

# Project Report

## Studienarbeit

VerfasserIn: Natalia Pavlik

Matrikel-  
nummer: 03691812

VerfasserIn: Nurdan Eren

Matrikel-  
nummer: 47720013

Dozent: Prof. Dr. Soceanu

Abgabe: 10. Januar 2017

## Summary

Die im Folgenden detaillierter beschriebene Studienarbeit beinhaltet die Implementierung eines Trivial File Transfer Protocols (TFTP). Die Implementierung des TFTP soll, basierend auf dem User Datagram Protocol (UDP), in Java erfolgen. Die wesentlichen Bestandteile der Implementierung umfassen die zwei Hauptprozesse Client und Server.

Aufgabe des Clients ist das Akzeptieren der Konsoleingaben, um diese im Terminal anzeigen zu können. Diese Eingaben soll der Client anschließend per TFTP an den Server senden. Zudem soll er vom Server gesendete Eingaben empfangen und im Terminal anzeigen können.

Der Server hingegen ist für den Empfang der Eingaben des Clients sowie für den Versand der Eingaben an den Client zuständig.

Beim Versand und Empfang der Daten können unterschiedliche Fehler auftreten. Die Behandlung dieser Fehler war ein weiterer Bestandteil dieser Studienarbeit.

---

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
<b>2</b>	<b>Analyse</b>	<b>3</b>
2.1	<i>Projektgestaltung</i>	3
2.2	<i>Implementierung</i>	4
2.3	<i>Traffic Analyse</i>	5
2.3.1	Fehlerfreier RRQ (Wireshark Analyse)	6
2.3.2	Fehlerfreier WRQ (Wireshark Analyse)	6
2.3.3	Zuerst wird der Client und anschließend der Server gestartet (Fallbeispiel)	7
2.3.4	Server-Port wird im Client geändert (Fallbeispiel)	8
2.3.5	Client und Server wurden bereits gestartet und werden erneut gestartet	8
2.3.6	ACK kommt nicht an/ Paket mit gleicher Blocknummer bereits erhalten	9
2.3.7	Ein oder mehrere Pakete einer Data Message sind verloren gegangen	10
<b>3</b>	<b>Fazit</b>	<b>10</b>
	<b>Abkürzungsverzeichnis</b>	<b>IV</b>
	<b>Abbildungsverzeichnis</b>	<b>IV</b>
	<b>Literaturverzeichnis</b>	<b>V</b>

# 1 Einleitung

Das Trivial File Transfer Protocol ist ein einfaches Protokoll zur Datenübertragung. Der Hauptbestandteil dieser Studienarbeit ist die Implementierung eines TFTP basierend auf dem UDP. Eigentlich setzt das TFTP meistens auf UDP auf, allerdings wird hier die Verwendung anderer Transportprotokolle nicht ausgeschlossen.

Die Basis des Projekts sieht folgendermaßen aus:

- Anwendungsschicht: TFTP
- Transportschicht: UDP (Server-Port: 2017)
- Netzwerkschicht: IP (127.0.0.1 – Localhost)

Um mit der Implementierung beginnen zu können, musste zunächst nicht nur die Funktionsweise eines TFTP, sondern auch des UDP grundlegend verstanden werden. Hierfür wurde Literatur und Onlinerecherche betrieben.

Bei dem User Datagram Protocol handelt es sich um ein verbindungsloses und nicht-zuverlässiges Netzwerkprotokoll. Es ermöglicht bei Anwendungen den Versand von Datagrammen in IP-Rechnernetzen. So kann nicht gewährleistet werden, dass Pakete in der richtigen Reihenfolge oder überhaupt ankommen. Daher sollte eine Anwendung, die auf UDP basiert, ein Verbindungsmanagement besitzen, um mit solchen Fehlern (z.B. Paketverlust) umgehen zu können. Bei TFTP kann dies mithilfe diverser Funktionen umgesetzt werden, auf welche im Folgenden genauer eingegangen wird.

Die Kommunikation bei UDP läuft über Datagram-Sockets ab. Diese können mithilfe der Adress-Informationen, bspw. IP oder Portnummer, eine Verbindung zu demselben oder anderen Computern im Netzwerk herstellen können. In der Studienarbeit wären es jeweils Client und Server.

Das TFTP recht einfach implementierbar und nicht sonderlich umfangreich. Es unterstützt nur das Schreiben und Lesen von Dateien auf einem Server. Somit kann das TFTP bspw. keine Authentifizierung, Verschlüsselungen durchführen oder Verzeichnisinhalte auflisten. Daher wird das TFTP für automatisierte Übertragungen oder Konfigurationen auf Rechnern in sicheren Umgebungen verwendet.

Beim TFTP-Protocol wird in der Regel der Server Port 69 (well-known) verwendet, in der Implementierung wurde der Port 2018 angegeben, da nur lokal gearbeitet wurde. So kann der Client auf diesem Port zwar anfragen, der Server aber mit dem angegebenen Port als Quell-

Port arbeiten. Auf diese Weise ist der Port unmittelbar nach dem Verbindungsaufbau wieder verfügbar bzw. frei. Weitere Kommunikation findet über TID (Transfer Identifier) statt. Diese liegen im Bereich von 1024 und 65535.

