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.

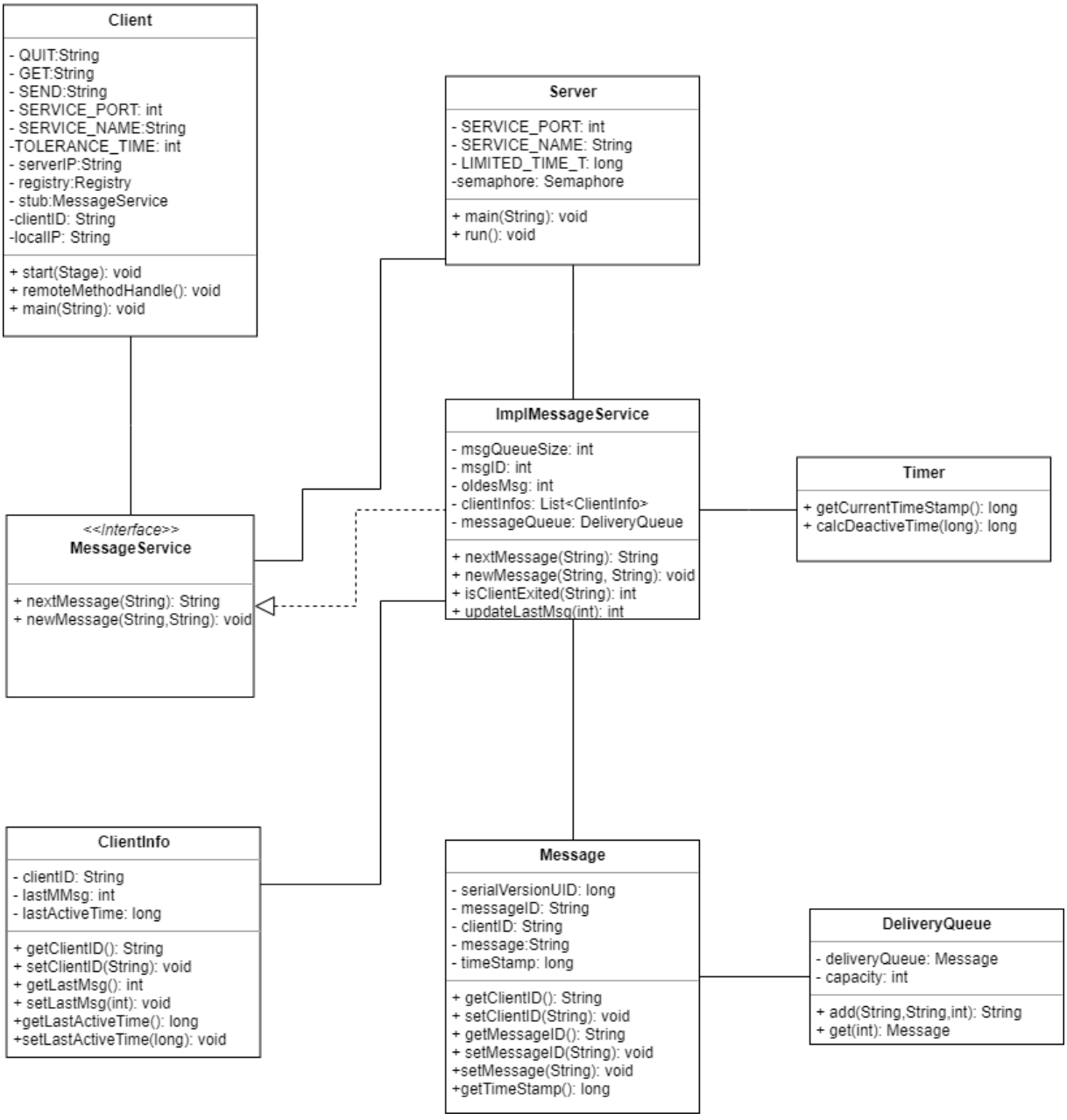
RMI:

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 RMI Methoden 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 kann[1].

