Leistungsnachweis in

Rechnerkommunikation

Sommersemester 2009
5. Oktober 2009

Name: _____
Matrikelnummer: _____
Geburtsdatum: _____
Studienfach: _____
Fachsemester: _____

- Bitte verwenden Sie einen blauen oder schwarzen Kugelschreiber (kein rot, keinen Bleistift).
- Schriftliche Aufzeichnungen (sowohl eigene Aufzeichnungen wie auch Bücher) sind als Hilfsmittel zugelassen. Auch ein Taschenrechner ist erlaubt und hilfreich. Nicht zugelassen sind dagegen Computer, PDAs, Mobiltelefone und sonstige Kommunikationsmittel.
- Legen Sie den Ausweis (mit Lichtbild) griffbereit auf den Platz.
- Bitte überprüfen Sie, ob Sie alle 19 Blätter erhalten haben.
- Schreiben Sie die Antworten jeweils in den freien Raum hinter den Fragen. Sollte dieser nicht ausreichen, steht noch freier Raum am Ende der Klausur zur Verfügung. Bitte kennzeichnen Sie dort deutlich, welche Aufgabe Sie bearbeiten. Gesondert beigelegte Blätter werden nicht bewertet!
- Schreiben Sie deutlich! Unleserliche Antworten gehen nicht in die Bewertung ein!
- Die Arbeitskopien können der Klausur entnommen werden und müssen nicht mit abgegeben werden.

Ich habe die Hinweise auf dieser Seite zur Kenntnis genommen und alle 19 Blätter der Klausur empfangen:

_____
Unterschrift

Bewertung:

1	2	3	4	Σ

1 Allgemeine Fragen (20 Punkte)

a) (2 Punkt) Was ist Timed Wait beim Verbindungsabbau von TCP und wofür wird es verwendet?

b) (2 Punkte) Nennen Sie stichpunktartig 4 Einsatzbereiche des Netzwerkmanagements nach ISO.

c) (1 Punkt) Nennen Sie 2 Realisierungs-Möglichkeiten für die Switching Fabric.

d) (2 Punkte) Beschreiben Sie kurz die Bedingungen, bei denen das Hidden-Terminal Problem auftritt.

e) (2 Punkte) Was bedeutet die Abkürzung SDP und was ist dessen Aufgabe?

f) (1 Punkte) Nennen Sie einen Vor- und einen Nachteil des Pseudo-Headers bei UDP.

g) (3 Punkte) Was bedeutet die Abkürzung FDMA? Welche Möglichkeit für den Mehrfachzugriff beim Medienzugriff realisiert FDMA? Nennen Sie außerdem 2 weitere Möglichkeiten für den Mehrfachzugriff und hierfür jeweils eine Beispiel-Technologie.

h) (2 Punke) Was bedeutet die Abkürzung MPLS und für was ist es ein Beispiel? Nennen Sie außerdem die Wurzeln und einen Vorteil von MPLS.

i) (2 Punkte) Nennen Sie 4 Möglichkeiten der Leitungskodierung.

j) (3 Punkte) Nennen und beschreiben Sie die 3 wesentlichen Aspekte der Strukturierung in Schichten. Verwenden Sie hierbei Fachbegriffe.

2 Verzögerungszeiten (28 Punkte)

In folgenden wird von der Netzwerk-Topologie gemäß Abbildung 1 ausgegangen. Host C und Host S können nur indirekt über Host G miteinander kommunizieren. Mit Host G sind noch weitere Netzwerkteilnehmer verbunden, die ebenfalls Daten mit Host S austauschen wollen. Dadurch entsteht ein sogenannter "bottleneck" (Engpass) der dazu führt, dass alle Pakete, die über Host G zu Host S gesendet werden, in einer Warteschlange zwischengespeichert werden müssen. Alle anderen Verbindungen müssen nicht gepuffert werden. Pakete benötigen in den einzelnen Knoten keine Verarbeitungszeit. Gehen Sie weiterhin davon aus, dass alle Links symmetrisch sind und die Warteschlange nie überläuft. Ein Paketverlust tritt ebenfalls nicht auf.