Im TFTP gibt es fünf Pakettypen, die mit einem 2-Byte langem Feld gekennzeichnet sind. Im Folgenden ist die Auflistung der Pakettypen gefolgt von einem beispielhaften Aufbau eines Pakets:

Opcode	Operation	
1	RRQ	Read Request
2	WRQ	Write Request
3	DATA	Data
4	ACK	Acknowledgement
5	ERROR	Error

**Abbildung 1: Pakettypen**

2 Bytes	RRQ	1 Byte		1 Byte
RRQ=1	„File Name“	0	„Mode“	0

**Abbildung 2: Beispielhafter Aufbau eines Pakets**

Bei einer Request Nachricht wird zudem der Übertragungsmodus benötigt:

- 8-Bit ASCII-Mode
- Octet = 8-Bit Daten
- Mail = Datei wird Empfänger gesendet

Wie in diesem Kapitel zuvor beschrieben, wird aufgrund des UDP richtiges Verbindungsmanagement benötigt. Diese Verbindungskontrolle muss eigens implementiert werden, um sichergehen zu können, dass die Pakete auch richtig übertragen werden. Nachfolgend werden Funktionen zur Gewährleistung einer sicheren Übertragung gelistet.

- Alle DataMessages sind mit einer Blocknummer (BN) nummeriert. Diese ist aufsteigend und wird wie bei zuvor beschriebenen Pakettypen, durch ein 2-byte langes Feld gekennzeichnet.
- Sämtliche bereits erhaltenen DataMessages werden mit ACK (Acknowledgement) bestätigt. Diese beinhaltet die Blocknummer der erhaltenen DataMessage.
- Das Senden und Empfangen funktioniert nach dem „Stop & Wait“ – Prinzip. Die Nächste Nachricht wird erst dann gesendet, sobald der Empfang der vorherigen bestätigt wurde.

- Die Fehlerbehandlung erfolgt über einen „Waiting Timer“ und „Repeating Counter“. Bei dem „Waiting Timer“ wird geprüft, ob die Bestätigung einer Nachricht innerhalb der vorgegebenen Zeit erfolgt. „Repeating Counter“ legt fest, wie oft die Nachricht erneut versendet werden soll, falls innerhalb des gesetzten Timers keine Bestätigung eingeht.
- Mithilfe weiterer Funktionen, kann bspw. überprüft werden, ob bereits eine Nachricht mit derselben Blocknummer existiert oder ein Paket verloren gegangen ist. (Darauf wird im Kapitel Error Handling genauer eingegangen)

In TFTP gibt es folgende Fehlercodes:

0	Not defined
1	File not found
2	Access violation
3	Disk full or allocation needed
4	<b>Illegal TFTP Operation</b>
5	Unknown transfer ID
6	File already exists
7	No such user

**Abbildung 3: Fehlercodes TFTP**

In der Implementierten Lösung wurden die Fehlercodes 1 (no ACK received), 4 (too many resending requests) und 6 (a message with the blocknumber already exists).

Der Start des TFTP erfolgt mit einer clientseitigen Read- oder Write-Request an den Server. Auf die RRQ reagiert der Server mit dem Senden des ersten Datenpakets. Eine WRQ erhält eine ACK. Nach der Acknowledgement kann der Client Daten an den Server senden.

## 2 Analyse

### 2.1 Projektgestaltung

Im Rahmen der Studienarbeit sollte ein Trivial File Transfer Protocol (TFTP) in JAVA programmiert werden. Hierfür wurde die Entwicklungsumgebung Eclipse verwendet. Sämtliche Ein- und Ausgaben der Implementierung erfolgen über die Konsole in Eclipse. Die Analyse der Datenverbindung wurde mithilfe der Loopback-Funktion in Wireshark umgesetzt. Für eine erfolgreiche Zusammenarbeit an der Umsetzung und zur Versionsverwaltung wurde Git (GitHub) eingesetzt.

## 2.2 Implementierung

Die Implementierung des TFTP in Java ging aus der wesentlichen Funktionsweise eines TFTP (basierend auf UDP) hervor. Statt des well-known Ports 69 wurde ein frei ausgewählter Port 2018 verwendet. Die erarbeitete Lösung umfasst sämtliche Projektvorgaben und –komponenten und ermöglicht daher die vorgegebene Verhaltensweise.

Nach dem Start des Servers und des Clients, hat der User die Wahl zwischen einem WRQ und einem RRQ. Die Eingabe durch den Nutzer erfolgt direkt in der Eclipse-Konsole. Fällt die Wahl auf einen RRQ, sendet der Server einen, in der Implementation vorgegebenen, Text an den Client. Der Textabschnitt dient lediglich zur beispielhaften Visualisierung des RRQ der umgesetzten Lösung.

