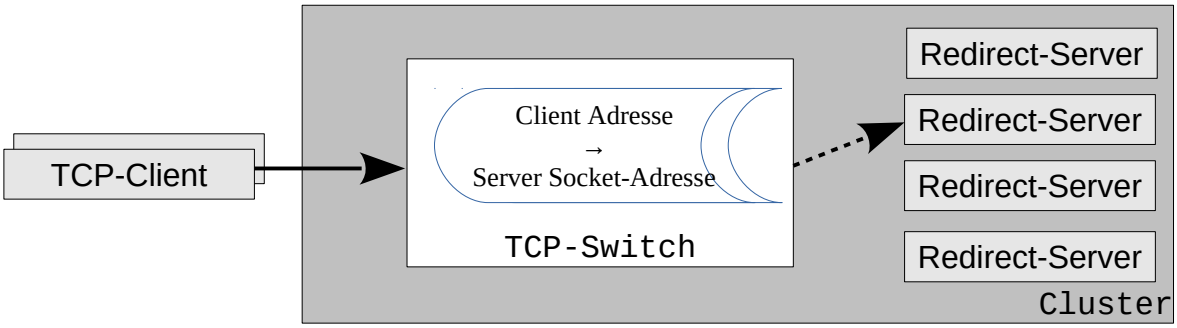
Verteilte Systeme

Übung C1

Bearbeitungszeit 2 Wochen

Die Übung adressiert den Umgang mit TCP sowie Multi-Programming mittels Verteilung von Protokoll-Anfragen in Clustern. Weiterführende Informationen zu den TCP- und HTTP-Protokollen gibt es unter https://en.wikipedia.org/wiki/Tcp_protocol, sowie unter https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocolprotocol.

Aufgabe: TCP-Weiche



Kopiert die Klasse **`TcpSwitchServerSkeleton`** nach **`TcpSwitchServer`**. Diese Klasse repräsentiert den Rahmen für einen TCP-Switch Server im *Acceptor/Service* Multithreading-Entwurfsmuster. Implementiert die run()-Methode des Connection-Handlers unter Beachtung der dort unter TODO hinterlegten Hinweise, und testet Eure Implementierung durch Starten von **`TcpSwitchApplication`**.

Bei *session-awareness=false* sollte die TCP-Switch alle ankommenden TCP-Verbindungen zufällig auf die Redirect-Server verteilen: Im Falle von HTTP-Servern kommt die HTML-Seite von einem der Zielserver, eine CSS-Datei vom anderen, usw. Verwendet dazu die Klasse **`ThreadLocalRandom`**, welche einen Zufallszahlengenerator pro Thread bereit stellt, sowie die in der Instanzvariable *`redirectHostAddresses`* gespeicherten Socket-Adressen (IP-Adresse & Port) der Redirect-Server. Achtet beim Verbindungsaufbau darauf dass ein ausgewählter Knoten temporär unverfügbar sein kann, dann sollte ein anderer ausgewählt werden (nur bei *session-awareness=false* möglich)!

Bei *session-awareness=true* dagegen soll nur bei der ersten Anfrage eines Clients einer der nachgeschalteten Zielserver zufällig ausgewählt, und diese Auswahl dann für alle nachfolgenden Anfragen desselben Clients wiederverwendet werden. Diese Auswahl soll basierend auf der Client-Adresse (`clientConnection.getInetAddress()`) erfolgen, ein Verfahren welches für Web 1.0 Applikationen wesentlich ist.

Dabei sind zwei Verfahren möglich:

- **Low-Tech** → Verwendung einer Instanzvariable in **`TcpSwitch`** welche Client-Adressen auf selektierte Knoten-Adressen abbildet (z.B. `Map<InetAddress,InetSocketAddress>`). Falls eine Client-Adresse beim Verbindungsaufbau in der Map noch nicht als Schlüssel registriert ist, muss die Socket-Adresse eines Knotens zufällig ausgewählt, und in der Map zusammen mit der Client-Adresse registriert werden. Problem: Memory-Footprint für die Map, sowie notwendige Thread-Synchronisation beim Zugriff auf dieselbe!
- **High-Tech** → Abbildung der Client-Adresse auf die Socket-Adresse eines Knoten mittels eines Scramblers, i.e. eines Zufallszahlengenerators der bei der Konstruktion mit dem Hash-Code der Client-Adresse gesendet wurde. Solche gesendeten Generatoren haben die Eigenschaft für eine gegebene Client-Adresse immer die gleiche nächste „Zufallszahl“ zu generieren, was zur Abbildung der Client-Adresse auf einen „zufälligen“ Knoten-Index genutzt werden kann.

In beiden Fällen gilt die Auswahl der zugeordneten Knoten-Adresse für lange Zeit, daher ist es hier nicht notwendig den Fall zu behandeln dass ein Knoten temporär unverfügbar ist; schließlich kann es auch passieren dass der gewählte Knoten kurz nach der Selektion unverfügbar wird.

Vor starten der TCP-Weiche müssen zuerst mindestens zwei Web-Server Knoten (z.B. zwei `de.htw.ds.tcp.HttpContainer`, oder zwei Apache-Instanzen) mit identischem Inhalt (ein Webserver-Cluster) auf den Ports 8001, 8002, usw. gestartet werden. Verwendet z.B. die SelfHtml-Seiten für den Inhalt der Knoten-Instanzen, wie in der Übung. Startet Euren TCP-Switch dann mit den folgenden Parametern:

- 8010 (Service-Port)
- true/false (Session-Awareness)
- localhost:8001 (IP-Socketadresse des Zielservers 1)
- localhost:8002 (IP-Socketadresse des Zielservers 2)
- usw.