Übungen zu Einführung in Rechnernetze
3. Übung

Tim Gerhard, Sebastian Friebe
[tim.gerhard, friebe]@kit.edu

Institut für Telematik, Prof. Zitterbart

Einführung in Rechnernetze – 3. Übungsblatt

1. TCP-Sockets
    1. ChatClient
    2. Socket-Wettbewerb
2. ARQ-Verfahren, Auslastung, Leistungsbewertung
3. Sequenznummern, Flusskontrolle
4. Transportschicht
5. TCP
6. Internet-Prüfsumme
7. Hamming Code

(f) – Chat

Übung 1

- Lesen von Nachrichten:

```
GET /chat HTTP/1.1
Host: i72tmdjango.tm.kit.edu
```

- Senden einer Nachricht:

```
POST /chat HTTP/1.1
Host: i72tmdjango.tm.kit.edu
Content-Type: application/x-www-form-urlencoded
Content-Length: 26

user=someone&message=Hallo
```

Unser Chat-Client: Usage

- CLI-Tool
- Zwei Funktionen: Lesen oder Senden

- Lesen des Chats:
  - `java ChatClient`

- Senden einer Nachricht
  - `java ChatClient “[USERNAME]” “[MESSAGE]”`

HTTP über TCP-Sockets: Vorgehen

- Schritte:
  - Öffnen der Verbindung
  - Verarbeiten der Eingabe (nur falls senden)
  - Senden der Anfrage (GET oder POST)
  - Empfang und Verarbeitung der Antwort
  - Schließen der Verbindung

- Einfacher Ablauf in main-Funktion:

```
 1 import java.io.InputStreamReader;
 2 import java.io.PrintWriter;
 3 import java.io.BufferedReader;
 4 import java.io.IOException;
 5
 6 import java.net.Socket;
 7 import java.net.URLEncoder;
 8
 9 public class ChatClient {
10   public static void main(String[] args) {
11     try {
12
13       // out actual code
14
15     } catch (IOException e){
16       System.out.println("ERROR: connection. closing");
17     }
18   }
19 }
```

HTTP über TCP-Sockets: Verbindungsaufbau

```
 9   public static void main(String[] args) {
10     try {
11
12       String host = "i72tmdjango.tm.kit.edu";
13       int port = 8000;
14       Socket httpcli = new Socket(host, port);
15
16       PrintWriter out = new PrintWriter(httpcli.getOutputStream(), true);
17
18       BufferedReader in = new BufferedReader(
19         new InputStreamReader(httpcli.getInputStream())
20       );
```

ChatClient: Verarbeiten der Eingabe

Spec: application/x-www-form-urlencoded

- Control names and values are escaped. Space characters are replaced by `+', and then reserved characters are escaped as described in [RFC1738], section 2.2: Non-alphanumeric characters are replaced by `%HH', a percent sign and two hexadecimal digits representing the ASCII code of the character. Line breaks are represented as "CR LF" pairs (i.e., `%0D%0A').

- The control names/values are listed in the order they appear in the document. The name is separated from the value by `=' and name/value pairs are separated from each other by `&'.

[https://www.w3.org/TR/html401/interact/forms.html#h-17.13.4.1]

ChatClient: Verarbeiten der Eingabe

```
23        String request;
24
25        if (args.length == 0) {
26
27          request = "GET WHAT?";
28
29        } else if (args.length == 2) {
30
31          String username = args[0];
32          String message = args[1];
33          String httpcontent = "user="+URLEncoder.encode(username, "UTF-8")
34                               + "&message="+URLEncoder.encode(message, "UTF-8");
35
36          request = "POST WHAT?";
37
38        } else {
39          //show usage on screen
40          System.exit(0);
41        }
```

ChatClient: Requests

- GET-Request zum lessen des Chats

```
request = "GET /chat HTTP/1.1\r\n"
        +"Host: "+host+"\r\n\r\n";
```

- POST-Request zum senden einer Chatnachricht

```
int contentLength = httpcontent.getBytes("UTF-8").length;
request = "POST /chat HTTP/1.1\r\n"
        +"Host: "+host+"\r\n"
        +"Content-Type: application/x-www-form-urlencoded\r\n"
        +"Content-Length: "+Integer.toString(contentLength)+"\r\n"
        +"\r\n" + httpcontent;
```

- *Hinweis:* In beiden Fällen ist die Reaktion des Servers gleich zu interpretieren: Nachrichten auf dem Bildschirm ausgeben.

ChatClient: Kommunikation mit dem Server

- Senden der Anfrage:

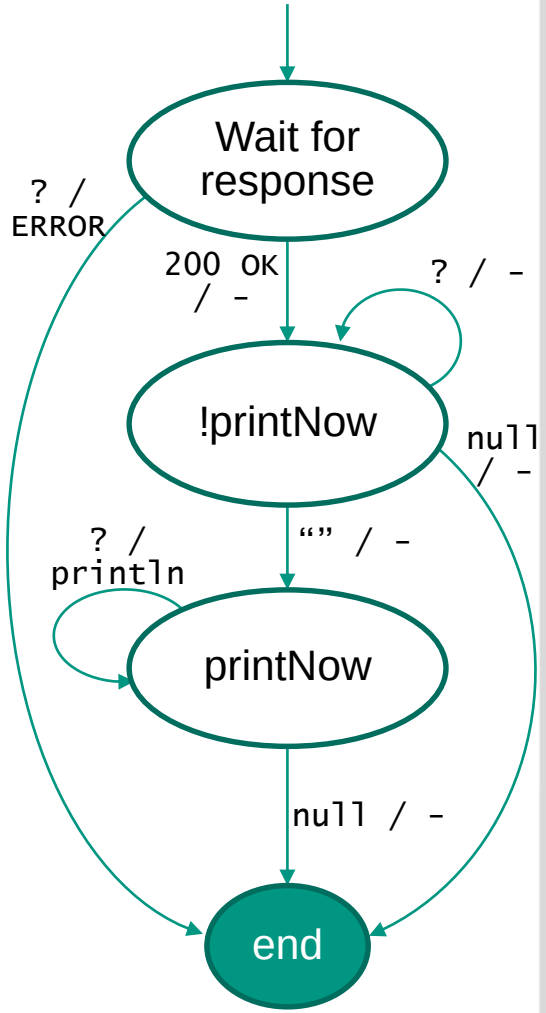
```
out.println(request);
```

- Verarbeiten der Antwort
  - Erste Zeile: HTTP-Statuscode prüfen

```
String ln = in.readLine();
if (! ln.endsWith("200 OK") ) {
  System.out.println("ERROR: Unexpected Response Header: "+ln);
  System.exit(1);
}
```

  - Danach: restlichen HTTP-Header überspringen dann alle Zeilen auf Bildschirm drucken.

```
boolean printNow = false;
while ((ln = in.readLine()) != null) {
  if (!printNow) {
    if (ln.length() == 0) {
      printNow = true;
    }
  } else {
    System.out.println(ln);
  }
}
```



ChatClient: Schließen der Verbindung

- Geschieht automatisch
  - Garbage Collection
  - JVM Shutdown

Gesamtes Programm (1/2)

```
1  import java.io.InputStreamReader;
2  import java.io.PrintWriter;
3  import java.io.BufferedReader;
4  import java.io.IOException;
5
6  import java.net.Socket;
7  import java.net.URLEncoder;
8
9  public class ChatClient {
10   public static void main(String[] args) {
11     try {
12
13       String host = "i72tmdjango.tm.kit.edu";
14       int port = 8000;
15       Socket httpcli = new Socket(host, port);
16
17       PrintWriter out = new PrintWriter(httpcli.getOutputStream(), true);
18
19       BufferedReader in = new BufferedReader(
20         new InputStreamReader(httpcli.getInputStream())
21       );
22
23       String request;
24
25       if (args.length == 0) {
26         request = "GET /chat HTTP/1.1\r\n"
27                  +"Host: "+host+"\r\n\r\n";
28       } else if (args.length == 2) {
29         String username = args[0];
30         String message = args[1];
31         String httpcontent = "user="+URLEncoder.encode(username, "UTF-8")
32                            + "&message="+URLEncoder.encode(message, "UTF-8");
33
34         int contentLength = httpcontent.getBytes("UTF-8").length;
35         request = "POST /chat HTTP/1.1\r\n"
36                  +"Host: "+host+"\r\n"
37                  +"Content-Type: application/x-www-form-urlencoded\r\n"
38                  +"Content-Length: "+Integer.toString(contentLength)+"\r\n"
39                  +"\r\n" + httpcontent;
```

Gesamtes Programm (2/2)

```
40        } else {
41          System.out.println("USAGE:");
42          System.out.println("to read from the chat:");
43          System.out.println("  java ChatClient");
44          System.out.println("to write to the chat:");
45          System.out.println("  java ChatClient \"[USERNAME]\" \"[MESSAGE]\"");
46          System.exit(0);
47          request = ""; //fix might-not-have-been-initialized
48        }
49
50        out.println(request);
51
52        String ln = in.readLine();
53        if (! ln.endsWith("200 OK") ) {
54          System.out.println("ERROR: Unexpected Response Header: "+ln);
55          System.exit(1);
56        }
57
58        // we only want to show the output after the double-linebreak
59        // (i.e., skip HTTP Header)
60        boolean printNow = false;
61        while ((ln = in.readLine()) != null) {
62
63          if (!printNow) {
64
65            if (ln.length() == 0) {
66              printNow = true;
67            }
68
69          } else {
70            System.out.println(ln);
71          }
72        }
73
74      } catch (IOException e){
75        System.out.println("ERROR: connection. closing");
76      }
77    }
78  }
```

Einführung in Rechnernetze – 3. Übungsblatt

1. TCP-Sockets
    1. ChatClient
    2. Socket-Wettbewerb
2. ARQ-Verfahren, Auslastung, Leistungsbewertung
3. Sequenznummern, Flusskontrolle
4. Transportschicht
5. TCP
6. Internet-Prüfsumme
7. Hamming Code

Beispiellösung: Permutations.java

- Berechnung aller Permutationen eines Arrays
- Implementierung von Heap's Algorithmus

```java
import java.util.Arrays;
import java.util.LinkedList;

public class Permutations<T> {
  protected LinkedList<T[]> result;

  public Permutations(T[] a) {
    this.result = new LinkedList<T[]>();
    this.generate(a.length, a);
  }

  private void generate(int n, T[] A) {
    if (n==1) {
      this.result.add(Arrays.copyOf(A, A.length));
    } else {
      for(int i=0; i<n-1; i++) {
        generate(n-1, A);
        if (n % 2 == 0) {
          T t = A[i];
          A[i] = A[n-1];
          A[n-1] = t;
        } else {
          T t = A[0];
          A[0] = A[n-1];
          A[n-1] = t;
        }
      }
      generate(n-1, A);
    }
  }

  public Iterable<T[]> r() {return this.result;}

}
```

