Zusammenfassung - Netzwerke II

Drahtlose Netzwerke

1.1.1 Grundlagen

Wireless Host (Drahtloser Teilnehmer): Endystem 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), Teilnehme 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

Übliche Datenraten

GSM (2G)	0.56 Mb/s
UMTS (3G)	4 Mb/s
LTE (4G) und 802.11b	5 - 11 Mb/s
802.11ag	54 Mb/s
802.11n	200 Mb/s

Beispiele für Single und Multi-Hop

	Single Hop	Multiple Hops
Infrastruktur	Host verbindet sich mit Basisstation (Wifi, zellulare Netzwerke) und diese dann mit dem Internet	Host muss möglicherweise durc mehrere drahtlose Geräte um sich mit dem Internet zu verbinden: *Mesh Net*
Keine Infrastruktur	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: *MANET*, *VANET*

Herausforderungen bei drahtloser Übertragung

- Teilnehmer zeitweise nicht erreichbar (Funkloch)
- IP-Adresse ändert sich
- Höhere Anzahl an Übertragungsfehlern durch Inteferenz (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.
TODO: BEHEBUNG / VERMINDERUNG DURCH?

Aufteilen eines Mediums:

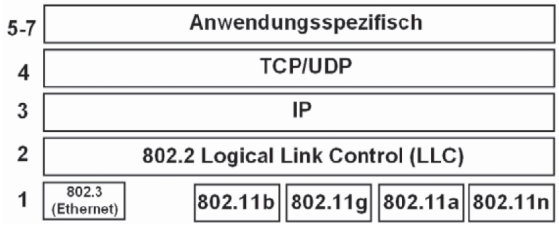
- TDMA (Time Division Multiple Access)
  1. synchron: Jeder Teilnehmer hat festen Zeitslot, nur in diesem kann er senden
  2. asynchron: keine festen Zeitslot, jeder nutzt aktuellen Zeitslot wenn er Daten hat - Absender wird in Header geschrieben

- 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 ⇔ **Chips**
  3. Empfänger multipliziert empfangende 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 (⇒ Umso mehr Teilnehmer umso geringerer SNR ⇒ Sendeleistung erhöhen)

CDMA - Beispiel zur Kodierung
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: $Z_{1,m} = (-1,-1,-1,1,-1,1,1,1)$, $Z_{0,m} = (1,1,1,-1,1,-1,-1,-1)$. Der Empfänger dekodiert folgendermaßen: $\frac{\sum_{m=1}^{M} Z_{i,m} \cdot c_m}{M}$.
Dabei ist M Länge des Spreizcodes (in diesem Beispiel 8), c_m Spreizfaktor an der Stelle m, $Z_{i,m}$ die gespreizten Daten, d.h:
$d_1 = \frac{(-1)\cdot 1+(-1)\cdot 1+(-1)\cdot 1+1\cdot(-1)+(-1)\cdot 1+1\cdot(-1)+1\cdot(-1)+1\cdot(-1)}{M} = \frac{-8}{8} = -1$
$d_0 = \frac{1\cdot 1+1\cdot 1+1\cdot 1+(-1)\cdot(-1)+1\cdot 1+(-1)\cdot(-1)+(-1)\cdot(-1)+(-1)\cdot(-1)}{M} = \frac{8}{8} = 1$
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:

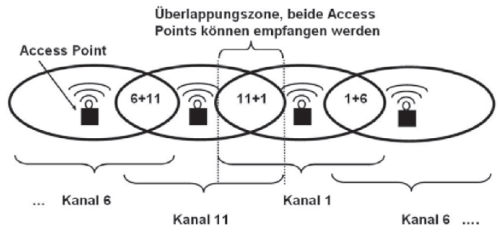


802.11: '97, FHSS/DSSS, 1-2MBit/s, 2.4 GHz
802.11b: '99, DSSS, 1 - 11MBit/s, 2.4 GHz
802.11n: '09, OFDM/MIMO, 6 - 600MBit/s, 2.4 oder 5 GHz
802.11ac: '14, MU-MIMO, bis zu 6.93 GBit/s, 5 GHz
802.11ay: '19, 20 - 40 GBit/s, 60GHz

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:

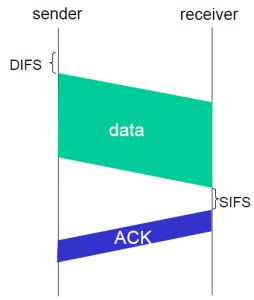


∘ Bis zu 13 Kanäle mit je 5 MHz zwischen 2410 MHz - 2483 MHz.
∘ Bei **DSSS** Kanalbreite = 22MHz, bei **OFDM** = 20 MHz / 40 MHz (ohne / mit Kanalbündelung). Störungsfreier Betrieb nur bei passendem Abstand (5 Kanäle bei DSSS).
Hierzu bitte auch das erste Übungsblatt vom Praktikum durchlesen!

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	**Aktives Scannen**
PC ist passiv, hört Kanal ab	PC sendet Probe Request
APs senden *Beacons*	APs senden Probe Response
PC sendet Association Request an ausgewählten AP	PC sendet Association Request an ausgewählten AP
AP sendet Association Response an PC	AP sendet Association Response an PC

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

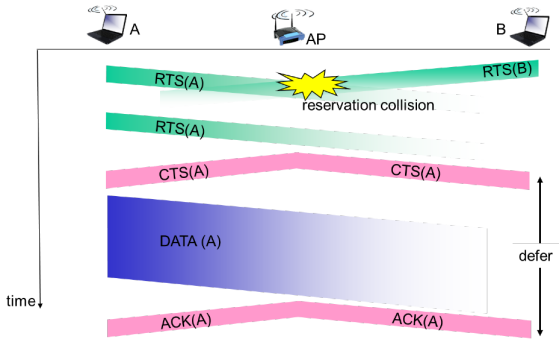


- Sender wartet Zeitspanne *DIFS* (Distributed Coordination Function Interframe Spacing) während der Medium frei sein muss
- Sender überträgt Daten (kein Collision Detection)
- Empfänger prüft CRC
- Empfänger sendet ACK falls CRC korrekt nach Wartezeit *SIFS* (Short Interframe Spacing) (SIFS < DIFS) um Senden eines normalen Frames dazwischen zu verhindern außerdem 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 wenn mehrere Stationen gleiches DIFS abgewartet haben und es sonst zu Kollision käme. Nach Ablauf des random backoffs Daten übertragen
- Falls nicht frei: Warten bis Kanal frei, wenn Kanal länger als 1 DIFS frei, dann *random backoff* verringern & anschließend Daten senden
- Falls kein ACK erfolgt: Contention Window verdoppeln und Daten erneut senden
- Empfänger sendet ACK nach Ablauf eines **SIFS** bei korrekt erhaltenen Daten