Verwenden Sie für Ihre Berechnungen die Werte gemäß Tabelle 1. Für die Warteschlangenverzögerung können Sie von einer konstanten mittleren Verzögerung, die gemäß Formel 1 berechnet werden kann, ausgehen. Die Ausbreitungsverzögerung für den Link zwischen Host C und G ist so gering, dass sie vernachlässigt werden kann (idealisiert: $l_1 = 0\ m$).

$$d_{queue} = \frac{\rho L}{R(1-\rho)} \tag{1}$$

Ausbreitungsgeschwindigkeit:	$c = 2 \cdot 10^8\ \frac{m}{s}$
Leitungslängen:	$l_1 = 0\ m$ $l_2 = 200\ km$
Raten:	$R_1 = 100\ \frac{Mbit}{s}$ $R_2 = 1\ \frac{Mbit}{s}$
Paketlänge:	$L = 1250\ Byte$
Verkehrsintensität:	$\rho = \frac{1}{3}$

Tabelle 1: Konstanten

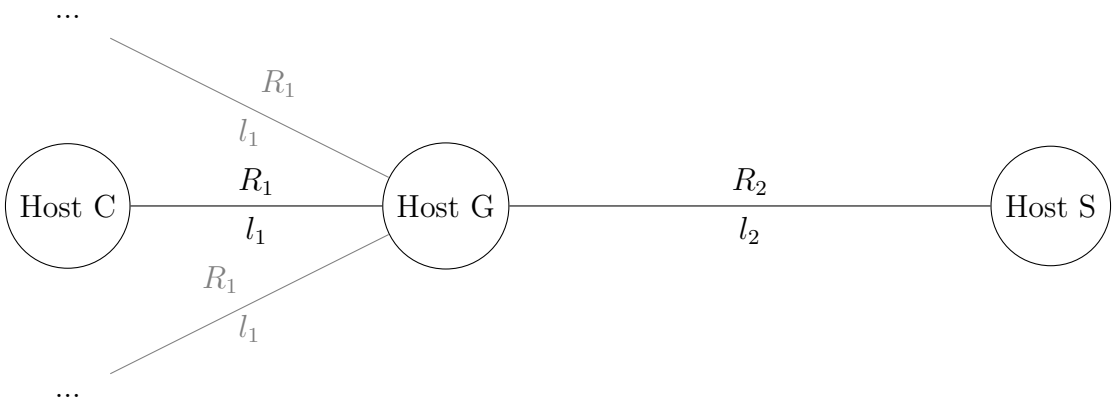

Abbildung 1: Netzwerk-Topologie

a) (2 Punkte) Berechnen Sie die Ausbreitungsverzögerung für den Link zwischen Host C und G (d_{propCG}) sowie für den Link zwischen Host G und S (d_{propGS}).

b) (8 Punkte) Nehmen Sie an, Host C baut zu Host S eine TCP Verbindung auf, um Daten von Host S zu erhalten. Host S beginnt ohne Anfrage so bald wie möglich mit dem Senden der Daten. Gehen Sie in dieser Teilaufgabe davon aus, dass der Link zwischen Host C und G eine geringe Ausbreitungsverzögerung besitzt ($d_{propCG} > 0;\ d_{propCG} \ll d_{propGS}$).

Zeichnen Sie in Abbildung 2 ein Ablaufdiagramm, unter Berücksichtigung oben genannter Eigenschaften des Netzwerks, bis einschließlich des zweiten Sendefensters und dessen Bestätigung. Berücksichtigen Sie in Ihrer Skizze die Host-Namen und qualitativ die unterschiedlichen Ausbreitungszeiten der Links. Markieren Sie auf der Seite des Hosts C die erste Round Trip Time (RTT) mit Δt_1 und das Zeitintervall des ersten Sendefensters inklusive der Wartezeit auf Seiten des Hosts S mit Δt_2.