Beispiellösung: NemesisDefeater.java (1/3)

- Import, Felder, Konstruktor, Verbindungsaufbau

```java
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.io.BufferedReader;
import java.io.IOException;

import java.net.Socket;

import java.util.LinkedList;
import java.util.List;

public class NemesisDefeater {

  protected String username;
  protected String server;
  protected int port;

  protected BufferedReader in;
  protected PrintWriter out;
  protected Socket sock;

  public NemesisDefeater(String username, String server, int port) {
    this.username = username;
    this.server = server;
    this.port = port;
  }

  public void connect() throws IOException {
    this.sock = new Socket(this.server, this.port);
    this.out = new PrintWriter(sock.getOutputStream(), true);
    this.in = new BufferedReader(new InputStreamReader(sock.getInputStream()));
  }
```

Beispiellösung: NemesisDefeater.java (2/3)

- Lösung einer Aufgabe
  - Iteration über alle Permutationen
  - Erstes Palindrom zurückgeben

- Protokoll-Handler
  - Bearbeitung
    - Eingehende Nachricht lesen
    - Fallunterscheidung
    - Antwort senden
  - Debug-Output:
    - Alle Nachrichten in Terminal ausgeben

```
33  private String solve_challenge(String w) {
34    for (String[] p: new Permutations<String>(w.split(" ")).r()) {
35      String possibleSolution = String.join("", p);
36      boolean isPalindrome = true;
37      int i1 = 0;
38      int i2 = possibleSolution.length() - 1;
39      while (i2>i1) {
40        if (possibleSolution.charAt(i1) != possibleSolution.charAt(i2)) {
41          isPalindrome = false;
42          break;
43        }
44        i2--;
45        i1++;
46      }
47      if(isPalindrome) {
48        return possibleSolution;
49      }
50    }
51    return null;
52  }
53  public String solve() throws IOException {
54    String in_ln;
55    while((in_ln = this.in.readLine()) != null) {
56      System.out.println("recv:  " + in_ln);
57
58      if (in_ln.startsWith("TMCTF{") && in_ln.endsWith("}")) {
59        return in_ln;
60      }
61
62      String answer;
63      switch(in_ln) {
64      case "Who are you?":
65        answer = this.username;
66        break;
67      case "You are out of time!":
68        System.out.println("Too slow.... sorry :(");
69        return null;
70      case "WRONG!":
71        System.out.println("Nope, can't do that!");
72        return null;
73      default:
74        answer = this.solve_challenge(in_ln.trim());
75      }
76      System.out.println("send:  " + answer);
77      this.out.println(answer);
78    }
79    return null;
80  }
```

Beispiellösung: NemesisDefeater.java (2/3)

- Main-Methode

```
82    public static void main(String[] args) {
83      NemesisDefeater def = new NemesisDefeater("myusername", "i72tmdjango.tm.kit.edu", 5005);
84      try {
85        def.connect();
86        def.solve();
87      catch (IOException e) {
88        e.printStackTrace();
89      }
90    }
91
92  }
```

Einführung in Rechnernetze – 3. Übungsblatt

1. TCP-Sockets
2. ARQ-Verfahren, Auslastung, Leistungsbewertung
3. Sequenznummern, Flusskontrolle
4. Transportschicht
5. TCP
6. Internet-Prüfsumme
7. Hamming Code

Aufgabe 2 (a)

- Wozu dienen ARQ-Verfahren?
  - ARQ = Automatic Repeat Request
  - Automatische Wiederholung von verlorengegangenen oder beschädigten Paketen

- Kann durch ein ARQ-Verfahren ein zuverlässiger Dienst bereitgestellt werden?

Aufgabe 2 (a) – Pingo

- Kann durch ein ARQ-Verfahren ein zuverlässiger Dienst bereitgestellt werden?
  - Ja!
  - Nein

Aufgabe 2 (a)

- Wozu dienen ARQ-Verfahren?
  - ARQ = Automatic Repeat Request
  - Automatische Wiederholung von verlorengegangenen oder beschädigten Paketen

- Kann durch ein ARQ-Verfahren ein zuverlässiger Dienst bereitgestellt werden?
  - Minimale Bestandteile eines ARQ-Verfahrens
    - Sendewiederholungen
    - Positive Quittungen
  - Bestätigter Dienst?
    - Ja!
  - Zuverlässiger Dienst?
    - Nein

Aufgabe 2 (a)

- Die fünf Eigenschaften eines zuverlässigen Dienstes (vgl. Vorlesung)

- 1. Der Empfänger hat die Daten in der richtigen Reihenfolge
  - Sequenznummern
- 2. Keine Duplikate
  - Sequenznummern
- 3. Keine Phantom- Pakete (aus früheren Verbindungen)
  - Aushandlung einer zufälligen Start-Sequenznummer bei Verbindungsaufbau
- 4. Alle **Daten** sind vollständig
  - Sequenznummern und Quittungen für ganze Pakete
- 5. Alle **Daten** sind korrekt
  - Durch ARQ-Verfahren **nicht** sichergestellt!
  - Die Erkennung von Bitfehlern benötigt weitere Mechanismen
  - Paritätsbits, Prüfsummen

Aufgabe 2 (b)

- Es soll eine Datenmenge von 2 MByte mit Hilfe des Stop-and-Wait-Verfahrens übertragen werden. Die maximale Größte eines Pakets betrage 750 Byte.

- Wie lange dauert die gesamte Datenübertragung, von Beginn bis zum Zeitpunkt an dem der Sender sicher davon ausgehen kann, dass die Übertragung korrekt abgeschlossen ist?

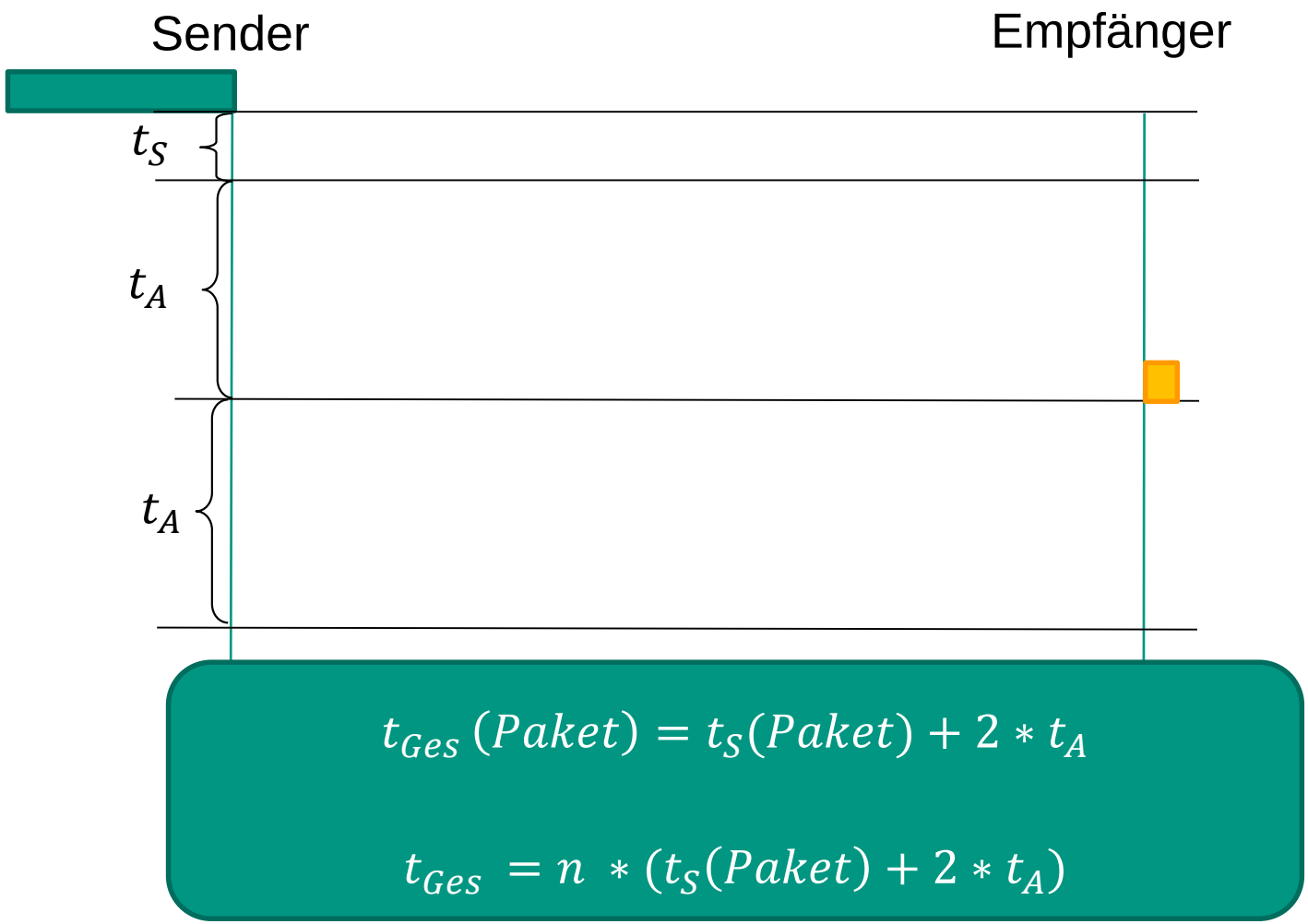
Aufgabe 2 (b) – Pingo

- Wie lange dauert die gesamte Datenübertragung, von Beginn bis zum Zeitpunkt an dem der Sender sicher davon ausgehen kann, dass die Übertragung korrekt abgeschlossen ist?

  - Datenrate: 6 Mbit/s
  - Ausbreitungsverzögerung: 20 Millisekunden
  - ARQ-Verfahren: Stop-and-Wait
  - Datenmenge: 2 MByte
  - Größe eines Pakets: 750 Byte

- $50{,}002\ s$
- $50{,}00\overline{6}\ s$
- $50{,}019\ s$
- $109{,}306\ s$
- $109{,}34\overline{6}\ s$
- $109{,}347\ s$

Ablauf und Zeitdauern Stop-and-Wait



Aufgabe 2 (b)

- Datenrate: 6 Mbit/s
- Ausbreitungsverzögerung t_A: 20 Millisekunden
- ARQ-Verfahren: Stop-and-Wait
- Datenmenge: 2 MByte
- Größe einer Paket: 750 Byte

- Berechnung:
  - $n = \frac{2\ MByte}{750\ Byte} = 2666,\overline{6}$
  - $t_{S,750\ Byte} = \frac{6000\ Bit}{6\ \frac{Mbit}{s}} = 0{,}001s$
  - $t_{S,500\ Byte} = \frac{4000\ Bit}{6\ \frac{Mbit}{s}} = 0{,}000\overline{6}s$
  - $t_A = 0{,}02s$
  - $t_{ges} = 2666 * \left(t_{S,750\ Byte} + 2 * t_A\right) + t_{S,500\ Byte} + 2 * t_A = \mathbf{109,34\overline{6}\ s}$

$$t_S = \frac{L}{d}$$

Datenmenge L

Datenrate d des Mediums

Aufgabe 2 (c)

- Welche Auslastung des Mediums wird in Teilaufgabe (b) erreicht?

