TKN Telecommunication Networks Group

# Praktikum
# Rechnernetze und Verteilte Systeme

Block 3

— Verteilte Systeme und RPC —

**Termin: 12.-16.11.2018 & 19.-23.11.2018**

## 1 Theoretische Vorbereitungsaufgaben

Die folgenden Aufgaben sollen Ihnen helfen, sich auf den Vorbereitungstest vorzubereiten. Klären Sie bitte mögliche Fragen oder Unklarheiten unbedingt vor den ISIS-Testaten!

**Aufgabe 1:**
Sie kennen Sich aus dem privaten und universitären Umfeld mit dem Konzept der "elektronischen Post" (E-Mail) aus. Nutzen sie das Beispiel des Abrufs von E-Mails durch Clients bei einem Server, um die folgenden Fragen für das Client-Server-Prinzip im Allgemeinen zu beantworten:

- Muss der betreffende Rechner (Client/Server) immer angeschaltet und ans Netz angeschlossen sein? Was hätte es für Konsequenzen, wenn beide Kommunikationspartner gleichwertig sind, also es keinen expliziten Server bzw. Client gibt?
- Welche Art von Adresse haben Client und Server? Was hätte es für Konsequenzen, wenn beide Kommunikationspartner gleichwertig sind, also es keinen expliziten Server bzw. Client gibt?
- Mit wem kommunizieren Clients in der Regel direkt? Mit wem Server?
- Wo sehen Sie Performanz-/Skalierbarkeitsprobleme?

**Aufgabe 2:**
Sie wollen einen Service per "Remote Procedure Call" (RPC) aufrufen und wissen aus der Vorlesung, dass Sie sich dazu zunächst eine Antwort auf folgende Fragen überlegen müssen:

a) Was ist im Allgemeinen ein Service?

b) Was bedeutet "idempotent" und wie vereinfacht eine solche idempotente Operation den RPC-Aufruf?

c) Stellen Sie die Semantiken "at-most-once" und "at-least-once" gegenüber.

d) Was ist der Unterschied zwischen asynchronem und synchonem RPC?

e) Stellen Sie kurz die Übergabesemantiken *call-by-value* und *call-by-reference* des *klassischen* Prozeduraufrufs gegenüber.

f) Sie möchten RPC mit den zuvor genannten Übergabesemantiken verwenden. Bei welcher Semantik sehen sie die größeren Probleme bezüglich der Umsetzbarkeit und wie lösen sie diese?

g) Welche Formen der Transparenz gibt es? Welche Form der Transparenz steht für Sie beim RPC-Konzeptes im Vordergrund?

**Aufgabe 3:**
Ein Client nutzt synchrones RPC, um eine entfernte Prozedur mit Rückgabewert aufzurufen. Der Client braucht initial 5 Millisekunden, um seinen eigenen lokalen Client Stub mit den jeweiligen Parametern aufzurufen. Der Server braucht 10 Millisekunden, um seine lokale Prozedur aufzurufen und das Ergebnis zu erhalten. Die Verarbeitungsverzögerung (Processing Delay) für jede Sende- oder Empfangsoperation bei Client und Server sind jeweils 0,5 Millisekunden. Alle übrigen Verzögerungen (Queueing Delay (Warteschlangenverzögerung), Transmission Delay (Übertragungsverzögerung) und Propagation Delay (Ausbreitungsverzögerung)) addieren sich zwischen Client und Server in jeder Richtung auf jeweils 3 Millisekunden. Marshalling und Unmarshalling benötigen jeweils 0,5 Millisekunden.

- Wie lange dauert ein Aufruf vom lokalen Aufruf des Client Stubs bis zur Rückgabe des Ergebnisses an die aufrufende Prozedur?[1]

## 2 Präsenzaufgaben

Die folgenden Aufgaben werden im Termin unter Anleitung des Tutors durchgeführt.

