



**Vorlesung Rechnernetze**

**Theorieübung Socketbefehle und Pakete**

**Prof. Dr. Dirk Staehle**

Die Abgabe erfolgt durch Hochladen der Lösung in Moodle und exemplarisches Vorrechnen in der Laborübung.

**Bearbeitung in Zweier-Teams**

**Team-Mitglied 1:**

**Team-Mitglied 2:**

In dieser Aufgabe geht es darum, den Zusammenhang von Socket-Befehlen und übertragenen Pakete zu verstehen. Dazu ist in Abbildung 1 ein Python-Code dargestellt. Dieser Python Code wird auf einem Rechner zweimal mit unterschiedlichen Konfigurationen nacheinander ausgeführt. Bei der ersten Ausführung (A) ist My\_IP=127.0.0.2 und Remote\_IP=127.0.0.1. Bei der zweiten Ausführung (B) ist My\_IP=127.0.0.1 und Remote\_IP=127.0.0.2.

Ein Mitschnitt der ausgetauschten Pakete in Tabelle 1 dargestellt.

1. Tragen Sie in die erste Spalte von **Fehler! Verweisquelle konnte nicht gefunden werden.** ein, welche Code-Zeile bei welcher Ausführung des Codes (A oder B) diese Paketübertragung bewirkt hat, sofern diese explizite Zuordnung möglich ist d.h. wenn die Übertragung des Pakets umittelbar durch eine Code-Zeile ausgelöst wird.
2. Tragen Sie in die zweite Spalte von **Fehler! Verweisquelle konnte nicht gefunden werden.** ein, welche „blockierende“ Befehlszeile bei welcher Ausführung des Codes (A oder B) erfolgreich (ohne Fehler) „beendet“ wird, wenn das Paket empfangen wird. Auch hier soll natürlich nur dann ein Eintrag erfolgen, wenn der Empfang dieses Pakets die „Fertigstellung“ eines Socket-Befehls bewirkt.

Hinweis: Die Eintragungen in die Tabelle sollen also beispielweise die Form A10 haben, wenn Programmzeile 10 der ersten Ausführung (A) des Codes die Übertragung des Pakets bewirkt hat, oder B10 wenn dementsprechend die zweite Ausführung (B) des Codes die Übertragung des Pakets bewirkt hat.

Abbildung 1 Code Listing

Ausführung (A) ist My\_IP=127.0.0.2 und Remote\_IP=127.0.0.1

Ausführung (B) ist My\_IP=127.0.0.1 und Remote\_IP=127.0.0.2

1. import socket
2. socket.setdefaulttimeout(30)
3. My\_IP = '127.0.0.2' / '127.0.0.1'
4. My\_PORT = 50000
5. Remote\_IP= '127.0.0.1' / '127.0.0.2'
6. Remote\_PORT=50000
7. def start\_task(sock,message):
8. sock.send(message.encode('utf-8'))
9. msg=sock.recv(1024)
10. sock.close()
11. def start\_server():
12. sock=socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)
13. sock.bind((My\_IP, My\_PORT))
14. sock.listen(1)
15. try:
16. conn, addr = sock.accept()
17. start\_task(conn,"Thx for connecting!!!")
18. except socket.timeout:
19. pass
20. sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)
21. try:
22. sock.connect((Remote\_IP, Remote\_PORT))
23. start\_task(sock,"Thx for accepting!!!");
24. except socket.error:
25. start\_server(

|  |  |  |
| --- | --- | --- |
| Ausführung A | Ausführung B |  |
| A26 |  | Socket Connect to Server |
| A28 |  | Socket Error (Server not running) |
| A29 |  | Request start Server |
| A16 |  | Start Server on IP: 127.0.0.2 |
| A19 |  | Waiting on Clients to Connect |
|  | B26 | Client (127.0.0.1) connect to Server 127.0.0.2 / 127.0.0.2 |
|  | B27/10 | Start Task and Send Msg “Thx for accepting” to Server [Len: 20] |
|  | B11 | Waiting on Server response |
| A20/10 |  | Start Task and Send Msg “Thx for Connecting” to Client [Len: 21] |
|  | B12 | Close Client Socket |
| A11 |  | Waiting on Client resond |
| A21 |  | Waiting till Timeout or Connections close |
|  |  |  |