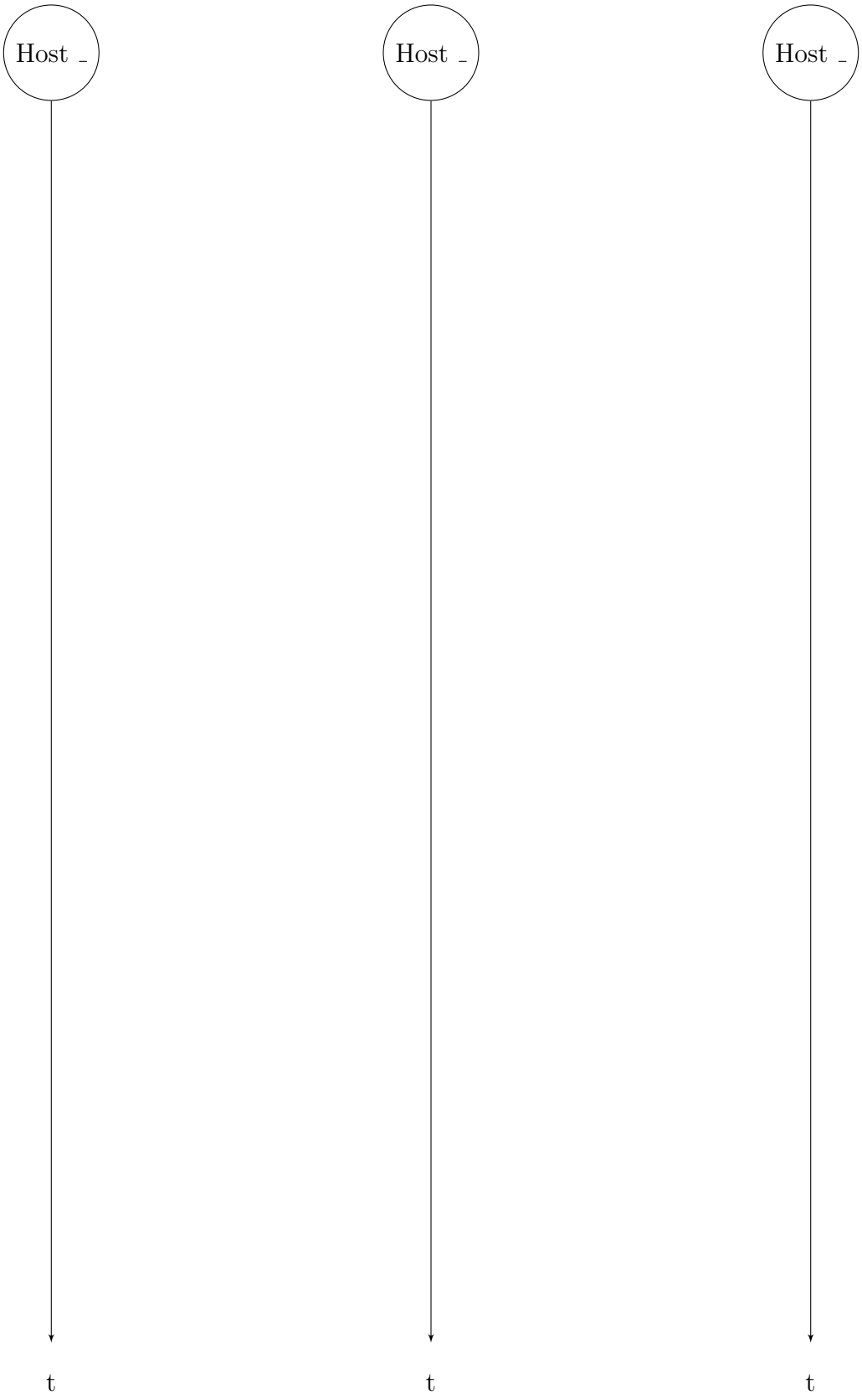

Abbildung 2: Ablauf Diagramm für eine TCP Verbindung

c) (5 Punkte) Geben Sie eine Formel an, mit der man die RTT für die in Abbildung 1 gezeigte Topologie berechnen kann. Wie groß ist die Verzögerung einer RTT für die in Tabelle 1 angegeben Werte. Verwenden Sie für Ihre Berechnungen eine idealisierte Paketgröße von 0 *byte*.