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 implementiert[2]. 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

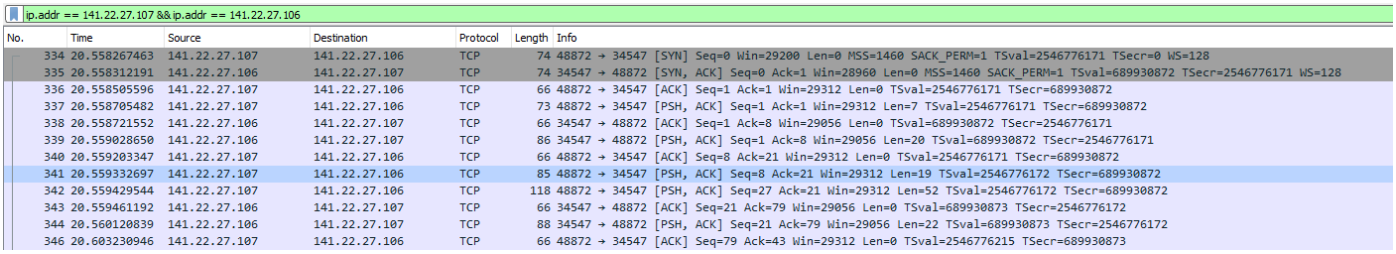
The Kiet Dang
Sofian Wüsthoff

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:


ip.addr == 141.22.27.107 && ip.addr == 141.22.27.106

No.	Time	Source	Destination	Protocol	Length	Info
334	20.558267463	141.22.27.107	141.22.27.106	TCP	74	48872 → 34547 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=2546776171 TSecr=0 WS=128
335	20.558312191	141.22.27.106	141.22.27.107	TCP	74	34547 → 48872 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=689930872 TSecr=2546776171 WS=128
336	20.558505596	141.22.27.107	141.22.27.106	TCP	66	48872 → 34547 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=2546776171 TSecr=689930872
337	20.558705482	141.22.27.107	141.22.27.106	TCP	73	48872 → 34547 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=7 TSval=2546776171 TSecr=689930872
338	20.558721552	141.22.27.106	141.22.27.107	TCP	66	34547 → 48872 [ACK] Seq=1 Ack=8 Win=29056 Len=0 TSval=689930872 TSecr=2546776171
339	20.559028650	141.22.27.106	141.22.27.107	TCP	86	34547 → 48872 [PSH, ACK] Seq=1 Ack=8 Win=29056 Len=20 TSval=689930872 TSecr=2546776171
340	20.559203347	141.22.27.107	141.22.27.106	TCP	66	48872 → 34547 [ACK] Seq=8 Ack=21 Win=29312 Len=0 TSval=2546776171 TSecr=689930872
341	20.559332697	141.22.27.107	141.22.27.106	TCP	85	48872 → 34547 [PSH, ACK] Seq=8 Ack=21 Win=29312 Len=19 TSval=2546776172 TSecr=689930872
342	20.559429544	141.22.27.107	141.22.27.106	TCP	118	48872 → 34547 [PSH, ACK] Seq=27 Ack=21 Win=29312 Len=52 TSval=2546776172 TSecr=689930872
343	20.559461192	141.22.27.106	141.22.27.107	TCP	66	34547 → 48872 [ACK] Seq=21 Ack=79 Win=29056 Len=0 TSval=689930873 TSecr=2546776172
344	20.560120839	141.22.27.106	141.22.27.107	TCP	88	34547 → 48872 [PSH, ACK] Seq=21 Ack=79 Win=29056 Len=22 TSval=689930873 TSecr=2546776172
346	20.603230946	141.22.27.107	141.22.27.106	TCP	66	48872 → 34547 [ACK] Seq=79 Ack=43 Win=29312 Len=0 TSval=2546776215 TSecr=689930873

