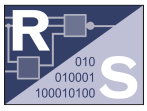
Implementierung eines multithreaded TCP/IP Stacks für einen auf AMIDAR basierten Java Prozessor

Bachelorarbeit
Robert Wiesner
31. Mai 2017

TECHNISCHE UNIVERSITÄT DARMSTADT



Erklärung gemäß § 22 Abs. 7 APB

Hiermit erkläre ich gemäß § 22 Abs. 7 der Allgemeinen Prüfungsbestimmungen (APB) der Technischen Universität Darmstadt in der Fassung der 4. Novelle vom 18. Juli 2012, dass ich die Arbeit selbstständig verfasst und alle genutzten Quellen angegeben habe und bestätige die Übereinstimmung von schriftlicher und elektronischer Fassung.

Darmstadt, den 31. Mai 2017
Ort, Datum

Robert Wiesner

Fachbereich Elektro- und Informationstechnik
Institut für Datentechnik
Fachgebiet Rechnersysteme
Prüfer: Prof. Dr.-Ing. Christian Hochberger
Betreuer: Dipl.-Inform. Changgong Li

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Diese Arbeit basiert auf den im Fachgebiet Rechnersystem entwickelten AMIDAR Prozessor und der dafür angepassten Java API. Zum Zeitpunkt der Arbeit verfügt AMIDAR mit der dazugehörigen API über grundlegende Netzwerk Funktionalitäten. Dazu gehören die Unterstützung für die Protokolle Ethernet, ARP, begrenzt IPv4 und UDP. Mit UDP kann keine korrekte und verlustfreie Datenübertragung garantiert werden, welche für viele Netzwerkanwendungen vorausgesetzt wird. Im Rahmen dieses Projektes wird TCP Stack entwickelt, sowie der IP Stack erweitert um eine geordnete und verlustfreie Datenübertragung zu realisieren.

1.2 Ziel der Arbeit

2 Grundlagen

2.1 Netzwerk Schichtenmodell

Netzwerk Kommunikation zwischen Anwendungen wird üblicherweise als Schichtenmodell beschrieben. Die unterste Schicht stellt dabei das physical layer. Das beschreibt die Physische Übertragung von Daten. Darüber sorgt die Sicherungsschicht für eine funktionierende Verbindung zwischen Endgeräten und den Übertragungsmedium. Auf dieser Schicht wird zum Beispiel das Ethernet Protokoll eingesetzt, das die übertragenen Daten auf Fehler überprüft und im Zweifel verwirft. Darunter kommt die Vermittlungsschicht, in der die Endgeräte Adressiert werden und Routing und Datenflusskontrolle gesteuert werden. Ein wichtiges Protokoll dieser Schicht ist das IP Protokoll.
Bei einer Datenübertragung von einer Anwendung zu einer auf einen anderen Endgerät laufenden werden die Daten in durch die Schichten nach unten gereicht, wobei in jeder Schicht ein neuer Header erzeugt wird, der für die jeweilige Schicht wichtige Informationen enthält. Zum Beispiel werden die Daten von der Anwendung mit betriebsystemsabhängigen Systemaufrufen an den TCPStack übergeben. Dieser erzeugt ein TCP Paket, das außer den Daten einen Header enthält, welcher Information bereitstellt, die unter anderen für das richtige Zusammensetzen der einzelnen Datenpakte beim Empfänger, als auch für die Zuordnung der übertragen Daten zu der jeweiligen Anwendung benötigt werden. Bei der anschließenden Erzeugung des IP Pakets bilden das TCP Paket bestehend aus Daten und TCP-Header die zu übertragenden Daten. Der IP Header enthält unter anderen die IP-Adressen des Ziel und Quell Geräts. Das IP Paket bleibt im Normalfall unverändert, bis das Zielgerät erreicht ist. An der nächst unteren Ebene steht das Ethernet Datagramm. Es enthält neben dem IP Paket die Physischen Adressen von des Quell Endgeräts und der nächsten Zwischenstation auf dem Weg zum Ziel. Bei Zwischenstation wird anhand der der Daten des IP Pakets der nächste Wegpunkt ermittelt und ein neues Datagramm erzeugt.
Wenn ein Datagramm das Ziel erreicht wird das IP Paket extrahiert und daraus das TCP Paket. Anhand der Port Nummer kann das TCP Paket der Anwendung zugeordnet werden.

2.2 Ethernet

Ethernet nach der IEEE Norm 802.3 ist seit den 90ern der am weitesten verbreitete Standard für Lokale Netzwerke und beschreibt sowohl die Bitübertragungs als auch die Sicherungsschicht.

2.2.1 Verfahren

Um zu ermöglichen das mehrere Endgeräte auf den selben Physischen Medium kommunizieren können wurde früher ein Zeitmultiplexverfahren eingesetzt, das durch den CSMA/CD Algorithmus gesteuert wird. Wenn eine Stelle Daten zum senden bereithält, wartet diese bis das

Medium ungenutzt ist und fängt dann an die Daten zu übertragen. Wenn 2 Stellen gleichzeitig beginnen zu senden wechseln beide auf ein SStörung-ErkanntSSignalmuster und beenden die Übertragung. Nach einer zufällig langen Pause wird jeweils ein erneuter Übertragungsversuch gestartet.
Mittlerweile werden Kollisionen durch die Einführung von Switches verhindert. In diesen können Ethernet Pakete zwischengespeichert werden bis diese gesendet werden können. Dadurch wird eine Vollduplex Übertragung zwischen Switches und anderen Endgeräten ermöglicht. Es kann jedoch vorkommen, das Switches bei zu großen Datenaufkommen überlastet werden weswegen die Ethernet Flow Control Datenpakete verwerfen kann. Deswegen ist es wichtig, das Protokolle auf den darüber liegenden Schichten verworfene Datenpakete erkennen und erneut senden können um eine zuverlässige Datenübertragung zu gewährleisten.

