Zürcher Hochschule
für Angewandte Wissenschaften

School of Engineering

InES Institute of Embedded Systems

Projektarbeit Informatik

PA14 wlan 1

Performance-Evaluation Ethernet für Echtzeit-Datenerfassung

Autoren	Mauro Guadagnini (guadamau@students.zhaw.ch) Prosper Leibundgut (leibupro@students.zhaw.ch)
Hauptbetreuung	Hans Weibel (wlan@zhaw.ch)

Datum	19.12.2014

Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

Erklärung betreffend das selbständige Verfassen einer Projektarbeit an der School of Engineering

Mit der Abgabe dieser Projektarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Projektarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

Ort, Datum: Unterschriften:

…… ……

……..

……

Das Original dieses Formulars ist bei der ZHAW-Version aller abgegebenen Projektarbeiten zu Beginn der Dokumentation nach dem Titelblatt mit Original-Unterschriften und -Datum (keine Kopie) einzufügen.

Zusammenfassung

In Deutsch

Abstract

In Englisch

Vorwort

Da wir eine Begeisterung für Netzwerktechnik haben und offen für Neues sind, fiel uns dieses Thema ins Auge.

Stellt den persönlichen Bezug zur Arbeit dar und spricht Dank aus.

Inhaltsverzeichnis

Teil I.

Einführung und Grundlagen

1. Einleitung

1.1. Ausgangslage

1.1.1. Stand der Technik

Das Unterbrechen der Übertragung von Frames durch Express-Frames (IET) sowie das dadurch zu verwendende Mframe-Format sind noch nicht in Hardware implementiert. Es ist ein Entwurf in Entwicklung [5], der voraussichtlich Ende 2015 zum Standard werden soll und seit Mai 2014 keine weiteren Features mehr erhält, technische Änderungen jedoch noch bis März 2015 eingeführt werden können [6].

Aufgrund dieser Tatsache findet man im Internet kaum etwas zu IET sowie dem Mframe-Format, ausser einigen Unterlagen vom IEEE-Verband, der den Standard entwickelt.

Bezüglich HSR (High Availability Seamless Redundancy) gibt es einiges mehr zu finden. Das HSR-Protokoll ist seit Februar 2010 unter dem Titlel «IEC 62439-3 Cl. 5» ein Standard und wurde schon bei einigen Firmen implementiert [8]. Da IET noch nicht in Hardware implementiert wurde, existieren auch keine Berichte zur Implementation von IET in einem HSR-Netzwerk.

1.1.2. Bestehende Arbeiten

Die bestehende Vertiefungsarbeit [1] behandelt das Simulieren von Frames in einem HSR-Netzwerk, jedoch ohne IET-Implementation, die dazugehörige Frame-Priorisierung und dem Mframe-Format. Aufgrund dem derzeitigen Stand der Technik wurden keine Arbeiten gefunden, die sich mit unserer Thematik (HSR-Netzwerk mit IET-Implementation) beschäftigen.

1.2. Zielsetzung / Aufgabenstellung / Anforderungen

Durch das Institute of Embedded Systems der ZHAW wurde den Autoren am 24. September 2014 eine Aufgabenstellung[13] (siehe Kapitel 11 auf Seite 56) zugestellt, welche die nachfolgenden Hauptanforderungen umfasst:

1.2.1. Modell für HSR-Knoten erweitern

Das betrachtete Netzwerk ist ein HSR-Ring. Die bestehende Simulationsumgebung [1] soll so erweitert bzw. angepasst werden, dass folgende Funktionen/Mechanismen simuliert werden können:

Anforderungs-Nr.	Beschreibung
1.1	Der Knoten soll zwei Prioritäten unterstützen, d.h. zwei Warteschlangen pro Interface bewirtschaften.
1.2	Der Knoten soll Interspersing Express Traffic (IET) unterstützen, d.h. Express Frames können die aktuell ablaufende Übertragung eines Frames unterbrechen.
1.3	Der in den Ring einfliessende Traffic kann limitiert werden.
1.4	Die Vortrittsregeln bezüglich der im Ring zirkulierenden Frames und den Frames, die in den Ring einfliessen, können variiert werden (z.B. «zirkulierende Frames haben immer Vortritt» oder «minimaler Zufluss wird garantiert»).
1.5	Der Knoten implementiert ein Zeitschlitzverfahren, welches dem zeitkritischen Traffic und dem Bulk Traffic je eine Phase zuordnet.

Tabelle 1.1.: Anforderungen an HSR-Knoten [13]

1.2.2. Lastmodell beschreiben und implementieren

Das durch die Anwendung generierte Verkehrsaufkommen ist zu studieren und zu beschreiben. Lastgeneratoren sollen implementiert werden, die das Verkehrsaufkommen für die Simulation generieren durch die Überlagerung von Strömen mit folgender Charakteristik:

Anforderungs-Nr.	Beschreibung
2.1	Lastgenerator mit konstanter Framerate.
2.2	Lastgenerator mit zufälliger zeitlicher Verteilung der Frames.
2.3	Lastgenerator, der spontane Einzelmeldungen erzeugt.

Tabelle 1.2.: Anforderungen an Lastmodell [13]

1.2.3. Simulationen durchführen und Resultate interpretieren

Anforderungs-Nr.	Beschreibung
3.1	Das Zeitverhalten der verschiedenen Weiterleitungsvarianten soll durch entsprechende Simulationsläufe ermittelt werden. Die Resultate sind zu vergleichen und zu interpretieren.

Tabelle 1.3.: Allgemeine Anforderungen [13]

1.2.4. Erwartetes Resultat

Das Resultat der Arbeit soll verschiedene Verhaltensweisen von Frames in einem HSR-Ring mit IET-Implementation bei unterschiedlichen Bedingungen aufzeigen. Durch eine Interpretation der Verhaltensweisen ist dann die bestmögliche Konfiguration des HSR-Rings zu ermitteln, mit welcher zeitkritische Frames am schnellsten übermittelt werden.

1.2.5. Vorausgesetztes Wissen

In den theoretischen Grundlagen (siehe Kapitel 2 auf der nächsten Seite) werden unter anderem OMNeT++, das HSR-Protokoll und der Aufbau eines HSR-Netzwerks inklusive dessen Gerätetypen behandelt.

Zum Verständnis dieser Projektarbeit ist ein Vorwissen über die allgemeine Netzwerkkommunikation nötig. Dieses Vorwissen umfasst folgende Bereiche:

- Allgemeine Netzwerk- und Hardware-Begriffe wie z.B. MAC-Adresse, Ethernet-Port oder Ethernet-Frame
- Funktionsweise eines Netzwerks inklusive der Übertragung eines Ethernet-Frames und dem Aufbau dessen Headers

2. Theoretische Grundlagen

2.1. OMNeT++

OMNeT++ ist ein C++-Framework, welches es erleichtert, Netzwerke und all deren Kompnenten mit einem sehr hohen Detaillierungsgrad zu modellieren und den Netzwerk-Datenverkehr zu simulieren. Die Simulationen können grafisch dargestellt werden. Zur Auswertung der Simulationen steht eine grosse Auswahl an verschiedenen Diagrammtypen zur Verfügung. In dieser Projektarbeit wird die Version 4.5 verwendet.

Die Abhandlung der Simulationen in OMNeT++ erfolgt sequentiell, trotzdem lassen sich gleichzeitige Vorkommnisse simulieren, da rein die Simulationszeit in Betracht gezogen wird und somit in der Simulation zum Beispiel zwei Frames zum Zeitpunkt t=2.0s versendet werden können (im Hintergrund werden diese immer noch nacheinander abgehandelt). Somit hat diese Eigenschaft keinen Einfluss auf die Resultate der Simulation. Der Nachteil ist, dass man keine Schleifen in die Simulation implementieren kann (die Simulation hängt sich auf, wenn sie sich in einer While-Schleife befindet). Stattdessen muss man mit der Simulationszeit und Events arbeiten, was aber nach kurzer Zeit kein grosses Hindernis mehr ist. Es besteht die Möglichkeit, die Simulation in mehreren Threads zum Laufen zu bringen, jedoch erfordert dies einen sehr hohen Aufwand und bringt eine hohe Komplexität mit sich, weshalb nur dazu geraten wird, wenn es absolut nicht anders geht [12].

2.2. High-availability Seamless Redundancy (HSR)

HSR ist ein Ethernet-Redundanz-Protokoll, welches im Fehlerfall keine Ausfallzeit hat und es erlaubt, Geräte zusammen zu schliessen, um ein kosten-effektives Netzwerk betreiben zu können. Es ermöglicht komplexe Topologien wie Ringe und Ringe von Ringen und ist einfach in Hardware zu implementieren. Besonders für Anwendungen, bei denen Unterbrüche nicht tolerierbar sind, ist HSR ein attraktives Protokoll [8].

Die Bedingung für ein Gerät in einem HSR-Netzwerk ist, dass es mindestens zwei Ethernet Ports haben muss. Ein nicht HSR-fähiges Gerät wird mittels eines Redundancy Box (RedBox) angeschlossen, welche die HSR-Funktionen stellvertretend erbringt. Wenn ein Frame versendet werden muss, wird eine Kopie davon erstellt und auf den Ring gleichzeitig in beide Richtungen übertragen. In einem fehlerfreien Netzwerk kommen somit immer zwei oder mehr Frames beim Empfänger an. Es ist die Aufgabe des Empfängers, Duplikate nicht auf höheren Schichten weiter zu geben. Unicast-Frames und dessen Duplikate werden vom Empfänger vom Ring entfernt. Multicast- und Broadcast-Frames werden vom Sender vom Ring entfernt. Der Sinn davon ist, dass bei einem Ausfall eines Gerätes mindestens ein Frame trotzdem an seinen Empfänger

gelangt. Der Frameverlust wird demnach bei einem Fehlerfall verhindert [7]. Höhere Schichten bekommen von dem Ganzen nichts mit [8].

Die Duplikat-Erkennung findet anhand der Source-MAC-Adresse und der Sequenznummer statt. Jedes Gerät führt eine Liste mit den Sequenznummern für jede MAC-Adresse und kann somit einfach bereits erhaltene Frames feststellen. Diese Einträge werden solange behalten wie sich das Frame im HSR-Netzwerk befindet.

2.2.1. Gerätetypen

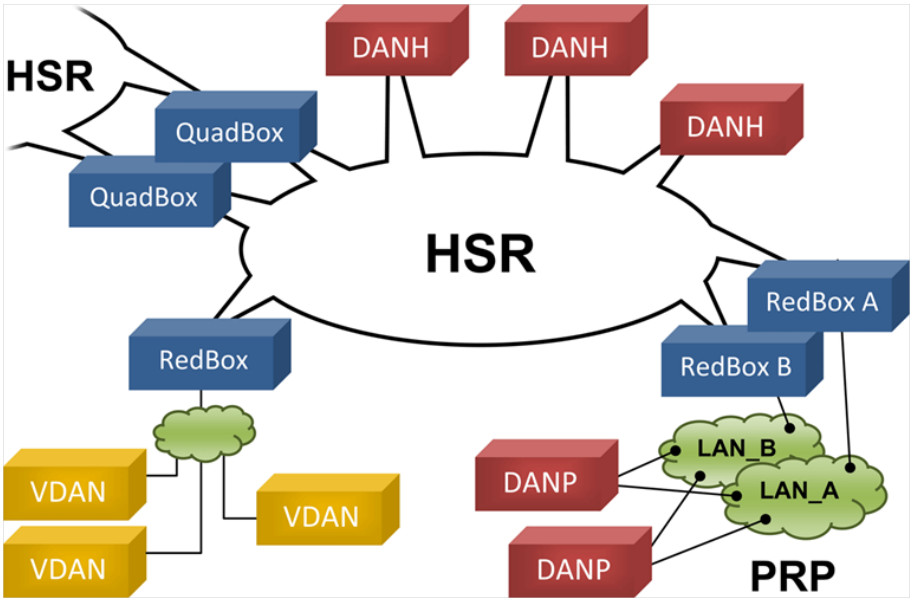

Abbildung 2.1.: HSR-Ring mit allen möglichen Gerätetypen[7]

Name	Beschreibung
DANH	Double Attached Node implementing HSR: Direkt in den Ring eingefügter Knoten mit Ethernet Ports. Die beiden Ports teilen sich dabei die MAC- und IP-Adresse [9].
RedBox	Redundancy Box: Dient dazu nicht HSR-fähige Geräte an einem Netzwerk anzuschliessen.
QuadBox	Dienen, um HSR-Ringe miteinander zu koppeln. Eine Quadbox hat 4 Ethernet Ports. Um zwei HSR-Ringe miteinander zu koppeln benötigt es zwei Quadboxen.
VDAN	Virtual Doubly Attached Node: (Nicht HSR-fähige) Knoten, die mittels RedBox am HSR-Netzwerk angeschlossen sind.
DANP	Double Attached Node implementing PRP (siehe Glossar auf Seite 51): Endknoten in einem PRP-Netzwerk, welches über RedBoxes an einem HSR-Netzwerk angeschlossen ist. Ein DANP muss für die Kommunikation mit einem DANH das HSR-Protokoll nicht kennen.

Tabelle 2.1.: Gerätetypen in einem HSR-Netzwerk [7]

2.3. Interspersing Express Traffic (IET)

Interspersing Express Traffic ist ein Entwurf des Standardisierungsgremiums IEEE unter der Bezeichnung «IEEE 802.3br», welcher voraussichtlich Ende 2015 zum Standard werden soll [6]. Die Recherche für den Entwurf wurde von einer Gruppe beim IEEE erarbeitet, die unter dem Namen «IEEE 802.3 Distinguished Minimum Latency Traffic in a Converged Traffic Environment (DMLT) Study Group» daran gearbeitet hat. Für das erarbeiten des Entwurfs ging die Gruppe dann zur «IEEE P802.3br Interspersing Express Traffic Task Force» über [2].

Ziel ist es auf OSI Layer 2 «Data Link» den Standard IEEE 802.3 «Ethernet» zu erweitern, um IET zu ermöglichen. Mittels IET wird es möglich sein dass Geräte, die IET unterstützen, zwischen sogenannten Express und Normal Frames unterscheiden, den Sendevorgang der normalen Frames unterbrechen und somit Express Frames (auch IET Frames genannt) schneller senden können [4].

Der Grund dafür ist, dass neue Märkte wie zum Beispiel die industrielle Automatisierung und Transport (Flugzeuge, Züge und grosse Fahrzeuge) Ethernet adaptiert haben und die Nachfrage nach einer kleinen Latenz aufgrund von ihrer Hochverfügbarkeit steigern [4]. Dringende Meldungen können dann dem üblichen Traffic vorgezogen und schneller erkannt werden.