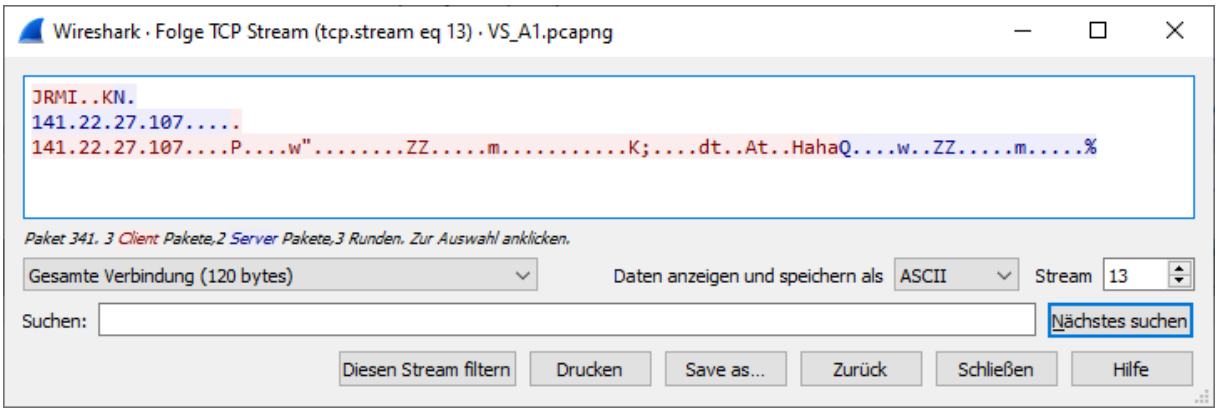
1 - Verbindungsaufbau mit Nachbarsystem

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.

The Kiet Dang
Sofian Wüsthoff

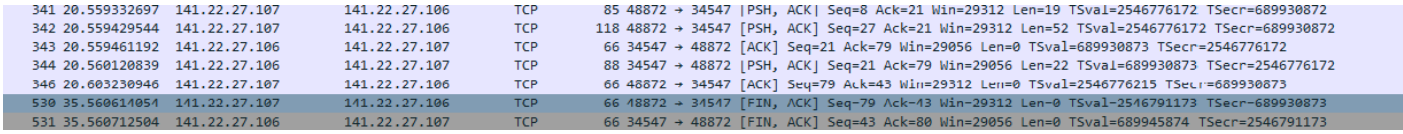
Datenübertragung:



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 verscickt[4]. 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.

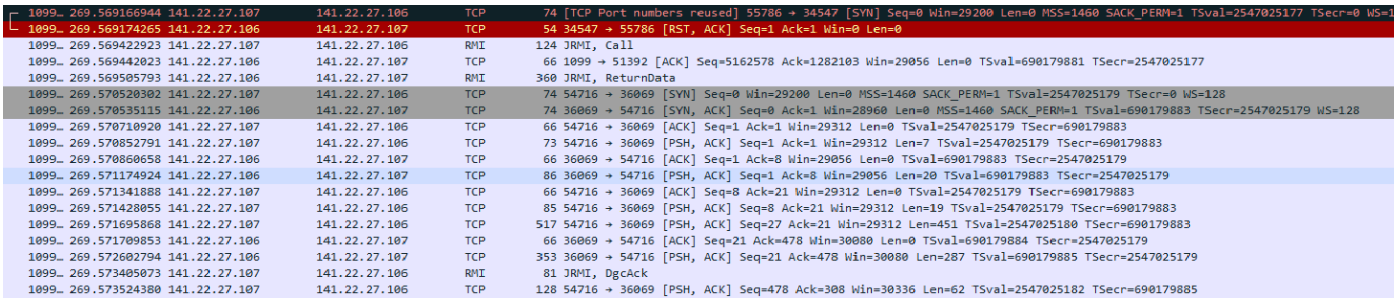
Serverabsturz:
In der folgenden Abbildung wurde der Server beendet und der Client hat versucht eine Nachricht zu verschicken. In der Abbildung kann man erkennen, dass immer wieder vom Client versucht wird den Server zu erreichen. Das Packet mit dem RST – Flag wird verwendet, wenn eine Verbindung abgebrochen 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 wurde[5].

