Rechnernetze, Übungsblatt 3, Sommer 2024

Aufgabe 1

Um Nachrichten mit Hilfe des Programms *nc_udp* bzw. *nc_tcp* zu schicken und zu empfangen geht man folgendermaßen vor:

- Zum Empfangen von Nachrichten ruft man *nc -l <port>* auf, wobei *port* die Portnummer angibt, unter der die Instanz erreichbar ist.
- Zum Versenden von Nachrichten ruft man *nc <IP-Adresse> <Port>* auf, wobei die beiden Argumente angeben, wie der Empfänger erreichbar ist.

UDP

Zunächst habe ich obiges mit dem UDP-Programm ausprobiert, d.h. die Instanz zum Empfangen habe ich mittels *nc_udp -l 44* und die Instanz zum Senden mittels *nc_udp 127.0.0.1 44* gestartet. Daraufhin habe ich die beiden Nachrichten *„Hello Listener“* und *„Have a nice day!“* versendet. Der aufgezeichnete Traffic befindet sich in der Datei *ncUDP.pcapng*. Mit dem Display-Filter *„udp.port == 44“* sind nur noch die für uns relevanten Pakete sichtbar. Hierbei handelt es sich lediglich um die beiden Pakte mit den Nummern 68 und 69, welche von der sendenden Instanz (Port 49582) an die empfangende Instanz (Port 44) gesendet wurden. Tatsächlich enthält ein Paket jeweils eine der beiden gesendeten Nachrichten in den Nutzdaten.

TCP

Anschließend habe ich selbiges mit dem TCP-Programm ausgeführt: Hierbei habe ich die Instanz zum Empfangen mittels *nc_tcp -l 43* und die Instanz zum Senden mittels *nc_tcp 127.0.0.1 43* gestartet. Daraufhin habe ich wieder die beiden Nachrichten *„Hello Listener“* und *„Have a nice day!“* versendet. Der aufgezeichnete Traffic befindet sich in der Datei *ncTCP.pcapng*, welcher mit dem Display-Filter *„tcp.port == 43“* auf die für uns relevanten Pakete begrenzt wird.

Im Gegensatz zu UDP ist TCP ein verbindungsorientiertes Protokoll. Daher werden bereits vor dem Schicken der beiden Nachrichten Paketen zwischen den beiden Instanzen zum Verbindungsaufbau versendet, wobei es sich um einen Three-Way-Handshake handelt. Zunächst sendet der Client ein SYN-Paket (SYN wie synchronize) an den Server (Paketnummer 30), der wiederum ein SYN-ACK-Paket (ACK wie acknowlegement) an den Client schickt (Paketnummer 31). Schließlich bestätigt der Client mit einem ACK-Paket (Paketnummer 32).

Doch auch nach dem Verbindungsaufbau lassen sich Unterschiede im Vergleich zum Verhalten der UDP-Instanzen beobachten: Die Nachricht *„Hello Listener“* befindet sich im Payload des Pakets 38 vom Client (Port 52081) an den Server (Port 43). Während UDP einen Best-Effort-Ansatz verfolgt, wird durch TCP sichergestellt, dass die Nachricht tatsächlich beim Empfänger angekommen ist. Daher sendet der Server ein ACK-Paket zur Bestätigung an den Client zurück (Paketnummer 39). Selbiges Verhalten lässt sich für die zweite Nachricht anhand der Pakete 40 und 41 nachvollziehen.

Aufgabe 2

Um das gegebene Programm zu einem Chatprogramm zu erweitern, muss eine Instanz gleichzeitig Nachrichten senden und empfangen können, wobei für jede der beiden Aufgaben ein Thread gestartet wird. Mein Chatprogramm wird folgendermaßen aufgerufen: *chat_udp <Port> <Name>*, wobei *Port* die gewünschte Portnummer der eigenen Instanz und *Name* den eigenen Namen angibt. Um sich bei einer anderen Instanz zu registrieren, gibt man *register <IP-Adresse> <Port>* ein, wobei sich die beiden Parameter auf die Instanz beziehen, die man kontaktieren möchte. Daraufhin gibt

diese den Namen der neu registrierten Instanz aus und sendet unter anderem ihren eigenen Namen zurück, welcher nun bei der anfragenden Instanz ausgegeben wird. Daraufhin lassen sich mit *send <Name> <Nachricht>* Nachrichten versenden.

Anhand der Traffic-Aufzeichnung *chatUDP.pcapng* lässt sich (mit dem Display-Filter *„udp“*) folgender Vorgang nachvollziehen:

- Paket 33: Die Instanz mit dem Namen Marvin registriert sich bei der Instanz mit dem Namen Otto
- Paket 34: Die Instanz mit dem Namen Otto antwortet, indem sie sich ihrerseits bei Marvin registriert
- Paket 35: Marvin schreibt an Otto *„How are you?”*
- Paket 36: Otto schreibt an Marvin *„I am fine.”*

Aufgabe 3

Um das gegebene Programm zu einem Chatprogramm zu erweitern, muss eine Client-Instanz gleichzeitig Nachrichten senden und empfangen können, wobei für jede der beiden Aufgaben ein Thread gestartet wird. Eine Server-Instanz wird durch *chat_tcp -l <Port>* aufgerufen, wobei *Port* die Portnummer angibt, unter welcher der Server erreichbar ist. Eine Client-Instanz wird mit *chat_tcp <IP-Adresse> <Port> <Name>* aufgerufen, wobei sich *IP-Adresse* und *Port* auf den Server beziehen und *Name* den eigenen Namen angibt. Direkt nach Programmstart registriert sich eine Client-Instanz beim Server und ist nun für andere Client-Instanzen unter ihrem Namen erreichbar. Nachrichten werden mit *send <Name> <Nachricht>* versendet.

Anhand der Traffic-Aufzeichnung *chatTCP.pcapng* lässt sich (mit dem Display-Filter *„tcp.port==43“*) folgender Vorgang nachvollziehen:

- Pakete 27 bis 29: Verbindungsaufbau zwischen Otto und dem Server
- Pakete 33 und 34: Registrierung von Otto beim Server (und Bestätigung)
- Pakete 51 bis 53: Verbindungsaufbau zwischen Marvin und Server
- Pakete 54 und 55: Registrierung von Marvin beim Server (und Bestätigung)
- Pakete 68 und 69: Nachricht von Marvin an Otto wird an den Server gesendet (und Bestätigung)
- Pakete 70 und 71: Marvins Nachricht wird vom Server an Otto gesendet (und Bestätigung)
- Pakete 72 und 73: Nachricht von Otto an Marvin wird an den Server gesendet (und Bestätigung)
- Pakete 74 und 75: Ottos Nachricht wird vom Server an Marvin gesendet (und Bestätigung)

Aufgabe 4

Da die zu codierende Bitfolge oft während einer Periode des Clocksignals wechselt, stellt sich die Frage, welcher Zustand der Bitfolge berücksichtigt für die Manchester-Codierung werden soll. Ich habe mich jeweils für den Zustand in der ersten Hälfte der Periode des Clocksignals entschieden, woraus 1000111 als zu codierende Bitfolge resultiert. Die Manchester-Codierung erfolgt nach dem Standard IEEE 802.3.