Für den Standard ist unter anderem vorgesehen, dass das Ethernet-Frame-Format beibehalten wird, keine Änderungen auf OSI Layer 1 «Physical» gemacht werden und die Express Frames von Geräten, die nicht IET-fähig sind, verworfen werden [5]. Die normalen Frames werden, wenn sie durch Express Frames unterbrochen werden, aufgeteilt in Mframes («MAC Merge Frame», siehe Kapitel 2.4 auf der nächsten Seite), von denen man auf OSI Layer 1 nichts mitbekommt [5]. So wird das bereits Gesendete nicht verworfen, sondern wird mit dem später ankommenden Rest wieder zusammengesetzt. Express Frames werden nicht fragmentiert.

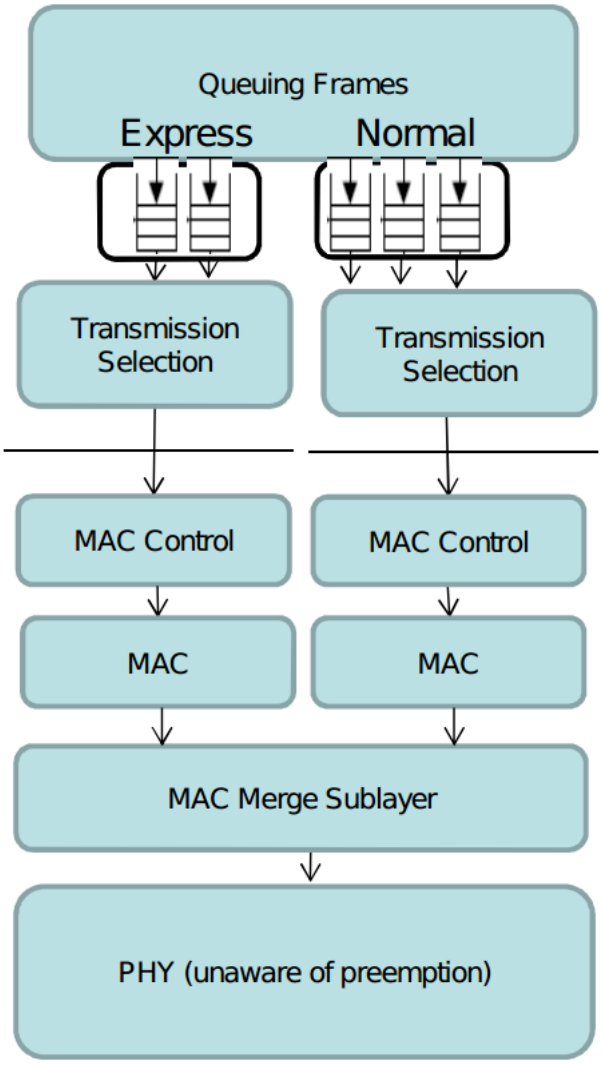

Abbildung 2.2.: MAC Merge Layer [5]

2.4. Mframe

Ein Mframe (steht für «MAC Merge Frame») ist eine Einheit, welche ganze Frames und Fragmente von unterbrechbaren Frames beinhalten kann und wie ein normales Frame auf dem OSI Layer 1 aussieht [5]. Die Struktur eines Mframes ist dem eines Ethernet Frames sehr ähnlich.

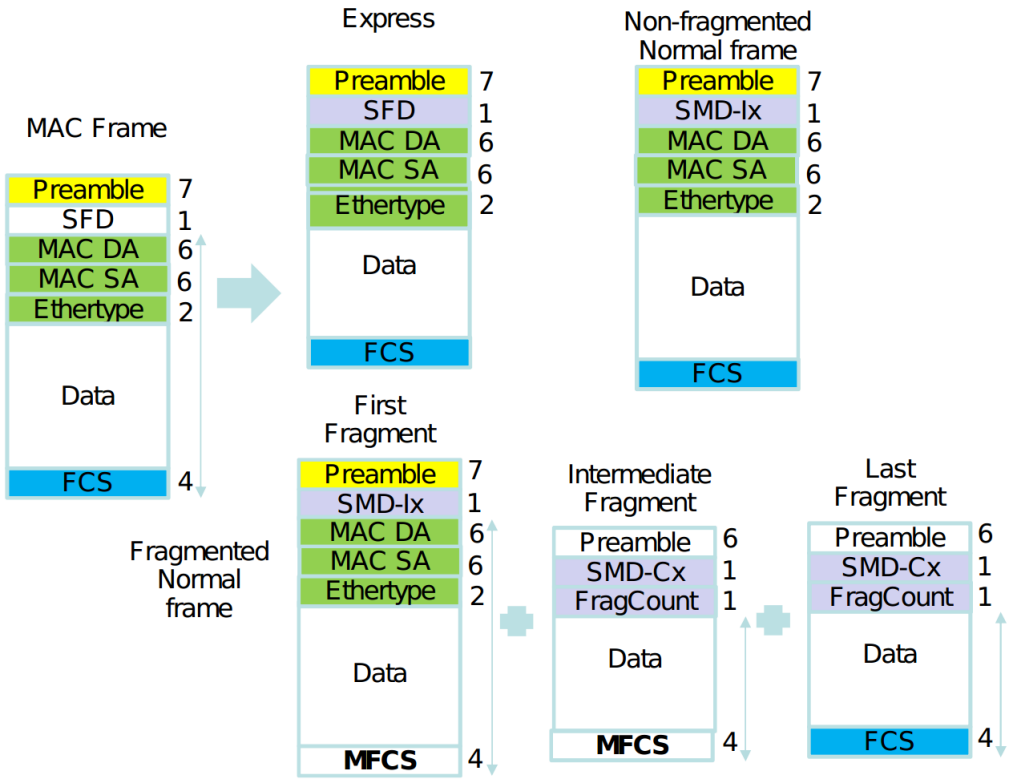

Abbildung 2.3.: Mframe Format [5]

Das SMD-Feld nach der Präambel (anstelle des SFD (Start Frame Delimiter) Feldes) wird in einem Mframe verwendet, um die Fragmente zu kennzeichnen. So hat das erste Fragment einen SMD-Ix-Wert, um zu signalisieren, dass es sich um das erste Fragment eines Frames handelt. Die darauf folgenden Fragmente haben einen SMD-Cx-Wert und ein FragCount-Feld. Im SMD-Ix- und SMD-Cx-Feld wird jeweils eine Framenummer vergeben, wobei im FragCount-Feld die Nummer des Fragments steht. Dabei wechselt jedes dieser Felder jeweils zwischen 4 verschiedenen Werten (siehe Tabelle 2.2 auf der nächsten Seite). Für jedes neue Fragment wird das FragCount-Feld mittels Modulo-4-Zähler hochgezählt [3]. Gibt es dann ein Fragment mit dem FragCount Wert 3, so hat das nächjste Frame wieder den FragCount-Wert 0. Mittels FragCount-Feld wird davor geschützt, dass falsche Frames zusammengesetzt werden wenn bis zu 3 Fragmente verloren gegangen sind. Die FragCount-Werte haben jeweils eine Hamming-Distanz von 4 zueinander, um den Wert bei Bitfehlern in der Übertragung trotzdem erkennen zu können [5].

Mframe Typ	Frame #	SMD
SFD (express)	NA	0xD5
SMD-Ix	0	0xE6
	1	0x4C
	2	0x7F
	3	0xB3
SMD-Cx	0	0x61
	1	0x52
	2	0x9E
	3	0xAD

FragCount	Frag
0	0xE6
1	0x4C
2	0x7F
3	0xB3

Tabelle 2.2.: SMD und FragCount Codierungen [5]

Die Grösse des Data-Felds umfasst vom ersten Oktett nach dem SFD/SMD-Feld bis und mit dem letzten Oktett vor dem CRC und hat mindestens eine Grösse von 60 Bytes [3]. Da im ersten Fragment im Data-Bereich mehr als nur Daten sind, beträgt dort der effektiv kleinste Datenbereich (abzüglich MAC-Empfänger-, MAC-Sender-Adresse und EtherType-Feld, also 14 Bytes) 46 Bytes. Hat das ursprüngliche Frame zudem ein VLAN-Tag können die effektiven Daten mindestens 42 Bytes gross sein.

Eine Fragmentierung findet jedoch nur statt wenn mindestens 64 Data-Bytes versendet und mindestens 64 Data-Bytes noch ausstehen, jedoch muss das bereits Versendete ein Vielfaches von 8 sein [3].

MFCS ist die Blockprüfzeichenfolge (oder Frame Check Sequence (FCS)) eines nicht-finalen Fragments, dessen Wert derselbe wie einer FCS ist, wobei die ersten 2 der 4 Bytes invertiert sind (XOR FFFF0000). Das letzte Fragment hat dann wieder eine FCS anstelle einer MFCS, um zu signalisieren, dass es sich um das letzte Fragment dieses Frames handelt [5].

2.5. Beispiel mit IET und Mframe

Folgendes Szenario ist zu betrachten: Es wird gerade ein normales Frame gesendet. Während diesem Sendevorgang trifft ein Express-Frame ein. Der Sendevorgang des normalen Frames muss nun unterbrochen werden, indem das normale Frame fragmentiert und in das Mframe Format umgewandelt wird. Die Sequenz der zu sendenden Frame würde folgendermassen aussehen (für das Mframe Format siehe Abbildung 2.3 auf der vorherigen Seite):

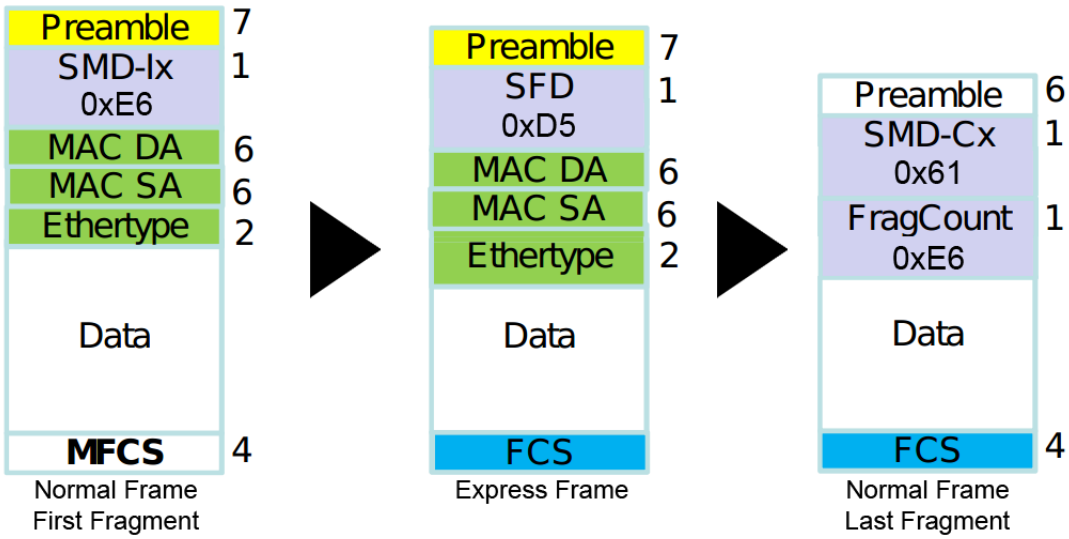

Abbildung 2.4.: Beispiel eines Sendevorgangs, bei dem ein Express Frame ein normales Frame unterbricht

Wie im Kapitel 2.4 auf Seite 16 erwähnt, können maximal 4 Frames in je maximal 5 Fragmente aufgeteilt werden. Sobald das letzte Fragment eintrifft wurde das Frame komplett übertragen und dessen Framenummer (siehe Tabelle 2.2 auf der vorherigen Seite) wird wieder für ein neues Frame verfügbar.

Teil II.

Engineering

3. Vorgehen / Methoden

- (Beschreibt die Grundüberlegungen der realisierten Lösung (Konstruktion/Entwurf) und die Realisierung als Simulation, als Prototyp oder als Software-Komponente)
- (Definiert Messgrössen, beschreibt Mess- oder Versuchsaufbau, beschreibt und dokumentiert Durchführung der Messungen/Versuche)
- (Experimente)
- (Lösungsweg)
- (Modell)
- (Tests und Validierung)
- (Theoretische Herleitung der Lösung)

3.1. Aufbau der Simulation

In OMNeT++ wird ein HSR-Ring-Netzwerk aufgebaut, in welchem der Traffic simuliert wird. Mittels Konfigurationsdatei kann man die zu verwendenden Prioritäten (siehe Kapitel 3.1.1) und Mechanismen (siehe Kapitel 3.1.2 auf der nächsten Seite) definieren, welche in den weiteren Kapiteln genauer behandelt werden.

3.1.1. Prioritäten der Ethernet-Frames

Für die Frames sind 3 verschiedene Prioritäten vorgesehen, nämlich sogenannte Express-, High- und Low-Frames. Dabei handelt es sich um das in Kapitel 2.3 auf Seite 15 beschriebene Express-Frame, wobei die High- und Low-Frames als normales Frame gelten und sich lediglich bei der Priorisierung unterscheiden.

In dieser Arbeit werden wir diese Frames wie folgt auf technischer Ebene unterscheiden:

- Ein Express-Frame wird durch einen eigenen EtherType-Wert im Ethernet-Header definiert. So werden diese Frames auch nur von den Geräten erkannt, die diesen EtherType-Wert kennen. Bis jetzt gibt es noch keine konkreten Vorschläge wie ein Express-Frame gekennzeichnet ist. Die hier erwähnte Kennzeichnung ist eine der Autoren ausgedachten Lösungen.

- Bei einem High-Frame handelt es sich um ein normales Ethernet-Frame das über ein VLAN-Tag mit der Priorität 1 versehen ist. Da das VLAN-Priority-Feld 3 Bits gross ist wären 8 verschiedene Prioritäten möglich, jedoch wird in dieser Arbeit keine Prioritätsnummer, die höher als 1 ist, verwendet.
- Als ein Low-Frame werden alle Frames definiert, welche ein VLAN-Tag mit der Priorität 0 oder keine besondere Kennzeichnung haben.

3.1.2. Mechanismen zur Trafficregelung

In diesem Kapitel werden Mechanismen erläutert, welche den Traffic regeln und in der Simulation implemeJeder Node (DANH oder Redbox, siehe Kapitel 2.2.1 auf Seite 14) im Netzwerk verfügt über einen internen Switch, der entscheidet wohin die Frames gesendet werden sollen. Zum Beispiel verfügt ein DANH in der Simulation über 3 Ports, zwei für die Frames vom und auf den Ring (Ethernet-Ports) und einen für die Frames von und zur CPU.

Für jeden möglichen Ausgang ist ein Scheduler zuständig (d.h. bei einem DANH-Gerät hat es 3 Scheduler), in welchem die Mechanismen zur Trafficregelung (siehe Kapitel 3.1.2) implementiert sind und die Frames priorisiert abhandelt. Der Scheduler ist innerhalb des Switches implementiert und erhält vom Switch Zugang zu dessen Ausgängen, die er benötigt und die Frames, die er für seinen Ausgang zu koordinieren hat.ntiert werden. Es besteht dann die Möglichkeit für jeden Knoten zwischen den Mechanismen auszuwählen oder diese sogar zu kombinieren.

