HTTP

Grundlagen

- Client Server Architektur
- Request-Response-Schema
- statuslos
- zustandslos
- TCP:80
- textbasiert
- Stop-n-Wait-Protokoll
- **URI**
  - Uniform Ressource Identifier
  - einheitliche Ressourcenbezeichnung
  - Schema://login:password@server.domain:portpath?parameter#fragment
- **URL**
  - Uniform Ressource Locator
  - Unterart der URI
  - Angabe von Zugriffsmechanismus und Ort
- **URN**
  - Uniform Ressource Name
  - dauerhafter, Ortsunabhängiger Bezeichner für eine Ressource

Aufbau

- **Message Type**
  - Typ der Nachricht -> GET,POST,PUT,DELETE...
  - erste Zeile der Nachricht -> GET Request-URI HTTP/1.1 CRLF
- **Header**
  - in HTTP/0.9 und 1.0 komplett optional
  - in HTTP/1.1 ist Host Header pflicht!
  - gibt spezifische Informationen an
  - Syntax: Headername: Wert CRLF
  - Bsp: Authorization, Refer, User-Agent, Encoding, Length, Content-Type
- **Body**
  - enthaelt Daten von POST/PUT Request oder GET Response
  - optional
  - wird durch Leerzeile von Header getrennt
  - Ende durch Length Header markiert

Status-Line (Response)

- in Response Nachricht -> HTTP/1.1 Code Text
- **Code**
  - 1xx Informationen -> zB Anfrage wird noch bearbeitet, weil ...
  - 2xx Success
  - 3xx Redirection
  - 4xx Client Error
  - 5xx Server Error

Neuerungen in 1.1

- Pipelining -> mehrere Anfragen ohne auf Response zu warten
- Persistence -> bestehen bleibende Verbindung nach 1 REQU/RESP durchlauf
  - Connection: keep-alive
  - Keep-Alive: 115
  - Connection: close
  - ohne Header bleibt die Verbindung bestehen
- Caching
- Kompression: gzip

HTTP-Proxy

- Caching
- Zugangskontrolle
- Anonymisierung
- Lastverteilung

Cookies

- verwendet um zustand speichern zu können
- Server generiert Cookie und sendet es im Response an den Client
- Client schickt bei weiteren Requests Cookie mit -> Server sieht, was der Client bisher gemacht hat -> Verbindung zu anderen Webseiten herstellen zur Personalisierung (Werbung)

SPDY

Allgemein

SPDY-Draft-3

- Verbesserung zu HTTP
- Reduktion der Ladezeiten
- keine Änderung an Webseite notwendig

- Header kompression
- unnütze Header nur 1mal übertragen
- SSL/TLS
- Server Push -> Server kann Daten auch ohne Request senden
- fügt Session Layer zu HTTP hinzu

Streams

- parallele Streams über 1 TCP Verbindung
- Reduzierung der SYNs -> Server muss weniger Verbindungen verwalten
- höhere Effizienz der TCP Verbindungen
- Unabhängige, bidirektionale Sequenzen von Frames
- Initialisierung durch Client oder Server
- Priorisierung möglich: 0-7 (Hoch- Niedrig)
- Anzahl begrenzt auf 31bit
- Stream von Client -> ungerade ID
- Stream von Server -> gerade ID
- **Unterteilung in**
  - Kontrollframe
    - C Flag gesetzt
    - Version: 3
    - Type: um Steuerinformationen auszutauschen -> z.B. für Syn, RST
    - Length
    - Daten
  - Datenframes
    - C Flag: 0
    - Stream ID
    - Flags: FIN, COMPRESS...
    - Length
    - Daten- Frames sind binär Kodiert

Flow Control

- als Kontrol Frame mit Typ “Window Update”
- beinhaltet Stream ID und Delta Window Size

Session

- beginnt mit Beendigung der TCP Initialisierung
- asynchrone Konfiguration
- Möglichkeit der Zertifikatsübertragung mit CREDENTIAL Control Frame
- Konfiguration aus ID/Value Paaren
- ID Feld aus 8bit Flags und 24bit ID
- Berechnung der RTT mit Ping

HTTP mit SPDY

- erhalt der HTTP Struktur
- HTTP REQU/RESP in SPDY Streams
- HTTP als Name/Value Paar im Body
- vorangestellter Doppelpunkt vor dem Namen
- Headernamen in lowercase
- gzip Unterstützung ist Pflicht!

