

2 Grundlagen

2.1 Echtzeitdatenübertragung und Industrial Ethernet

2.1.1 Klassifizierung von Echtzeitkommunikationssystemen

Der Begriff „Echtzeit“ wird in unterschiedlichen Themenbereichen mit zum Teil sehr unterschiedlicher Bedeutung verwendet. Im Zusammenhang mit eingebetteten Systemen und der Datenübertragung in Automatisierungsumgebungen spricht man davon, dass ein System echtzeitfähig ist, wenn es bestimmte Aufgaben zuverlässig vor dem Erreichen einer vorgegebenen Zeitschranke (Deadline) abschließen kann. Bei der rechtzeitig abzuschließenden Aufgabe kann es sich zum Beispiel um eine lokal ausgeführte Berechnung oder um eine Datenübertragung über ein Netzwerk handeln.

Mit der Unterscheidung zwischen weichen und harten Echtzeitanforderungen einer Aufgabe werden die Folgen bei Nichteinhaltung von Deadlines beschrieben [2]:

- Bei Aufgaben mit **harten** Echtzeitanforderungen kann die Nichteinhaltung von Deadlines katastrophale Folgen haben, wie z.B. eine Beschädigung der Anlage oder eine Gefährdung von Personen.
- Bei Aufgaben mit **weichen** Echtzeitanforderungen führt die Nichteinhaltung von Deadlines nur zu einem geringeren Nutzen des erzeugten Ergebnisses. Das Gesamtsystem kann seine Arbeit in der Regel mit verringerter Leistung fortsetzen.

Auf vergleichbare Weise kann die Echtzeitfähigkeit von Systemen beschrieben werden. Weiche Echtzeitfähigkeit bedeutet, dass das System Deadlines in der Regel erfüllen kann, Ausnahmen sind jedoch möglich. Dementsprechend kann es dann nur für Aufgaben mit weichen Echtzeitanforderungen genutzt werden. Ein System mit harter Echtzeitfähigkeit kann die Einhaltung von Deadlines garantieren und ist daher für Aufgaben mit harten Echtzeitanforderungen einsetzbar. Wird in dieser Arbeit von Echtzeitfähigkeit gesprochen, so ist harte Echtzeitfähigkeit gemeint, falls nicht explizit etwas anderes genannt wird.

Im Bereich der Automatisierungsumgebungen wird die echtzeitfähige Gerätevernetzung in den meisten Fällen zur Übertragung von Sensor- und Steuerungsdaten benötigt, die regelmäßig aktualisiert werden müssen. Dementsprechend wiederholt sich das gleiche Kommunikationsmuster zwischen den Geräten immer wieder und man spricht daher von einer zyklischen Kommunikation. Die Zeit, nach der sich das Kommunikationsmuster wiederholt, wird als Zykluszeit bezeichnet. Je geringer die Zykluszeit ist, desto häufiger können Sensor- und Steuerungsdaten aktualisiert werden. Anspruchsvolle Anwendungen erfordern eine sehr häufige Aktualisierung von Sensor- und Steuerungsdaten und benötigen somit eine geringe Zykluszeit. Das Problem hierbei ist, dass die Zykluszeit nicht beliebig klein gewählt werden kann, da ein Zyklus ausreichend lang sein muss, um die Übertragung aller pro Zyklus zu sendenden Daten mit den zur Verfügung stehenden Netzwerkressourcen abzuschließen. Ein System zur echtzeitfähigen Datenübertragung zeichnet sich daher durch eine effiziente Verwaltung der zur Verfügung stehenden Netzwerkressourcen aus, sodass auch Anwendungen mit hohen Anforderungen an die Zykluszeit betrieben werden können. Echtzeitkommunikationssysteme werden daher oft nach der minimal erreichbaren Zykluszeit in Klassen eingeteilt [1]:

- Klasse 1: Weiche Echtzeitfähigkeit, Zykluszeiten im Bereich von 100 ms
- Klasse 2: Harte Echtzeitfähigkeit, Zykluszeiten zwischen 1 ms und 10 ms
- Klasse 3: Isochrone Echtzeitfähigkeit, Zykluszeiten zwischen 250 μ s und 1 ms mit Jitter unterhalb von 1 μ s

2.1.2 Industrial Ethernet

Eine bereits seit vielen Jahrzehnten etablierte Möglichkeit zur echtzeitfähigen Gerätevernetzung in industriellen Anwendungen stellen die Feldbusse dar. Im Bereich der Feldbusse existiert eine Vielzahl von Systemen unterschiedlicher Hersteller, die sich in der Vergangenheit als zuverlässig erwiesen haben und mit denen auf Grund des langjährigen Einsatzes umfangreiche Erfahrungen vorliegen. Daher werden Feldbusse auch heute noch zur Fabrikvernetzung eingesetzt. Seit einigen Jahren zeigen sich jedoch verschiedene Schwächen dieser Technologien. Sie sind in der Regel weder untereinander noch zu modernen Internettechnologien kompatibel. Des Weiteren ist die Anzahl der vernetzbaren Geräte beschränkt, was ein Problem für die Skalierbarkeit des Netzwerks darstellt.

Daher haben sich mittlerweile verschiedene auf Ethernet basierende Systeme zur echtzeitfähigen Vernetzung entwickelt, die sogenannten Industrial-Ethernet-Systeme. Zu den bedeutendsten Zielen der Industrial-Ethernet-Systeme zählt die Integration von Geräten auf Feldebene (d.h. beispielsweise Geräte in einer Fertigungsanlage) in die IT-Infrastruktur des Unternehmens. So sollen die PCs aus dem Büronetzwerk mit den Geräten der Feldebene kommunizieren können, um beispielsweise Statusinformationen abzufragen oder Änderungen an der Konfiguration vorzunehmen. Durch die bessere

Kompatibilität zwischen den gleichermaßen auf Ethernet basierenden Büronetzwerken und Industrial-Ethernet-Systemen sollen dabei im Vergleich zum Einsatz von Feldbussen keine oder wesentlich weniger aufwendige Gateways zwischen Büronetzwerk und Fabriknetzwerk notwendig sein. Die nicht zeitkritischen IT-Daten werden dabei über das Industrial Ethernet übertragen, ohne dass die zuverlässige Übertragung von zeitkritischen Prozessdaten negativ beeinflusst wird. Ein weiterer Vorteil der Industrial-Ethernet-Systeme ist es, dass bei Konzept und Implementierung der Systeme auf viele bereits durch IEEE 802.3 (Ethernet) standardisierte Komponenten zurückgegriffen werden kann. Dies reicht über die Verwendung von standardisierten Kommunikationsmedien und Steckerverbindungen bis hin zur Integration etablierter Protokolle aus dem Bereich der Internettechnologien (IP, TCP, UDP). Zur Übertragung nicht zeitkritischer IT-Daten können beispielsweise TCP/IP-Verbindungen oder UDP/IP-Pakete verwendet werden, während nur für die Übertragung von Prozessdaten die für die jeweilige Anwendung üblichen, speziellen Protokolle eingesetzt werden. Damit die nicht zeitkritischen Daten keine negativen Auswirkungen auf die Echtzeitdatenübertragung haben, kann entweder eine strikte Priorisierung der zeitkritischen Daten erfolgen, oder die Medienzugriffsregelung des IE-Systems steuert die Übertragung nicht zeitkritischer Daten ebenso wie die Übertragung zeitkritischer Daten. Da dieser Vorgang in der Regel transparent für die Anwendung ist, bleibt die Kompatibilität zu IT-Anwendungen dabei erhalten. Aus der Verwendung von durch IEEE 802.3 standardisierten Komponenten resultieren auch Kostenvorteile, wie z.B. durch die Verwendung von bereits existierender Ethernet-Hardware. Außerdem bieten die zur Verfügung stehenden Ethernet-Technologien wesentlich größere Bandbreiten, als typischerweise durch Feldbusse zur Verfügung gestellt werden. Zusammen mit einer effizienten Verwaltung dieser Netzwerkreisourcen durch das verwendete Industrial-Ethernet-System soll dies zukünftig die Skalierbarkeit verbessern. So sollen mehr Geräte über ein einziges, einheitliches Netzwerk kommunizieren können und auch komplexere Kommunikationsvorgänge ermöglicht werden, die den Austausch von größeren Datenmengen erfordern.

Industrial-Ethernet-Systeme sind vor allem für die Regelung des Zugriffs auf das Übertragungsmedium verantwortlich, da die Medienzuteilung nach den standardmäßig für Ethernet vorgesehenen Mechanismen keine echtzeitfähige Datenübertragung ermöglicht. Beim historisch vorgesehenen CSMA/CD-Verfahren (Carrier Sense Multiple Access with Collision Detection) können mehrere Netzwerkteilnehmer gleichzeitig auf der gleichen Leitung senden, sodass es zu Kollisionen kommt und keine der Datenübertragungen erfolgreich abgeschlossen werden kann. In diesem Fall ist es vorgesehen, dass die Datenübertragung nach einer zufälligen Wartezeit erneut versucht wird. Dies führt zu einer unzuverlässigen Datenübertragung mit unvorhersehbarer Zustellzeit, sodass die Einhaltung von Zeitschranken nicht garantiert werden kann. Beim Einsatz von Full-Duplex-Ethernet mit Switches werden derartige Verzögerungen durch Kollisionen zwar

vermieden, allerdings kann es beim unregelmäßigen Netzwerkzugriff der Netzwerkteilnehmer zur Überlastung von einzelnen Verbindungen des Netzwerks kommen, wodurch ebenfalls unerwartete Verzögerungen und Paketverluste auftreten können. Senden beispielsweise mehrere Teilnehmer zur gleichen Zeit Daten an das gleiche Ziel, so treffen die Daten in einem Switch vor dem Ziel aufeinander. Dieser speichert die eintreffenden Pakete der verschiedenen Sender zwischen, da nicht alle Pakete gleichzeitig zum Ziel weitergeleitet werden können. Die zwischengespeicherten Pakete werden der Reihe nach weitergeleitet, wodurch später gesendete Pakete unerwünschte Verzögerungen aufweisen. In Extremfällen kann der Zwischenspeicher des Switches vollständig gefüllt sein und weitere eintreffende Pakete werden verworfen. Bestimmte Switches weisen Funktionen zur Priorisierung von ausgewählten Paketen auf, wodurch die Zuverlässigkeit bestimmter Verbindungen im Netzwerk erhöht werden kann. Zur Erfüllung harter Echtzeitanforderungen sind diese Maßnahmen jedoch in der Regel nicht ausreichend.