3.1.2.1. Limitierung des Zuflusses von neuem Traffic

Jedes Gerät generiert intern mit einer fixen Rate eine Anzahl an Tokens bis zu einem bestimmten Maximum. Möchte das Gerät ein neues, intern generiertes Frame versenden, kann es das Frame erst versenden wenn genügend Tokens vorhanden sind. Dabei verbraucht ein Frame mit n Bytes genau n Tokens. Wenn zum Beispiel ein Frame mit 800 Bytes versendet werden muss und nur 600 Tokens vorhanden sind, muss das Gerät mit dem Versand warten bis genügend Tokens vorhanden sind.

So kann der Zufluss von neuem Traffic limitiert werden, womit kein Gerät das Netzwerk mit neuen Frames überschütten kann.

3.1.2.2. Vortrittsregeln bezüglich Frames im und zum Ring

Damit die Frames innerhalb des Rings (HSR-Netzwerks) schneller zum Ziel kommen, kann diesen Frames der Vortritt gegenüber Frames von Aussen gewährt werden. Dies ist bei gleich priorisierten Frames der Fall. Es kann hier jedoch die Möglichkeit geben, dass so nie Frames von Aussen versendet werden.

Je nach auftretenden Prioritäten kann es auch sein, dass von Aussen ein Express-Frame und vom Ring ein High-Frame kommt. Dann hat das Express-Frame trotz der Herkunft von Aussen aufgrund dessen Priorität Vortritt.

Die andere Möglichkeit ist, den Frames, die auf den Ring sollen, den Vortritt zu gewähren. So kann ein minimaler Zufluss auf den Ring garantiert werden. Hier kann es den umgekehrten Fall wie wenn die Frames vom Ring Vortritt haben geben: Wenn ständig neue Frames von Aussen kommen, werden die Frames vom Ring nie versendet.

3.1.2.3. Reissverschluss

Ein Derivat des vorherigen Mechanismus ist, den Vortritt zwischen Frames vom Ring und von Aussen zu alternieren. Die Frames würden dann wie folgt priorisiert werden (Angenommen es kommen alle 3 Prioritäten (Express, High und Low) vor):

Liste 1

1. Express-Frame vom Ring
2. Express-Frame von Aussen
3. High-Frame vom Ring
4. High-Frame von Aussen
5. Low-Frame vom Ring
6. Low-Frame von Aussen

Liste 2

1. Express-Frame von Aussen
2. Express-Frame vom Ring
3. High-Frame von Aussen
4. High-Frame vom Ring
5. Low-Frame von Aussen
6. Low-Frame vom Ring

So würde die Priorisierung zwischen der aus Liste 1 und der aus Liste 2 nach jedem Frame-Versand abwechseln.

3.1.2.4. Zeitschlitzverfahren

Im Zeitschlitzverfahren werden Zeitschlitze definiert, in denen man High- und Low-Frames senden kann, ein Express-Frame kann immer versendet werden.

Jeder Zeitschlitz hat eine sogenannte Grün-Phase, in welcher High-Frames gesendet werden können. Den Rest des Zeitschlitzes wird für den Sendevorgang oder Low-Frames verwendet. Sollten keine High-Frames da sein so können auch Low-Frames gesendet werden. Dies wiederholt sich in jedem Zeitschlitz. Mit diesem Mechanismus wird verhindert, dass Low-Frames nicht gesendet werden, wenn ständig High-Frames zum Senden bereitstehen.

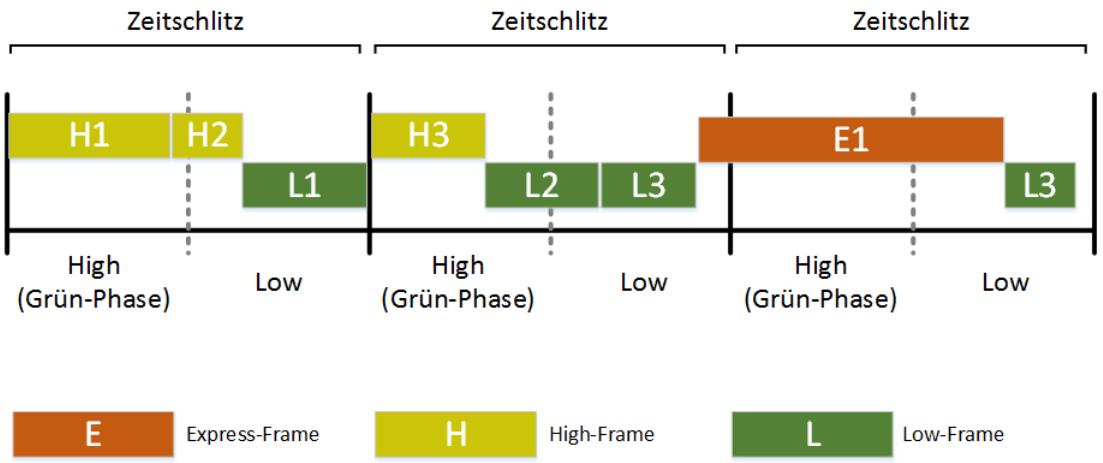

Abbildung 3.1.: Beispiel von Frameabfolge mit Zeitschlitzverfahren

3.1.3. Definition der Knoten

Jede Komponente eines Knotens (Netzwerkinterface, Switch, etc.) ist eine eigene C++-Klasse, welche die jeweiligen Funktionen enthält. Ein Knoten wird dann aus diesen Komponenten zusammengesetzt,be indem eine sogenannte NED-Datei (NED steht für Network Description) erstellt wird, in welcher die Komponenten initialisiert und verbunden werden [11]. Der Knoten selbst exisitert nicht als C++-Klasse, sondern lediglich als NED-Datei.

Der Aufbau eines DANH-Knoten in OMNeT++ sieht wie folgt aus:

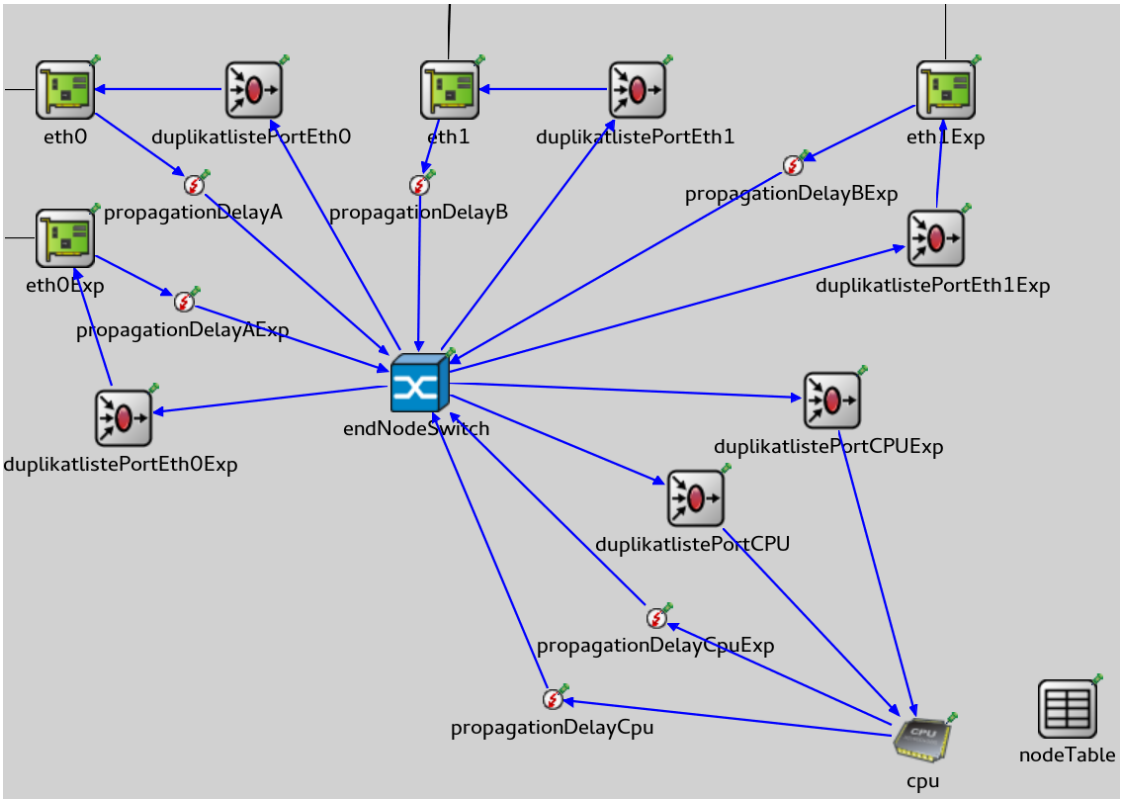

Abbildung 3.2.: DANH-Knoten in OMNeT++

Es gibt in einem DANH-Gerät 3 Ports und für jeden Port gibt es zwei Netzwerkinterfaces, einen für normale und einen für Express-Frames. Der genaue Grund, warum für jeden Port zwei Netzwerkinterfaces existieren ist im Kapitel 3.1.5.2 auf Seite 29 aufgeführt. Zwei Ports (im Bild «eth0» und «eth1») sind für den Anschluss an den Ring. Der dritte Port ist mit der CPU verbunden, die für die Generierung der neuen Frames verantwortlich ist.

Zwischen jedem Port und Komponente befindet sich eine Duplikatliste, durch die jedes Frame hindurch läuft. Die Duplikatliste überprüft anhand der Sequenznummer und der Quell-MAC-Adresse, ob dasselbe Frame schon mal die Duplikatliste passiert hat. Wenn dasselbe Frame schon mal von der Duplikatliste erkannt wurde wird es gelöscht. Durch diese Duplikatliste wird kein Frame mehrmals über den selben Port versendet.

Alle Frames, die beim DANH-Knoten ankommen oder von dessen CPU generiert werden, gelangen zum internen Switch. Dieser analysiert das Frame und weist es dem entsprechenden Scheduler zu, der die Frames je nach Priorität und aktiven Mechanismen in die entsprechende Warteschlange stellt. Dem Scheduler ist immer ein Port (also auch zwei Netzwerkinterfaces) zugeteilt, was heisst, dass ein Switch über 3 Scheduler verfügt. Der Scheduler ist zudem für das Senden der Frames zuständig. Er entscheidet wann über welches Netzwerkinterface welches Frame versendet wird.

Des Weiteren verursacht ein Knoten eine Verzögerung von 6us pro Frame (Vorgabe von unserem Betreuer, siehe Kapitel 12.2.11 auf Seite 64). Dies ist die Zeit, die ein Knoten benötigt, um beim Cut-Through-Switching genug vom Frame eingelesen zu haben, damit der Entscheiden kann was er mit dem Frame machen soll.

3.1.4. Netzwerkaufbau

So wie ein Knoten als NED-Datei seinen Aufbau mit den Komponenten beschreibt, wird ein Netzwerk auch als NED-Datei definiert, in welchem die Knoten initialisiert und verbunden werden. Verbunden werden die Knoten über eine 20m lange 100Mbps-Verbindung, welche in der Simulation durch eine Verzögerung von 100ns dargestellt wird. Das folgende Bild zeigt ein Beispielaufbau mit 10 DANH-Knoten, die als Merging und Protection Units definiert wurden (mehr dazu siehe Kapitel 3.3.1 auf Seite 38):

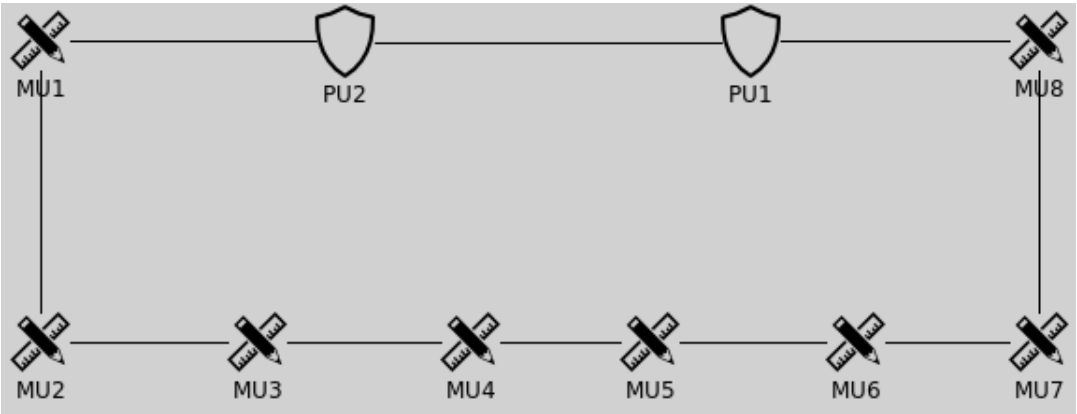

Abbildung 3.3.: Beispielaufbau eines Netzwerks in OMNeT++

Das Verhalten in diesem Beispiel ist bei jedem Knoten das Selbe, da es sich lediglich um DANH-Knoten handelt. Was variiert sind die MAC-Adresse (in der NED-Datei des Netzwerks definiert) und die Anzahl und Häufigkeit der Frames, die generiert werden. Für die Generierung von Frames wird eine XML-Datei definiert, bei der für jeden Knoten bestimmt wird wie viele Frames dieser Knoten an wen sendet. Das Generieren von Multi- und Broadcast-Frames ist auch möglich.

Während der Simulation wird die Komponente, die gerade eine Aktion durchführt, rot umrandet. Zudem verändert sich die Farbe der Verbindungen und der Netzwerkinterfaces innerhalb der Knoten («eth...», siehe Abbildung 3.2 auf der vorherigen Seite). Die Farben haben folgende Bedeutungen:

Gelb	Am übertragen
Rot	Kollision erkannt (wird in den Simulationen dieser Arbeit nicht vorkommen, da Vollduplex-Verbindungen verwendet werden)

Tabelle 3.1.: Bedeutung der Farben bei den Verbindungen und Netzwerkinterfaces in der Simulation [10]

Es folgt ein Beispiel einer XML-Konfiguration für den Lastgenerator eines bestimmten Knoten:

```
1 <!--
2 Define framegenerator for node with MAC 00-15-12-14-88-04
3 (has to be defined in NED-file from the network)
```

```
4  -->
5  <source>00-15-12-14-88-04
6
7    <lastmuster>
8      <typ>hsr</typ>
9
10 <!--
11     Behavior
12       Constant load -> "konstante last",
13       Evenly distributed load -> "gleichverteilte last",
14       Normally distributed load -> "normalverteilte last"
15 -->
16     <verhalten>konstante last</verhalten>
17
18 <!--  Time when the first frame will be generated -->
19     <startzeit>0.001</startzeit>
20
21 <!--  Generate frame until this time is reached -->
22     <stopzeit>0.06</stopzeit>
23
24 <!--  Generate a frame every x seconds -->
25     <interval>0.00025</interval> <!-- 4000x per second -->
26
27 <!--
28     (Only for constant load behavior)
29     Generate bea random value between 0 and epsilon that
30     will be added to the value of the previous tag
31     every time a new frame will be generated
32 -->
33     <epsilon>0.0000125</epsilon> <!-- 12.5 us -->
34
35 <!--  Set destination of frame (can be Uni-, Multi- or Broadcast) -->
36     <destination>01-15-12-14-88-0A</destination>
37
38 <!--  Set datasize of frame (Size + 24 = Total Framesize) -->
39     <paketgroesse>136</paketgroesse>
40
41 <!--  Set priority of frame ("EXPRESS", "HIGH", "LOW") -->
42     <priority>HIGH</priority>
43
44   </lastmuster>
45
46 </source>
```

Listing 3.1: Konfiguration des Lastgenerators eines Knotens

Für einen Knoten können mehrere solcher Konfigurationen gleichzeitig gesetzt werden, wenn ein Knoten z.B. Frames der Priorität «HIGH» und «LOW» versenden soll. Des Weiteren können auch Knoten im Netz ohne einen Eintrag im XML existieren. Diese Knoten generieren dann zwar selber keine Frames, leiten jedoch Frames normal weiter wie jeder andere Knoten auch.