Server Push

- Server kann mehrere Response auf 1 Request machen
  - geringere Latenzzeiten
  - ist unidirektionaler Stream
  - wird mittels assoiated-stream-id-Feld mit dem Request assoziiert
  - Parameter werden aus dem assoziierten Request genommen
  - Abweisung mittels RST-STREAM

Compression

- Komprimierung von Name/Value Headerblock zwingend
- Verwendung des gzip aus der zlib
- Datenbereich kann komprimiert werden (Compression Flag)

Email

Allgemein

- elektronischer Nachrichtendienst
- asynchrone Kommunikation
  - Sender/Empfänger müssen nicht gleichzeitig online sein
- 1:n Kommunikation
- Ortsunabhängig
- **Struktur**
  - MUA - Mail User Agent -> Darstellung der Mail, nutzt MTA zum versenden/empfangen (Mutt, Outlook)
  - MTA - Mail Transfer Agent -> Zwischenspeichern und weiterleiten an andere MTA's (fetchmail)
- **Protokolle**
  - SMTP -> versenden vom Client zum Server und Server zu Server
  - IMAP, POP3 -> abholen vom Server zum MUA
- 7 Bit ASCII Format
- Zeilenbasiert (CRLF am Ende jeder Zeiler)

- Format:
  - Header -> Steuerinformationen
    - als Typ:Wert Paar CRLF
    - FROM
    - REPLY TO
    - TO
    - CC
    - BCC
    - Subject
    - Date
    - Comments
    - Keywords
    - Received
    - Return-Path
    - MessageID
    - InReplyTo
    - X-..... -> eigener Header
  - Leerzeile
  - Body -> ende mit CRLF . CRLF

MIME

- Multipurpose Internet Mail Extension
- ermöglicht Media Types in Email
- Layer 6
- Textbasiert
- **Struktur**
  - discrete Media -> nur 1 Medientyp
  - composite Media -> mehrere Mediendaten in einer MIME Nachricht
- **Header**
  - MIME-Version: 1.0
  - Content-Type: multipart/mixed...
  - Content-Transfer-Encoding: 7bit, base84
  - boundary: grenzen der MIME Message
- **Body**
  - Content-Type:
  - Content-Transfer-Encoding
- **Types**
  - angabe der Medientypen
  - text, image, video, audio...

Base64

SMTP

- Protokoll der MTA
- 7bit ASCII
- kommandoorientiert
- TCP:25 und Bytestrom
- Layer 5
- Architektur: Client-Server
- Verfahrensweise: REQU/RESP
- Erweiterung mit ESMTP
  - Zugangskontrolle per Login/Passwort
  - SSL/TLS
  - 8bit MIME
  - abwärtskompatible
- **Befehle**
  - HELO -> Anmelden des Clients auf Server
  - EHLO -> ESMTP Variante des HELO
  - MAIL -> Beginn einer Transaktion, Absenderangabe
  - RCPT -> Empfängeradresse
  - DATA -> Eigentliche Email mit Header, Body und abschließen CRLF . CRLF
  - RSET -> Abbruch der Transaktion
  - VRFY -> Benutzername und Mailboxadresse verifizieren
  - QUIT -> SMTP Beenden
  - SAML -> Send and Mail -> an Adresse und Terminal senden

POP3

- Post Office Protocol
- abrufen der Mails
- Transfer zum MUA
- löschen auf dem Server
- Authentisierung per Login/Passwort
- TCP:110/995(SSL)
- Layer 5
- Client-Server
- REQU/RESP
- **Session**
  - Authentifizierung -> USER/PASS -> unverschlüsselt
  - Transaction state -> Zugriff auf Mailbox und abfragen/löschen der Nachrichten; mit RSET zurücksetzen der Session
  - Update state -> initiiert durch QUIT
    - erst hier erfolgt die Löschung der Nachrichten auf dem Server

