

Zusammenfassung - Netzwerke II

Drahtlose Netzwerke

1.1.1 Grundlagen

Wireless Host (Drahtloser Teilnehmer): Endsystem auf dem die Applikation läuft (stationär oder mobile), z.B. Smartphone, PC

Wireless Link (Drahtlose Verbindung): Verbindet Teilnehmer direkt oder per Basisstation (Abdeckung, Datenrate)

Basisstation (Base Station): Überträgt Datenpakete zwischen drahtgebundenem zu drahtlosem Netzwerk, meist mit drahtgebundenem Netzwerk verbunden (WLAN Access Point, UMTS Basisstation)

Drahtloses Infrastruktur Netzwerk: Netzwerkteilnehmer sind über Basisstation mit dem Netz verbunden

Drahtloses Ad-Hoc Netzwerk: Keine Infrastruktur (Basisstationen), Teilnehmer bilden das Netz selbst.

Nachteile: passive Teilnehmer haben trotzdem Stromverbrauch, eigene Daten landen auf fremden Mobiltelefonen und höhere Latenz

Single-Hop: Genau ein wireless Link

Multi-Hop: Übertragung geht über mehrere wireless Links in Folge

| Beispiele für Single und Multi-Hop | | |
|---|--|---|
| Übliche Datenraten | Single Hop | Multiple Hops |
| GSM (2G) 0.56 Mb/s | Host verbindet sich mit Basisstation (Wifi, zellulare Netzwerke) | Host muss möglicherweise durch mehrere drahtlose Geräte um sich mit dem Internet zu verbinden: <i>Mesh Net</i> |
| UMTS (3G) 4 Mb/s | und diese dann mit dem Internet | |
| LTE (4G) und 802.11b 5 - 11 Mb/s | | |
| 802.11ag 54 Mb/s | Keine Basisstation und auch keine Verbindung zu weiterem Internet (z.B. Bluetooth) | Keine Basisstation und auch keine Verbindung zu weiterem Internet. Muss durch mehrere drahtlose Geräte: <i>MANET, VANET</i> |
| 802.11n 200 Mb/s | | |

Herausforderungen bei drahtloser Übertragung

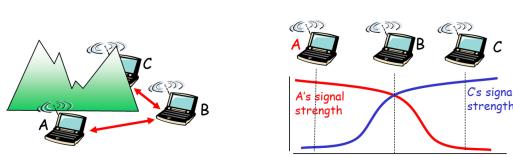
- Teilnehmer zeitweise nicht erreichbar (Funkloch)
- IP-Adresse ändert sich
- Höhere Anzahl an Übertragungsfehlern durch Interferenz (Störung durch andere Teilnehmer) oder Dämpfung → Bessere Fehlerbehandlung
- Kurzer Paketverlust führt bei TCP zu angeblicher Netzüberlastung (obwohl nur kurzzeitige Störung)
- Medium kann abgehört werden
- Mehrwege-Ausbreitung: Signale werden an unterschiedlichsten Oberflächen reflektiert → Am Empfänger sowohl konstruktive als auch destruktive Überlagerung möglich

⇒ Funkkanal ist zeit- und ortsvariant!

Modulationsarten: Frequenz-, Amplituden- & Phasenmodulation, Quadraturamplitudenmodulation (QAM) ⇒ Kombination von Amplituden- und Phasenmodulation (*QAM-8*: 3 Bit pro Symbol, *QAM-1024*: 10 Bit pro Symbol).

Höhere Modulationsarten bieten höhere Übertragungsrate sind aber fehleranfälliger. Bei größerem Signal-Rausch-Abstand (SNR - Stärke des Nutzsignals bezogen auf Störung) kann höhere Modulation eingesetzt werden da Kanal anscheinend nicht so stark gestört (*QAM-16 = 4Mbps, QAM-256 = 8Mbps*)

Bit-Error-Rate (BER): Wahrscheinlichkeit, dass ein fehlerhaftes Bit übertragen wird.



- **Hidden Terminal Problem:** Teilnehmer A, B & C. A und B hören sich, B und C hören sich aber A und C hören sich nicht → Bei Übertragung $A \rightarrow B$ und $C \rightarrow B$ stören sie sich unbewusst gegenseitig.
 - Beheben durch *RTS/CTS*-Verfahren
 - Beheben mit Sendestärke bei A & C erhöhen, sodass sie sich hören können und CSMA/CA funktioniert
 - Teilnehmer bewegen

Aufteilen eines Mediums:

- **TDMA (Time Division Multiple Access)**

1. synchron: Jeder Teilnehmer hat festen Zeitslot, nur in diesem kann er senden
2. asynchron: keine festen Zeitslot, jeder nutzt aktuellen Zeitslot wenn er Daten hat - Absender wird in Header geschrieben

- **FDMA (Frequency Division Multiple Access)**

1. Teilnehmer nutzen unterschiedliche Frequenzen

- **CDMA (Code Division Multiple Access)**

1. Teilnehmer nutzen unterschiedliche Spreizcodes, **Vorteil:** Störungsunempfindlicher, **Nachteil:** Mehr Datenübertragung
2. Zu übertragende Daten werden vom Sender mit Spreizcode multipliziert, Ergebnisbits \Leftrightarrow Chips
3. Empfänger multipliziert empfangene Daten mit Spreizcode des Senders
4. Teilnehmer senden zur gleichen Zeit im gleichen Band, Daten werden beim Empfänger durch bitweise Multiplikation mit Code zurückgewonnen
5. Andere Teilnehmer wirken als zusätzliches Rauschen (\Rightarrow umso mehr Teilnehmer umso geringerer SNR \Rightarrow Sendeleistung erhöhen)

CDMA - Beispiel zur Kodierung

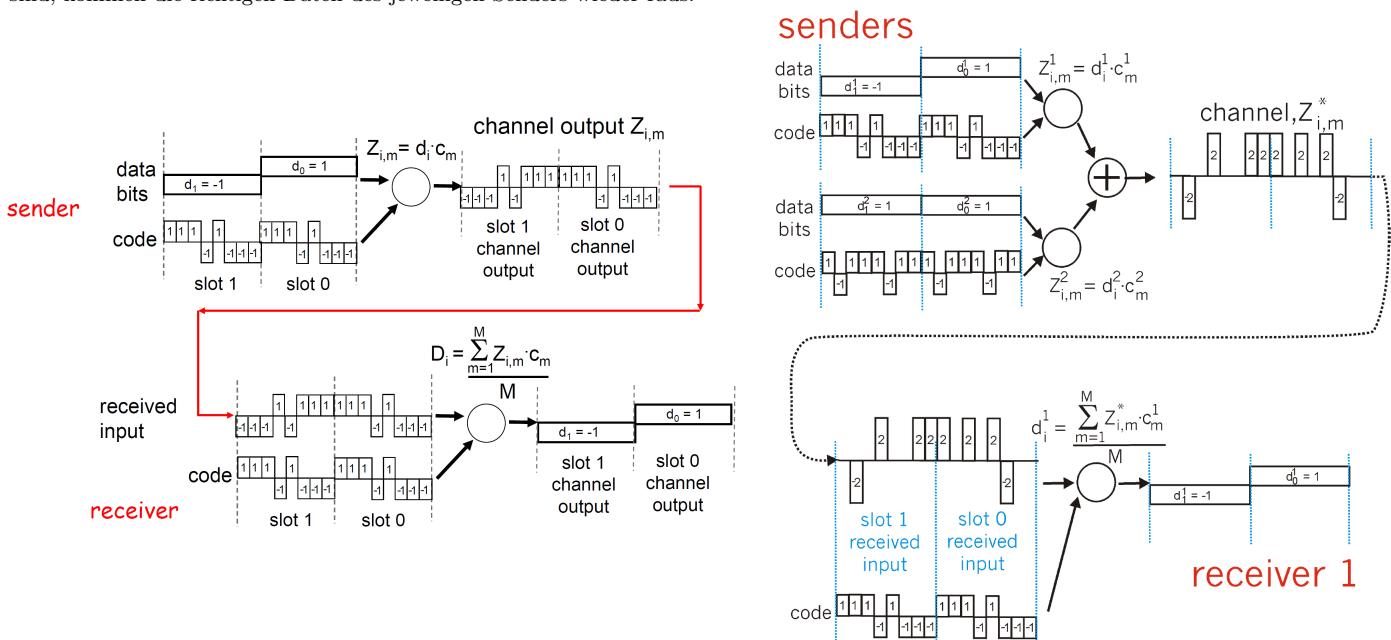
- o Sender hat Spreizcode $(1, 1, 1, -1, 1, -1, -1, -1)$ und versendet Daten $d_1 = -1, d_0 = 1$. Nach Multiplikation der Daten mit Spreizcode: $Z_{1,m} = (-1, -1, -1, 1, -1, 1, 1, 1)$, $Z_{0,m} = (1, 1, 1, -1, 1, -1, -1, -1)$.

- o Empfänger dekodiert folgendermaßen: $\frac{\sum_{m=1}^M Z_{i,m} \cdot c_m}{M}$ mit M = Länge des Spreizcodes (in diesem Beispiel 8), c_m = Spreizfaktor an der Stelle m , $Z_{i,m}$ = die gespreizten Daten, d.h.:

$$d_1 = \frac{(-1) + (-1) \cdot 1 + (-1) \cdot 1 + 1 \cdot (-1) + (-1) \cdot 1 + 1 \cdot (-1) + 1 \cdot (-1)}{8} = \frac{-8}{8} = -1$$

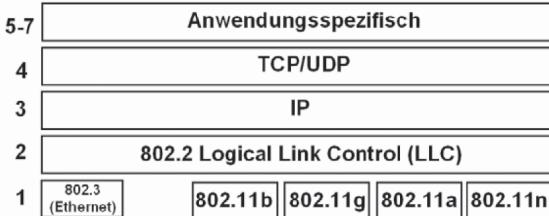
$$d_0 = \frac{1 \cdot 1 + 1 \cdot 1 + 1 \cdot (-1) + 1 \cdot 1 + (-1) \cdot 1 + (-1) \cdot (-1) + (-1) \cdot (-1)}{8} = \frac{8}{8} = 1$$

- o Bei mehreren Sendern multipliziert der Empfänger das überlagerte Signal mit dem jeweiligen Spreizcode, da diese orthogonal zueinander sind, kommen die richtigen Daten des jeweiligen Senders wieder raus.



1.1.2 Wireless Local Area Networks

Protokollstack:



802.11: '97, FHSS/DSSS, 1-2MBit/s, 2.4 GHz

802.11b: '99, DSSS, 1 - 11MBit/s, 2.4 GHz

802.11n: '09, OFDM/MIMO, 6 - 600MBit/s, 2.4 oder 5 GHz

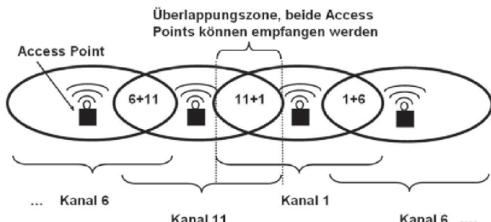
802.11ac: '14, MU-MIMO, bis zu 6.93 GBit/s, 5 GHz

802.11ay: '19, 20 - 40 GBit/s, 60GHz

Begriffe:

- **Basic Service Set (BSS):** Stationen die auf dem gleichen Übertragungskanal Daten austauschen
- **Extended Service Set (ESS):** Zusammenschluss mehrerer BSS zu Kommunikationsnetz, Roaming zwischen den BSS
- **Service Service Set ID (SSID):** Name des Netzwerkes
- **Ad-Hoc Mode / Independent BSS (IBSS):** Alle Stationen gleichberechtigt, kein Access Point
- **Infrastructure BSS:** Geräte kommunizieren über AP (=Übergang zu drahtgebundenem Netz)

Kanäle:



- Bis zu 13 Kanäle mit je 5 MHz zwischen 2410 MHz - 2483 MHz.
- Bei **DSSS** Kanalbreite = 22MHz, bei **OFDM** = 20 MHz / 40 MHz (ohne / mit Kanalbündelung). Störungsfreier Betrieb nur bei passendem Abstand (5 Kanäle bei DSSS).

Hierzu bitte auch das erste Übungsblatt vom Praktikum durchlesen!

Accesspoint sendet regelmäßig **Beacon Frames** mit SSID und MAC-Adresse. Wireless Stations scannen Kanäle nach diesen Frames, wählen verfügbaren AP (Einstellungen & Signalstärke) aus, führt Authentifizierung durch und erhält anschließend IP-Adresse per DHCP.

Passives Scannen

PC ist passiv, hört Kanal ab

APs senden **Beacons**

PC sendet Association Request an ausgewählten AP

AP sendet Association Response an PC

Keine Kollisionserkennung (Collision Detection (CD)) möglich ⇒ ACKs auf Schicht 2 nötig

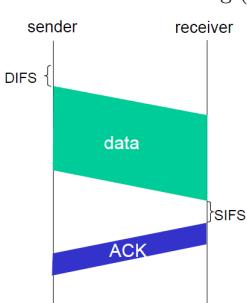
Aktives Scannen

PC sendet Probe Request

APs senden Probe Response

PC sendet Association Request an ausgewählten AP

AP sendet Association Response an PC



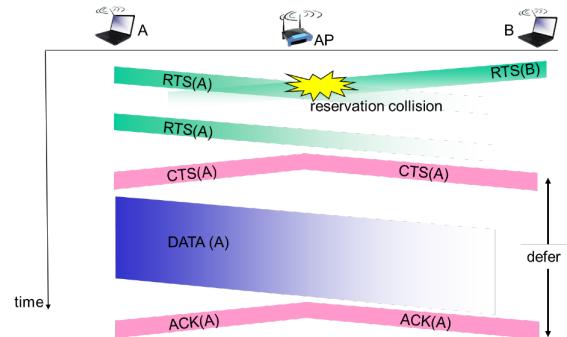
- Sender wartet Zeitspanne **DIFS** (Distributed Coordination Function Interframe Spacing) während der Medium frei sein muss
- Sender überträgt Daten (kein Collision Detection)
- Empfänger prüft CRC
- Empfänger sendet ACK falls CRC korrekt nach Wartezeit **SIFS** (Short Interframe Spacing), SIFS < DIFS um Senden eines normalen Frames dazwischen und somit Kollision zu verhindern außerdem Zeit benötigt zum Umschalten von *Empfangen* auf *Senden*

Carrier Sense Multiple Access / Collision Avoidance (CSMA/CA):

- Sender lauscht (*Carrier Sense*) ob Medium frei (Dauer: 1 DIFS)
- Falls frei: *random backoff* im aktuellen *Contention Window* würfeln, da wenn mehrere Stationen gleiche Zeit abwarten würden es sonst zu Kollision käme. Nach Ablauf des random backoffs Daten übertragen
- Falls nicht frei: Warten bis Kanal frei (Ablauf des *Netzbelegungsvektors*), wenn anschließend länger als 1 DIFS frei, dann *random backoff* Timer runterzählen & dann Daten senden
- Falls kein ACK erfolgt: **Contention Window verdoppeln** und Daten erneut senden
- Empfänger sendet ACK nach Ablauf eines **SIFS** bei korrekt erhaltenen Daten
- Kollision durch gleichzeitig ablaufenden *backoff timer* führt zu *ACK-Timeout*

Nachteile CSMA/CA: Senden dauert lange, Kollision wird vom Sender beim Senden nicht erkannt

Kollision ist sehr zeitaufwändig und sollte daher vermieden werden → besser eine Kollision bei kurzen Kontrollpaketen als bei langen Datenpaketen ⇒ RTS/CTS-Verfahren



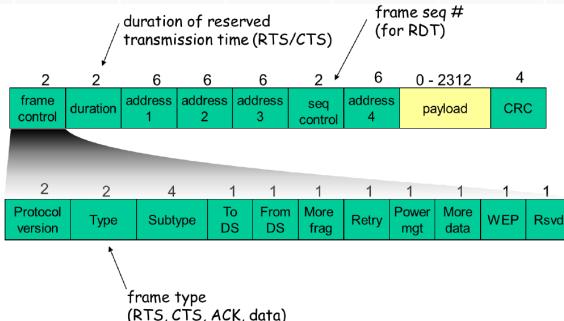
Ablauf:

- Sender reserviert Kanal mit kurzem **Request-To-Send (RTS)**-Paket, Kollisionen möglich aber weniger schlimm da nur kleines Paket welches günstig erneut gesendet werden kann
- Empfänger antwortet mit **Clear-To-Send (CTS)**
- Sender sendet Datenpaket
- ⇒ Andere Stationen empfangen RTS & CTS und berücksichtigen belegten Kanal
- ⇒ Funktioniert auch bei *Hidden Terminal* da CTS empfangen wird

WLAN Rahmenformat

| frame control | duration | address 1 | address 2 | address 3 | seq control | address 4 | payload | CRC |
|---------------|----------|-----------|-----------|-----------|-------------|-----------|----------|-----|
| 2 | 2 | 6 | 6 | 6 | 2 | 6 | 0 - 2312 | 4 |

| Funktion | ToDS | FromDS | Add. 1 | Add. 2 | Add. 3 | Add. 4 |
|--------------|------|--------|-------------|-------------|-------------|--------|
| IBSS | 0 | 0 | destination | source | BSSID | unused |
| To AP | 1 | 0 | BSSID | source | destination | unused |
| From AP | 0 | 1 | destination | BSSID | source | Unused |
| WDS (bridge) | 1 | 1 | receiver | transmitter | destination | source |

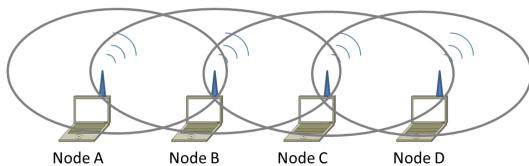


Unterschied CSMA/CA↔CSMA/CD:

CSMA/CD bei Ethernet sendet *JAM*-Signal (= kurzes Störsignal damit alle anderen Teilnehmer Kollision ebenfalls erkennen), CSMA/CA erkennt keine Kollision (versucht nur zu verhindern)

- Bedeutung der Adressfelder variiert je nach Funktion, angezeigt über *ToDS* und *FromDS* Bits (DS=DistributionSystem) in Frame Control
- Max. 2313 Bytes an Nutzdaten
- 3. Adresse erlaubt Umsetzung auf Ethernet-Rahmen
- **WLAN-Reichweitenvergrößerung** aufgrund auf 100mW begrenzte Sendeleistung durch überlappende BSSs in einem IP-Subnetz:
 - IP-Adresse bleibt gleich da identisches Subnetz, nur AP ändert sich
 - Zuständiger, selbstlernender Switch zwischen APs ändert Port↔IP-Zuordnung wenn sich Teilnehmer vom neuen AP meldet