Nachteile CSMA/CA: Senden dauert lange, Kollision wird nicht erkannt - 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

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

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

- Max. 2313 Bytes an Nutzdaten
- 3. Adresse erlaubt Umsetzung auf Ethernet-Rahmen
- WLAN-Reichweitenvergrößerung durch überlappende BSSs, IP-Adresse bleibt gleich da identisches Subnetz - nur AP ändert sich

⇒ Switch ändert Port↔IP-Zuordnung wenn sich Teilnehmer vom neuen AP meldet

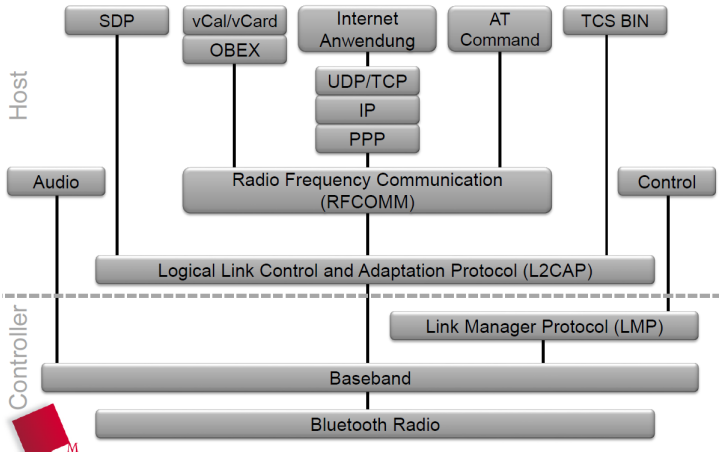
Unterschied CSMA/CA↔CSMA/CD: CSMA/CD bei Ethernet sendet *JAM*-Signal, CSMA/CA erkennt keine Kollision (versucht nur zu verhindern)

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, aktiviert geparkte

Bluetooth Profile
Profil spezifiziert Anwendung von Bluetooth für bestimmten Zweck



∘ **AT Kommando:** Kommando zur Modemsteuerung
∘ **Baseband:** Basisband, Paketformate
∘ **L2CAP:** Logical Link Control and Adaptation Protocl:
Bietet verbindunsorientierte- 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)
Verbindungsauf- und abbau, Wiederaufbau abgebrochener Verbindungen

Sensoren (z.B. Pulsmesser, Thermometer) sollen lange Laufzeit (⇒ geringer Stromverbrauch) aufweisen. Bluetooth 4.0 beinhaltet *Bluetooth Smart* (Low-Energy Profil auf Basis von einem *Generic Attribute Profile* (GATT)).

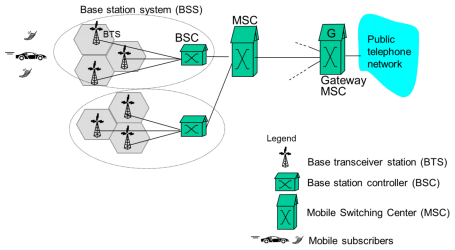
ZigBee
∘ Ziel: Drahtlose Übertragung bei geringem Stromverbrauch (geringe Datenraten: 20 - 250kbit/s, selten aktiv (*low duty-cycle*)).
∘ Übertragen von Sensordaten, Heim- und Gebäudeautomatisierung
∘ **Endgerät:** Reduced Function Device *RFD* - nur Teil des ZigBee Protokolls implementiert (geringere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

1.1.4 Zellulare Netzwerke

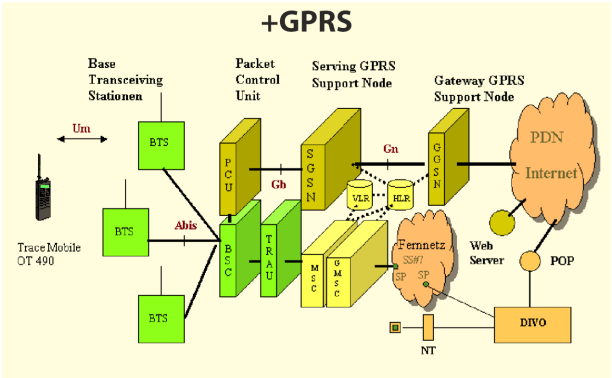
∘ **1G:** Analog (A/B/C-Netz) ∘ **2G:** GSM ab '92, 2.5G = GPRS, 2.75G = EDGE ∘ **3G:** UMTS ab '03 ∘ **4G:** ab '14 LTE ∘ **5G** ab '21, Latent < 1ms
∘ **Mobilfunkzelle:** Von Base Transceiver Station(BTS) abgedeckter Bereich ∘ **Air-Interface:** Untere 2 Netzwerkschichten Mobile Station $\Leftrightarrow$ BTS

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

2G Netzwerkarchitektur (GSM):

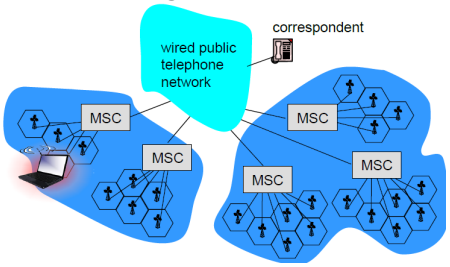


- **Base Station Controller (BSC):** übernimmt Ressourcenzuweisung und Mobilitätsmanagement in einem Base Station Subsystem (BSS)
- **Mobile Switching Center (MSC):** Anrufauf- und Abbau, Verbindung ins Festnetz, Mobilitätsmanagement
- **Gateway-MSC:** Vermittlungsfunktionen, Verbindung zu anderen Netzen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