Eine bessere Regelung des Medienzugriffs zur Vermeidung solcher Probleme kann von den Industrial-Ethernet-Systemen durch Veränderungen auf verschiedenen Schichten des bekannten ISO/OSI-Referenzmodells erfolgen. Eine Möglichkeit ist die Anpassung der Hardware, um die für Ethernet üblichen Medienzugriffsverfahren zu ersetzen. Diese Hardwareanpassung kann in den Endgeräten und/oder den Switches erfolgen. Eine Alternative zu Hardwareanpassungen stellen in Software implementierte Verfahren zur Regelung des Medienzugriffs dar. Diese sind den Schichten oberhalb von Ethernet (Schicht 1 und 2) zuzuordnen. Diese Unterscheidung ist von großer Bedeutung, da hiermit Kosten und Flexibilität der Lösung beeinflusst werden. Sofern die Lösung spezielle Hardware benötigt, kann nicht auf weitverbreitete und somit kostengünstige Standard-Ethernet-Hardware zurückgegriffen werden. Die Flexibilität wird eingeschränkt, da Verbesserung und Weiterentwicklung des Systems, ebenso wie der Wechsel zwischen verschiedenen Systemen, unter Umständen mit einem aufwändigen Hardwaretausch verbunden ist. Softwarelösungen sind daher vorzuziehen, sofern sie die gestellten Anforderungen erfüllen können. Zusätzlich zur Klassifizierung nach der Zykluszeit sollen daher folgende zwei Kategorien zur Einteilung der Industrial-Ethernet-Lösungen verwendet werden:

- Kategorie H: Die echtzeitfähige Kommunikation erfordert spezielle Hardware
- Kategorie S: Echtzeitfähige Kommunikation wird durch spezielle Software ermöglicht, ohne dass besondere Hardware erforderlich ist

Eine weitere Unterscheidung der verschiedenen Industrial-Ethernet-Systeme besteht hinsichtlich der Protokolle, die oberhalb von Ethernet bei der Datenübertragung eingesetzt werden können. Die Prozessdaten können entweder in einem speziellen angepassten Protokoll direkt im Ethernet Frame übertragen werden, oder innerhalb von UDP/IP-Paketen und TCP/IP-Verbindungen. Der Vorteil speziell angepasster Proto-

kolle ist ein verringerter Overhead und eine effiziente Verarbeitung durch die Netzwerkteilnehmer. UDP/IP- und TCP/IP-basierte Datenübertragung kann dagegen verwendet werden um Daten über Subnetze hinweg zu übertragen und die optimale Kompatibilität zu IT-Dienste zu gewährleisten. Diese Möglichkeiten sind bei den speziell angepassten Protokollen in der Regel nicht gegeben. Die Echtzeitfähigkeit der Übertragung wird dabei normalerweise nur innerhalb eines Subnetzes gewährleistet. TCP-Verbindungen weisen allgemein schlechte Eigenschaften hinsichtlich der Echtzeitfähigkeit auf und werden daher typischerweise primär für nicht zeitkritische IT-Dienste eingesetzt, während UDP/IP-Pakete innerhalb eines Subnetzes ebenso wie speziell angepasste Protokolle zur echtzeitfähigen Kommunikation geeignet sind. Im Optimalfall sollte ein Echtzeitkommunikationssystem alle drei der genannten Möglichkeiten unterstützen. Während UDP/IP und TCP/IP die Integration des Systems in eine bestehende IT-Infrastruktur erleichtern, bieten speziell angepasste Protokolle zur Übertragung von Prozessdaten die beste Effizienz bei der Echtzeitdatenübertragung.

Weitere interessante Aspekte der verschiedenen Industrial-Ethernet-Systeme sind der manuelle Konfigurationsaufwand und die Zuverlässigkeit. Im besten Fall besitzt ein Industrial-Ethernet-System umfangreiche Funktionen zur automatischen Konfiguration für verschiedene Netzwerktopologien und Anforderungen. Für eine optimale Zuverlässigkeit sollte das System keine zentrale Steuerungseinheit besitzen, deren Ausfall auch zu einem Ausfall der gesamten Echtzeitkommunikation führt. Derartige Schwachstellen werden als Single Point of Failure (SPoF) bezeichnet. Eine Übersicht über einige bekannte Industrial-Ethernet-Systeme und deren wichtigste Eigenschaften ist in Tabelle 1 gezeigt.

Zwischen der minimal möglichen Übertragungslatenz und der maximalen Geräteanzahl besteht eine gegenseitige Abhängigkeit. Weitere Abhängigkeiten dieser Werte bestehen zur verwendeten Topologie. Außerdem unterscheidet sich die den Netzwerkteilnehmern zur Verfügung stehende Datenrate zwischen den einzelnen Systemen. Angaben zu Topologien und Bandbreiten sind z.B. in [3] zu finden. Das Problem bei der Beurteilung von IE-Systemen ist das Fehlen von standardisierten Testfällen. Daraus resultiert, dass die Leistungsangaben der Systeme (z.B. in Tabelle 1) oftmals nicht direkt vergleichbar sind. Dennoch sollen die präsentierten Angaben an dieser Stelle zur groben Einordnung der Systeme ausreichen, da ein detaillierter Vergleich der Systeme im Rahmen dieser Arbeit weder möglich noch angestrebt war.

Die Übersicht zeigt, dass keins der Systeme alle wünschenswerten Eigenschaften in sich vereinen kann: Alle etablierten IE-Systeme, die die höchsten Echtzeitanforderungen der Klasse 3 erfüllen, benötigen hierfür spezielle Hardware. Einige dieser Systeme (z.B. SERCOS III, Ethernet Powerlink) können auch ohne angepasste Hardware eingesetzt werden, allerdings werden dadurch auch die minimal erreichbaren Zykluszeiten und

IE-System	Übertragungs- latenz [ms]	Klasse/ Kategorie	Beinhaltet SPoF?	Skalierbarkeit: Maximale Geräteanzahl	Funktionen zur automatischen Konfiguration?
Modbus-TCP²⁾	1-15	1/S	Nein	Unbegrenzt [4]	Ja
Ethernet Powerlink¹⁾	0.4	3/HS	Ja	4	Ja
EtherCAT¹⁾	0.15	3/H	Ja	180	Ja
TCnet¹⁾	2	2/H	Keine Angabe	24	Nein
TTEthernet	Keine Angabe	2/H	Ja	Keine Angabe	Nein
CC-Link IE Field²⁾	1.6	2/HS	Ja	254 [4]	Nein
Profinet¹⁾	1	3/HS	Ja	60	Nein
Ether- Net/IP¹⁾⁴⁾	0.13	3/H	Nein	90	Nein
SERCOS III¹⁾	0.0398	3/HS	Ja	9	Ja
DRTP³⁾	5-10	3/S	Ja	Keine Angabe	Ja
DARIEP³⁾	0.1-0.3	3/H	Ja	Keine Angabe	Nein
HaRTKad³⁾	0.7	2-3/S	Nein	Keine Angabe	Ja

Tabelle 1: Vergleich verschiedener IE-Systeme (vgl. [1] [5]). Angegebene Übertragungslatenzen entsprechen den laut Spezifikation minimal erreichbaren Latenzen. Angaben für die maximale Geräteanzahl gelten für den Fall der angegebenen minimal erreichbaren Latenz. Größere Netzwerke mit höherer Übertragungslatenz können mit den meisten Systemen ebenfalls realisiert werden. ¹⁾Originalquelle der Angaben für Latenz und Geräteanzahl ist [3] ²⁾Originalquelle zur Geräteanzahl ist [4], Originalquelle für Latenzen unbekannt ³⁾Systeme befinden sich in der Entwicklung ⁴⁾EtherNet/IP steht für EtherNet Industrial Protocol

Übertragungslatenzen verschlechtert. Auch die Kombination aus umfangreichen Funktionen zur automatischen Konfiguration und keinem SPoF kann keines der etablierten Systeme bieten. Unter den in der Entwicklung befindlichen Systemen befindet sich mit HaRTKad nur ein System, dass alle gewünschten Kriterien erfüllt. Bei den in der Entwicklung befindlichen Systemen muss sich allerdings noch zeigen, ob die angestrebten Ziele im praktischen Einsatz auch tatsächlich erfüllt werden können.

Insgesamt lässt sich daher sagen, dass die Entwicklung eines neuen, leistungsfähigen und softwarebasierten Systems der Klassen 2 und 3, ohne SPoF und mit Unterstützung

vieler Netzwerkteilnehmer und einer weitgehend automatischen Konfiguration, erstrebenswert ist (vgl. zu diesem Kapitel [1] [4] [5]).