Aufgabe 2 (c) – Pingo

- Welche Auslastung des Mediums wird in Teilaufgabe (b) erreicht?

  - Datenrate: 6 Mbit/s
  - Ausbreitungsverzögerung: 20 Millisekunden
  - ARQ-Verfahren: Stop-and-Wait
  - Datenmenge: 2 MByte
  - Größe eines Pakets: 750 Byte
  - $t_{ges} \approx 109,347\ s$

- 1,4%
- 1,8%
- 2,4%
- 3,6%

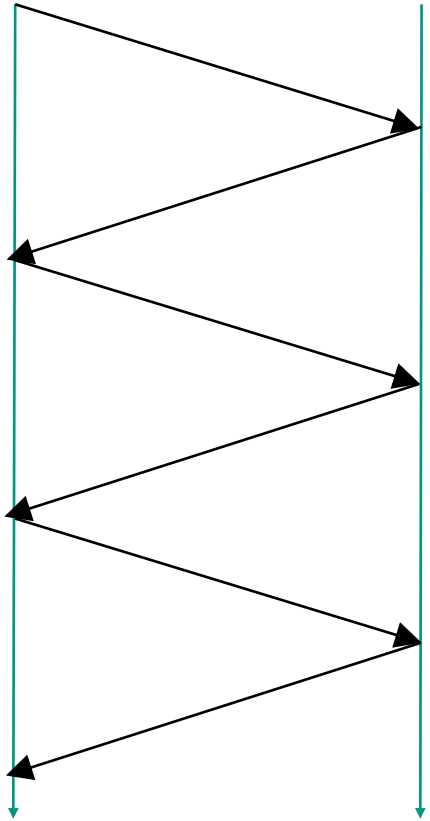
Aufgabe 2 (c)

- Lösungsweg 1: Analytisch
  - Auslastung $U =$ „Verhältnis von tatsächlicher zu möglicher Nutzung“
  - Betrachteter Zeitraum in Aufgabe (b): $109{,}347\ \text{s}$ ← gerundet
  - Tatsächliche Übertragung: $2\ \text{MByte}$
  - Mögliche Übertragung: $6\frac{\text{Mbit}}{\text{s}} * 109{,}347\ \text{s} \approx 656\ \text{Mbit} = 82\ \text{MByte}$
  - $U = \frac{2\ \text{MByte}}{82\ \text{MByte}} = 0{,}0244 \approx 2{,}4\%$
- Lösungsweg 2: via Formel aus der Vorlesung
  - Auslastung bei Stop-and-Wait $U = \frac{t_S}{t_S + 2 * t_A}$
  - $t_S(Paket) = 0{,}001s$
  - $T_A = 0{,}02s$
  - $U = \frac{T_S}{T_S + 2 * TA} = \frac{0{,}001\text{s}}{0{,}001\text{s} + 2 * 0{,}02\text{s}} = \frac{0{,}001\text{s}}{0{,}41\text{s}} = 0{,}0244 \approx 2{,}4\%$

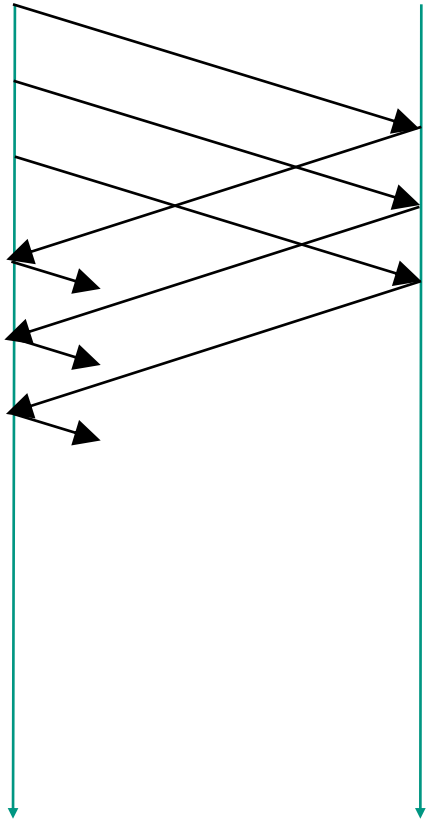
Aufgabe 2 (d)

- Auf welchem Wege wird beim Go-back-N-Verfahren versucht, die Übertragung zu beschleunigen?
- Der Sender kann mehrere Pakete senden, bevor er auf zurückkommende Quittungen warten muss.

Stop-and-Wait:



Go-back-N



Aufgabe 2 (e)

- Berechnen Sie die Auslastung, die im gegebenen Beispiel aus Teilaufgabe (b) unter Verwendung des Go-back-N-Verfahrens erreicht wird. Gehen sie von einer statischen Fenstergröße von 3.000 Byte und alternativ einer Größe von 120.000 Byte aus.
- Erläutern Sie den Einfluss der Fenstergröße.

Aufgabe 2 (e) – Pingo

- Berechnen Sie die Auslastung, die im gegebenen Beispiel aus Teilaufgabe (b) unter Verwendung des Go-back-N-Verfahrens erreicht wird, bei einer statischen Fenstergröße von 3.000 Byte.

- Datenrate: 6 Mbit/s
- Ausbreitungsverzögerung: 20 Millisekunden
- Datenmenge: 2 MByte
- Größe eines Pakets: 750 Byte

- 3,85%
- 9,76%
- 10,21%
- 19,04%

Aufgabe 2 (e)

- Einfluss der Fenstergröße W : Kann der Sender durchgehend senden oder muss er auf Quittungen warten?
- Parameter a: Verhältnis aus Übertragungsdauer und Sendedauer
  - Siehe nächste Folie
- Auslastung U abhängig von Fenstergröße W und Parameter a
- Auslastung U im fehlerfreien Fall:

$$U = \begin{cases} 1 & W \geq 1 + 2a \\ \dfrac{W}{1 + 2a} & W < 1 + 2a \end{cases}$$

mit $a = \dfrac{t_A}{t_S}$

Parameter a

$$U = \begin{cases} 1 & W \geq 1 + 2a \\ \dfrac{W}{1 + 2a} & W < 1 + 2a \end{cases}$$

mit $a = \dfrac{t_A}{t_S}$

- Für volle Auslastung ($U = 1$):
- Je größer Parameter a ist, desto größer muss Fenstergröße W sein, damit ununterbrochen gesendet werden kann

- „Großes a“ ?
  - t_A ist hoch ➔ Die Quittungen treffen langsamer ein
  - t_S ist niedrig ➔ Der Sender legt die Pakete schneller aufs Medium, füllt sein Sendefenster schneller

- Je ungünstiger dieses Verhältnis, desto größer muss Fenstergröße W zum Ausgleich sein.

- Sonst: Wartezeit des Senders auf Quittungen ➔ $U < 1$

Aufgabe 2 (e)

- Einfluss der Fenstergröße W : Kann der Sender durchgehend senden oder muss er auf Quittungen warten?
- Auslastung U abhängig von Fenstergröße W und Parameter a
- Auslastung U im fehlerfreien Fall:

$$U = \begin{cases} 1 & W \geq 1 + 2a \\ \dfrac{W}{1 + 2a} & W < 1 + 2a \end{cases}$$

mit $a = \frac{t_A}{t_S}$

- Kochrezept zur Berechnung von U (Go-back-N):
  - 1.) Fenstergröße W (in **Anzahl der Pakete**) bestimmen
  - 2.) Parameter a bestimmen (abhängig von $t_S(Paket)$ und t_A)
  - 3.) Prüfen: $W \geq 1 + 2a$?
    - Ja: U = 1, fertig
    - Nein: W und a In Formel einsetzen

Aufgabe 2 (e)

- Fall 1: Fenstergröße ist $3.000\ \text{Byte}$

- 1.) Fenstergröße W (in Anzahl der Pakete) bestimmen
  - Größe eines Pakets: $750\ \text{Byte}$
  - $W = \frac{3.000\ \text{Byte}}{750\ \frac{\text{Byte}}{\text{Paket}}} = 4\ \text{Pakete}$

- 2.) Parameter a bestimmen
  - Bedeutet: t_S eines Pakets verwenden!
  - $a = \frac{t_A}{t_S} = \frac{0{,}02\ s}{0{,}001\ s} = 20$

- 3.) Prüfen: $W \geq 1 + 2a$?
  - $4 \geq 41$? ➔ Nein, in Formel einsetzen
  - $U = \frac{W}{1+2a} = \frac{4}{1+2*20} = \frac{4}{41} \approx 0{,}0976 = 9{,}76\%$

Aufgabe 2 (e)

- Fall 2: Fenstergröße ist $120.000\ \text{Byte}$

- 1.) Fenstergröße W (in Anzahl der Pakete) bestimmen
  - Größe eines Pakets: $750\ \text{Byte}$
  - $W = \frac{120.000\ \text{Byte}}{750\ \frac{\text{Byte}}{\text{Paket}}} = 160\ \text{Pakete}$

- 2.) Parameter a bestimmen
  - Da die Größe eines Pakets und damit t_S wie in Fall 1 ist, gilt auch hier $a = 20$

- 3.) Prüfen: $W \geq 1 + 2a$?
  - $160 \geq 41$? ➔ Ja ➔ $U = 1 = 100\%$
  - Quittungen treffen schnell genug ein, bevor das Sendefenster voll ist
  - Sender kann ununterbrochen senden

Aufgabe 2 (f)

- Nehmen Sie an, dass ein Paket nicht mehr 750 Byte groß ist, sondern dass es möglich ist, die komplette Fenstergröße von 120.000 Byte aus Teilaufgabe (e) an einem Stück, also als ein einziges Paket zu senden.
- Berechnen Sie die Auswirkungen auf die Auslastung und erklären Sie das Ergebnis.