The Kiet Dang
Sofian Wüsthoff

No.	Time	Source	Destination	Protocol	Length	Info
4046	257.133894941	141.22.27.106	141.22.27.107	TCP	54	34547 → 48902 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
4047	257.134314052	141.22.27.107	141.22.27.106	RMI	124	JRMI, Call
4048	257.134346354	141.22.27.106	141.22.27.107	TCP	66	1099 → 51392 [ACK] Seq=315 Ack=158 Win=29056 Len=0 TSval=690167446 TSecr=2547012742
4049	257.134801666	141.22.27.107	141.22.27.106	RMI	360	JRMI, ReturnData
4050	257.135407011	141.22.27.107	141.22.27.106	RMI	81	JRMI, DgcAck
4051	257.135584199	141.22.27.107	141.22.27.106	TCP	74	48904 → 34547 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=2547012744 TSecr=0 WS=128
4052	257.135605846	141.22.27.106	141.22.27.107	TCP	54	34547 → 48904 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
4053	257.136009436	141.22.27.107	141.22.27.106	RMI	124	JRMI, Call
4054	257.136040701	141.22.27.106	141.22.27.107	TCP	66	1099 → 51392 [ACK] Seq=609 Ack=231 Win=29056 Len=0 TSval=690167448 TSecr=2547012744
4055	257.136475691	141.22.27.107	141.22.27.106	RMI	360	JRMI, ReturnData
4056	257.137013707	141.22.27.107	141.22.27.106	RMI	81	JRMI, DgcAck
4057	257.137198254	141.22.27.107	141.22.27.106	TCP	74	48906 → 34547 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=2547012745 TSecr=0 WS=128
4058	257.137215221	141.22.27.106	141.22.27.107	TCP	54	34547 → 48906 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
4059	257.137579963	141.22.27.107	141.22.27.106	RMI	124	JRMI, Call
4060	257.137600947	141.22.27.106	141.22.27.107	TCP	66	1099 → 51392 [ACK] Seq=903 Ack=304 Win=29056 Len=0 TSval=690167450 TSecr=2547012745
4061	257.137945690	141.22.27.107	141.22.27.106	RMI	360	JRMI, ReturnData
4062	257.138526297	141.22.27.107	141.22.27.106	RMI	81	JRMI, DgcAck
4063	257.138707828	141.22.27.107	141.22.27.106	TCP	74	48908 → 34547 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=2547012747 TSecr=0 WS=128
4064	257.138725331	141.22.27.106	141.22.27.107	TCP	54	34547 → 48908 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
4065	257.139097936	141.22.27.107	141.22.27.106	RMI	124	JRMI, Call
4066	257.139129494	141.22.27.106	141.22.27.107	TCP	66	1099 → 51392 [ACK] Seq=1197 Ack=377 Win=29056 Len=0 TSval=690167451 TSecr=2547012747
4067	257.139488565	141.22.27.107	141.22.27.106	RMI	360	JRMI, ReturnData
4068	257.140427758	141.22.27.107	141.22.27.106	RMI	81	JRMI, DgcAck
4069	257.141337047	141.22.27.107	141.22.27.106	TCP	74	48910 → 34547 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=2547012750 TSecr=0 WS=128
4070	257.141356147	141.22.27.106	141.22.27.107	TCP	54	34547 → 48910 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
4071	257.141750671	141.22.27.107	141.22.27.106	RMI	124	JRMI, Call
4072	257.141777683	141.22.27.106	141.22.27.107	TCP	66	1099 → 51392 [ACK] Seq=1491 Ack=450 Win=29056 Len=0 TSval=690167454 TSecr=2547012749
4073	257.142140018	141.22.27.106	141.22.27.107	RMI	360	JRMI, ReturnData
4074	257.142662401	141.22.27.107	141.22.27.106	RMI	81	JRMI, DgcAck
4075	257.142846316	141.22.27.107	141.22.27.106	TCP	74	48912 → 34547 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=2547012751 TSecr=0 WS=128
4076	257.142862533	141.22.27.106	141.22.27.107	TCP	54	34547 → 48912 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
4077	257.143215489	141.22.27.107	141.22.27.106	RMI	124	JRMI, Call
4078	257.143247636	141.22.27.106	141.22.27.107	TCP	66	1099 → 51392 [ACK] Seq=1785 Ack=523 Win=29056 Len=0 TSval=690167455 TSecr=2547012751
4079	257.143601897	141.22.27.107	141.22.27.106	RMI	360	JRMI, ReturnData
4080	257.144113778	141.22.27.107	141.22.27.106	RMI	81	JRMI, DgcAck
4081	257.144290075	141.22.27.107	141.22.27.106	TCP	74	48914 → 34547 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=2547012753 TSecr=0 WS=128
4082	257.144306834	141.22.27.106	141.22.27.107	TCP	54	34547 → 48914 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
4083	257.144675307	141.22.27.107	141.22.27.106	RMI	124	JRMI, Call
4084	257.144699877	141.22.27.106	141.22.27.107	TCP	66	1099 → 51392 [ACK] Seq=2079 Ack=596 Win=29056 Len=0 TSval=690167457 TSecr=2547012752
4085	257.145152789	141.22.27.107	141.22.27.106	RMI	360	JRMI, ReturnData
4086	257.145664209	141.22.27.107	141.22.27.106	RMI	81	JRMI, DgcAck
4087	257.145863054	141.22.27.107	141.22.27.106	TCP	74	48916 → 34547 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=2547012754 TSecr=0 WS=128
4088	257.145880273	141.22.27.106	141.22.27.107	TCP	54	34547 → 48916 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

