Verteilte Systeme Praktikum WS18/19

Protokoll Muhammed Selim Sinan und Rayhana Akbari

Projektplan (Termin 1)

Anforderungsanalyse Gesamtsystem:

Aufgabe 1	Funktionale Anforderungen	Nicht-Funktionale Anforderungen
Gesamtsystem	- Das Gesamtsystem soll tatsächtlich verteilt sein, sodass einzelne Komponenten von verschiedenen Systemen aus gestartet werden können. - Es sollen mehrere Smart-Homes (min. 3) parallel samt ihrer Sensoren laufen.	- Die Bedienungsoberflächen sollten so gestaltet sein, dass der Nutzer versteht was passiert. - Das Starten des Gesamtsystems soll mit nicht viel Aufwand verbunden sein.
Aufgabe 2	- Sensoren sollen Informationen, die sich ständig ändern erfassen/simulieren. - Die Information soll an eine Zentrale in einem geeigneten Nachrichtenformat über UDP übertragen werden. - Die Zentrale soll die empfangenen Informationen an eine Standartausgabe unter Angabe von IP, Port und Sensortyp ausgeben.	- Die Übertragung sollte keine hohe Latenz haben. - Die Zentrale soll in der Lage sein bis zu 20 Sensoren parallel zu verarbeiten.
Aufgabe 3	- Die Zentrale soll einen HTTP-Server besitzen, der den HTTP GET Befehl mindestens beherrscht. - Die HTTP-Schnittstelle der Zentrale soll über REST-API den Zugriff auf die einzelnen Sensordaten und deren Historie ermöglichen.	- Die HTTP Seiten sollten benutzerfreundlich sein. - Die einzelnen Sensordaten sollten einen Zeitstempel haben und sortiert sein
Aufgabe 4	- Die Smart-Home-Zentralen sollen ihren Status über MQTT an den Hersteller übermitteln. - Der Hersteller soll die Daten persistent speichern.	- Die Server des Herstellers sollen aufgrund der Ausfallsicherheit redundant ausgelegt werden. - Die Übertragung an den Hersteller soll den Betrieb der Smart-Home-Zentrale nicht beeinträchtigen.
Aufgabe 5	- Die Herstellerserver sollen über eine RPC Schnittstelle die Daten durch Wetterdaten anreichern und an die Zentrale zurückschicken. - Die Wetterdaten sollen im HTTP Interface der einzelnen Smart-Home Zentralen abgerufen werden können.	- Die Software benutzt die reale REST-API von OpenWeatherMap - Die Server des Herstellers kann mit bis zu 10 Zentralen effizient kommunizieren.

Grobes Systemdesign (Aufgabe 1)

Im Groben gibt es ein Smart-Home-System bestehend aus verschiedenen Sensoren und einer Zentrale. Daneben unterhält der Hersteller Server, welche mit Daten vom Wetterdienst und Daten von den Smart-Home-Zentralen versorgt werden.

Eigenständige Programme/Projekte sind:
- Sensoren
- Smart-Home-Zentrale
- Herstellerserver

Die Sensoren sind getrennt von der Smart-Home Zentrale und kommunizieren mit dieser ausschließlich per UDP. Ein Sensor läuft in einem eigenen Thread und läuft somit parallel mit anderen Sensoren verschiedener Typen.

Sensoren Features:
- UDP Socket, zum verschicken von Messdaten
- Multithreading, zum darstellen verteilter Peripherie
- Zufällige Messdaten

Die Smart-Home Zentrale empfängt die anströmenden Sensordaten und startet pro Nachricht einen neuen Thread, um diese zu speichern. Das sorgt dafür, dass Nachrichten auch bei einem Fluten des Servers nicht verloren gehen. Die Speicherung der Daten erfolgt im Producer-Consumer-Pattern, sodass sie an anderer Stelle effizient abgefragt werden bei gleichzeitiger Speicherung. Die Zentrale präsentiert die Sensordatenhistorie auf einem Webserver, welcher parallel läuft und HTTP und REST implementiert.