- Mit GPRS-Komponenten erstmals paketvermittelte Datendienste
  - verwendet bereits bestehende BTS mit
- **Packet Control Unit (PCU):** Kommuniziert über den BSC mit Endgerät und auch mit der SGSN, überwacht und verwaltet Datenpakete, Ressourcenverteilung
- **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
- Sprachdaten laufen über $BTS \Leftrightarrow BSC \Leftrightarrow MSC \Leftrightarrow G-MSC \Leftrightarrow Festnetz$
- Paketdaten laufen über $BTS \Leftrightarrow BSC \Leftrightarrow PCU \Leftrightarrow SGSN \Leftrightarrow GGSN \Leftrightarrow Internet$

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 Verbindungsaufbau über das *VLR* des MSC
  ⇒ Falls gerade keine Verbindung besteht: Broadcast-Nachricht über alle Basisstationen des jeweiligen MSC (*Paging*), Mobile meldet sich ggf., damit genaue Basisstation bekannt

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

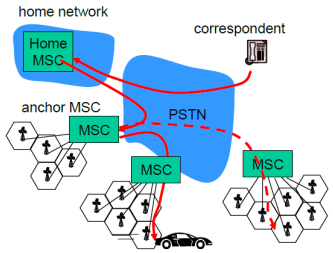
Ablauf Inter-BSC Handover:

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

Arten von Handover:

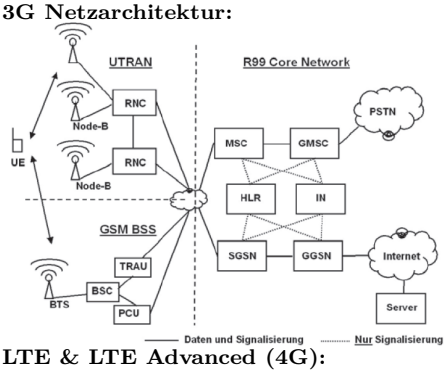
- *Intra BSC:* Aktuelle und neue Zelle gehörten zum selben BSC
- *Inter BSC:* Aktuelle und neue Zelle gehören zu unterschiedlichen BSC aber gleichen MSC
- *Inter MSC:* Aktuelle und neue Zelle gehören zu unterschiedlichen MSC
- *Subsequent Inter MSC:* Teilnehmer wechselt nach *Inter MSC* in Zelle eines 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:

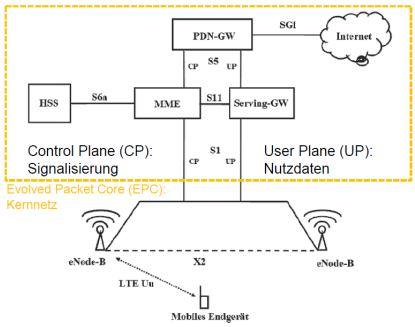


- Mit UMTS '99 neues Air-Interface *Universal Terrestrial Radio Access Network* (**UTRAN**)
- basierend auf Wideband-CDMA
- Ab UMTS v4 Umstellung auf IP basierte Kommunikation (Sprache & Daten)

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 + 2 Releases**: *LTE Advanced* - 3 Gbit/s DL bzw. 1.5 Gbit/s UL, Bündelung bis zu 5 Carrier
- **LTE + 3/4 Releases:** *LTE Advanced Pro* - Bündelung von bis zu 32 Carriern, QAM-256, LTE Narrow Band IoT, **5G** Standardisierung läuft
- **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)
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 ⇒ 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 Veringerung ⇒ Datenrate sinkt
⇒ Verzögerung durch Link-Layer Retransmission (Auswirkungen auf Echtzeitanwendungen), Drahtloser Link meist geringere Datenrate

Security

1.2.1 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

1.2.2 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. $k = 3$: 000 ⇒ 110, 001 ⇒ 111, 010 ⇒ 101, 011 ⇒ 100, 100 ⇒ 011, 101 ⇒ 010, 110 ⇒ 000, 111 ⇒ 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
  $c(1) = K(m_1\ XOR\ c(0))$
  $c(2) = K(m_2\ XOR\ c(1))$

Data Encryption Standard (DES):
56-Bit symmetrischer Schlüssel, Verarbeitung von 64-Bit Blöcken mit *Cipher Block Chaining*, per *Brute Force* knackbar, *3DES* etwas sicherer

Adanved 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 = öffentlicher Schlüssel)
∘ Symmetrische Verschlüsselung wesentlich schneller ⇒ 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) ∘ *SHA-1* mit 160-Bit Hash (inzwischen auch kritisch) ∘ *SHA-2* mit 224/256/384/512-Bit Hash (empfohlen)

Message Authentication Code (MAC)
∘ Gemeinsames Geheimnis besteht aus *Authentication Code* s und Nachricht m, Sender hängt $H(m + s)$ (=MAC) an Nachricht m an
∘ Übertragung von Nachricht + MAC an Empfänger, dieser berechnet ebenfalls Hash aus $m + s$ und prüft, ob gesendeter MAC passt

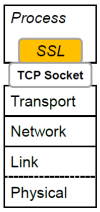
Digitale Signaturen
∘ Unterschrift als Bestätigung der Urheberschaft eines Dokuments, MAC mit *shared key* **s** ungeeignet, da **s** sowohl Unterzeichner als auch Prüfer bekannt sein muss ⇒ Prüfer kann Unterschrift fälschen
∘ Lösung:
Berechne $H(m)$, m := zu unterschreibende Nachricht
Unterzeichner 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 ⇒ 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 einmal verwendeten, zufälligen Zahl (*Nonce*) die mit symmetrischem Schlüssel verschlüsselt wird