Praktikumsaufgabe 1.1



- 1Mbit/s Datenrate mit TDMA Verfahren auf Layer-2, Node A muss Datei an Node D übertragen, **Maximale Datenrate:**
 - A kann nicht direkt an D senden, sondern nur über B und C, bei *Synchrones TDMA* hat jeder Sender festen Zeitslot, in der er senden darf die Datenrate reduziert sich daher bei *synchrones TDMA* auf $\frac{1}{4}$
 - Bei *Asynchrones TDMA* werden nicht-verwendete Zeitslots besser ausgenutzt so kann z.B. A an B senden während C noch an D sendet die Datenrate reduziert sich daher bei *asynchrones TDMA* auf $\frac{1}{3}$
- D muss nun auf **Transportschicht** jedes Datenpaket mit ACK bestätigen, A & D arbeiten mit **Selective Repeat**, **Maximale Datenrate:**
 - 6 benötigte Zeitslots für 1 Datenpaket: **3x Daten senden:** A→B, B→C, C→D und **3x ACK:** D→C, C→B, B→A, zu einem Zeitpunkt können gleichzeitig Daten von A→B und ein ACK von D→C gesendet werden, **Datenrate reduziert sich daher auf $\frac{1}{5}$**

Praktikumsaufgabe 2.1

- Selbe Anordnung, arbeiten aber nun im **AdHoc-Modus** - welchen Kanal stellt man an jeder Node für optimalen Durchsatz A→D ein?
 - Alle Nodes verwenden den selben Kanal da sich die sonst im **AdHoc-Modus** gar nicht erreichen können
- Nun jede Node **mit zwei WLAN-Adaptern** - welchen Kanal jeweils einstellen und welche **Änderung des Durchsatzes?**

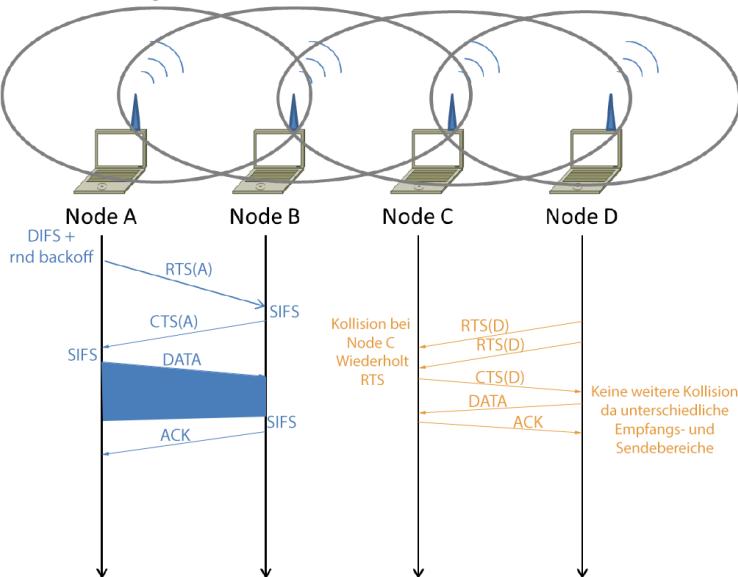
| A | B | C | D |
|-----------|-----------|-----------|-----------|
| Adapter 1 | Adapter 2 | Adapter 1 | Adapter 2 |
| 1 | 11 | 1 | 6 |
| 6 | 6 | 11 | 6 |
| 11 | | | 11 |

- **Verdreifachung des Durchsatzes** da während A→¹B gleichzeitig auch B→⁶C und C→¹¹D senden können

Praktikumsaufgabe 2.2

- ACK-Timeout für Indoor-WLAN abschätzen
 - Übertragungszeit (Lichtgeschwindigkeit) + Umschaltzeit der Antenne von Receiver↔Transmitter
- Selber ACK-Timeout-Wert nun bei 7km Entfernung per Richtantenne - was passiert?
 - Optimalfall: ACK kommt verzögert nach Timeout an und nächstes Paket wird erst dann gesendet
 - Schlimmer Fall: Datenpaket & ACK kollidieren und nichts kommt richtig an
- A, B & C befinden sich in einem BSS - AP versendet ARP-Request um MAC-Adresse der Station A zu bestimmen
 - Welche Zieladresse hat der Rahmen mit dem ARP-Request? Broadcast MAC
 - Welche Station bestätigt den Empfang des Rahmens mit einem ACK? Keine, da Koordination zwischen A, B & C nicht möglich
 - Was passiert, wenn SIFS==DIFS? Kollision möglich, da Sender immer DIFS abwartet, bis er senden darf

Praktikumsaufgabe 2.3



- A möchte Paket an B senden, es wird RTS/CTS genutzt, welche Rahmen werden ausgetauscht? (blau)
- Während B das CTS an A sendet, möchte **gleichzeitig** D ein Paket an C senden - ist dies parallel möglich? Welche Rahmen werden ausgetauscht? (orange)

Praktikumsaufgabe 3.1

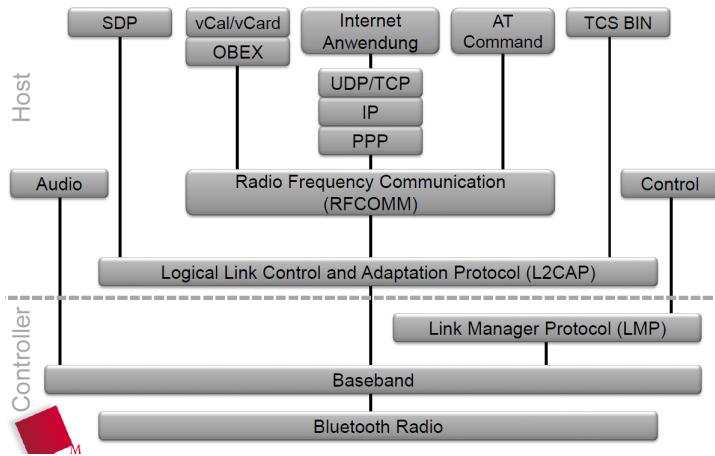
- WLAN Stick mit 150 Mbit/s
 - Welcher maximale Goodput bei direkter Empfangsreichweite ohne Störungen? Üblicherweise die Hälfte - 75 Mbit/s
 - Unterschiede im Transportprotokoll? TCP geringfügig langsamer aufgrund Overhead für Header & ACKs
- Zusätzlich mit 802.11b/g/n ältere WLAN-Karte mit ausschließlich 802.11b 11Mbit/s im Netz
 - Welche Konsequenzen für Arbeitsweise des AP? RTS/CTS, Beacons & Broadcasts werden im ältesten Verfahren gesendet
 - Welche Auswirkungen auf Goodput? Schlechtere Performance da geringere, maximale Datenrate

1.1.3 Personal Area Networks (PAN)

Drahtlos oder drahtgebundenes (Ad Hoc) Netzwerk von Kleingeräten, oft nur wenige Meter Reichweite.

Bluetooth

- Mehrfachzugriff mit TDMA, Frequenzsprungverfahren (Kanalwechsel nach jedem Zeitslot ⇒ Robustheit gegen Störer)
- Class 1: 100mW - 100m Reichweite, ◦ Class 2: 2.5mW - 10m Reichweite, ◦ Class 3: 1mW - 1m Reichweite
- '99: Einführung, 732,2kbit/s ◦ '04: v2.0 mit bis 2.1Mbit/s ◦ '16: v5.0 mit IoT Erweiterungen
- Ad Hoc Netzwerk (keine Infrastruktur nötig) ◦ ≤ 8 aktive & ≤ 255 geparkte Geräte ◦ Master gibt Zeit vor, gewährt Slaves das Recht zu senden, aktiviert geparkte Slaves
- Profil spezifiziert Anwendung von Bluetooth für bestimmten Zweck
 - *Serial Port Profile* für serielle Datenübertragung
 - *Health Device Profile* zur Verbindung medizinischer Geräte



- **AT Kommando:** Kommando zur Modemsteuerung
- **Baseband:** Basisband, Paketformate
- **L2CAP:** Logical Link Control and Adaptation Protocol: Bietet verbindungsorientierte- und lose Dienste zwischen Baseband und höheren Schichten
- **MCAP:** Multi-Channel Adaptation Protocol
Stellt Kontrollkanal *MCL* und Datenkanäle *MDL* bereit
- **RFCOMM:** Virtuelle, serielle Verbindungen, Emulation serieller Ports
- Geräte im HealthCare Bereich ursprünglich per RFCOMM angebunden
⇒ '08 Verabschiedung von standardisiertem *Health Device Profile*
- Zwei Rollen: **Source** = Datenquelle, **Sink** = Empfänger (Smartphone)
Verbindungsau- und abbau, Wiederaufbau abgebrochener Verbindungen

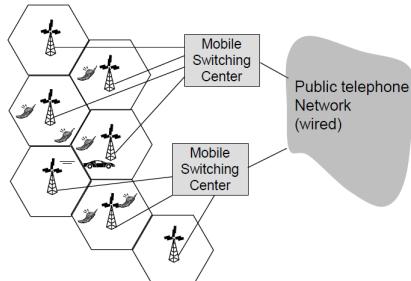
Health Device Profile (HDP)

- Sichert *Interoperabilität durch Standardisierung*
- Zwei Rollen: **Source** (Datenquelle, z.B. Waage) und **Sink** (Datensenke, z.B. Smartphone)
- **Kernfunktion:** Verbindungsau- und abbau, Wiederaufbau abgebrochener Verbindungen, Synchronisation
- Sensoren (z.B. Pulsmesser, Thermometer) sollen lange Laufzeit (⇒ geringer Stromverbrauch) aufweisen
- Bluetooth ab 4.0 beinhaltet *Bluetooth Smart* (Low-Energy Profil auf Basis von einem *Generic Attribute Profile (GATT)*).

ZigBee

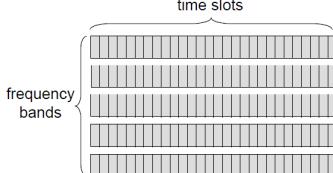
- Ziel: Drahtlose Übertragung bei geringem Stromverbrauch (geringe Datenraten: 20 - 250kbit/s, selten aktiv (*low duty-cycle*)).
- Übertragen von Sensordaten, Heim- und Gebäudeautomatisierung
- **Endgerät:** Reduced Function Device *RFD* - nur Teil des ZigBee Protokolls implementiert (geringere Kosten)
- **Router:** Full Function Device *FDD*, kann Daten weiterleiten ◦ **Koordinator:** Gibt zusätzliche Parameter vor, koordiniert das PAN

1.1.4 Zelluläre Netzwerke



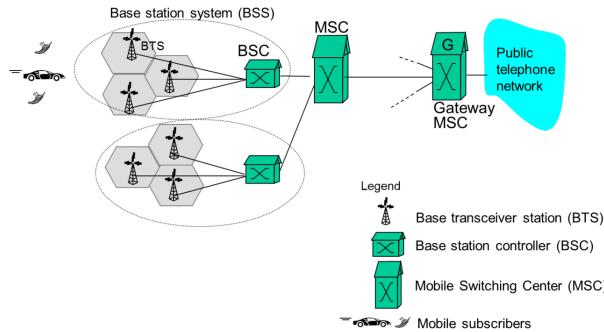
- **1G:** Analog (A/B/C-Netz)
- **2G:** GSM ab '92, 2.5G = GPRS, 2.75G = EDGE
- **3G:** UMTS ab '03
- **4G:** ab '14 LTE
- **5G:** ab '21, Latent < 1ms
- **Mobilfunkzelle:** Von Base Transceiver Station (BTS) abgedeckter Bereich
- **Air-Interface:** Untere 2 Netzwerkschichten Mobile Station ↔ BTS
- **Mobile Switching Center (MSC):** Anrufauf- & abbau, Verbindung ins Festnetz

Ressourcenzuteilung in der Zelle:

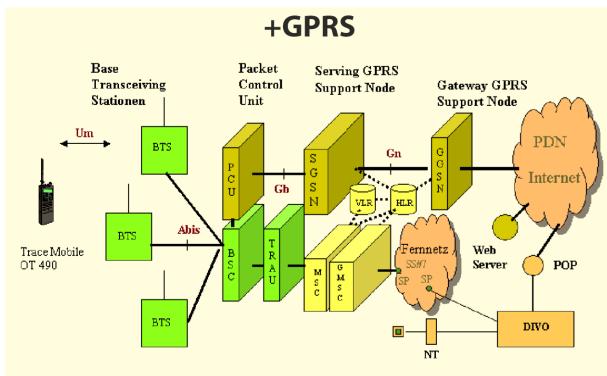


- **GSM:** Kombination aus FDMA & TDMA, Spektrum wird in einzelne Frequenzkanäle, jeder Kanal wiederum in Zeitschlitzte aufgeteilt
- **UMTS:** CDMA Verfahren - Unterschiedlicher Code für unterschiedliche Nutzer

2G Netzwerkarchitektur (GSM):



- **Base Station Controller (BSC):** übernimmt Ressourcenzuweisung und Mobilitätsmanagement in einem Base Station Subsystem (BSS)
- **Mobile Switching Center (MSC):** Anrufauf- und Abbau, Verbindung ins Festnetz, Mobilitätsmanagement
- **Gateway-MSC:** Vermittlungsfunktionen, Verbindung zu anderen Netzen bzw. Festnetz
- **PSTN:** Public Switched Telephone Network

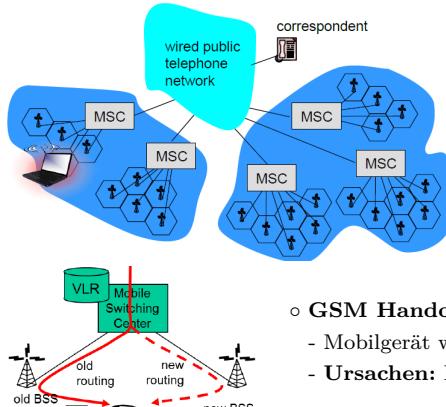


- Mit GPRS-Komponenten erstmals paketvermittelte Datendienste - paralleles Datennetzwerk, welches bereits bestehende BTS mitverwendet
- **Packet Control Unit (PCU):** Kommuniziert über den BSC mit Endgerät und auch mit der SGSN, überwacht und verwaltet Datenpakete, Ressourcenverteilung in Form von *time slots*
- **Serving GPRS Support Node (SGSN):** Übernimmt Vermittlung der Datenpakete und die Funktion des VLR
- **Gateway GPRS Support Node (GGSN):** Ist der Router, der das Mobilfunknetz mit dem Internet verbindet und die IP-Adresse zur Verfügung stellt
- **Home Location Register (HLR):** Datenbank mit Informationen zum Nutzer, enthält Rufnummer und zuletzt bekannten Aufenthaltsort
- **Visitor Location Register (VLR):** Datenbank mit Informationen zu allen Nutzern, die sich im vom MSC bedienten Bereich befinden (Kopie aus HLR)
- Sprachdaten laufen über *BTS ↔ BSC ↔ MSC ↔ GMSC ↔ Festnetz*
- Paketdaten laufen über *BTS ↔ BSC ↔ PCU ↔ SGSN ↔ GGSN ↔ Internet*

Praktikumsaufgabe 5.1

- Welche Aufgabe erfüllt die Base Transceiver Station (BTS) im GSM Netz?
 - Frequenzmodulation, Kodierungsverfahren (Fehlerschutz), Synchronisation mit dem Endgerät durch Senden von *Bursts*
- Warum kann sie beim Übergang von GSM → UMTS nicht weiterverwendet werden?
 - GMS und UMTS nutzen unterschiedliche Frequenzen & unterschiedliche Verfahren zur Datenübertragung (TDMA vs CDMA)
- Ein mobiler Rechner nutzt GPRS/EDGE zur Verbindung ins Internet - woher bekommt er seine IP?
 - Über GGSN, die Überblick über alle BTS & BSCs hat: keine IP wird doppelt vergeben,
kann bei Bewegung & Handovers beibehalten werden
- Welche Umstände führen **zur Änderung** seiner IP?
 - Wenn Verbindung zum Teilnehmer abreißt, wird die IP-Adresse nach gewisser Zeit wieder durch GGSN freigegeben und Teilnehmer erhält neue IP
- Kunde beschwert sich über träge reagierendes System sobald Kommunikation über Drahtlosnetzwerk - Ursachen & Problemanalyse?
 - Hohe Latenz bzw. geringe Datenübertragung durch alten Netzstandard (z.B. *EDGE*)
 - Viele Teilnehmer führen zu Störungen durch Paketkollision (Hinweis darauf ist ein kleiner Wert bei TCP Congestion Window)
 - Abfragen des Drahtlosmoduls per AT-Kommandos für weitere Informationen

Mobilitätsmanagement:

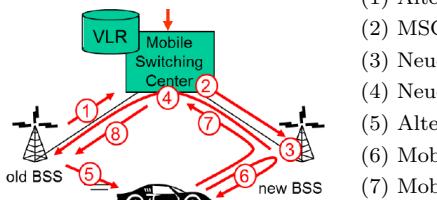


- Nutzer kann sich zwischen Zellen verschiedener *MSCs*, auch von anderen Providern, bewegen
- Mobilgerät prüft im eingeschalteten Zustand, ob sich aktuelle Location Area ändert
⇒ sendet Location Update mit neuer Area ⇒ Home Location Register (HLR) wird aktualisiert
- Eingehender Anruf: Befragung von *HLR* nach aktuellem Ort des Angerufenen über dessen temporäre *Roaming-Nummer*, anschließend Verbindaufbau über das *VLR* des *MSC*
⇒ Falls gerade keine Verbindung besteht: Broadcast-Nachricht über alle Basisstationen des jeweiligen *MSC* (*Paging*), Mobile meldet sich ggf., damit ist genaue Basisstation bekannt (*indirektes Routing*)

GSM Handover (= MSC/Inter-BSC Handover)

- Mobilgerät wechselt bei bestehender Verbindung von einer Basisstation zur anderen.
- Ursachen: Bewegung des Nutzers (stärkeres Signal eines anderen BSS), aktuelle Zelle überlastet

Ablauf Inter-BSC Handover:

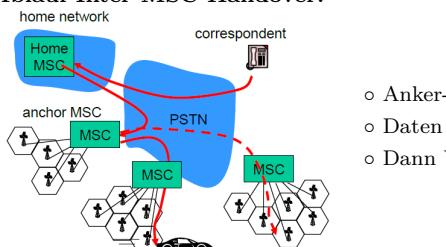


- (1) Altes BSS informiert MSC über anstehendes Handover
- (2) MSC reserviert Ressourcen zu neuer BSS
- (3) Neue BSS reserviert Zeitslot (TDMA)
- (4) Neue BSS signalisiert an MSC und alte BSS Bereitschaft zum Handover
- (5) Alte BSS weist Mobilgerät an, Handover zu neuer BSS durchzuführen
- (6) Mobilgerät aktiviert Kanal in neuer BSS
- (7) Mobilgerät bestätigt Handover an MSC, diese leitete Daten um
- (8) MSC weist alte BSS an, Ressourcen des Mobilgeräts freizugeben

Arten von Handover:

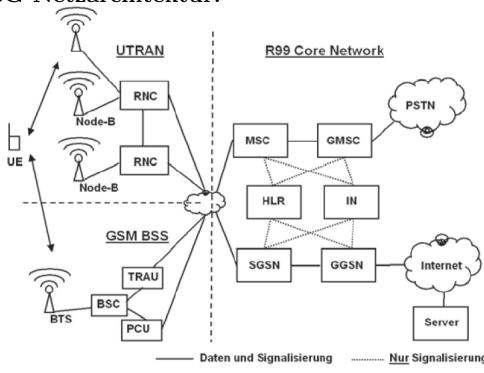
- *Intra BSC*: Aktuelle und neue Zelle gehören zum selben BSC
- *Inter BSC*: Aktuelle und neue Zelle gehören zu unterschiedlichen BSC aber gleichen MSC
- *Inter MSC*: Aktuelle und neue Zelle gehören zu unterschiedlichen MSC
- *Subsequent Inter MSC*: Teilnehmer wechselt nach *Inter MSC* in Zelle eines dritten MSC
- *Subsequent Handback*: Teilnehmer wechselt nach *Inter MSC* zurück in Gebiet des ersten MSC

Ablauf Inter-MSC Handover:



- Anker-*MSC* = erste *MSC* während eines Anrufs
- Daten bzw. der Anruf wird zunächst an Anker-*MSC* geleitet
- Dann Weiterleitung zu aktueller *MSC*

3G Netzarchitektur:

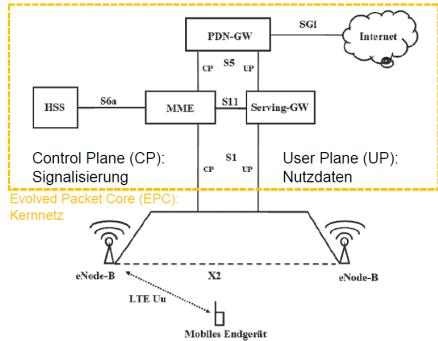


- Mit UMTS '99 neues Air-Interface *Universal Terrestrial Radio Access Network (UTRAN)*
- basierend auf Wideband-CDMA
- Ab UMTS v4 Umstellung auf IP basierte Kommunikation (Sprache & Daten)

LTE & LTE Advanced (4G):

- Umstellung auf *Orthogonal Frequency Devision Multiplexing (OFDM)*
 - Übertragungstechnik mit flexibler Bandbreite zwischen 1.25 bis 20 MHz
- LTE-Endgerät muss Mehrantennenverfahren unterstützen (*Multiple Input, Multiple Output MIMO*)
 - LTE-Netz rein paketbasiert (Sprache per VoIP)

1.1.5 Long Term Evolution (LTE, LTE-A)



- **LTE Release 10:** LTE Advanced - 3 Gbit/s DL bzw. 1.5 Gbit/s UL, Bündelung bis zu 5 Carrier
- **LTE Release 13/14:** LTE Advanced Pro - Bündelung von bis zu 32 Carriern, QAM-256, LTE Narrow Band IoT, **5G** Standardisierung läuft
- **LTE Release 14:** 5G Phase 1, 2018 fertiggestellt, New Radio + 5G Standalone
- **LTE Release 15:** 5G Phase 2, geplant für 20.12.2020
- **Packet Data Network Gateway:** Internetschnittstelle
- **Mobility Management Entity:** Mobilitätsmanagement
- **Serving-Gateway:** Weiterleitung von Nutzdaten ins Kernnetz
- **Home Suscriber Service:** entspricht HLR bei GSM
- **evolved Node-B (eNode-B):** Basisstation

1.1.6 5G

- **Enhanced Mobile Broadband (eMBB):** Hohe Datenraten
- **Massive Machine Type Communications (mMTC):** IoT Anwendungen
- **Ultra-Reliable and Low-Latency Communication (uRLLC):** z.B. drahtlose Vernetzung in der Produktion
- **5G New Radio:** Nutzung mehrerer Antennen (*MIMO*), mmWave-Bänder (24 - 30GHz), flexible und kürzere Slotzeiten (<1ms)
- **Service-Driven 5G Architecture:** Höhere Flexibilität durch *Software-Defined Networking*
- **Ziele:** E2E Latenz < 1ms, 1000x höheres Datenvolumen, 10-100x mehr Geräte, 10-100x mehr typische Nutzerdatenraten
- **Umsetzung:** 5G Zellen mit LTE-Kernnetz (non-standalone), 2-3Gbit/s, dann 5G mit 5G-Kernnetz (standalone), mmWave Bänder

1.1.7 Zusammenfassung Drahtlose Netzwerke

- Ersetzen der unteren 2 Schichten (Link- & Physical-Layer) durch drahtlose Varianten
 - ⇒ Auswirkungen auf obere Schichten minimal, aber:
 - Fehlinterpretation von Paketverlusten auf drahtlosem Link von TCP führt zu Congestion Window Verringerrung ⇒ Datenrate sinkt
 - Verzögerung durch Link-Layer Retransmission (Auswirkungen auf Echtzeitanwendungen)
 - Drahtloser Link meist geringere Datenrate

Ereignisorientierte Systeme

- Untersuchung von Netzwerken durch Messungen aufwändig (Hardware/Vielzahl an Messungen/interagierender Komponenten)
- ⇒ Modellierung und Simulation eines Systems

1.2.1 Grundlagen



- Modellierung des Systemverhaltens als zeitlich diskrete Abfolge einzelner Ereignisse
- Zustandsübergänge werden immer durch Ereignis verursacht
 - Zwischen zwei zeitlich aufeinanderfolgenden Ereignissen kann es keine Zustandsänderung geben

Komponenten:

- Zustandsmodell
- Ereignisse (mit diskreter Zeit & Ziel)
- Ereigniswarteschlange
- Ereignisroutinen
- Aktuelle, in der Simulation angenommene Zeit

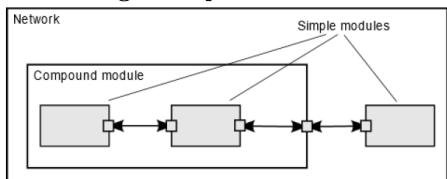
Ablauf:

- Zeitlich nächstes Event aus *Event Queue* entnehmen
- Simulationszeit auf Zeitpunkt des nächsten Events setzen
- Passenden Event-Handler in Event-Ziel aufrufen (generiert ggf. neue Events und fügt sie in Event-Queue ein)
- Falls Bedingung für Simulationsende erreicht: Ende, ansonsten weiter bei 1.

1.2.2 Simulationsplattform OMNeT++

- In Industrie und Forschung weit verbreitetes Discrete Event Network Simulation Framework
- Zur Modellierung von Kommunikationsnetzwerken, Analyse komplexer Softwaresysteme und Validierung von Hardware-Architekturen
- Ist **kein** Simulator **sondern** Plattform um Simulationen zu entwickeln
- Bietet
 - objektorientiertes, modulares Grundgerüst
 - Konzepte für einfache und zusammengesetzte Module sowie Kommunikation zwischen Modulen
 - Logging, Visualisierung, Auswertung (Text und 2D-Graphisch), Debugging
 - Parallele und Verteilte Simulation

Modellierungskonzepte



- Modell** besteht aus **Modulen** die miteinander über **Messages** kommunizieren
- Einfache Module werden in C++ implementiert
- Module lassen sich zu **Compound Modules** kombinieren
- Beschreibung der Struktur in **NED-Language**
- Gesamtmodell wird als **Network** bezeichnet

Bestandteile einer Simulation

- Laufzeitparameter (festgelegt in Datei *omnetpp.ini*)
- Simulationsmodell
 - Struktur der Simulation (festgelegt in Datei *.ned*)
 - Struktur der Nachrichten (festgelegt in Datei *.msg* (generiert *_m.cc* und *_m.h*))
 - Verhalten (festgelegt über C++ Code)

NEtwork Description Language (NED)

- Festlegung** von Ein- und Ausgängen der Module und **Deklaration** von Signalen zur Kommunikation
- Einzelne Module zu größeren Modulen (*Compound Modules*) mit *hierarchische Strukturen* verbinden
- Weitere Features: Vererbung, Packages, Innere Typen, Annotations

Nachrichten und Pakete

- Message:** allgemeine Nachricht mit Feldern für Name, Art der Nachricht, *time stamp*, *send-* & *arrival time*
- Paket:** Felder für Paketlänge, Inneres Paket der darüberliegenden Schicht, *bit error flag*

Statistiken

- Aufzeichnung einzelner (z.B. versendete Datenmenge) oder vieler (z.B. zeitlicher Verlauf der Senderate) Werte pro Simulationslauf

Security

1.3.1 Grundlagen

- **Vertraulichkeit:** Nur Sender und rechtmäßige Empfänger sollen die Nachricht verstehen können
- **Integrität:** Sicherstellen, dass Nachricht unverändert ist (ob durch Übertragungsfehler oder Angriff)
- **Authentisierung:** Nachweis, dass eine Person diejenige ist, die sie vorgibt zu sein
- **Authentifizierung:** Sicherstellen, dass Kommunikationspartner derjenige ist, für den er sich ausgibt (Prüfung der *Authentisierung*)
- **Authorisierung:** Nachweis von speziellen Rechten
- **Betriebssicherheit:** Absicherung des Firmennetzes gegen Eingriffe von außen

1.3.2 Grundlagen der Kryptographie

Arten von Verschlüsselung

- K_A : Schlüssel für Verschlüsselung, K_B : Schlüssel für Entschlüsselung, m : Klartext, $K_A(m)$: Ciphertext - Klartext verschlüsselt mit K_A
 $m = K_B(K_A(m))$
- **Symmetrisch:** K_A identisch zu K_B , Verfahren z.B. *AES*
 - **Public Key:** Paar unterschiedlicher Schlüssel: $K_A = K_B^+$ und K_B^- , ein Schlüssel ist beiden bekannt (*public key* - K_B^+), der andere nur Empfänger (*private key* - K_B^-), Verfahren z.B. *RSA*

Arten von Angriffen

- **Cipher-Text only:** Angreifer hat nur Geheimtext, entweder alle möglichen Schlüssel ausprobieren (*brute force*) oder statistische Analyse
- **Known-Plaintext:** Angreifer kennt Klar- und zugehörigen Geheimtext, kann Rückschlüsse auf Schlüssel ziehen
- **Chosen-Plaintext:** Angreifer kann Geheimtext zu selbstgewählten Klartext bekommen, Verschlüsselungsalgorithmus ggf. ausnutzbar

Einfache symmetrische Verschlüsselungen

- **Cesar-Chiffre:** Verschiebung des Alphabets als Schlüssel, Abbildung des Klartextes auf verschobenes Alphabet ergibt Ciphertext
- **Substitutions-Chiffre:** Schlüssel ist eine Abbildungsvorschrift (Buchstabe b wird abgebildet auf b')
- **Blockchiffre:** Verarbeitung des Klartextes in k -Bit großen Blöcken (d.h. Abbildung eines k -Bit Blocks auf k -Bit Geheimtext)
z.B. für $k = 3$: $000 \Rightarrow 110, 001 \Rightarrow 111, 010 \Rightarrow 101, 011 \Rightarrow 100, 100 \Rightarrow 011, 101 \Rightarrow 010, 110 \Rightarrow 000, 111 \Rightarrow 001$
Der Schlüssel ist die Abbildungstabelle, Anzahl möglicher Abbildungen: $(2^k)!$, d.h. für $k = 3$ gibt es 8!, also 40320, Abbildungsmöglichkeiten
⇒ Heutige Blockchiffren (*DES*, *3DES*, *AES*) verwenden Funktionen um Abbildungen zu erzeugen da bereits kleine k riesige Tabellen erzeugen

Cipher Block Chaining

- Identischer Klartext produziert identischen Geheimtext
⇒ Rückschlüsse auf Schlüssel möglich, daher bitweise *XOR*-Operation mit zufälligem Bitmuster auf Klartext
- Identischer Klartext erzeugt nun anderen Geheimtext, Empfänger benötigt zum Entschlüsseln das zufällige Bitmuster
- Um nicht doppelt so viele Daten (Geheimtext + Zufallsmuster) versenden zu müssen wird *Cipher Block Chaining* angewandt
⇒ Nur das erste zufällige Bitmuster (*Initialization Vector VI*) wird unverschlüsselt an Empfänger gesendet
⇒ Danach ist vorhergehender Geheimtext das zufällige Bitmuster für den nächsten Block Klartext
- Für $c =$ Ausgegebener Geheimtext (*Cipher*), $K =$ Schlüssel, $m =$ Klartext, ist der Verlauf dann wie folgt:
 $c(0) =$ Initialisierungsvektor ⇒ $c(1) = K(m_1 \text{ } XOR \text{ } c(0)) \Rightarrow c(2) = K(m_2 \text{ } XOR \text{ } c(1))$

Data Encryption Standard (DES):

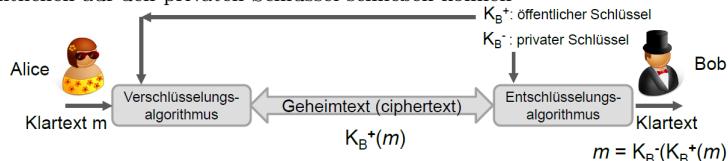
56-Bit symmetrischer Schlüssel, Verarbeitung von 64-Bit Blöcken mit *Cipher Block Chaining*, per *Brute Force* knackbar, *3DES* etwas sicherer

Advanced Encryption Standard (AES):

Nachfolger von *DES*, 128/192 oder 256-Bit symmetrischer Schlüssel mit 128-Bit Blöcken, AES-256 kann nicht geknackt werden

Public Key (asymmetrische) Verschlüsselung:

- Öffentlicher Schlüssel K_B^+ wird zur Verschlüsselung verwendet, Privater Schlüssel K_B^- zur Entschlüsselung
- Öffentlicher Schlüssel sowohl Empfänger als auch Sender bekannt, privater Schlüssel ist aber nur Empfänger bekannt
- Sender überträgt $K_B^+(m)$, Empfänger entschlüsselt mit $K_B^-(K_B^+(m))$, d.h. es muss gelten $K_B^-(K_B^+(m)) = m$
- Man darf nicht vom öffentlichen auf den privaten Schlüssel schließen können



RSA Algorithmus:

- Text wird als Bitmuster angesehen, die Verschlüsselung ergibt wieder ein Bitmuster - den Geheimtext
- RSA nutzt modulare Arithmetik und die Tatsache, dass Teilerbestimmung einer gegebenen Zahl sehr rechenaufwändig sind
- Verschlüsselung mit public key ist allerdings auch sehr rechenaufwändig, da z.B. m^e berechnet wird ($e = \text{öffentlicher Schlüssel}$)
- Symmetrische Verschlüsselung wesentlich schneller \Rightarrow Asymmetrische Verschlüsselung zum Aufbau einer sicheren Verbindung
 - danach mit einem zweiten Schlüssel symmetrische Verschlüsselungen austauschen
 - Teilnehmer A & B verwenden RSA um symmetrischen Schlüssel K_S auszutauschen, danach symmetrische Verschlüsselung (AES) mit K_S

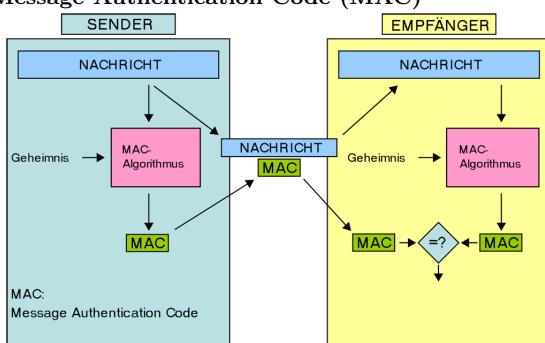
Integrität einer Nachricht sicherstellen

- Berechnen einer Prüfsumme (Hash) über Nachricht + gemeinsames Geheimnis (ansonsten könnte Angreifer die Nachricht verändern und dann erneut eine gültige Prüfsumme über die veränderte Nachricht berechnen), Empfänger berechnet Hash selbst und überprüft ihn mit übertragenem Hash

Kryptographische Hashfunktionen

- Berechnung eines Strings $H(m)$ fester Größe aus Nachricht m , vom Rechenaufwand her nicht möglich, Kollision zu erzeugen, so dass $H(x) = H(y)$
- $MD5$ mit 128-Bit Hash (unsicher) $\circ SHA-1$ mit 160-Bit Hash (inzwischen auch kritisch) $\circ SHA-2$ mit 224/256/384/512-Bit Hash (empfohlen)

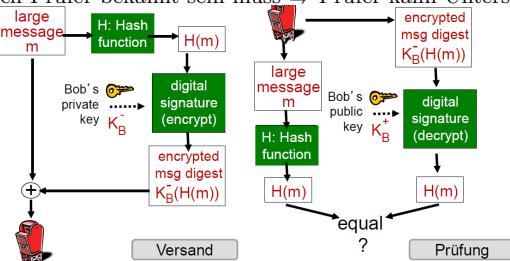
Message Authentication Code (MAC)



- Gemeinsames Geheimnis: *Authentication Code s*, Nachricht m , Sender hängt Hash $H(m + s)$ (= MAC) an Nachricht m an
- Übertragung von Nachricht + MAC an Empfänger, dieser berechnet ebenfalls Hash aus $m + s$ und prüft, ob gesendeter MAC passt
- Zur Prüfung der Integrität einer Nachricht

Digitale Signaturen

- Unterschrift als Bestätigung der Urheberschaft eines Dokuments, MAC mit *shared key s* ungeeignet, da s sowohl Unterzeichner als auch Prüfer bekannt sein muss \Rightarrow Prüfer kann Unterschrift fälschen



Lösung:

- Berechne $H(m)$, $m :=$ zu unterschreibende Nachricht
- Unterzeichner nutzt private key zur Berechnung von $K_B^-(H(m))$, wird zusammen mit m versendet
- Prüfer prüft ob $H(m) = K_B^+(K_B^-(H(m)))$ (entschlüsseln mit öffentlichem Schlüssel), falls ja: Unterschrift gültig

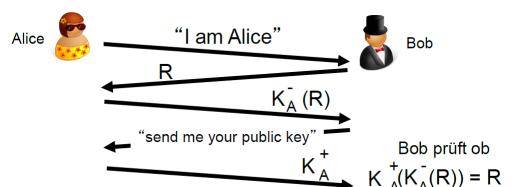
Zertifizierung von öffentlichen Schlüsseln

- Angreifer kann behaupten, dass dessen public key gleich dem vom Unterzeichner ist \Rightarrow Öffentlicher Schlüssel muss Person zugeordnet werden
- Certification Authority (CA)* prüft Identität (z.B. Personalausweis)
 - erstellt Zertifikat, dass public key zur Person gehört (Angabe der Domain) und unterschreibt das Zertifikat per digitale Signatur
- Unterzeichner sendet eigenen public key und Zertifikat an Empfänger, dieser prüft mit öffentlichem CA-Schlüssel, ob Zertifikat gültig (d.h. public key richtig) ist

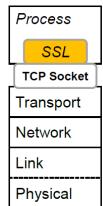
Authentifizierungsprotokoll mit Shared Secret

- Kommunikationsteilnehmer prüft, ob ein anderer derjenige ist, für den er sich ausgibt
 - IP-Adresse kann gefälscht sein oder Angreifer führt Playback-Angriff durch (Kommunikation aufzeichnen & wieder abspielen)
- Lösungen: *shared secret* oder Verwendung einer nur 1x verwendeten, zufälligen Zahl (*Nonce*), die mit symmetrischem Schlüssel verschlüsselt wird

- Sender sendet eigenen Namen an Empfänger
- Empfänger sendet Nonce R zurück
- Sender sendet mit eigenem privaten Schlüssel verschlüsseltes R
- Empfänger fordert public key an
- Sender sendet public key (ggf. mit Zertifikat einer CA) an Empfänger
- Empfänger prüft ob $K_A^+(K_A^-(R)) = R$

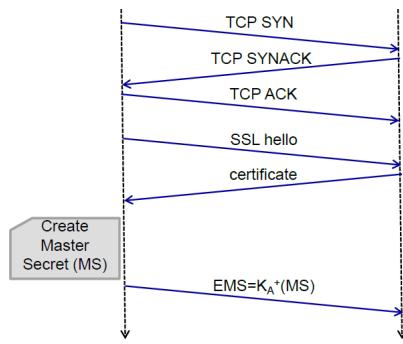


1.3.3 Secure Sockets Layer (SSL)



- Viele TCP-basierte Anwendungen benötigen **Vertraulichkeit, Integrität, Authentifizierung**
- Keines der Internet-Transportprotokolle unterstützt Verschlüsselung übertragener Informationen
- Auf Anwendungsschicht implementiert (z.B. *java.net.ssl* oder *OpenSSL*)
- SSL stellt diese Funktionalitäten durch Verschlüsselung, Message Authentication Code und mit Zertifikaten zur Verfügung
- SSL v3.1 von *IETF* als **Transport Layer Security TLS 1.0** standardisiert
- Drei Phasen: Handshake, Key Derivation (Herleitung von Schlüsseln), Data Transfer (eigentliche Datenübertragung)

SSL Handshake: Webbrower als Client, Server besitzt public & private key + Zertifikat für public key zur Identitätsbestätigung (Domainname)



- TCP-Verbindung aufbauen
- Client sendet Liste an unterstützten *SSL Cipher Suites* + Nonce an Server
- Server sendet Wahl der *Cipher Suite*, CA-Zertifikat und Server-Nonce zurück an Client
- Client:
 - prüft Server-Zertifikat
 - generiert **Pre-Master Secret (PMS)**
 - sendet mit Public Key des Servers verschlüsseltes PMS an den Server
 - Server kann das PMS mit seinem nur ihm bekannten Private Key entschlüsseln
- Client & Server leiten mit *KDF* aus PMS und Nonces das **Master Secret (MS)** her
- Aus MS werden die Schlüssel E_A, E_B, M_A, M_B hergeleitet
 - ⇒ **Nun alle Nachrichten verschlüsselt + MAC authentifiziert**
- Client sendet *MAC* aller Handshake-Nachrichten, die vom Server überprüft werden
- Server sendet *MAC* aller Handshake-Nachrichten, die vom Client überprüft werden
 - ⇒ **Verhinderung von z.B. Wahl einer schwächeren Cipher Suite**

SSL Key Derivation

- Statt *Master Secret* direkt als Schlüssel zu verwenden werden aus Sicherheitsgründen vier unterschiedliche, **symmetrische** Schlüssel aus einer *Key Derivation Function KDF* mit Eingabeparametern Master Secret + Zufallsdaten, abgeleitet:

E_B : Verschlüsselung Client → Server

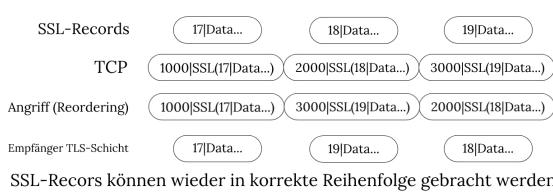
M_B : MAC zur Integritätsprüfung der Daten von Client → Server

E_A : Verschlüsselung Server → Client

M_A : MAC zur Integritätsprüfung der Daten von Server → Client

- Master-Secret und die 4 Schlüssel sind nur für eine Sitzung gültig!

SSL Datenübertragung



- Aufteilung des TCP-Byte-Stroms in separate Abschnitte (*SSL Records*)
- Jeder Record enthält **zusätzliche Sequenznummer** (beginnend bei 0) um Reordering/Replay/Entfernung von Records verhindern
- *MAC* wird über Daten + MAC-Schlüssel (M_A bzw. E_B) + Sequenznummer berechnet
- MAC an Daten anhängen, Verschlüsseln mit E_A bzw. E_B
- Nutzung von *Nonces* als Zufallswert verhindert Replay-Angriff (Wiederabspielen einer Datenverbindung)

SSL Record

| Type | Version | Length | Data | MAC |
|------------------------------------|---------|--------|------|-----|
| verschlüsselt mit E_A bzw. E_B | | | | |

- **Type:** Unterscheidung von Handshake/Nutzdaten/Beenden der Verbindung
- **Version:** SSL-Versionsnummer
- **Length:** Länge der Daten (ohne Header) - Zur Extraktion der Records aus dem TCP-Byte Strom

SSL Cipher Suites

- Enthält Algorithmus für: *Public Key* Verschlüsselung, *symmetrische* Verschlüsselung und für *Message Authentication Code*
- Client bietet Reihe von *SSL Cipher Suites* an (vom Client favorisierte ganz oben), Server sucht eine aus (unabhängig von Clientpriorität)

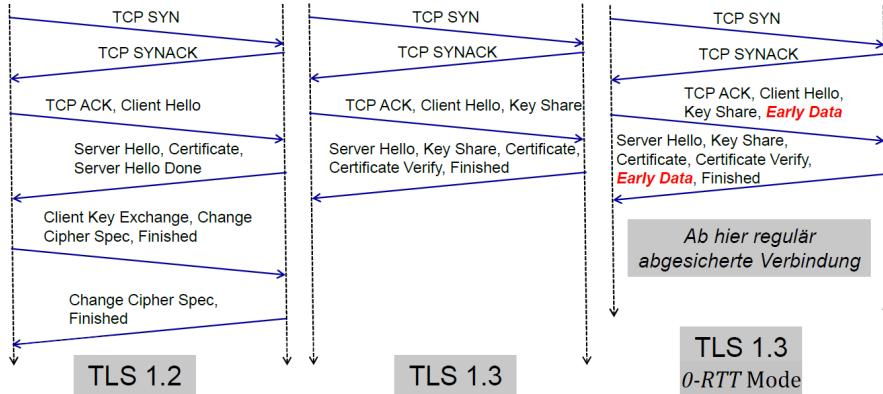
| Symmetrisch | Public Key | MAC |
|----------------|---|-------------------|
| DES, 3DES, AES | RSA, DH (Diffie-Hellmann), ECDH (Elliptic-Curve DH) | SHA-1, SHA-2, MD5 |

Vorzeitiges Schließen einer SSL-Verbindung

- Wird durch zusätzlichen *SSL-Record* zum ordnungsgemäßen Beenden verhindert (auch *MAC*-Authentifiziert)

1.3.4 TLS v1.3

- Ciphersuite-Konzept geändert: MAC berechnen & Verschlüsselung erfolgt nun in einem Schritt
- Verbindungsaufbau (*full handshake*) von 2 RTT auf 1 RTT verringert, *0-RTT-Mode*: Daten bereits in erster Nachricht enthalten
- Verhindern von Downgrades durch Signierung der Liste an Verschlüsselungsverfahren, unsichere kryptographische Verfahren entfallen
- **Perfect Forward Secrecy:** Kommunikationspartner generieren nur für diese Verbindung gültige, frisch zufällig erzeugte Kurzzeitschlüssel, die nach Verbindungsende verworfen werden und nicht aus Langzeitgeheimnis rekonstruiert werden können

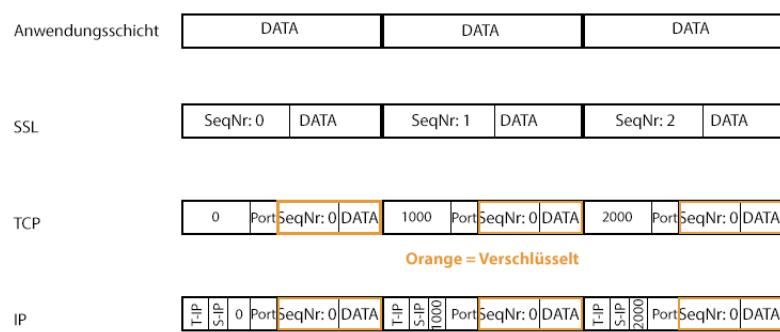


◦ 0-RTT Mode:

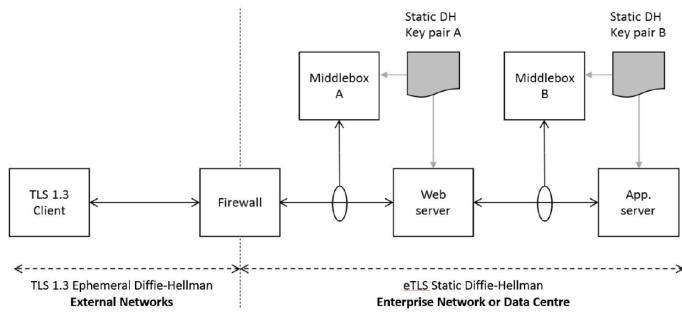
- Client & Server teilen *Pre-shared Key* z.B. aus vorheriger Verbindung, und können diesen gleich zur Authentifizierung und Verschlüsselung der *Early Data* verwenden
- allerdings: Early Data schlechter abgesichert:
 - Kein *Forward Secrecy*,
 - anfälliger ggü. Replay-Attacken

Praktikumsaufgabe 6.1

- Ist es möglich, dass ein Angreifer, ohne die Shared Keys der Teilnehmer zu kennen, ein gefälschtes TCP Segment (mit korrekter TCP Checksumme/Sequenznummer, sowie korrekten IP-Adressen und Portnummern) in eine laufende SSL-Session einschleusen kann?
 - Daten wie Quell- und Ziel-IP, Port und TCP-Sequenznummer sind unverschlüsselt und können vom Angreifer ausgelesen werden, d.h. der Angreifer kann in der Tat solch ein gefälschtes TCP-Segment einschleusen, allerdings hat er keinen Zugriff auf die verschlüsselte SSL-Sequenznummer
- Wird dieses Segment der Anwendung übergeben?
 - Da die Sequenznummer nicht stimmen kann bzw. es allgemein schon an der Verschlüsselung mit richtigen Keys scheitert, wird das TCP-Segment nicht der Anwendung übergeben



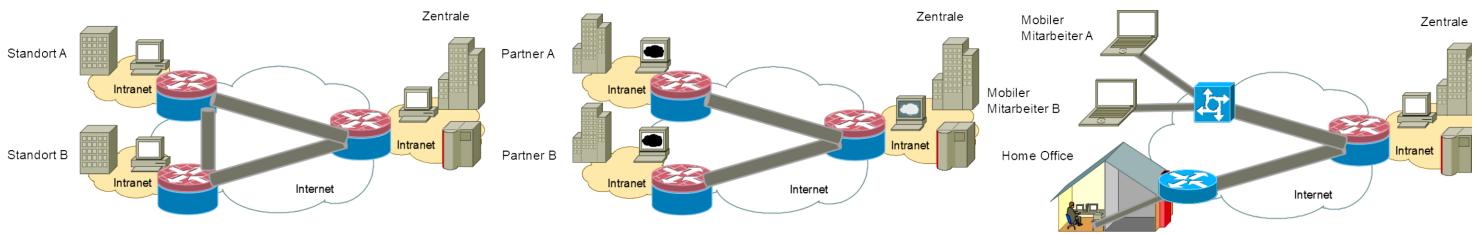
1.3.5 Enterprise Transport Security (ETS)



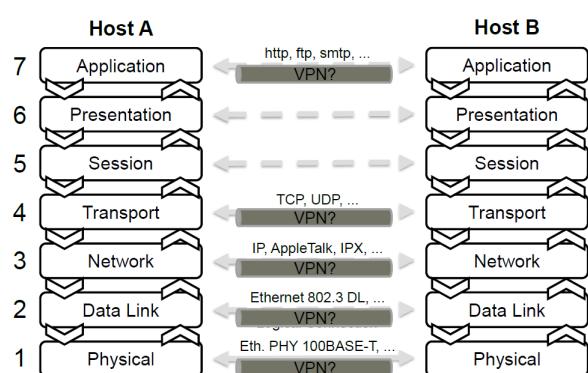
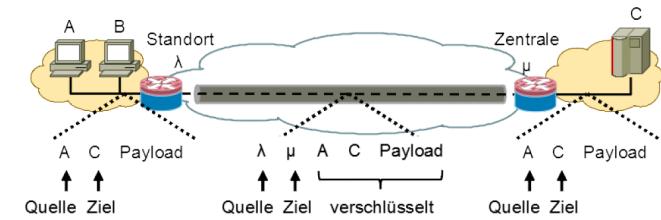
- Unternehmensnetze setzen viele Middleboxes zur Angriffserkennung (*Intrusion Detection*) oder als Application Layer Firewall ein, Datenverkehr kann aufgrund von *Perfect Forward Secrecy* nicht entschlüsselt werden → keine Überwachung durch Middleboxen möglich
⇒ Verzicht auf *Perfect Forward Secrecy*, TLS 1.3 wird um statisches Diffie-Hellmann Schlüsselpaar zwischen Middlebox und Unternehmensserver erweitert

1.3.6 Virtual Private Networks (VPN)

- Stellt über ein öffentliches, ggf. unsicheres, Netzwerk eine gesicherte, private Vernetzung zwischen Endgeräten her
- Zur Anbindung von auswärtigen Standorten, externen Partnern oder Mitarbeitern (*Remote Access*)
- Implementierung auf verschiedenen Schichten möglich (Transportschicht: *Cisco-Client*, Netzwerk-Schicht (IP): *IPSec*)



- Tunneling:** Weiterleitung beliebiger Datenpakete über unsicheres Transitnetz
- VPN nutzt eigenen, unabhängigen Adressraum
 - $A, B, C := \text{VPN-Adressraum}; \lambda, \mu := \text{Internet-Adressraum}$
- Am Anfang und Ende des Tunnels werden die zusätzlichen Header hinzugefügt bzw. wieder entfernt
- Authentisierung** (Sicherstellen der Identität):
 - Password Authentication Protocol PAP* - unsicher da Klartextaustausch
 - Challenge Handshake Authentication Protocol CHAP*
 - Challenge-Response-Verfahren: Nutzer authentisiert sich mit Hash über Passwort und zufällige, vom Server gesendete Challenge
- Software:**
 - Server (VPN-Gateway) per VPN-Server Software
 - Client per VPN-Client (Cisco) / VPN-Adapter
- Protokolle:**
 - Point-to-Point Tunneling Protocol PPTP*
 - Layer Two Tunnelling Protocol L2TP* (PPTP-Nachfolger)
 - Tunnelt beliebige Pakete über UDP
 - Authentifizierung über *PAP/CHAP*



1.3.7 IPSec

- o Dominierendes Protokoll für VPNs auf Netzwerkschicht, verschiedene Verfahren:

- **Authentication Header (AH)**

- Garantiert Integrität der übertragenen Daten, Authentifizierung der Quelle eines Paketes (**keine** Verschlüsselung)

Header:

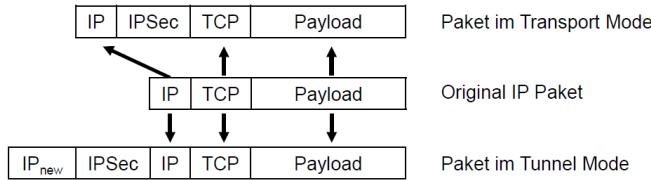
- *Integrity Check Value (ICV)*: HMAC (Hash über Nutzdaten + Geheimnis) über IP-Paket
- *Security Parameter Index (SPI)*: Index auf eine *Security Association* (kommt später)
- *Sequence Number (SN)*: Steigende Nummer zum Schutz vor *Replay-Attacken*

- **Encapsulated Security Payload ESP**

- Vertraulichkeit der Daten durch symmetrische Verschlüsselung + zusätzliche Authentifizierung der Quelle eines Paketes

Header:

- Ebenfalls *ICV* (hier wird äußerer IP-Header **nicht** mit einbezogen), *SPI* und *SN*
- *Initialization Vector (IV)*: Initialisierungsvektor für symmetrische Verschlüsselungsverfahren
- *Encrypted Payload*: Verschlüsselte Nutzdaten



- **Transport Mode:** Einfügen des *IPSec-Headers* in original IP-Paket
⇒ Gesicherte IP-Verbindung zwischen zwei Hosts (*End-to-End*)
- **Tunnel Mode:** Kapselung des Original IP-Pakets (inkl. Header) in neues IP-Paket mit *IPSec-Header*
⇒ Gesicherte Verbindung zwischen Host/Netz und Netz/Netz
⇒ Im neuen IP-Header neue Src- bzw. Dest-Adressen (die Tunnelendpunkte)