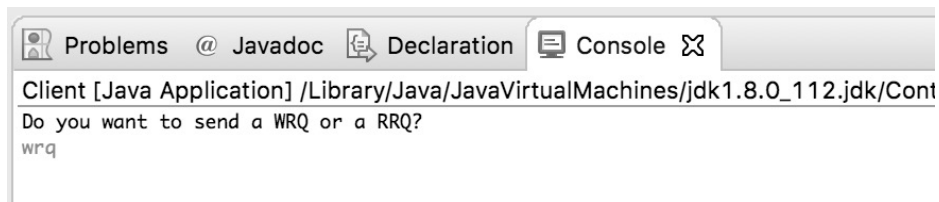
Wird ein WRQ gewählt, wird der User aufgefordert die zu versendende Nachricht in der Konsole einzugeben. Nach eingegangener Bestätigung (ACK), wird die Nachricht an den Server gesendet. Nach erfolgreicher Übertragung sendet der Client automatisch einen RRQ an den Server und erhält die zuvor eingegebene Nachricht vom Server zurück. Sämtliche eingegangenen Data Messages werden mit ACK bestätigt. Diese Lösung entspricht nicht dem eigentlichen Verhalten eines üblichen TFTP. Dieses basiert lediglich auf Lese- und Schreibabfragen von Dateien.

Das Programm enthält zudem einen Timer und Wiederholzähler. Bei dem Senden einer Data Message wird der Timer auf 3 Sekunden gesetzt. Geht in dieser Zeit eine Bestätigung ein, so wird der Timer wieder auf 0 gesetzt. Wenn in dieser Zeit jedoch keine Bestätigung ankommt, erfolgt ein erneuter Sendeversuch des Pakets. Nach drei Versuchen, wird das Senden der Message abgebrochen. Darüber wird man ebenfalls über die Ausgabe in der Konsole informiert.

Zum Erkennen und zum Abfangen doppelt mehrmals gesendeter Messages bzw. Pakete, werden im Programm alle bereits enthaltenen Pakete in einer Liste festgehalten. Zum Identifizieren verlorengegangener Pakete wird die Kennzahl „erwartete Blocknummer“ verwendet.

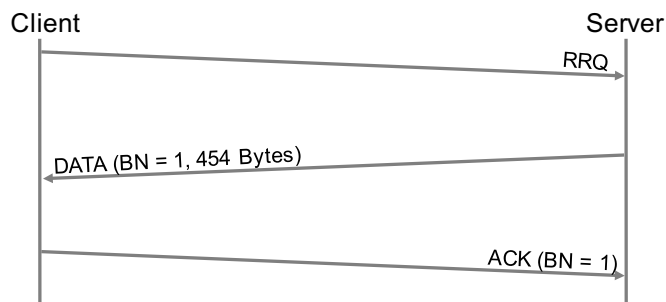
Die Folgenden Abbildungen sowie Konsolen- und Wireshark-Screenshots zeigen den fehlerfreien Ablauf der implementierten Lösung. Auf die Fehlerbehandlung wird im Kapitel 3 eingegangen.

### 1. Wahl zwischen Cases: RRQ und WRQ



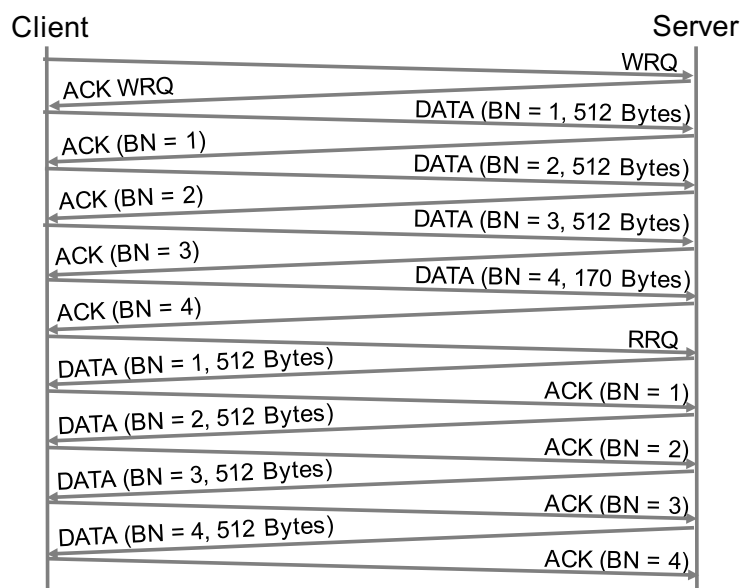
**Abbildung 4: WRQ oder RRQ - Wahlmöglichkeiten des Users**

### 2. Case RRQ (fehlerfrei) - (im Kapitel 2.3.1 genauer beschrieben)



**Abbildung 5: Fehlerfreier RRQ**

### 3. Case WRQ (fehlerfrei) (im Kapitel 2.3.2 genauer beschrieben)



**Abbildung 6: Fehlerfreier WRQ**

## 2.3 Traffic Analyse

Wireshark bietet die Möglichkeit zahlreiche Protokolle aufzuzeichnen. Aufgrund der Relevanz für die Studienarbeit, beschränkt sich die Anzeige mithilfe der getroffenen Einstellungen auf TFTP.