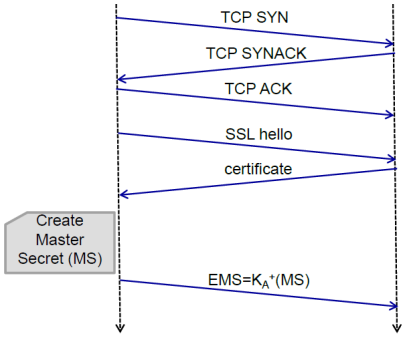
- Sender sendet eigenen Namen an Empfänger
- Empfänger sendet Nonce R zurück
- Sender sendet mit eigenem privaten Schlüssel verschlüsseltes R
- Empfänger fordert publick key an
- Sender sendet public key (ggf. mit Zertifikat einer CA) an Empfänger
- Empfänger prüft ob $K_A^+(K_A^-(R)) = R$

1.2.3 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: Webbrowser als Client, Server besitzt public & private key + Zertifikat für public key zur Identitätsbestätigung (Domainname)



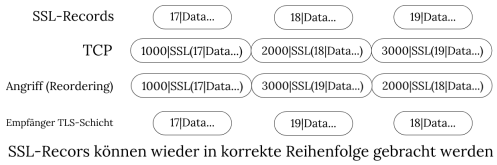
- TCP-Verbindung aufbauen
- Client sendet Liste an unterstützten *SSL Cipher Suites* + Nonce
- Server sendet Wahl der *Cipher Suite*, CA-Zertifikat und Server-Nonce
- Client:
  - prüft Server-Zertifikat
  - generiert *Pre-Master Secret* **(PMS)**
  - sendet mit Public Key des Servers verschlüsseltes PMS an den Server
- Client & Server leiten mit *KDF* aus PMS und Nonces das *Master Secret* (**MS**) her
  - Empfänger verschlüsselt *MS* mit public key des Servers =**EMS**
- 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 Schlüssel aus einer *Key Derivation Function* **KDF** mit Eingabeparametern Master Secret und Zufallsdaten, abgeleitet (nur für eine Sitzung gültig):

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

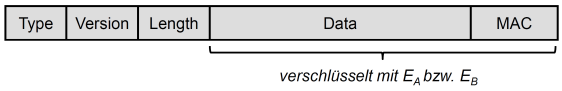
SSL Datenübertragung



SSL-Recors können wieder in korrekte Reihenfolge gebracht werden

- Aufteilung des TCP-Byte-Stroms in separate Abschnitte (*SSL Records*)
- Jeder Record enthält zusätzliche Sequenznummer (beginnend bei 0) um Reordering von Records verhindern
- *MAC* wird berechnet über Daten + MAC-Schlüssel (M_A oder M_B) + Sequenznummer
- Nutzung von *Nonces* verhindert Replay-Angriff (Wiederabspielen einer Datenverbindung)

SSL Record



- **Type:** Unterscheidung von Handshake/Nutzdaten/Beenden
- **Version:** SSL-Versionsnummer
- **Length:** Länge der Daten (ohne Header) - Extraktion 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 Suite*s an (vom Client favorisierte ganz oben), Server sucht eine aus (unabhängig von Clientpriorität)

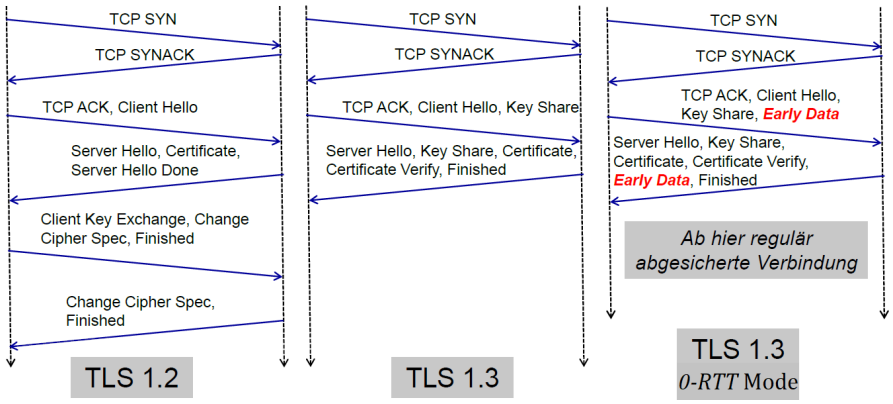
Symmetrisch	**Public Key**	**MAC**
DES, 3DES, AES	RSA, DH (Diffie-Hellmann), ECDH (Elliptic-Curve DH)	SHA-1, SHA-2, MD5

Vorzeitiges Schließen einer SSL-Verbindung

- Wird durch zusätzlichen *SSL-Record* zum ordnungsgemäßen Beenden verhindert (Authentifizierung erfolgt über *MAC*)

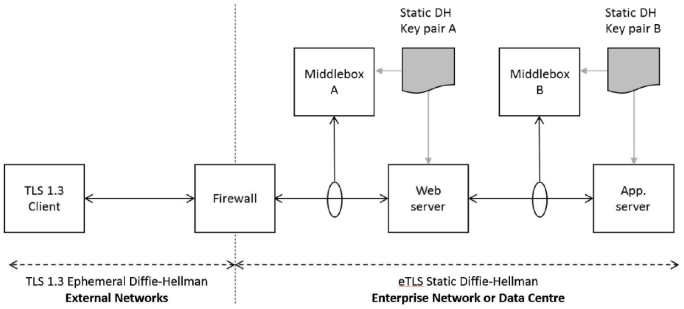
1.2.4 TLS v1.3

- 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 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: Kein *Forward Secrecy*, anfälliger ggü. Replay-Attacken

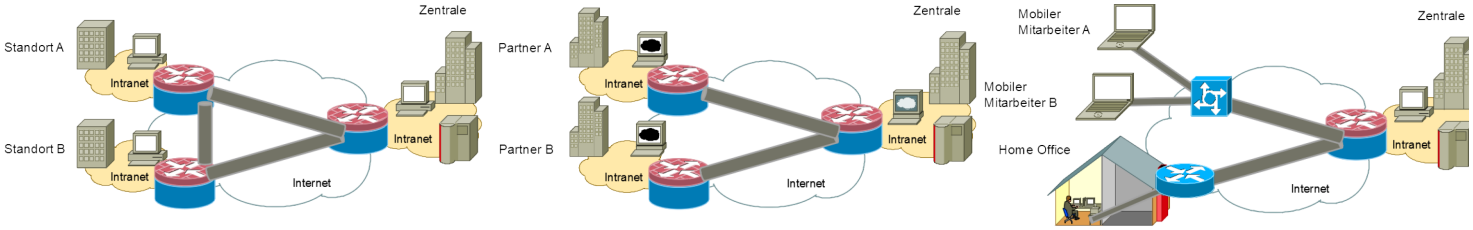
1.2.5 Enterprise Transport Security (ETS)

