Testfälle

- **Was testen wir?**
- Wie testen wir?
- Wo testen wir?
- Erwartungen – nur ggf. bei Performance-Tests
- Was ist das Ergebnis?

Aufgabe 1

Geschwindigkeit der Paketübertragung

- Die Menge eines Produktes wird auf x festgelegt und bei jedem Schleifendurchgang, bei dem jedes Mal eine Nachricht an den Client gesendet wird, dekrementiert.
- Main.main()

Anzahl Pakete	Benötigte Zeit
1	43 ms
10	47 ms
1.000	448 ms
1.000.000	93.623 ms

Verlustquote der übertragenen Daten

- Der Client sendet x Pakete an den Server, dieser antwortet sofort mit einem String, welcher die Anzahl der bis dato empfangen Pakete enthält. Sobald der Client die Anzahl der festgelegten Pakete gesendet hat, gibt dieser eine Meldung über die Anzahl der gesendeten und empfangenen Pakete aus.
- Main.main()
- Wir erwarten keine Verluste!
- Es gab bei der Durchführung des Tests keine Verluste!

Verbotene Zeichen in Produktnamen

- Der Name des Produktes wird mithilfe eines regulären Ausdrucks geprüft. Wir schließen nicht explizit Zeichen aus, sondern schreiben erlaubte Zeichen vor.
- Main.main()
- Es wird erwartet, dass die Anwendung nicht abstürzt und eine Rückmeldung über die Fehleingabe liefert.
- Eingaben wie „Bana(ne“ oder „Schweinshaxe€“ werden abgelehnt. Eingaben wie „Käse“ oder „Würstcheneintopf mit Zwiebeln und Lauch“ werden akzeptiert.

Erreichbarkeit des Servers

- Client sendet ein Paket an Server und erwartet eine Rückmeldung innerhalb von 2 Sekunden. Antwortet der Server innerhalb des Timeouts, ist der Server erreichbar.
- UDPSocketClient.sendMsg()
- Wenn der Server erreichbar ist, wird er antworten. Wenn der Server nicht erreichbar ist, wird er nicht antworten.
- Das erwartete Ergebnis wird bestätigt.

Aufgabe 2

Request und Response (Anfrage und Antwort)

- Die angefragte URL wird vom Server analysiert, woraufhin der Server entsprechend reagiert. Wenn der angefragte URL-Pfad explizit festgelegten Mustern entspricht, reagiert der Server mit dem Status 200 - alle anderen Fälle (bad requests) werden durch den Server abgefangen und mit dem Status 400 beantwortet.
- UDPSocketServer.showWeb()
- Der Test zeigt, dass der Server korrekt auf korrekte und inkorrekte Anfragen reagiert.

Belastbarkeit des Servers

- Es werden 5, 50 und 100 Clients mithilfe eines Bash-Scripts gestartet.
- Bash-Script (start-udp-socket-client-sensor.sh)
- Wir erwarten keine marktreife Belastbarkeit.
- Der Server konnte alle 100 Clients bedienen, wobei wir den Test nach einigen Minuten vorzeitig beendeten.

Farbliche Darstellungsinformationen

- Es ist vorgesehen, dass vorrätige Produkte blau und ausgegangene Produkte rot dargestellt werden. Dies testen wir, indem wir die vom Server generierte Antwort im Server abfangen und den in der Antwort enthaltenen Quelltext aufschlüsseln und analysieren.
- UDPSocketServer.showWeb() und UDPSocketServer.testColor(String, String)
- Der Test zeigt, dass die Farbinformationen korrekt übertragen werden.

Vergleich von Sensordaten und Response

- Die Antwort sollte ein Abbild dessen sein, das die Sensordaten liefern. Dies testen wir, indem wir die vom Server generierte Antwort im Server abfangen und den in der Antwort enthaltenen Quelltext aufschlüsseln und analysieren.
- UDPSocketServer.showWeb() und UDPSocketServer.testShowList(String)
- Der Test zeigt, dass die Sensordaten korrekt in der Antwort übertragen werden.

Aufgabe 3

Verarbeitungsgeschwindigkeit der Bestellungen

- Anzahl der Bestellungen wird mit der Konstanten NUMBER_OF_ORDERS festgelegt und die Methode orderTest() wird ausgeführt. Eine Schleife generiert die festgelegte Anzahl an Bestellungen. Die Dauer der Ausführung der Methode orderTest() wird über einen Timer ermittelt.
- UDPSocketServer.orderTest()

Anzahl Bestellungen	Benötigte Zeit
1	3001 ms
10	30.024 ms
100	300.353 ms

Vergleich zwischen Bestell- und Liefermenge

- Anzahl der Bestellungen wird mit der Konstanten NUMBER_OF_ORDERS festgelegt und die Methode orderTest() wird ausgeführt. Eine Schleife generiert die festgelegte Anzahl an Bestellungen. In jedem Durchlauf der Schleife wird die bestellte Menge mit der gelieferten Menge verglichen und das Ergebnis des Vergleichs ausgegeben.
- UDPSocketServer.orderTest() → testNameAndValue()
- Während unserer Tests wurden keine Abweichungen der Bestell- und Liefermenge festgestellt.