**Aufgabe 4:**
**Tafelaufgabe** - Wird im Termin vorgeführt. Mitarbeit und damit Vorbereitung werden aber vorausgesetzt.
Sie möchten aus Ihrem Programm, das von einem 32-Bit-Prozessor mit Little-Endian-Darstellung ausgeführt wird, einen RPC aufrufen, der als Argument ein Paar aus ID und Namen verlangt. Das Programm speichert das Paar intern wie in Listing 1 dargestellt.

a) Worauf müssen Sie beim (Un-)Marshalling dieses Werte-Paares achten?

b) Ein weiterer RPC benötigt eine Liste dieser Werte-Paare. Die Liste wird intern gemäß Listing 2 gespeichert. Welche zwei zusätzlichen Probleme können hier auftreten?

c) Treten alle Probleme aus b) auch bei einem binären Baum wie in Listing 3 auf? (Tipp: Welche speziellen Eigenschaften besitzt ein Baum in der Informatik?)

Listing 1: Werte-Paar

```
typedef struct ValueType {
    int          id;
    const char*  name;
} ValueType;
```

Listing 2: Doppelt-verkettete Liste

```
typedef struct ListNode {
    ValueType  value;
    ListNode*  prev;
    ListNode*  next;
} ListNode;

ListNode* myListHead;
ListNode* myListTail;
```

[1] Folien 34-36 von Unit 3 sollten hier helfen. Kontrollergebnis: 25ms

Listing 3: Binärer Baum

```
typedef struct TreeNode {
    ValueType  value;
    TreeNode*  left;
    TreeNode*  right;
} TreeNode;

TreeNode* myTreeRoot;
```

# 3 Praktische Aufgaben

Die praktischen Aufgaben sind in Kleingruppen von i. d. R. 4 Personen zu lösen. Die Ergebnisse des ersten Termins führen Sie im zweiten Termin dem Tutor vor. Reichen Sie bitte des weiteren den Quelltext bzw. Lösungen bis Sonntag vor dem dem zweiten Termin 23:55 Uhr per ISIS ein.

Im zweiten Termin werden vertiefende praktische Aufgaben gestellt, während der Tutor Lösungen des ersten Termins abnimmt. Reichen Sie bitte den Quelltext bzw. Lösungen dieser Aufgaben bis Sonntag vor dem nächsten Termin 23:55 Uhr per ISIS ein.

Es besteht in beiden Terminen grundsätzlich Anwesenheitspflicht.