        * bei Abbruch der TCP Session erfolgt kein QUIT und damit keine Löschung

- **Befehle**
  - Verbindungsaufbau zum server zB über telnet PortNr
  - USER -> Usernamen oder Benutzerkonto auf dem Server
  - PASS -> Passwort in Klartext
  - STAT -> Status und Anzahl der neuen Emails
  - LIST -> Auflistung der Nummer und Größe jeder Mail
  - RETR -> holt die Mail Nr ... vom Server -> RETR 1
  - DELE -> löscht die Mail Nr ... vom Server -> DELE 1
  - NOOP -> keine Funktion, nur Antworttest, Zeitüberbrückung
  - QUIT -> beendet Sitzung
  - RSET -> Reset, Löschungen werden rückgängig gemacht
  - APOP -> Authentifizierung via Challenge
  - TOP -> Header und x Zeilen der Mail Nr....
- **APOP**
  - kein Verbindungsaufbau sendet der Server einen TimeStamp mit
  - MD5(ServerTimeStamp + Passwort)
  - Client: APOP Username Digest zur Authentifizierung

IMAP

- Internet Message Access Protocol / Interactive Mail Access Protocol (alte Bezeichnung bis Draft3)
- Manipulation auf Mailbox im Server -> kein lokales Speichern
- Authentisierung per Login/Password
- TCP:143/993
- Layer 5
- Client-Server
- REQU/RESP und REQU/Muliple-RESP

Vergleich POP vs IMAP

- **Gemeinsamkeiten**
  - Server läuft ständig
  - Zugriff via Internet
- **Vorteil POP3**
  - einfach Implementierung
  - geringe Anforderung an Server
- **Vorteil IMAP**
  - Speicherung/Archivierung auf Server
  - Offline-Verfügbarkeit durch Kopie vom Server
  - Server speichert Zustand der Nachricht
  - Auswahl der Teile, die herunter geladen werden sollen (nur Header)

Sicherheit

- Transportsicherheit -> TLS/SSL für MTA und MUA
- SPAM-Schutz -> sinvoll schwere Adresse, fehlerfreie Software, Adresse sinnvoll weitergeben
- Empfängerschutz -> BCC nutzen
- Integrität und Authentizität via Signatur mit Private Key
- Vertraulichkeit -> Verschlüsselung, zB PGP
- **PGP**
  - Pretty Good Privacy -> Dezentrales Verfahren mit "Web of Trust"
- **X.509c**
  - zentraler Ansatz mit Public Key Infrastruktur
- Listing in verschiedene Gruppen:
  - Blacklist: immer ablehnen
  - Whitelist: immer annehmen
  - Graylist: unsichere Herkunft
- Routing innerhalb der Unternehmensserver
- Überprüfung auf Server ob User existiert und ggf löschen
- Lastverteilung und Ausfallsicherheit mit mehreren Servern und DNS Prio

Base16

- aus 1 Byte werden 2 Byte
- Zeichenvorrat von 16 (0-9,A-F)
- Verdopplung des Speicherbedarfs
- ergänzen mit 4 führenden Nullen

Base64

- aus 3 Byte werden 4 Byte
- es müssen immer 3 Byte kodiert werden und 4 Byte entstehen (wenn nötig mit Paddern auffüllen)
- 8Bit block nehmen
- zwei führende Nullen, dann die 6 höchsten Bits ergeben neuen Wert
- wieder 2 führende Nullen + Rest des vorhergehenden Bytestrom + Folgebytestrom = 8 Bit
- usw bis alles kodiert
- am Ende mit Nullen auffüllen bei nicht vollen Bitfolgen
- leere 8Bit Blöcke werden durch = dargestellt