Vergleich zwischen bestellten und gelieferten Produkt

- Anzahl der Bestellungen wird mit der Konstanten NUMBER_OF_ORDERS festgelegt und die Methode orderTest() wird ausgeführt. Eine Schleife generiert die festgelegte Anzahl an Bestellungen. In jedem Durchlauf der Schleife wird das bestellte Produkt mit dem gelieferten Produkt verglichen und das Ergebnis des Vergleichs ausgegeben.
- UDPSocketServer.orderTest() → testNameAndValue()
- Während unserer Tests wurden keine Abweichungen zwischen bestellten und gelieferten Produkten festgestellt.

Vergleich zwischen Anzahl Bestellungen und Anzahl Rechnungen

- Anzahl der Bestellungen wird mit der Konstanten NUMBER_OF_ORDERS festgelegt und die Methode orderTest() wird ausgeführt. Eine Schleife generiert die festgelegte Anzahl an Bestellungen. In jedem Durchlauf der Schleife wird die Anzahl der Bestellungen mit der Anzahl der Rechnungen verglichen und das Ergebnis des Vergleichs ausgegeben.
- UDPSocketServer.orderTest()
- Während unserer Tests stimmte die Anzahl der Bestellungen mit der Anzahl der Rechnungen überein.

Aufgabe 4

Geschwindigkeit der Nachrichtensendungen

- Die Konstante DO_TIME_TEST wird mit true initialisiert. Die Anzahl der zu sendenden Nachrichten wird mit der Konstanten NUMBER_OF_MESSAGES festgelegt und die Methode run() wird ausgeführt. Eine Schleife generiert die festgelegte Anzahl an Nachrichten. Die Dauer der Ausführung der Schleife wird über einen Timer ermittelt.
- mqttPublisher/src/main/java/de.hda.fbi.ds.ks.mqtt/Publisher.java → run()
  Zeilen 94 bis 107

Anzahl Nachrichten	Benötigte Zeit
10	4.163 ms
100	43.168 ms
1000	451.786 ms

Vergleich zwischen gesendeter und empfangener Nachricht

- Die Konstante DO_TIME_TEST wird mit false initialisiert und die Methode run() wird ausgeführt. Es wird die gesendete mit der empfangenen Nachricht, welche sich in einer Textdatei befindet, verglichen und das Ergebnis des Vergleichs ausgegeben.
- mqttPublisher/src/main/java/de.hda.fbi.ds.ks.mqtt/Publisher.java → run()
  Zeilen 153 bis 179
- Während unserer Tests wurden keine Abweichungen der gesendeten und empfangenen Nachrichten festgestellt.

Verlustrate gesendeter Nachrichten

- Die Konstante DO_TIME_TEST wird mit true initialisiert. Die Anzahl der zu sendenden Nachrichten wird mit der Konstanten NUMBER_OF_MESSAGES festgelegt und die Methode run() wird ausgeführt. In einer Schleife werden die zu sendenden Nachrichten generiert und gesendet – diese sollten im Anschluss entsprechend empfangen werden. Anschließend wird die Anzahl der gesendeten mit jener der empfangenen Nachrichten verglichen und das Ergebnis des Vergleichs ausgegeben.
- mqttPublisher/src/main/java/de.hda.fbi.ds.ks.mqtt/Publisher.java → run()
  Zeilen 109 bis 118
- Während unserer Tests wurden keine Verluste der gesendeten Nachrichten festgestellt.

Verbotene Zeichen in zu sendenden Nachrichten

- Die Konstante DO_TIME_TEST wird mit false initialisiert und die Methode run() wird ausgeführt. Eine Nachricht wird mithilfe eines regulären Ausdrucks auf verbotene Zeichen geprüft. Wir schließen nicht explizit Zeichen aus, sondern schreiben erlaubte Zeichen vor.
- mqttPublisher/src/main/java/de.hda.fbi.ds.ks.mqtt/Publisher.java → run()
  Zeilen 134 bis 141
- Die Nutzung regulärer Ausdrücke funktioniert wie erwartet.