d) (5 Punkte) Nehmen Sie an, daß in der TCP-Verbindung nun Stop-and-Wait verwendet wird. Wie lange dauert es aus Sicht des Hosts S ohne Verbindungsaufbau und Anfrage ein Objekt der Größe $O = 5000$ *byte* erfolgreich (d.h. inklusive der letzten Bestätigung) zu übertragen? Verwenden sie für Ihre Berechnungen das Ergebnis aus Teilaufgabe c), ansonsten können sie für die RTT den Wert $d_{RTT} = 9$ ms annehmen. (Hinweis: Store-and-Forward)

e) (8 Punkte) Im Folgenden wird nun wieder von einer normalen TCP Verbindung (mit Fenstern) ausgegangen. Ändern Sie die in der Vorlesung hergeleitete Formel 2 für die Gesamtverzögerungszeit, die benötigt wird, um ein Objekt der Größe O zu übertragen, von dem Fall einer direkten Verbindung auf die in Abbildung 1 gezeigten Netzwerk-Topologie um. Gehen Sie davon aus, dass die Anzahl der Wartezeiten P bekannt ist. Geben Sie im Weiteren an, ob sich, für die Berechnung von P, K oder/und Q ändert. Begründen Sie für beide Variablen Ihre Antwort. Gehen Sie davon aus, das die RTT als Konstante bekannt ist.

Berechnen Sie, wie lange es dauert, bis ein Objekt der Größe $O = 3750\ bytes$ auf Host C verfügbar ist. Verwenden sie für Ihre Berechnungen das Ergebnis aus Teilaufgabe c), ansonsten können sie für die RTT den Wert $d_{RTT} = 9\ ms$ annehmen. (Hinweis: Die benötigte Zeit kann auch händisch ausgerechnet werden.)

$$d = 2RTT + \frac{O}{R} + P\left(RTT + \frac{L}{R}\right) - \left(2^P - 1\right)\frac{L}{R} \tag{2}$$

3 Socket-Programmierung (26 Punkte)

Ein Dienst wartet auf Datagramm-Pakete, die eine Temperatur enthalten. Empfängt der Dienst ein Paket, ruft er `handle(packet)` von `BehaviorImpl` auf und verhält er sich wie im Statechart (Abb. 3) dargestellt.

Aufgabe: Realisieren Sie dieses Verhalten in JAVA. Leiten Sie dazu die Klasse `BehaviorImpl` von der Klasse `Bahavior` ab und implementieren Sie die Methoden `handle(...)` und `encode(...)` unter Berücksichtigung folgender Aspekte:

- `byte[] encode(int value)` soll einen Integer in das `byte[]` der String-Darstellung umwandeln (vgl. Übung);
- „sende ... via ...“ im Statechart beinhaltet alle Vorbereitungen zum Versenden der Nachtricht, das Versenden der Nachricht, sowie das saubere Beenden des Sendevorgangs; Verwenden Sie hierfür die Konstanten `SERVER`, `PORT` und `ALARM` von Behavior;
- Verwenden Sie die Methoden `decode(...)` und `mean(...)` von `Behavior` wie im Statechart angegeben;
- Exceptions müssen berücksichtigt, aber nicht behandelt werden.

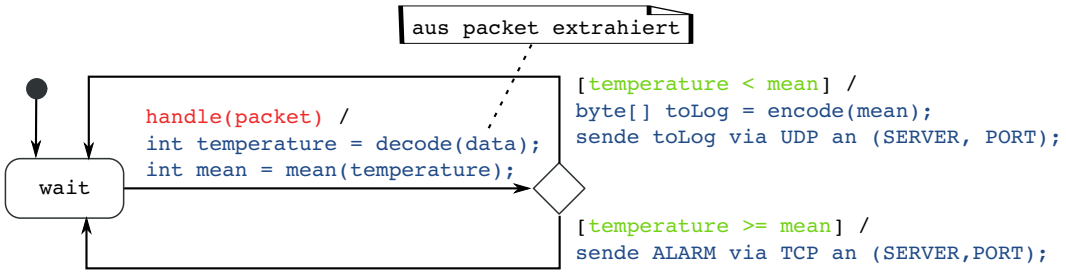

Abbildung 3: Verhalten des Servers.

```
import java.net.DatagramPacket;

public abstract class Behavior {
   final static String     SERVER = "192.3.45.2"; // Server-Adresse
   final static int        PORT   = 12345;        // Server-Port
   final static String     ALARM  = "Achtung";    // Alarmnachricht

   int decode(byte[] data) { xxx }  // Umwandeln byte[] --> int

   int mean(int value) { xxx }      // Berechnung eines Mittelwerts

   // Abarbeiten eines Datgramm-Pakets                  !!! TODO !!!
   abstract void handle(DatagramPacket packet) throws Exception;

   // Umwandeln int --> byte[]                          !!! TODO !!!
   abstract byte[] encode(int value) throws Exception;
}
```