DNS

Allgemein

- Domain Name System

- TCP/UDP :53
- UDP bei Query <512kB
- sollte die länge Überschritten werden, dann wird TC Flag gesetzt und alle weiteren Nachrichten via TCP gesendet
- TCP Nachrichten enthalten 2 Byte Feld zu beginn mit Länge der gesamten Nachricht enthält
- unverschlüsselte Übertragung
- auflösen von IP zu Hostnamen und umgekehrt
- hierarchischer Aufbau mit Teile und Herrsche Prinzip
  - Aufteilung in kleiner werden Zonen -> Lastverteilung
- Grund für Hostnamen: leichter zu merken; Namensraum um Zonen zu bilden -> everdown.de
- früher nur: /etc/hosts
  - per Hand jede IP eintragen, auf jeden Rechner
  - austausch der Host-Dateien
  - sehr aufwendig bei akuteller Internetgröße :)
- **Komponenten**
  - Nameserver -> Rechner mit Namensdatenbank
  - Resolver -> anfragende Software eines Rechners
- **Hierarchie**
  - Root (Darstellung als .): zentral verwaltete globale Server
  - TLD (de, uk, arpa, mil, xxx) -> Unterteilung in Länder, Organisationen und andere Gruppierungen

Auflösungsverfahren

- **Iterativ**
  - anfrage an Nameserver
  - dieser liefert die Adresse des Hosts oder des nächst besseren Nameservers
  - danach muss Client den nächsten Server anfragen
  - Vorteil: weniger Last für Server
  - Nachteil: mehr Last für Client
- **Rekursiv**
  - anfrage an den Nameserver durch client
  - hat der Nameserver keinen Eintrag fragt er selbst beim nächst höheren Nameserver nach
  - Req/Resp laufen den Strang rekursiv ab (Antwort geht auch über jeden Nameserver)

FQDN

- Fully Qualified Domain Name
- komplette Darstellung der Ressourcen in hierarchischer Reihenfolge endent mit .
- wird von rechts nach links gelesen -> Root.TLD.SD.Zone.Dienst

- maximal 255 Zeichen (‘A-Z’, ‘a-z’. ‘0-9’, Minus, Unterstrich)
- einzelner Abschnitt = Label
  - 1-63 Zeichen lang
  - beginnt mit Buchstaben
  - endet nie mit - oder _

Zonendatei und RR

- Zonendatei beinhaltet Hosts und Nameserver für die Zone
- beginn mit Globaler Konfiguration
  - globale TTL -> $TTL 300
  - ’@ IN SOA ns1.bsp.de. admin.bsp.de.(globale config)
    - @ für den Ursprung also bsp.de
    - SOA -> Start of Authority
    - ns1... -> Master NS der Zone
    - admi... Email ohne @
- RR -> Resource Record -> Eintrag in der Datei
- Aufbau: ‘ ’
  - Name: Ressource die angesprochen werden soll
  - TTL: optionales Time To Live in sec
  - Class: meist IN für Internet?
  - Type: Art des Eintrages -> NS, MX, A, AAAA, CNAME, PTR
  - RDATA: Adresse auf die verwiesen wird (IP, FQDN’s)

Servertypen

- **Primary Server**
  - beinhaltet die Zonendatei, die alle Server verwenden
  - manuelle Pflege der Zonendatei
  - beantwortung von DNS Anfragen
- **Secondary Server**
  - Zonedatei wird über Sync vom Primary bezogen
  - regelmäßiger automatischer Sync anhand der Timer der SOA
  - beantworten von DNS Anfragen
- **Caching Server**
  - zwischenspeichern bereits getätigter Responses -> Entlastung anderer Server und schnellere Bearbeitung von Anfragen
  - positive sowie negative Responses werden gespeichert
  - Speicherdauer begrenzt auf TTL in sec
- **Forwarder/Client Server**
  - agieren nie mit Root Server

Aufbau DNS Nachricht

- Header
  - ID
  - QR Flag (Query/Request)
  - TC Flag (TrunCation lange Nachricht)
  - Anzahl der Questions, Antwort RR, Authority RR, Additional RR
- Question
- Answer
- Authority
- Additional

Sicherheit

- Spoofing -> Umleitung auf andere Adressen
- Host Datei manipulieren
- DDOS auf DNS
- Cache Poisoning -> falsche Daten in Cache einbringen (Resp mit falschen Daten ins Netzwerk bringen)
- einbringen von Sicherheit mittels asymmetrischer Verschlüsselung (public key wird mitgeschickt, private key zur Signierung)

autoconfig

- ohne manuelle Eingabe von Daten
- eigenständige Installation von Host Systemen
- für Unternehmensnetzwerke, diskless Nodes, Mobile Devices