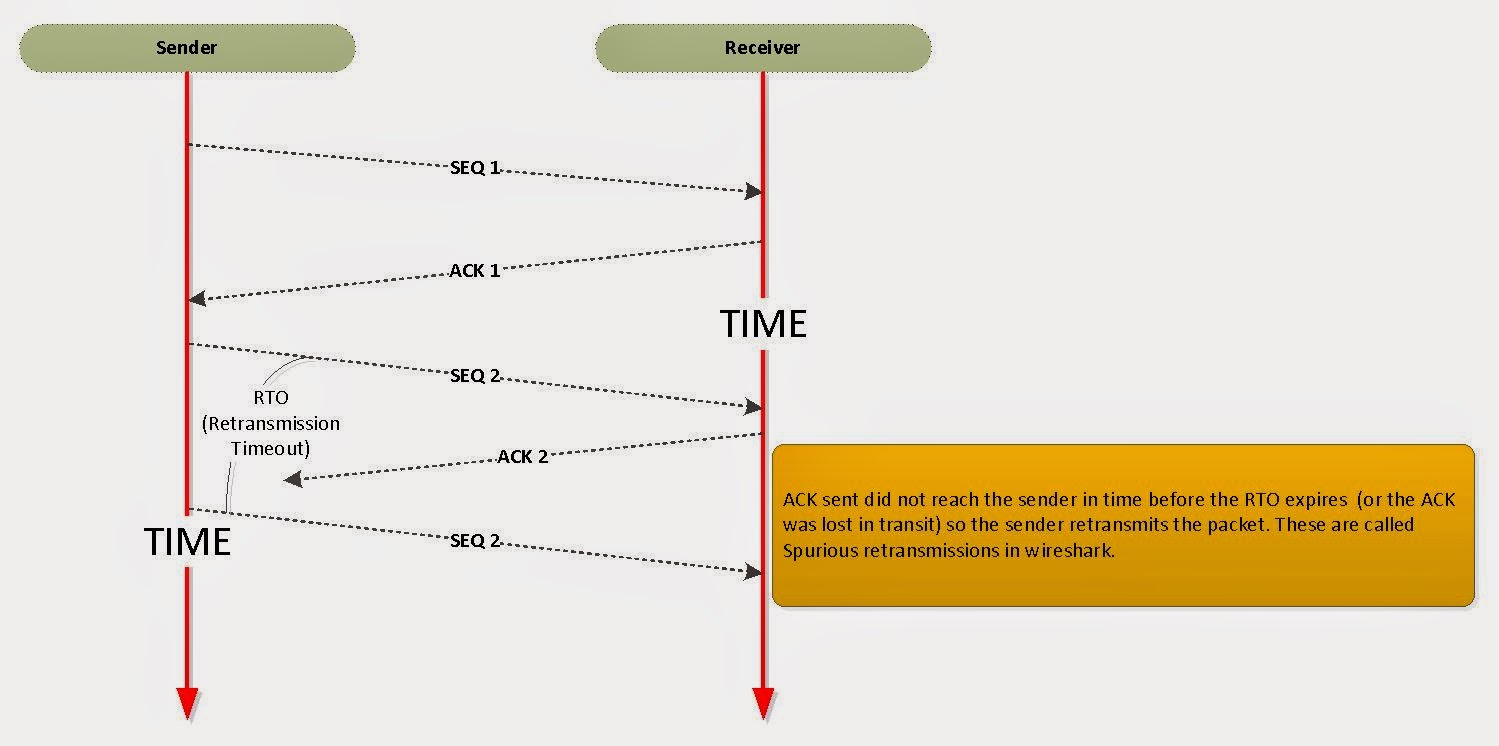


Tabelle 1: TCPdump

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Info | Übertragung  ausgelöst von  Programmzeilem | Empfang bewirkt  "Fertigstellung " von Prog. Zeile | No. | Time | Source | Destination | Protocol | Length | Src Port | Dst Port | Info |
| C. can’t connect, start Server | **A26** |  | 1 | 1,340 | 127.0.0.2 | 127.0.0.1 | TCP | 52 | 56835 | 50000 | 56835 > 50000 [SYN] Seq=0 |
|  | **???** |  | 2 | 1,340 | 127.0.0.1 | 127.0.0.2 | TCP | 40 | 50000 | 56835 | 50000 > 56835 [RST, ACK] Seq=1 Ack=1 |
| Retry Conn. (now S response) | **A26** |  | 3 | 1,842 | 127.0.0.2 | 127.0.0.1 | TCP | 52 | 56835 | 50000 | [TCP Spurious Retransmission] 56835 > 50000 [SYN] Seq=0 |
|  | **???** |  | 4 | 1,842 | 127.0.0.1 | 127.0.0.2 | TCP | 40 | 50000 | 56835 | 50000 > 56835 [RST, ACK] Seq=1 Ack=1 |
| Retry Conn. (now S response) | **A26** |  | 5 | 2,342 | 127.0.0.2 | 127.0.0.1 | TCP | 48 | 56835 | 50000 | [TCP Spurious Retransmission] 56835 > 50000 [SYN] Seq=0 |
|  | **???** |  | 6 | 2,342 | 127.0.0.1 | 127.0.0.2 | TCP | 40 | 50000 | 56835 | 50000 > 56835 [RST, ACK] Seq=1 Ack=1 |
| Client Request Connection | **B26** |  | 7 | 3,646 | 127.0.0.1 | 127.0.0.2 | TCP | 52 | 56835 | 50000 | 56837 > 50000 [SYN] Seq=0 |