Sind keine Frames mehr im Umlauf und werden keine mehr generiert (wenn die Simulationszeit über der Zeit im «Stopzeit»-Tag ist), dann wird die Simulation automatisch beendet. Während der Simulation wurde einiges aufgezeichnet, was nachher eingesehen und als Grafik dargestellt werden kann (Siehe Kapitel 4 auf Seite 41).

3.1.5. Abhandlung der Frames innerhalb eines Gerätes

Sobald ein Gerät ein Frame erhält, wird es in dessen Switch analysiert und die Herkunft und das Ziel ermittelt, um entscheiden zu können, an welchen Ausgang das Frame weitergeleitet

werden muss. Je nach Ausgang wird das Frame dann an den jeweiligen Scheduler weitergeleitet. Muss ein Frame an alle Ethernet-Ports gesendet werden (wenn z.B. ein Frame von der CPU des Gerätes generiert wird), dann wird dies vom Switch dupliziert und an die betroffenen Scheduler gesendet. Der Scheduler ermittelt dann die Priorität (und eventuell die Herkunft) des Frames, um diese dann in die entsprechende Queue einzuordnen. Die Herkunft ist nur je nach Mechanismus notwendig, wenn z.B. Frames, die vom Ring kommen, priorisiert werden (siehe Kapitel 3.1.2.2 auf Seite 22).

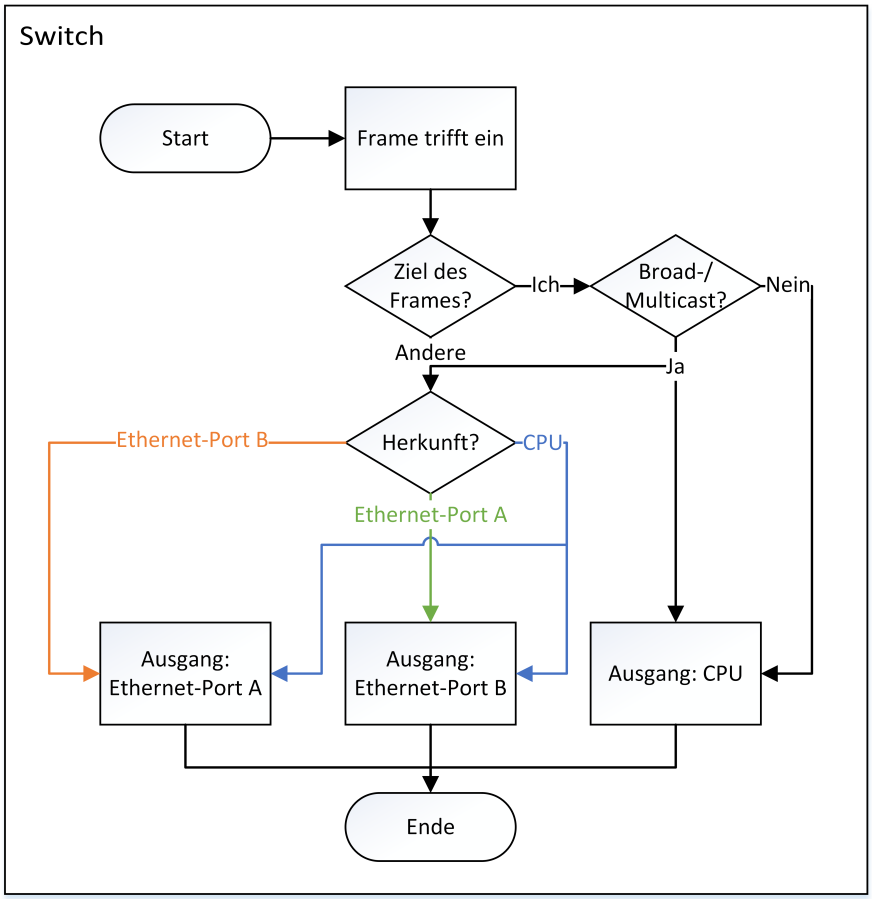

Abbildung 3.4.: Beispielablauf und Schedulerzuordnung eines Frames in einem Switch

Der Scheduler arbeitet nach jedem neuen Frame seine Queues ab. Dabei überprüft er, ob auf seinem Ausgang gerade etwas gesendet wird und sendet bei einem freien Ausgang das nächste Frame, das je nach Priorität und Mechanismus an der Reihe ist. Wird gerade etwas auf dem Ausgang übertragen, versucht es der Scheduler erst nochmals sobald die Übertragung zu Ende ist.

3.1.5.1. Express-Frame

Bei einem Express-Frame muss der Scheduler sofort handeln. Anders als bei einem normalen Frame muss das Express-Frame auch bei einem besetzten Ausgang versendet werden, weshalb das Frame, das gerade übertragen wird, wenn möglich fragmentiert werden muss.

Wichtig ist, dass kein Fragment unter der kleinstmöglichen Ethernet-Frame-Grösse von 64 Bytes liegt. Aus diesem Grund kann es sein, dass Frames manchmal nicht fragmentiert werden können und der Scheduler warten muss bis dieses versendet wurde.

Kann das Frame jedoch fragmentiert werden, so wird das derzeitige Fragment zu Ende gesendet und dann das Express-Frame versendet. Danach werden die restlichen Fragmente versendet.

3.1.5.2. Fragmentierung und Zeitberechnung

In der Simulation findet jedoch keine richtige Fragmentierung statt, jedoch wird diese über Zeitberechnung miteinbezogen. Das heisst, dass ein Express-Frame bei einem belegten Ausgang gesendet wird, wenn das Fragment inklusive IFG versendet worden wäre. Die Ankunftszeit des «fragmentierten» Frame wird dann um die Dauer des Express-Frames inkl. dessen IFG verlängert.

Da keine wirkliche Fragmentierung statt findet, ist der Ausgang bei dem das zu unterbrechende Frame gesendet wird immer noch belegt. Aus diesem Grund gibt es für jeden Ausgang einen weiteren Ausgang ausschliesslich für Express-Frames.

Somit sind die Ankunfts- und Sendezeiten der Frames dieselben wie wenn eine tatsächliche Fragmentierung stattgefunden hätte.

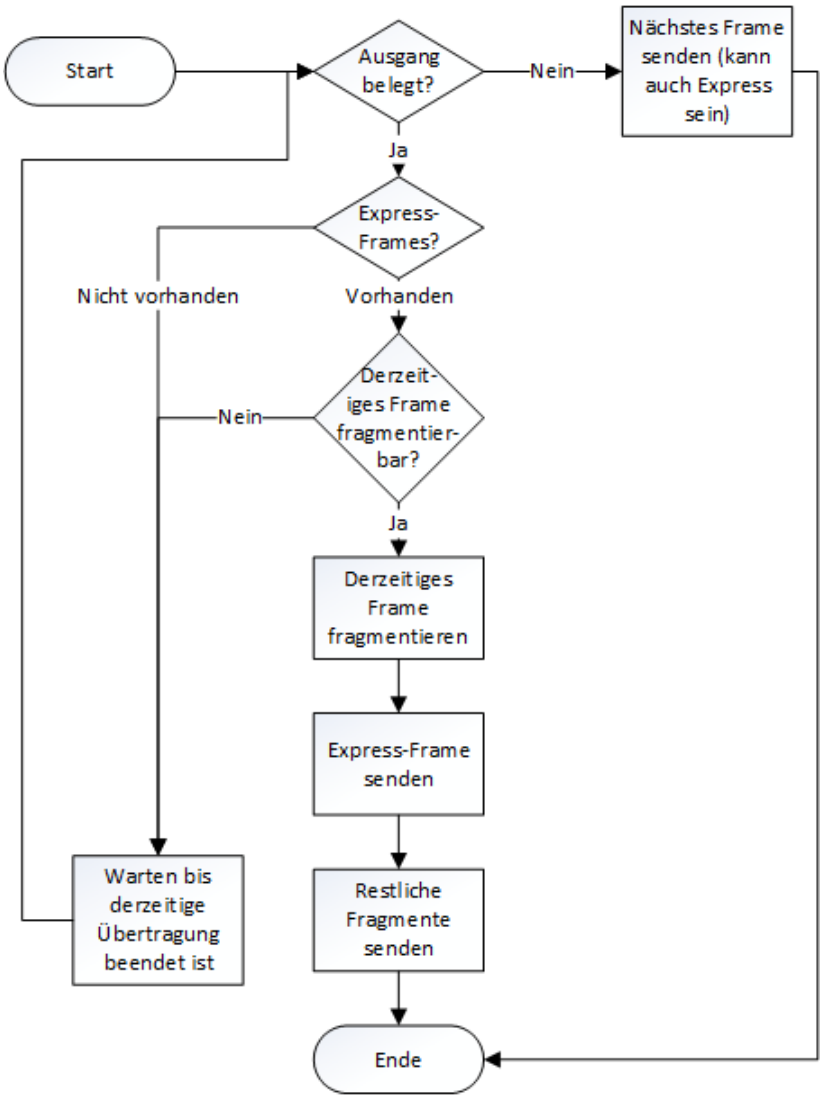

Abbildung 3.5.: Abhandlung von Frames inklusive Express-Frames

Der Datenbereich eines Mframes umfasst vom ersten Oktett nach dem SFD/SMD-Feld bis und mit dem letzten Oktett vor dem CRC und hat mindestens eine Grösse von 60 Bytes. Dabei muss dessen Grösse durch 8 (Alignment der Fragmentierung) teilbar sein (siehe Kapitel 2.4 auf Seite 16).

Natürlich kann bei einem Frame, das gerade übertragen wird nur noch der Teil fragmentiert werden, der aussteht. Demnach wird wie folgt vorgegangen, um herauszufinden, ob ein Frame fragmentierbar ist:

Variablen:

$$simtime = Derzeitige\,Simulationszeit\,(Jetztiger\,Zeitpunkt)\,[s]$$

$$sendtime = Zeit,\,an\,der\,die\,Übertragung\,der\,Präambel\,begonnen\,hat\,[s]$$

$$F = Grösse\,des\,zu\,fragmentierenden\,Frames\,ohne\,Präambel\,und\,SMD\,[Byte]$$
$$V = Bereits\,versendete\,Bytes\,inkl.\,Präambel\,und\,SMD\,[Byte]$$
$$O = Noch\,offen\,stehender\,Teil\,des\,zu\,fragmentierenden\,Mframes\,[Byte]$$
$$D = Grösse\,des\,Datenbereichs\,eines\,Mframes,\,wenn\,man\,jetzt\,fragmentieren\,würde\,[Byte]$$
$$t = Zeitpunkt,\,zu\,dem\,das\,Express\,Frame\,versandt\,wird\,[s]$$

Konstanten:

$$datarate = Übermittlungsrate\,[Byte/s]$$
$$P = Präambel\,+\,Start\,of\,Frame\,bzw.\,Mframe\,Delimiter = 8\,Bytes$$
$$C = FCS\,bzw.\,MFCS = 4\,Bytes$$
$$I = Interframe\,Gap\,(IFG) = 12\,Bytes$$
$$A = Alignment\,der\,Fragmentierung = 8\,Bytes$$

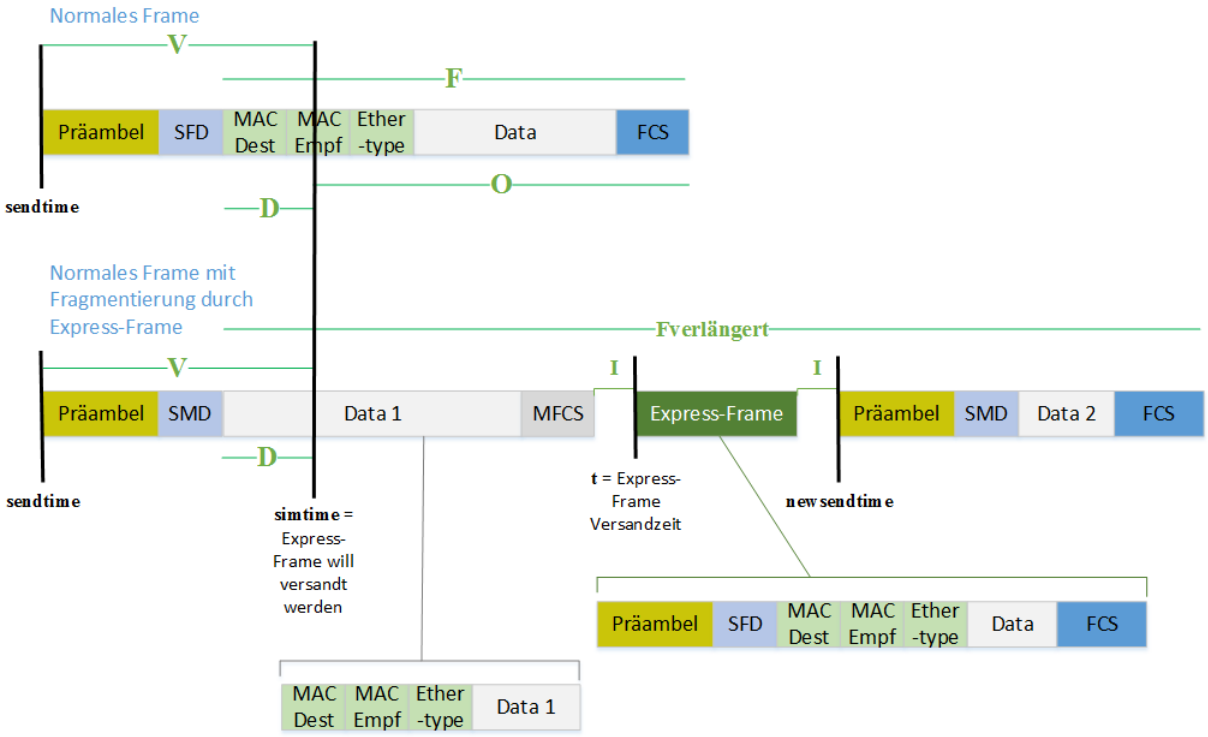

Abbildung 3.6.: Berechnung der Express-Frame-Versandzeit

1. Express-Frame muss versendet werden.

2. Hole Grösse des gesamten Frames «F», das gerade gesendet wird.
   Beträgt «F» weniger als 136 Bytes («P», 64 Bytes + 60 Bytes und «C» zusammen), kann das Frame nicht fragmentiert und der weitere Vorgang abgebrochen werden. Das heisst, dass man mit dem Senden des Express-Frames warten muss, bis das nicht fragmentierbare Frame und dessen IFG fertig versandt wurden.