4 Routingverfahren (16 Punkte)

Wie in Abbildung 4 dargestellt hat der Knoten A in einem Netzwerk die Nachbarn B und C und kennt die angegebenen Kosten für die jeweiligen direkten Verbindungen. Weiterhin gibt es die Knoten D und E im Netzwerk, die A nur über seine direkten Nachbar erreichen kann. Der Distanz-Vektor-Routing-Algorithmus hat bereits die Konvergenz erreicht.

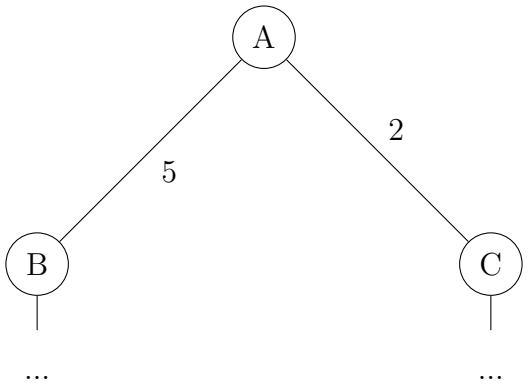

Abbildung 4: Netzwerk

a) (1 Punkt) Vervollständigen sie zunächst die Distanztabelle Tabelle 2, die der Knoten A vorhält. Die Notation ist wie in der Vorlesung gewählt. D.h. $nh_A(\cdot)$ bezeichnet den Vektor der aktuellen nächsten Knoten auf den kürzesten Wegen von A zu den anderen Knoten. $D_A(\cdot)$ beschreibt die dafür anfallenden Kosten.

von A zu	$D_A(\cdot)$	$nh_A(\cdot)$
A		
B		
C		
D	5	C
E	7	B

Tabelle 2: Distanztabelle von A

b) (5 Punkte) Im folgenden Schritt wird nun eine neue Verbindung eingeführt. Knoten A und E besitzen nun eine direkte Verbindung mit den Kosten 1. Tabelle 3 beschreibt den Distanzvektor, den Knoten E an Knoten A sendet. Tragen Sie die neue Distanztabelle von A in Tabelle 4 ein. Parallel dazu sendet auch Knoten A an Knoten E seine Informationen. Ändert sich für den Knoten E etwas in seinem Distanzvektor? Wenn ja, dann streichen sie die Kosten für den jeweiligen Knoten in Tabelle 3 und ersetzen ihn durch den neuen Wert.

von E zu	$D_E(\cdot)$
A	1
B	2
C	6
D	3
E	0

Tabelle 3: Distanzvektor von E

von A zu	$D_A(\cdot)$	$nh_A(\cdot)$
A		
B		
C		
D		
E		

Tabelle 4: Distanztabelle von A

c) (6 Punkte) Knoten A sendet im nächsten Schritt seine Änderung an seine direkten Nachbarkonten. Wie ändern sich die Distanztabellen, wenn sie vorher gemäß Tabelle 5 gefüllt waren und kein anderer Knoten neue Informationen verschickt. Tragen Sie die Änderungen in Tabelle 6 ein.

Ändert sich im darauf folgenden Schritt die Distanztabelle von A, wenn nur noch Knoten B und C aktualisierte Informationen aus dem Schritt zuvor im Netzwerk verteilen? Tragen Sie die Änderungen ggf. in Tabelle 7 ein oder streichen Sie sie durch.