4 – Serverabsturz

Server Neustart:



1099…	269.569166944	141.22.27.107	141.22.27.106	TCP	74	[TCP Port numbers reused] 55786 → 34547 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=2547025177 TSecr=0 WS=1
1099…	269.569174265	141.22.27.106	141.22.27.107	TCP	54	34547 → 55786 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
1099…	269.569422923	141.22.27.107	141.22.27.106	RMI	124	JRMI, Call
1099…	269.569442023	141.22.27.106	141.22.27.107	TCP	66	1099 → 51392 [ACK] Seq=5162578 Ack=1282103 Win=29056 Len=0 TSval=690179881 TSecr=2547025177
1099…	269.569505793	141.22.27.106	141.22.27.107	RMI	360	JRMI, ReturnData
1099…	269.570520302	141.22.27.107	141.22.27.106	TCP	74	54716 → 36069 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=2547025179 TSecr=0 WS=128
1099…	269.570535115	141.22.27.106	141.22.27.107	TCP	74	36069 → 54716 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=690179883 TSecr=2547025179 WS=128
1099…	269.570710920	141.22.27.107	141.22.27.106	TCP	66	54716 → 36069 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=2547025179 TSecr=690179883
1099…	269.570852791	141.22.27.107	141.22.27.106	TCP	73	54716 → 36069 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=7 TSval=2547025179 TSecr=690179883
1099…	269.570860658	141.22.27.106	141.22.27.107	TCP	66	36069 → 54716 [ACK] Seq=1 Ack=8 Win=29056 Len=0 TSval=690179883 TSecr=2547025179
1099…	269.571174924	141.22.27.106	141.22.27.107	TCP	86	36069 → 54716 [PSH, ACK] Seq=1 Ack=8 Win=29056 Len=20 TSval=690179883 TSecr=2547025179
1099…	269.571341888	141.22.27.107	141.22.27.106	TCP	66	54716 → 36069 [ACK] Seq=8 Ack=21 Win=29312 Len=0 TSval=2547025179 TSecr=690179883
1099…	269.571428055	141.22.27.107	141.22.27.106	TCP	85	54716 → 36069 [PSH, ACK] Seq=8 Ack=21 Win=29312 Len=19 TSval=2547025179 TSecr=690179883
1099…	269.571695868	141.22.27.107	141.22.27.106	TCP	517	54716 → 36069 [PSH, ACK] Seq=27 Ack=21 Win=29312 Len=451 TSval=2547025180 TSecr=690179883
1099…	269.571709853	141.22.27.106	141.22.27.107	TCP	66	36069 → 54716 [ACK] Seq=21 Ack=478 Win=30080 Len=0 TSval=690179884 TSecr=2547025179
1099…	269.572602794	141.22.27.106	141.22.27.107	TCP	353	36069 → 54716 [PSH, ACK] Seq=21 Ack=478 Win=30080 Len=287 TSval=690179885 TSecr=2547025179
1099…	269.573405073	141.22.27.107	141.22.27.106	RMI	81	JRMI, DgcAck
1099…	269.573524380	141.22.27.107	141.22.27.106	TCP	128	54716 → 36069 [PSH, ACK] Seq=478 Ack=308 Win=30336 Len=62 TSval=2547025182 TSecr=690179885

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.

The Kiet Dang
Sofian Wüsthoff

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