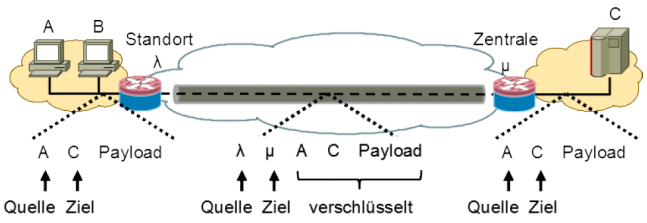


- Unternehmensnetze setzen viele Middleboxes zur Angriffserkennung (*Intrusion Detection*) oder als Application Layer Firewall ein, Datenverkehr kann aufgrund von *Perfect Forward Secrecy* nicht entschlüsselt werden → keine Überwachung durch Middleboxen möglich
  ⇒ Verzicht auf *Perfect Forward Secrecy*,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

1.2.6 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*)



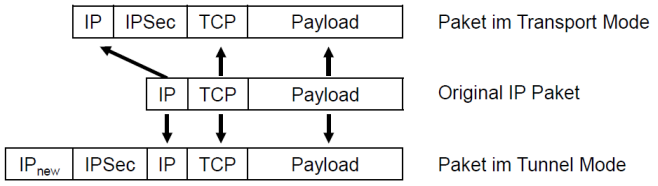


- **Tunneling:** Weiterleitung beliebiger Datenpakete über unsicheres Transitnetz
- VPN nutz eigenen Adressraum
- A, B, C := VPN-Adressraum, λ, μ := Internet-Adressraum
- **Authentisierung**:
  Password Authentication Protocol **PAP** - unsicher da Klartextaustausch
  Challenge Handshake Authentication Protocol **CHAP** - Challenge-Response
- **Software:**
  Server (VPN-Gateway) per VPN-Server Software
  Client per VPN-Client (Cisco) / VPN-Adapter
- **Protokolle:**
  Point-to-Point Tunnelling Protocol **PPTP**
  Layer Two Tunnelling Protocol **L2TP** (*PPTP*-Nachfolger)
  - Tunnelt beliebige Pakete über UDP

1.2.7 IPSec

◦ Dominierendes Protokoll für *VPN*s 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* (**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-Paketes (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
- legen fest, **was** in **welcher Art** verschlüsselt wird
- Jede *SA* hat eindeutigen *Security Parameter Index* (**SPI**)
- IPSec Einheit erhält die *SPI*s in *Security Association Database* (**SPD**)

Problem mit *Network Address Translation* (NAT):

◦ *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):

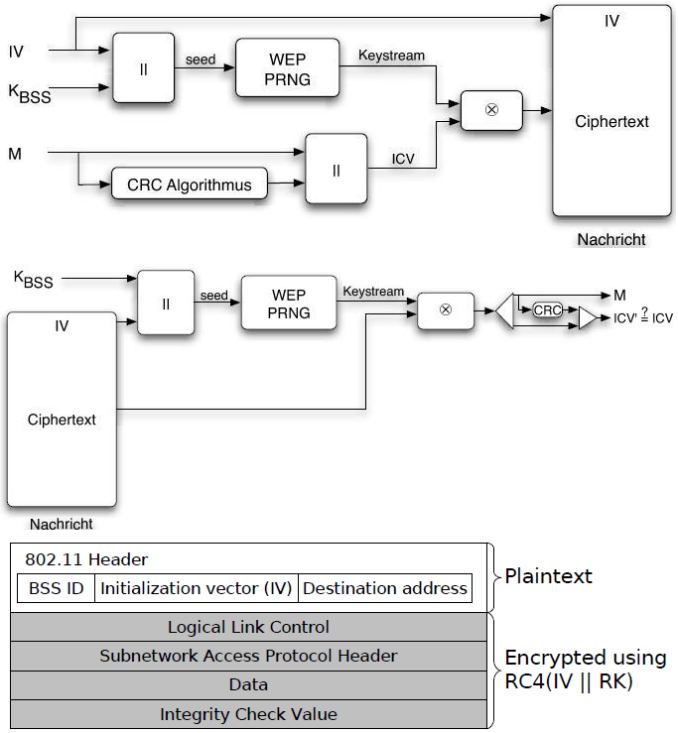
- Automatisiertes Aushandeln der *SA*s (mit Schlüssel)
- Basiert auf *Security Association Key Management Protocol* (**ISAKMP**)
- Variante 1: *Pre-Shared Key* (**PSK**)
  - Beide Seiten sind mit gemeinsamen Geheimnis vorkonfiguriert
  - *IKE* wird genutzt um SAs aufzusetzen
- Variante 2: *Public Key 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*, entweder im *Aggressive Mode* (1.5 RTT) oder *Main Mode* (3 RTT)
- Phase 2: Aushandlung von IPSec SA (ebenfalls mit ISAKMP), Teilnehmer signieren ihre Nachrichten

1.2.8 Alternative VPN Protokolle

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

1.2.9 Sichere WLANs

802.11: Wired Equivalent Privacy (WEP)



- Schlüsselverteilung nicht Teil von *WEP*, Aufgaben lediglich Authentifizierung und Verschlüsselung (mit Stromchiffre *RC4*)
- **Schlüssel:**
  - 24-Bit Initialisierungsvektor (IV), wechselt von Rahmen zu Rahmen, wird im Klartext übertragen
  - 40-Bit (später 104-Bit) symmetrischer Schlüssel (*Root Key*)
- $m(i)$:= Nachricht-Byte; $ks(i)$:= Schlüsselstrom-Byte; $c(i)$:= Cipher-Byte
  - Sender: $c(i) = ks(i)\ XOR\ m(i)$; Empfänger: $m(i) = ks(i)\ XOR\ c(i)$
- *Integrity Check Value* (**ICV**):
  - *CRC32* über zu verschlüsselnde Daten
  - *Root Key* und IV zur Verschlüsselung von Daten und *ICV*
  - schützt vor zufälligen Fehlern, nicht vor Angreifer (kein geeigneter *MAC*)
- **Authentifizierung:**
  - **Open System:** Alle Authentifizierungsnachrichten w/o Prüfung zulassen
    ⇒ Client kann ohne Schlüssel keine korrekten Nachrichten senden
  - **Shared Key:** Client sendet Request an AccessPoint
    AP sendet 128-Byte *Nonce*, Client verschlüsselt *Nonce* mit *shared key*
    AP entschlüsselt Ciphertext mit IV und *shared key* (Ergebnis = *Nonce*)