Aufgabe 2 (f)

- 1.) Fenstergröße W (in Anzahl der Pakete) bestimmen
  - Größe eines Pakets: ~~750~~ 120.000 Byte
  - $W = \frac{120.000\ \text{Byte}}{120.000\ \frac{\text{Byte}}{\text{Paket}}} = 1\ \text{Paket}$
- 2.) Parameter a für ein Paket bestimmen
  - t_S und a müssen neu berechnet werden
  - $t_S = \frac{120.000\ \text{Byte}}{6\ \frac{\text{Mbit}}{\text{s}}} = \frac{960.000\ \text{Bit}}{6\ \frac{\text{Mbit}}{\text{s}}} = \frac{9{,}6 * 10^5\ \text{Bit}}{6 * 10^6\ \frac{\text{Bit}}{\text{s}}} = 0{,}16\text{s}$
  - $a = \frac{t_A}{t_S} = \frac{0{,}02\ s}{0{,}16\ s} = 0{,}125$
- 3.) Prüfen: $W \geq 1 + 2a$?
  - $1 \geq 1{,}25$? ➔ Nein, in Formel einsetzen
  - $U = \frac{W}{1+2a} = \frac{1}{1+2*0{,}125} = \frac{1}{1{,}25} = 0{,}8 = 80\%$

Aufgabe 2 (f)

- Fazit: Wir haben keine volle Auslastung mehr (80% statt 100%)
- Durch $W = 1$ de facto wieder Stop-and-Wait
- Dennoch gute Auslastung durch sehr kleines a (lange Sendezeit, kurzes Warten auf Quittungen)

- Stop-and-Wait kann keine volle Auslastung haben… warum?

- Begründung 1 (anschaulich):
  - Der Sender muss immer auf Quittungen warten und kann während dieser Zeit nicht senden.
- Begründung 2 (gemäß Formeln):
  - Es gilt: $U = \frac{W}{1+2a}$
  - Wenn $U = 1$ gelten soll und $W = 1$ ist, muss also $a = 0$ sein
  - Ferner gilt: $a = \frac{t_A}{t_s}$
  - Wenn $a = 0$ gelten soll, muss also $t_A = 0$ sein (physikalisch unmöglich)

Aufgabe 2 (g)

- Welche Auslastung erreicht das Szenario mit Stop-and-Wait aus Teilaufgabe (b), falls bei der Übertragung eine Verlustwahrscheinlichkeit von 10% besteht?
- Wie sieht die Situation bei 30% aus?

Aufgabe 2 (g) – Pingo

- Welche Auslastung erreicht das Szenario mit Stop-and-Wait aus Teilaufgabe (b), falls bei der Übertragung eine Verlustwahrscheinlichkeit von 10% besteht?

- Datenrate: 6 Mbit/s
- Ausbreitungsverzögerung: 20 Millisekunden
- Datenmenge: 2 MByte
- Größe eines Pakets: 750 Byte

- 0,9%
- 1,4%
- 1,7%
- 2,2%

Aufgabe 2 (g)

- Auslastung U bei Stop-and-Wait mit Fehlerwahrscheinlichkeit p

$$U = \frac{1 - p}{1 + 2a}$$

- Verlustwahrscheinlichkeit 10% ➔ $p = 0{,}1$
- $U = \frac{1 - 0{,}1}{1 + 2t_A/t_S} = \frac{0{,}9}{1 + 2 * 0{,}02s/0{,}001s} = \frac{0{,}9}{1 + 2 * 20} = 0{,}02195 \approx 2{,}2\%$

- Verlustwahrscheinlichkeit 30% ➔ $p = 0{,}3$
- $U = \frac{1 - 0{,}3}{1 + 2t_A/t_S} = \frac{0{,}7}{1 + 2 * 0{,}02s/0{,}001s} = \frac{0{,}7}{1 + 2 * 20} = 0{,}01707 \approx 1{,}7\%$

- Zum Vergleich – Ergebnis aus Teilaufgabe (b) ohne Fehler: $U \approx 2{,}4\%$

Aufgabe 2 (h)

- Beschreiben Sie kurz, wodurch sich das Verfahren Selective Reject von Go-Back-N unterscheidet.

Aufgabe 2 (h)

- Selective Reject
  - Selective Repeat-Variante, die negative Quittungen verwendet
  - Vorteile von Selective Repeat
    - Der Empfänger kann im Fehlerfall nachfolgende, korrekt empfangene Pakete puffern und quittieren
    - Der Sender muss nur die fehlerhaften Pakete wiederholen
  - Vorteile von negative Quittungen
    - Der Sender muss nicht erst in einen Timeout für eine Sendewiederholung laufen…
    - …sondern kann explizit vom Empfänger auf den Fehler hingewiesen werden

- Go-Back-N
  - Im Fehlerfall verwirft der Empfänger alle nachfolgenden Pakete
  - Der Sender muss alles ab dem fehlerhaften Paket erneut senden
  - Auch hier Variante mit negativen Quittungen denkbar

Einführung in Rechnernetze – 3. Übungsblatt

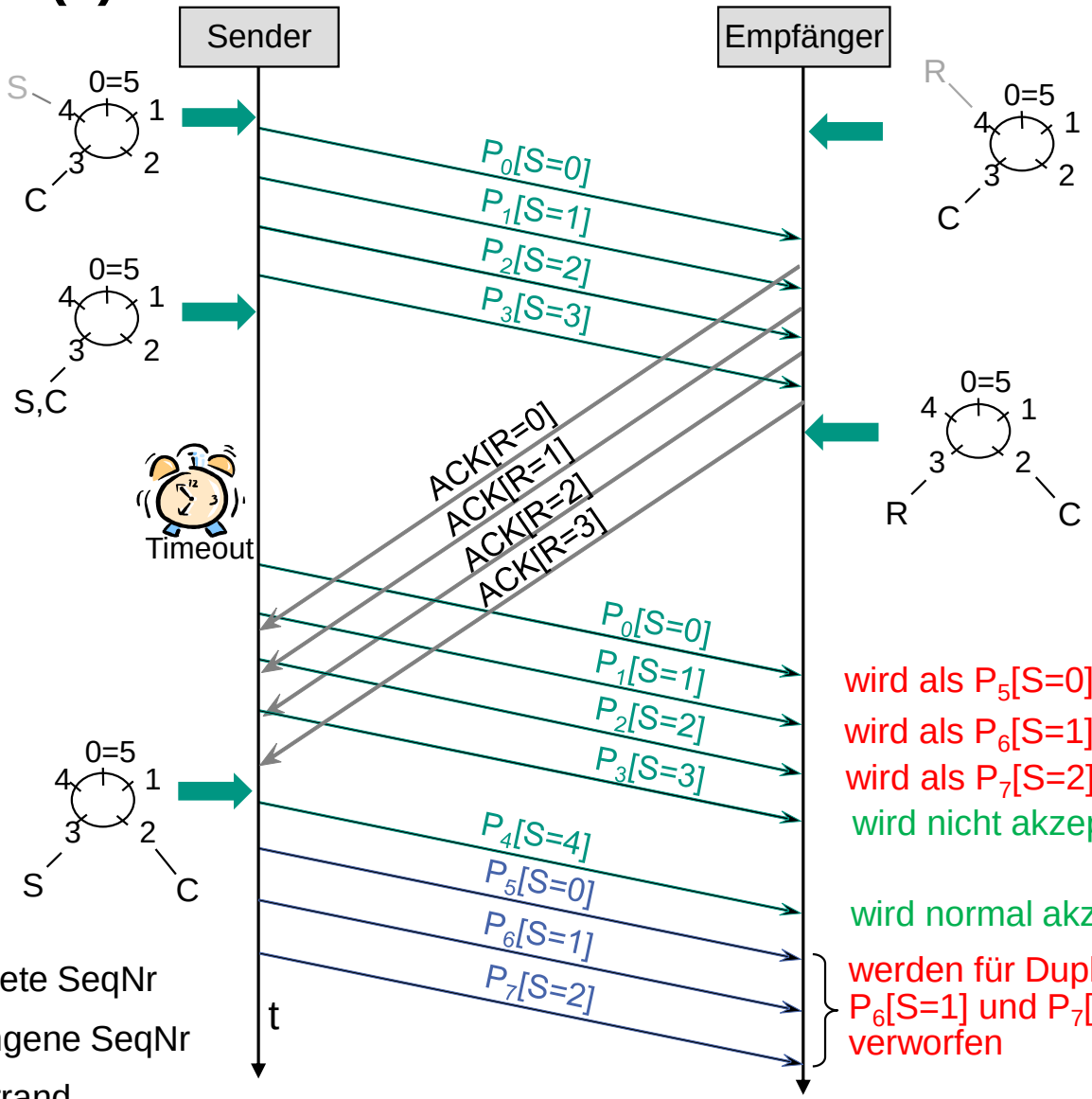
1. TCP-Sockets
2. ARQ-Verfahren, Auslastung, Leistungsbewertung
3. Sequenznummern, Flusskontrolle
4. Transportschicht
5. TCP
6. Internet-Prüfsumme
7. Hamming Code

Aufgabe 3 (a)

- Sie möchten den Sliding-Windows-Algorithmus (mit Selective-Repeat-Verfahren) mit Sende/-empfangsfenstergröße 4 und maximal 5 Sequenznummern ausführen.
- Das N-te Paket $P_N[S=i]$ enthält folglich i = (N mod 5) in seinem Sequenznummernfeld.
- Führen Sie ein Beispiel auf, bei dem der Algorithmus ein „unerwünschtes“ Ergebnis liefert, d.h. ein Szenario, bei dem der Empfänger das fünfte Paket ($P_5[S=0]$) erwartet, aber das nullte Paket ($P_0[S=0]$) entgegen nimmt.
- Zeichen Sie hierfür ein Weg-Zeit-Diagramm.

- Hinweis: S = Sende-Sequenznummer, wie in der Vorlesung. Es sollen keine Reihenfolgevertauschungen austreten, sondern ein ungünstiges Zusammenspiel zwischen Quittung und Timeout-Mechanismus.

Aufgabe 3 (a)



wird als $P_5[S=0]$ angenommen

wird als $P_6[S=1]$ angenommen

wird als $P_7[S=2]$ angenommen

wird nicht akzeptiert

wird normal akzeptiert

werden für Duplikate zu $P_5[S=0]$, $P_6[S=1]$ und $P_7[S=2]$ gehalten und verworfen

S: Zuletzt gesendete SeqNr

R: Zuletzt empfangene SeqNr

C: oberer Fensterrand

Aufgabe 3 (b)

- Wie viele unterschiedliche Sequenznummern sind bei Selective Repeat abhängig von der Sende- und Empfangsfenstergröße W mindestens nötig, damit das in Teilaufgabe (a) beschriebene Problem nicht mehr auftreten kann?

Aufgabe 3 (b) – Pingo

- Wie viele unterschiedliche Sequenznummern sind bei Selective Repeat abhängig von der Sende- und Empfangsfenstergröße W mindestens nötig, damit das in Teilaufgabe (a) beschriebene Problem nicht mehr auftreten kann?