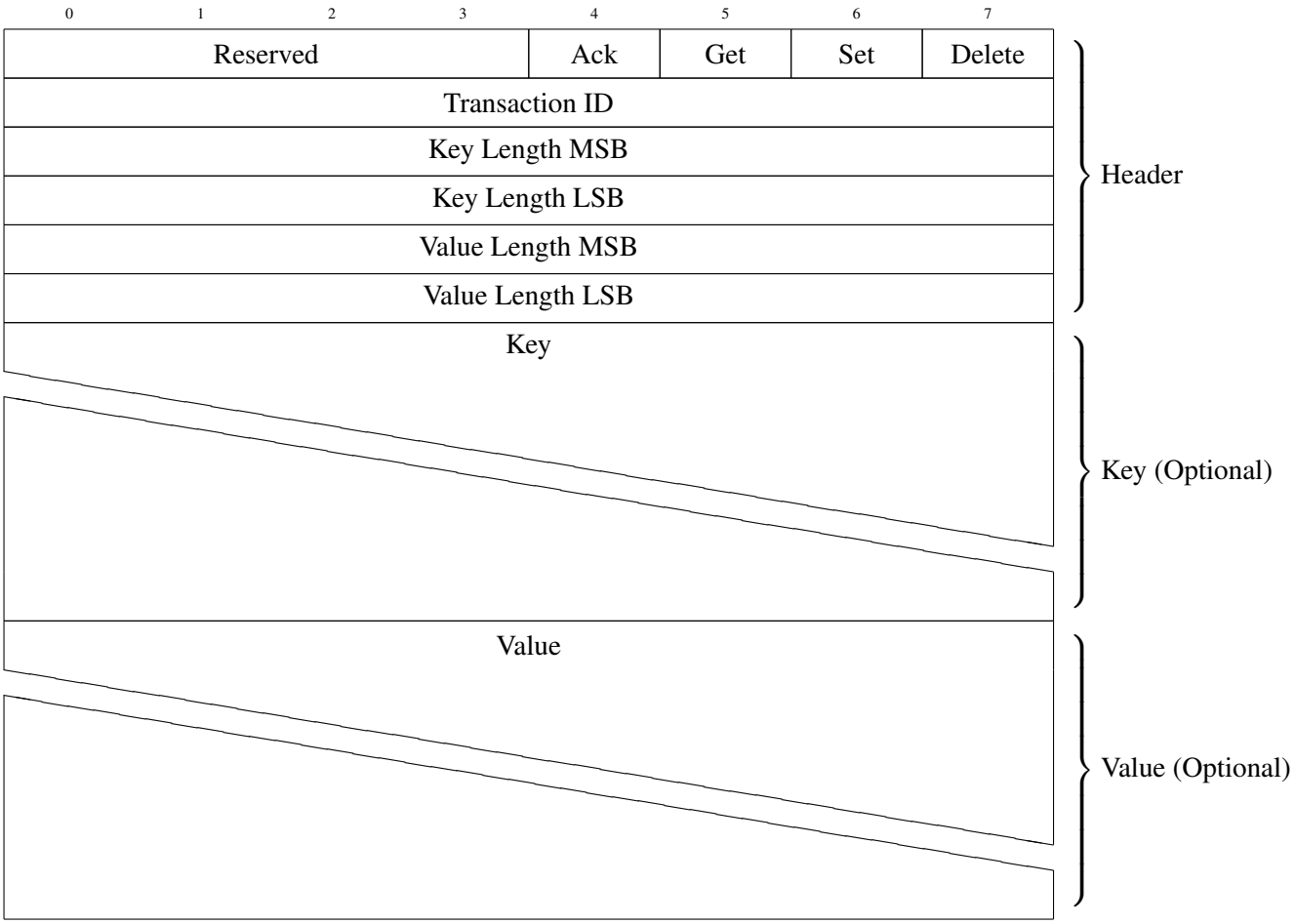
## 3.1 In der ersten Woche zu lösen

**Aufgabe 5:**
Sie haben in Block 2 einen Stream-Socket Client und Server bzw. einen Datagram-Socket Client geschrieben. Sie haben damit schon eine Art "Remote Procedure Call" (RPC) auf der Serverseite angeboten bzw. auf der Clientseite aufgerufen. Dabei war die Rückgabe eines Zitats der angebotene Service.

Wir wollen auf den nächsten Blättern eien Art Micro-Service implementieren, den ein imaginärer Streaminganbieter[2] möglicherweise benötigt, um seinen Kunden die Plattform bereitzustellen. Die erste Aufgabe ist die Implementierung einer Film- bzw. Seriendatenbank. Wir wollen dazu der Einfachheit halber eine Hash-Table benutzen. In einer Hash-Table werden Tupel der Form `<key, value>` gespeichert[3]. Ihre Datenbank sollte die Befehle `set()` um einen Wert zu speichern oder zu aktualisieren, `get()` um einen Wert auszulesen und `delete()` um einen Wert zu löschen anbieten.

Client und Server verwenden für Aufrufe bzw. Rückgabewerte ein Protokoll, welches wir hier vorgeben, damit im Anschluss die Server und Clients aller Gruppen miteinander arbeiten können und ein automatisches Testen möglich ist. Halten Sie sich deshalb genau an das Protokoll! Das Protokoll basiert auf TCP, das heißt die unten angegebenen Nachrichten werden über einen zuverlässigen Bytestrom geschickt. Mehrere Nachrichten werden potentiell über eine einzige Verbindung gesendet, d.h. es wird nicht unbedingt für jede Nachricht eine neue Verbindung aufgebaut!

[2] `https://www.netflix.com/`
[3] `https://de.wikipedia.org/wiki/Hashtabelle`




Das Protokoll ist aufgeteilt in einen festen Header, den Key mit variabler Länge und einen optionalen Value variabler Länge. Der Header enthält zunächst einige reservierte Bits. Diese sollen mit 0 gefüllt werden und könnten später für ein Versions-Feld benutzt werden. Danach folgen Bits für die Befehle bzw. die Server-Antwort. Ein gesetztes Bit bedeutet hier, dass es sich bei der Nachricht um diesen Anfragetyp handelt. Der Client kann hier das Get, Set oder Delete Bit setzen. Es soll dabei angenommen werden, dass immer nur ein Bit gesetzt ist. Der Server bestätigt alle Befehle bei erfolgreicher Durchführung mit einem Acknowledgment und setzt dafür das entsprechende Bit. Der Server setzt auch die Bits für die Aktion, die erfolgreich durchgeführt wurde. Darauf folgt eine Transaction-ID, die der Client beliebig setzen kann und in der Serverantwort unverändert enthalten ist. Sie ermöglicht dem Client im Prinzip die Zuordnung des Acknowledgments zu der entsprechenden Anfrage.