2.1.3 Zeitschlitzverfahren

Ein grundlegendes Prinzip, dass zur Regelung des Medienzugriffs in einem echtzeitfähigen Kommunikationssystem genutzt werden kann, ist die zeitlich abgegrenzte Vergabe der Sendeerlaubnis (Time Division Multiple Access, TDMA). In herkömmlichen Systemen, die nach diesem Prinzip arbeiten (z.B. CC-Link IE Field [6] [7], Ethernet Powerlink [8]) erhält dabei immer nur ein Netzwerkteilnehmer die Sendeerlaubnis, während alle anderen Netzwerkteilnehmer in dieser Zeit nur Daten empfangen. Die Sendeerlaubnis ist stets zeitlich begrenzt und wird den Netzwerkteilnehmern in einer von einem zentralen Managementknoten festgelegten Reihenfolge erteilt. Die Sendeerlaubnis kann z.B. durch ein sogenanntes Token Passing direkt von einem Netzwerkteilnehmer zum nächsten weitergegeben (z.B. bei CC-Link IE Field [6]) oder jedem Netzwerkteilnehmer direkt vom Masterknoten erteilt werden (z.B. bei Ethernet Powerlink [8]). Durch diese sogenannten Zeitschlitzverfahren wird das gleichzeitige Senden durch mehrere Netzwerkteilnehmer unterbunden und es kann weder durch Kollisionen in Netzwerken mit Hubs noch durch lange Paketwarteschlangen in Netzwerken mit Switches zu unerwarteten Verzögerungen bei der Datenübertragung kommen. Auf Grund der typischerweise zyklischen Kommunikationsanforderungen in Automatisierungsumgebungen wiederholt sich die Vergabe der Sendeerlaubnis entsprechend der geforderten Zykluszeit. Das Prinzip ist in Abbildung 1 dargestellt.

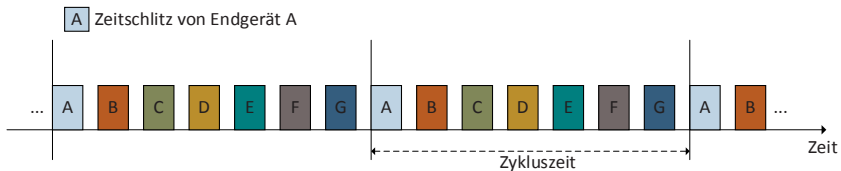


Abbildung 1: Beispiel für ein Zeitschlitzverfahren.

Während seines zugeteilten Zeitschlitzes kann ein Netzwerkteilnehmer Nutzdaten senden. Die Abstände zwischen den Zeitschlitzes müssen zur Weitergabe der Sendeerlaubnis genutzt werden (z.B. Tokenweitergabe oder Erteilung der Sendeerlaubnis durch den Master). Eine weitere Möglichkeit zur Realisierung von Zeitschlitzverfahren besteht darin, die Uhren der Netzwerkteilnehmer exakt zu synchronisieren. Steht allen Netzwerkteilnehmern eine einheitliche Zeitbasis zur Verfügung, so kann für jeden Netzwerkteilnehmer ein fester Zeitraum definiert werden, während dem er die Sendeerlaubnis hat. Jeder Netzwerkteilnehmer kann dann anhand der aktuellen Zeit feststellen, ob er im Moment die Sendeerlaubnis besitzt. Eine explizite Signalisierung der Sendeerlaubnis

ist dann nicht mehr notwendig. Doch auch in diesem Fall sind geringe Abstände zwischen den Zeitschlitzten im Allgemeinen notwendig. Hat ein Netzwerkteilnehmer seinen Sendevorgang abgeschlossen, so benötigen die gesendeten Daten noch Zeit, um beim Empfänger einzutreffen, bevor das nächste Gerät die Sendeerlaubnis erhalten darf und ebenfalls mit der Datenübertragung beginnt. Andernfalls kann es dazu kommen, dass die in unterschiedlichen Zeitschlitzten gesendeten Daten durch sich überschneidende, unterschiedliche lange Routen auf dem Weg zu ihrem Ziel aufeinander treffen und es daher zu unerwarteten Verzögerungen in Switches oder Kollisionen im Falle von Netzwerken mit Hubs kommt. Eine kontinuierliche Verwendung des Netzwerks zur Übertragung von Nutzdaten ist mit derartigen Zeitschlitzverfahren also nicht möglich. Die einzuhaltenden Abstände hängen von der Netzwerktopologie und der verwendeten Ethernet-Hardware ab, da hierdurch vorgegeben wird, wie lange es dauert bis z.B. eine Datenübertragung abgeschlossen oder die explizite Weitergabe der Sendeerlaubnis erfolgt ist.

Da die Netzwerkteilnehmer sich in einer Automatisierungsumgebung nicht häufig ändern, kann die Vergabe der Sendeerlaubnis vor Inbetriebnahme des Kommunikationssystems statisch festgelegt werden. Um Unterbrechungen des Produktionsbetriebes einer Anlage zu vermeiden, ist es für ein Echtzeitkommunikationssystem dennoch wünschenswert, dass es dynamisch auf Veränderungen einzelner Netzwerkteilnehmer reagieren kann, ohne dass andere Netzwerkteilnehmer in ihrer Echtzeitkommunikation beeinflusst werden oder ein Stillstand der Anlage erforderlich ist. Aus diesem Grund bieten viele der etablierten IE-Systeme Mechanismen, um Hot-Plug von Geräten oder auch eine schnelle Anpassung an Ausfälle zu ermöglichen. Mit dem Trend, die Anzahl vernetzter Geräte auch in industriellen Anwendungen zu erhöhen [1], sind Mechanismen zur dynamischen Anpassung, ebenso wie die Skalierbarkeit, eine wichtige Eigenschaft für ein zukunftssicheres Echtzeitkommunikationssystem.

2.1.4 Analyse der Übertragungslatenz

Bei der Entwicklung von Echtzeitkommunikationssystemen muss die Übertragungslatenz genau untersucht werden, um Aussagen über die Einhaltung von Echtzeitanforderungen treffen zu können. Bei der Verwendung von Ethernet und einem in der Regel durch das Betriebssystem zur Verfügung gestellten Netzwerkstack (zur Implementierung verschiedener Protokollschichten von MAC bis Transportschicht) kann die Latenz der Übertragung in fünf Bestandteile unterteilt werden. Startet eine Anwendung das Senden von Daten über das Netzwerk durch den Aufruf einer entsprechenden Sendefunktion des Netzwerkstacks, so vergeht bis zum Beginn des Sendevorgangs auf dem physikalischen Medium noch einige Zeit. In dieser Zeit müssen Funktionen des Netzwerkstacks durchlaufen und die jeweiligen Daten in den Sendepuffer des Netzwerkcontrollers kopiert werden. Da ein Großteil der Verzögerung auf die Softwareverarbeitung im Netzwerkstack zurückzuführen ist, wird diese Verzögerung im Rahmen dieser Arbeit

als $t_{Software_S}$ bezeichnet. Die Dauer des darauf folgenden Sendevorgangs, der durch die Netzwerkhardware ausgeführt wird, ist durch die Bandbreite der Verbindung und die Größe des zu sendenden Ethernet Frames bestimmt und wird als *transmission delay* bzw. $t_{Transmission}$ bezeichnet. Bis zur Ankunft beim Empfänger müssen unter Umständen mehrere Kabel durchlaufen werden, deren hinzugefügte Übertragungsverzögerungen (*propagation delay*) von den jeweiligen Kabellängen und dem verwendeten Medium abhängen. Die Verzögerung durch das i -te durchlaufene Kabel wird hier mit $t_{Propagation_i}$ bezeichnet. Auf dem Weg durchlaufene Switches fügen jeweils eine Verzögerung von t_{Switch_i} hinzu. Nachdem das Empfängergerät das Paket erhalten hat, muss noch ein Kopiervorgang aus dem Empfangspuffer erfolgen und wiederum Funktionen des Netzwerkstacks durchlaufen werden, bevor eine auf die Daten wartende Anwendung das Paket erhält. Diese abschließende Verzögerung wird als $t_{Software_E}$ bezeichnet. Die Bedeutung der fünf Parameter wird in Abbildung 2 veranschaulicht.

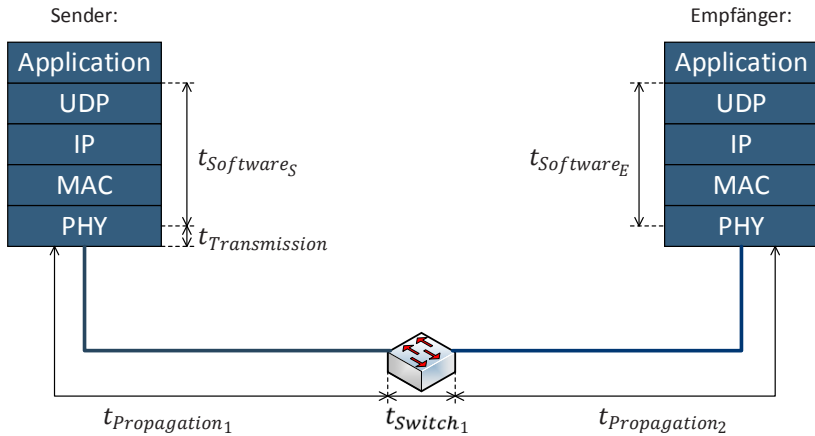


Abbildung 2: Darstellung der beschriebenen Verzögerungsparameter bei der Übertragung von Daten über das Netzwerk.

Die fünf genannten Einflüsse werden hier als $t_{Delivery}$ zusammengefasst:

$$t_{Delivery} = t_{Software_S} + t_{Transmission} + \sum_{i=1}^{N+1} t_{Propagation_i} + \sum_{i=1}^N t_{Switch_i} + t_{Software_E} \quad (2.1)$$

$t_{Transmission}$ beschreibt die Zeit, die vom Beginn des Sendevorgangs des ersten Bytes

des zu sendenden Frames bis zum Ende des Sendevorgangs des letzten Bytes des Frames vergeht. Nach dem Ablauf der Zeit

$$t_{Software_S} + t_{Transmission} + \sum_{i=1}^{N+1} t_{Propagation_i} + \sum_{i=1}^N t_{Switch_i}$$

seit dem Aufruf der Sendefunktion durch die Anwendung ist somit auch bereits das letzte Byte eines Frames beim Empfängergerät angekommen und die Verarbeitung des Frames durch den Netzwerkstack kann beginnen. $t_{Transmission}$ darf daher auf der Seite des Empfängers nicht noch einmal hinzugezählt werden.

