**Client/Server-Anwendung „Verteilte Nachrichten-Queue“ – Protokoll**

**Aufgabenstellung:**

In dieser Aufgabe wird eine Java RMI-Schnittstelle genutzt, um Nachrichten zwischen mehreren Clients auszutauschen, welche mit einem Server verbunden sind. Ziel der Aufgabe ist es eine interoperable Anwendung zu erstellen, welche mit auf verschiedenen verteilten Systemen mit anderen Anwendungen kommunizieren kann. Dazu soll die Anwendung auch Fehler tolerieren und entsprechend auf diese reagieren.

**RMI:**

Die Remote Method Invocation (RMI) ist eine Art des Remote Procedure Call (RPC), welche in Java genutzt wird. RPC realisiert die Interprozesskommunikation, welches den Informationsaustausch zwischen Prozessen ermöglicht. Daher können mit RMIMethoden entfernter Objekte aufgerufen werden. Der Server meldet bei der RMI-Registry, unter einem Namen, das Skeleton an sodass der Client darüber einen Stub des entfernten Objekts erhalten kann und somit auf die entfernten Methoden des Objekts zugreifen kann1.

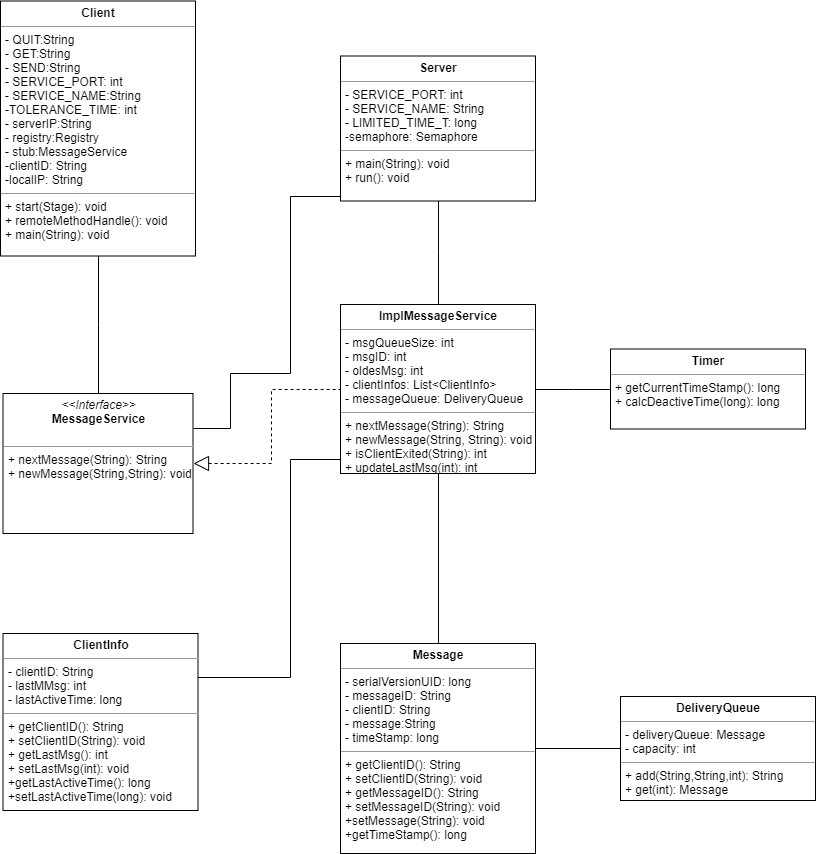
**Programmablauf und Implementierung:**

Der Server versucht anfangs bei der RMI-Registry ein entferntes Objekt, unter einem Namen, zu registrieren, weshalb diese als erstes gestartet werden muss über den Aufruf „rmiregistry“. Wenn der Server läuft kann der Client gestartet werden. Der Client versucht dann, mit der IP-Adresse des Servers und der Methode getRegistry(), einen Verweis auf RMI-Registry zu finden. Wenn dies erfolgt, wird die lookup()-Methode genutzt, welche einen Stub liefert, welcher die Server-Schnittstelle implementiert2. Anschließend ist der Client bereit, um Nachrichten zu verschicken und abzurufen. Die Client Anwendung wird von einer GUI begleitet, welche es dem Benutzer erlaubt mit einem Dropdown Menü zwischen den Funktionen des Sendens und Erhaltens von Nachrichten zu wechseln.

Bei dem Verschicken einer Nachricht wird die Nachricht mit der ID des Clients und einer eigenen ID in eine Message Queue hinzugefügt und beim Abrufen einer Nachricht wird geprüft ob der abrufende Client noch Verbunden ist und wenn dies der Fall ist wird die Nachricht aus der Message Queue genommen und and den Client zurückgegeben. Dabei wird die älteste Nachricht in der Queue verfolgt und zurückgegeben. Wenn die Queue voll ist wird die älteste Nachricht aktualisiert, wenn die Queue voll ist wird die nächstältere Nachricht gemerkt. Über eine MesageID wird die Position der Nachricht gespeichert, welche als nächstes aufgerufen werden soll.