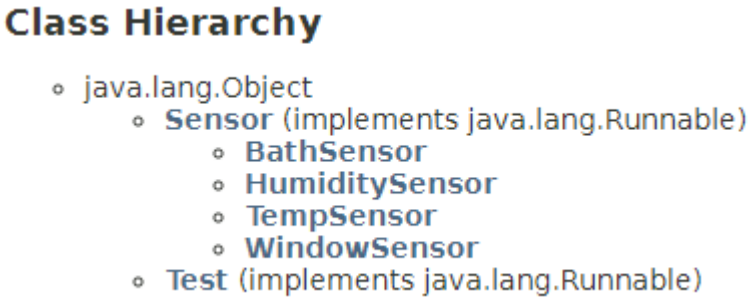
Smart-Home-Zentrale Features:
- UDP Socket, zum empfangen von Messdaten
- Producer-Consumer Pattern, zum effizienten speichern der Daten
- HTTP Requests, um die Daten abzufragen
- REST API, um die Daten abzufragen
- MQTT, zum Datenaustausch mit dem Hersteller

Der Hersteller unterhält Server welche redundant ausgelegt sind aufgrund von Ausfallsicherheit und kommunizieren per MQTT mit dem Webserver der Smart-Home-Zentralen. Es wird auf die Sensordaten der Smart-Home-Zentralen zugegriffen und mit Wetterdaten aus einem externen weiteren Server angereichert.

Hersteller Features:
- MQTT, zum Datenaustausch mit der Smart-Home-Zentrale
- HTML & REST, um Wetterdaten anzureichern
- MySQL, um die Sensordaten persistent zu speichern
- Redundanz, um Ausfallsicherheit zu gewährleisten.

Weiteres zum Projektplan (z.B. Tasks) befindet sich auf Gitlab.

Sensoren und UDP Sockets(Aufgabe 2)



Die Sensoren werden in einem eigenständigen Projekt im IDE entwickelt.

Wie man aus der Klassenhierarchie entnehmen kann besteht dieses Projekt aus einer Oberklasse **Sensor**, welche die gemeinsamen Features, wie UDP Sockets oder Multithreading implementiert. Die Unterklassen { **BathSensor, HumiditySensor, TempSensor, WindowSensor** } erben von ihrer Oberklasse und überschreiben die Methode measure(), um je nach Sensortyp die Messwerte anzugeben.

Alle Sensoren benutzen den gleichen DatagramSocket, um Nachrichten zu senden, da sonst viele verschiedene Ports belegt werden müssen.

Jeder Sensor läuft in einem eigenen Thread und hat eine run() funktion.

HTTP und TCP-Sockets(Aufgabe 3)

Die Smart Home Zentrale ist ebenfalls ein eigenes Projekt in der IDE und empfängt die Daten über Datagram Sockets. Sobald eine neue Message kommt wird ein neuer Thread erstellt, der diesen performant abarbeitet. Die Daten werden in einer Klasse **SensorData** gespeichert, welcher in einem ProducerConsumer Pattern die JSON Objekte in einen Vector speichert. Dieser Vector kann sortiert werden nach der MessageNR. Zeitgleich wird beim Hinzufügen eines neuen Eintrags direkt ein Thread erstellt, welcher über Mqtt diese Zeile an den Hersteller publisht. Gleichzeitig ist die Zentrale auch MqttSubscriber für die Wetterdaten vom Hersteller.

MQTT(Aufgabe 4)

Die Herstellerserver sind so ausgelegt, dass mehrere Instanzen daraus gestartet werden können. Dadurch wird Ausfallsicherheit gewährleistet. Die Herstellerserver Subscriben per Mqtt auf die Sensordaten und empfangen sie. Gleichzeitig starten Sie bei empfangener Nachricht einen neuen Thread, der eine Verbindung zur Mysql Datenbank aufbaut, um die Daten persistent zu speichern.

RPC Rest(Aufgabe 5)