Die Softwareverzögerungen im Sender und Empfänger werden an einigen Stellen dieser Arbeit zusammengefasst:

$$t_{Software} = t_{Software_S} + t_{Software_E} \quad (2.2)$$

Kommt ein Echtzeitkommunikationssystem zum Einsatz, das den Medienzugriff mit einem Zeitschlitzverfahren regelt, so muss vor dem Beginn des Sendevorgangs außerdem auf die Sendeerlaubnis gewartet werden. Da hierbei oftmals mehrere Pakete in einer Warteschlange auf die Sendeerlaubnis warten, wird diese Verzögerung als t_{Queue} bezeichnet. Die vollständige Übertragungslatenz $t_{Latency}$ eines entsprechend arbeitenden Echtzeitkommunikationssystems setzt sich daher folgendermaßen zusammen:

$$\begin{aligned} t_{Latency} &= t_{Queue} + t_{Delivery} \\ &= t_{Queue} + t_{Software_S} + t_{Transmission} + \\ &\quad \sum_{i=1}^{N+1} t_{Propagation_i} + \sum_{i=1}^N t_{Switch_i} + t_{Software_E} \end{aligned} \quad (2.3)$$

Ist auf Grund der Datenmenge die Übertragung mehrerer Ethernet Frames notwendig und die Latenz bis zur Ankunft aller Daten ist relevant, so muss in der obigen Betrachtung nur $t_{Transmission}$ entsprechend angepasst werden. Im Falle ausreichend schneller Endgeräte stehen die zu sendenden Frames dem Ethernetcontroller direkt nacheinander zur Verfügung und $t_{Transmission}$ wird weiterhin vollständig durch die Verbindungsgeschwindigkeit und die zu sendende Datenmenge bestimmt. Der Abstand zwischen den Frames entspricht der minimalen Interframe-Gap des jeweiligen Verbindungstyps. Für langsamere Endgeräte kann $t_{Transmission}$ größer werden, als auf Grund von Datenmenge und Verbindungsgeschwindigkeit eigentlich zu erwarten ist. In diesem Fall kann

der Ethernetcontroller nicht schnell genug mit den zu sendenden Daten versorgt werden, was sich in größeren Lücken zwischen den Frames anstelle der minimalen Inter-frame-Gaps äußert.

Für die in Tabelle 1 gezeigten Übertragungslatenzen mit der Originalquelle [3] soll es sich laut [3] um Latenzen handeln die auf Anwendungsebene gemessen wurden und daher mit dem hier erläuterten $t_{Latency}$ vergleichbar sind. Allerdings wird in [3] auch darauf hingewiesen, dass die Angaben ursprünglich vom jeweiligen Hersteller des Systems stammen und die Angaben nicht durch unabhängige Messung verifiziert wurden. Eine sich unterscheidende Interpretation von einer Messung „auf Anwendungsebene“ ist daher möglich und die Vergleichbarkeit der Angaben kann nicht garantiert werden. Für die anderen Latenzangaben aus Tabelle 1 gilt die gleiche Einschränkung.

Soll eine bestimmte Menge von Daten in einem Zeitschlitz übertragen werden, so muss die Länge des Zeitschlitzes (Dauer der Sendeerlaubnis) hierfür entsprechend $t_{Transmission}$ gewählt werden, dass das sendende Geräte für diese Datenmenge benötigt, da $t_{Transmission}$ der tatsächlichen Sendedauer für diese Daten entspricht. Die Länge von vergebenen Zeitschlitzes wird im Rahmen dieser Arbeit mit $t_{SlotLength}$ bezeichnet. Wird zwischen zwei Zeitschlitzes eine zu sendende Datenmenge erzeugt, die maximal so groß ist, dass sie noch im nächsten Zeitschlitz vollständig gesendet werden kann, so lässt sich auch die maximal auftretende Übertragungslatenz für die Übertragung der gesamten Datenmenge leicht angeben. Hierfür muss in Formel (2.1) bzw. (2.3) $t_{Transmission}$ durch $t_{SlotLength}$ ersetzt werden:

$$t_{Delivery} = t_{Software_S} + t_{SlotLength} + \sum_{i=1}^{N+1} t_{Propagation_i} + \sum_{i=1}^N t_{Switch_i} + t_{Software_E} \quad (2.4)$$

$$\begin{aligned} t_{Latency} &= t_{Queue} + t_{Delivery} \\ &= t_{Queue} + t_{Software_S} + t_{SlotLength} + \sum_{i=1}^{N+1} t_{Propagation_i} + \sum_{i=1}^N t_{Switch_i} + t_{Software_E} \end{aligned} \quad (2.5)$$

Nach dem Ablauf von $t_{Latency}$ hat dann die gesamte erzeugte Datenmenge die Anwendungsebene des Empfängers erreicht.

Zu $t_{software_S}$ ist außerdem anzumerken, dass diese Verzögerung ein Problem für die Einhaltung von Zeitschlitzten darstellen kann. Das Warten auf einen zugeteilten Zeitschlitz sei im Endgerät beispielsweise auf Anwendungsebene implementiert. Wurde der zugeteilte Zeitschlitz erreicht, so endet der Wartevorgang und die Sendefunktion des Netzwerkstacks wird mit einem wartenden Paket aufgerufen. Bis der Netzwerkstack durchlaufen wurde und der Sendevorgang durch den Netzwerkadapter erfolgt, ist ein Teil des vorgesehenen Zeitschlitzes bereits vergangen. Erfolgt dann ein Senden entsprechend der zugeteilten Zeitschlitzlänge, wird am Ende noch über den zugeteilten Zeitschlitz hinaus gesendet. Der Sendevorgang ist also auf Grund von $t_{software_S}$ gegenüber dem vorgesehenen Zeitschlitz verschoben. Ist $t_{software_S}$ jedoch konstant und vorab bekannt, so kann dieser Fehler korrigiert werden, indem der Aufruf der Sendefunktion auf Anwendungsebene entsprechend früher erfolgt. Zur Vereinfachung wird in den folgenden Analysen (Kapitel 3) davon ausgegangen, dass dies zutrifft. In der Praxis ist $t_{software_S}$ im Allgemeinen nicht konstant. Die dann notwendigen Maßnahmen werden in Kapitel 5 erläutert.

Switchverzögerungen hängen normalerweise von der Auslastung der verwendeten Verbindung ab, da bei der Nutzung von stark ausgelasteten Verbindungen Wartezeiten für die Pakete im Switchpuffer entstehen. Bei der Nutzung eines Zeitschlitzverfahrens werden derartige Wartezeiten vermieden, da durch die Regelung des Netzwerkzugriffs niemals mehrere für das gleiche Ziel bestimmte Pakete gleichzeitig in einem Switch eintreffen und sich daher keine größeren Datenmengen im Switchpuffer ansammeln können. Die Switchverzögerung t_{switch_i} hängt dann nur noch von der Switch-Implementierung und der Paketgröße ab. Falls nicht explizit anders genannt, ist im Verlauf dieser Arbeit immer die Switchverzögerung in diesem speziellen Fall des geregelten Netzwerkzugriffs gemeint.

2.1.5 Anforderungsparameter

Zur Angabe der Anforderungen einer Anwendung bzw. eines Gerätes an ein Netzwerk mit einem Zeitschlitzverfahren können folgende Parameter verwendet werden:

- $t_{SendCycle}$ beschreibt die Dauer eines Sendezyklus der Anwendung. In einer Automatisierungsumgebung kann dies beispielsweise das Intervall zwischen dem wiederholten Senden von Sensordaten sein.
- $t_{RequiredSlotLength}$ gibt an, wie lang ein der Anwendung zugeteilter Zeitschlitz sein muss, damit darin alle innerhalb von einem Sendezyklus erzeugten Daten gesendet werden können.
- $t_{MaxLatency}$ beschreibt die maximal zulässige Latenz der Datenübertragung. Der Parameter bezieht sich auf den Zeitraum vom Zeitpunkt des Sendewunsches einer Anwendung bis zum Zeitpunkt des Eintreffens aller Daten auf der Anwendungsebene des Empfängers (vgl. Formel (2.5)).

Diese Angaben können bei der Planung der Kommunikation verwendet werden, um zu entscheiden, in welchen Abständen eine bestimmte Anwendung einen Zeitschlitz erhält und wie lang dieser sein muss. Der Abstand zwischen zwei Zeitschlitzen einer Anwendung (vgl. Abbildung 1) wird im Rahmen dieser Arbeit als Zykluszeit bzw. $t_{cycleTime}$ bezeichnet. Im einfachsten Fall wird $t_{cycleTime}$ entsprechend dem Sendezyklus $t_{sendCycle}$ der Anwendung gewählt und die Länge $t_{slotLength}$ des vergebenen Zeitschlitzes entsprechend dem Anforderungsparameter $t_{requiredSlotLength}$ festgelegt. Auf diese Weise wird der Anwendung genau die gewünschte Datenrate zur Verfügung gestellt. Diese einfache Zeitschlitzkonfiguration ist nicht immer möglich. Diese Problematik wird ebenso wie die Bedeutung von $t_{maxLatency}$ in Kapitel 3.3.4 erläutert.