2.2.2 Ethernet Frame

Ein Ethernet Paket beginnt mit einer sieben Bit langen Präambel die aus einer alternierenden Folge von Einsen und Nullen besteht. Diese wird für die Synchronisation der Verbindung benötigt und ermöglicht es die Folgen Daten von Hintergrundrauschen zu unterscheiden. Unterbrochen wird die Präambel durch das auf Eins gesetzte SStart of Frame"Bit was mit den letzten Bit der Präambel 2 aufeinander Folgende Einsen ergibt. Der eigentliche Ethernet Frame beginnt mit der aus Sechs Byte bestehenden Ziel MAC-Adresse gefolgt von der Quell MAC-Adresse. Dazu kommen 2 Byte die den Typ des darüber liegenden Protokolls angeben. Zum Beispiel 0x0800 gibt IPv4 an. Dahinter kommen 46-1500 Bytes an Daten, gefolgt von 4 Bytes Frame Check Sequence. Diese besteht aus einer CRC Checksumme.

2.3 Internet Protocol Version 4 (IPv4)

Das IP Protokoll ist das für die Datenübertragung wichtigste Protokoll, auf der Vermittlungsschicht. Es wurde Entwickelt um eine paketvermittelte Kommunikation über mehrere Computernetzwerke hinweg zu ermöglichen. Quellen und Ziele der Übertragungen werden jeweils als Adressen mit fester 32 Bit Länge angegeben. Es gibt keine Mechanismen für Zuverlässige Übertragung, Flusskontrolle und Sequenzierung, weswegen das darüber liegen Protokoll dies sicher stellen muss. Es gibt jedoch Möglichkeiten zur Paketfragmentierung, falls Datenpakete die Maximale Segment Größe für Pakete der darunter liegenden Schicht überschreiten sollten.

2.3.1 Paket Aufbau

Version: Gibt in an welche Version des IP Protokolls verwendet wird. (4Bit)

IHL: Steht für Internet Header Length und gibt an wie 32Bit Wörter von dem IP-Header belegt werden.

ToS: Type of Service beinhaltet abstrakte Parameter zur Bestimmung der Qualität des gewünschten Services. Dabei geben die Bits Null bis Zwei die Priorität der Daten an. Die Bits Drei, Vier und Fünf stehen für niedrige Latenz, hohen Durchsatz und hohe Zuverlässigkeit. Die letztgenannten

Parameter werden von Netwerkgeräten von unterschiedlich interpretiert, in dem meisten ruft bessere Performance für einen der Parameter eine Verschlechterung bei einen anderen herbei.
Paketlänge: Gesamtlänge eines Pakets in Bytes einschließlich des Headers. Die 16Bit ergeben eine theoretische Gesamtlänge von 65.535 Bytes, was für viele Netzwerke jedoch nicht geeignet ist. Als Mindestgröße die alle Hosts unterstützen müssen wurde 576 Bytes festgelegt. Das ermöglicht es 512 Byte Daten und 64 Byte Header in einen Paket zu übertragen. Da der IP Header selber nur 20 Byte benötigt bleibt noch ein Puffer von 44 Byte für den Header des darüber liegenden Protokolls.

Kennung: Ein Identifikationswert, der benötigt wird um fragmentierte Pakete wieder zusammen zu setzen. (16Bit)

Flags: 3 Bits von denen das erste reserviert ist und dauerhaft auf Null gesetzt wird. Das zweite Bit gibt an, ob das Paket fragmentiert werden darf. Das letzte Bit wird gesetzt, wenn nach dem Paket noch weiter Fragmente folgen.

Fragment-Offset: Besteht aus 13 Bit und gibt an, an welche Stelle des Datagramms die Daten dieses Fragments gehören.

TTL (Time-to-live): Gibt die maximale Zeit in Sekunden an, die ein Paket im Netzwerk unterwegs sein darf. Sobald die TTL den Wert Null erreicht muss das Paket gelöscht werden. Da der Wert bei jeder Zwischenstation, unabhängig von der eigentlichen Verarbeitungszeit ebenfalls um eins Reduziert werden muss ist die übliche Lebensdauer eines Pakets deutlich kürzer als in der TTL angegeben.

Protokoll: Gibt an welches Protokoll im nächst höheren Level verwendet wird.

Header Checksumme: Checksumme nur über den Header. Da sich der Header beim Weg Zwischenstationen verändern kann, zum Beispiel wegen der Time to Life , muss die Checksumme nach jeder Verarbeitung des Headers an den Zwischenstationen neu berechnet werden.

Quell-IP-Adresse: IP Adresse des Hosts, der das Paket ursprünglich versendet hat.(4 Byte)

Ziel-IP-Adresse:IP Adresse des Zielendgeräts. (4 Byte)

Optionen/Füllbits

2.3.2 Adressierung

Um mehrere Netzwerke innerhalb eines großen zu ermöglichen, werden IP Adressen interpretiert, in dem eine bestimmte Menge der höherwertigen Bits das Netzwerk spezifiziert und die restlichen Bits den genauen Hosts des gewählten Netzwerks adressieren. Dabei soll Flexibilität bei der Unterteilung von kleineren Teilnetzwerken ermöglicht werden. Daher wurde das

Adressfeld entsprechend in bestimmte Klassen unterteilt. Es wurden 3 Klassen A,B,C angelegt. Die Klassen unterscheiden sich jeweils durch die Aufteilung des Adressbereichs zwischen Netzwerk Adresse und Host Adresse innerhalb des Netzwerks. Die ersten 1-3 Bits der Adresse geben jeweils Ausschluss darüber zu welcher Netzwerkklasse die jeweilige IP-Adresse gehört. Um innerhalb dieser starren Netzwerke feiner Aufgeteilte Subnetze zu erzeugen kann die Subnetmask genutzt werden. Die Subnetmask ist ebenfalls eine 4 Byte Zahl, deren niederwertige Bits auf Null gesetzt werden und damit den Variablen Teil innerhalb des subnetz angeben. Der Rest der Maske wird dabei auf Eins gesetzt.