Probleme mit *Shared Key Authentication* bei WEP:

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

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 mit pre-shared key

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 und Authentifizierungsserver über einen Access Point
- *EAP-TLS*: Nutzung von Zertifikaten; *EAP-PEAP:* Verschlüsselter Tunnel für Authentifizierung

802.1X

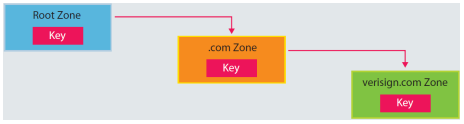
- Erweitert *802.11* durch *EAP* um Sicherheitsfunktionen wie: Schlüsselmanagement, Nutzeridentifikation, Tokens...

1.2.10 DNS Security Extensions (DNSSEC)

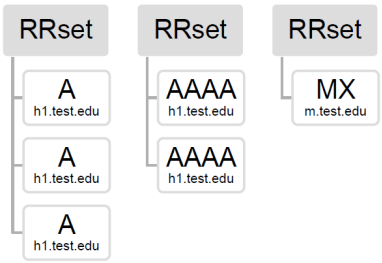
∘ Absicherung von Authentizität und Integrität der DNS-Informationen (**nicht** Vertraulichkeit)
⇒ Verwendung von Signaturen, umgesetzt mit asymmetrischen Kryptosystem (public & private Key):
- *Resource Record* (RR) wird mit geheimen Schlüssel unterschrieben
- DNS Client prüft mit öffentlichem Schlüssel die Authentizität und Integrität des empfangenen *RR*s
- *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:

RR Typ	**Erläuterung**
RRSIG	Digitale Signatur eines *Resource Record Sets* (RRset)
DNSKEY	Public Signing Key, wird vom Resolver zur Prüfung digitaler Signaturen verwendet
DS	Delegation Signer: Hash des *DNSKEY* um dessen Authentizität sicherzustellen,delegiert Vertrauen eine Ebene weiter in Chain-of-Trust
CNSKEY, CDS	Child Zone informiert Parent: Key-Signing-Key ändern, DNSSEC an/aus

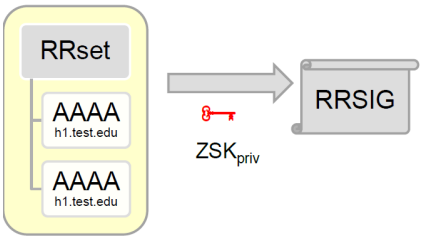


1



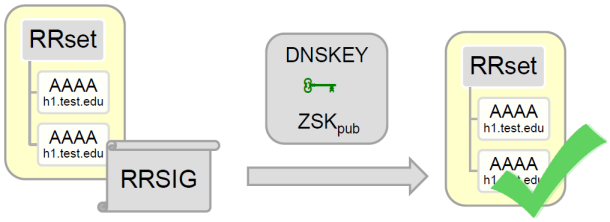
∘ Gruppieren von gleichen RRs (Label, Klasse & Typ) zu einem *RRset*
∘ RRset kann später gemeinsam signiert werden

2



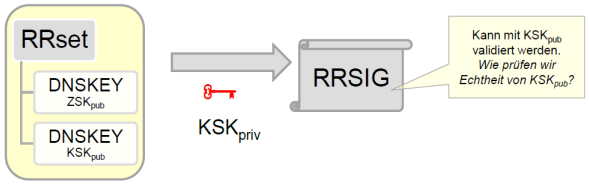
∘ Für Zone zuständiger, authoritativer DNS-Server nutzt privaten Schlüssel des *Zone Signing Key Pair* (ZSK) um RRset zu signieren → RRSIG

3



∘ DNS Resolver benötigt öffentlichen Schlüssel des *ZSK*
→ wird über *DNSKEY* verbreitet
∘ DNS Server liefert RRSIG mit an den DNS Resolver, dieser verifiziert RR durch DNSKEY

4



∘ Zone erstellt weitere Schlüsselpaar: *Key Signing Key* (KSK)
∘ Hierarchischer Ansatz: Parent bestätigt KSK und KSK validiert ZSK
∘ ZSK kann nun unabhängig vom Parent geändert werden (falls ZSK kompromittiert)
∘ RRset bestehend aus ZSK_{pub} & KSK_{pub} wird mit KSK_{priv} signiert

5 Chain-of-Trust: Vertrauen muss von Parent auf Child Zone übertragen werden können → *Delegation Signer* (DS)-Record
Zonen-Operator der Child-Zone erstellt Hash von DNSKEY mit KSK_{pub} & gibt ihn an Parent:
- Dieser veröffentlicht ihn in *DS*-Record

1.2.11 DNS over TLS

∘ 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 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

1.2.12 DNS over HTTPS

- Absicherung von Vertraulichkeit
- 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 (Cloudfare, Google) bieten DNS-over-HTTPS

→ Konzentration auf wenige Anbieter, Privatssphäre Gefärdung

- Viel Overhead (Anwendungsschicht: DNS, HTTP, TLS Transportschicht: TCP statt UDP)
- Blockieren schwierig da Port *443* HTTPS Standardport

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

1.2.13 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 verwerfen 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 Insepction (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 inspiziert Pakete
  - Konkret **Deep-Packet-Inspection:** Lässt Verbindung laufen und inspiziert gleichzeitig Paketinhalt