2.2 Software Defined Networking (SDN)

Heutige Netzwerke (hiermit sind nicht nur IE-Systeme gemeint) bestehen aus verschiedenen Netzwerkelementen. Zu den eingesetzten Bestandteilen zählen beispielsweise einfache Switches, Router, Firewalls und Load Balancer. Diese Bestandteile realisieren jeweils unterschiedliche Funktionen im Netzwerk. Jedes dieser Netzwerkelemente besitzt zur Realisierung dieser Funktionen eine eigene Kontrolllogik, die eintreffende Pakete analysiert und Entscheidungen über deren Verarbeitung trifft. Die Kontrolllogik ist direkt in das jeweilige Netzwerkelement integriert und daher mit der Hardware verbunden, die die vorgesehenen Verarbeitungsschritte anschließend ausführt. Typische Verarbeitungsschritte umfassen die gezielte Paketweiterleitung mit oder ohne Veränderung bestimmter Headerfelder und das Verwerfen von Paketen. Die Gesamtheit der im Netzwerk vorhandenen Kontrolllogiken wird auch als Control Plane bezeichnet, die Gesamtheit der ausführenden Hardware als Forwarding Plane oder Data Plane. Heutige Netzwerke besitzen durch die direkte Verbindung von Kontrolllogik und verarbeitender Hardware eine verteilte Control Plane. Während dies hinsichtlich der Ausfallsicherheit von Netzwerken vorteilhaft ist, haben sich in den vergangenen Jahren auch einige daraus resultierende Probleme gezeigt (vgl. zu diesem Abschnitt [9] [10]).

Für Veränderungen der Netzwerkkonfiguration steht kein zentraler Controller zur Verfügung, sondern es müssen viele einzelne Geräte individuell neu konfiguriert werden. Um diesen Vorgang zu erleichtern werden von den großen Netzkaustrüstern Managementtools angeboten, die zentrale Konfigurationsmöglichkeiten bieten. Auf Grund unterschiedlicher Protokolle und Schnittstellen bieten derartige Tools jedoch keine umfassende Kompatibilität zur Hardware verschiedener Hersteller. Die hiermit insgesamt geringe Flexibilität von Netzwerken hat zu einer Verlangsamung der Weiterentwicklung von Netzwerktechnologien geführt, da neue Technologien nur unter großem Aufwand in bestehende Netzwerke übernommen werden können (vgl. zu diesem Abschnitt [9] [10]).

Das Software Defined Networking soll diese bestehenden Probleme überwinden und weitere neue Möglichkeiten bieten, indem grundlegende Maßnahmen zur Umstrukturierung von Netzwerken ergriffen werden: Die Control Plane wird von den einzelnen Netzwerkelementen getrennt und in einem zentralen Controller zusammengeführt. Dieser wird in Software implementiert, um eine optimale Anpassbarkeit an die jeweiligen Bedürfnisse zu gewährleisten (vgl. zu diesem Abschnitt [9] [10] [11]).

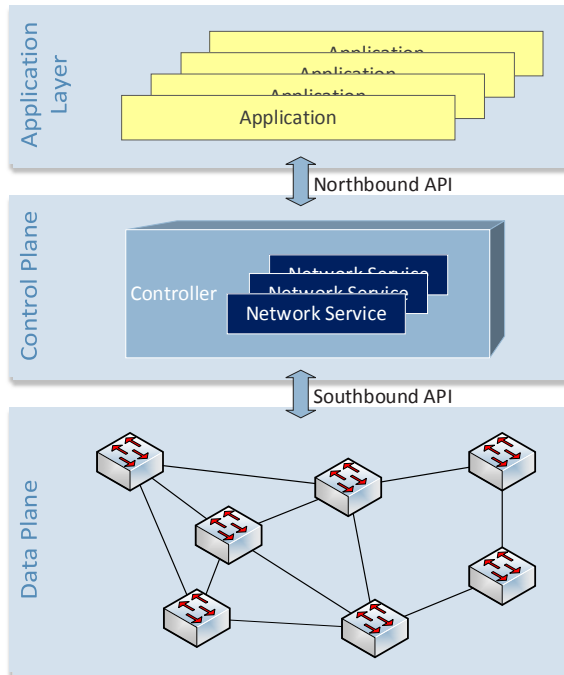


Abbildung 3: Konzept des Software Defined Networking [11] [9].

Der Controller kommuniziert mit den Elementen der Data Plane über eine standardisierte Schnittstelle, die auch als Southbound API (Application Programming Interface) bezeichnet wird. Da die Elemente der Data Plane hier im Allgemeinen keine eigene Kontrolllogik besitzen, treffen sie keine Entscheidungen über die Verarbeitung von Paketen. Stattdessen nutzen sie die Southbound API, um die korrekte Verarbeitung von bei ihnen eingetroffenen Paketen beim Controller zu erfragen. Um das auf diesem Weg entstehende Datenaufkommen zwischen Controller und Elementen der Data Plane zu reduzieren, kann der Controller Regeln in den einzelnen Data-Plane-Elementen installieren, die die Verarbeitung für bestimmte Pakete festlegen. Die Elemente der Data Plane müssen dann nur noch Anfragen beim Controller stellen, wenn Pakete eintreffen,

für die keine passende Regel existiert. Die Southbound API ist für den Betrieb eines SDN zwingend notwendig (vgl. zu diesem Abschnitt [11] [12] [9]).

Optional ist die Verwendung einer sogenannten Northbound API möglich, um eine Kommunikation zwischen dem Controller und den Anwendungen, die das Netzwerk nutzen, zu realisieren. Über diese Schnittstelle können Anwendungen beispielsweise Anforderungen an das Netzwerk gegenüber dem Controller äußern. Eine mögliche Verwendung dieser Schnittstelle ist die Implementierung von Quality-of-Service-Funktionen. Welche Informationen über die Northbound API ausgetauscht werden und welche Funktionen mit ihrer Hilfe realisiert werden ist stark anwendungsabhängig. Aus diesem Grund gibt es hier im Gegensatz zur Southbound API noch keinen etablierten Standard. Da die Nutzung einer Northbound API jedoch große Vorteile bieten kann, ist sie ein aktuelles Forschungsthema (vgl. zu diesem Abschnitt [13] [9]). Zur Realisierung einer Northbound API kann entweder ein eigenes Protokoll entwickelt werden, oder es wird ein bereits existierendes Datenaustauschformat wie die Java Script Object Notation (JSON) verwendet.

Abbildung 3 zeigt die vorgesehene Netzwerkstruktur beim SDN. Durch die Nutzung von SDN ergeben sich einige Vorteile gegenüber herkömmlichen Netzwerken (vgl. [10] [11] [9]):

- **Vereinfachte Hardware** der Data Plane. Die Elemente der Data Plane müssen nur noch eine durch Standards festgelegte Menge an Funktionen bieten und über eine ebenfalls standardisierte Southbound API kommunizieren können. Komplexere Kontrolllogiken entfallen und die Geräte sollen dementsprechend kostengünstiger werden.
- **Kompatibilität.** Die Verwendung herstellerspezifischer Managementinterfaces entfällt für die meisten netzwerkspezifischen Aufgaben, da diese über die standardisierte Southbound API und einen passenden SDN-Controller realisiert werden.
- **Flexibilität.** Das dynamische Regelsystem ermöglicht eine Veränderung der Netzwerkfunktionalität einzelner Elemente der Data Plane, die bei Elementen herkömmlicher Netzwerke in der Regel nicht in vergleichbarem Umfang möglich ist.
- **Zentrales Management.** Der SDN-Controller bietet einen zentralen und einheitlichen Zugriff auf die Konfiguration des gesamten Netzwerks.
- **Programmierbarkeit.** Die Implementierung des Controllers in Software ermöglicht die Implementierung eines komplexen Netzwerkverhaltens und eine hohe Anpassungsfähigkeit. Neue Netzwerkkonzepte können auf einfache Weise sowohl in realen als auch in simulierten Umgebungen getestet werden und bestehende Netzwerke im Produktiveinsatz lassen sich auf einfache Weise und mit relativ geringen Kosten aktualisieren.

2.2.1 OpenFlow

OpenFlow ist eine bekannte und weit verbreitete Southbound API. Das Ziel bei der Entwicklung von OpenFlow war die Etablierung programmierbarer Netzwerke mit umfangreichen Testmöglichkeiten für neue Netzwerkkonzepte. Um Wissenschaftlern die Möglichkeit zu bieten, neue Netzwerkkonzepte in großen, realen Netzwerken zu testen, sollte OpenFlow in bestehende produktiv eingesetzte Netzwerke integrierbar sein. Um Tests in produktiv eingesetzten Netzwerken zu erlauben, musste eine zuverlässige Trennung des Traffics von Experimenten und des „normalen“ Traffics ermöglicht werden. Der normale Traffic kann dann nach herkömmlichen Mechanismen weitergeleitet werden, während gleichzeitig mit experimentellem Traffic neue Netzwerkkonzepte getestet werden. Der Einsatz von OpenFlow in kleineren Testaufbauten im Labor ist ebenfalls möglich. Ein weiteres wichtiges Ziel von OpenFlow ist die Kompatibilität zu existierender Hardware, um von den Herstellern von Netzwerkhardware unterstützt zu werden und eine schnelle Verbreitung zu ermöglichen. Aus diesem Grund wurde bei der Entwicklung von OpenFlow darauf geachtet, dass der größte Teil der an die Data-Plane-Elemente gestellten Anforderungen durch existierende Switches und Router bereits erfüllt wird, sodass die Unterstützung von OpenFlow durch ein einfaches Update der Firmware/Software implementiert werden kann [10]. Für OpenFlow ist der Einsatz in Unternehmen ebenso bedeutend wie der Einsatz in der Wissenschaft. Unternehmen können beim Einsatz von OpenFlow für das normale Trafficmanagement stark durch die vereinfachte Verwaltung und beschleunigte Weiterentwicklung ihrer Netzwerke profitieren. Die praktische Anwendbarkeit von OpenFlow wird beispielsweise durch den Einsatz in Googles Backbone bestätigt [14] (vgl. zu diesem Abschnitt [9]).