2.3.3 Fragmentierung

Paketfragmentierung wird nötig, wenn ein Paket aus einen Netzwerk kommt das eine große maximale Segmentgröße erlaubt und durch eines geleitet wird, das nur eine kleinere Segmentgröße erlaubt. Dabei können die Pakete in eine theoretisch nahezu endlose Anzahl von kleinen Paketen zerlegt werden. Dabei müssen die Datenblocks alle Fragmente bis auf das letzte ein vielfaches von 64 Byte an Daten beinhalten. Damit fragmentierte Paket richtige zusammen gesetzt werden können, haben alle Fragmente, eines Pakets, die selbe Identifikationsnummer. Für das Zusammensetzen des Datenblock wird der Fragmentoffset benötigt der angibt, an welcher Stelle des ursprünglichen Datenblocks die Daten des Fragments stehen. Wobei das fragmented-flag angibt, ob noch weitere Fragmente folgen, oder ob dies das letzte Teil ist. Sobald alle Fragmente eines Pakets beim Ziel eingetroffen sind, kann das ursprüngliche Paket wiederhergestellt werden.

2.4 Transmission Control Protocol (TCP)

Da die Protokolle der unteren Schichten, IP und Ethernet keine fehlerfreie und Verlustlose Übertragung der Daten garantieren können, wird ein Protokoll auf der Transportschicht benötigt, das eine zuverlässige Übertragung von Daten zwischen Anwendungen auf entfernten Hosts sicherstellen kann, auch wenn auf jeden Hosts eine große Anzahl an Anwendungen läuft, die TCP verwenden. Für diesen Zweck wurde das TCP-Protokoll erschaffen. Um dabei die Datenpakete jeweils der richtigen Anwendung zuordnen zu können werden Port-nummern verwendet.
Es handelt sich bei TCP um ein verbindungsorientiertes Protokoll, das bedeutet, dass es zu beginn einer Übertragung einen klar definierten Verbindungsaufbau gibt. Nachdem die Verbindung etabliert es können Daten vollduplex in beide Richtungen übertragen werden. Wobei durch Sequenznummern und Acknowledge Nummern die richtige Reihenfolge und Vollständigkeit der Datenpakete sichergestellt wird. Pakete deren Erhalt nicht bestätigt wurde werden automatisch neu übertragen. Desweiteren verfügt TCP über Mechanismen um eine Überlast auf dem Übertragungsweg rechtzeitig zu erkennen und zu beheben.

2.4.1 Paketaufbau

Quell Port (16Bit): Gibt die Portnummer der Anwendung auf Senderseite an
Ziel Port (16Bit): Gibt die Portnummer der Anwendung auf Empfängerseite an
Sequence Number (32Bit):
Acknowledge Number (32Bit):
Data Offset (4Bit):Anzahl von 32Bit Wörtern aus denen der Header besteht.
Reserved (6Bit): reserviert für zukünftige Nutzung
Steuerungsbits (6Bit): 6 Flags die für die Ablauf Steuerung der Übertragung genutzt werden:
URG: Urgent Pointer, wird in moderner Software nicht mehr genutzt und wird von vielen Implementierungen ignoriert.
ACK: Acknowledgment gibt an, ob das Paket eine gültige Bestätigung für empfangene Pakete enthält.
PSH: Push, ist dieses Flag gesetzt, werden die empfangenen Daten sofort an die Hostsoftware weitergereicht. RST: Resettet die Verbindung. Wird im Fehlerfall gesendet und bricht die Verbindung ab. FIN: Signalisiert das ende der Verbindung, wenn es keine zu übertragenen Daten gibt.

Window (16Bits): Gibt die Größe des Emfpangswindows des Absenders an. Gilt für den Empfänger als obere Grenze des Congestion Windows
Checksum (16Bits): Für die Berechnung der Checksumme über das TCP Paket wird vorher ein Pseudoheader bestehend aus den Quell- und Ziel IP-Adresse, des IP-Codes für das verwendete Protokoll und die Gesamtlänge des eigentlichen TCP Pakets. Die Checksumme wird daraufhin über den Pseudeheader, den Header und die Payload berechnet. Dafür werden diese in 16 Bit große Blöcke aufgeteilt und im einer Komplement die Summe über diese gebildet.
UrgentPointer (16Bits): Gibt den Urgentpointer als Offset zu der Sequenznummer an. Wird nur interpretiert, wenn das URG Flag gesetzt ist.
Options: (Variabel): Optionale Informationen, die von TCP Implementierungen unterstützt werden können um Sicherheit und Performance zu verbessern. Optionen bestehen entweder nur aus einen Optionsbyte oder haben zusätzlich ein weiteres Byte das die Länge der jeweiligen Op-

tionen angibt. Padding: Falls der TCP Header mit den Optionen einen teilweise genutzten 32Bit Block hat, wird dieser mit Nullen aufgefüllt.