Danach folgt die Länge des Keys und die Länge des Values, jeweils als 16bit vorzeichenlose Ganzzahl. Alle Angaben beziehen sich falls nicht anders angegeben auf die Network-Byte-Order! Get und Delete-Anfragen enthalten nur einen Key, so dass die Länge des Values auf 0 gesetzt wird. Antworten auf Get-Anfragen enthalten Key und Value, die anderen Antworten enthalten weder Key noch Value. Nach diesem Header wird je nach Anfrage bzw. Antwort der Key gesendet, welcher variabel Lang sein kann. Darauf folgt optional der Value, der ebenfalls variabler Länge ist. Key und Value sind dabei nicht unbedingt Null-terminiert. Der Key enhält keine Null Bytes. Der Value kann beliebege Bytes, also auch das Null-Byte enthalten. Diese sollten dann entsprechend mitgespeichert werden. Der Hashtable-Teil kann beliebig in C implementiert werden, z.B. mit Hilfe von uthash[4].

Halten Sie sich genau an die Vorgaben des Protokolls. Als Parameter soll ihr Server den Port übergeben bekommen. Ein Beispielaufruf sieht dann so aus:

```
./server 4711
```

Sie können Ihre Implementierung mit dem von uns bereitgestellten Client testen.

[4] `https://troydhanson.github.io/uthash/`

Reichen Sie bitte ihre Lösungen bis Sonntag vor dem zweiten Termin 23:55 Uhr per ISIS ein. Die Lösungen werden **automatisch überprüft** und müssen ohne Ausnahme in einer Datei mit dem Namen **Block3a.TXXGYY.tar.gz** (XX = Praktikumstermin, YY = Gruppennummer) eingereicht werden. Darin enthalten erwarten wir in einem Ordner mit dem Namen Block3a.TXXGYY ein Makefile, welches die Binärdatei `server` entsprechend kompiliert.

### 3.2 In der zweiten Woche zu lösen

**Aufgabe 6:**
Nehmen Sie als Ausgangspunkt Ihre Implementierung aus der letzten Aufgabe. Sie wollen nun zu ihrem Server den entsprechenden Client schreiben. **Der Client soll dabei beim Aufruf der Methoden Stubs benutzen, um die entfernte Ausführung des Prozeduraufrufs und damit einhergehende Übertragung von Argumenten und Ergebnis weitestgehend zu verstecken.** Gestalten Sie also die Schnittstellen bzw. Methodenaufrufe im Client so, als wäre die Funktionalität des entfernten Systems lokal vorhanden. Die nötige Vorbereitung des Sockets bzw. das Schließen darf weiterhin explizit passieren.

Der Client soll als Kommandozeilenargument den DNS-Namen bzw. die IP-Adresse des Servers, die Methode (`SET`, `GET`, `DELETE`), den Key und optional eine Value übergeben bekommen. Der Client soll dann die entsprechende Funktion ausführen. Geben Sie nur bei `GET` den zurückgegebenen Value ohne zusätzliche Zeichen aus und ansonsten nichts, damit wir automatisch testen können. Einige Beispielaufrufe:

```
./client localhost 4711 SET TKN abc123
./client localhost 4711 GET TKN
./client localhost 4711 DELETE TKN
```

Sie sollen darüber hinaus (trotz TCP) garantieren, dass der Server die Anfrage auch tatsächlich mindestens einmal ausführt bzw. der Client nicht endlos auf eine Antwort wartet, wenn ein Fehler auftritt. Implementieren Sie deswegen eine at-least-once Semantik mit einem Timeout von 2 Sekunden, z.B. mit Hilfe von `select()`.