Security Association (SA):

- (unidirektionale) Festlegung der für IPSec Kommunikation zwischen zwei IPSec-Hosts verwendeten Parameter: Protokoll, Modus, Verschlüsselung
- *Security Policies (SP)* legen fest, **was in welcher Art** verschlüsselt wird → *Security Policy Database (SPD)*
- Jede SA hat eindeutigen *Security Parameter Index (SPI)*
- IPSec Einheit erhält die *SPIs* in *Security Association Database (SAD)*

Problem mit Network Address Translation (NAT):

- o NAT ändert Port und IP-Adresse (klappt nicht, wenn die im *Tunnel-Mode* verschlüsselt sind)
⇒ *IPSec NAT Traversal*: UDP Datagram zum Übertragen der IPSec Pakete

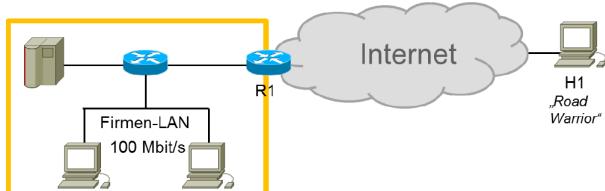
Internet Key Exchange (IKE):

- Manuelles Eintragen der Schlüssel für die SAs bei großer Anzahl IPSec-Verbindungen unpraktikabel
- Automatisiertes Aushandeln der SAs (mit Schlüssel)
- Basiert auf *Security Association Key Management Protocol (ISAKMP)*
- Variante 1: *Pre-Shared Key (PSK)*
 - Beide Seiten sind mit gemeinsamen Geheimnis vorkonfiguriert
 - IKE wird genutzt um SAs aufzusetzen
- Variante 2: *Public Key Infrastruktur (PKI)*
 - Beiden Seiten haben Public- und Private Key + **Zertifikat**
 - IKE zur Authentifizierung und Aufsetzen der SAs (ähnlich SSL Handshake)
- Phase 1: Aufbau einer bi-direktionalen *IKE SA (keine IPSec SA)*
 - Verwendet ISAKMP zwischen den Routern, entweder im *Aggressive Mode* (1.5 RTT) oder *Main Mode* (3 RTT)
- Phase 2: Nutzen von ISAKMP zur Aushandlung von IPSec SA, IPSec-Teilnehmer signieren ihre Nachrichten

1.3.8 Alternative VPN Protokolle

- o **OpenVPN:** per OpenSSL verschlüsselte *TLS*-Verbindung (TCP/UDP), *Bridge-Mode*: Layer 2 Tunnel, *Routing-Mode*: IP-Tunnel
- o **Port Forwarding / Tunnelling mit SSH:** Tunnelling von TCP-Verbindungen über *SSH* (Anwendungsschicht) - kein echtes VPN

Praktikumsaufgabe 7.1

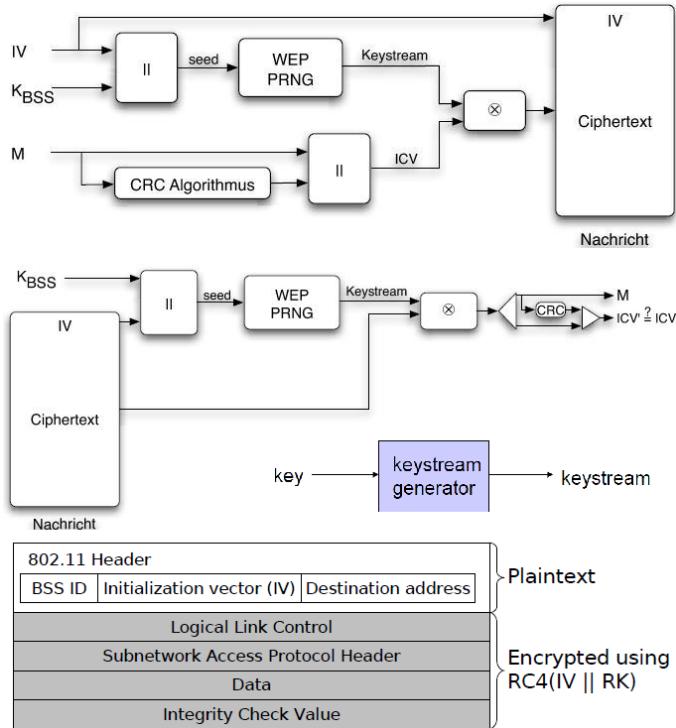


- Firmennetz ist über R_1 mit Internet verbunden,
 H_1 soll von Unterwegs Zugriff aufs Firmennetz bekommen,
Absicherung der Vertraulichkeit und Integrität der Daten per **IPSec**
- Welches Verfahren (*AH* oder *ESP*)?
- *ESP*, *AH* dient nur der Authentifizierung,
danach werden **keine Daten verschlüsselt**

- Einsetzen des *Tunnel-Modus* - wo beginnt und endet der Tunnel?
- Tunnel geht von R_1 bis H_1
- Ist es möglich, dem Rechner H_1 eine IP-Adresse aus dem Firmen-Adressbereich zuzuteilen?
- Ja, im Tunnel-Modus wird vor das IP-Paket, das die tatsächlichen IP-Adressen enthält, ein IPSec-Header mit Quell- und Zieladressen der Tunnelpunkte, geschaltet. Im Original-IP-Header kann H_1 mit einer firmeninternen IP adressiert werden.
- Der Server sendet ein IP-Datagramm an H_1 - welche IP-Adressen sind im Header des IP-Paketes als Quelle und Ziel eingetragen?
- **Quelle:** Server-IP, **Ziel:** IP von H_1 intern
- Unter welchen Umständen kann man den Inhalt des Paketes im Klartext mitlesen?
- Verschlüsselung + IPSec-Header geschieht erst bei R_1 , davor ist das Paket unverschlüsselt
- IP-Datagramm wird nun bei R_1 ins Internet weitergeleitet - ändern sich Quell- und Zieladressen des Datagramms?
- IP-Datagramm wird als Payload in neues IP-Datagramm mit IPSec-Header gepackt. Dieses enthält als Quell-Adresse den Tunnelpunkt R_1 und als Ziel-Adresse den Tunnelpunkt H_1

1.3.9 Sichere WLANs

802.11: Wired Equivalent Privacy (WEP)



Probleme mit *Shared Key Authentication* bei WEP:

- Angreifer kann IV , $Nonce$ und $Antwort$ beobachten, daraus kann er gültigen Schlüsselstrom und IV erzeugen:
 $\text{Nonce XOR } (\text{Nonce XOR keystream}) = \text{keystream} \Rightarrow$ Angreifer kann gültige Authentifizierungsnachrichten für beliebige $Nonce$ erstellen
 \Rightarrow **Unsicherer** als *Open System*
 - **Problem 1:** Nur 2^{24} verschiedene Schlüssel (IV -Länge) $\Rightarrow IV$ wird aufgrund heutiger Datenraten in weniger als einem Tag erneut verwendet
Angreifer kann selbst Datenverkehr generieren (z.B. ARP-Requests, da fixe Länge & Inhalt bekannt) und Schlüsselstrom mitschneiden
 - **Problem 2:** IV wird im Klartext übertragen \Rightarrow Angreifer kann Wiederholung leicht erkennen

Beispiel: Known Plaintext Angriff

- Angreifer motiviert AP großteils bekannten Klartext zu verschlüsseln (z.B: ARP-Request oder Übertragung einer im Klartext bekannten Datei)
 - Angreifer kennt Klartext $m(i)$ und kann Schlüsselstrom aufzeichnen: $ks(i) = c(i) \text{ XOR } m(i)$
 - Sobald Wiederholung des IV detektiert wird: Abfolge des Schlüsselstroms bekannt und Entschlüsselung möglich

802.11i: Temporal Key Integrity Protocol (TKIP)

- Ersatz für *WEP*, aber mit selber Hardware funktionsfähig, *IV* erweitert auf 48-Bit, neue Hash-Funktion, *Personal-Mode* (*WPA-PSK*) mit pre-shared key, *WPA Enterprise* auf Basis von *EAP*
 - Generiert dynamischen, individuellen Schlüssel für jeden Nutzer - wird außerdem periodisch erneuert

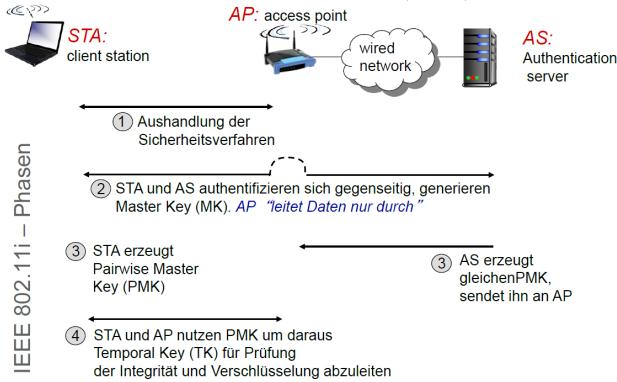
802.11i: Counter Mode with Cipher Block Chaining Message Authentication Code Protocol (CCMP)

- Verwendet *AES* mit 128-Bit Schlüssel und 48Bit *IV* (anstatt *RC4*)
 - Teil des *WPA2*-Verfahrens, *WPA2* würde alternativ auch *TKIP* erlauben (**empfohlen ist allerdings *WPA2 + CCMP***)

802.11i: WPA3

- 128-Bit (*personal mode*) bzw. 192-Bit (*enterprise mode*) + *Forward Secrecy*

Extensible Authentication Protocol (EAP)



- Protokoll für sichere Ende-zu-Ende Authentifizierung zwischen mobilen Clients (**STA**) und Authentifizierungsserver (**AS**) über einen Access Point
- *EAP-TLS*: Nutzung von Zertifikaten
- *EAP-PEAP*: Verschlüsselter Tunnel für Authentifizierung

802.1X

- Basiert auf *EAP*, Erweitert *802.11* um Sicherheitsfunktionen wie: Schlüsselmanagement, Nutzeridentifikation, Tokens...
- Client als *Supplicant*, der Zugriff auf 802.1X-Dienste von *AS* anfordert

Zusammenfassung

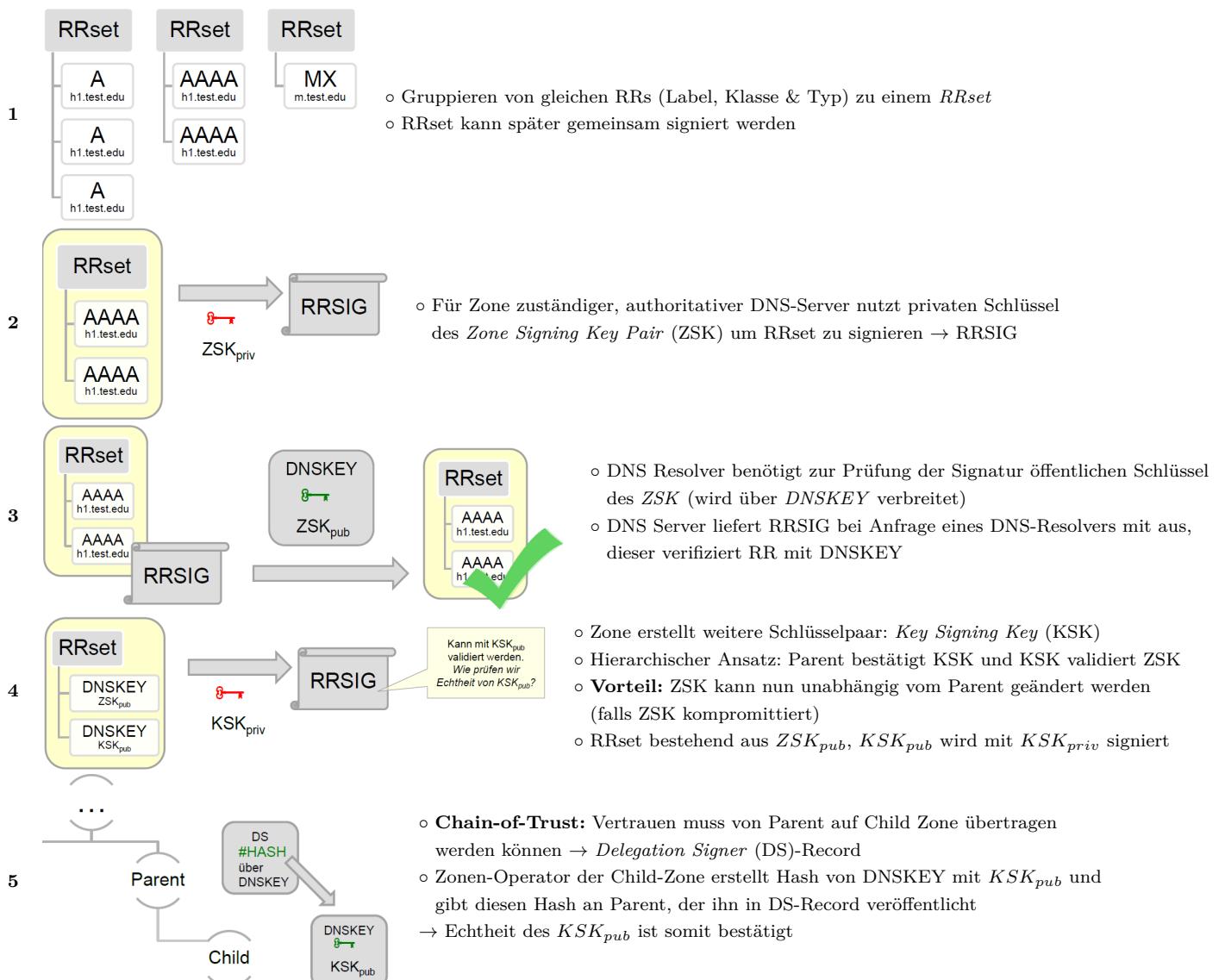
- Zusätzliche Herausforderungen ggü. drahtgebundenen Netzen: Geteiltes Medium (*shared broadcast medium* & Mobilität der Nutzer)
- Ursprünglich mit *802.11* definierter Standard WEP hat gravierende Sicherheitsdefizite
- *802.11i* definiert neue Sicherheitsansätze, WPA2 CCMP wird empfohlen
- Heimnetze ohne zentrales Sicherheitskonzept: WPA2-PSK, Firmennetze: 802.1X/EAP bzw. EAP-TLS

1.3.10 DNS Security Extensions (DNSSEC)

- Absicherung von Authentizität und Integrität der DNS-Informationen (**nicht** Vertraulichkeit)
 - ⇒ Verwendung von Signaturen, umgesetzt mit asymmetrischen Kryptosystem (public & private Key):
 - *Resource Record* (RR) wird mit geheimen Schlüssel unterschrieben
 - DNS Client prüft mit öffentlichem Schlüssel die Authentizität und Integrität des empfangenen *RRs*
 - *Chain-of-Trust*: Höhere Zone im DNS-Baum bestätigt durch Signierung die öffentlichen Schlüssel der unteren Zonen,
→ der Anfragende muss **nur** obersten public key kennen (*Key Signing Key*)

○ Realisierung über neue Resource Record Typen auf den DNS-Servern:

| RR Typ | Erläuterung | |
|--------------|---|--|
| RRSIG | Digitale Signatur eines <i>Resource Record Sets</i> (RRset) | |
| DNSKEY | Public Signing Key, wird vom Resolver zur Prüfung digitaler Signaturen verwendet | |
| DS | Delegation Signer: Hash des <i>DNSKEY</i> um dessen Authentizität sicherzustellen, delegiert Vertrauen eine Ebene weiter in Chain-of-Trust | |
| CDNSKEY, CDS | Child Zone informiert Parent: Key-Signing-Key ändern, DNSSEC an/aus | |
| NSEC, NSEC3 | Ring-Verkettung aller DNS-Einträge einer Zone in alph. Reihenfolge Angreifer kann keine Einträge hinzufügen/entfernen (bei NSEC3 als Hash) | |



1.3.11 DNS over TLS

- Konfiguration des DNS-Client (*DNS Stub Resolver*): IP-Adresse/Name + Port des verwendeten Resolvers und **Hash des öffentlichen Schlüssels (SPKI)** des Resolvers zur Zertifikatsprüfung
- Absicherung von Vertraulichkeit
 1. DNS Client (DNS Stub Resolver) baut TCP-Verbindung zu Resolver auf
 2. TLS Handshake wird durchgeführt (*DNS Stub Resolver* = DNS Client, *DNS Resolver* = Server)
 - Resolver sendet TLS Zertifikat
 - DNS Stub Resolver prüft Zertifikat mit Hash des öffentlichen Schlüssels (*SPKI*) des Resolvers
 3. Ab jetzt sendet DNS Stub Resolver alle DNS Anfragen über mit TLS geschützte TCP-Verbindung
 - Übertragung einer Nachricht: 2-Byte Längenangabe + Nachricht
 - beim Zusammensetzen muss man wissen, wie lange die ursprüngliche Nachricht war, wenn sie in einzelne TCP-Pakete zerstückelt wurde
 4. Verbindung bleibt offen bis DNS Stub Resolver sie schließt
 5. Umsetzung: Als Betriebssystem-Dienst (z.B. *systemd-resolved* unter Linux) oder lokaler Forwarder der als localhost-DNS-Server im Betriebssystem eingetragen wird und Anfragen über TLS weiterleitet

1.3.12 DNS over HTTPS

Anfrage über HTTP Methode GET

```
GET /dns-query?dns=AAABAAAAAABAAAAAAA3d3dw1eGFtcGx1A2NvbQAAAQAB HTTP/2
Accept: application/dns-message
```

Anfrage über HTTP Methode POST

```
POST /dns-query HTTP/2
Accept: application/dns-message
Content-Type: application/dns-message
Content-Length: 33
```

<33 bytes represented by the following hex encoding>
 00 00 01 00 00 01 00 00 00 00 00 03 77 77 77
 07 65 78 61 6d 70 6c 65 03 63 6f 6d 00 00 01 00
 01

base64 kodierte DNS Query

DNS Query (33 Byte)