Da die Wireshark-Aufzeichnung durch Loopback erfolgt und auf einem Localhost läuft, ist die Source und Destination IP-Adresse 127.0.0.1. Die Länge einer Nachricht setzt sich zusammen aus 14 Bytes Ethernet (DataLink), 20 Bytes IP, 8 Bytes UDP, 0-516 Bytes TFTP (2 Bytes Opcode, 2 Bytes blocknumber, Rest sind die Daten (bzw. die eigentliche Nachricht)). Der Übertragungsmodus des Protokolls erfolgt im Octet (8-Bit-Dateien).

### 2.3.1 Fehlerfreier RRQ (Wireshark Analyse)

Nach dem Programmstart, hat der User die Wahlmöglichkeit zwischen RRQ und WRQ. Fällt die Wahl auf einen RRQ, sendet der Client den Read Request an den Server. Da bei der Implementierung nicht mit Dateien gearbeitet wird, sendet der Server direkt eine im Code hinterlegte Beispielnachricht an den Client (s. Kapitel 2). Es handelt sich hierbei um eine „Lorem Ipsum“ - Nachricht in der Message - Klasse. Nach Erhalt der Nachricht, wird vom Client eine Bestätigung an den Server gesendet.

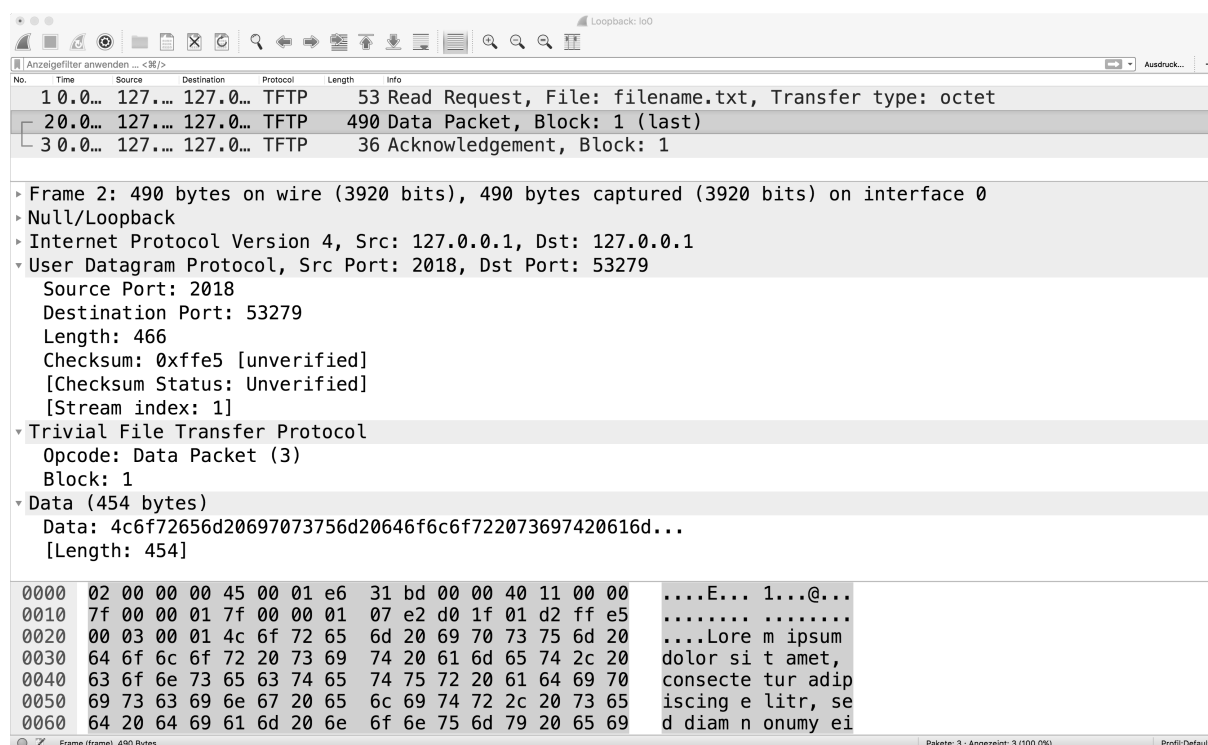
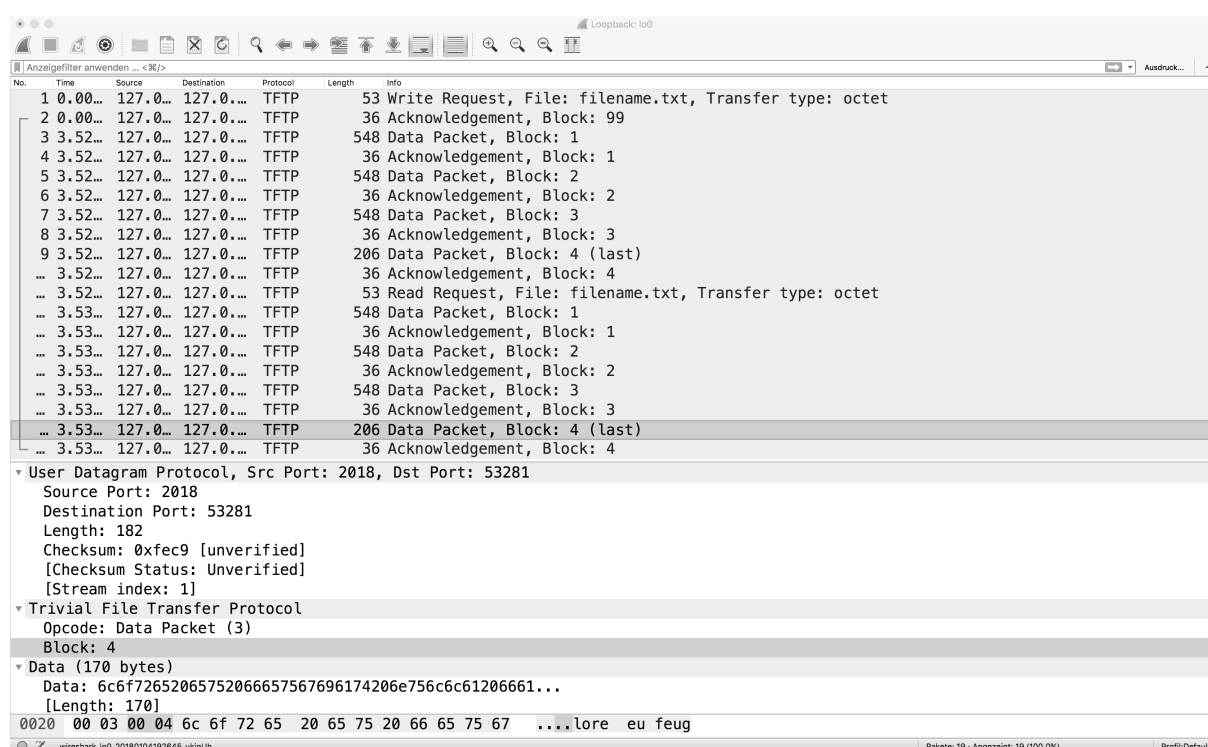