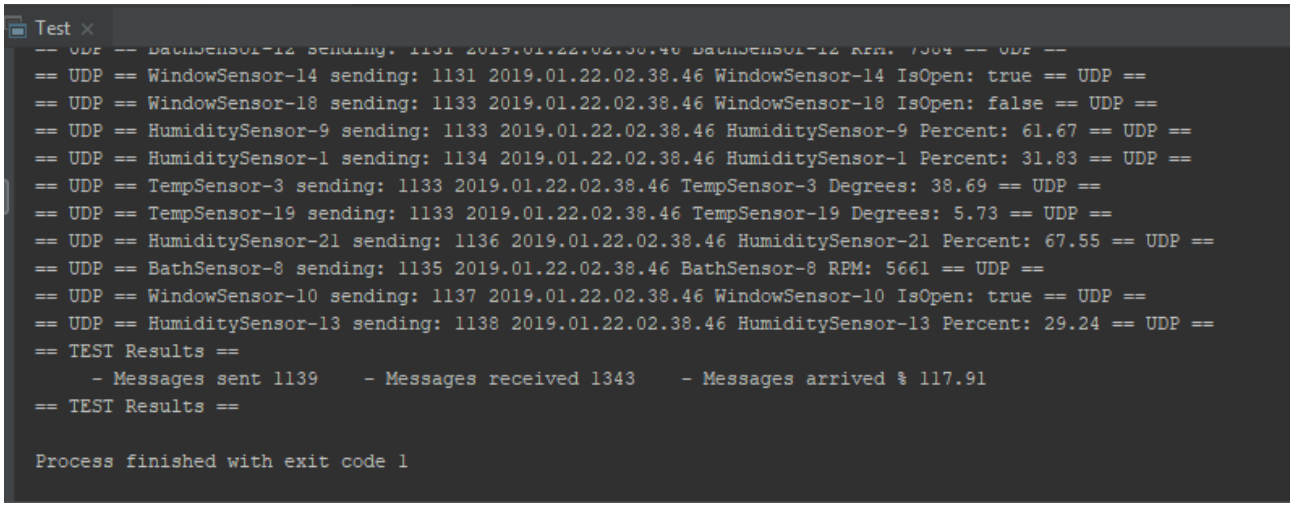
Die Herstellerserver bauen eine HTML Verbindung zum Wetterdienst auf und empfangen per REST Api die Wetterdaten. Diese Werden über Mqtt an die Smart-Home-Zentralen zugeschickt, welche sie auf der Website darstellen.

Performance Tests

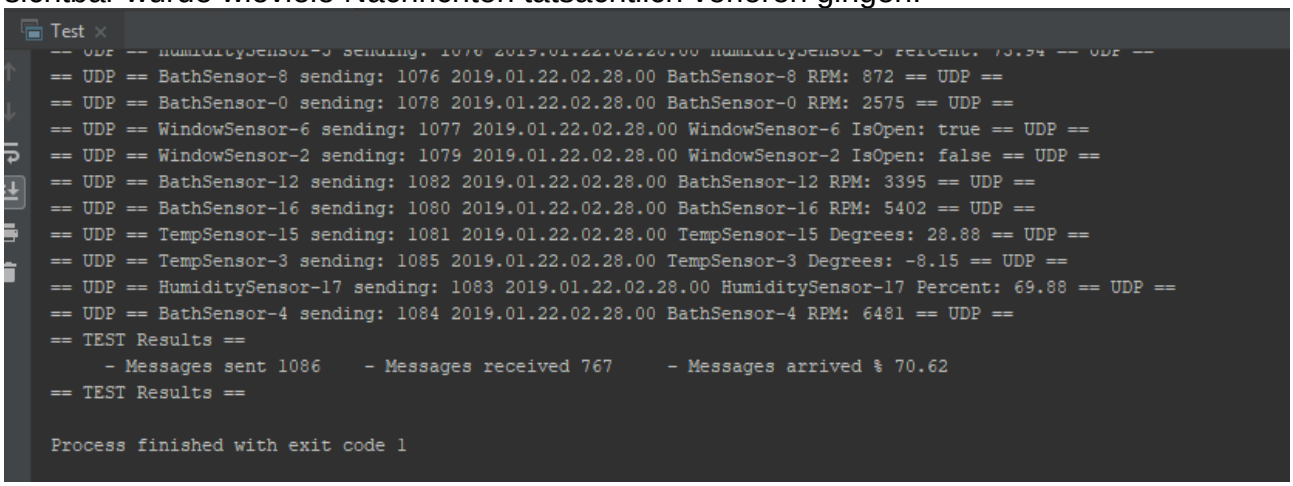
Der Test kann in zwei Modusen ausgeführt werden. Der Normale Modus, welcher testet, wieviele Nachrichten angekommen sind im Normalbetrieb und der Performance Modus, welcher 24 Sensoren parallel im Burst Modus mit geringen Intervallen betreibt und UDP Pakete flutet.

Die Testklasse läuft aktiv parallel mit und vergleicht die Anzahl verschickten Messages mit der Anzahl tatsächtlich angekommener Messages. Diesen bekommt Sie exklusiv von der Smart-Home-Zentrale über Mqtt zugeschickt.

Kurioserweise kamen bei einem Performance-Test mehr Pakete an als verschickt wurden.