2.4.2 Zustände

TCP ist im Gegensatz zu den bisher genannt Protokollen Zustandsorientiert, was bedeutet, das es ja nach Zustand anders auf Nutzeraktionen und ankommende Pakete reagiert. Zu den Zuständen gehören : Closed : Das ist der Start zu stand einer TCP Instanz. Die Verbindung ist geschlossen. Usercalls außer Öpen"werden mit Fehlermeldungen quittiert und ankommende Pakete werden verworfen und mit einen Reset-Paket beantwortet.
SYN-sent: Nachdem eine Verbindung initiiert wurde, wird auf eine Antwort des "remote Hosts"gewartet.
SYN-received : Nachdem ein SYN Pakte erhalten und ein SYN-ACK gesendet wurde wird auf das ACK gewartet, um den 3-Wege-Handschlag abzuschließen.
ESTABLISHED: Nach dem der Verbindungsaufbau erfolgreich abgeschlossen wurde befinden sich beide Hosts im ESTABLISHED Zustand, in dem die eine Vollduplex Komminikation möglich ist.
FIN-WAIT-1: Wenn ein Verbindungsabbau initiert wurde wird ein FIN Paket gesendet und in den Zustand FIN-WAIT-1 gewechselt und auf die Bestätigung des Erhalts gewartet.
FIN-WAIT-2: In dem Fall das die Gegenstelle noch Daten zu übertragen hat, bleibt die Verbindung einseitig offen um die letzten Pakete zu empfangen.
CLOSING:
TIME-WAIT:

Zu den Zuständen kommen noch eine Reihe von Ereignissen auf, auf die, abhängig von den jeweiligen Zuständen entsprechend reagiert werden muss. Dazu gehören neben eintreffenden Paketen und Timeouts die Usercalls.
Active OPEN: Öffnet einen Port und initiiert den Verbindungsaufbau zu einen Remote Host.
Passive OPEN: Öffnet einen Port ohne eine Verbindung zu initiieren und wartet auf einen Verbindungsaufbau,
SEND: Fügt dem Sendepuffer Daten hinzu und sendet gegebenenfalls ein Datenpaket.
RECEIVE: Überprüft, ob genug Daten vorhanden sind und gibt diese an die Anwendung zurück.
CLOSE: Initiiert den Abbau der Verbindung, wenn es keine Daten mehr zu übertragen gibt.
ABORT: Bricht die Verbindung ab. In den meisten Zuständen wird in dem Fall ein Reset Paket gesendet.

2.4.3 Sequenznummern

Ein wichtiges Grundprinzip von TCP das jedes Byte an Daten eine eigene Sequenznummer zugewiesen werden kann, dadurch ist es möglich, das jedes Byte einzeln bestätigt werden kann. Der in TCP dazu verwendete Mechanismus arbeitet kumulativ. Das bedeutet das wenn eine Sequenznummer bestätigt wird, alle Sequenznummern kleiner als die Bestätigungsnummer als angekommen gelten. Dadurch wird es einfach den Verlust einzelner Pakete zu erkenne und die-

se nochmal zu senden. Das erste Byte eines Datenpaketes entspricht dabei der Sequenznummer des Pakets.

2.4.4 Verbindungsaufbau

Um zuverlässig einen sicheren Verbindungsaufbau zu gewährleisten verwendet TCP einen Drei Wege Handshake. Soll eine neue Verbindung aufgebaut werden wird die Initiale Sequenznummer zuverlässig generiert. Das SYN Paket enthält nur den HEADER mit der um eins inkrementierten Initialen Sequenznummer und den SYN bit auf eins gesetzt. Nach dem Sendevorgang wird in den Zustand SYN-SENT gewechselt. Wenn die Empfängerseite sich im Zustand LISTEN befindet kann die Verbindunganfrage bestätigt werden. Dafür wird ebenfalls eine Initiale Sequenznummer generiert. Zur Antwort wird ein Paket erzeugt, das die neue initiale Sequenznummer verwendet und als Acknummer die um eins erhöhte Sequenznummer des SYN Pakets verwendet. Die Flags für SYN und ACK werden gesetzt. Nach dem senden wird in den Zustand SYN-RECEIVED gewechselt.
Nach dem Erhalt des SYN-ACK Pakets sendet die initiierende Seite ein leeres ACK Paket und wechselt in den ESTABLISHED Zustand.

2.4.5 Datenübertragung

Wenn beide Seiten einer Verbindung den Established-Zustand erreicht haben, kann die eigentliche beidseitige Datenübertragung beginnen. Dabei wird die Übertragung der Daten als kontinuierlicher Datenstrom abstrahiert. Eine Anwendung schreibt dabei Bytes auf die in einen Datenpuffer im TCP-Stack. Die Datenwerden nicht sofort Übertragen, da sonst viele Pakete mit wenig Nutzdaten gesendet werden würden und die Verbindung schnell überlastet wäre, wobei ein großer Teil der übertragenen Daten für den durch TCP-, IP- und Ethernet-Header erzeugten Overhead genutzt werden müssten. Sobald der Schreibpuffer eine bestimmte Größe erreicht hat, oder die Software einen PUSH signalisiert, wird ein TCP-Paket erzeugt, das die Daten aus dem Puffer überträgt. Jedes Datenpaket wird mit einen Zeitstempel in der Retransmitqueue zwischengespeichert bis der Erhalt dieser bestätigt wurde. Nach dem Eintreffen eines Pakets, dessen ACK Flag gesetzt wurde, gelten alle Pakete, der Sequenznummer unter der Acknowledge-Nummer des angekommenen Paktes liegen bestätigt. Diese werden aus der Retransmitqueue entfernt. Wenn für ein Paket eine bestimmte Zeit lang keine Bestätigung eingetroffen ist, wird dieses nochmal gesendet.