3. Berechne «V» und «D»:

$$V = (simtime - sendtime) * datarate$$

$$D = V - P - C - I = V - 24$$

4. Berechne «O» und rechne die «C», das pro neuem Fragment dazukommt, hinzu. Des Weiteren muss die Präambel aus «V» nicht mit abgezogen werden, da die Präambel nicht zur Framegrösse gehört:

$$O = F - V + C + P = F - V + 12$$

5. Wenn «V» grösser gleich 72 Bytes (Präambel + SMD + Data + (M)FCS) und IFG (von der Simulationsumgebung vorgegeben) zusammen beträgt, «D» durch 8 geteilt 0 ergibt und «O» grösser als 64 Bytes ist (Data-Felder je mindestens 60 Bytes + 4 Bytes (M)FCS), kann das Express-Frame jetzt gesendet werden. Die Schritte 6 und 7 müssen nicht mehr erledigt werden, da das Express-Frame versandt wurde und der Vorgang somit erfolgreich war. In diesem Fall beträgt «t»:

$$t = simtime$$

Da es sich hier um eine Zeitberechnung haltet, wird das normale Frame nicht wirklich fragmentiert, sondern verlängert. Die Ankunftszeit des verlängerten Frames ist die gleiche wie wann alle Fragmente eines Frames ankommen würden.

Folgende Werte müssen dem «fragmentierten» Frame dazugerechnet werden:
Grösse normales Frame «F»
+ MFCS des ersten Fragments («C»)
+ IFG des ersten Fragments («I»)
+ Präambel und SFD des Express-Frames («P»)
+ Grösse Express-Frame
+ IFG des Express-Frames («I»)
+ Präambel und SMD des nächsten Fragments («P»)
= Totale Grösse des verlängerten Frames «Fverlängert» (siehe vorherige Abbildung)

$$Fverlängert = F + C + I + P + (Grösse\,Express\,Frame) + I + P$$

$$= F + C + 2 * I + 2 * P + (Grösse\,Express\,Frame) = F + 44 + (Grösse\,Express\,Frame)$$

Die «sendtime» des fragmentierten Frames wird nun auf die Zeit gesetzt, an der das Express-Frame + IFG fertig versendet wurde («newsendtime», siehe Abbildung 3.6 auf der vorherigen Seite). So wird sichergestellt, dass wenn das Frame erneut fragmentiert werden muss, es ab Beginn des nächsten Fragments betrachtet wird. So können keine zu kleine Fragmente entstehen.

$$newsendtime = Zeitpunkt,\, zu\,dem\,Express\,Frame + I\,fertig\,versandt\,wurden$$

$$sendtime = newsendtime$$

Somit kann dann bei der erneuten Fragmentierung das nächste Fragment so betrachtet werden, als wäre es das Erste des Frames (die Framegrösse muss jedoch dem entsprechend angepasst werden, nur noch das Fragment mit dem «Data 2» Feld in Abbildung 3.6 auf der vorherigen Seite betrachten).

6. Wenn «V» nicht grösser gleich 72 Bytes und IFG zusammen beträgt oder «D» bei der Division durch 8 nicht 0 ergibt, «O» aber grösser gleich 64 Bytes ist, dann muss abgewartet werden bis folgender Zeitpunkt «t» erreicht ist:

$$t = \begin{cases} (V \geqq 72 + I)\ \&\ (D\%8 = 0) & simtime \\ (V < 72 + I)\ \&\ (D\%8 = 0) & simtime + \frac{72+I-V}{datarate} \\ (V \geqq 72 + I)\ \&\ (D\%8\,! = 0) & simtime + \frac{8-(D\,\%\,8)}{datarate} \\ (V < 72 + I)\ \&\ (D\%8\,! = 0) & simtime + \frac{72+I-V+(8-(D\,\%\,8))}{datarate} \end{cases}$$

   Diese Formel berechnet den nächsten Zeitpunkt, ab dem eine Fragmentierung möglich wäre. Ist «V» schon über oder gleich der Minimalgrösse (72 Bytes) + IFG, muss lediglich abgewartet werden bis «D» durch 8 dividiert 0 ergibt. Ist «V» kleiner als die Minimalgrösse + IFG, muss abgewartet werden bis diese erfüllt ist und die selben Bedingungen erfüllt sind wie wenn «V» die Minimalgrösse + IFG schon erreicht hat.

7. Ist der Zeitpunkt erreicht, muss die Überprüfung wieder von Anfang an gemacht werden.

3.1.6. Generierung von Traffic (Lastprofile)

In jedem Node ist ein Generator implementiert, welcher mittels Konfigurationsparameter Frames in verschiedenen Folgemustern generieren kann. In diesem Kapitel werden verschiedene Muster von Traffic (Lastprofile) aufgezeigt, die in der Simulation generiert werden können.

Die Lastprofile teilen dabei folgende Eigenschaften bezüglich der Prioritäten der Frames:

- Express-Frames sind klein und kommen selten vor. Sie sollen Alarme in einem System darstellen, welche schnellstmöglich übermittelt werden müssen.
- Frames der Priorität «High» stellen den normalen Verkehr dar. Dies kann unter anderem Down- und Upload von Dateien, Monitoring, etc. sein. Die Grösse der Frames kann klein und gross sein.
- Die Frames mit der Priorität «Low» bilden den Background-Traffic, der oft vorkommt und wie der normale Verkehr verschieden gross sein kann.

Bezüglich der Auftretenswahrscheinlichkeit der Frames lässt sich demnach folgendes sagen: Je höher die Priorität, desto seltener kommt das Frame vor.

Ein Beispiel eines Lastprofils findet man in der XML-Konfiguration in Kapitel 3.1.4 auf Seite 26.

3.2. Überprüfung der Implementation

Damit man sich sicher sein kann, dass die Simulation wie geplant verläuft, wird die Implementation vor der Simulation der Szenarien überprüft.

3.2.1. Aufbau der Testumgebung

Für die Tests wird in der Simulation ein HSR-Ring mit 3 DANH-Geräten («Node1», «Node2» und «Node3») implementiert.

3.2.2. Verhaltensüberprüfung

3.2.2.1. Frames richtig weiterleiten und empfangen

Soll

Ein Frame wird von einem Knoten, der nicht der einzige Empfänger des Frames ist, an den nächsten Knoten weitergeleitet. Das Frame wird bei einem Unicast-Frame vom Empfänger und bei einem Multi-/Broadcast-Frame vom Sender vom Netz entfernt.

Ist

Der sich im Knoten befindende Switch analysiert die Quell- und Ziel-Adresse und den Herkunftsport des Frames und leitet es an den entsprechenden Port (links, rechts auf den Ring oder zur CPU) weiter.
Frames, welche vom eigenen Switch stammen und nicht von der CPU kommen werden entfernt, um zirkulierende Frames (können Uni-, Multi- oder Broadcast sein) zu verhindern.
Multi- und Broadcast-Frames (die nicht vom Knoten selbst stammen) werden an alle Ports ausser dem Herkunftsport weitergeleitet.

Nachweis

Für den Nachweis dieses Tests werden 3 verschiedene Fälle betrachtet und den Nachrichtenverlauf der jeweiligen Simulation angezeigt. Im «Namen» des Frames (siehe nächstes Bild) steht der Knoten, der das Frame generiert hat, die Priorität und die Sequenznummer des Frames. Am Anfang des «Namens» steht zudem, ob es sich um ein Unicast-, Multicast- oder Broadcast-Frame handelt.

Unicast-Frame von «Node1» zu «Node3»

Event#	Time	Src/Dest	Name
#7	0.001	Node1 --> Node2	Uni From: Node1 (LOW) Seq#0
#8	0.001	Node1 --> Node3	Uni From: Node1 (LOW) Seq#0
#15	0.001013794881	Node2 --> Node3	Uni From: Node1 (LOW) Seq#0

Multicast-Frame von «Node1»

Event#	Time	Src/Dest	Name
#7	0.001	Node1 --> Node2	Multi From: Node1 (LOW) Seq#0
#8	0.001	Node1 --> Node3	Multi From: Node1 (LOW) Seq#0
#17	0.001013794881	Node2 --> Node3	Multi From: Node1 (LOW) Seq#0
#25	0.001013825794	Node3 --> Node2	Multi From: Node1 (LOW) Seq#0
#39	0.001027606399	Node3 --> Node1	Multi From: Node1 (LOW) Seq#0
#46	0.00102765022	Node2 --> Node1	Multi From: Node1 (LOW) Seq#0

Broadcast-Frame von «Node1»

Event#	Time	Src/Dest	Name
#7	0.001	Node1 --> Node2	Broad From: Node1 (LOW) Seq#0
#8	0.001	Node1 --> Node3	Broad From: Node1 (LOW) Seq#0
#17	0.001013794881	Node2 --> Node3	Broad From: Node1 (LOW) Seq#0
#25	0.001013825794	Node3 --> Node2	Broad From: Node1 (LOW) Seq#0
#39	0.001027606399	Node3 --> Node1	Broad From: Node1 (LOW) Seq#0
#46	0.00102765022	Node2 --> Node1	Broad From: Node1 (LOW) Seq#0

Tabelle 3.2.: Test: Frameablauf in der Testumgebung

3.2.2.2. Beachten der Priorisierung

Soll	Wenn zwei Frames mit unterschiedlichen Prioritäten ankommen soll das mit der höheren Priorität vor dem anderen verarbeitet werden.
Ist	Der sich im Knoten befindende Switch hat für jeden seiner Ports einen Scheduler. Jeder dieser Scheduler verfügt über Queues, für jede Priorität und Herkunft (von Ring oder Aussen) eine, also total 6 Queues. Der Switch fügt ein ankommendes Frame beim jeweiligen Scheduler in die jeweilige Queue ein. Der Scheduler beginnt darauf mit dem Abarbeiten der Queues und startet dabei mit der Queue der höchsten Priorität. Wenn z.B. zwei Frames, eines mit der Priorität «HIGH» und eines mit der Priorität «LOW» zur exakt selben Zeit bei einem Scheduler eingereiht werden, wird garantiert das Frame mit der Priorität «HIGH» zuerst versendet.
Nachweis	Folgende Frames werden zur exakt selben Zeit generiert: • Unicast-Frame von «Node1» zu «Node3» mit Priorität «EXPRESS» • Unicast-Frame von «Node1» zu «Node3» mit Priorität «HIGH» • Unicast-Frame von «Node1» zu «Node3» mit Priorität «LOW»

Tabelle 3.3.: Test:

3.2.2.3. Express-Frames und Fragmentierung

Soll	Soll ein Express-Frame versendet werden, so wird dies sofort versandt wenn derzeit keine anderen Frames auf demselben Ausgang versendet werden. Wird jedoch etwas darauf gesendet, so wird das Express-Frame zu dem Zeitpunkt versandt, an dem ein Fragment des anderen Frames (inklusive Interframe Gap) fertig versendet worden wäre. Die Dauer, die das Express-Frame inkl. IFG beanspruchte, wird der Ankunftszeit des «fragmentierten» Frames angerechnet.
Ist	
Nachweis	Langes Frame und 2 Express Nicht fragmentierbar Was wenn fragmentierbar?

Tabelle 3.4.: Test:

3.2.2.4. Zuflusslimitierung

Soll
Ist
Nachweis

Tabelle 3.5.: Test:

3.2.2.5. Vortrittsregeln im und zum Ring

Soll
Ist
Nachweis

Tabelle 3.6.: Test:

3.2.2.6. Reissverschluss

Soll
Ist
Nachweis

Tabelle 3.7.: Test:

3.2.2.7. Zeitschlitzverfahren

Soll
Ist
Nachweis

Tabelle 3.8.: Test:

3.2.2.8. xxxxxxxxxxxxxxxxx

Soll
Ist
Nachweis

Tabelle 3.9.: Test:

3.2.2.9. xxxxxxxxxxxxxxxxxx

Soll
Ist
Nachweis

Tabelle 3.10.: Test:

Verhält sich Frame wie geplant?

Was wenn....?

Prioritäten?

«Preemption»?

3.3. Simulation

In diesem Kapitel werden die Szenarien definiert, die in dieser Arbeit simuliert werden. Die Resultate der jeweiligen Simulation sind im Kapitel 4 auf Seite 41 aufgeführt. Wie diese Szenarien selber simuliert werden können, wird in der Anleitung in 13 auf Seite 65 beschrieben.

3.3.1. Szenario 1: Substation Automation

Der Aufbau dieses Szenarios ist ein HSR-Ring mit DANH-Knoten. Die Aufgabe im Anwendungsfall Substation Automation ist der Schutz von Schaltern und Leitungen sicher zu stellen.

Im HSR-Ring befinden sich sogenannte Merging Units (MU), welche die Messwerte mit Sensoren erfassen und diesemit Zeitstempel und sonstigen Steuerinformationen in ein Frame packen. Ein solches Frame hat eine Gesamtgrösse von 160 Bytes (inklusive Header) und die Priorität «High». Eine MU verschickt konstant 4000 mal pro Sekunde ein solches Frame via Multicast (Publisher / Subscriber Modell). Ziel dieser Frames sind Protection Units (PU) und eventuell andere MUs. Protection Units sind im Netzwerk doppelt vorhanden und treffen anhand der erhaltenen Messwerte Entscheidungen. MUs und PUs sind DANH-Knoten.

Neben den erwähnten Frames gibt es spontane Einzelmeldungen (Express-Frames, auch Multicast), welche selten und zufällig vorkommen. Die grösse dieser Express-Frames beträgt ca. 100 Bytes.

Als Background-Traffic wird von 2 Knoten TCP-ähnlicher Traffic generiert, bei dem ein Knoten (Knoten A) alle 200 Sekunden Frames mit einer Grösse von 1500 Bytes an den anderen Knoten (Knoten B) sendet (Unicast). Knoten B sendet zur selben Zeit auch alle 200 Sekunden Frames à 64 Bytes an Knoten A.

Sofern es nicht anders erwähnt wird, befinden sich in den Simulationen 14 MUs und 2 PUs, die in einem Ring mit einer 100Mbps-Verbindung verbunden sind. Die Prioritäten der Frames werden in jeder Simulation berücksichtigt.

3.3.1.1. Simulation 1.1: First Come First Serve

Szenario 1 wird ohne spezielle Mechanismen in den Knoten simuliert. Es spielt in dieser Variante keine Rolle, ob ein Frame von Aussen oder vom Ring kommt.