- W + 2
- $\frac{3}{2}$ W
- 2W
- 3W+2

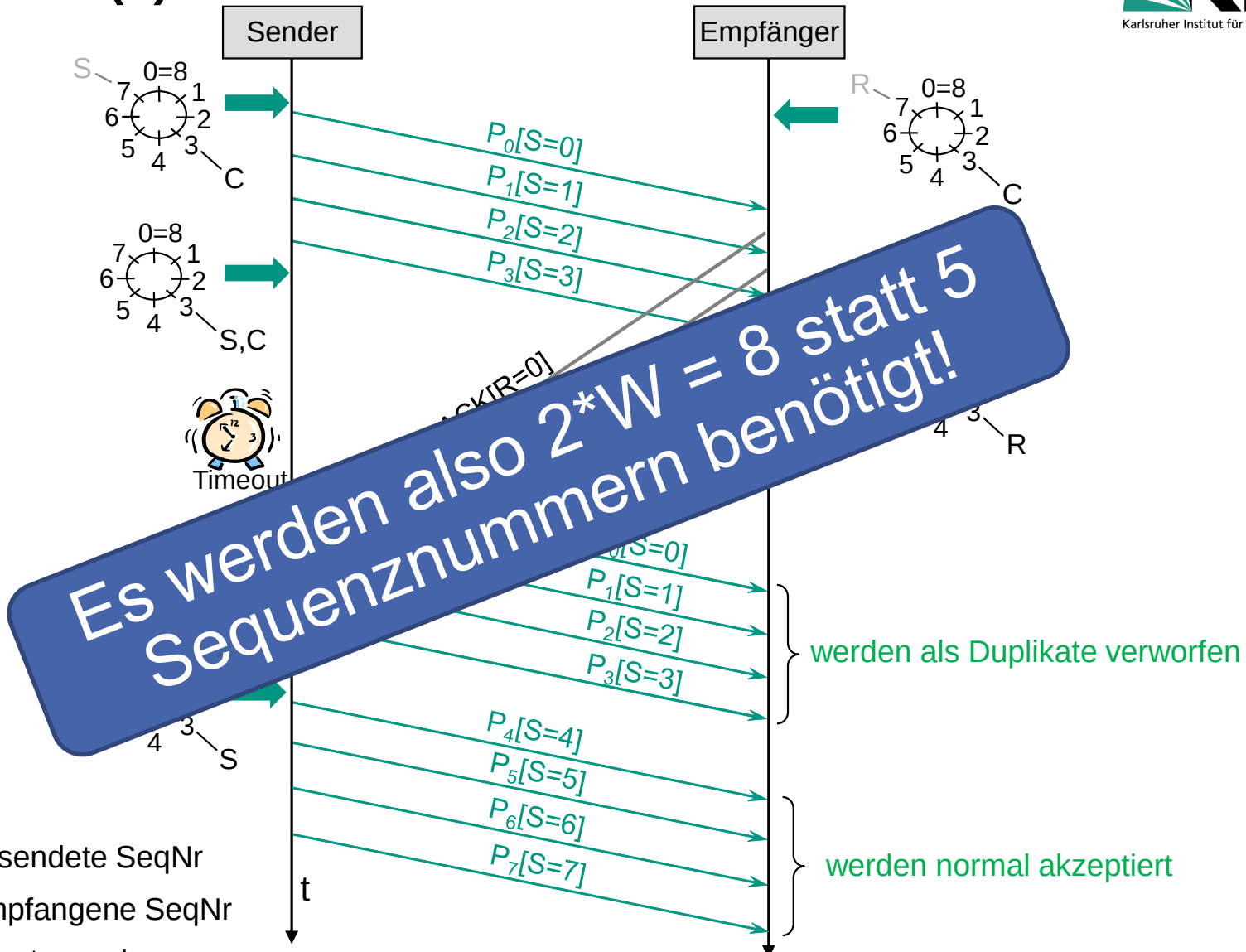
Aufgabe 3 (b)

- Der Empfänger muss neue, erwartete Pakete von Duplikaten bereits empfangener Pakete unterscheiden können

- Abhängig von Fenstergröße W können also beim Empfänger eintreffen
  - a) durch Timeouts Sendewiederholungen der alten Pakete 0..W-1
  - b) nach vollständiger Quittierung neue Pakete W..2W-1

- Der Sequenznummernraum muss also mindestens **2W** groß sein, damit der Empfänger die Pakete 0..2W-1 unterscheiden kann

Aufgabe 3 (b)



S: Zuletzt gesendete SeqNr

R: Zuletzt empfangene SeqNr

C: oberer Fensterrand

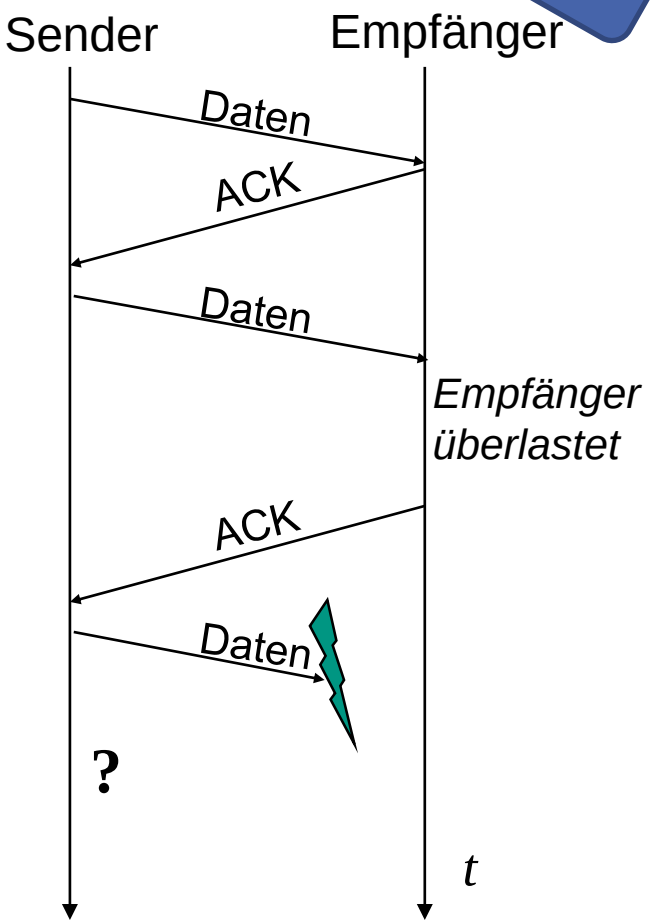
Aufgabe 3 (c)

- Bei der Verwendung des Sliding Windows ist es möglich, dies gleichzeitig für die Ausführung der Flusskontrolle einzusetzen.
- Wie kann dies erreicht werden?
- Welchen grundlegenden Nachteil hätte ein solches Vorgehen?

Aufgabe 3 (c)

Vorlesung

- Implizite Flusskontrolle
- Funktionsweise
  - Durch Zurückhalten der Quittung (z.B. ACK/NACK) kann der Sender gebremst werden
  - Das bedeutet, dass ein Verfahren zur Fehlererkennung für die Flusskontrolle mitbenutzt wird
  - Beispiel: Stop-and-Wait
- Problem?
  - Der Sender kann nicht mehr unterscheiden, ob
    - sein Paket verloren ging
    - der Empfänger die Quittung wegen Überlast zurückgehalten hat



Einführung in Rechnernetze – 3. Übungsblatt

1. TCP-Sockets
2. ARQ-Verfahren, Auslastung, Leistungsbewertung
3. Sequenznummern, Flusskontrolle
4. Transportschicht
5. TCP
6. Internet-Prüfsumme
7. Hamming Code

Aufgabe 4 (a)

- Die Website des Instituts für Telematik wird von einem Webserver mit der IP-Adresse 141.3.70.4 bereitgestellt, der Anfragen auf TCP-Port 80 erwartet.
- Eine Informatik-Studentin ruft an ihrem Rechner mit der IP-Adresse 217.237.151.142 die Website in ihrem Browser auf, welcher mehrere HTTP-Requests (für Text und Bilder) zu dem Webserver sendet.
- Da ihre Verbindung sehr langsam ist, nutzt sie die Zeit um sich den Vorgang auf Transportschicht vorzustellen.

- Können Sie gemeinsam mit ihr die folgenden Fragen klären?

Aufgabe 4 (a)

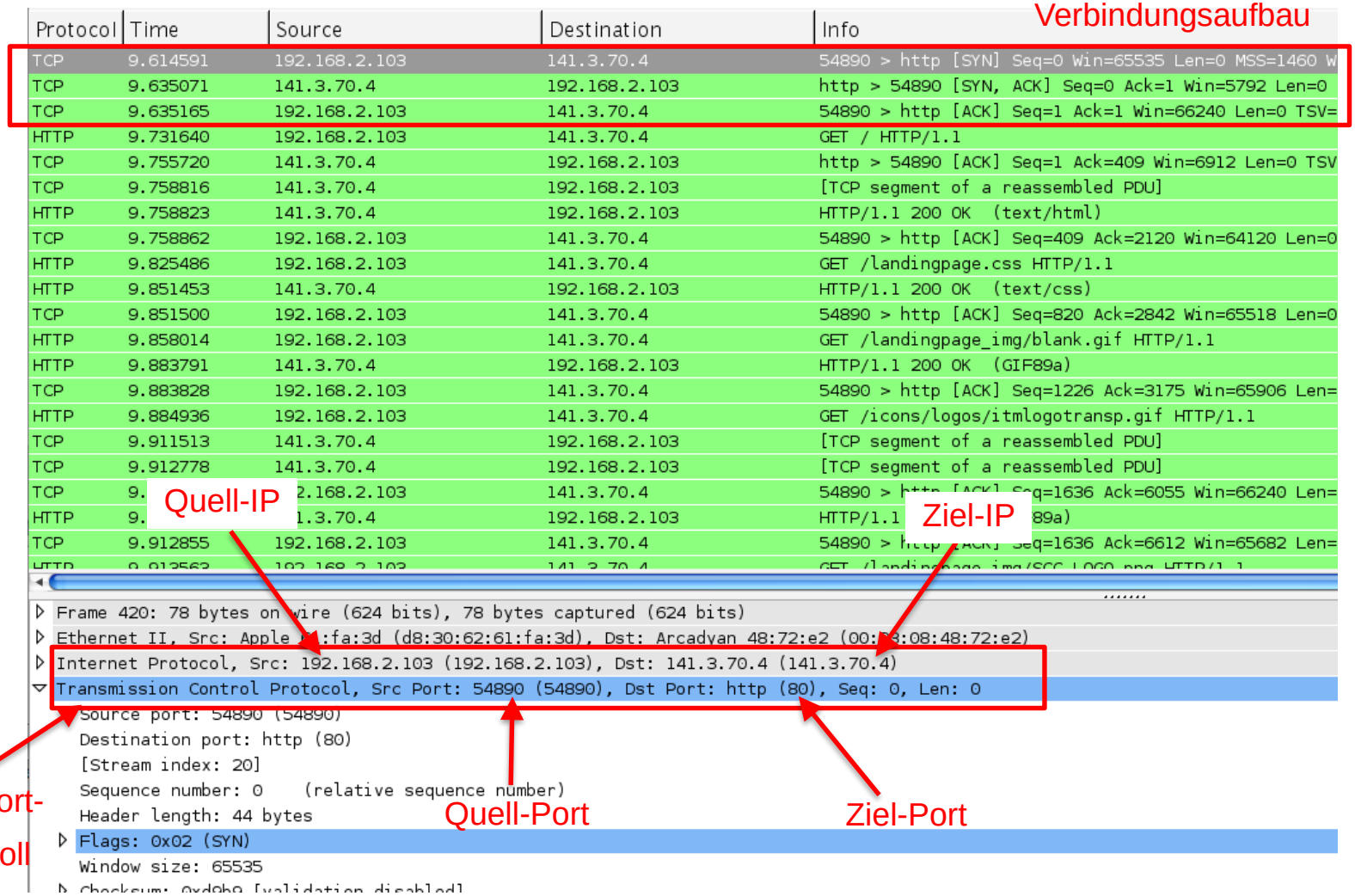
- Durch welches 5-Tupel kann ein HTTP-Request der Studentin eindeutig beschrieben werden? Gibt es mehrere richtige Antworten? Warum (nicht)?