von B zu	$D_B(\cdot)$	$nh_B(\cdot)$
A	5	A
B	0	-
C	4	D
D	1	D
E	2	E

von C zu	$D_C(\cdot)$	$nh_C(\cdot)$
A	2	A
B	4	D
C	0	-
D	3	D
E	6	D

Tabelle 5: Distanztabelle von B und C (vorher)

von B zu	$D_B(\cdot)$	$nh_B(\cdot)$
A		
B		
C		
D		
E		

von C zu	$D_C(\cdot)$	$nh_C(\cdot)$
A		
B		
C		
D		
E		

Tabelle 6: Distanztabelle von B und C (nachher)

von A zu	$D_A(\cdot)$	$nh_A(\cdot)$
A		
B		
C		
D		
E		

Tabelle 7: Distanztabelle von A

d) (4 Punkte) Vervollständigen Sie anschließend das gesamte Netzwerk anhand der in den Tabellen verfügbaren Informationen (inklusive der Verbindungskosten) in Abbildung 5. Für jede falsche Verbindung wird ein Punkt abgezogen.

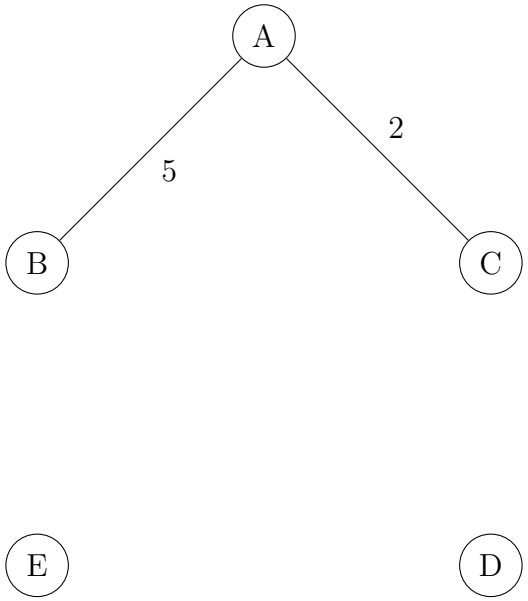

Abbildung 5: Gesamtes Netzwerk

Zusatzblatt

Zusatzblatt

ARBEITSKOPIE: Verzögerungszeiten

In folgenden wird von der Netzwerk-Topologie gemäß Abbildung 6 ausgegangen. Host C und Host S können nur indirekt über Host G miteinander kommunizieren. Mit Host G sind noch weitere Netzwerkteilnehmer verbunden, die ebenfalls Daten mit Host S austauschen wollen. Dadurch entsteht ein sogenannter "bottleneck" (Engpass) der dazu führt, dass alle Pakete, die über Host G zu Host S gesendet werden, in einer Warteschlange zwischengespeichert werden müssen. Alle anderen Verbindungen müssen nicht gepuffert werden. Pakete benötigen in den einzelnen Knoten keine Verarbeitungszeit. Gehen Sie weiterhin davon aus, dass alle Links symmetrisch sind und die Warteschlange nie überläuft. Ein Paketverlust tritt ebenfalls nicht auf.

Verwenden Sie für Ihre Berechnungen die Werte gemäß Tabelle 8. Für die Warteschlangenverzögerung können Sie von einer konstanten mittleren Verzögerung, die gemäß Formel 3 berechnet werden kann, ausgehen. Die Ausbreitungsverzögerung für den Link zwischen Host C und G ist so gering, dass sie vernachlässigt werden kann (idealisiert: $l_1 = 0\ m$).

$$d_{queue} = \frac{\rho L}{R(1-\rho)} \tag{3}$$

Ausbreitungsgeschwindigkeit:	$c = 2 \cdot 10^8\ \frac{m}{s}$
Leitungslängen:	$l_1 = 0\ m$ $l_2 = 200\ km$
Raten:	$R_1 = 100\ \frac{Mbit}{s}$ $R_2 = 1\ \frac{Mbit}{s}$
Paketlänge:	$L = 1250\ Byte$
Verkehrsintensität:	$\rho = \frac{1}{3}$