2.4.6 Überlastkontrolle

Ein weiterer Schwerpunkt bei TCP ist das verhindern der Überlastung des Netzwerks, das zwischen Sender und Empfänger liegt, als auch die des Empfängers selber. Die beiden Mechanismen werden "Flow Controlünd "Congestion Control"genannt. Flow Control verhindert dabei die Überlastung des Empfängers und Congestion Control die des Übertragungsweges. Für das senden von Daten wird für Flow und Congestion Control jeweils ein Fenster bestimmt. Das Fenster gibt an, wie viele Bytes gesendet werden könne, ohne das diese bestätigt wurden. Wenn die Differenz der Sequenznummern und der letzten erhalten Acknowledgenummer so groß ist wie das

Fenster, werden keine neuen Datenpakete mehr versenden, bis weiter Pakete bestätigt wurden. Es wird jeweils das kleinere Fenster als Grenze genommen.
Das Flow Control vom Empfänger im TCP Header der von ihm versendeten Pakete angeboten. Das Problem bei der Congestion Control ist, das die Kapazität der Verbindung nicht trivial zu bestimmen ist. Für eine Annäherung kommen im normal Fall vier Algorithmen zum Einsatz. Mit Hilfe der Variable SSlow Start Threshold"wird festgelegt, welcher der Algorithmen im aktuellen Zeitpunkt zum Einsatz kommt. So lange unter der Slow Start Threshold liegt, wird der Slow Start Algorithmus verwendet. Eine Überlastung der Verbindung kann festgestellt werden, wenn mehrere Acknowledge Pakete für die selbe Sequenznummer empfangen werden.
Zu beginn einer Übertragung liegen noch keine Informationen über die mögliche Bandbreite des Netzwerks vor, weswegen die Bandbreite langsam sondiert werden muss. Dabei wird Slow Start Algorithmus genutzt, mit durch schnelles ansteigen der Übertragungsrate der Grenzwert erreicht wird. Die Slow Start Phase wird genutzt, bis die Größe des Congestion Windows die Slow Start Threshold übersteigt, oder eine Überlastung der Verbindung festgestellt wird. Da die Grenze der Verbindungskapazität sondiert werden soll wird die Slow Start Threshold zu Beginn auf einen Wert gesetzt, der nicht erreicht werden kann. wenn es zu einen Packet Timeout, wird die Slow Start Threshold auf einen Wert, der einen Bruchteil des erreichten Fensters entspricht gesetzt und der Slow Start Algorithmus, mit einen größeren Initialen Fenster, weiter verwendet. Diesmal wird die Slow Start Phase durch das Überschreiten der Slow Start Threshhold beendet. Für den Rest der Übertragung wechseln sich die beiden Algorithmen ab, bis die Verbindung beendet ist.

Es gibt eine Reihe von Varianten für die Congestion Control Implementierung. Alle davon verwenden eine Variation der folgenden Algorithmen. Hier wird als Beispiel die Reno Variante genannt.

Slow Start: Vergrößert das Congestion Window mit jeden bestätigten Segment um die Größe des Segments. Effektiv eine Verdoppelung des Windows pro Roundtrip. N := Anzahl der im letzten Ack bestätigten Sequenznummern

Congestion Avoidance : Arbeitetet nach dem Prinzip Additiv Erhöhen, multiplikativ erniedrigen.

Fast Retransmit : Findet statt, wenn mehrere ACKNOWLEDGE-Pakete für das selbe Segment empfangen werden. Dabei kann sicher davon ausgegangen werden, das dieses Segment verloren gegangen ist. In diesen Fall wird nicht auf einen Timteout für dieses Paket gewartet, stattdessen wird dieses sofort gesendet.In den meisten Algorithmen wird ein Fast Retransmit nach einem drei mal wiederholten Acknowledge durchgeführt. Die Slow Start Threshold wird nach einen Fast Retransmit auf dem Wert des halbierte Congestion Windows gesetzt und das Congestion Window auf die neue Slow Start Threshold gesetzt und um drei erhöht. Dadurch wird die Slow Start Phase übersprungen, da durch die Ankunft der Acknowledge Pakete davon ausgegangen werden kann, das es in dem Netzwerk nur einen temporären Engpass gab.

Fast Recovery : Der Fast Recovery Algorithmus wird verwendet, wenn nach einen Fast Retransmit noch weitere duplizierte Acknowledge Pakete ankommen. Dabei werden, wenn noch weitere Daten zum senden bereitstehen diese gesendet, da davon ausgegangen wird, das nur das Paket das im Fast Retransmit gesendet wurde verloren gegangen ist und die Datenübertragung ohne große Geschwindigkeitseinbußen fortgesetzt werden kann.

Timeout Wenn die Verbindung nicht mit den Fast Retransmit und Fast Recovery Algorithmen weitergeführt werden kann, oder erst gar keine Acknowledge Pakete ankommen, muss von einer größeren Überlastung oder Änderungen auf dem Übertragungsweg ausgegangen werden. Nachdem ein Paket einen bestimmten Zeitraum lang nicht bestätigt wurde wird dieses nochmal gesendet. In dem Fall wird die Slow Start Threshold ebenfalls auf die Hälfte des Congestion Windows gesetzt. Das Congestion Window selber wird dabei jedoch auf eins gesetzt und die Übertragung mit dem Slow Start Algorithmus fortgesetzt.

2.4.7 Verbindungsabbau

Bei dem beenden einer Verbindung muss sicher gestellt werden, das eine Verbindung abgebaut werden kann, ohne das es zu Datenverlust kommt. Ein Host, der keine Daten mehr zum senden hat kann die Close Operation ausführen. Dabei sendet dieser ein Paket mit den "FIN" Flag gesetzt, was der Gegenseite signalisiert, das der Host die Verbindung beendet möchte. Er lässt die Verbindung aber weiterhin offen für eingehende Datenpakete, bis die Gegenstelle signalisiert, das sie bereit ist die Verbindung zu beenden. Die Gegenstelle reagiert auf das FIN-Paket entweder mit einen ACK-Paket, wenn sie noch Daten zu übertragen hat, oder mit einen FIN-Paket, wenn sie keine Daten mehr zu übertragen hat und die Verbindung beendet werden kann.