Stateless Firewall

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

Stateful Firewall

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

1.2.14 Intrusion Detection Systems (*IDS*)

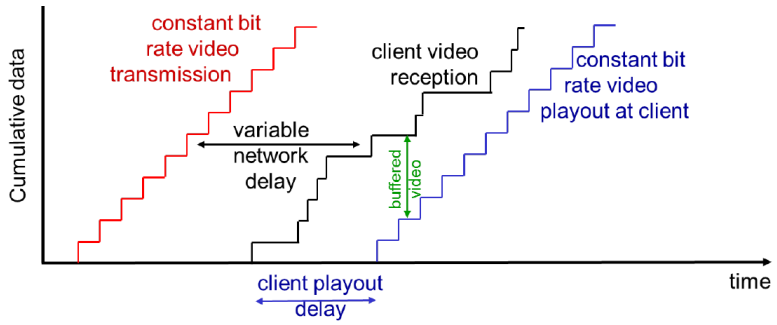
◦ Erkennt Angriffe von außen und ermöglicht (automatisierte) Reaktion
◦ Überprüfen des Paketinhalts auf verdächtige Inhalte (z.B. *SQL Injection*) sowie Korrelation zwischen mehreren Paketen erkennen (*DoS*, *Port Scanning*)

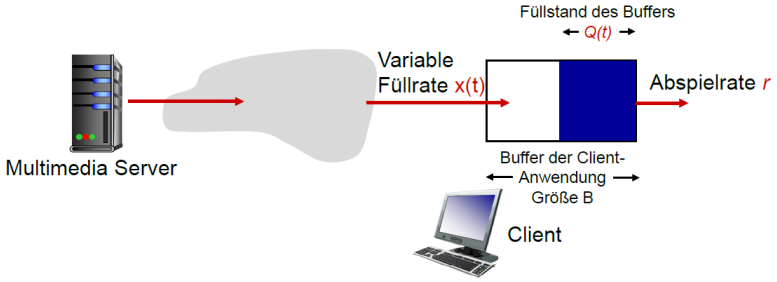
Multimedia

◦ 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

1.3.1 Grundlagen



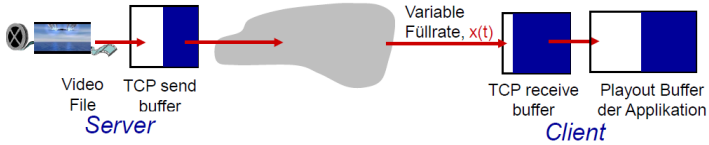


◦ Puffern und verzögertes Abspielen (**playout buffer/delay**)
Kompensation von Verzögerungen (*network delay*)
und Jitter (Schwankungen in der Verzögerung)
◦ Buffern:
- Initiales Befüllen des Buffers bis zum Start des Abspielens (t_p)
- Füllstand $Q(t)$ des Buffers ändert sich über Zeit aufgrund *variabler*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

Streaming über UDP:

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

Streaming über HTTP (TCP):



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

1.3.2 Dynamic Adaptive Streaming over HTTP (DASH)

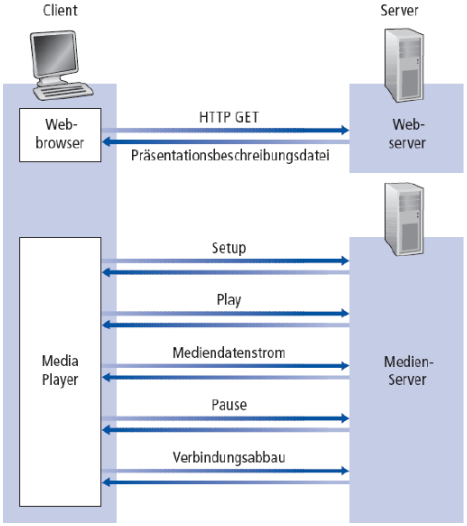
◦ Problem TCP-Streaming: Bandbreite variiert durch HTTP+TCP, *DASH:*

- 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 abhängig von Bandbreite

◦ 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)

1.3.3 Real-Time Streaming Protocol (RTSP)



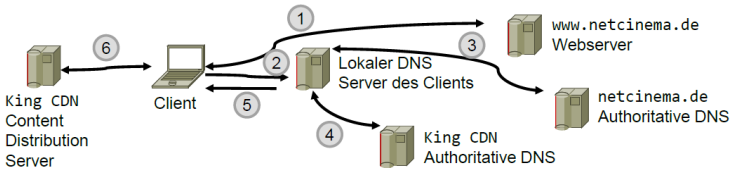
- Client-Server Protokoll auf App.-Layer zur Steuerung des Multimedia-Datenstroms (*Play*, *Pause*, *Fast-Forward*)
- Separate *Kontrollverbindung*, Mediendaten selbst per UDP oder TCP
- *RTSP* spezifiziert **nicht:** Kompression Audio/Video, *wie* Übertragen wird, Pufferverhalten
- *Präsentationsbeschreibungsdatei:*
  $< title > Twister < /title >< session >< grouplanguage = enlipsync >$
  $< switch >< tracktype = audiosrc =' rtsp://audio.example.com/audio.en/lofi' >$
  $< tracktype = audiosrc =' rts://audio.example.com/audio.en/hifi' >< /switch >$
  $< tracktype =' video/jpeg' src = "rtsp://video.example.com/twister/video >$
  $< /group >< /session >$

1.3.4 Content Distribution Networks (CDNs)

Alternative 1: **Ein großer Server**	Alternative 2: **Content Distribution Network**
• Fehleranfällig („single point of failure“)	• Kopien der Inhalte auf mehreren geographisch verteilten („Replica-“)Servern
• Gefahr der Netzüberlast vor Ort	• Privates CDN: Inhaltsanbieter betreibt CDN
• Lange Wege zu entfernten Nutzern	• Third-Party CDN: Anderer Anbieter betreibt CDN
NICHT SKALIERBAR	Beispiel-Anbieter: Akamai, Amazon, NTT Europe