RARP

- Reverse Address Resolution Protocol
- IP aus MAC erhalten
- HOST erfragt seine eigene IP
- Server sendet die IP
- Gegensatz zur ARP:
  - ARP wird die MAC zu einer zugehörigen IP eines anderen Gerätes erfragt
- Nachteil:
  - begrenzt auf Broadcast Domäne
  - keine Subnetzmaske
  - kein Routing
  - kein DNS-Server

BootP

- Bootstrap Protocol
- Vorgänger von DHCP
- UDP (Port 68 Client, 67 Server)
- liefert IP, SN, GW. DNS. Bootdatei
- für jede L2 Adresse muss IP manuell vorkonfiguriert werden

DHCP

- Dynamic Host Configuration Protocol
- UDP (68 Client, 67 Server)
- Ablauf:
  - client DISCOVER als BC ins Netz
  - server OFFER als BC
  - client wählt IP aus
  - client REQUEST mit ausgewählter IP und Server ID als BC
  - Server werten ID aus und nehmen Absage/Annahme an
  - Server senden ACK/NACK für IP des CLients
  - Client prüft mit ARP die Verfügbarkeit der IP
  - antwortet ein Host, dann schickt Client ein DECLINE an den Server
- **Relay Agent**
  - übermittel DHCP Nachrichten in ein anderes Subnetz
  - ermöglicht den Betrieb eines DHCP Servers für mehrere Subnetze
- **Scope**
  - stellt einen DHCP Bereich da
  - wird im Server konfiguriert
- **lease**
  - stellt die Gültigkeitsdauer da, in der ein Gerät die zugewiesene Adresse nutzen kann
  - danach wird sie RELEASE zurückgegeben
  - mit REQUEST erneut angefordert

IPv6 Stateless Address Autoconf

- NDP -> Neighbor Discovery Protocol
- SLAAC
  - link local adresse bilden
  - ROuter Solicitation auf Multicast Adresse
  - Router Advertisement mit Präfix Informationen
  - Generierung neuer Adressen aus den Präfixen
  - Überprüfen mit Duplicate Address Detection
- Nachbarrouter ermitteln (ICMP 133 Router Solicitation)
- Link Layer Adresse des Nachbarn ermitteln (Ersatz für ARP)

- Neighbor Cache
- Duplicate Address Detection (ermitteln ob Adresse bereits verwendet wird)
- Neighbor Unreachable Detection

DHCPv6

- dynamische Zuweisung der Netzconfig
- UDP (Client 546, Server 547)
- **Ablauf**
  - link local generieren (Unterschied zu DHCP)
  - SOLICIT an Multicast FF02::1:2
  - Server senden ADVERTISE
  - Client schickt an 1 Server einen REQUEST für Parameter
  - Server sendet REPLY
- **Wiederverwenden**
  - Client schickt RENEW
  - Server REPLY mit Konfigparameter
- **Gemeinsamkeiten zu DHCP**
  - automatische Konfiguration
  - lease und lease time identisch
  - Relay Agents werden benötigt
- **Unterschiede**
  - RECONFIGURE
    - Server sendet Information, dass der Client eine neue Adresse anfordern muss wegen NEUKONFIG
  - CONFIRM
    - Client kann die Gültigkeit seiner Adressen überprüfen lassen
  - Authentifizierung von DHCPv6 Nachrichten möglich
    - keine unauthorisierten DHCP Server
    - nur authorisierte Clients werden bedient
    - zw. Relay und Server wird IPSec genutzt
    - Schutz der Daten mit Authentication-Option

Automatische Dienstkonfiguration

- über Well-Known-Ports, DNS, DHCP, BC, MC, AC
- **DNS:** Dienst.Transportprotokoll.Domaene TTL CLASS PRIO WEIGHT PORTNR ADRESSE(KEIN ALIAS)

- **DHCP:** spezielle DHCP anfrage via BC oder MC
- PXE
  - Booten, PXE
  - DHCP (Netzkonfig, Filename, TFPT
  - file laden
  - Image prüfen
  - Image booten
  - mounten
- UPnP