2.5 Dynamic Host Control Protocol (DHCP)

Mit größeren lokalen Netzwerken mit wechselnden Teilnehmer kam der bedarf nach einer Zentralen Einrichtung, die die Verteilung der IP-Adressen innerhalb eines Netzwerkes verwaltet. Dafür wurde aufbauend auf den älteren Bootstrap Protokoll DHCP entwickelt. DHCP ist deswegen weitgehend kompatibel mit Bootstrap, weswegen mit Bootstrap Clients und Servern zusammengearbeitet werden kann. Neben der IP-Adresse können auch andere Parameter abgefragt werden. Zum Beispiel Gateway, Netzmaske, Zeitserver und Nameserver.

2.5.1 Paket Format DHCP (Bootstrap)

Op: Gibt den generellen Typ der Nachricht an. Ein Wert von 1 signalisiert eine Anfrage, wobei eine 2 eine Antwort signalisiert.

HType: Spezifiziert die Art der dem Netzwerk zugrunde liegende Hardware. Der Code 1 steht dabei für Ethernet.

HLen: Bezeichnet die Länge der physischen Adressen im Netzwerk. Im Falle von Ethernet Netzwerken sind dies 6 Byte.

Hops: Wird beim Absenden auf 0 gesetzt und von jedem Relay Agentüm eins erhöht.

XID: Eine Identifikationsnummer mit der Server Antworten den Anfragen zugeordnet werden können.

Secs: Dieses Feld ist für das Bootstrap Protokoll ungenau definiert und wird oft nicht verwendet. Für DHCP wird es für die Zeit in Sekunden genutzt die vergangen ist, seit der Client damit begonnen hat nach einen neuen "Leaseßu fragen.

Flags: 8 Bits de für als Flags genutzt werden können. Das erste Bit wird gesetzt, falls die Nachricht als Broadcast gesendet wird, was den DHCP Server signalisiert, das dieser über keine gültige IP Adresse verfügt.

CIAddr: Steht für "Client IP Adresse"Der Client schreibt seine eigene IP-Adresse in dieses Feld. Im falle von DHCP kann dieses nur in den Zuständen BOUND, RENEWING oder REBINDING genutzt werden. In allen anderen Fällen bleibt dieses Feld auf Null.

YIAddr: Abkürzung für "Your IP Address". Wird vom Server gesetzt um den Client eine Ip Adresse zuzuweisen.

SIAddr: Der Server gibt hier die Adresse des Servers an, dem der Client seine nächste Anfrage schicken soll.

GIAddr: Wird nur im Bootstrap Protokoll für die Kommunikation zwischen Client und Server verwendet, wenn diese nicht im selben Netzwerk oder Subnetz liegen. Wird im DHCP nicht dafür verwendet das Standardgateway anzugeben.

CHAddr: Hardware Adresse des Clients. Bei Nutzung von Ethernet die Mac-Adresse.

SName: Der DHCP Server kann hier optional seinen Namen angeben. Alternativ kann dieses Feld durch die öption overload"Funktion auch für Optionen genutzt werden.

File: Kann genutzt werden um bei einen DHCPDISCOVER oder OFFER eine Bootdatei anzugeben.

Options: Für das Bootstrap Protokoll gibt es eine große Menge an Optionen, einige davon werden exklusiv für DHCP genutzt.

Die Anzahl und Art der verwendeten Optionen ist variabel. Der Aufbau von diesen folgt jedoch einen definierten Format. Die ersten 8 Bit eines Optionsblock enthalten den Code der Option, weitere 8 Bit geben die Länge des folgenden Datenfeldes an. Das Optionfeld beginnt im Falle von DHCP mit der "Magic Number"99.130.83.99 nur dann können die exklusiven DHCP Optionen genutzt werden. Dazu gehören:

Requested IP Address Diese Option kann während eines dhcp Discovers gesetzt werden um die Verfügbarkeit einer bestimmten IP-Adresse anzufragen.
Code: 50; Länge: 4 Byte;

IPAddress Lease Time Kann in einer DHCP-Request oder DHCP-Offer Nachricht gesetzt werden. Der Client kann dabei eine bestimmte Lease Time in Sekunden angeben.
Code:51; Länge 4 Byte;

Option Overload Diese option signalisiert, das Option Overload genutzt wird. Das bedeutet das weitere Optionen anstelle der Felder SNAME oder FILE geschrieben werden.
Code 52; Länge 1; Es gibt 3 mögliche Werte die geschrieben werden können.

- 1: Optionen stehen im File Feld.
- 2: Optionen stehen im SNAME Feld.
- 3: Optionen stehen in beiden Feldern.

DHCP Message Type dieses Feld ist bei DHCP Nachrichten immer vorhanden und gibt die Art der DHCP Nachricht an.
Code 53; Länge 1;

- 1 DHCPDISCOVER
- 2 DHCPOFFER
- 3 DHCPREQUEST
- 4 DHCPDECLINE
- 5 DHCPACK
- 6 DHCPNAK
- 7 DHCPRELEASE
- 8 DHCPINFORM

Server Identifier Die Option kann von einem Client bei einer DHCPREQUEST gesetzt werden, um bei Unicast Nachrichten die Adresse von einen bestimmter Server anzugeben. Ein DHCP-Server kann dieses Feld setzten, damit ein Client mehrere DHCP Offer unterscheiden kann.
Code 54; Länge 4;

Parameter Request List Es kann eine Liste von Parametern übergeben werden um die Werte von diesen bei einen Server anzufragen. Dazu kann eine beliebig Lange Folge aus DHCP/Bootstrap Optionscodes genutzt werden
Code 54; Länge n;