- Absicherung von Vertraulichkeit
- **Per POST:** Anfrage im HTTP-Body
- **Per GET:** Anfrage in angegebener Ressource
- Probleme: Genutzter DNS-Server ist Anwendungsspezifisch (z.B. Browser), firmeneigene DNS-Server werden umgangen
- Internet Web-/Maildienste funktionieren nicht mehr (werden nur vom internen DNS aufgelöst)
- DNS-Filter gegen Malware unwirksam
- Nur wenige Firmen (Cloudflare, Google) bieten DNS-over-HTTPS → Konzentration auf wenige Anbieter, Gefährdung Privatsphäre
- Viel Overhead (Anwendungsschicht: DNS, HTTP, TLS Transportschicht: TCP statt UDP)
- Blockieren schwierig da Port 443 HTTPS Standardport

Discovery des Resolvers

- Special-Use-Domain
 - Anwendung (z.B. Browser) fragt Standard-Resolver nach TXT-Record
 - Resolver liefert Liste der DNS-over-HTTP-Server (vom Netz-Admin zu konfigurieren)
 - Anwendung übernimmt (ggf. firmeninternen) DNS-over-HTTP-Server
- Well-Known-DoH-Server
 - Unter der IP des Standard-Resolvers läuft Webserver, der Liste an DNS-over-HTTP-Server ausliefert
 - Anwendung (z.B. Browser) fragt diese URI per HTTPS an
 - Anwendung übernimmt (ggf. firmeninternen) DNS-over-HTTP-Server

| DNS-over-TLS (DoT) | DNS-over-HTTPS (DoH) |
|---|---|
| DNS Nachrichten direkt über TLS-gesicherte TCP Verbindung ausgetauscht: Overhead gering (2 Byte Längenangabe) | DNS Nachrichten über HTTP Methode GET oder POST geschickt, höherer Overhead durch HTTP Nachricht und Header |
| DNS (Stub) Resolver vom Betriebssystem → durch Admin konfiguriert | DNS Resolver in Anwendung (Browser) konfiguriert – sofern kein Discovery-Ansatz verwendet wird |
| Üblicherweise TCP Verbindung über Port 853 – leicht zu in Firewall zu blockieren | Nutzung von Port 443 (wie reguläre https Anfrage) – schwer in Firewall zu blockieren |

1.3.13 Firewall

- Pakete mit bestimmten Ziel-Ports dürfen passieren, andere werden geblockt, z.B:
 - *Denial of Service* (DoS) Attacken von außen abwehren
 - Zugriff auf interne Dienste blockieren
 - Zugriff aus Firmennetz auf Internetdienste beschränken
- Typen:
 - **Stateless Packet Filters:** Entscheidung ob durchlassen oder verworfen wird pro Paket einzeln getroffen, basiert auf Quell- oder Zieladresse, Quell- oder Zielport, Protokolltyp, TCP SYN/ACK
 - **Stateful Packet Filters:** Verfolgt Status der Protokolle (z.B. TCP), nur zum Status passende Pakete werden durchgelassen
 - **Proxy Firewall / Application Layer Firewall / Deep Packet Inspection (DPI):** Filtern Pakete basierend auf Applikationsdaten & IP/TCP/UDP-Header, ermöglicht Blockieren bestimmter Applikationen für bestimmte Nutzer
 - Konkret **Proxy Firewall:** Tunnelt interne Anfrage über separate Verbindung nach draußen und inspiert Pakete
 - Konkret **Deep-Packet-Inspection:** Lässt Verbindung laufen und inspiert gleichzeitig Paketinhalt

| Action | Source-Addr | Dest-Addr | Protocol | Source-Port | Dest-Port | Flag |
|--------|-------------------|-------------------|----------|-------------|-----------|------|
| allow | 222.22/16 | outside 222.22/16 | TCP | >1023 | 80 | any |
| allow | outside 222.22/16 | 222.22/16 | TCP | 80 | >1023 | ACK |

Stateless Firewall

| Action | Source-Addr | Dest-Addr | Protocol | Source-Port | Dest-Port | Flag | Check Conn. |
|--------|-------------------|-------------------|----------|-------------|-----------|------|--|
| allow | 222.22/16 | outside 222.22/16 | TCP | >1023 | 80 | any | Prüfung Verbindungsstatus erforderlich |
| allow | outside 222.22/16 | 222.22/16 | TCP | 80 | >1023 | ACK | × |

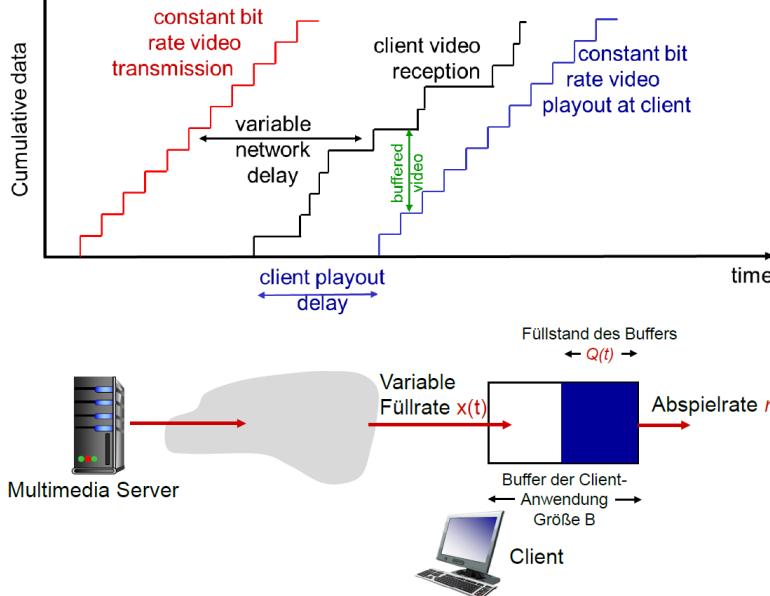
1.3.14 Intrusion Detection Systems (*IDS*)

- Erkennt Angriffe von außen und ermöglicht (automatisierte) Reaktion
- Überprüfen des Paketinhalts auf verdächtige Inhalte (z.B. *SQL Injection*) sowie Korrelation zwischen mehreren Paketen erkennen (*DoS, Port Scanning*)

Multimedia

- Verbreitung von Multimedia-Daten (Video-Streaming, Internet-TV, VoIP) mit hohen Datenmengen & so geringer Verzögerungen wie möglich
- Streaming aufgezeichneter Inhalte:** Da gespeicherte Daten kann der Nutzer vor- & zurückspulen
Kontinuierliche Ausgabe, Vermeidung von Wartezeiten durch Streaming
- Streaming von Live-Audio und Video:** Inhalte liegen nicht vorab vor (**kein** Vorspulen)
Viele Clients sehen gleiches Programm, eventuell Verteilung per *Multicast*
- Interaktives Audio und Video:** Verzögerungen > 150ms störend → kaum Vorpuffern möglich

1.4.1 Grundlagen

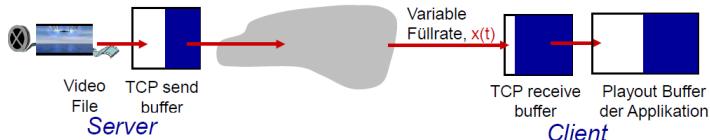


- Puffern und verzögertes Abspielen (**playout buffer/delay**)
Kompensation von Verzögerungen (*network delay*) und Jitter (Schwankungen in der Verzögerung)
- Buffern:
 - Initiales Befüllen des Buffers bis zum Start des Abspielens (t_p)
 - Füllstand $Q(t)$ des Buffers ändert sich über Zeit aufgrund variabler Füllrate $x(t)$
 - mittlere Füllrate $\bar{x} < r$: Buffer leert sich
⇒ Video stockt, Rebuffering
 - mittlere Füllrate $\bar{x} > r$: Buffer wird nie komplett leer solange B ausreichend groß um für Schwankungen von $x(t)$ abzufangen
- Großes B bedeutet größere Startverzögerung

Streaming über UDP:

- Prinzipiell UDP besser als TCP da geringere Verzögerung und Multicast möglich
- Sender:** Erzeugt UDP-Pakete mit Datenrate des kodierten Multimedia-Datenstroms
- Client:** kleiner Buffer (2-5s), Applikationsschicht ist zuständig für Fehlerbehebung
- Aber:** UDP geht oft nicht durch Firewalls/NAT, separate Kontrollverbindung für Stop/Pause über TCP (z.B. RTSP), Reaktion auf Änderung der verfügbaren Bandbreite

Streaming über HTTP (TCP):

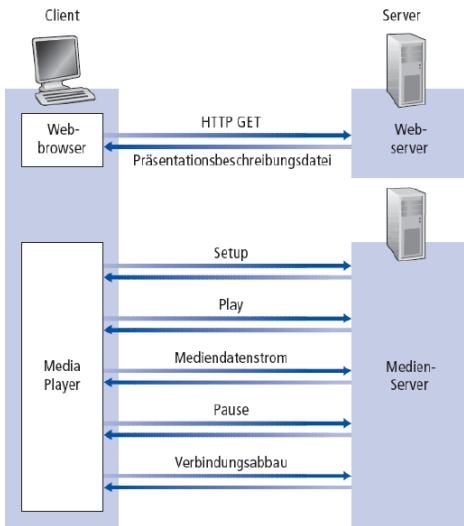


- Abruf der Multimedia-Datei per *GET*
- Sendrate variiert (aufgrund TCP *Congestion Control, Flow Control* und *Retransmissions*)
- HTTP/TCP passiert Firewalls meist ohne Probleme

1.4.2 Dynamic Adaptive Streaming over HTTP (DASH)

- Problem bei HTTP+TCP-Streaming: Bandbreite variiert abhängig von Netzanbindung
- DASH:**
 - Aufteilen der Multimedia-Datei in *Chunks*
 - Mehrfache Kodierung jedes Chunks in unterschiedlichen Versionen mit anderer Bitrate (→ Qualitätsstufen)
 - Manifest-Datei: URLs der unterschiedlichen Versionen, wird von Media-Server an Client geschickt
 - Client wählt passende Version in der Manifest-Datei abhängig von Bandbreite aus
- Intelligenz beim Client:
 - Misst periodisch Datenrate & fordert passend kodierten *Chunk* an (Reaktion auf Bandbreiten-Schwankungen)
 - Bestimmt, wann welcher Chunk angefordert wird (*Buffer-Handling* demnach auch beim Client)
 - Bestimmt, von wo Chunk angefordert wird (z.B. der am besten erreichbare Server)

1.4.3 Real-Time Streaming Protocol (RTSP)



- Client-Server Protokoll auf Applikationsschicht zur Steuerung des Multimedia-Datenstroms (*Play, Pause, Fast-Forward*)
 - Hat separate Kontrollverbindung (Port 554)
 - Mediendaten selbst werden per UDP oder TCP übertragen
 - RTSP spezifiziert nicht:** Kompression von Audio/Video, wie Übertragen wird (definiert durch *RTP*), Pufferverhalten des Players
 - Präsentationsbeschreibungsdatal:**
- ```
< title > Twister < /title >< session >< grouplanguage = enlipsync >
< switch >< tracktype = audiosrc = 'rtsp : //audio.example.com/audio.en/lofi' >
< tracktype = audiosrc = 'rts : //audio.example.com/audio.en/hifi' >< /switch >
< tracktype = 'video/jpeg' src = "rtsp : //video.example.com/twister/video" >
< /group >< /session >
```

### 1.4.4 Content Distribution Networks (CDNs)

| Alternative 1:<br>Ein großer Server                                                          | Alternative 2:<br>Content Distribution Network                                                                                                     |
|----------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>Fehleranfällig („single point of failure“)</li> </ul> | <ul style="list-style-type: none"> <li>Kopien der Inhalte auf mehreren geographisch verteilten („Replica-“)Servern</li> </ul>                      |
| <ul style="list-style-type: none"> <li>Gefahr der Netzüberlast vor Ort</li> </ul>            | <ul style="list-style-type: none"> <li>Privates CDN: Inhaltsanbieter betreibt CDN</li> </ul>                                                       |
| <ul style="list-style-type: none"> <li>Lange Wege zu entfernten Nutzern</li> </ul>           | <ul style="list-style-type: none"> <li>Third-Party CDN: Anderer Anbieter betreibt CDN<br/>Beispiel-Anbieter: Akamai, Amazon, NTT Europe</li> </ul> |
| <b>NICHT SKALIERBAR</b>                                                                      |                                                                                                                                                    |

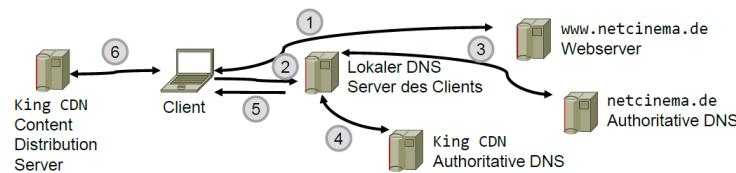
#### o Enter Deep:

- Große Anzahl Cluster innerhalb der Zugangsnetze der *ISPs*
- Nah beim Endnutzer (geringe Latenz, wenig Hops)
- Verfolgt Akamai (> 1700 Cluster an verschiedenen Orten)

#### o Bring Home:

- Geringere Anzahl Cluster, verbunden über privates Hochgeschwindigkeitsnetz
- Nahe der *Points-of-Presence* von *Tier-1 ISPs*
- Leichter zu verwalten aber etwas mehr Hops zum Nutzer

- (1) Nutzer surft auf [www.netcinema.de](http://www.netcinema.de)
- (2) Klickt auf Video [video.netcinema.de/4711abc](http://video.netcinema.de/4711abc)  
→ Client sendet *DNS-Request* für [video.netcinema.de](http://video.netcinema.de)
- (3) Lokaler DNS leitet Anfrage an für [netcinema.de](http://netcinema.de) zuständigen DNS  
→ erkennt Video-Anfrage (nutzt *CDN*)  
→ liefert keine IP, sondern Hostname aus CDN zurück  
(z.B. [a42.kingcdn.de](http://a42.kingcdn.de))
- (4) Lokaler DNS sendet Anfrage nach [a42.kingcdn.de](http://a42.kingcdn.de) an dafür zuständigen DNS-Server (bereits Teil des *CDN*)  
→ Entscheidung, welcher Content-Distribution Server liefern soll  
→→ liefert IP des Distribution Server nahe dem User
- (5) Lokaler DNS leitet IP an Client weiter
- (6) Client ruft Video per HTTP GET ab  
o mit *DASH* liefert Server *Manifest*-Datei, Client wählt dynamisch aus
- CDNs auch per HTTP-Weiterleitung möglich, bei DNS-Ansatz sieht Client aber die verborgene DNS-Struktur nicht!**
- Passender *Content Distribution Cluster* bestimmen durch: Geographische Lage, geringste Verzögerung (*Hops*), *IP-Anycast* (Server mit kürzester Route antwortet), oder Client ermittelt den Besten via *ping*-Anfragen
- Kankan* in China wählt Peer-to-Peer (ähnlich *BitTorent*) Ansatz für CDN (Hash-Tabelle zum Auffinden der Inhalte)



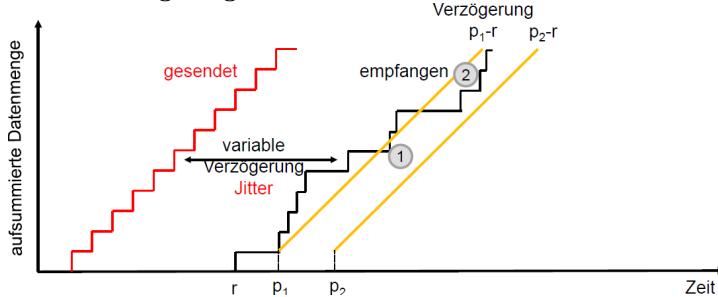
### 1.4.5 Voice over IP (VoIP)

- Sprachdigitalisierung mit 8000 Bytes/s (64 kbit/s) (*Nyquist-Shannon-Abtasttheorem* ⇒ Rekonstruktion von max. 4kHz Frequenzen)  
*Abtastung:* Alle 8000Hz eine Messung, *Quantisierung:* Analoger Messwert in 8-bit Digitalwert überführen
- Digitalisierte Sprache per UDP von 160Bytes alle 20ms senden

- Herausforderungen:

- Paketverluste:** UDP Segmente gehen verloren (1 - 20% je nach Kodierung vertretbar)
- Ende-zu-Ende Verzögerung:** < 150ms: nicht störend, > 400ms: sehr störend, Teilnehmer *fallen sich ins Wort*
- Jitter:** Pakete haben unterschiedliche Verzögerung (Empfänger kann nicht jedes empfangene sofort abspielen)
  - Zeitstempel vor jedem gesendeten Block (zu 160 Bytes) einfügen
  - Feste- oder adaptive Verzögerung des Abspielens (*playout delay*)

**Feste Verzögerung:**



Nur geringe Verzögerung  $q = p_1 - r$ : kurzer Aussetzer bei (1), langer Aussetzer bei (2)

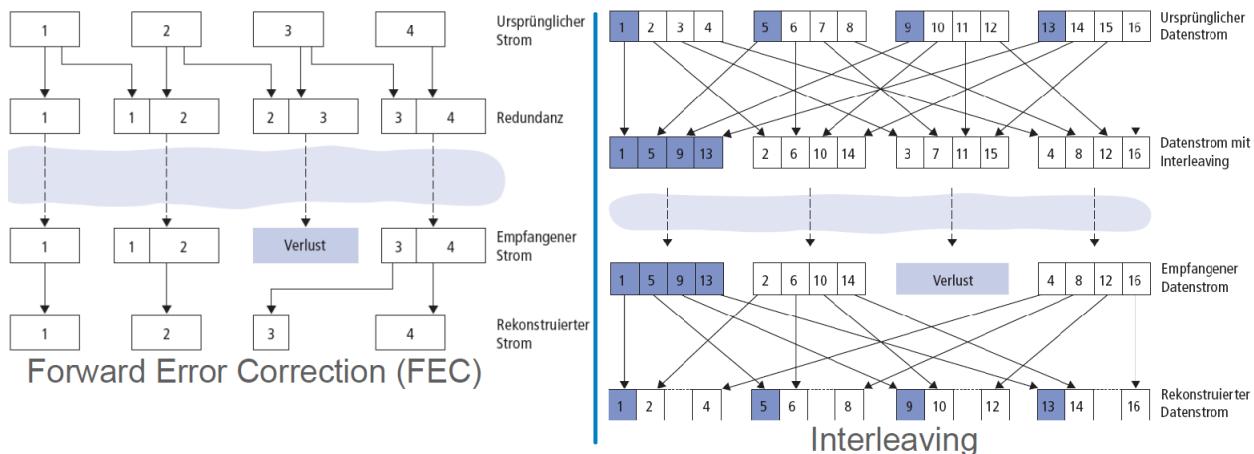
Bei größerer Verzögerung  $q = p_2 - r$ : kein Aussetzer