Aufgabe 4 (a)

- 5-Tupel zur eindeutigen Identifizierung einer Verbindung
  - Quell-IP: 217.237.151.142
  - Quell-Portnummer: **???**
  - Ziel-IP: 141.3.70.4
  - Ziel-Portnummer: 80
  - Transportprotokoll: TCP

- Wo kommt die Quell-Portnummer her?
  - Wird für jeden Verbindungsaufbau dynamisch vom Betriebssystem gewählt
  - Client stellt die Verbindung her und nennt dem Server den gewählten Port
  - Client-Port muss also nicht vorab bekannt sein
  - Server-Port muss bekannt sein: HTTP standardmäßig auf Port 80

Aufgabe 4 (a)



Protocol	Time	Source	Destination	Info
TCP	9.614591	192.168.2.103	141.3.70.4	54890 > http [SYN] Seq=0 Win=65535 Len=0 MSS=1460 W
TCP	9.635071	141.3.70.4	192.168.2.103	http > 54890 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0
TCP	9.635165	192.168.2.103	141.3.70.4	54890 > http [ACK] Seq=1 Ack=1 Win=66240 Len=0 TSV=
HTTP	9.731640	192.168.2.103	141.3.70.4	GET / HTTP/1.1
TCP	9.755720	141.3.70.4	192.168.2.103	http > 54890 [ACK] Seq=1 Ack=409 Win=6912 Len=0 TSV
TCP	9.758816	141.3.70.4	192.168.2.103	[TCP segment of a reassembled PDU]
HTTP	9.758823	141.3.70.4	192.168.2.103	HTTP/1.1 200 OK (text/html)
TCP	9.758862	192.168.2.103	141.3.70.4	54890 > http [ACK] Seq=409 Ack=2120 Win=64120 Len=0
HTTP	9.825486	192.168.2.103	141.3.70.4	GET /landingpage.css HTTP/1.1
HTTP	9.851453	141.3.70.4	192.168.2.103	HTTP/1.1 200 OK (text/css)
TCP	9.851500	192.168.2.103	141.3.70.4	54890 > http [ACK] Seq=820 Ack=2842 Win=65518 Len=0
HTTP	9.858014	192.168.2.103	141.3.70.4	GET /landingpage_img/blank.gif HTTP/1.1
HTTP	9.883791	141.3.70.4	192.168.2.103	HTTP/1.1 200 OK (GIF89a)
TCP	9.883828	192.168.2.103	141.3.70.4	54890 > http [ACK] Seq=1226 Ack=3175 Win=65906 Len=
HTTP	9.884936	192.168.2.103	141.3.70.4	GET /icons/logos/itmlogotransp.gif HTTP/1.1
TCP	9.911513	141.3.70.4	192.168.2.103	[TCP segment of a reassembled PDU]
TCP	9.912778	141.3.70.4	192.168.2.103	[TCP segment of a reassembled PDU]
TCP	9.	2.168.2.103	141.3.70.4	54890 > http [ACK] Seq=1636 Ack=6055 Win=66240 Len=
HTTP	9.	1.3.70.4	192.168.2.103	HTTP/1.1 89a)
TCP	9.912855	192.168.2.103	141.3.70.4	54890 > http [ACK] Seq=1636 Ack=6612 Win=65682 Len=

```
Frame 420: 78 bytes on wire (624 bits), 78 bytes captured (624 bits)
Ethernet II, Src: Apple 61:fa:3d (d8:30:62:61:fa:3d), Dst: Arcadyan 48:72:e2 (00:23:08:48:72:e2)
Internet Protocol, Src: 192.168.2.103 (192.168.2.103), Dst: 141.3.70.4 (141.3.70.4)
Transmission Control Protocol, Src Port: 54890 (54890), Dst Port: http (80), Seq: 0, Len: 0
    Source port: 54890 (54890)
    Destination port: http (80)
    [Stream index: 20]
    Sequence number: 0    (relative sequence number)
    Header length: 44 bytes
  Flags: 0x02 (SYN)
    Window size: 65535
```

Aufgabe 4 (b)

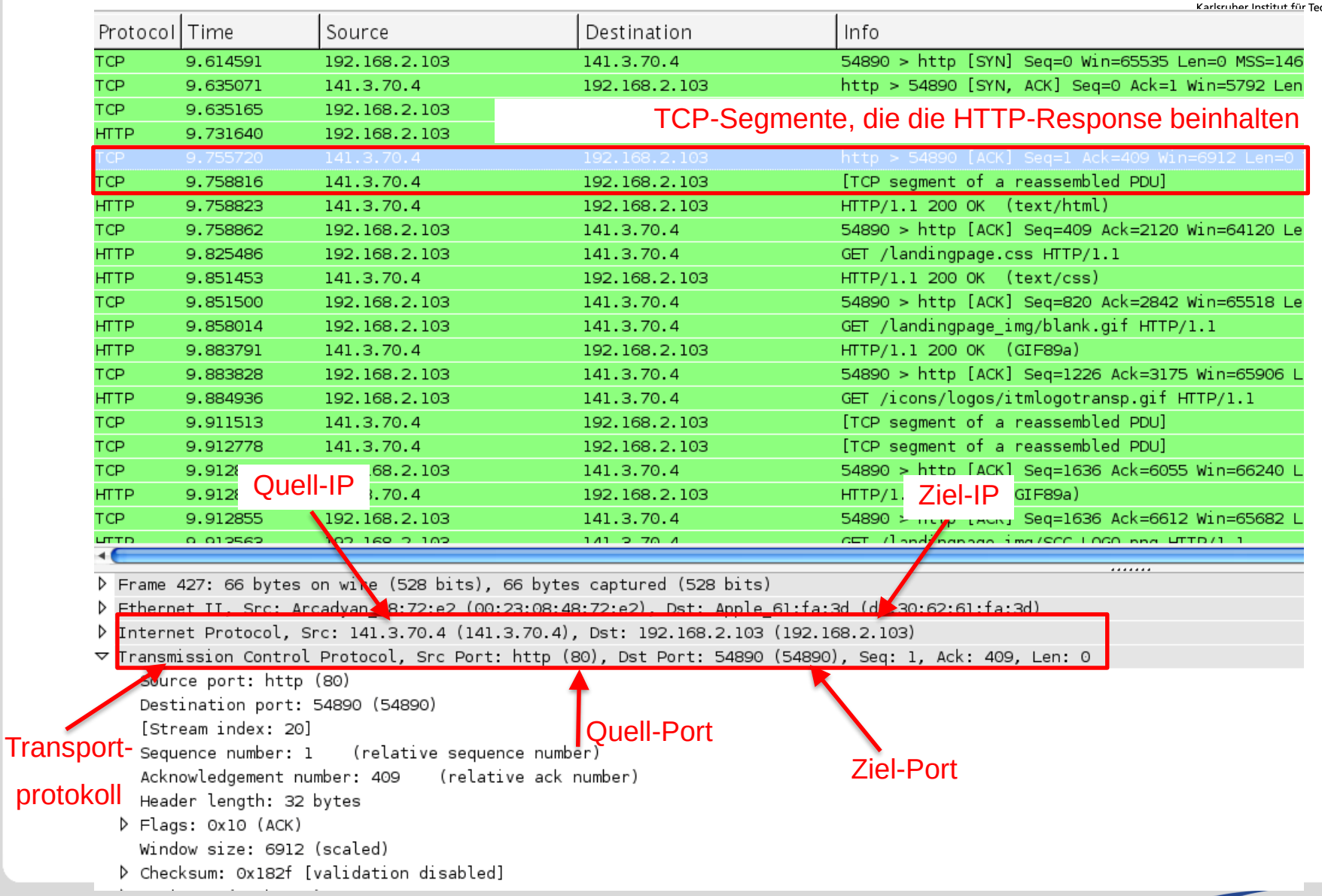
- Wie sieht dementsprechend das 5-Tupel der zugehörigen HTTP-Response aus?

- HTTP-Response: Antwort des Servers auf die zuvor eingegangene Anfrage

- Verwendet die zuvor hergestellte Verbindung

		Response
Quell-IP:	217.237.151.142	141.3.70.4
Quell-Portnummer:	54890	80
Ziel-IP:	141.3.70.4	217.237.151.142
Ziel-Portnummer	80	54890
Transportprotokoll:	TCP	TCP

Aufgabe 4 (b)


Protocol Time Source Destination Info
TCP 9.614591 192.168.2.103 141.3.70.4 54890 > http [SYN] Seq=0 Win=65535 Len=0 MSS=146
TCP 9.635071 141.3.70.4 192.168.2.103 http > 54890 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len
TCP 9.635165 192.168.2.103
HTTP 9.731640 192.168.2.103
TCP-Segmente, die die HTTP-Response beinhalten
TCP 9.755720 141.3.70.4 192.168.2.103 http > 54890 [ACK] Seq=1 Ack=409 Win=6912 Len=0
TCP 9.758816 141.3.70.4 192.168.2.103 [TCP segment of a reassembled PDU]
HTTP 9.758823 141.3.70.4 192.168.2.103 HTTP/1.1 200 OK (text/html)
TCP 9.758862 192.168.2.103 141.3.70.4 54890 > http [ACK] Seq=409 Ack=2120 Win=64120 Le
HTTP 9.825486 192.168.2.103 141.3.70.4 GET /landingpage.css HTTP/1.1
HTTP 9.851453 141.3.70.4 192.168.2.103 HTTP/1.1 200 OK (text/css)
TCP 9.851500 192.168.2.103 141.3.70.4 54890 > http [ACK] Seq=820 Ack=2842 Win=65518 Le
HTTP 9.858014 192.168.2.103 141.3.70.4 GET /landingpage_img/blank.gif HTTP/1.1
HTTP 9.883791 141.3.70.4 192.168.2.103 HTTP/1.1 200 OK (GIF89a)
TCP 9.883828 192.168.2.103 141.3.70.4 54890 > http [ACK] Seq=1226 Ack=3175 Win=65906 L
HTTP 9.884936 192.168.2.103 141.3.70.4 GET /icons/logos/itmlogotransp.gif HTTP/1.1
TCP 9.911513 141.3.70.4 192.168.2.103 [TCP segment of a reassembled PDU]
TCP 9.912778 141.3.70.4 192.168.2.103 [TCP segment of a reassembled PDU]
TCP 9.912 .68.2.103 141.3.70.4 54890 > http [ACK] Seq=1636 Ack=6055 Win=66240 L
HTTP 9.912 .70.4 192.168.2.103 HTTP/1. GIF89a)
TCP 9.912855 192.168.2.103 141.3.70.4 54890 > Seq=1636 Ack=6612 Win=65682 L
Quell-IP
Ziel-IP
Frame 427: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
Ethernet II, Src: Arcadyan 8:72:e2 (00:23:08:48:72:e2), Dst: Apple_61:fa:3d (d 30:62:61:fa:3d)
Internet Protocol, Src: 141.3.70.4 (141.3.70.4), Dst: 192.168.2.103 (192.168.2.103)
Transmission Control Protocol, Src Port: http (80), Dst Port: 54890 (54890), Seq: 1, Ack: 409, Len: 0
Source port: http (80)
Destination port: 54890 (54890)
[Stream index: 20]
Sequence number: 1 (relative sequence number)
Acknowledgement number: 409 (relative ack number)
Header length: 32 bytes
Flags: 0x10 (ACK)
Window size: 6912 (scaled)
Checksum: 0x182f [validation disabled]
Transport-protokoll
Quell-Port
Ziel-Port