Durch die hervorragenden Eigenschaften hinsichtlich Kompatibilität und Testmöglichkeiten sowie die durch das SDN-Konzept gebotenen Vorteile ist OpenFlow in den vergangenen Jahren auf großes Interesse in Wissenschaft und Wirtschaft gestoßen und hat maßgeblich zu Akzeptanz und Verbreitung von SDN beigetragen. OpenFlow wurde 2008 ursprünglich von Nick McKeown et al. an der Universität von Stanford entwickelt und in [15] vorgestellt. Mittlerweile wird OpenFlow von der Open Networking Foundation unterstützt und weiterentwickelt. Die OpenFlow-Switch-Spezifikation legt die von einem OpenFlow-Switch geforderten Fähigkeiten sowie das Protokoll zur Kommunikation zwischen OpenFlow-Switches und -Controller fest. Die im Rahmen dieser Arbeit vorgestellten Informationen zu OpenFlow basieren neben [15] auf der Spezifikation der Version 1.0.0 [16], da das später verwendete Controller-Framework derzeit nur diese Version vollständig unterstützt (vgl. zu diesem Abschnitt [9]).

2.2.1.1 Switch

Dem Konzept von SDN folgend erfolgt die Verarbeitung von eingetroffenen Paketen in einem OpenFlow-Switch entsprechend der Vorgaben des OpenFlow-Controllers. Damit nicht für jedes eingetroffene Paket eine Kommunikation zwischen Switch und

Controller erforderlich ist, wird die Verarbeitung von Paketen in den meisten Fällen durch Regeln festgelegt, die der Controller im Switch installiert.

Das Regelsystem von OpenFlow arbeitet mit sogenannten Flows und Aktionen. „Flow“ ist die Bezeichnung für eine bestimmte Kommunikationsverbindung im Netzwerk. Jede Regel besteht aus einer Flow-Definition und keiner bis mehreren zugeordneten Aktionen. Die Flow-Definition legt fest, für welche Pakete die Regel angewendet werden darf. Tabelle 2 zeigt die Headerfelder, die von einem OpenFlow-Switch bei der Flow-Definition laut [16] mindestens unterstützt werden müssen. Neben den Headerfeldern bekannter Internetprotokolle zählt auch der Port, auf dem ein Paket am Switch eingetroffen ist, zur Flow-Definition (In Port).

In Port	MAC SRC	MAC DST	Ether-type	VLAN ID	VLAN Prio.	IP SRC	IP DST	IP Proto.	IP ToS/DSCP	TCP/UDP SRC	TCP/UDP DST
---------	---------	---------	------------	---------	------------	--------	--------	-----------	-------------	-------------	-------------

Tabelle 2: Paketheader zur Definition von Flows [16].

Bei der Definition von Flows müssen nicht alle Headerfelder angegeben werden. Trifft ein Paket beim Switch ein, so wird sein Header mit den Flow-Definitionen der vorhandenen Regeln verglichen. Alle in der Flow-Definition angegebenen Felder müssen mit dem Header des Paketes übereinstimmen, damit die Regel angewendet werden darf. In der Flow-Definition nicht angegebene Felder können im Paketheader beliebig sein (Wildcard). Für IP-Adressen kann ein Switch außerdem Subnetzmasken in der Flow-Definition unterstützen, um Regeln für Pakete bestimmter Subnetze zu ermöglichen [16]. Zusätzlich zu den in Tabelle 2 gezeigten Headerfeldern muss ein Switch einige Headerfelder des Address Resolution Protocol (ARP) und des Internet Control Message Protocol (ICMP) unterstützen (weitere Details in [16]). Theoretisch kann das Konzept auch auf weitere Protokolle ausgeweitet werden, allerdings ist dies nicht Bestandteil der Spezifikation der Version 1.0.0.

In Port	MAC SRC	MAC DST	Ether-type	IP SRC	IP DST	IP Proto.	IP ToS/DSCP	TCP/UDP SRC	TCP/UDP DST	Aktionen
*	*	*	IPv4	10.0.1.2	10.0.1.3	*	*	*	*	Out#2
*	*	*	IPv4	10.0.1.3	10.0.1.5	TCP	*	7777	6100	Out#3
*	*	*	IPv4	*	10.0.1.6	UDP	*	*	4800	-

Tabelle 3: Beispiele für Flows und zugeordnete Aktionen. * bedeutet, dass der jeweilige Headereintrag im eingetroffenen Paket beliebig sein darf (Wildcard). Nicht gezeigte Headerfelder sollen ebenfalls Wildcards beinhalten.

Die flow-basierte Verarbeitung erlaubt eine fein strukturierte Klassifizierung von Paketen und ermöglicht daher ein sehr flexibles Traffic-Management. Tabelle 3 zeigt einige Beispiele für Flow-Definitionen und die zugeordneten Aktionen.

Eine einfache Aktion ist beispielsweise die Weiterleitung des Paketes über einen bestimmten Port (im Beispiel mit Out#2 und Out#3 für die Weiterleitung über Port 2 bzw. 3 bezeichnet). Wird für eine Flow-Definition keine Aktion festgelegt, werden entsprechende Pakete verworfen. Die Unterstützung der Weiterleitung über physikalische Ports ist für eine OpenFlow-Switch-Implementierung zwingend notwendig. Optional kann eine Switch-Implementierung auch Aktionen zur Veränderung einzelner Headerfelder (z.B. für Network Address Translation) unterstützen, was die Einsatzmöglichkeiten des Switches deutlich vergrößert. Da OpenFlow-Switches häufig als Erweiterung existierender Switches und Router implementiert werden, ist auch eine optionale Aktion zur Weiterleitung von Paketen an die eigene, herkömmliche Kontrolllogik in der OpenFlow-Switch-Spezifikation enthalten.

Die Regeln werden im Switch in der sogenannten Flow-Tabelle gespeichert, die für jeden Eintrag neben der Flow-Definition und den zugeordneten Aktionen auch eine Statistik über die Nutzung des Flows enthält. Die Einträge der Flow-Tabelle besitzen außerdem Prioritäten, die die Verarbeitung bestimmen, falls ein Paketheader mit mehreren Flow-Definitionen übereinstimmt. Bei mehreren passenden Regeln gleicher Priorität kann ein Switch eine beliebige Regel zur Ausführung auswählen. Regeln, die alle Headerfelder definieren (keine Wildcards) erhalten automatisch die höchste Priorität [16].

Wird für ein Paket keine passende Regel gefunden, so wird eine spezielle Nachricht an den Controller gesendet, um die Verarbeitung zu erfragen. Die Nachricht enthält den Anfangsteil des Paketes, da dieser den Paketheader enthält, anhand dem der Controller eine passende Entscheidung für die Verarbeitung treffen kann. Wie viele Bytes des Paketes an den Controller gesendet werden kann der Controller durch eine Konfigurationsvariable in jedem Switch festlegen. Der Controller beantwortet die Anfrage in der Regel entweder mit einem Kommando zur korrekten Verarbeitung des Paketes, ohne eine neue Regel zu installieren, oder mit der Installation einer neuen passenden Regel, die auf das jeweilige Paket auch gleich angewendet wird. Neben dieser reaktiven Installation von Regeln kann ein Controller auch eine proaktive Installation vornehmen.

Das Senden einer Nachricht an den Controller mit einem Teil eines eingetroffenen Paketes kann außerdem als explizite Aktion, die einer Regel zugeordnet ist, erfolgen. Auch diese Aktion muss in einem OpenFlow-Switch laut Spezifikation zwingend implementiert sein. Sie ist nicht gleichbedeutend mit einer normalen Weiterleitungsaktion, da bei der Weiterleitung an den Controller eine spezielle Nachricht des OpenFlow-Protokolls erzeugt wird, die meist nur einen Teil des ursprünglichen Paketes enthält (vgl. zu diesem Kapitel [9] [15] [16]).

2.2.1.2 Protokoll

Das OpenFlow-Protokoll ist die standardisierte Schnittstelle zwischen einem OpenFlow-Controller (Control Plane) und den OpenFlow-Switches (Data Plane). Die Kommunikation erfolgt über eine TCP-Verbindung, die mittels TLS verschlüsselt und authentifiziert werden sollte. Die Verschlüsselung steht aktuell jedoch nicht in allen Controller- und Switch-Implementierungen zur Verfügung. Initiiert wird die Verbindung durch den Switch, der hierfür einen benutzerdefinierten physikalischen oder logischen Port des Switches nutzt. Bei dem verwendeten Switchport kann es sich entweder um einen Port handeln, der nur für die Verbindung zur Control Plane genutzt wird (Out-of-Band Control), oder um einen Port, der gleichzeitig auch für den normalen Traffic der Data Plane in Verwendung ist (In-Band Control). Die IP-Adresse des Controllers muss laut Switch-Spezifikation der Version 1.0.0 ebenfalls manuell im Switch konfiguriert werden. Allerdings sind Mechanismen zur automatischen Controllererkennung (IP und Port) geplant [16]. Der Controller wartet auf dem Port 6633 auf eingehende TCP-Verbindungen. Die ausgetauschten Nachrichten werden in drei Kategorien eingeordnet [16]:

- **Symmetrische Nachrichten**, die von beiden Seiten auf die gleiche Weise genutzt werden, wie z.B. Echo-Nachrichten zur Aufrechterhaltung der Verbindung.
- **Controller-to-Switch-Nachrichten**, die vom Controller zum Abfragen von Statusinformationen, zum Setzen von Konfigurationsparametern und zur Installation von Regeln genutzt werden. Sie erfordern in einigen Fällen eine Antwort durch den Switch.
- **Asynchrone Nachrichten**, die vom Switch ohne Anfrage des Controllers generiert werden. Sie werden in der Regel durch aufgetretene Ereignisse ausgelöst. Ein typisches Beispiel ist die Packet-In-Nachricht eines Switches nach Eintreffen eines Paketes ohne passende Regel.

Die im Rahmen dieser Arbeit wichtigsten Nachrichten werden im Folgenden erläutert.