Abbildung 7: Fehlerfreier RRQ - Wireshark Analyse

### 2.3.2 Fehlerfreier WRQ (Wireshark Analyse)

Erneut erhält der Nutzer die Möglichkeit zwischen RRQ und WRQ zu wählen. Wird ein WRQ gewählt, so sendet der Client einen Write Request an den Server. Nach einer Bestätigung des Servers (ACK), kann der Client Daten an den Server schicken. In diesem Fall gibt der User

eine Nachricht in die Konsole ein. Übersteigt die Nachricht 512 Bytes, wird diese in mehrere Blöcke aufgeteilt. Wenn der Block weniger als 512 Bytes enthält, erkennt das Programm, dass es sich hierbei um den letzten oder einzigen Block handelt. Im seltenen Fall, dass der (letzte) Block genau 512 Bytes beträgt, kommt ein weiterer Block mit 0 Bytes hinzu. Jeder übertragene Block wird vom Server bestätigt (ACK). Sobald das Senden der Nachricht abgeschlossen ist, sendet der Client automatisch einen RRQ an den Server. Anschließend sendet der Server die zuvor eingegebene Nachricht an den Client zurück. Diese wird je nach Umfang, erneut in Blöcken versendet. Auch hier erfolgt nach jedem Block clientseitig eine Bestätigung (ACK). Nach vollständiger Übermittlung der Blöcke, wird die komplette Nachricht in der Konsole angezeigt.



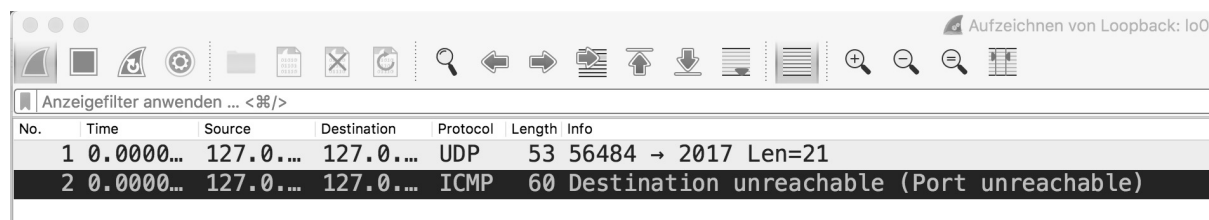
**Abbildung 8: Fehlerfreier WRQ - Wireshark Analyse**

### 2.3.3 Zuerst wird der Client und anschließend der Server gestartet (Fallbeispiel)

In der erarbeiteten Lösung hat die umgekehrte Reihenfolge keinen Einfluss auf den Programmablauf bzw. die Funktionsweise des Protokolls. Der einzige Unterschied liegt bei der Konsole. Da man den Server in dem Fall zum Schluss starten würde, bleibt die Server-Konsole im Vordergrund. So müsste man zunächst in die Konsole des Clients wechseln, um die für den Workflow notwendigen Texteingaben machen zu können. Diesen Schritt kann man sich sparen, wenn man zunächst den Server und anschließend den Client startet.