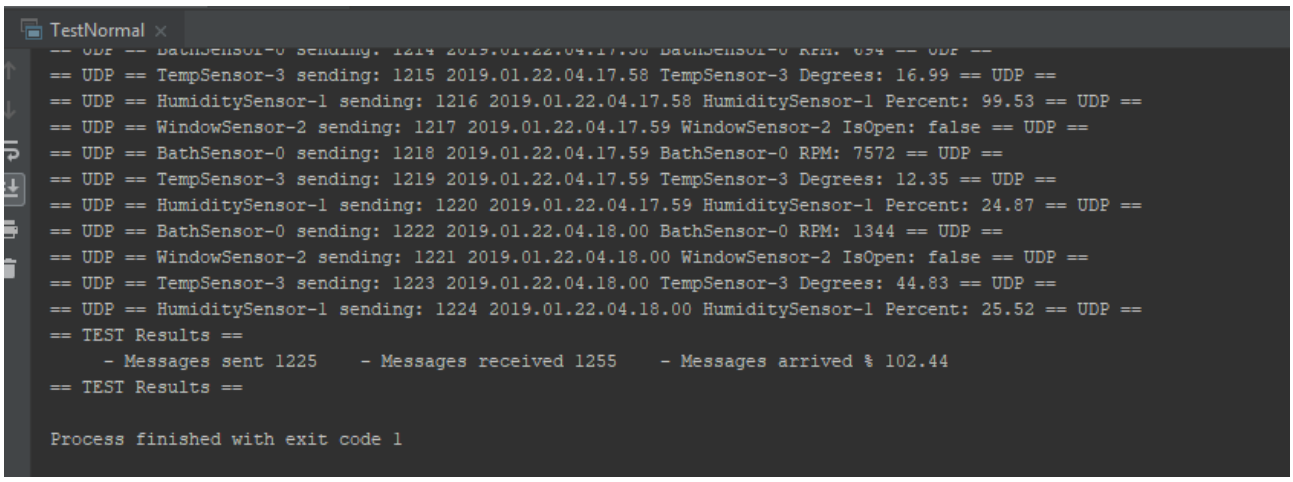


Manche Pakete wurden mehrfach von der Zentrale verarbeitet und somit doppelt gespeichert. Dazu wurde eine Methode innerhalb der Smart-Home-Zentrale entwickelt, welcher die Nachrichtenliste sortiert (Insertion-Sort, da schon teilweise sortiert) und gleichzeitig doppelte Nachrichten löscht. Dies führte im performance Test dazu, dass sichtbar wurde wieviele Nachrichten tatsächtlich verloren gingen.



Zwar sind ca. 30% verloren gegangen, aber in diesem Performance-Test wurden auch über 1000 UDP Pakete innerhalb von 5 Sekunden verschickt.

Wenn man diesen Test im „Normalbetrieb“ wiederholt stellt man fest, dass alle Nachrichten ankommen und fast alle doppelten Nachrichten eliminiert wurden:


TestNormal
== UDP == TempSensor-3 sending: 1215 2019.01.22.04.17.58 TempSensor-3 Degrees: 16.99 == UDP ==
== UDP == HumiditySensor-1 sending: 1216 2019.01.22.04.17.58 HumiditySensor-1 Percent: 99.53 == UDP ==
== UDP == WindowSensor-2 sending: 1217 2019.01.22.04.17.59 WindowSensor-2 IsOpen: false == UDP ==
== UDP == BathSensor-0 sending: 1218 2019.01.22.04.17.59 BathSensor-0 RPM: 7572 == UDP ==
== UDP == TempSensor-3 sending: 1219 2019.01.22.04.17.59 TempSensor-3 Degrees: 12.35 == UDP ==
== UDP == HumiditySensor-1 sending: 1220 2019.01.22.04.17.59 HumiditySensor-1 Percent: 24.87 == UDP ==
== UDP == BathSensor-0 sending: 1222 2019.01.22.04.18.00 BathSensor-0 RPM: 1344 == UDP ==
== UDP == WindowSensor-2 sending: 1221 2019.01.22.04.18.00 WindowSensor-2 IsOpen: false == UDP ==
== UDP == TempSensor-3 sending: 1223 2019.01.22.04.18.00 TempSensor-3 Degrees: 44.83 == UDP ==
== UDP == HumiditySensor-1 sending: 1224 2019.01.22.04.18.00 HumiditySensor-1 Percent: 25.52 == UDP ==
== TEST Results ==
- Messages sent 1225 - Messages received 1255 - Messages arrived % 102.44
== TEST Results ==
Process finished with exit code 1