3	300
4	-300
5	300

Tabelle A.4.: Auf jedem Switch befindet sich eine Uhr mit konstantem Uhrendrift. Die Vorzeichen der Uhrendrifts alternieren zwischen den Switches.

Gate Konfiguration			
Switch-ID	Zeit offen (µs)	Zeit geschlossen (µs)	Offset (µs)
0	17.96	122.04	14.37
1	38.43	161.57	28.74
2	56.39	143.61	43.11
3	75.60	124.40	57.48
4	94.81	105.19	71.85
5	114.02	85.98	86.22

Tabelle A.5.: Die Zeitfenster wurden um 25% vergrößert.

Gate Konfiguration			
Switch-ID	Zeit offen (µs)	Zeit geschlossen (µs)	Offset (µs)
0	21.56	178.44	14.37
1	46.11	153.89	28.74
2	67.66	122.34	43.11
3	90.72	109.28	57.48
4	113.77	80.23	71.85
5	138.83	161.18	86.22

Tabelle A.6.: Die Zeitfenster wurden um 50% vergrößert.

Gate Konfiguration			
Switch-ID	Zeit offen (µs)	Zeit geschlossen (µs)	Offset (µs)
0	29.74	170.26	14.37
1	61.48	138.52	28.74
2	90.22	109.78	43.11
3	120.96	79.04	57.48
4	151.70	48.30	71.85
5	182.44	17.56	86.22

Tabelle A.7.: Die Zeitfenster wurden um 100% vergrößert.

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig angefertigt und ohne fremde Hilfe verfasst habe. Dazu habe ich keine außer den von mir angegebenen Hilfsmitteln und Quellen verwendet und die den benutzten Werken inhaltlich und wörtlich entnommenen Stellen habe ich als solche kenntlich gemacht.

Rostock, den 06.06.2023

_____

Behrens, Nico