TFTP

- **nur** Datentransfer
- UDP
- Paketorientiert

Transportprotokolle

TCP

- Transmission Control Protocol
- verbindungsorientiert, Duplex, E2E, L4
- genutzt von: HTTP, Mail, SFTP etc
- **HEADER**
  - SRC Port 16bit
  - DEST Port 16bit
  - SQNr 32bit
  - ACKNr 32bit
  - Offset 3bit
  - Reserved 10bit
  - FLAGS 6bit
    - URG: dringende Daten vorhanden/bevorzugte Behandlung
    - ACK: zeigt an, dass die ACKNr beachtet werden soll
    - PSH: übergehen des Puffers
    - RST; Abbruch der Verbindung
    - SYN: neue Verbindungsanfrage
    - FIN: Zeigt Ende der Verbindung an
  - Window 16bit
    - zeigt die Anzahl der empfangbaren Datenoktette
  - Checksum 16bit

  - Urgent Pointer 16bit Zeigt auf dringende Daten (FLAG!)
  - Options 0-40B
  - Padding füllt Header auf 32bit grenze auf

- **PORTS**
  - stellen Verbindung zu Anwendungen höherer Schichten dar
  - 0-1023: Well-Known-Ports
  - 1023 dynamische Zuweisung an Software
  - Dienst/Port Zuweisung bei TCP und UDP gleich aber nicht immer von Beiden verwendet

- jedes Paket wird mit ACK vom Empfänger bestätigt -> langsam und Zeit/Kapzität wird verschwendet (Halbduplex)

- **Sliding Window**
  - Sendefenster gibt Anzahl der Datenpakete an(ohne ACK))
  - mehrere Segmente ohne ACK senden
  - Daten und ACK gleichzeitig (vollduplex)
  - ACK kann im Datensegment übertragen werden
  - Nachteil: ab fehlerhaften Segment werden die Daten neu übertragen

- **Fast Retransmit**
  - Empfänger besteht immer das zuletzt erwartet *richtige* Paket
  - erhält Sender 3mal die gleiche ACKNr sendet er das fehlende Paket nach
  - Nachteil: bei größerer Fehlersequenz muss für jedes Packet 3 ACKs abgewartet werden

- **Selective ACK Options**
  - SACK Permitted Option beim Verbindungsaufbau
  - Gruppierung der ACK in "von bis" Gruppen mit den Sequenznummern

- **Congestion Control**
  - Vermeidung von Überlast ("Stau")
  - Steuerung durch den Sender
  - mittels Slow Start –> langsam vergrößern des *congestion window* mit jedem RTT inkrementieren
    * empfang von dACK Window wird halbiert
    * time out -> Slow Start wird neu begonnen
  - Nagle Algorithmus
    * kleine Segmente brauchen jedes mal ein ACK
    * große Segmente können ohne vorheriges ACK gesendet werden
    * verhindert Überflutung mit Header bei vielen kleinen Paketen
    * Nachteil: Anwendungen mit kleinen Paketen werden behindert (telnet ssh)
  - Silly Window Syndrome

        * verhindern von vielen ACK Paketen aufgrund sehr kleiner TCP Segmente
        * ACK wird gesendet, wenn 1 MSS(Maximum Segment Size) oder halber Empfangspuffer erreicht ist
        * Sender sendet, wenn 1MSS erreicht ist oder kein ACK erwartet wird

- **Flow Control**
    - Empfänger teilt Sender die freie Größe des Buffers mit
    - Signalisierung im Window-Feld (TCP Header)
    - bei Window == 0 werden 1 Byte Daten gesendet bis Buffer wieder leer
    - Persist Time wird bei jeder *Zero Probe* verdoppelt bis maximal 60sec

- **Sicherheit**
    - SYN FLOOD
        * viele Anfragen mit ggf falscher Senderadresse –> Speicherreservierung für SYN Anfragen –> kein Verbindung mehr möglich (DOS)
        * Vermeidung: Verwendung eines SYN Cookie –> IP, Port, TimeStamp etc
    - Spoofing
        * einbringen falscher Pakete durch erraten der Sequenznummern
        * Vermeidung: MD5, Timestamp, oder andere Protokolle wie SCTP und DCCP
    - Fingerprint
        * Betriebssystemermittlung aus TCP Daten mit nmap
        * für Angriffe auf das OS
        * Vermeidung: Fingerprint verwischen/fälschen

