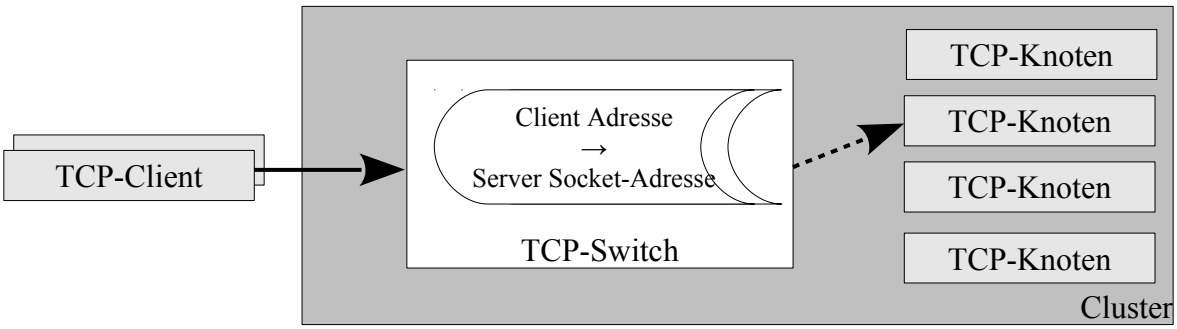
Verteilte Systeme

Übung C1 - Einzelarbeit

Bearbeitungszeit 2 Wochen

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

Aufgabe: TCP-Weiche



Kopiert die Klasse *TcpSwitchSkeleton* nach *TcpSwitch*. Diese Klasse repräsentiert den Rahmen für einen TCP-Server im Acceptor/Service Multithreading-Entwurfsmuster. Implementiert die run()-Methode des Connection-Handlers unter Beachtung der dort unter TODO hinterlegten Hinweise:

Bei session-awareness=false sollte die TCP-Weiche alle ankommenden TCP-Anfragen zufällig auf die Zielserver verteilen: Im Falle von HTTP-Servern kommt die HTML-Seite von einem der Zielserver, das CSS-File vom anderen, usw. Verwendet dazu den unter der statischen Variable „RANDOMIZER“ zur Verfügung stehenden Zufallszahlengenerator, sowie die in der Instanzvariable „nodeAddresses“ gespeicherten Socket-Adressen (IP-Adresse & Port) der Zielserver. Achtet beim Verbindungsaufbau darauf dass ein ausgewählter Knoten temporär unverfügbar sein kann!

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. Dabei sind zwei Verfahren möglich:

- **Low-Tech** → Verwendung einer Instanzvariable in TcpSwitch welche Client-Addressen auf selektierte Knoten-Addressen abbildet (z.B. Map<InetAddress,InetSocketAddress>). Falls eine Client-Addresse 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.
- **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 geseedet wurde. Solche geseedeten 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.http.HttpServer2, de.htw.ds.http.HttpContainer1*, oder zwei Apache-Instanzen) mit identischem Inhalt (ein Webserver-Cluster) auf den Ports 8002, 8003, usw. gestartet werden. Verwendet 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:8002 (IP-Socketadresse des Zielservers 1)
- localhost:8003 (IP-Socketadresse des Zielservers 2)
- usw.

Beachtet dass das Cluster-Konzept nur funktioniert wenn die Ziel-Knoten dieselben Ressourcen unter denselben URL-Pfäden zur Verfügung stellen, achtet also darauf dass dies zutrifft! Falls ihr die Memos im *MemoChat-Beispiel* zum Testen verwenden möchten, müssen zudem

- Instanzen von *HttpContainer1* als Knoten verwendet werden
- die RequestHandler-Klassen aus der JAR-Datei *distributed-systems-plugins.jar* unter `de/htw/ds/http/handler` in das Kontext-Verzeichnisse der Knoten entpackt werden
- in der URL-Zeile eines Browsers das MemoChat Beispiel mittels der URL http://localhost:8010/de.htw.ds.http.handler.MemoChat.java geladen werden.