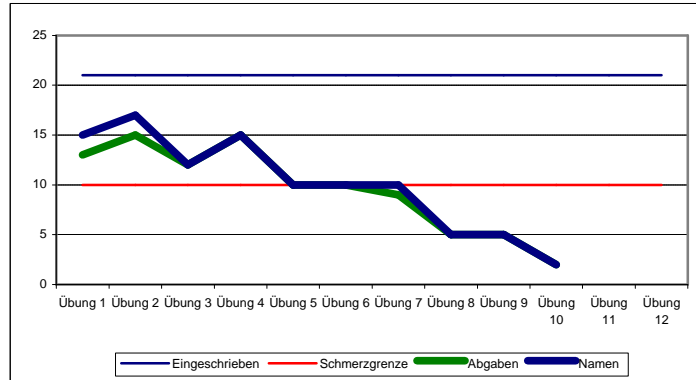


Vernetzte Systeme, Übungsstunde 9: Hinweise

Gabor Cselle, gabor@student.ethz.ch, <http://www.gaborcselle.com/vs/>, 10. Juni 2005

Organisatorisches

- Vorbesprechung Serie 11, Nachbesprechung Übung 9



Vorbesprechung Übung 11

Interaktiv

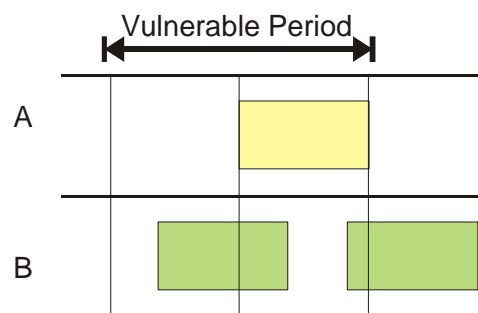
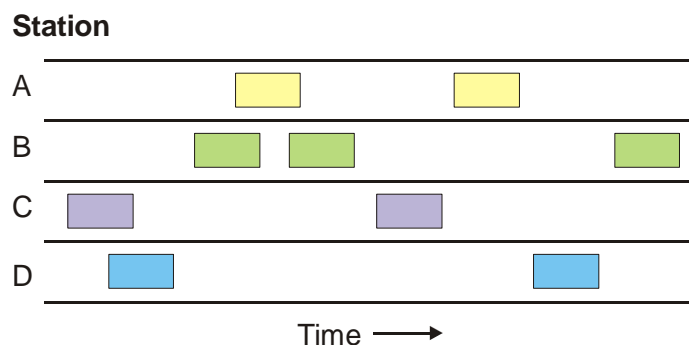
Welcher Typ von "Multiple Access Protocol" ist Pure Aloha?

- A. Channel Partitioning
- B. "Taking Turns"
- C. Random Access

Antwort:

- A. Channel Partitioning: Kanalkapazität aufteilen, z.B. in Zeitslots
- B. Round Robin per Token wäre ein Beispiel hierfür
- C. Richtige Antwort. Bei Aloha sendet jede Station, wenn sie etwas zu senden hat, sofort. Bei Kollisionen wird zufällig lange gewartet und nochmal gesendet.

Pure Aloha

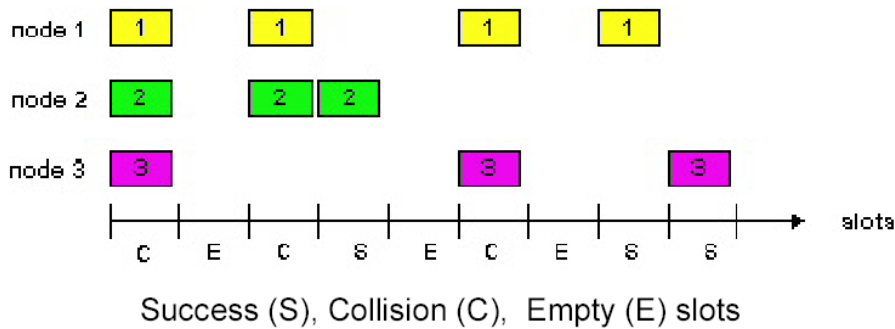


Problem: Vulnerable Period ist 2x so "lang" wie das Paket selbst.

Mögliche Verbesserung?

- Carrier Sense Multiple Access: When a station has data to send, it first listens to the channel to see if anyone else is transmitting at that moment.

Slotted Aloha



- Time is divided into equal size slots
 - A slot is equal to the packet transmission time
- Node with new arriving packet: Transmit at beginning of next slot
- If collision: retransmit packet in future slots with probability p , until successful

Analysis

- We assume that the stations are perfectly synchronous
- In each time slot, each station transmits with probability p

$$P_1 = \Pr[\text{Station 1 succeeds}] = p (1-p)^{n-1}$$

$$P = \Pr[\text{any station succeeds}] = nP_1$$

$$\text{maximize } P: \frac{dP}{dp} = n(1-p)^{n-2}(1-pn) = 0 \rightarrow pn = 1 \rightarrow p = \frac{1}{n}$$

In any time slot, every station should send with $p = 1/n$!

What is the P ? $P = (1 - \frac{1}{n})^{n-1} \geq \frac{1}{e}$, where e is the famous number 2.718...

→ in Slotted Aloha, stations can transmit successfully with prob. $1/e \approx 36\%$

A1.a. Netzwerk mit einem Server, n Clients. Die Sende-wkeit des Servers ist konstant p_S . Was ist die Optimale Sende-wkeit p der n Clients?

$$P_S = \Pr[\text{Server succeeds}] = p_S (1-p)^n$$

$$P_1 = \Pr[\text{One of } n \text{ Clients succeeds}] = p (1-p)^{n-1} (1-p_S)$$

$$P = \Pr[\text{Any Station succeeds}] = n P_1 + P_S$$

Please try at home: Was ist nun der optimale Wert für p ?