Tabelle 8: Konstanten

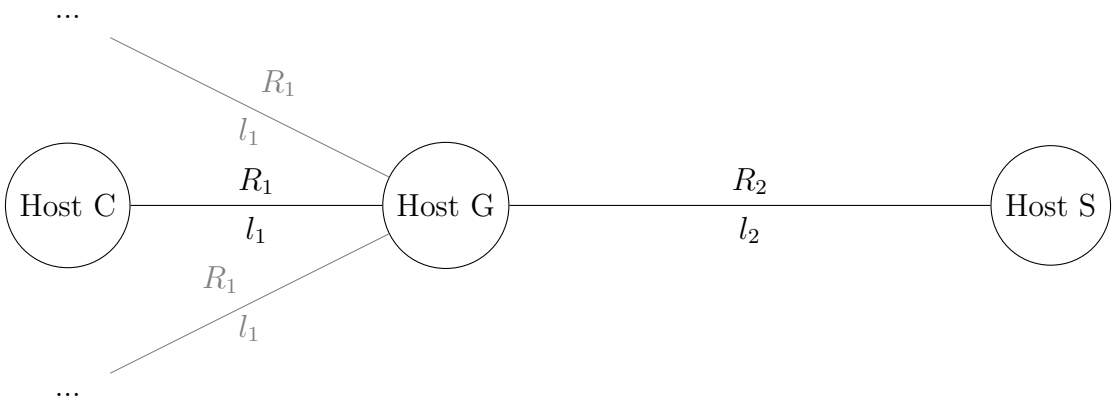

Abbildung 6: Netzwerk-Topologie

ARBEITSKOPIE: Socket-Programmierung

Ein Dienst wartet auf Datagramm-Pakete, die eine Temperatur enthalten. Empfängt der Dienst ein Paket, ruft er `handle(packet)` von `BehaviorImpl` auf und verhält er sich wie im Statechart (Abb. 7) dargestellt.

Aufgabe: Realisieren Sie dieses Verhalten in JAVA. Leiten Sie dazu die Klasse `BehaviorImpl` von der Klasse `Bahavior` ab und implementieren Sie die Methoden `handle(...)` und `encode(...)` unter Berücksichtigung folgender Aspekte:

- `byte[] encode(int value)` soll einen Integer in das `byte[]` der String-Darstellung umwandeln (vgl. Übung);
- „sende ... via ...“ im Statechart beinhaltet alle Vorbereitungen zum Versenden der Nachtricht, das Versenden der Nachricht, sowie das saubere Beenden des Sendevorgangs; Verwenden Sie hierfür die Konstanten `SERVER`, `PORT` und `ALARM` von Behavior;
- Verwenden Sie die Methoden `decode(...)` und `mean(...)` von `Behavior` wie im Statechart angegeben;
- Exceptions müssen berücksichtigt, aber nicht behandelt werden.

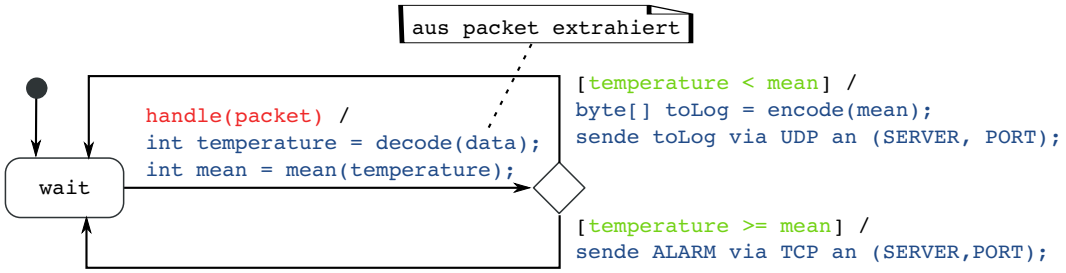

Abbildung 7: Verhalten des Servers.

```
import java.net.DatagramPacket;

public abstract class Behavior {
   final static String     SERVER = "192.3.45.2"; // Server-Adresse
   final static int        PORT   = 12345;        // Server-Port
   final static String     ALARM  = "Achtung";    // Alarmnachricht

   int decode(byte[] data) { xxx }  // Umwandeln byte[] --> int

   int mean(int value) { xxx }      // Berechnung eines Mittelwerts

   // Abarbeiten eines Datgramm-Pakets                  !!! TODO !!!
   abstract void handle(DatagramPacket packet) throws Exception;

   // Umwandeln int --> byte[]                          !!! TODO !!!
   abstract byte[] encode(int value) throws Exception;
}
```