### 2.3.4 Server-Port wird im Client geändert (Fallbeispiel)

Der Client sucht dann die Verbindung zu dem Port 2017 (geändert). Da der Server unter dem Port 2018 erreichbar ist, kommt in Wireshark die Meldung „Destination unreachable“.



Aufzeichnen von Loopback: lo0

Anzeigefilter anwenden ... <=>/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.0000...	127.0.0.1	127.0.0.1	UDP	53	56484 → 2017 Len=21
2	0.0000...	127.0.0.1	127.0.0.1	ICMP	60	Destination unreachable (Port unreachable)

Abbildung 9: Fallbeispiel - Serverport wird geändert

### 2.3.5 Client und Server wurden bereits gestartet und werden erneut gestartet

Wenn der Server nach dem Start erneut gestartet wird, wird eine BindException: „Address already in use.“ ausgelöst bzw. geworfen. Dies soll verdeutlichen, dass der Server bereits gestartet wurde und nicht erneut gestartet werden kann.

8	7.839595	127.0.0.1	127.0.0.1	TFTP	53	Write Request, File: filename.txt, Transfer type: octet
9	7.839720	127.0.0.1	127.0.0.1	TFTP	36	Acknowledgement, Block: 99
10	8.999919	127.0.0.1	127.0.0.1	TFTP	38	Data Packet, Block: 1 (last)
11	9.000160	127.0.0.1	127.0.0.1	TFTP	548	Error Code, Code: File already exists, Message: A message with the same blocknumber was already received and will therefore be discarded.

Abbildung 10: Fallbeispiel - Client und Server wurden bereits gestartet und werden erneut gestartet - Fall 1

Wenn der Server und Client gestartet wurden und der Client bei laufendem Server ein weiteres Mal gestartet wird, gibt es auch da zwei unterschiedliche Möglichkeiten:

- Soll ein RRQ gesendet werden, ist dies kein Problem. Der im Code fest implementierte Text wird an den Client gesendet.
- Soll jedoch ein WRQ gesendet werden, obwohl bei dem laufenden Server vorher ein WRQ gesendet wurde, entsteht ein Fehler. Der Server hat die Blocknummern (BN) bereits in einem Array gespeichert. Der Neustart des Clients setzt diese nur clientseitig zurück und kann somit von vorne (mit einer 1) beginnen. Deshalb erhält der User eine Error-Nachricht, dass ein Paket mit derselben Blocknummer (BN) bereits empfangen wurde.

8	7.839595	127.0.0.1	127.0.0.1	TFTP	53	Write Request, File: filename.txt, Transfer type: octet
9	7.839720	127.0.0.1	127.0.0.1	TFTP	36	Acknowledgement, Block: 99
10	8.999919	127.0.0.1	127.0.0.1	TFTP	38	Data Packet, Block: 1 (last)
11	9.000160	127.0.0.1	127.0.0.1	TFTP	548	Error Code, Code: File already exists, Message: A message with the same blocknumber was already received and will therefore be discarded.

Abbildung 11: Fallbeispiel - Client und Server wurden bereits gestartet und werden erneut gestartet - Fall 2

### 2.3.6 ACK kommt nicht an/ Paket mit gleicher Blocknummer bereits erhalten

Wenn client- oder serverseitig eine Nachricht (Data Message) versendet wird, wird eine Bestätigung (ACK) erwartet. Sobald aber keine Bestätigung kommt, muss das vom TFTP behandelt werden können. Um solche Fehler festzustellen, arbeitet TFTP mit Timern und Wiederholzählern.

Bei der Implementierung wurde das folgendermaßen umgesetzt:

Sobald eine Nachricht gesendet wird, wird ein Timer gestartet (client- und serverseitig). Der Timer ist auf drei Sekunden gesetzt. Sollte in dieser Zeit keine Bestätigung ankommen, springt das Programm direkt zur Methode `handleMissingAck()`. So werden noch drei weitere Sendeversuche (Wiederholzähler = 3) unternommen, bevor das Programm abbricht.

Hinzu kommt das Problem, dass bei erneutem Versenden der Nachricht (`handleMissingAck()`) dieselbe Blocknummer wieder verwendet wird. Dies muss clientseitig berücksichtigt und abgefangen werden. Dafür wurde ein Array eingesetzt, welches die Blocknummern der bereits empfangenen Nachrichten enthält. Bei Empfangen einer neuen Nachricht, wird die Blocknummer mit den bereits enthaltenen verglichen. Existiert bereits die Blocknummer im Array, wird die Nachricht ignoriert. Zusätzlich wird eine Fehler-Message an den Server gesendet, dass ein Paket mit der Blocknummer bereits vorhanden war.