Packet-In-Nachrichten sind asynchrone Nachrichten, die vom Switch an den Controller gesendet werden, wenn für ein eingetroffenes Paket entweder keine Regel oder eine Regel mit einer expliziten Weiterleitungsaktion an den Controller existiert. In der Packet-In-Nachricht kann entweder das vollständige Paket enthalten sein, oder nur ein Teil (abhängig von Switch-Konfiguration und -Fähigkeiten). Wird nur ein Teil des Paketes an den Controller gesendet, so wird das vollständige Paket normalerweise im Switch zwischengespeichert und eine entsprechende Buffer-ID in der Packet-In-Nachricht mit an den Controller gesendet. In der Packet-In-Nachricht ist außerdem die eindeutige ID des Switches sowie die Portnummer des Ports, an dem das zugehörige Paket angekommen ist, enthalten. Auf diese Nachricht kann eine Antwort vom Controller folgen, die die Verarbeitung des Paketes festlegt.

Send-Packet-Nachrichten sind Controller-to-Switch-Nachrichten und werden vom Controller dazu verwendet, Pakete aus einem Switch heraus zu senden. Sie enthalten standardmäßig das zu sendende Paket sowie die mit diesem Paket auszuführenden Aktionen. Die möglichen Aktionen sind die gleichen, die auch bei der Erstellung von Regeln verwendet werden können (z.B. Modifikation von Headerfeldern, Weiterleitung über einen bestimmten Port). Dieser Nachrichtentyp kann entweder dazu verwendet werden, vom Controller generierte Pakete zu versenden, oder als Antwort auf Packet-In-Nachrichten folgen, um die Verarbeitung eines einzelnen Paketes zu steuern. Wird die Send-Packet-Nachricht als Antwort auf eine Packet-In-Nachricht versendet, die eine Buffer-ID eines im Switch zwischengespeicherten Paketes enthielt, so kann die Buffer-ID in der Send-Packet-Nachricht verwendet werden, um das zu verarbeitende Paket zu referenzieren. Das zu verarbeitende Paket ist dann nicht in der Send-Packet-Nachricht enthalten.

Die **Modify-Flow-Entry**-Nachricht kann zur Veränderung von Einträgen in der Flow-Tabelle genutzt werden. Hiermit können Regeln hinzugefügt, verändert und gelöscht werden. Es handelt sich um eine Controller-to-Switch-Nachricht, die proaktiv oder als Reaktion auf eine Packet-In-Nachricht versendet wird. Sie enthält ein Kommando (Eintrag hinzufügen, ändern oder löschen), die Flow-Definition und außer im Falle des Löschens von Einträgen auch die für den Flow auszuführenden Aktionen. Beim Versand als Reaktion auf eine Packet-In-Nachricht mit Buffer-ID kann diese Buffer-ID wiederverwendet werden, damit die neue Regel auch auf das entsprechende zwischengespeicherte Paket angewendet wird.

Feature-Request- und **Feature-Reply**-Nachrichten zählen zu den Controller-to-Switch-Nachrichten und werden nach dem Verbindungsaufbau zwischen Controller und Switch ausgetauscht. Der Controller sendet die Feature-Request-Nachricht, auf die ein Switch mit der Feature-Reply-Nachricht antwortet. Diese enthält die eindeutige ID des Switches (Datapath ID bzw. DPID genannt), eine Liste vorhandener Ports, eine Liste unterstützter Aktionen und Informationen über weitere besondere Fähigkeiten des Switches (z.B. Anzahl vorhandener Buffer, abrufbare Statistiken).

2.2.1.3 Controller

Der Controller hält eine Verbindung zu den OpenFlow-Switches eines Netzwerkes aufrecht und ist durch die Beantwortung von Packet-In-Nachrichten mit einzelnen Verarbeitungsbefehlen (Send Packet) oder neuen Regeln (Modify Flow Entry) für eine sinnvolle Funktion des Netzwerks verantwortlich. Das Erfragen der korrekten Verarbeitung eines Paketes beim Controller bedeutet auf Grund des zusätzlichen Kommunikationsaufwandes und der Verarbeitung der Anfrage in Software eine relativ große Verzögerung der Weiterleitung des betroffenen Paketes. Außerdem ist die Verzögerung im Controller lastabhängig und es muss darauf geachtet werden, dass der Controller

nicht durch Anfragen überlastet wird. Eine Installation von Regeln in OpenFlow-Switches ist daher in den meisten Anwendungsfällen unerlässlich. Beim Vorhandensein einer passenden Regel ist die Verzögerung bei der Paketweiterleitung allein von der Switch-Implementierung abhängig (vgl. zu diesem Abschnitt [15] [9]).

Der Controller entscheidet über das Verhalten eines jeden OpenFlow-Switches im Netzwerk und ist daher maßgeblich für die Eigenschaften des gesamten Netzwerkes. Die Komplexität des Controllers und der daraus resultierenden Netzwerkeigenschaften sind vollständig von der jeweiligen Controller-Implementierung abhängig, ohne dass OpenFlow Vorgaben bezüglich der Kontrolllogik macht. Für ein funktionierendes Subnetz ohne Schleifen in der Topologie wäre beispielsweise ein Controller ausreichend, der für jeden verbundenen OpenFlow-Switch das Verhalten eines einfachen Layer-2-Switches (Flooding bei unbekannter Ziel-MAC-Adresse, gezielte Weiterleitung bei bekannter Ziel-MAC-Adresse) realisiert. Komplexere Funktionen herkömmlicher Netzwerkhardware (z.B. Router, Firewalls) sind ebenfalls problemlos möglich. Das eigentliche Ziel bei der Nutzung von SDN ist jedoch in der Regel die Realisierung eines Netzwerkverhaltens, das weit über die in einem herkömmlichen Netzwerk vorhandenen Möglichkeiten hinausgeht. So ist beispielsweise durch die flow-basierte Paketverarbeitung eine sehr fein strukturierte Unterscheidung des Traffics verschiedener Geräte und Anwendungen möglich und für jede Verbindung können unterschiedliche, komplexe Strategien des Traffic-Managements verfolgt werden (vgl. zu diesem Abschnitt [15] [9]).

Zur Erhöhung der Ausfallsicherheit können Backup-Controller verwendet werden. Beim Ausfall des primären Controllers sollte dann von den Switches eine Verbindung zu einem der Backup-Controller hergestellt werden. Zur Verbesserung von Skalierbarkeit und Ausfallsicherheit ist außerdem die gleichzeitige Verwendung mehrerer Controller in einem Netzwerk [15] und die simultane Verbindung eines Switches mit mehreren Controllern [16] vorgesehen. Die genaue Funktionsweise ist jedoch noch nicht Bestandteil der Switch-Spezifikation der Version 1.0.0.

2.2.2 POX-Controller

Seit der Entwicklung von OpenFlow sind einige Controller-Frameworks entstanden, auf deren Basis OpenFlow-Controller implementiert werden können. Das Hauptziel eines Controller-Frameworks ist es, das OpenFlow-Protokoll zu implementieren und Verbindungen zu Switches zu verwalten, damit der Programmierer einer Kontrolllogik sich vollständig auf die gewünschte Netzwerkfunktionalität konzentrieren kann, ohne sich mit den Details der Controller-Switch-Kommunikation befassen zu müssen. Das Sekundärziel eines Controller-Frameworks ist die Bereitstellung von Komponenten, die bei der Controllerentwicklung häufig benötigte Funktionen realisieren und daher bei der Entwicklung eigener Kontrolllogiken wiederverwendet werden können.

Im Rahmen dieser Arbeit wurde der Controller auf Basis von POX implementiert. POX [17] [18] [19] ist ein Controller-Framework, das aus NOX, dem ersten OpenFlow-Controller [20], hervorgegangen ist. POX ist in Python geschrieben und kann zur Entwicklung eigener Controller durch Python-Module erweitert werden. Das Framework verwaltet die Verbindungen zu den OpenFlow-Switches und stellt eine OpenFlow API bereit, die der unkomplizierten Erstellung und dem Versand von OpenFlow-Nachrichten dient. Außerdem werden eingetroffene OpenFlow-Nachrichten von einem Parser verarbeitet und stehen dem Programmierer zur einfachen Handhabung als Python-Objekte zur Verfügung. Des Weiteren besitzt POX ein Event-System, das die verschiedenen Controllermodule über Ereignisse wie eingetroffene Packet-In-Nachrichten oder neue Verbindungen zu Switches informiert. Das Event-System kann auch zur Kommunikation und Synchronisation zwischen mehreren Controllermodulen eingesetzt werden (vgl. zu diesem Abschnitt [9]).

Um ein OpenFlow-Netzwerk schnell in Betrieb zu nehmen, bietet POX vorgefertigte Module, die einfache Kontrolllogiken (z.B. Layer-2-Switch-Verhalten für jeden verbundenen OpenFlow-Switch) realisieren. Weitere Module stellen in einem Controller häufig benötigte Funktionen bereit und können vom Entwickler eines neuen Controllers (gegebenfalls in angepasster Form) weiterverwendet werden. Die im praktischen Teil dieser Arbeit verwendeten Module werden im Folgenden kurz beschrieben.

In POX steht eine Topologiedatenbank zur Verfügung, um die aktuelle Netzwerktopologie und Informationen über die einzelnen Geräte (Switches, Hosts) zu speichern. Sie setzt sich aus den Modulen **topology** und **openflow.topology** zusammen. **topology** stellt die eigentliche Datenbank dar, in der eine Liste vorhandener Geräte gespeichert ist. Im **topology**-Modul sind außerdem Klassen der verschiedenen Gerätetypen (Switches, Hosts) implementiert, die in der Datenbank zum Abspeichern der Geräte genutzt werden. **openflow.topology** verwaltet die Datenbank, indem es aufgetretene Events auswertet. In POX werden beispielsweise beim Verbindungsaufbau oder Verbindungsabbruch zu einem Switch Events generiert, die **openflow.topology** dazu veranlassen, den jeweiligen OpenFlow-Switch zur Datenbank hinzuzufügen oder aus dieser zu entfernen.