Reichen Sie bitte ihre Lösungen bis Sonntag nach dem zweiten Termin 23:55 Uhr per ISIS ein. Die Lösungen werden **automatisch überprüft** und müssen ohne Ausnahme in einer Datei mit dem Namen **Block3b.TXXGYY.tar.gz** (XX = Praktikumstermin, YY = Gruppennummer) eingereicht werden. Darin enthalten erwarten wir in einem Ordner mit dem Namen Block3b.TXXGYY ein Makefile, welches die Binärdatei `client` entsprechend kompiliert.

## 4 Vertiefungsaufgaben

Diese Aufgaben sind zu Ihrer eigenen Vertiefung in Hinblick auf die Klausurvorbereitung gedacht:

**Aufgabe 7:**
Was bedeuten im Kontext von verteilten Systemen folgende Ausdrücke? Achten Sie auf eine möglichst genaue Definition!

- Autonomie
- Transparenz
- Skalierbarkeit
- Middleware

**Aufgabe 8:**
Sie haben in der Vorlesung gelernt, dass eine wesentliche Eigenschaft eines verteilten Systems das Verbergen der Verteilung ist. In diesem Zusammenhang haben Sie den Begriff der Transparenz kennengelernt. Welche Formen der Transparenz werden bei folgenden Beispielen (a und c) realisiert? Erläutern Sie diese bitte jeweils kurz!

a) Beim NFS (Network Filesystem) wird ein auf einem Server befindliches Dateisystem beim Client in den lokalen Verzeichnisbaum eingehängt und erscheint wie ein lokales Verzeichnis mit den entsprechenden Dateien und Unterverzeichnissen des NFS Volumes. Programme greifen auf Dateien und Verzeichnisse des NFS Volumes mit denselben Systemaufrufen und Bibliotheksfunktionen zu, wie auf lokale Dateien/Verzeichnisse. Bezeichnet ein Name eine Datei auf dem NFS Volume, so werden Zugriffe auf die Datei automatisch durch das Betriebssystem mittels RPCs (Remote Procedure Calls) über das Netzwerk zum Server übertragen und ggf. der entsprechende Teil der Datei geändert (Schreiboperation) oder an den Client übertragen (Leseoperation). Ist dies nicht erfolgreich, wird der RPC ggf. mehrfach wiederholt. Mehrere Clients können gleichzeitig lesend und schreibend auf verschiedene Dateien des Volumes oder nur lesend auf die selben Dateien zugreifen. Wird schreibend zugegriffen, so erscheint die Datei ggf. als hätte sie sich von selbst verändert, oder eine Datei kann explizit gegen gleichzeitigen Zugriff geschützt werden. Wird ein NFS Volume auf einen anderen Server verschoben, so kann nun dieses neue Volume unter dem bisher bekannten Verzeichnis eingehängt werden.

b) Ist vollständige Failure Transparency realisierbar? Wie sieht das insbesondere in Hinblick auf das vorherige Beispiel aus?

c) Ein Drucker wird über seinen Namen angesprochen. Zum Ausdrucken werden Druckjobs als Postscript-Dateien und unter Angabe des Druckernamens an den Druck-Server geschickt. Dieser konvertiert den Druckjob gegebenenfalls in die Sprache des Druckers, wenn der Drucker nicht Postscript-fähig ist. Mehrere gleichzeitige Druckjobs werden automatisch in eine Warteschlange eingereiht. Ist der Drucker überlastet (die Warteschlange zu lang) oder gar defekt, wird der Job automatisch an einen anderen Drucker weitergeleitet.

d) Sind alle beim vorherigen Beispiel auftretenden Formen der Transparenz überhaupt sinnvoll? Begründen Sie Ihre Antwort.

**Aufgabe 9:**
Nennen Sie jeweils 3 Vor- und Nachteile der Client-Server-Architektur.

**Aufgabe 10:**
Die Zuordnung von Domainnamen zu IP-Adressen erfolgt im Internet mit Hilfe des Domain Name Systems (DNS). Beschreiben Sie anhand der folgende Fragen den Aufbau und die Funktionsweise von DNS:

a) Wie sind die Verantwortlichkeiten im Namenssystem aufgeteilt?

b) Wie funktioniert das rekursive bzw. iterative Auflösen von Namen mit DNS?

c) Welche Funktion erfüllen die TTL-Einträge?