- **Enter Deep:**
  - Große Anzahl Cluster innerhalb der Zugangsnetze der *ISPs*
    Nah beim Endnutzer (geringe Latenz, wenig Hops)
  - Verfolgt Akamai (> 1700 Cluster an verschiedenen Orten)
- **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*
- (2) Klickt auf Video *video.netcinema.de/4711abc*
  → Client sendet *DNS-Request* and *video.netcinema.de*
- (3) Lokaler DNS leitet Anfrage an für *netcinema.de* zuständigen DNS
  → erkennt Video-Anfrage (nutzt *CDN*)
  → liefert keine IP, sondern Hostname aus CDN (z.B. *a42.kingcdn.de*)
- (4) Lokaler DNS sendet Anfrage an *a42.kingcdn.de* an dafür zuständigen DNS-Server (Teil des *CDN*)
  → Entscheidung, welcher Content-Distribution Server liefern soll
  →→ IP des Distribution Server nahe dem User
- (5) Lokaler DNS leitet IP an Client weiter
- (6) Client ruft Video per HTTP GET ab
- mit *DASH* liefert Server *Manifest*-Datei, Client wählt dynamisch



- *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 *pings*)
- *Kankan* in China wählt Peer-to-Peer (ähnlich *BitTorent*) Ansatz für CDN (Hash-Tabelle zum Auffinden der Inhalte)

1.3.5 Voice over IP (VoIP)

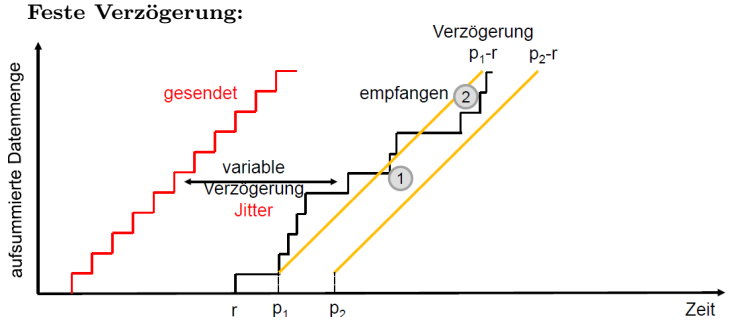
- Digitalisierung der Sprache mit 8000 Bytes/s (64 kbit/s) (*Nyquist-Shannon-Abtasttheorem* ⇒ Rekonstruktion von max. 4kHz)
  Abtastung: Alle 8000Hz eine Messung, *Quantisierung:* Analoger Messwert in 8-bit Digitalwert überführen
- Digitalisierte Sprache per UDP von 160Bytes alle 20ms senden

◦ 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 $q = p_1 - r$: Aussetzer bei (1)

Bei größerer Verzögerung $q = p_2 - r$: kein Aussetzer bei (2)

Abspielverzögerung q

◦ Empfänger spielt Daten mit Verzögerung q ab (Zeit $t \rightarrow$ Abspielen bei $t + q$)

◦ Paketverlust wenn Paket nach $t + q$ ankommt

◦ **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)

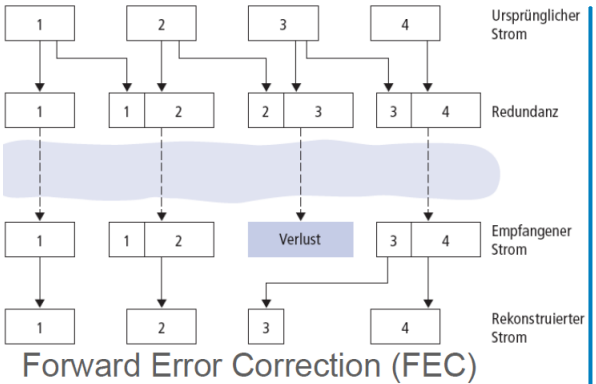
→ Exponentiell gewichteter, gleitender Mittelwert: $d_i = (1 - \alpha)d_{i-1} + \alpha(r_i - t_i)$ mit

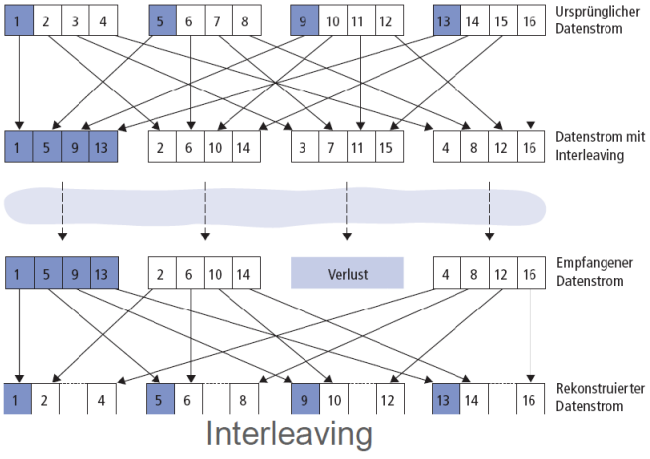
d_i := Schätzung nach Paket i, α := Konstante < 1, r_i := Empfangszeitpunkt Paket i, t_i := Zeitstempel Paket i (Sendezeitpunkt)

→ Standardabweichung $v_i = (1 - \beta)v_{i-1} + \beta|r_i - t_i - d_i|$

→ *zu Beginn* der Redeperiode Berechnung Abspielzeitpunkt $p_i = t_i + d_i + K \cdot v_i$ mit K := positive Ganzzahl-Konstante

⇒ Nur so lange verzögern, wie nötig





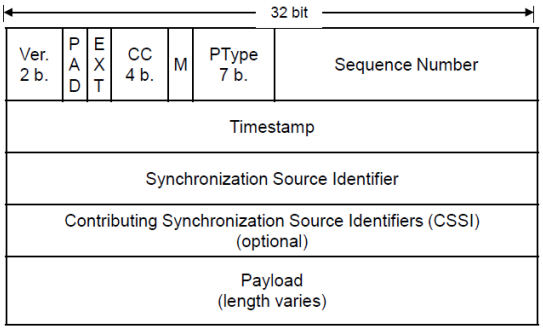
◦ **Forward Error Correction:** Einfügen redundanter Informationen in Nutzdaten, zur Korrektur von Fehlern/Verlusten

◦ **Interleaving:** Umordnen & Verzahnen 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)

1.3.6 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*
  - **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
  - **CSSI-Count**: Anzahl *CSSI*-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 der Multimediadaten



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