Da mehrere Clients Nachrichten abrufen und verschicken können, werden Semaphoren benutzt, damit nur ein Client zurzeit eine Nachricht abrufen/verschicken kann.

Nebenbei läuft im Server ein Thread, welcher die letzte Aktivität prüft und Clients aus seiner Liste wirft, welche länger als eine Minute inaktiv waren.



Klassendiagramm

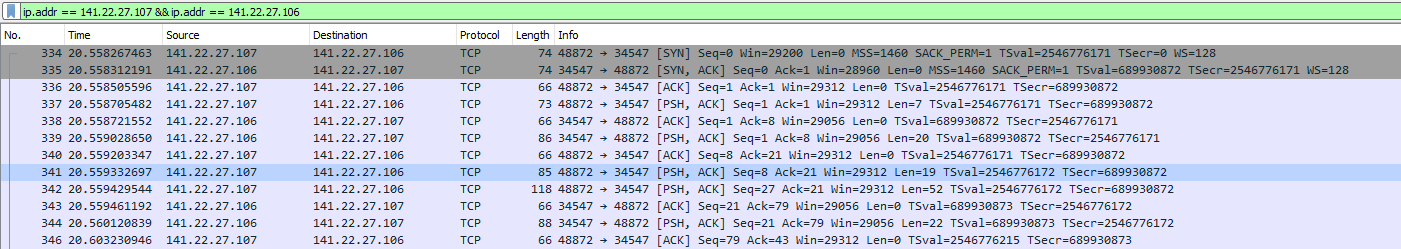
**Fehlersemantik:**

**Client** – Der Client implementiert bei seiner newMessage()-Methode die ‘At-least-once‘ Fehlersemantik. Bei dieser Fehlersemantik wird im Fehlerfall der Request erneut gesendet und Duplikate werden nicht gefiltert. Dies bedeutet in unserer Implementierung, dass der Client immer wieder versucht die Nachricht zu versenden. Dabei muss aber auch jedes Mal versucht werden die RMI-Schnittstelle zu erreichen, da bei einem Server Neustart eine neue Referenz auf das Objekt erhalten werden muss. Das wiederholte Versuchen wird jedoch nur innerhalb eines Toleranzintervalls durchgeführt.

Bei der nextMessage()-Methode wird die ‘Maybe‘ Fehlersemantik implementiert, bei der im Fehlerfall der Request nicht noch einmal verschickt. Daher beendet sich der Client, wenn bei nextMessage() der Server nicht erreichbar ist.

**Server** – Beim Server wird die ‘At-most-once‘ Fehlersemantik beim Ausliefern der Nachricht implementiert. Dabei wird im Fehlerfall der Request nochmal verschickt, es wird jedoch darauf geachtet, dass keine Duplikate vorkommen. Dies wird beim Server sichergestellt, da jede Nachricht eine einheitlich messageID hat. Die messageID wird bei jedem Hinzufügen einer Nachricht inkrementiert und ist für jede Nachricht somit, zur Laufzeit des Servers, einheitlich.

**Versuchsprotokoll:**

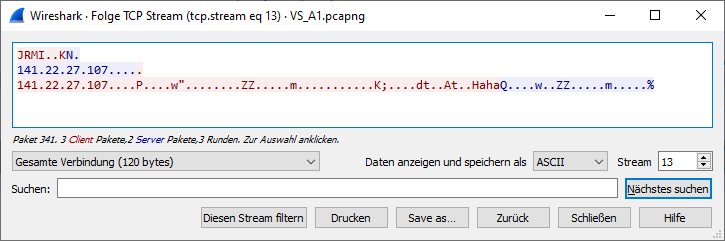
****

1 - Verbindungsaufbau mit Nachbarsystem

**Verbindungsaufbau:**

Die Anwendung wurde mit einem nebenliegenden Rechner mit der IP Adresse “141.22.27.106“ getestet, auf dem der Server lief. In der obigen Abbildung kann man den Verbindungsaufbau, in der Form des “three way handshakes“ erkennen. Der Client mit der IP Adresse “141.22.27.107“ sendet ein Packet, mit dem SYN-Flag gesetzt, an den Server und dieser Antwortet mit einem Packet, in dem sowohl das SYN – Flag, als auch das ACK – Flag gesetzt sind. Anschließend sendet der Client dann ein Packet, in dem das ACK – Flag gesetzt ist und die Sequence Number um eins erhöht wurde, um den Erhalt des Packet vom Server zu bestätigen.