3.3.1.2. Simulation 1.2: Vortritt für Frames vom Ring

Jeder Knoten im Ring gewährt den Frames, die vom Ring kommen, den Vortritt gegenüber den Frames, die von Aussen kommen. Die Frames, die von Aussen kommen, sind dabei die, die vom Knoten selbst generiert werden.

3.3.1.3. Simulation 1.3: Vortritt für Frames von Aussen

Jeder Knoten im Ring gewährt den Frames, die von Aussen kommen, den Vortritt gegenüber den Frames, die vom Ring kommen. Die Frames, die von Aussen kommen, sind dabei die, die vom Knoten selbst generiert werden. Wenn ein Knoten ständig Frames generiert kann es sein, dass die Frames vom Ring bei diesem Knoten nie zum Zuge kommen und somit nie weitergeleitet werden.

3.3.1.4. Simulation 1.4: Vortritt für Frames von Aussen mit Zuflusslimitierung

Die Simulation 1.3 wird mit einer Zuflusslimitierung erweitert. Mit Hilfe dieser Limitierung kann ausgeschlossen werden, dass Frames vom Ring nie weitergeleitet werden wenn ein Knoten ständig Frames generiert.

3.3.1.5. Simulation 1.5: Reissverschluss

Es wird in jedem Knoten abwechslungsweise der Vortritt für Frames vom Ring und von Aussen gewährt. Wurde zum Beispiel ein Frame vom Ring versendet, wird als nächstes ein Frame von Aussen versendet, sofern eines gesendet werden muss.

3.3.1.6. Simulation 1.6: Zeitschlitzverfahren

In jedem Knoten wird das Zeitschlitzverfahren (siehe Kapitel 3.1.2.4 auf Seite 23) angewendet.

3.3.1.7. Simulation 1.7: Maximale Auslastung

Anstelle von 14 werden 17 MUs eingesetzt. Zudem werden in dieser Simulation lediglich Messwerte und Express-Frames versendet, der TCP-ähnliche Traffic wird hier nicht simuliert. Laut unserem Betreuer (siehe Besprechung in Kapitel 12.2.10 auf Seite 62) sollte dann neben dem Traffic nichts mehr sonst Platz auf dem Ethernet haben.

3.3.1.8. Simulation #.#: xxxxxxxxxxxxxxxxxxxxxx

Mechanismen:

Keine (FCFS), Zuflusslimitierung, Vortritt Ring / von Aussen, Reissverschluss, Zeitschlitzverfahren

4. Resultate und Interpretation

Die Resulate der Simulationen aus Kapitel 3.3 auf Seite 38 werden in diesem Kapitel aufgeführt und interpretiert. Folgende Eigenschaften werden während der Simulation aufgezeichnet, um sie anschliessend analysieren zu können:

- **Übertragungsdauer eines Frames von der Erstellung bis zur Löschung**
  Wird ein Frame erstellt, wird es in die jeweilige Queue beim Knoten eingereiht. Das heisst, dass die Zeit zwischen der Erstellung und dem ersten Versand auch zur Übertragungsdauer gehört. Ein Unicast-Frame wird gelöscht, wenn es bei seinem Empfänger oder wieder beim Sender ankommt. Multi-/Broadcast-Frames werden gelöscht, sobald sie wieder beim Sender ankommen.
  Die Resultate dieser Eigenschaft werden in einem Histogramm gespeichert, auf welchem ersichtlich sein wird, wie viele Frames welche Übertragungsdauer haben.

- **Grösse der Queues bei den Schedulern**
  Sobald ein Frame bei einer Queue eingereiht oder von einer Queue entnommen wird, wird die Grösse der Queue jedes mal aufgezeichnet. So lässt sich feststellen, zu exakt welchem Zeitpunkt ein Frame in die Queue kommt und wann ein Frame die Queue verlässt.

(Zusammenfassung der Resultate)

5. Diskussion und Ausblick

- Bespricht die erzielten Ergebnisse bezüglich ihrer Erwartbarkeit, Aussagekraft und Relevanz
- Interpretation und Validierung der Resultate
- Rückblick auf Aufgabenstellung, erreicht bzw. nicht erreicht
- Legt dar, wie an die Resultate (konkret vom Industriepartner oder weiteren Forschungsarbeiten; allgemein) angeschlossen werden kann; legt dar, welche Chancen die Resultate bieten

5.1. Erfüllung der Aufgabenstellung

5.1.1. HSR-Knoten

5.1.1.1. Zwei Prioritäten

Soll	Der Knoten soll zwei Prioritäten unterstützen, d.h. zwei Warteschlangen pro Interface bewirtschaften.
Ist	Wenn zwei Frames mit unterschiedlichen Prioritäten (Low & High) zur gleichen Zeit eintreffen wird das Frame mit der Priorität High als Erstes weitergesendet.
Nachweis	Die Frames werden in unterschiedliche Queues (Warteschlangen) gesetzt und diese werden der Reihe nach abgearbeitet (Absteigend nach Priorität). Es existieren pro Scheduler 6 Queues, dessen Nummerierungen in einem Enum gespeichert sind:

```
1 typedef enum {
2     EXPRESS_RING,
3     EXPRESS_INTERNAL,
4     HIGH_RING,
5     HIGH_INTERNAL,
6     LOW_RING,
7     LOW_INTERNAL
8 } queueName;
```

Listing 5.1: hsrDefines.h - Enum mit Queuenummerierung

Der Switch analysiert das erhaltene Frame nach Priorität und Herkunft und teilt dem Scheduler mit, in welcher Queue er das Frame einordnen soll. Ein Ausschnitt dieser Analyse und Zuteilung (Herkunft des Frames ist im Beispiel von der CPU und wird an beide Ausgänge zum Ring zugeteilt) ist in Kapitel 15.1 auf Seite 68 einsehbar.
Nach der Zuteilung werden die Queues je nach Mechanismus im Scheduler abgearbeitet.

Tabelle 5.1.: Nachweis: Knoten unterstützt zwei Prioritäten

5.1.1.2. IET

Soll	Der Knoten soll Interspersing Express Traffic (IET) unterstützen, d.h. Express Frames können die aktuell ablaufende Übertragung eines Frames unterbrechen.
Ist	Die Unterbrechung findet nicht wirklich, sondern nur in der Zeitberechnung statt. Sobald ein Express-Frame ankommt wird es zu dem Zeitpunkt, zu dem ein Fragment des zu unterbrechenden Frames übertragen wurde, mit Interframe Gap (IFG), versendet. Danach wird die Zeit, die für die Übertragung des Express-Frames plus IFG und Grösse der neu entstandenen Felder der weiteren Fragmente verwendet wurde, dem «fragmentierten» Frame hinzugefügt. Dies hat für die Auswertung den selben Effekt wie wenn eine Fragmentierung stattgefunden hätte (Express-Frame wird während normalem Frame versandt, normales Frame kommt später an).
Nachweis	

Tabelle 5.2.: Nachweis: Knoten unterstützt IET

5.1.1.3. Limitierung Ringzufluss

Soll	Der in den Ring einfliessende Traffic kann limitiert werden.
Ist	Es wurde der Mechanismus implementiert, dass Frames erst versendet werden können, sobald eine gewisse Anzahl an Tokens generiert wurde (es gibt ein Maximum an Tokens). Beim Versand werden diese Tokens verbraucht (ein Token pro Byte). Ein weiteres Frame kann erst wieder versandt werden, sobald genügend Tokens vorhanden sind. Für genauere Informationen zum Mechanismus siehe Kapitel 3.1.2.1 auf Seite 22.
Nachweis	

Tabelle 5.3.: Nachweis: Knoten kann Zufluss limitieren

5.1.1.4. Variierung der Vortrittsregeln

Soll	Die Vortrittsregeln bezüglich der im Ring zirkulierenden Frames und den Frames, die in den Ring einfliessen, können variiert werden (z.B. „zirkulierende Frames haben immer Vortritt“ oder „minimaler Zufluss wird garantiert“).
Ist	Bezüglich der Herkunft gibt es zwei Arten von Verkehr: Vom Ring und von Aussen auf den Ring. Dabei sind 3 Arten von Vortrittsregeln implementiert: • Frames vom Ring haben Vortritt («zirkulierende Frames haben immer Vortritt») • Frames von Aussen haben Vortritt («minimaler Zufluss wird garantiert») • Reissverschluss (Der Vortritt für Frames vom Ring und von Aussen wechselt sich ab: Wenn ein Frame vom Ring Vortritt hatte, hat als Nächstes das Frame von Aussen Vortritt)
Nachweis	Der Mechanismus ist in der Datei «omnetpp.ini» definiert und wird vom Switch bei der Initialisierung in dessen Schedulern gesetzt. ... Ein Ausschnitt, bei dem aufgezeigt wird wie die Mechanismen ablaufen, ist in Kapitel 15.2 auf Seite 69 einsehbar.

```
1 ...
2 # Define Schedulermode ("FCFS", "RING_FIRST", "INTERNAL_FIRST", "ZIPPER
      ", "TOKENS")
3 **.endNodeSwitch.schedulerMode = "ZIPPER"
4
5 # Define Schedulermode for one device, for example EndNode "MU1"
6 **.MU1.endNodeSwitch.schedulerMode = "FCFS"
```

Listing 5.2: omnetpp.ini - Definition Mechanismus im Switch

Tabelle 5.4.: Nachweis: Knoten verfügt über unterschiedliche Vortrittsregeln

5.1.1.5. Zeitschlitzverfahren

Soll	Der Knoten implementiert ein Zeitschlitzverfahren, welches dem zeitkritischen Traffic und dem Bulk Traffic je eine Phase zuordnet.
Ist	Es ist ein Mechanismus implementiert, welcher einen Intervall in zwei Phasen aufteilt. In der ersten Phase werden so viele Frames mit der Priorität High (Zeitkritischer Traffic) verschickt wie möglich. Wenn in dieser Phase noch Platz frei ist, es jedoch keine Frames mit der Priorität High mehr gibt, dann können Frames mit der Priorität Low (Bulk Traffic) auch innerhalb dieser Phase versendet werden. In der zweiten Phase können lediglich Frames der Priorität Low versendet werden. Express-Frames (gehören auch zum zeitkritischen Traffic) können natürlich zu jeder Zeit versendet werden.
Nachweis	

Tabelle 5.5.: Nachweis: Knoten kann nach Zeitschlitzverfahren arbeiten

5.1.2. Lastgenerator

5.1.2.1. Konstante Framerate

Soll	Der Lastgenerator generiert für die Simulation ein Verkehrsaufkommen mit konstanter Framerate.
Ist	Es lässt sich ein Strom mit konstanter Framerate generieren. Mittels Konfigurationsdatei kann man die Rate (in welchen Zeitabständen ein neues Frame generiert werden soll) bestimmen.
Nachweis	

Tabelle 5.6.: Nachweis: Lastgenerator kann Frames mit konstanter Rate erzeugen

5.1.2.2. Zufällige zeitliche Verteilung

Soll	Der Lastgenerator generiert für die Simulation ein Verkehrsaufkommen mit einer zufälligen zeitlichen Verteilung der Frames.
Ist	Es kann ein Verkehrsaufkommen generiert werden, bei dem die Zeitabstände zwischen den Frames zufällig gewählt wird. Zudem kann man bestimmen, innerhalb welchem Bereich der Zeitabstand zufällig gewählt werden soll (z.B. mindestens 100ns und maximal 500ns).
Nachweis	

Tabelle 5.7.: Nachweis: Lastgenerator kann Frames mit zufülliger zeitlicher Verteilung erzeugen

5.1.2.3. Spontane Einzelmeldungen

Soll	Der Lastgenerator generiert für die Simulation ein Verkehrsaufkommen mit spontanen Einzelmeldungen.
Ist	Spontane Einzelmeldungen in Form von Express-Frames können generiert werden. Diese werden per Zufall versandt.
Nachweis	

Tabelle 5.8.: Nachweis: Lastgenerator kann spontane Einzelmeldungen versenden

5.1.3. Durchführbarkeit der Simulationen und Interpretation der Resultate

Soll	Das Zeitverhalten der verschiedenen Weiterleitungsvarianten soll durch entsprechende Simulationsläufe ermittelt werden. Die Resultate sind zu vergleichen und zu interpretieren.
Ist	Verschiedene Weiterleitungsvarianten können simuliert und analysiert werden. Statistiken können zudem generiert werden, was den Vergleich der Varianten untereinander vereinfacht.
Nachweis	

Tabelle 5.9.: Nachweis: Simulationen durchführ- und interpretierbar

Teil III.

Verzeichnisse

6. Literaturverzeichnis

[1] Gemperli, Alfred: *Vertiefungsarbeit: HSR Simulation mit OMNeT++*. Technischer Bericht, Institute of Embedded Systems, ZHAW School of Engineering, Juli 2011.

[2] IEEE 802.3 DMLT Study Group: *Distinguished Minimum Latency Traffic in a Converged Traffic Environment (DMLT) @http://www.ieee802.org/3/DMLT/*, Dezember 2013.

[3] IEEE Computer Society, LAN/MAN Standards Committee of the: *IEEE P802.3br/D1.0 Draft Standard for Ethernet Amendment: Specification and Management Parameters for Interspersing Express Traffic*, Dezember 2014.

[4] IEEE P802.3br IET Task Force: *Interspersing Express Traffic (IET) Project Authorization Request (PAR) @http://www.ieee802.org/3/br/P802d3br_PAR.pdf*, September 2013.

[5] IEEE P802.3br IET Task Force: *Interspersing Express Traffic (IET) Baseline @http://www.ieee802.org/3/br/Baseline/8023-IET-TF-1405_Winkel-iet-Baseline-r3.pdf*, Juni 2014.

[6] IEEE P802.3br IET Task Force: *Interspersing Express Traffic (IET) Schedule @http://www.ieee802.org/3/br/8023-IET-TF-1401_IET_schedule.pdf*, Januar 2014.

[7] Institute of Embedded Systems: *HSR-Konzept @http://ines.zhaw.ch/de/engineering/institute-zentren/ines/forschung-und-entwicklung/high-availability/hsr.html*, September 2014.

[8] Kirrmann,Hubert: *HSR – High Availability Seamless Redundancy @http://lamspeople.epfl.ch/kirrmann/Pubs/IEC_62439/IEC_61439-3/IEC_62439-3.5_HSR_Kirrmann.ppt*, August 2010.

[9] NetModule AG: *Applying PRP and HSR Protocol for Redundant Industrial Ethernet @http://www.netmodule.com/en/technologies/interfaces_networks/IEC62439*, September 2014.

[10] OMNeT++ Community: *INET Framework for OMNeT++ Manual @http://inet.omnetpp.org/doc/INET/inet-manual-draft.pdf*, Juni 2012.

[11] OMNeT++ Community: *OMNeT++ User Manual @http://www.omnetpp.org/doc/omnetpp/manual/usman.html*, November 2014.