2.5.2 Ablauf

Zu Beginn eines DHCP-Vorgangs, wenn zum Beispiel ein Client hochgefahren wird, verfügt dieser weder über eine IP-Adresse, noch über andere Information die Netzerktopologie betreffend. Als erstes sendet der Client ein Discover Paket. Dafür wird der Operationscode auf 1 gesetzt um eine request Paket zu signalisieren, die MAC-Adresse wird für die Physische Client Adresse verwendet, das Broadcast-Flag wird gesetzt und der DHCP-Message-Type wird auf Discover gesetzt. Aus diesen DHCP-Paket wird ein UDP-Paket erzeugt, mit den Quellport 68 und dem Zielport 67. Die Nachricht wird als Broadcast gesendet.
Auf diese Anfrage können einer oder mehrere DHCP Server antworten. Wenn kein bestimmter in der DISCOVER-Nachricht spezifiziert wurde, schicken alle Server in den jeweiligen Netzwerk ein "OFFER". Diese Nachricht wird ebenfalls als Broadcast gesendet, da der Client noch keine eigene IP zugewiesen hat. Das OFFER enhält einen Vorschlag mit einer IP Adresse in dem Feld

"Your IP Address". Anhand der "Transaction ID" und dem Feld CHADDR kann der Client die Offer eindeutig seiner Anfrage zuordnen.
Der Client antwortet auf das "OFFER" mit einer "REQUEST" Nachricht. Dafür wird die Option "Requested IP Address" gesetzt.
Auf das Request antwortet der Server wiederum mit einen ACK zu Bestätigung. Wenn der Client die Bestätigung erhält, prüft dieser ob die IP-Adresse schon genutzt wird, in dem er eine ARP Request für diese startet. Wird diese Beantwortet ist die Adresse schon in Benutzung. Ist dies nicht der Fall übernimmt der Client die Adresse. Die ACK Nachricht enthält die Lease-Zeit, welche angibt, wie lange eine IP gültig ist. Nachdem die hälfte der Lease-Zeit abgelaufen ist, sendet der Client eine weitere REQUEST-Nachricht. Da er zu diesem Zeitpunkt noch über eine gültige IP-Adresse verfügt sendet er die Nachricht nicht als Broadcast, sondern als Unicast direkt an den DHCP-Server, von dem er die IP zugewiesen bekommen hat. In dem Fall, das er auf die Request keine Antwort bekommen hat, startet er nach Ablauf der Lease Zeit mit einen DISCOVER.

Während dieses Vorgangs können neben der IP auch noch andere Informationen abgefragt werden. Wie beispielsweise die Adressen von DNS- und Zeitservern.

2.6 AMIDAR

3 Implementierung

Im Rahmen dieser Arbeit wurde ein TCP Stack für die API des AMIDAR Microprozessor entwickelt. Darüber hinaus wurde

3.1 Überblick

Vor Beginn dieses Projekts verfügte die AMIDAR Java API über Grundlegende Netzwerk Funktionen. Dazu gehört der Netzwerktreiber, ein IP-Stack mit ARP Funktionalität und ein UDP Stack. Neu geschrieben wurde im Rahmen dieses Projekts der Multithreading Fähige TCP Stack. Dieser wurde in die vorhandene Software Integriert. Desweiteren wurde der IP-Stack erweitert und optimiert.
Sowohl beim Senden als auch beim Empfangen von Datenpaketen greifen die einzelnen Module in einander über. Zum empfangen von Daten Prüft der Prozess des Netzwerktreibers ob neue Ethernet Frames vorliegen. Wenn das der Fall wird, eine Funktion im IP Stack aufgerufen, die die Ethernet Frames überprüft. Der IP-Stack unterscheidet die Pakete zwischen ARP und IP. ARP anfrage werden geprüft und gegebenenfalls beantwortet. Handelt es sich bei den Datagramm um ein IP-Paket, wird ein entsprechendes Objekt erzeugt und nach weiterer Überprüfung entweder an den UDP-Stack oder an den TCP-Stack übergeben. Die Stacks für TCP und UDP beinhalten jeweils einen Table mit den vorhandenen Verbindungen, die durch Ziel und Quell Port identifiziert werden können. Ihnen können gegebenenfalls die Erzeugten Pakete weiter gegeben werden, wo sie zwischengespeichert werden. Im Falle von UDP wird von den dieses die Payload ausgelesen, wenn auf "receive" Methode der UDP-Connection, von einen anderen Thread aufgerufen wird. Die TCP-Connections können jeweils in ihren eigenen Thread laufen, da eine Zeitnahe Verarbeitung der angekommen Pakete nötig ist um die Verbindung zu managen. In diesem Thread werden die angekommenen Pakete ausgewertet und die dazu entsprechenden Reaktionen berechnet und ausgeführt.
Wenn UDP Datenpakete versendet werden sollen, wird die SSend" Methode aufgerufen, die ein UDP-Paket erzeugt und diesen an den UDP-Stack weitergibt. Der wiederum ruft den erzeugt aus dem UDP-Paket ein IP-Paket. Mit dem die SSend" Methode des IP-Stacks aufgerufen wird. Die letztendlich einen Ethernetframe erzeugt und mit den EthernetWrapper den Sendevorgang startet.
Bei dem senden von Daten über TCP wird von der Anwendung der ebenfalls die SSend" Methode der TCP-Connection aufgerufen. Die werden dabei jedoch nicht sofort gesendet, sondern in einen Puffer zwischengespeichert, vorausgesetzt der aktuelle Status der Verbindung erlaubt das. Bei Ausführung des Threads der Verbindung, werden gegebenenfalls die zu übertragenden TCP-Pakete in IP Pakete umgewandelt und analog wie die UDP-Pakete versendet.

3.2 IP Stack

Der IP Stack erfüllt mehrere Funktionen, die für eine zuverlässige Netzwerkkommunikation benötigt werden. Dazu gehört, das Senden und Empfangen von IP-Pakten, als auch die Unterstützung der ARP Funktionalitäten.