**Datenübertragung:**

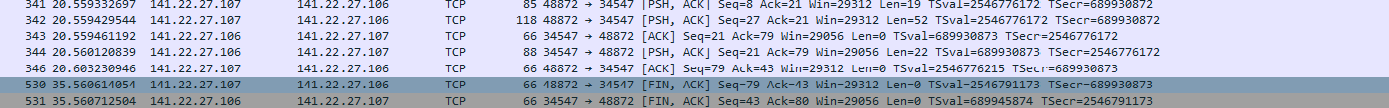


2 - TCP Stream

Aus der Abbildung 1 kann man die Datenübertragung an den Stellen erkennen, an denen ein Packet mit dem PSH – Flag verschickt wurde. Das PSH – Flag gibt an, dass der eingehende und auch ausgehende Puffer übergangen werden soll. Somit werden kleinere Übertragungen zu einer Größeren gebündelt und dann verscickt4.

In dem TCP – Strom aus Abbildung 2 kann man dann beispielsweise die Client IP sehen, welche als clientID an den Server gesendet wurde, sowie die Nachricht „Haha“.

**Verbindungsabbau:**

****

3 - Verbindungsabbau

Die Verbindung wurde vom Client beendet, daher wird diesmal ein Packet mit dem FIN – Flag und dem ACK-Flag, statt des SYN – Flags. Der Server antwortet dann ebenfalls mit der gleichen Art von Packet.

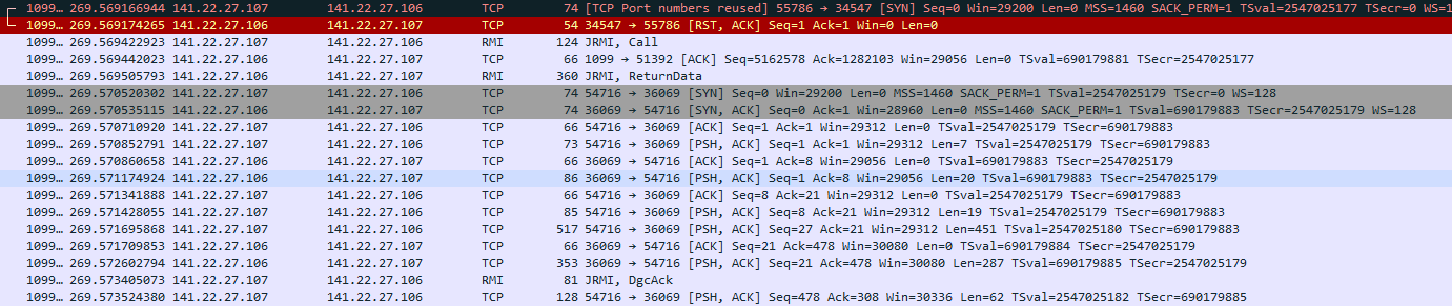
**Serverabsturz:**

In der folgenden Abbildung wurde der Server beendet und der Client hat versucht eine Nachricht zu verschicken. In der Abbildung kann man erkennen, dass immer wieder vom Client versucht wird den Server zu erreichen. Das Packet mit dem RST – Flag wird verwendet, wenn eine Verbindung abgebrochen werden soll. Dementsprechend wird dieses Packet vom Server verschickt. Mit dem RMI Protokoll wird versucht die RMI-Schnittstelle zu finden (Method Invocation durch JRMI Call). Da diese nicht beendet wurde reagiert der Rechner des Servers auch nach wie vor mit einem ACK und einem erfolgreichen “RMI-Call“ (JRMI ReturnData). Der Server antwortet mit DgcAck was eine Bestätigung ist, dass ein entferntes Objekt vom Server erhalten wurde5.

****

4 – Serverabsturz

**Server Neustart:**

****

5 - Server Neustart

Der Server wurde wieder neugestartet und da ein “rebind“ der Registry auf den gleichen Port vom Server gemacht wurde, wird dies bei wireshark angezeigt. Anschließend kann man erkennen das die Verbindung wieder aufgebaut wurde und es wieder Daten zwischen Client und Server ausgetauscht wurden.

**Quellen:**

1 - Einführung in RMI – 4.1 Stub- und Skeleton-Compiler rmic

<http://www.ti.uni-tuebingen.de/fileadmin/assets/csp_ws0809/aufgabe2/aufgabe2a.pdf>

2 – Programmieren mit Java – 19.1.3 Anwendungsentwicklung am Beispiel RMI

<https://www.dpunkt.de/java/Programmieren_mit_Java/Remote_Method_Invocation/5.html>

3 - Middleware in Java Kapitel 6 Seite 5: Struktur einer RMI-Anwendung

<https://www.informatik.uni-marburg.de/~mathes/download/k6.pdf>

4 – Das Transmission Control Protocol – Das Internet – Teil 7

<https://www.webschmoeker.de/grundlagen/tcp-transmission-control-protocol/>

5 – JavaDocs – 10.2 RMI Transport Protocol

<https://docs.oracle.com/javase/8/docs/platform/rmi/spec/rmi-protocol3.html>