A1.b.+c.: Einfach einsetzen

A1.c.+d.: Gefundene Werte kommentieren, mit Reservationssystem vergleichen.

Interaktiv

Wie drückt man Bitfolge "1010'0101" hexadezimal aus?

- A. "0xA5"
- B. "0x57"
- C. "0xC3"

Antwort: "A5x" Konversionstabelle:

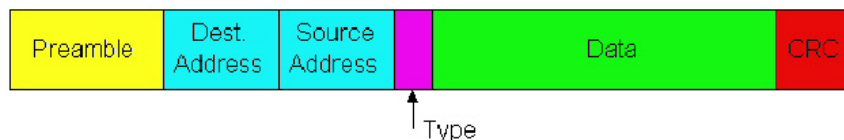
Bin	0000	0001	0010	0011	0100	0101	0110	0111
Hex	0	1	2	3	4	5	6	7
Bin	1000	1001	1010	1011	1100	1101	1110	1111
Hex	8	9	A	B	C	D	E	F

A2.a. Gegeben ein Ethernet-Paket als Hexstring:

AA AA AA AA AA AA AA AB 00 90 27 A2 C3 A7 00 D0 BC EC AA ...

Welche Pakete sind hier ineinander verschachtelt?

Format eines Ethernet-Paketes: Folie 5/52



Format IP-Paket: Folie 4/48; TCP: Folie 3/43; UDP: 3/9

Protokollfeld im IP-Header:

Aus: <http://www.sans.org/resources/tcpip.pdf>: TCP = 6, UDP = 17

A2.b.+c. Wie lang sind die eigentlichen Daten, was ist die Anwendung?

A2.d. Checksumme IP-Header überprüfen:

- Die Checksumme im IP-Header wird nur über den IP-Header selbst berechnet, nicht über die Daten.
- Während der Berechnung der Checksumme wird das Checksummen-Feld auf 0 gesetzt.

Berechnung:

- 16bit/2Byte-weise addieren, Übertrag an 1. Stelle wieder dazurechnen.
- Zum Schluss alle Bits negieren

Beispiel: Checksumme von 2B1A'4157'A121 berechnen

```

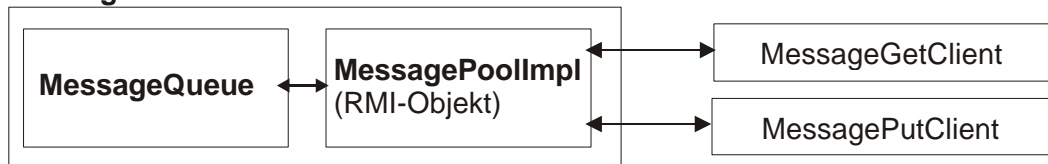
2B1A
+4157
-----
6C71
+A121
-----
10D92  1. Stelle:
Übertrag!
-----
0D93
Binär:  0000'1101'1001'0011
Negiert:1111'0010'0110'1100
    
```

Wie prüft man nachher auf Korrektheit des Pakets? Man addiert gesamtes Paket mit dem obigen Verfahren. Das Ergebnis sollte 1111 '1111 '1111 '1111 sein.

Nachbesprechung Übung 9

Aufgabe: Einfachen Queue-Server implementieren

MessagePoolServer



```
public class MessageQueue {
    private Vector queue = new Vector();

    //Get messages out of the FIFO Queue
    public String dequeue() {
        if (queue.isEmpty()) return null;
        else {
            String msg = ((Message) queue.remove(0)).getContent();
            return msg;
        }
    }
    //Add messages into the FIFO Queue
    public boolean enqueue(String msg) {
        int size = queue.size();
        if (size >= capacity) {
            return false;
        }
        else {
            queue.addElement(new Message(msg));
            return true;
        }
    }
}
```

```
public class MessagePoolServer {
    public static void main(String[] args) {
        try{
            MessagePoolImpl m = new MessagePoolImpl();
            Naming.rebind("messagePool" , m);
            System.out.println("MessagePool Server Ready!");
        }
    }
}
```

```

public class MessagePoolImpl implements MessagePool {
    public MessagePoolImpl() throws RemoteException{
        fifoQueue = MessageQueue.instance();
        UnicastRemoteObject.exportObject(this);
    }

    public void put(String msg) throws RemoteException {
        if (msg == null) {
            throw (new MessageNullException("Error: Message empty"));
        }
        else {
            boolean result = fifoQueue.enqueue(msg);
            if (!result){
                throw (new QueueFullException("Error: Queue full!"));
            }
        }
    }

    public String get() throws RemoteException {
        String msg = fifoQueue.dequeue();
        if (msg == null) {
            throw (new QueueEmptyException("Error: queue empty!!"));
        }
        return msg;
    }
}

```

```

public class MessageGetClient { //ohne Fehlerbehandlung!
    MessagePool pool = (MessagePool ) Naming.lookup(args[0]);
    while(true) {
        Thread.sleep(1000);
        String msg = pool.get();
        System.out.println("The content new message: " + msg);
    }
}

```

```

public class MessagePutClient { //ohne Fehlerbehandlung!
    MessagePool pool = (MessagePool ) Naming.lookup(args[0]);
    while(true) {
        Thread.sleep(1000);
        String msg = String.valueOf(System.currentTimeMillis());
        pool.put(msg);
    }
}

```