Aufgabe 4 (c)

- Der Studentin fällt auf, dass sich in ihrem Browser mehrere Bilder parallel aufbauen. Wie ist das möglich? Was bedeutet dies für die geöffneten Sockets?

- Bilder werden parallel übertragen
- Mehrere Verbindungen sind gleichzeitig geöffnet

- Hergestellte Verbindungen (aus Sicht des Clients)
  - Quell-IP: 217.237.151.142
  - Quell-Portnummer: 54890, 54891, 54892, ...
  - Ziel-IP: 141.3.70.4
  - Ziel-Portnummer: 80
  - Transportprotokoll: TCP

Aufgabe 4 (c)

Filter: tcp || http && ip.dst==141.3.70.4 Expression... Clear Apply

Protocol	Time	Source	Destination	Info
HTTP	9.858014	192.168.2.103	141.3.70.4	GET /landingpage_img/blank.gif HTTP/1.1
TCP	9.858178	192.168.2.103	141.3.70.4	54893 > http [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=1 TSV=1076650884 TS
HTTP	9.858242	192.168.2.103	141.3.70.4	GET /landingpage_img/de.gif HTTP/1.1
TCP	9.858469	192.168.2.103	141.3.70.4	54894 > http [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=1 TSV=1076650884 TS
TCP	9.858640	192.168.2.103	141.3.70.4	54895 > http [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=1 TSV=1076650884 TS
TCP	9.878189	192.168.2.103	141.3.70.4	54892 > http [ACK] Seq=1 Ack=1 Win=66240 Len=0 TSV=1076650884 TSER=3996
HTTP	9.878408	192.168.2.103	141.3.70.4	GET /landingpage_img/en.gif HTTP/1.1
TCP	9.883828	192.168.2.103	141.3.70.4	54890 > http [ACK] Seq=1226 Ack=3175 Win=65906 Len=0 TSV=1076650885 TSER
TCP	9.884581	192.168.2.103	141.3.70.4	54893 > http [ACK] Seq=1 Ack=1 Win=66240 Len=0 TSV=1076650885 TSER=3996
HTTP	9.884936	192.168.2.103	141.3.70.4	GET /icons/logos/itmlogotransp.gif HTTP/1.1
HTTP	9.885088	192.168.2.103	141.3.70.4	GET /icons/logos/cm72-transparent.gif HTTP/1.1
TCP	9.889738	192.168.2.103	141.3.70.4	54894 > http [ACK] Seq=1 Ack=1 Win=66240 Len=0 TSV=1076650885 TSER=3996
TCP	9.889782	192.168.2.103	141.3.70.4	54891 > http [ACK] Seq=404 Ack=454 Win=65786 Len=0 TSV=1076650885 TSER=3
HTTP	9.890068	192.168.2.103	141.3.70.4	GET /landingpage_img/modul-uebersicht-gfx/modulUebersicht.png HTTP/1.1
HTTP	9.890793	192.168.2.103	141.3.70.4	GET /icons/logos/mwrg.png HTTP/1.1
TCP	9.892277	192.168.2.103	141.3.70.4	54895 > http [ACK] Seq=1 Ack=1 Win=66240 Len=0 TSV=1076650885 TSER=3996
HTTP	9.892393	192.168.2.103	141.3.70.4	GET /icons/logos/tecologo.gif HTTP/1.1
TCP	9.905727	192.168.2.103	141.3.70.4	54892 > http [ACK] Seq=404 Ack=497 Win=65744 Len=0 TSV=1076650885 TSER=3
HTTP	9.906684	192.168.2.103	141.3.70.4	GET /icons/logos/itm72.gif HTTP/1.1
TCP	9.912802	192.168.2.103	141.3.70.4	54890 > http [ACK] Seq=1636 Ack=6055 Win=66240 Len=0 TSV=1076650885 TSER

```
▷ Frame 461: 503 bytes on wire (4024 bits), 503 bytes captured (4024 bits)
▷ Ethernet II, Src: Apple_61:fa:3d (d8:30:62:61:fa:3d), Dst: Arcadyan_48:72:e2 (00:23:08:48:72:e2)
▷ Internet Protocol, Src: 192.168.2.103 (192.168.2.103), Dst: 141.3.70.4 (141.3.70.4)
▽ Transmission Control Protocol, Src Port: 54894 (54894), Dst Port: http (80), Seq: 1, Ack: 1, Len: 437
    Source port: 54894 (54894)
    Destination port: http (80)
    [Stream index: 24]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 438    (relative sequence number)]
    Acknowledgement number: 1    (relative ack number)
    Header length: 32 bytes
  ▷ Flags: 0x18 (PSH, ACK)
    Window size: 66240 (scaled)
```

Einführung in Rechnernetze – 3. Übungsblatt

1. TCP-Sockets
2. ARQ-Verfahren, Auslastung, Leistungsbewertung
3. Sequenznummern, Flusskontrolle
4. Transportschicht
5. TCP
6. Internet-Prüfsumme
7. Hamming Code

Aufgabe 5 – Pingo

Angenommen, es besteht eine TCP-Verbindung, über die Host A eine große Datei an Host B sendet. Welche dieser Aussagen sind wahr?

a) Host B hat keine Daten um sie an Host A zu schicken. Host B wird deshalb keine ACKs zu Host A senden, da er keine Pakete hat, in denen er die ACKs senden kann.

b) Die Größe des *RcvWindow* ändert sich während der Verbindung nicht.

c) Die Anzahl der von Host A gesendeten unbestätigten Bytes kann nicht größer sein als die Größe des Empfangspuffers.

d) Wenn die Sequenznummer für ein Segment dieser Verbindung *m* ist, dann ist sie für das nächste Paket zwangsweise *m + 1*.

e) TCP-Segmente haben im TCP-Kopf ein Feld *RcvWindow.*

f) Host A sendet Host B ein Segment mit der Sequenznummer 38, welches 4 Bytes Daten enthält. Die Quittungsnummer in diesem Segment ist zwangsweise 42.

Aufgabe 5

Angenommen, es besteht eine TCP-Verbindung, über die Host A eine große Datei an Host B sendet. Welche dieser Aussagen sind wahr?

a) Host B hat keine Daten um sie an Host A zu schicken. Host B wird deshalb keine ACKs zu Host A senden, da er keine Pakete hat, in denen er die ACKs senden kann.
Falsch, es können auch Pakete ohne Daten versendet werden.

b) Die Größe des *RcvWindow* ändert sich während der Verbindung nicht.
Falsch, das *RcvWindow* gibt jeweils den freien Pufferplatz an.

c) Die Anzahl der von Host A gesendeten unbestätigten Bytes kann nicht größer sein als die Größe des Empfangspuffers.
Wahr, Host B bestimmt, wie viel Host A senden darf

Aufgabe 5

d) Wenn die Sequenznummer für ein Segment dieser Verbindung *m* ist, dann ist sie für das nächste Paket zwangsweise *m + 1*.
Falsch, die Sequenznummern beziehen sich auf Bytes, nicht auf Pakete.

e) TCP-Segmente haben im TCP-Kopf ein Feld *RcvWindow.*
Wahr, ist nötig um die Größe des freien Puffers mitzuteilen.

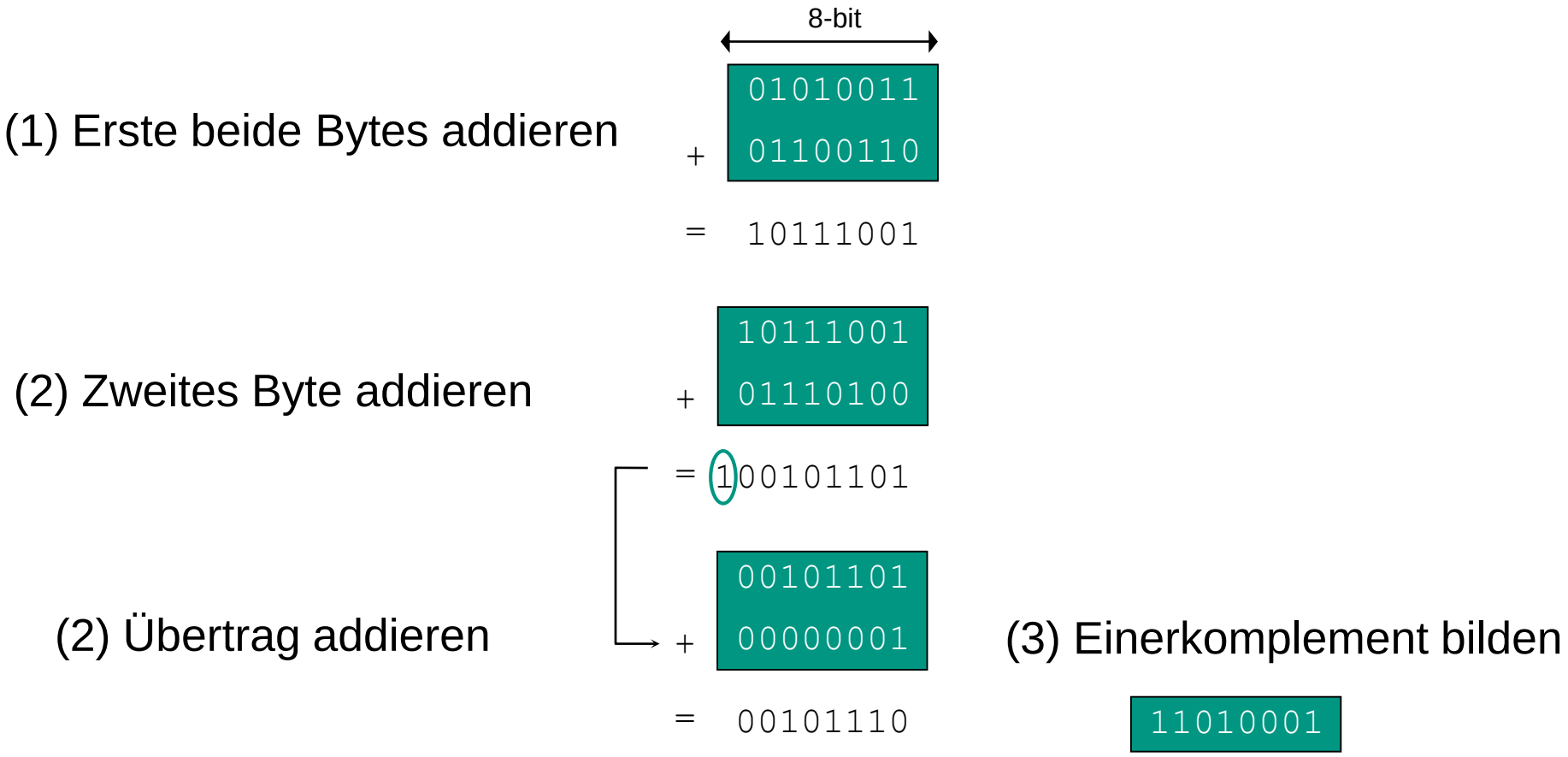
f) Host A sendet Host B ein Segment mit der Sequenznummer 38, welches 4 Bytes Daten enthält. Die Quittungsnummer in diesem Segment ist zwangsweise 42.
Falsch, die verwendeten Sequenznummern der Endsysteme haben keinen Zusammenhang.