[12] OMNeT++ Community: *OMNeT++ Wiki - Setting up Parallel Simulations* @*http://www.omnetpp.org/pmwiki/index.php?n=Main.SettingUpParallelDistributedSimulations*, November 2014.

[13] Weibel, Hans: *PA14_wlan_1: Projektarbeit im Fachgebiet Kommunikation*. Aufgabenstellung, September 2014.

7. Glossar

HSR High-availability Seamless Redundancy. Redundanzprotokoll für Ethernet basierte Netzwerke. HSR ist für redundant gekoppelte Ringtopologien ausgelegt. Die Datenübermittlung innerhalb eines HSR-Rings ist im Fehlerfall gewährleistet, wenn eine Netzwerkschnittstelle ausfallen sollte.

IET Interspersed Express Traffic. Erlaubt es Frames während des Sendevorgangs zu unterbrechen, um sogenannte Express Frames so schnell wie möglich versenden zu können.

IFG Interframe Gap. Minimaler zeitlicher Abstand zwischen zwei Frames.

PRP Parallel Redundancy Protocol. Hat ein ähnliches Funktionsprinzip wie HSR, aber auch Unterschiede wie z.B. das Frameformat. PRP- und HSR-Netze können jedoch über RedBoxen kommunizieren.

8. Abbildungsverzeichnis

9. Tabellenverzeichnis

10. Listingverzeichnis

Teil IV.

Anhang

11. Offizielle Aufgabenstellung

1 Ausgangslage In Anlagen für die Automatisierung der elektrischen Energieversorg hat sich Ethernet gut etabliert. Ein Anwendungsfeld ist jedoch noch mit Unsicherheiten behaftet: der Prozessbus von Unterstationen. Bei dieser Anwendung werden extrem viele Messdaten erfasst und übertragen. Gleichzeitig soll das Netzwerk Steuerbefehle (z.B. für Notabschaltung) mit sehr geringer Verzögerung übertragen können.

Um höchste Verfügbarkeit zu garantieren wird das Ethernet in einer Ringtopologie betrieben. Das Redundanzverfahren heisst HSR (High-availability Seamless Redundancy) und arbeitet verlustfrei, d.h. es übersteht den Ausfall einer Komponente oder eines Links, ohne dass Frames verloren gehen.

Es gibt verschiedene Ansätze, die Verzögerung kritischer Frames zu garantieren.

a) Die Erhöhung der Datenrate (in diesem Fall von 100 MBit/s auf 1 GBit/s) ist naheliegend. Damit kann das Problem aber nicht prinzipiell gelöst, sondern lediglich auf ein anderes Niveau verschoben werden. Diesen "Brute Force"- Ansatz möchte man wegen den damit verbundenen sehr viel höheren Anforderungen an die Hardware wenn möglich vermeiden und stattdessen lieber einen effizienten Algorithmus verwenden.

b) Wenn es um sehr zeitsensitive Anwendungen geht, hat Ethernet generell das Problem, dass ein langes Frame, dessen Aussendung schon begonnen hat, die Aussendung eines hoch priorisierten Frames verzögert. Das Zeitverhalten könnte mit einem Pre-Emption-Mechanismus verzögert werden, welcher es erlaubt, das Versenden eines langen Frames zu unterbrechen und später wieder aufzunehmen. In der Standardisierung gibt es Bestrebungen, einen solchen Mechanismus einzuführen.

c) Durch ein zeitgesteuertes Scheduling kann man Zeitfenster für kritische Kommunikation reservieren und somit Verzögerungszeiten garantieren.

2 Aufgabenstellung In der Arbeit soll untersucht werden, welchen Effekt die zur Diskussion stehenden Massnahmen für einen konkreten Anwendungsfall bringen. Das umfasst folgende Tätigkeiten:

2.1 Modell für HSR-Knoten erweitern Das betrachtete Netzwerk ist ein HSR-Ring. Die bestehende Simulationsumgebung soll so erweitert bzw. angepasst werden, dass folgende Funktionen/Mechanismen simuliert werden können:

a) Der Knoten soll zwei Prioritäten unterstützen, d.h. zwei Warteschlangen pro Interface bewirtschaften.

b) Der Knoten soll Interspersing Express Traffic (IET) unterstützen, d.h. Express Frames können die aktuell ablaufende Übertragung eines Frames unterbrechen.

c) Der in den Ring einfliessende Traffic kann limitiert werden.

d) Die Vortrittsregeln bezüglich der im Ring zirkulierenden Frames und den Frames, die in den Ring einfliessen, können variiert werden (z.B. „zirkulierende Frames haben immer Vortritt“ oder „minimaler Zufluss wird garantiert“).

e) Der Knoten implementiert ein Zeitschlitzverfahren, welches dem zeitkritischen Traffic und dem Bulk Traffic je eine Phase zuordnet.

2.2 Lastmodell beschreiben und implementieren

Das durch die Anwendung generierte Verkehrsaufkommen ist zu studieren und zu beschreiben. Lastgeneratoren sollen implementiert werden, die das Verkehrsaufkommen für die Simulation generieren durch die Überlagerung von Strömen mit folgender Charakteristik:

a) konstante Framerate

b) zufällige zeitliche Verteilung der Frames

c) spontane Einzelmeldungen

2.3 Simulationen durchführen und Resultate interpretieren

Das Zeitverhalten der verschiedenen Weiterleitungsvarianten soll durch entsprechende Simulationsläufe ermittelt werden. Die Resultate sind zu vergleichen und zu interpretieren.

3 Ziele

- Es liegt eine lauffähige und ausreichend dokumentierte Simulationsumgebung vor, welche
  - die verschiedenen Weiterleitungsvarianten implementiert,
  - Traffic unterschiedlicher Charakteristik generieren kann,
  - die Laufzeit der einzelnen Frames misst und geeignet visualisiert.
- Das zeitliche Verhalten einiger Konfigurationen ist für verschiedene Lastprofile simuliert. Die Resultate sind visualisiert, interpretiert und kommentiert.

12. Projektmanagement

12.1. Präzisierung der Aufgabenstellung

12.2. Besprechungsprotokolle

Die Besprechungsprotokolle wurden Stichwortartig in einem eigenen Wiki festgehalten. Der Inhalt dieser Protokolle lautet wie folgt:

12.2.1. Kalenderwoche 38: 17.09.2014

- Allgemeine Info, um was geht es:
  - Echtzeit und Hochverfügbarkeit
  - HSR-Ring-Netzwerk-Aufbau
  - Diverse Mechanismen (Welches Frame hat Vortritt?)
- Organisatorisches:
  - Wöchentlicher Rapport via E-Mail vor der Besprechung
  - Wöchentliche Besprechung jeden Donnerstag um 12:00
  - Nächste Besprechung fällt aus

12.2.2. Kalenderwoche 40: 02.10.2014

- Express-Priorität nicht anhand VLAN-Tag festlegen / feststellen
  - Z.B. im EtherType-Feld definieren
- IET spezifiziert, dass bereits gesendetes Fragment ankommen muss (Normales Frame muss bei einem Express-Frame unterbrochen und darf nicht verworfen werden)
  - Kein Fragment $<$ 64 Bytes (min. Size Ethernet Frame)

  - Zu Fragmentierendes Frame in richtige Teile aufsplitten

- In unserem Fall ist das Netzwerk fehlerfrei
- Express-Frames werden nicht fragmentiert
- PHY Layer soll von Ganzem nicht merken
- Jeder Switch in einem Gerät hat Scheduler

12.2.3. Kalenderwoche 41: 09.10.2014

- Anhand Zeitpunkt Frameversand und dessen Grösse ist die Dauer berechenbar
- Zeitberechnung
  - Frames müssen nicht fragmentiert werden wie in der Realität
  - Wie lange hat ein normales Frame, wann ein Express-Frame dazwischenkommt?
  - Express-Frame zu welchem Zeitpunkt senden? (Wann wäre Frame-Fragment versendet worden?)
- Frame mehrmals unterbrechbar
- Queuegrösse ist auch ein Faktor (Per INI-Datei der Simulation definierbar)

12.2.4. Kalenderwoche 42: 16.10.2014

- Traffic-Pattern für Testing (wird noch gesandt)
  - Mix konstanter Bitraten (Erzeugt Frame mit bestimmten Grösse jeden bestimmten Zeitabstand, es können sich auch mehrere überlagern)
  - Random (Background-Traffic)
  - Ab und zu Express-Frame (Alarm)
  - Lastgenerator in Node, für jeden Knoten spezifisch definierbar
- Besprechung Zeitschlitzverfahren
  - Zeitschlitze für High und Low, Express kann immer
  - Intervall wird vereinbart
    * In Zeitschlitz muss man maximales Frame durchbringen
    * Zeitschlitz hat Grün-Phase, in der High Frames (zyklische Messwerte) gesendet werden können, Rest der Zeit für Sendevorgang oder Low

12.2.5. Kalenderwoche 43: 23.10.2014

- Gerät soll sich selber benachrichtigen, sobald neues Frame zu versenden ist
- Lastgenerator Random Prioritäten, z.B. 10% Express, 20% High, 70% Low
- Gibt viel Multicast, wie senden?
- Demnächst Simulation im kleinen Stil mit First Come First Serve zum Laufen bringen

12.2.6. Kalenderwoche 44: 30.10.2014

- Simulation läuft, aber stürzt nach 2 Frames ab
  - Beinhaltet 3 DANH-Geräte, 2 verschiedene senden zeitgleich an anderen
- Scheduler pro Port implementieren, bis jetzt ist Scheduler pro Gerät implementiert
  - Macht wenig Sinn, Switch kann auf freien Ports gleichzeitig versenden
  - Frame kommt an, wo muss es hin? Entsprechendem Scheduler zuweisen
- Express: klein und selten, konstante Bitrate
- High: normaler Verkehr (Monitoring, Netzwerkmanagement, ...)
- Low: Background-Verkehr (z.B. TCP), oft und klein & gross

12.2.7. Kalenderwoche 45: 06.11.2014

- Gedanken zu IET (Zeitberechnung):
  - Sender behält normales Frame, um zu sagen wann Express-Frame versendet werden soll
  - Empfänger verlängert «fragmentiertes Frame» um die Dauer des dazwischen versendeten Express-Frames + IFG
    - Empf. überprüft, ob Sendezeit des Express-Frames zwischen Sende- und Ankunftszeit des normalen Frames ist. Wenn ja, wird das normale Frame verlängert
  - Zweiter Kanal für Express Frame?
    - Wenn Channel busy ist, kann Express-Frame gesendet werden?
    - Zweiter Kanal nur für Express Frames

    - Keine wirkliche Fragmentierung in Simulation vornehmen, Zeitrechnung reicht aus

  - FragCount sagt nicht max. 5 Fragmente, sondern ist ein Modulo-4-Counter
  - Frame abbrechen
    - Nur wenn Rest genug lang ist
    - Express muss Fragmentlänge + IFG abwarten
    - Normal Frame Ankunftszeit um Express + IFG verlängern

12.2.8. Kalenderwoche 46: 13.11.2014

- Anwendungsfall: Substation Automation
  - Schutz von Schaltungen und Leitungen sicherstellen
  - Merging Units (MUs) erfassen Messwerte
    - 7 Messwerte (4x Spannung, 3x Strom) in einem Frame plus Zeitstempel und sonst. Steuerinfos
      - High-Priority Frame von total 160 Bytes
      - Konstant 4000x pro Sekunde pro MU
    - Spontane Einzelmeldungen (Express-Frame, selten und kurz, ca. 100 Bytes)
      - Wird zufällig mit bestimmter Wahrscheinlichkeit versendet, z.B. bei 20% Wahrscheinlichkeit wenn Randomwert von 0 bis 1 zwischen 0 und 0.2 liegt
    - Frames werden via Multicast verteilt (Publisher / Subscriber Modell)
    - Ziel sind Protection Units (PUs), welche anhand der erhaltenen Werte Entscheidungen treffen und doppelt vorhanden sind
    - Max. Anzahl an MUs: 19 Stk., schauen wie es z.B. mit 10 ist
  - Background-Traffic (TCP, nicht wirklich simulierbar, Unicast, meistens mit Gerät ausserhalb Ring)
    - TCP-ähnlichen Traffic in 2 Knoten generieren
      - 200 Frames à 1500 Bytes pro Sekunde von A nach B
      - 200 Frames à 64 Bytes pro Sekunde von B nach A

- Was Intern generiert wird wird gleich behandelt wie das was von Aussen kommt
- Queue-Limit spielt keine allzu grosse Rolle
  - Wie lange ein Frame warten muss oder wie viele Frames verloren gehen sind äquivalente Aussagen
- Man muss am Schluss etwas zur Delay-Charakteristik von Strömen und Express-Frames sagen können
  - Wann generiert und wann Ankunft? Differenz ist Übermittlungszeit
  - Ankunftszeit dort feststellen, wo das Frame vom Netz entfernt wird
  - Nicht besseren, sondern schlechteren Fall anschauen
    - Duplikat anschauen (Was als 2tes ankommt)
    - Best-/Worst-Case spielt nur bei Unicast eine Rolle, bei Multi-/Broadcast macht es wahrscheinlich keinen grossen Unterschied
- Duplikaterkennung bei allen Ports implementieren
  - Kein Port soll dasselbe Frame mehrmals versenden

12.2.9. Kalenderwoche 47: 20.11.2014

- Testfälle für Überprüfung der Implementation
  - Nicht fragmentierbare Frames
  - Versenden eines langen, normalen Frames und 2 darauf folgende Express-Frames

12.2.10. Kalenderwoche 48: 27.11.2014

- Start der Simulation für Belastung nicht sehr interessant, ab dem Zustand nach der «Initialisierung» interessant
  - Ab «Ready State» aufnehmen
- Testing
  - Mit Multicast-Frames testen, weil diese beim Sender gelöscht werden (Bei einem Defekt würde die Dauer dieses Frames etwa der Dauer eines normalen Frames entsprechen

  - Express-Frames mit Multicast versenden
  - Mit 17 MUs (siehe Kapitel 12.2.8 auf Seite 61), die permanent Messwerte via Multicast versenden sollte so guet wie kein Platz mehr auf dem Ethernet frei sein.
  - Interval, in dem die Messwerte versendet werden, variiert von MU zu MU (Phase beginnt nicht überall zur exakt selben Zeit)
    * Beim Interval eine Zufallszeit dazu- oder abzählen, sodass der Interval nicht konstant ist
    * Delayunterschied zu Fall, an dem die MUs gleichzeitig und mit exakt selbem Interval senden untersuchen
  - Fall: 14, 15 MUs
    * Einfluss TCP (High-Frame wird angestaut, wenn gerade ein Low-Frame gesendet wird)
    * Was wenn TCP Löcuken zwischen High-Frames schamlos nutzt?
      · Zufluss TCP limitieren?
  - 100Mbps für die Verbindung verwenden
    * Untersuchen wie viele Frames pro Sekunde noch übertragbar sind (bevor sich die Queues nicht mehr leeren). Wie nahe kann man an diese Grenze gehen?
    * Frame mit Messwerten beträgt 160 Bytes: ca. 69444 solcher Frames über 100Mbps pro Sekunde
      $\frac{10^8}{(160+Präamble+IFG)*8} = \frac{10^8}{180*8} = 69444.\overline{4}$

- Visualisierung der Resultate
  - Histogramm (Wie viele haben Transmissiontime x?)
    * Klassenbreite (Transmissiontime auf ms/us genau)
  - Gesamtkapazität < Summe von allem MU- plus TCP-Traffic
  - Verifizierung Verhalten mit Frameverlauf in Simulation

- Anleitung
  - Ab CD wie starten
    * Möglich: Exe-Datei starten und Config editieren (VM auf CD?)
  - Wie Simulation laufen lassen
  - Wie Konfigurieren (in XML- und INI-Datei)
  - Wie bestimmter Fall simulieren

12.2.11. Kalenderwoche 49: 04.12.2014

- Was bedeuten in der Simulation die verschiedenen Farben der Verbindungen?
  - In Anleitung festhalten
- Interner Delay (im Knoten) von 6us implementieren
  - Zeit bis der Teil des Frames gelesen werde, um entscheiden zu können, was damit gemacht wird (Cut-Through-Switching)
- Für die Verbindung zwischen den Endknoten 20m 100Mbps Kabel verwenden
  - Delay von 5ns pro 1m Kabel
  - Delay 20m Kabel: 100ns
- Zuflusslimitierung bei TCP-ähnlichem Traffic auf z.B. 1Mbit/s setzen
- Notieren, wo Aufgabenstellung in welchem Code erfüllt wurde und wie man zu diesem Modul gekommen ist
- Testfälle
  - Nur ein Frame auf leeren Ring senden -> schnellster Übertragungswert
  - Frame bei t1, Express bei t2 (bisschen später als t1) versenden -> Wie sieht es mit der Preemption aus?
    - Was wenn das Frame nicht fragmentierbar ist?
    - Was wenn es fragmentierbar ist?
    - Kann ein langes Frame von mehreren Express-Frames unterbrochen werden?