Das **openflow.discovery**-Modul implementiert aktive Suchmechanismen, die der Entdeckung von Verbindungen zwischen OpenFlow-Switches dienen. Das Modul generiert bei der Entdeckung von Verbindungen Events, die von **openflow.topology** verarbeitet werden und zur Aufnahme der entdeckten Verbindung in die Topologiedatenbank führen.

Das **host_tracker**-Modul analysiert eingehende Packet-In-Nachrichten, um vorhandene Endgeräte und deren Verbindung zum Netzwerk zu entdecken. Generiert werden entsprechend nutzbare Packet-In-Nachrichten z.B. wenn ein Gerät Daten

sendet und noch kein passender Flow-Eintrag in einem OpenFlow-Switch vorhanden ist. Das `host_tracker`-Modul muss erkennen, ob eine Packet-In-Nachricht am OpenFlow-Switch durch ein Paket ausgelöst wurde, dass von einem anderen OpenFlow-Switch stammt, oder durch ein Paket, das direkt von einem angeschlossenen Endgerät kommt. Nur Packet-In-Nachrichten die durch ein Paket direkt von einem Host ausgelöst wurden, sind für die Hosterkennung relevant. Pakete, die an einem OpenFlow-Switch ankommen und von einem anderen OpenFlow-Switch kommen, sind bereits weitergeleitete oder durch den Controller generierte Pakete und kennzeichnen keine Verbindungsstelle zu einem Endgerät. Solche Packet-In-Nachrichten müssen vom `host_tracker` daher ignoriert werden. Das `host_tracker`-Modul ist daher abhängig von `openflow.discovery`, da `openflow.discovery` die benötigte Information liefern kann, ob eine Packet-In-Nachricht an einer Switch-Switch-Verbindung entstanden ist. Aus einer Packet-In-Nachricht kann das `host_tracker`-Modul die Adressinformationen des sendenden Endgerätes auslesen, sowie die DPID des Switches und den Port, an dem es angeschlossen ist. Das `host_tracker`-Modul arbeitet nicht mit `openflow.topology` zusammen, sondern speichert eine eigene Datenbank gefundener Hosts inklusive der zugehörigen Verbindung. Das `host_tracker`-Modul führt keine aktive Suche durch, sondern wertet nur eintreffende Packet-In-Nachrichten aus.

2.3 Dijkstra-Algorithmus

Da sich eine Erkennung der Netzwerktopologie mit Hilfe von OpenFlow leicht realisieren lässt, können im Controller auch Routingalgorithmen eingesetzt werden, die vollständige Topologiekenntnisse erfordern. Dazu zählt beispielsweise der von Edsger W. Dijkstra entwickelte *Single Source Shortest Path*-Algorithmus. Dieser dient dazu, in einem kantengewichteten Graphen von einem Startknoten ausgehend die kürzesten Wege zu allen anderen Knoten des Graphen zu finden. Mit dem kürzesten Weg ist der Weg gemeint, bei dem die Summe der Gewichte aller auf dem Weg durchlaufenen Kanten am geringsten ist. Der Graph kann beispielsweise die Topologie eines Netzwerks abbilden: Bei den Knoten des Graphen handelt es sich um Switches und angeschlossene Endgeräte, bei den Kanten um die in der Netzwerktopologie vorhandenen Verbindungen. Als Kantengewicht sind verschiedene Parameter vorstellbar, wie z.B. die Anzahl der Hops (jede Kante hat das Gewicht 1) oder die zusätzliche Verzögerung durch Nutzung der jeweiligen Verbindung (u.a. von Switch- und Kabelverzögerung abhängig). Auch komplexere Kantengewichte, die neben der Verzögerung und Hopanzahl auch die Bandbreite der Verbindung mit einbeziehen, sind möglich. Damit der Dijkstra-Algorithmus anwendbar ist, dürfen die Gewichte der Kanten jedoch nicht negativ sein. Die Kantengewichte werden auch als Kosten bezeichnet.

Der Algorithmus beginnt mit einer Initialisierung der Gesamtkosten die zum Erreichen eines Knotens vom Startknoten aus jeweils anfallen. Die Gesamtkosten werden zu Beginn für jeden Knoten mit Ausnahme des Startknotens auf unendlich festgelegt. Die Kosten zum Erreichen des Startknotens werden mit 0 initialisiert. Dann wird eine Liste der erreichbaren Knoten angelegt. Zunächst wird in diese Liste nur der Startknoten eingetragen. Anschließend wird das folgende Vorgehen wiederholt, bis keine Knoten mehr in dieser Liste der erreichbaren Knoten enthalten sind (vgl. [21]).

Aus der Liste erreichbarer Knoten wird der Knoten mit den geringsten Gesamtkosten als aktiver Knoten ausgewählt und aus der Liste entfernt. Ausgehend vom aktiven Knoten werden die Gesamtkosten zum Erreichen der direkten Nachbarknoten auf dem Weg über den aktiven Knoten berechnet. Die für einen Nachbarknoten berechneten Gesamtkosten sind demnach die Summe der Gesamtkosten des aktiven Knotens und der Kosten der Verbindung vom aktiven Knoten zum jeweiligen Nachbarknoten. Sind die so berechneten Gesamtkosten zum Erreichen des Nachbarknotens geringer als die bisher für diesen Knoten bekannten Gesamtkosten, so wird dessen Eintrag für die Gesamtkosten aktualisiert und der aktive Knoten als sein Vorgänger gespeichert. In diesem Fall wird der entsprechende Nachbarknoten außerdem der Liste erreichbarer Knoten hinzugefügt, falls er dort noch nicht eingetragen war. Sind die für einen Nachbarknoten berechneten Kosten höher als die bisher für diesen Knoten angegebenen Gesamtkosten, so wird nichts unternommen. Wurden alle Nachbarknoten des aktiven Knoten bearbeitet, wird wieder der Knoten mit den kleinsten Gesamtkosten aus der Liste erreichbarer Knoten entfernt und als aktiver Knoten ausgewählt und der Vorgang beginnt erneut. Einmal als aktiver Knoten ausgewählte Knoten werden auf diesem Wege niemals erneut in die Liste erreichbarer Knoten aufgenommen, da die für sie später berechneten Gesamtkosten auf Grund der ausschließlich positiven Kantengewichte immer größer sind, als zum Zeitpunkt der Auswahl als aktiver Knoten (vgl. [21]).

Ist die Liste erreichbarer Knoten leer, so ist die Suche nach Pfaden abgeschlossen. An allen Knoten sind dann die Gesamtkosten des kürzesten vom Startknoten ausgehenden Weges eingetragen. Der zugehörige Weg kann gefunden werden, indem man vom Zielknoten aus den eingetragenen Vorgängern folgt, bis der Startknoten erreicht ist (vgl. [21]).

Der Algorithmus ist mit einer Zeitkomplexität von $O(N^2)$ (N bezeichnet die Anzahl der Knoten) implementierbar und stellt damit eine einfache Möglichkeit dar, in Netzwerken mit bekannter Topologie die optimale Route zwischen zwei Knoten zu finden. Daher wurde er in dieser Arbeit als Grundlage für die benötigten Routenplanungsalgorithmen verwendet.

2.4 Java Script Object Notation (JSON)

Bei JSON [22] [23] handelt es sich um ein Datenformat, dass zum Austausch und Abspeichern von Datenstrukturen genutzt werden kann. Der Vorteil des Formats ist es, dass eine für Menschen lesbare Repräsentation der Daten (Textformat) verwendet wird und dass für viele Programmiersprachen Bibliotheken existieren, die zum Erstellen und Parsen von JSON-Daten genutzt werden können.

In JSON gibt es zwei übergeordnete Strukturen:

- Ein *Array* ist eine Liste von Werten, die durch Kommas getrennt und von eckigen Klammern umschlossen wird.
- Ein *Objekt* ist eine ungeordnete Menge von Name : Wert-Paaren, die durch Kommas getrennt werden. Die Menge wird durch geschweifte Klammern umschlossen.

Gültige Werte sind in JSON:

- Ein weiteres Array
- Ein weiteres Objekt
- Ein von Anführungszeichen umschlossener String, der aus Unicode Code Points besteht (Details zur Behandlung bestimmter Sonderzeichen in [23])
- Eine Zahl, deren genauer Syntax in [23] spezifiziert ist
- Einer der Bezeichner *true*, *false* oder *null*

Code 1 zeigt ein Beispiel für ein gültiges JSON-Dokument.

```
{
  "name" : "Office PC",
  "ip_int" : 3232235778,
  "ip_string" : "192.168.1.2",
  "features" : {
    "dhcp" : true
    "ipv6" : false
  }
}
```

Code 1: Beispiel für ein JSON-Dokument.

JSON kann beispielsweise zum Austausch von Daten über eine Netzwerkverbindung genutzt werden. Durch sein Format ist es prinzipiell weniger effizient als ein speziell für eine Anwendung entworfenes Protokoll, allerdings erlaubt es durch die bereits vorhandenen Codebibliotheken eine schnelle Implementierung eines Datenaustausches, während für ein angepasstes Protokoll erst Parser und Funktionen zur Erstellung der Nachrichten geschrieben werden müssten. JSON ist daher eine gute Wahl, wenn eine schnelle Implementierung notwendig ist, die Effizienz bei der Datenübertragung auf Grund eines geringen Datenaufkommens jedoch von geringer Bedeutung ist. Daher

wurde es im Rahmen dieser Arbeit zur Realisierung einer Controller-Anwendungs-Kommunikation (Northbound API) genutzt.

Entwicklung und Evaluierung eines SDN-gestützten
echtzeitfähigen Gerätenetzwerkes

Schweißguth, E.B.

2016, XV, 123 S. 22 Abb., Softcover

ISBN: 978-3-658-14746-4