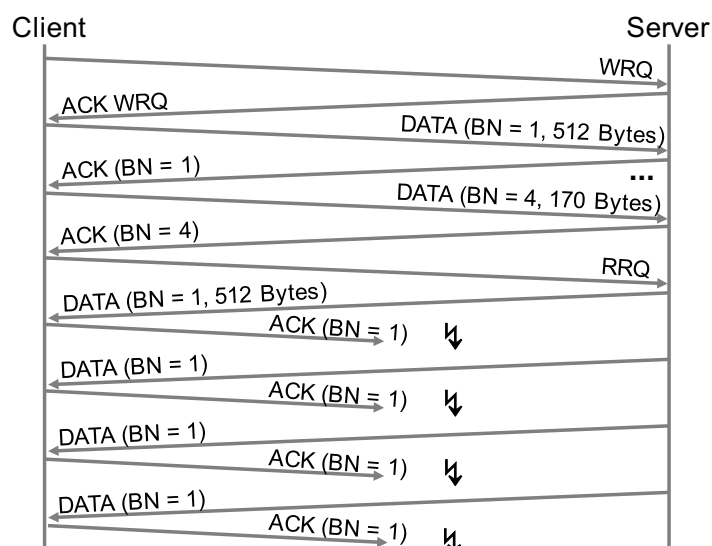


Abbildung 12: ACK kommt nicht an/ Paket mit gleicher BN erhalten

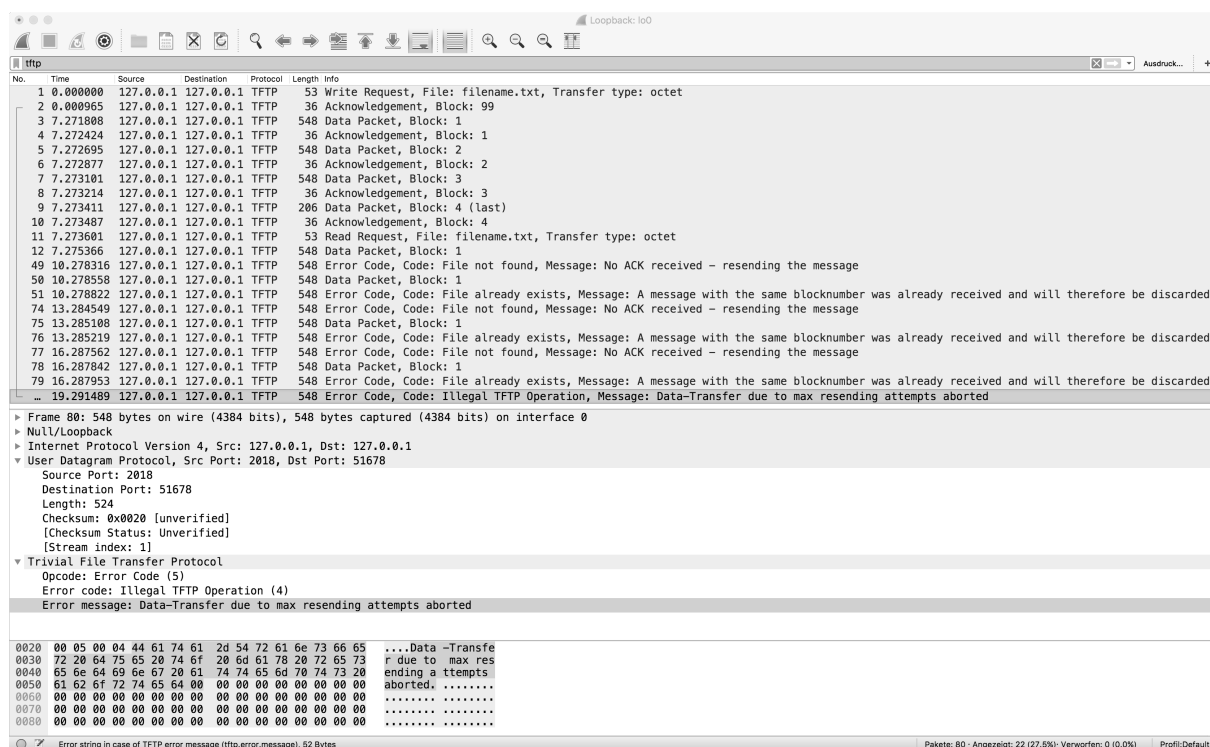


Abbildung 13: ACK kommt nicht an/ Paket mit gleicher BN erhalten - Wireshark Analyse

## 2.3.7 Ein oder mehrere Pakete einer Data Message sind verloren gegangen

Es kann durchaus dazu kommen, dass ein Datenpaket verloren geht. Um diesen Fall zu erkennen, gibt es die Variable, welche die erwartete Blocknummer beinhaltet. So kann bei Empfangen einer Nachricht die erhaltene Blocknummer mit der erwarteten verglichen werden. Wenn die Nummern nicht übereinstimmen, wird eine Error-Message versendet. Dieser Fehler wurde in der Implementierung nicht weiter behandelt.

## 3 Fazit

Während der Studienarbeit wurden diverse Ansätze ausprobiert. Die finale Lösung, die Implementierung eines TFTP, wurde unter Berücksichtigung der Vorgaben erarbeitet.