Einführung in Rechnernetze – 3. Übungsblatt

1. TCP-Sockets
2. ARQ-Verfahren, Auslastung, Leistungsbewertung
3. Sequenznummern, Flusskontrolle
4. Transportschicht
5. TCP
6. Internet-Prüfsumme
7. Hamming Code

Aufgabe 6 (a)

- TCP und UDP verwenden die Internetprüfsumme um die Daten zu sichern. Berechnen Sie die Prüfsumme der Bytes *01010011*, *01100110* und *01110100*.



Aufgabe 6 (b)

- Warum wird das Einerkomplement der Summe verwendet und nicht die Summe selbst? Wie erkennt der Empfänger mithilfe des Einerkomplements Fehler?
  - Nach korrekter Berechnung der Prüfsumme sollte sich ein Byte mit nur Einsen ergeben. Dies lässt sich im Computer effizient testen
  - Auch ermöglicht das Einerkomplement effizientere Berechnungen der Summe, z.B. lassen sich die Daten in Blöcke aufteilen und parallel aufsummieren

RFC 1071

Aufgabe 6 (c)

- Kann ein 1-Bit Fehler unbemerkt bleiben? Was ist mit einem 2-Bit Fehler?
  - 1-Bit Fehler werden immer bemerkt
  - 2-Bit Fehler können unbemerkt bleiben, wenn sie sich im Laufe der Addition heraus rechnen

Aufgabe 6 (d)

- Angenommen ein Host empfängt ein Segment mit der Prüfsumme *01011101 11110010*. Der Host addiert die 16 Bit Wörter über alle notwendigen Felder *außer der Prüfsumme* und erhält den Wert *00110010 00001101*. Wird dieses Segment als korrekt oder fehlerhaft betrachtet? Was wird der Empfänger tun?

(1) Berechnete Prüfsumme mit empfangener Prüfsumme addieren:

```
  01011101 11110010
+ 00110010 00001101
= 10001111 11111111
```

(2) Einerkomplement bilden:

```
10001111 11111111  →  01110000 00000000
```

- Da das Einerkomplement nicht nur aus Einsen besteht, wurde das Segment fehlerhaft übertragen. Die Reaktion des Empfängers hängt vom eingesetzten ARQ-Verfahren ab.

Einführung in Rechnernetze – 3. Übungsblatt

1. TCP-Sockets
2. ARQ-Verfahren, Auslastung, Leistungsbewertung
3. Sequenznummern, Flusskontrolle
4. Transportschicht
5. TCP
6. Internet-Prüfsumme
7. Hamming Code

Aufgabe 7 (a)

- Aus der Vorlesung ist der (7,4) Hamming Code bekannt. Im Rahmen dieser Aufgabe soll nun ein Hamming Code entwickelt werden, der 11 Bit an Nutzdaten prüfen kann.
- Wie viele Paritätsbits werden für den Code benötigt?

  - 11 Bit sollen geprüft werden
  - Da $2^3 < 11 <= 2^4$ müssen mindestens 4 Paritätsbits verwendet werden, um alle Datenbits abzudecken
  - Da $11 + 4 <= 2^4$ genügen auch 4 Bits
  - Damit ergibt sich ein (15,11) Hamming Code

Aufgabe 7 (b)

- Erstellen Sie eine Hamming-Tabelle wie aus der Vorlesung bekannt. Markieren Sie, welche Zellen *nicht* zum Prüfen von empfangenen Daten verwenden werden.

		p_4	p_3	p_2	p_1
Pos. in z	Code	2^3	2^2	2^1	2^0
3					
5					
6					
7					
9					
10					
11					
12					
13					
14					
15					

Aufgabe 7 (c)

- Geben sie die Gleichungen der Paritätsbits an.

		p_4	p_3	p_2	p_1
Pos. in z	Code	2^3	2^2	2^1	2^0
3					
5					
6					
7					
9					
10					
11					
12					
13					
14					
15					

- $p_1 = x_1 + x_3 + x_5 + x_7 + x_9 + x_{11} + x_{13} + x_{15}$
- $p_2 = x_2 + x_3 + x_6 + x_7 + x_{10} + x_{11} + x_{14} + x_{15}$
- $p_3 = x_4 + x_5 + x_6 + x_7 + x_{12} + x_{13} + x_{14} + x_{15}$
- $p_4 = x_8 + x_9 + x_{10} + x_{11} + x_{12} + x_{13} + x_{14} + x_{15}$

Aufgabe 7 (b)

Gegeben sei das Codewort 100110001101011.

- Wurde dieses Codewort korrekt empfangen?
- Geben Sie die korrekten Nutzdaten an.

Paritätsbits

100110001101011

Nutzdaten

Somit befindet sich das fehlerhafte Bit an Position $0001_2 = 1$ im Codewort

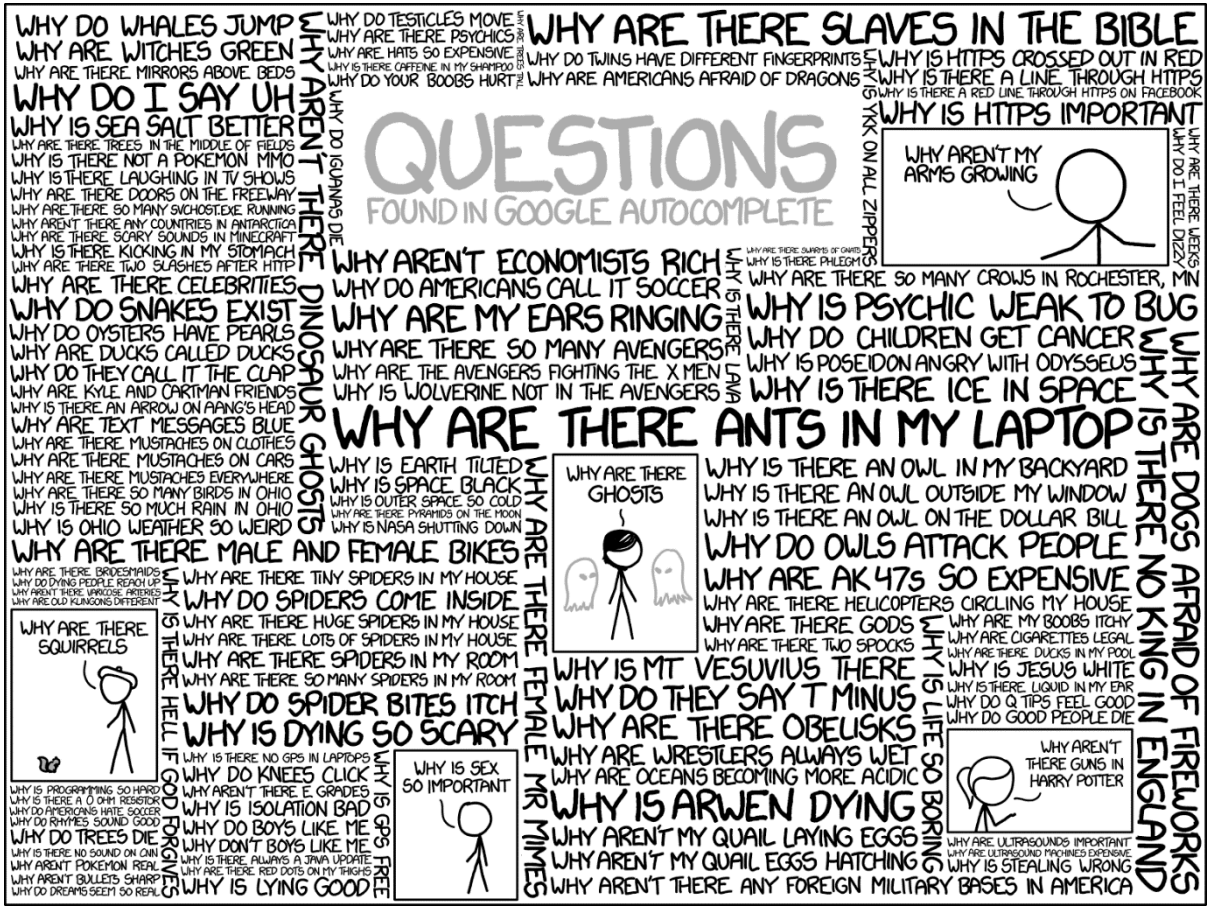
→ fehlerhaftes Paritätsbit

Die Nutzdaten wurden korrekt empfangen:

01001101011

		p_4	p_3	p_2	p_1
Pos. in z	Code	2^3	2^2	2^1	2^0
3	0			0	0
5	1		1		1
6	0		0	0	
7	0		0	0	0
9	1	1			1
10	1	1		1	
11	0	0		0	0
12	1	1	1		
13	0	0	0		0
14	1	1	1	1	
15	1	1	1	1	1
Berechnete Bits		1	0	1	1
Empfangene Bits		1	0	1	0
Fehlerposition		0	0	0	1

Vielen Dank! Fragen?



- Nächste Rechnernetze-Übung am

14.06.2017