- Offizielle Aufgabenstellung, Projektauftrag
- (Zeitplan)
- (Besprechungsprotokolle oder Journals)

13. Anleitung zur Simulationsumgebung und -Durchführung

13.1. Starten der virtuellen Maschine

Auf dem mitgelieferten USB-Stick befindet sich eine VDI-Datei. Diese Datei beinhaltet eine virtuelle Maschine (VM), auf der das 32Bit Linux-Betriebssystem Debian inklusive der OMNeT++-Umgebung und einer englischen Anleitung eingerichtet ist. Eine deutsche Anleitung wird in Kapitel 13 aufgeführt.

Damit diese virtuelle Maschine ausgeführt werden kann wird die Software VirtualBox benötigt. Diese kann unter `www.virtualbox.org` heruntergeladen werden.

Um die VDI-Datei einzubinden erstellt man in VirtualBox eine neue virtuelle Maschine und navigiert durch den Assistenten (Typ: «Linux», Version: «Debian (32 bit)»). Die virtuelle Maschine wurde mit einem zugewiesenen Arbeitsspeicher von 512MB getestet (es wird empfohlen 512MB oder mehr als Arbeitsspeicher auszuwählen). Anstelle einer neuen Festplatte zu erstellen wird nun die bestehende VDI-Datei eingebunden. Wurde die virtuelle Maschine erstellt, kann sie gestartet werden.

Für den Betrieb der virtuellen Maschine sind keine Login-Daten notwendig. Es wird automatisch eine Desktop-Umgebung gestartet, die in etwa folgendermassen aussieht (Desktop-Icons können variieren):

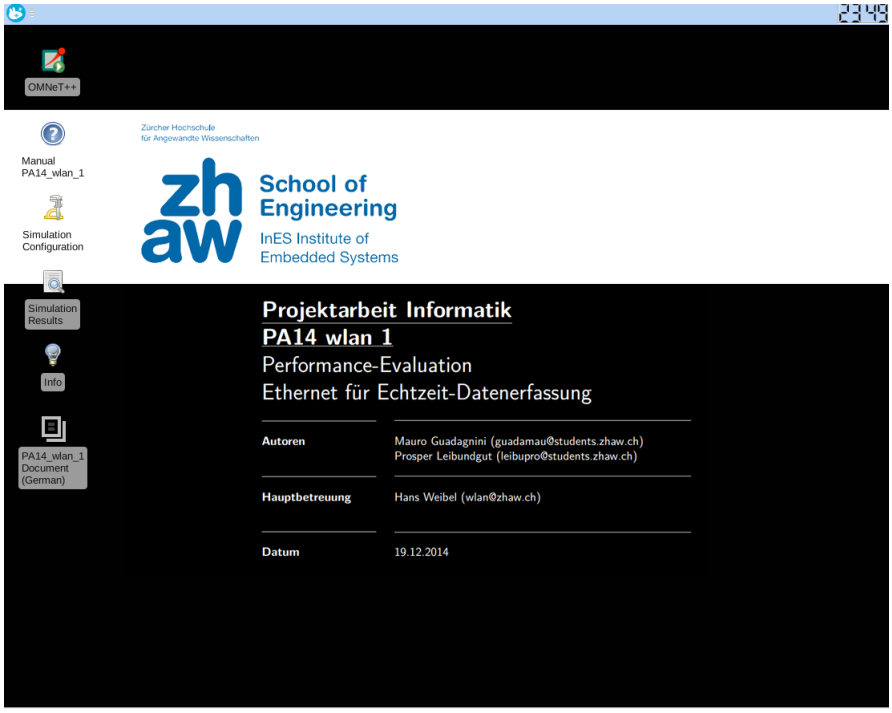

Abbildung 13.1.: Screenshot der gerade gestarteten virtuellen Maschine

Auf dem Desktop und dem Startmenü (links oben im Screenshot) werden alle benötigten Verknüpfungen zur Verfügung gestellt. Im geöffneten Startmenü gibt es rechts unten die Möglichkeit, die virtuelle Maschine herunter zu fahren und des Weiteren auch die Möglichkeit, nach Applikationen / Verknüpfungen zu suchen.

Sollte trotzdem ein Login benötigt werden, ist dieser in der VM unter dem Punkt «Info» zu finden.

13.2. Simulation konfigurieren

13.3. Simulation starten

13.4. Resultate ansehen

14. USB-Stick

Dieser Arbeit wird ein USB-Stick mit folgenden Eigenschaften mitgegeben:

Marke & Modell	SanDisk Extreme USB 3.0 Flash Drive
Kapazität	16 GB
Name	PA14_wlan_1
Dateisystem	NTFS

Tabelle 14.1.: USB-Stick Eigenschaften

Auf dem USB-Stick befinden sich folgende Dateien:

(Tree of USB-Stick content)

15. Codeausschnitte

15.1. Switch: Zuteiltung Frame von CPU an beide Ports nach Aussen

```
1  ...
2  void EndNodeSwitch::handleMessage( cMessage* msg ) {
3  ...
4   /* Schedulers */
5      Scheduler* schedGateAOut = HsrSwitch::getSchedGateAOut();
6      Scheduler* schedGateBOut = HsrSwitch::getSchedGateBOut();
7      Scheduler* schedGateCpuOut = HsrSwitch::getSchedGateCpuOut();
8  ...
9      /* Gates */
10     cGate* gateAIn = HsrSwitch::getGateAIn();
11     cGate* gateBIn = HsrSwitch::getGateBIn();
12     cGate* gateCpuIn = HsrSwitch::getGateCpuIn();
13
14     cGate* gateAInExp = HsrSwitch::getGateAInExp();
15     cGate* gateBInExp = HsrSwitch::getGateBInExp();
16     cGate* gateCpuInExp = HsrSwitch::getGateCpuInExp();
17 ...
18     /* Make a clone of the frame and take ownership.
19      * Then we have to downcast the Message as an ethernet frame. */
20     cMessage* switchesMsg = msg;
21     this->take( switchesMsg );
22 ...
23     /* Arrival Gate */
24     cGate* arrivalGate = switchesMsg->getArrivalGate();
25
26     /* Switch mac address. */
27     MACAddress switchMacAddress = *( HsrSwitch::getMacAddress() );
28
29     EthernetIIFrame* ethernetFrame = check_and_cast<EthernetIIFrame*>(
           switchesMsg );
30
31     /* Source and destination mac addresses */
32     MACAddress frameDestination = ethernetFrame->getDest();
33     MACAddress frameSource = ethernetFrame->getSrc();
34
35     EthernetIIFrame* ethTag = NULL;
36     vlanMessage* vlanTag = NULL;
37     hsrMessage* hsrTag = NULL;
38     dataMessage* messageData = NULL;
39
40     EthernetIIFrame* frameToDeliver = NULL;
41     EthernetIIFrame* frameToDeliverClone = NULL;
42
43     framePriority frameprio = LOW;
44
45     MessagePacker::decapsulateMessage( &ethernetFrame, &vlanTag, &hsrTag, &
           messageData );
46
47     /* After decapsulating the whole ethernet frame becomes a ethernet tag
48      * (only the header of an ethernet frame)
49      * ethTag variable just there for a better understanding. */
50     ethTag = ethernetFrame;
```

```
51
52      /* determine package prio and set enum */
53      if( vlanTag != NULL )
54      {
55          if( vlanTag->getUser_priority() == EXPRESS )
56          {
57              frameprio = EXPRESS;
58          }
59          else if( vlanTag->getUser_priority() == HIGH )
60          {
61              frameprio = HIGH;
62          }
63      }
64  ...
65          if( arrivalGate == gateCpuIn || arrivalGate == gateCpuInExp )
66          {
67              frameToDeliver = MessagePacker::generateEthMessage( ethTag, vlanTag,
                    hsrTag, messageData );
68              frameToDeliverClone = frameToDeliver->dup();
69
70              switch( frameprio )
71              {
72                  case EXPRESS:
73                  {
74                      schedGateAOut->enqueueMessage( frameToDeliver,
                            EXPRESS_INTERNAL );
75                      schedGateBOut->enqueueMessage( frameToDeliverClone,
                            EXPRESS_INTERNAL );
76                      break;
77                  }
78                  case HIGH:
79                  {
80                      schedGateAOut->enqueueMessage( frameToDeliver, HIGH_INTERNAL
                            );
81                      schedGateBOut->enqueueMessage( frameToDeliverClone,
                            HIGH_INTERNAL );
82                      break;
83                  }
84                  default:
85                  {
86                      schedGateAOut->enqueueMessage( frameToDeliver, LOW_INTERNAL
                            );
87                      schedGateBOut->enqueueMessage( frameToDeliverClone,
                            LOW_INTERNAL );
88                      break;
89                  }
90              }
91              scheduleProcessQueues('A');
92              scheduleProcessQueues('B');
93          }
94  ...
95  }
96  ...
```

Listing 15.1: switch/EndNodeSwitch.cc - Zuteiltung Frame von CPU an beide Ports nach Aussen

15.2. Scheduler: Mechanismen zur Queueverwaltung

```
1   ...
2   void Scheduler::processQueues( void )
3   {
4       queueName* currentSortOrder;
5
6
7       switch( schedmode )
8       {
```

```
9           /*
10           * FCFS: we have no distinction between Ring- and Internal frames.
11           * See the function above "enqueueMessage". Frames ordered to the ring
                 queues
12           * are redirected to the internal queues.
13           * Due to the above description, the behaviour FCFS and RING_FIRST have
                 not to
14           * be distinguished here.
15           *
16           * The Ring queues of each level are processed first,
17           * which is part of the RING_FIRST policy.
18           *
19           * In case of FCFS-scheduling-mode the ring queues are just always empty
                 ,
20           * see enqueueMessage-function.
21           *
22           * The processing order of the queues is as follows:
23           *
24           * EXPRESS_RING, EXPRESS_INTERNAL, HIGH_RING, HIGH_INTERNAL, LOW_RING,
                 LOW_INTERNAL
25           * */
26          case FCFS:
27          case RING_FIRST:
28          {
29              currentSortOrder = ringFirstSortOrder;
30              loopQueues( currentSortOrder );
31              break;
32          }
33
34          /*
35           * The Internal queues of each level are processed first,
36           * which is part of the INTERNAL_FIRST policy.
37           */
38          case INTERNAL_FIRST:
39          {
40              currentSortOrder = internalFirstSortOrder;
41              loopQueues( currentSortOrder );
42              break;
43          }
44
45          /*
46           * ZIPPER-Policy
47           * Alternates the queues between ring and internal.
48           *
49           * Processing order of the queues:
50           *
51           * [2n] even
52           * EXPRESS_RING, EXPRESS_INTERNAL, HIGH_RING, HIGH_INTERNAL, LOW_RING,
                 LOW_INTERNAL
53           *
54           * [2n-1] odd
55           * EXPRESS_INTERNAL, EXPRESS_RING, HIGH_INTERNAL, HIGH_RING,
                 LOW_INTERNAL, LOW_RING
56           * */
57          case ZIPPER:
58          {
59              switch( curQueueState )
60              {
61                  case RING:
62                  {
63                      currentSortOrder = ringFirstSortOrder;
64                      loopQueues( currentSortOrder );
65                      curQueueState = INTERNAL;
66                      break;
67                  }
68                  case INTERNAL:
69                  {
70                      currentSortOrder = internalFirstSortOrder;
71                      loopQueues( currentSortOrder );
```

```
72                      curQueueState = RING;
73                      break;
74                  }
75                  default:
76                  {
77                      break;
78                  }
79              }
80              /* break to case ZIPPER */
81              break;
82          }
83  ...
84      }
85  }
86
87  ...
88
89  void Scheduler::loopQueues( queueName* currentSortOrder )
90  {
91      cQueue* currentQueue = NULL;
92      queueName currentQueueName;
93
94      for( unsigned char i = 0; i < QUEUES_COUNT; i++ )
95      {
96          currentQueueName = currentSortOrder[ i ];
97          currentQueue = check_and_cast<cQueue*>( queues->get( currentQueueName )
                ;
98
99          if( currentQueue != NULL )
100         {
101             processOneQueue( currentQueue, currentQueueName );
102         }
103         else
104         {
105             throw cRuntimeError( " [ PANIC ] : current queue to process is NULL!
                    exiting ... \n" );
106         }
107     }
108 }
109 ...
```

Listing 15.2: switch/Scheduler.cc - Mechanismen zur Queueverwaltung

16. Weiteres

- CD mit dem vollständigen Bericht als pdf-File inklusive Film- und Fotomaterial
- (Schaltpläne und Ablaufschemata)
- (Spezifikationen u. Datenblätter der verwendeten Messgeräte und/oder Komponenten)
- (Berechnungen, Messwerte, Simulationsresultate)
- (Stoffdaten)
- (Fehlerrechnungen mit Messunsicherheiten)
- (Grafische Darstellungen, Fotos)
- (Datenträger mit weiteren Daten (z. B. Software-Komponenten) inkl. Verzeichnis der auf diesem Datenträger abgelegten Dateien)
- (Softwarecode)