Im Folgenden werden die Vorgaben gelistet, welche im Rahmen der Studienarbeit umgesetzt werden mussten.

- Implementierung eines Client-Prozesses
  - o Nutzereingaben
  - o Nutzereingaben an Server senden (TFTP)

- Eingaben vom Server empfangen
- Implementierung eines Server-Prozesses
  - Nutzereingaben empfangen (TFTP)
  - Eingaben an Client senden (TFTP)
- Testen der Implementierung mit zwei Prämissen (fehlerhaft und fehlerfrei)
- Stichproben des gemessenen Traffics mittels Wireshark zwischen Client und Server

Die durchgeführte Analyse verdeutlicht, dass sämtliche Zielvorgaben umgesetzt und erfüllt wurden. Das Senden und Empfangen von Daten zwischen Client und Server funktioniert in beiden Richtungen einwandfrei. Ebenfalls wurden die in den Fallbeispielen gelisteten Fehler berücksichtigt und behandelt. Die Implementierung wurde in Wireshark getestet. Die Analyse fiel sowohl im Normal- als auch im Fehlerfall wie erwartet aus. Dies belegen die Screenshots der letzten Kapitel.

Das fertiggestellte Programm weist zwar keine Fehler auf, allerdings hat es noch Verbesserungspotenzial. Beispielsweise hätten noch folgende Punkte umgesetzt werden können:

- Bei ERROR-NUMBER 4: Package got lost: Hier hätte man nach der Error-Ausgabe, den Error weiter behandeln können, bzw. einen weiteren Programmablauf einbauen können. Es hätte einen weiteren Stream geben können, in welchem der User aufgefordert wird seine Nachricht erneut einzugeben.
- Nach einer WRQ und der automatischen RRQ (nach der abgeschlossenen Nachricht), hätte man das Programm um weitere Interaktionen erweitern können.

Aufgrund des festgelegten Abgabetermins und des trotz Verschiebung entstandenen Zeitmangels, war es leider nicht mehr möglich Erweiterungen an der Lösung vorzunehmen.

## Abkürzungsverzeichnis

ACK	Acknowledge
BN	Blocknumber
Bspw.	Beispielsweise
RRQ	Read Request
TID	Transfer Identifier
TFTP	Trivial File Transfer Protocol
UDP	User Datagram Protocol
WRQ	Write Request

## Abbildungsverzeichnis

Abbildung 1: Pakettypen .....	2
Abbildung 2: Beispielhafter Aufbau eines Pakets.....	2
Abbildung 3: Fehlercodes TFTP.....	3
Abbildung 4: WRQ oder RRQ - Wahlmöglichkeiten des Users .....	5
Abbildung 5: Fehlerfreier RRQ .....	5
Abbildung 6: Fehlerfreier WRQ .....	5
Abbildung 7: Fehlerfreier RRQ - Wireshark Analyse .....	6
Abbildung 8: Fehlerfreier WRQ - Wireshark Analyse .....	7
Abbildung 9: Fallbeispiel - Serverport wird geändert.....	8
Abbildung 10: Fallbeispiel - Client und Server wurden bereits gestartet und werden erneut gestartet - Fall 1 .....	8
Abbildung 11: Fallbeispiel - Client und Server wurden bereits gestartet und werden erneut gestartet - Fall 2 .....	8
Abbildung 12: ACK kommt nicht an/ Paket mit gleicher BN erhalten .....	9
Abbildung 13: ACK kommt nicht an/ Paket mit gleicher BN erhalten - Wireshark Analyse ....	10

## Literaturverzeichnis

Kurose, J. (2014). *Computernetzwerke – Der Top-Down-Ansatz*. Hallbergmoos: Pearson Deutschland GmbH.

Prof. Dr. A. Soceanu (2017). *Skripte aus der Vorlesung*

## Internetquellen

- <http://einstein.informatik.uni-oldenburg.de/rechnernetze/tftp.htm> (zuletzt aufgerufen am: 09.01.2018)
- <http://www.webschmoeker.de/grundlagen/udp-user-datagram-protocol/> (zuletzt aufgerufen am 07.01.2018)
- <https://javapapers.com/java/java-nio-tftp-client/> (zuletzt aufgerufen am 09.01.2018)
- <https://javapapers.com/java/java-tftp-client/> (zuletzt aufgerufen am 10.01.2018)
- <https://github.com/jherrlin/tftp-server-java/tree/master/src/main/java/se/lnu/handlers> (zuletzt aufgerufen am 10.01.2018)
- <https://github.com/vilius/tftp/tree/master/java> (zuletzt aufgerufen am 10.01.2018)
- <http://www.java2s.com/Code/Java/Network-Protocol/AsimpleJavatftpclient.htm> (zuletzt aufgerufen am 10.01.2018)



## Erklärung

Die angegebenen Verfasser erklären durch ihre Unterschrift, dass sie diese Studienarbeit selbstständig erstellt, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet haben.

München, 17.12.2017



---

Unterschrift Natalia Pavlik



---

Unterschrift Nurdan Eren