| Server Accept Connection | **A19** | **B26** | 8 | 3,646 | 127.0.0.2 | 127.0.0.1 | TCP | 52 | 50000 | 56835 | 50000 > 56837 [SYN, ACK] Seq=0 Ack=1 |
| Client ACK Connection | **B26** | **A19** | 9 | 3,646 | 127.0.0.1 | 127.0.0.2 | TCP | 40 | 56835 | 50000 | 56837 > 50000 [ACK] Seq=1 Ack=1 |
| Send “Thx for accepting” | **B10** |  | 10 | 3,646 | 127.0.0.1 | 127.0.0.2 | TCP | 60 | 56835 | 50000 | 56837 > 50000 [PSH, ACK] Seq=1 Ack=1 Len=20 |
| *ACK from Server to Client* |  |  | 11 | 3,646 | 127.0.0.2 | 127.0.0.1 | TCP | 40 | 50000 | 56835 | 50000 > 56837 [ACK] Seq=1 Ack=21 Len=0 |
| Send “Thx for Connecting” | **A10** |  | 12 | 3,646 | 127.0.0.2 | 127.0.0.1 | TCP | 61 | 50000 | 56835 | 50000 > 56837 [PSH, ACK] Seq=1 Ack=21 Len=21 |
| *ACK from Client to Server* |  |  | 13 | 3,646 | 127.0.0.1 | 127.0.0.2 | TCP | 40 | 56835 | 50000 | 56837 > 50000 [ACK] Seq=21 Ack=22 Len=0 |
| Close Connection from C to S | **B12** |  | 14 | 3,646 | 127.0.0.1 | 127.0.0.2 | TCP | 40 | 56835 | 50000 | 56837 > 50000 [FIN, ACK] Seq=21 Ack=22 Len=0 |
| ACK Connection Close to C | **A12???** |  | 15 | 3,646 | 127.0.0.2 | 127.0.0.1 | TCP | 40 | 50000 | 56835 | 50000 > 56837 [FIN, ACK] Seq=22 Ack=22 Len=0 |
| Ack Connection Close to S | **???** |  | 16 | 3,646 | 127.0.0.1 | 127.0.0.2 | TCP | 40 | 56835 | 50000 | 56837 > 50000 [ACK] Seq=22 Ack=23 Len=0 |

Warum Ack in 11 & 13 ein Byte mehr als Len?