**Abspielverzögerung  $q$**

- Empfänger spielt Daten mit Verzögerung  $q$  ab (Zeit  $t \rightarrow$  Abspielen bei  $t + q$ )
- Paketverlust wenn Paket nach  $t + q$  ankommt aufgrund zu großer Verzögerung
- Tradeoff:** Großes  $q$ : wenig Paketverlust, Kleines  $q$ : interaktiver, weniger Verzögerung

**Adaption der Abspielverzögerung**

- Anpassen der Abspielverzögerung bei Perioden der Stille im Gespräch (Schätzung der Verzögerung mit Hilfe der Zeitstempel)
  - Exponentiell gewichteter, gleitender Mittelwert:  $d_i = (1 - \alpha)d_{i-1} + \alpha(r_i - t_i)$  mit
  - $d_i$  := Schätzung nach Paket  $i$ ,  $\alpha$  := Konstante  $< 1$ ,  $r_i$  := Empfangszeitpunkt Paket  $i$ ,  $t_i$  := Zeitstempel Paket  $i$  (Sendezzeitpunkt)
  - Standardabweichung  $v_i = (1 - \beta)v_{i-1} + \beta \cdot |r_i - t_i - d_i|$
  - zu Beginn der Redeperiode Berechnung des Abspielzeitpunktes  $p_i = t_i + d_i + K \cdot v_i$  mit  $K$  := positive Ganzzahl
  - ⇒ dynamische Adaption der Verzögerung (nur so lange verzögern, wie nötig)



- Forward Error Correction:** Einfügen redundanter Informationen in Nutzdaten, zur Korrektur von Fehlern/Verlusten
- Interleaving:** Umordnen & Verzähnen des Datenstroms, Verlust eines Paketes führt zu vielen kleinen (tolerierbaren) Lücken und nicht zu einer großen (auffälligen) Lücke
- Verschleierung (Error Concealment):** Empfänger generiert Audio-Daten zum Füllen der Lücke (*Interpolation*, Wiederholung des vorherigen Paketes)

#### 1.4.6 Real-Time Protocol (RPT)

- Standardisiertes Paketformat für Multimedia/VoIP
  - Unabhängig von Art der übertragenen Daten (MP3, H.263, etc.)
  - RTP Payload wird über UDP-Segmente übertragen (Unicast & Multicast)
  - Vorteile als Entwickler sind vorhandene Bibliotheken/Tools und Zusammenarbeit unterschiedlicher RTP-Anwendungen
  - *Header* (12-Byte)

### Zusammenarbeit unterschiedlicher RTP-Anwendungen

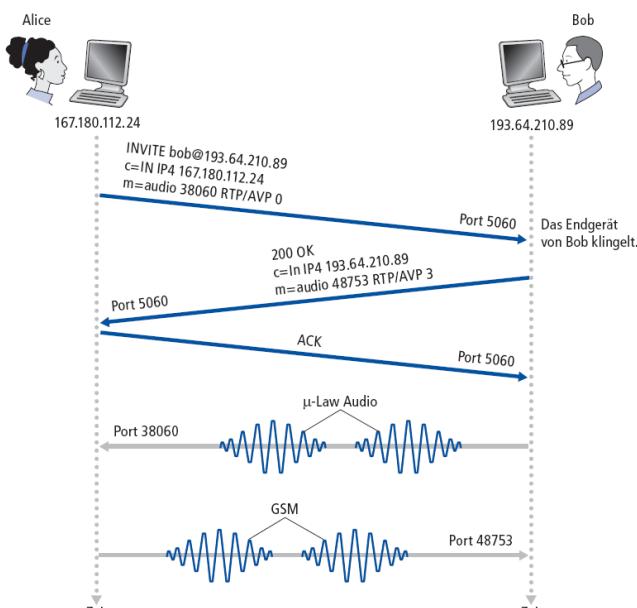
- **Header** (12-Byte)
    - **RTP-Version** mit 2 Bit
    - **Padding-Bit** gibt an, ob hinter Nutzdaten noch Padding eingefügt ist
    - **Extension-Bit** gibt an, ob hinter RTP-Header noch Extension-Header folgt
    - **CSSl-Count**: Anzahl *CSSl*-Werte
    - **Marker-Bit**: Art des Payloads
    - **Payload-Type**: PCM = 0; GSM = 3; H.261 = 31; MPEG2 = 33...
    - **Sequence Number** wird für jedes gesendete RTP-Paket erhöht  
(Empfänger erkennt Reordering/Verluste)
    - **Timestamp** gibt Zeitpunkt der Aufnahme des 1. Bytes an
    - **Synch. Source Identifier** ist eindeutiger Zufallswert, der Quelle der Daten identifiziert
    - **Contr. Synch. Source Identifiers (CSSI)** ist Angabe der Quellen

|              |             |             |            |   |               |                                                                      |
|--------------|-------------|-------------|------------|---|---------------|----------------------------------------------------------------------|
|              |             |             |            |   |               | 32 bit                                                               |
| Ver.<br>2 b. | P<br>A<br>D | E<br>X<br>T | CC<br>4 b. | M | PType<br>7 b. | Sequence Number                                                      |
|              |             |             |            |   |               | Timestamp                                                            |
|              |             |             |            |   |               | Synchronization Source Identifier                                    |
|              |             |             |            |   |               | Contributing Synchronization Source Identifiers (CSSI)<br>(optional) |
|              |             |             |            |   |               | Payload<br>(length varies)                                           |

- ### ○ Beispiel 64kbit/s PCM-kodierte Sprache:

- Abtastung Sprachsignal mit 8kHz ( $\rightarrow$  alle  $125\mu s$  ein 8-Bit Wert), Anwendung sammelt kodierte Audiodaten in *Chunks* (z.B. alle 20ms einen Chunk zu 160 Bytes)
  - RTP-Header: **Payload-Type=0; Timestamp=0** (für jedes Paket um 160 erhöht (Einheit ist Sampling-Periode, hier also  $125\mu s$ ))
  - Der Header bildet zusammen mit Chunk das RTP-Paket, das als UDP-Segment verpackt und versendet wird.

#### 1.4.7 Session Initiation Protocol (SIP)



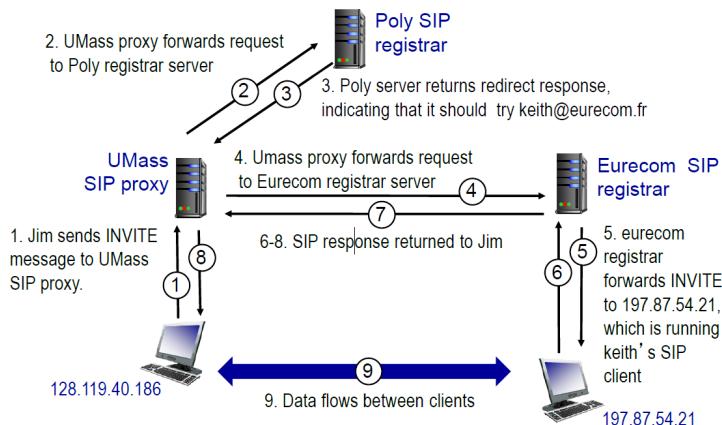
- Mechanismus zum **Auf-/Abbau von Kommunikationssitzungen**
    - Benachrichtigen des Angerufenen und Aushandeln der Kodierung der Multimedia-Inhalte
  - Mechanismus zur **Feststellung der IP-Adresse** des Angerufenen
    - Dynamisch per DHCP oder Anrufer nutzt unterschiedliche Geräte mit unterschiedlichen IPs
  - Mechanismus zum **Verbindungsmanagement**
    - Änderungen der Kodierung, Hinzufügen neuer Multimedia-Streams und neuer Nutzer, Anrufweiterleitung
  - Verbindungsaufbau:
    - **INVITE-Message** per *well-known* Port 5060 über UDP oder TCP
      - Enthält Informationen des Anrufers und des Angerufenen, zur gewünschten Kodierung und zum gewünschten Port
    - **SIP-Response** enthält Status-Code, Kodierung und Port
    - **SIP-Acknowledgement**
      - Anschließend Audio-Verbindung über RTP-Protokoll über separate Verbindung zwischen ausgehandelten Ports (willkürlich 38060 und 48753)
  - Syntax ähnlich HTTP - menschenlesbar, mit Ressource, Version, Methode
  - Header wird durch Leerzeile beendet, danach folgt Content

INVITE sip:08413708048@fritz.box:5060 SIP/2.0  
Via: SIP/2.0/UDP 192.168.56.1:52953  
From: sip:623@fritz.box;tag=aab36c6cb0dbbc46898c473859b  
To: sip:08413708048@fritz.box  
Call-ID: c92f5ab1b9ed4a25bc450368f60bd240  
CSeq: 16918 INVITE  
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, (...)  
Content-Type: application/sdp  
Content-Length: 459

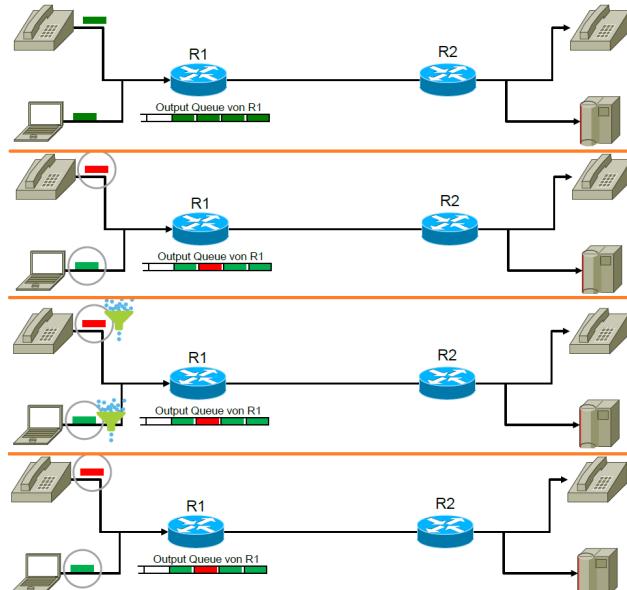
## SIP Adressen

- Direkte Angabe der IP-Adresse: `sip:bob@193.64.210.89`, alternativ
- Angabe einer eindeutigen Identifikation, die aufgelöst werden kann z.B. E-Mail: `sip:bob@domain.de` oder Telefonnummer
- SIP-Adressen können in Webseiten eingebettet werden

## Auffinden der Nutzer



## 1.4.8 Quality of Service (QoS)

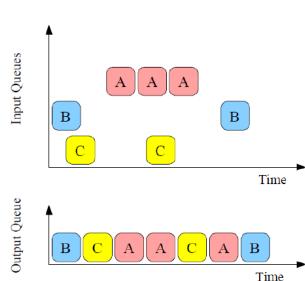


### Scheduling

- Auswahl des nächsten zu sendenden Paketes (z.B. `QoS Scheduler` in  $R_1$ )
- Möglichkeiten bei voller Queue vorzugehen:
  - **Tail-Drop**: neu angekommendes Paket verwerfen; **Priority**: Paket mit geringster Priorität verwerfen; **Random**: Zufälliges Paket verwerfen

### First-In First-Out (FIFO):

Beispiel: Einfaches FIFO



### Einfaches FIFO

- Pakete in Reihenfolge des Empfangs versenden
- Zeitlich zuerst Empfangenes Paket wird auch zuerst versendet

### Priority Queues

- Mehrere Klassen mit unterschiedlichen Prioritäten
- Separate Warteschlangen (*Queues*) für jede Priorität
- Paket aus befüllter Queue mit höchster Priorität wird gesendet
- Pakete gleicher Priorität werden nach FIFO gescheduled

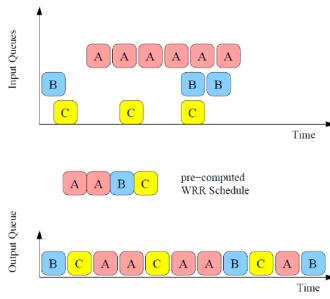
### Generalized Processor Sharing (GPS):

- Modell eines idealen Schedulers

- Jedem Datenstrom wird Gewicht  $w_i$  mit  $0 \leq w_i \leq 1$  zugeordnet
- Summe aller Gewichte ist 1:  $\sum_i w_i = 1$
- Jedem Datenstrom wird Ressourcenanteil entsprechend seines Gewichtes zugeordnet
- Nutzt ein Datenstrom seine Ressourcen nicht, so werden sie auf andere aufgeteilt: *work conserving*
- Theoretisches Modell - kann in real paketbasierten Netz nur angenähert werden da zu einem Zeitpunkt nur ein Paket gesendet werden kann

### Weighted Round Robing (WRR):

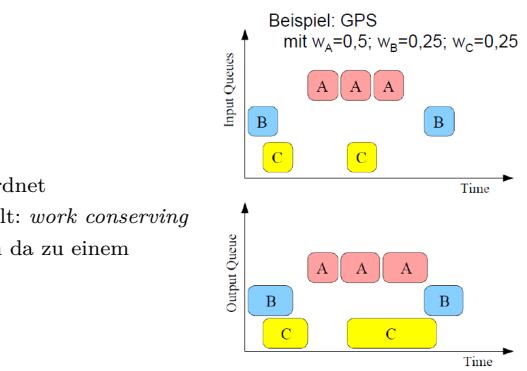
Beispiel: Weighted Round Robin  
mit  $w_A=0,5$ ;  $w_B=0,25$ ;  $w_C=0,25$



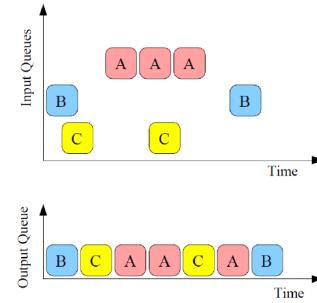
- Vorberechnung einer sich periodisch wiederholenden Zuteilung von Ressourcen an die Klassen/Datenströme
- Klassen kommen entsprechend ihrer Gewichte häufiger bzw. weniger häufig in WRR Schedule vor
- Zyklisches Prüfen der Klassen nach WRR Schedule → kann nun Paket senden
- Klasse, die nichts zu senden hat, wird übersprungen

### Weighted Fair Queuing (WFQ):

- Verallgemeinerung des von *WRR* verfolgten Ansatzes sowie paketbasierte Approximation von *GPS*
- Im Gegensatz zu *WRR* keine Vorberechnung der Schedule sondern stellt sicher, dass Klasse  $i$ , die im Intervall senden wollte, mindestens Bruchteil  $\frac{w_i}{\sum_j w_j}$  der Ressourcen erhält

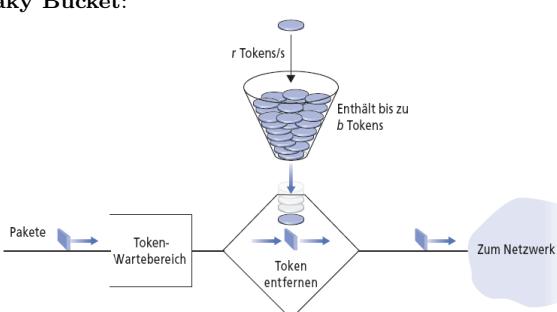


Beispiel: Weighted Fair Queuing  
mit  $w_A=0,5$ ;  $w_B=0,25$ ;  $w_C=0,25$



### Policing Mechanismen:

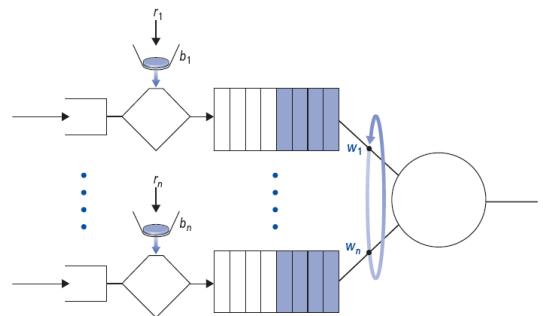
- Ziel ist Überwachung einer Klasse und das Regeln der Rate, mit der diese Pakete einspeist werden
- Kriterien sind
  - **Durchschnittliche Rate:** Pakete pro Zeitintervall
  - **Maximale Rate** als oberes Limit
  - **Burst-Größe:** maximale Anzahl von Paketen, die innerhalb eines sehr kurzen Zeitraumes (ohne Pause) gesendet werden dürfen
- Leaky Bucket:**



- Virtuelle Token werden mit fester Rate  $r$  generiert  
→ Limitiert *Durchschnittliche Rate*
- Jedes Paket benötigt genau einen Token zum Versand
- Token Bucket kann bis zu  $b$  Tokens aufnehmen → Limitiert *Burst-Größe*
- Insgesamt wird garantiert, dass innerhalb eines Zeitintervalls der Länge  $t$  die Anzahl der ins Netzwerk eingespeisten Pakete  $\leq r \cdot t + b$  ist

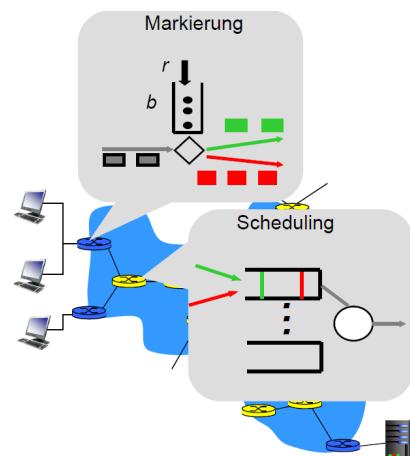
### Leaky Bucket kombiniert mit WFQ:

- Leaky Bucket limitiert Anzahl der in Warteschlange eingespeisten Pakete
- WFQ garantiert für jede Klasse eine bestimmte Mindestrate
- Ergibt beweisbare Maximalverzögerung → **Quality of Service Garantie**



### 1.4.9 Differentiated Services (DiffServ)

- Architektur zur Bereitstellung qualitativer Dienstklassen, ohne Zusage quantitativer Parameter
- Skalierbare Architektur:
  - **Einfache Funktion im Kernnetz:** per Hop Verhalten (*Per-Hop Behaviour - PHB*) des Routers basierend auf Paketmarkierungen
  - **Komplexe Funktionen am Rand des Netzes (Edge Router)** mit Paketklassifikation, Markierung und Traffic Policing



- **Edge Router mit Leaky Bucket**
  - Traffic Management pro Datenstrom (Quelle-Ziel Verbindung)
  - Markiert Pakete (*in-profile/out-profile*)
- **Core Router mit WFQ:**
  - Traffic Management pro Klasse (→ Markierung) sowie Buffering/Scheduling
  - Stellt relative QoS unter den Klassen sicher