3.2.1 Empfangen von IP Paketen

Die Methode readIpPakets() des IP-Stacks, die vom Netzwerktreiber Thread aufgerufen. In dieser werden in einer Schleife die angekommenen Ethernet Pakete eingelesen und die IP Pakete dazu erzeugt. Dabei werden im Zweifelsfall Fragmente von fragmentierten Paketen zwischengespeichert, bis diese vollständig sind.
Bei den so erzeugten IP-Paketen wird das darüber liegende Protokoll ausgelesen.
Damit UDP und TCP Pakete zugestellt werden können müssen die jeweiligen Stacks im IP-Stack registriert werden. Nach der Registrierung können angekommene Pakete den jeweiligen Stack zugeordnet werden. Dafür implementieren beide Stacks die Function "notificateByIpStack()"der eine Liste mit angekommenen UDP-Paketen übergeben wird.

Fragmentierung

IP Pakete können während der Übertragung fragmentiert werden. Um über Netzwerke mit einer zu niedrigen Maximalen Segment Größe übertragen zu werden. Diese werden vom Empfänger defragmentiert.
Fragmentierte Pakete können nach der dem erzeugen erkannt werden, in dem das entsprechende Flag ausgelesen wird. Die Flag gibt nicht explizit an, das diesen Paket Teil eines größeren fragmentierten Pakets ist, sondern dass sie ein Teil eines fragmentierten Pakets sind und weitere Pakete Fragmente folgen. Das bedeutet, dass das letzte Fragment eines Pakets nicht das Flag gesetzt hat, weswegen so lange fragmentierte Pakete zwischengespeichert sind, alle weiteren IP-Pakete auf eine Verbindung mit den Fragmentierten Paketen untersucht werden müssen.
Der IP Stack enthält eine Liste von für Fragmentierte Pakete, die Listen mit jeweils zusammengehörigen Fragmenten enthält. Wannn immer ein Paket ankommt, bei dem die "More Fragments" Flag gesetzt ist, oder die List mit Fragmentierten Paketen nicht leer ist, wird geprüft, ob dieses zu einem der fragmentierten Pakete gehört, und wird in diesem Fall an eine der Listen hinzugefügt. Die Zugehörigkeit wird dabei anhand der Identification des IP Pakets geprüft. Pakete die keine "More Fragments" Flag gesetzt haben, werden als nicht Fragmentierten, sofern sie keinen anderen fragmentierten Paket zugeordnet werden können.
Falls ein Paket ohne Flag einen anderen Fragmentierten Paket zugeordnet werden kann, ist dieses das letzte Fragment des ursprünglichen IP Pakets, mit dem das kombinierte Paket erzeugt werden kann.
Für diesen Zweck verfügt die Klasse IpPaket über die statische Methode "fuseFragmentedIpPackets()", die eine Liste von zusammengehörigen IP-Paket Fragmenten annimmt. Diese werden auf Kompatibilität geprüft, wobei auch die Größe der kombinierten Nutzlast berechnet wird. Die Fragmente müssen neben der identischen Identifikation über die selbe Quell und Ziel IP-Adresse verfügen. Des weiteren müssen die Angaben des Frame Offsets mit der Paketlänge Plausibel sein.

3.2.2 Senden von IP Paketen

3.3 TCP Stack

3.4 Zero Copy

Die TCP und IP Pakete werden jeweils durch eine entsprechende Klasse repräsentiert. Die Objekte dieser Klasse enthalten jeweils ein Array, dass Header und Nutzdaten des Pakets enthält. Beide Klassen verfügen über einen Konstruktor, der jeweils die andere der beiden Klassen als Parameter entgegen nimmt. So kann beim Senden aus einen TCP-Paket ein IP-Paket erzeugt werden und beim Empfangen aus einen IP-Paket ein TCP-Paket erzeugt werden. Dabei stellt das TCP-Paket die Nutzlast des IP-Pakets da. Um in diesen Fall lange Kopiervorgänge zu ersparen wird es vermieden ein neues Array anzulegen und die Daten rein zu kopieren. Anstelle dessen wird in der TCP-Paket Klasse im Array ein 20 Byte Puffer eingeplant, in dem der IP-Header später geschrieben werden kann. Bei der Erzeugung eines IP-Pakets aus einen TCP-Paket wird das im TCP-Paket erzeugte Array weiterverwenden und da die ersten 20 Byte leer sind, kann dort der IP Header geschrieben werden.

3.5 DHCP

3.6 Stream Sockets

4 Evaluation

4.1 Testaufbau

4.2 Funktionen

4.3 Performanz

5 Zusammenfassung

5.1 Fazit

5.2 Ausblick

Abkürzungsverzeichnis

CLB Configurable Logic Block

CPU central processing unit

ELF Executable and Linking Format

FPGA Field Programmable Gate Array

MC Microcontroller

CPU central processing unit

NCD Native Circuit Description Format

SoC System on Chip

Sph SpartanMC Hex Dateiformat

XDL Xilinx Design Language Format

Abbildungsverzeichnis

Literaturverzeichnis

[BUG] *BUG Xilinx XDL Programm*. http://forums.xilinx.com/t5/Spartan-Family-FPGAs/A-bug-for-Spartan-6/td-p/271026

[ELF] *Executable and linkable Format*. http://www.linux-kernel.de/appendix/ap05.pdf

[GNU] *GNU - make*. http://www.gnu.org/software/make/manual/make.html

[OSE] *Open Source ELF-Library*. https://www.hpi.uni-potsdam.de/hirschfeld/trac/SqueakMaxine/browser/com.oracle.max.elf?rev=1d81ff0c9a2a8a101fe33f4ac1495decc6147517#src/com/oracle/max/elf

[UCS] *XDL Use Case Senarios*. http://www.cs.indiana.edu/hmg/le/project-home/xilinx/ise_5.2/help/data/xdl/xdl-ucs-ext.html

[XDL] *Xilinx Design Language*. http://www.cs.indiana.edu/hmg/le/project-home/xilinx/ise_5.2/help/data/xdl/xdl.html