- Markierung vom Paketen (Klassenzugehörigkeit) erfolgt in **IP-Header** im *Type of Service* (TOS) Feld (IPv4) bzw. im *Traffic Class* (TC) Feld (IPv6) mit Eintragung eines *Differentiated Services Code Point* (DSCP) als **6-Bit Wert**, der die Behandlung im *Core-Router* bestimmt
- Zwei Unterschiedliche **Per Hop Behaviours**:
  - **Expedited Forwarding:** Verkehrsraten einer Klasse entspricht oder übersteigt voreingestellte Rate am Router
  - **Assured Forwarding:** 4 Verkehrsklassen, jede mit garantierter, maximaler Bandbreite pro Klasse jeweils 3 Kategorien für das Verwerfen bei Überlast

### 1.4.10 Integrated Services (IntServ)

- Garantiert Datenrate/Dienstgüte, die erfordern:
  - **Ressourcenreservierung:** Datenstrom werden bei Verbindungsauflaufbau die benötigten Ressourcen zugeteilt
  - **Call Admission:** Abweisung eines Datenstroms wenn er nicht mehr verfügbare Ressourcen anfordert
  - **Verbindungsauftauschalisierung (Call Setup):** Ressourcen werden entlang des Pfades (d.h. bei jedem Router) von Quelle zu Ziel reserviert (per Hop Allokierung) → Resource Reservation Protocol (*RSVP*)
- Am Router *QoS-Scheduling* (z.B. WFQ)

## Neue Technologien im Bereich Web-Traffic

### 1.5.1 HTTP/2

#### Nachteile HTTP 1.1

- Zuerst muss Hauptseite (*index.html*) angefragt werden damit alle weiteren Elemente (z.B. CSS, Bilder) bekannt sind und ebenfalls angefragt werden können
- Auf einer TCP-Verbindung kann zu einem Zeitpunkt nur ein Element (z.B. ein Bild) übertragen werden → **Head-of-line Blocking**
- Große, teilweise redundante Header (z.B. wird sich Alias bei Anfrage nicht verändern), 500-800 Byte pro Request

#### Zusätzliche Herausforderungen

- Vielzahl einzelner Requests - Browser nutzt zwar parallele TCP-Verbindungen zum Server, ab 6-8 Verbindungen aber kein Gewinn mehr
- TCP Slow Start Verhalten: Bandbreite wird bei kleinen Dateien nicht ausgenutzt (*Congestion Window* erreicht passende Größe nicht)

#### Neuerungen HTTP/2 & HPACK

- **Multiplexen mehrere parallele Streams über einer TCP-Verbindung**
  - Kein Slow Start Problem und auch kein *Head-of-line Blocking* mehr
- **Server Push**
  - Server sendet **ohne clientseitigen Request** Dateien, bei denen er davon ausgeht, dass sie der Client benötigt (z.B. CSS), der Client muss somit nicht erst das HTML parsen und Requests erstellen (falls Client die Datei nicht braucht, bricht er den Vorgang ab)
- **Komprimierung der Header** (Header Compression - HPACK)
  - Deutliche Reduktion des Overheads
- Realisierung als **binäres Protokoll** - effizienter aber schwerer zu Debuggen
- **Datenflusskontrolle** (*Flow Control*) und **Priorisierung** (*Prioritization*) zur Verteilung der Übertragungsbandbreite auf einzelne Streams

#### Frame Typen und Streams

- Jeder Request/Response nutzt unabhängigen, eigenen Stream - gekennzeichnet durch *Stream Identifier* (Unterscheidung mit Farben)
- Gleiches URI-Schema und **Standard-Ports** wie bei HTTP 1.1
- Eine Auswahl an Frame-Typen:



|   | HEADERS                          | DATA    | SETTINGS                 | WINDOW UPDATE                | PUSH PROMISE                               | RST STREAM                         | PRIORITY                          | CONTINUATION                           |
|---|----------------------------------|---------|--------------------------|------------------------------|--------------------------------------------|------------------------------------|-----------------------------------|----------------------------------------|
| • | Header des Requests/der Response | Payload | Einstellungen, Parameter | Flusskontrolle eines Streams | Ankündigungs-Stream für <b>Server Push</b> | Sofortiges Schließen eines Streams | Festlegung einer Stream Priorität | Fortsetzung eines <b>Header Blocks</b> |

#### Verbindaufbau ohne TLS

- Anhand Port nicht erkennbar ob HTTP/2 oder HTTP 1.1 → **Upgrade-Header**
- Client signalisiert Server HTTP/2-Fähigkeit (Protocol Identifier **h2c = unverschlüsselte Übertragung**)
- Server mit HTTP/2 reagiert, Server ohne HTTP/2 ignoriert Header

#### Client (mit HTTP/2 Support)

```
GET / HTTP/1.1
Host: server.example.com
Connection: Upgrade, HTTP2-Settings
Upgrade: h2c
HTTP2-Settings: <base64url encoding of HTTP/2 SETTINGS payload>
```

#### Server (mit HTTP/2 Support)

```
HTTP/1.1 101 Switching Protocols
Connection: Upgrade
Upgrade: h2c
[HTTP/2 connection ...]
```

#### Verbindaufbau mit TLS

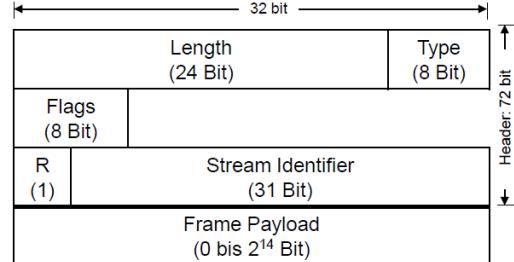
- TLS unterstützt Aushandlung von Protokollen auf der Anwendungsschicht (*Application-Layer Protocol Negotiation* (ALPN))
- HTTP/2 nutzt diesen Mechanismus um Nutzung von HTTP/2 abzustimmen
- Protocol Identifier **h2 = verschlüsselte Übertragung**

### Fortsetzung Verbindungsaufbau

- Nach Umschalten auf HTTP/2 wird von Client und Server folgende Sequenz gesendet: `PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n\r\n`
- HTTP 1.X-Server ignorieren alle folgenden Frames da es für die so aussieht, als würde ein Request auf eine Ressource mit Version HTTP/2.0 gesendet werden
- Direkt danach muss (ggf. leerer) *SETTINGS*-Frame folgen

### HTTP/2 Frame

- Fester Header mit 9 Byte
- Length:** Länge der Payload in Bytes, max.  $2^{14}$  wenn nicht über Settings angepasst
- Type:** Art des Frames, unbekannter Typ wird ignoriert
- Flags** (jeweils 1 Bit): Bedeutung je nach Typ, **R:** Reserviert
- Stream Identifier:** 31-Bit Integer zur Identifizierung eines Streams, Identifier **0x0** identifiziert Frames, die gesamte Verbindung (d.h. **nicht** nur einzelne Streams) betreffen



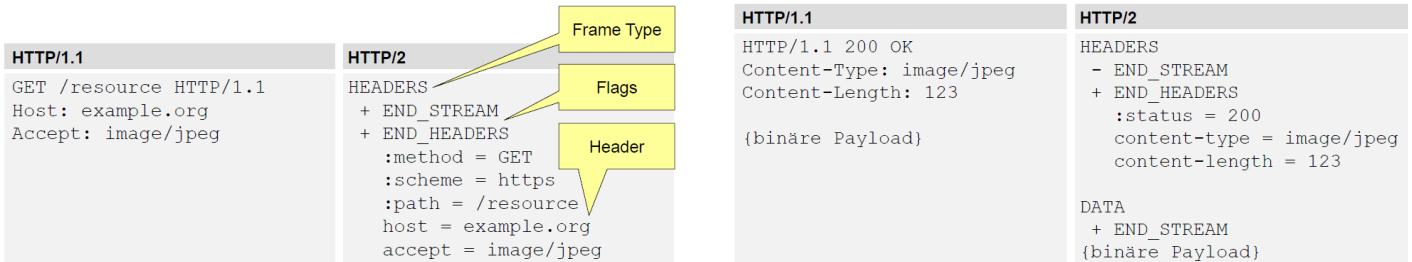
### Streams

- Verbindung kann aus Vielzahl gleichzeitig geöffneter Streams bestehen
- Stream kann entweder einseitig oder zwischen Server und Client zusammen genutzt sein
- Client und Server multiplexen die Frames der Streams über eine TCP-Verbindung
- Empfänger verarbeitet die einzelnen Frames eines Streams entsprechend der Sendereihenfolge
- Streams werden über *Stream Identifier* eindeutig identifiziert (wird vom Endpunkt vergeben, der Stream initiiert)
- Client vergibt immer **ungerade** Stream identifier, Server immer **gerade** Stream identifier → **keine Überschneidung**
- Stream kann von anderen Streams abhängig sein (→ Abhängigkeitsbaum zur Ressourcenverteilung)

### Flow Control

- Stream konkurrieren um Bandbreite einer TCP-Verbindung
- Flow Control erlaubt Regelung des Datenflusses einzelner Streams sowie der gesamten Verbindung
- Initiales Fenster: 65.535 für neuen Stream und Gesamtverbindung
- Daten dürfen nur gesendet werden, so lange Platz im Fenster
- Empfänger teilt Sender aktuellen Platz im Empfangsfenster über *WINDOW\_UPDATE* Frame mit (Kredit-basiertes Verfahren)
- Flow Control betrifft nur DATA:** Nur *DATA* Frames reduzieren den Platz im Fenster, alle anderen Typen dürfen immer gesendet werden → wichtige Control Frames werden nicht blockiert

### Request/Response



### Server Push

- Client sendet regulären Request
- Server erkennt Situation um durch *Server Push* Übertragung zu beschleunigen, z.B. eingebetteter Link auf ein Bild
- Server sendet *PUSH\_PROMISE*-Frames, die Requests für den zu pushenden Inhalt entsprechen
  - Request Header Feld gesetzt (wie bei regulärem Request)
  - Ankündigung des Stream Identifiers (*Promised Stream Identifier*), der für die Response verwendet wird
- Server liefert sowohl vom Client explizit in Schritt (1) angeforderte Inhalte als auch *Push Response* über *DATA*-Frames mit den entsprechenden Stream Identifiern aus
- Client kann selbst kein Push verwenden und kann *Server Push* per Flag *SETTINGS\_ENABLE\_PUSH* 0 deaktivieren
- Falls *Server Push* vom Client unerwünscht, kann er *RST\_STREAM* Frame mit zugehöriger Stream ID senden

### Schwierigkeit bei Dekodierung in Wireshark

- Binäres Protokoll, mehrere Streams laufen gleichzeitig über eine TCP Verbindung
- Zur Dekodierung benötigt man die vom Browser verwendeten Schlüssel da praktisch alle HTTP/2-Verbindungen TLS nutzen

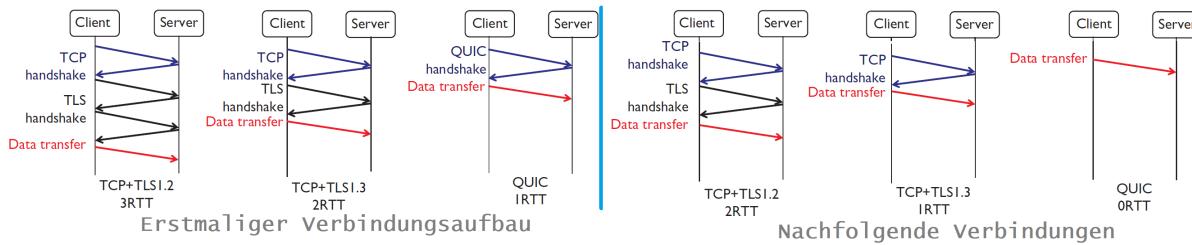
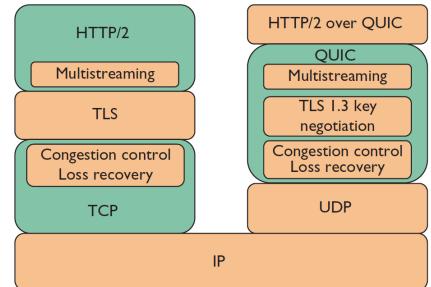
### 1.5.2 QUIC UDP Internet Connections (QUIC)

#### Nachteile TLS + TCP

- Problematisch für Anwendungen, die nur geringe Verzögerungen tolerieren können (HTTP/2 Übertragung mit mehreren Streams)
- In HTTP/2 kann weiterhin **Head-of-line blocking** auftreten, wenn ein Paket verloren geht und TCP daher die Übertragungsrate drosselt → alle anderen Frames der gemultiplexten Streams werden blockiert
- Implementiert als Teil des Betriebssystems, daher schwierig upzudaten

#### Grundlegender Ansatz

- Transportmechanismen werden in der Netzwerk hierarchie oberhalb von UDP im *User-Space* implementiert (Updates einfacher möglich)
- QUIC basiert, wie HTTP/s auch, auf Frames mit unterschiedlichen Frame-Typen
- Optimierung auf HTTP/2 Stream Multiplexing + Datenflusskontrolle (kein **Head-of-line blocking** mehr)
- Im erstmaligen Verbindungsaufbau werden sowohl Parameter für **QUIC** als auch für **TLS** übertragen
- Client speichert **Cryptographic Cookie**, der für nachfolgende Verbindungen verwendet wird (Server authentifiziert Client über Cookie)



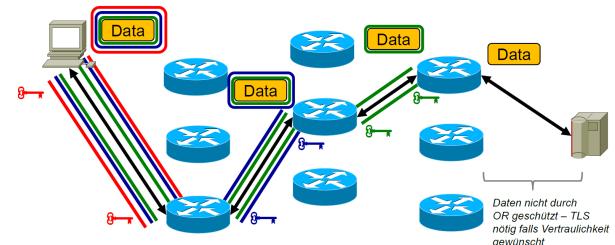
### 1.5.3 Onion Routing / Tor

#### Motivation

- Trennung von **Identifikation** und **Routing** (Kommunikationspartner können über IP-Adresse identifiziert werden)
- Weiterleitende Knoten und angefragter Server sollen die Identität des Anfragenden **nicht** ermitteln können
- **Mehrfache, ineinander geschaltete Verschlüsselung** der Daten (wie eine Zwiebel)

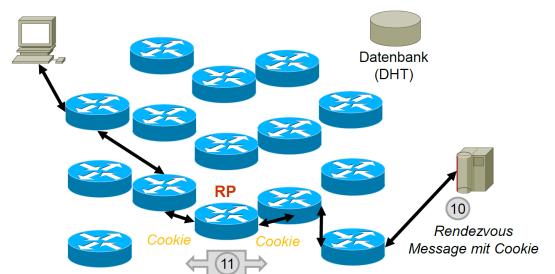
#### Implementierung im Tor-Projekt

- Kommunikation über **geographisch verteilte, von unterschiedlichen Betreibern** gemanagte, **zufällig ausgewählte Relay-Knoten**
- Auswahl wird periodisch geändert (**Guard**: Erster Knoten nach dem Client)
- Pro Hop wird eigener, symmetrischer Schlüssel ausgehandelt
- Nur **Entry-Onion-Router** sieht Quell-IP
- Nur **Exit-Onion-Router** sieht Ziel-IP (Nutzdaten auch sichtbar wenn kein TLS verwendet wird!)
- Andere Onion-Router sehen weder Quelle noch Ziel



#### Hidden Services

- Dienstanbieter soll ebenfalls anonym bleiben (bisher nur Client)
- Dienstanbieter erzeugt Schlüsselpaar, dass den Dienst langfristig identifiziert
- Hostname ist halbiertes **SHA-1 Hash** des **Public Key** des Dienstanbieters
- **Hidden Service Descriptor** wird in dezentrales Verzeichnis hochgeladen (wird vom Client mit Hilfe der Onion-Adresse des Dienstes angefragt)
- Client verwendet **zufällig ausgewählte Onion-Router** (Relays) zum Verbindungsaufbau mit Hidden Service (Zuordnung erfolgt über **One-Time-Secret/Cookie**)
- **Schwachstellen:** Hidden Services identifizierbar durch veraltetes SHA-1 Hasverfahren, RSA verwendet nur 1024-Bit



## Praktikumsprojekt

### 1.6.1 MQTT

- Verbindet (ressourcenbeschränkte) *eingebettete Geräte* und Netzwerke
- *Publish/Subscribe*-Pattern
- Verwendet TCP als Transportprotokoll
- **Subscriber**
  - Registriert sich beim *Broker* für bestimmte Topics
  - Wird von Broker benachrichtigt, wenn neue Nachrichten für Topic verfügbar

- **Publisher**
  - Überträgt neue Topic-Nachrichten an den Broker

- **Broker**
  - Stellt autorisierte Verbindungen mit *Publisher & Subscriber* und damit die Sicherheitsfunktionalität her

- **Header**

- **Message Type:** Art der Nachricht (CONNECT, CONNACK, PUBLISH, SUBSCRIBE)
- **DUP:** Duplikat, Empfänger hat Nachricht vllt. schon empfangen
- **QoS:** Häufigkeit der Zustellung (höchstens 1x, mindestens 1x)
- **Retain:** Server soll zuletzt empfangene PUBLISH-Nachricht vorhalte

- **Verbindungsaubau MQTT Client ↔ MQTT Broker**

- Unter einem MQTT Client versteht man **sowohl** einen Subscriber **als auch** einen Publisher
- Client sendet CONNECT an Broker
  - \* **Clean Session:** Client möchte persistente Verbindung aufzubauen (Clean Session = 0) → Broker speichert Nachrichten
  - \* **Benutzername und Passwort:** im Klartext, sollte nur in Verbindung mit TLS verwendet werden
  - \* **Keep Alive**
- Broker bestätigt mit CONNACK

- **Publish**

- Nach Verbindungsaubau kann der Client eine Nachricht mit PUBLISH in einem Topic veröffentlichen
- Enthält Topic-Name und Payload
- Broker sendet neue Nachricht an Clients, die das Topic abonniert haben

- **Subscribe**

- Um ein Topic zu abonnieren, schickt der Client ein SUBSCRIBE an den Broker
  - \* **List of Subscriptions:** ggf. mehrere Topics, die der Client abonnieren möchte
- Broker antwortet mit SUBACK Nachricht, die für jedes abonnierte Topic einen Returncode enthält

- **Unsubscribe** wie *Subscribe*, nur das UNSUBSCRIBE alle Topics enthält, die deabonniert werden sollen