UDP

- verbindungslos
- **Header:**
    - Source Port 16bit
    - Dest Port 16bit
    - Length 16bit
    - Checksum 16bit
- **Nachteile:**
    - ungesicherte Übertragung
    - keine Flusskontrolle
    - kein Congestion Control
- **Vorteile:**
    - schnelle Übermittlung, da kein Verbindungsaufbau
    - wenig Overhead
    - kein Retransmit (Echtzeitanwendung)

SCTP

- Stream Control Transmission Protocol
- verbindungsorientiert
- unterstützt Multihoming
- Common Header
  - SRC Port 16bit
  - DEST Port 16 bit
  - Verification Tag 32bit(Zufallszahl die zu beginn ausgetauscht wird und bei jeder Msg enthalten sein muss)
  - Checksumme 32bit
- Chunk Header (1 bis n mal vorhanden)
  - Type 8bit
  - Flags 8bit
  - Length 16bit -> 4Byte Header + Value
  - Value 32bit
- Assoziation
  - SCTP Verbindung wird *Assoziation* genannt
  - Charakterisiert durch:
    - SRC/DEST Port/IP
    - mehrere IPs aber nur ein Port Paar
  - etabliert durch *four way handshake*
    - INIT(VTAG)
    - INIT ACK(VTAG, Cookie)
    - Cookie Echoed
    - Cookie ACK
  - shutdown durch *Three way handshake*
    - SHUTDOWN
    - SHUTDOWN ACK
    - SHUTDOWN COMPLETE
  - Abbruch mit *abort chunk* (TCP-Reset)
- Stream
  - unidirektionale Verbindung
  - Festlegung der Streamanzahl beim Aufbau
  - Streams einer Assoziation beeinflussen einander *NICHT*!!!
  - Multistreaming möglich
- Multihoming
  - Anbindung über n verschiedene IPs möglich
  - Festlegung eines primären Pfades
  - Alternative Pfade werden bei Datenverlust genutzt sowie auf USER/Programm Befehl
  - Stream ACK SACK über den Streamkanal
- HEARTBEAT

  - periodisches Überprüfen des Pfadzustandes
  - inkrementieren eines Path Max Retrans bis zu *unerreichbar* Wert
  - nur auf inaktiven Pfaden (kein Datenstream)
- Flow/Congestion Control
  - wie bei TCP mit Sliding Window und Receiver Window
  - Slow Start
  - Separat für jeden Stream

DCCP

- Data Congestion Control Protocol
- verbindungsorientiert
- Transfer unsicher (ACK ohne Retransmit)
- Echtzeitdaten
- Congestion Control

Vergleich

Merkmal	**SCTP**	**TCP**	**UDP**	**DCCP**
Vollduplex	OK	OK	OK	OK
Verbindungsorientiert	OK	OK	X	OK
sichere Übertragung	OK	OK	X	X
geordnete Übertragung	OK	ok	X	X
ungeordnete Übertragung	OK	X	OK	OK
Flow Control	OK	OK	X	X
Congestion Control	OK	OK	X	OK
Selective ACKs	OK	optional	X	OK
Paketorientiert	OK	X	OK	OK
Path MTU Discovery	OK	OK	X	OK
PDU Fragmentierung	OK	OK	X	X
PDU Bündelung	OK	OK	X	X
Multistreaming	OK	X	X	X
Schutz gegen Syn Flooding	OK	optional	-	OK
Fehlererkennung	OK	OK	OK	OK
Pseudo-Header für Checksumme	X	OK	OK	OK

AAA

Allgemein

- Authentication -> Nutzer- und Geräte-Authentifizierung
  - durch Passwörter, Zertifikate, Smartcard und Kombinationen
- Authorisation -> Verwaltung von Zugriffen auf Dienste und Ressourcen
  - durch ACL (Access Control List) und Policies
- Accounting -> Abbrechnung von Dienstnutzungen & Ressourcenplanung
- Authenticator -> System, dass die Zugangskontrolle verwaltet (RADIUS)
- Supplicant -> Nutzer/System das Zugriff erhalten möchte
- NAS -> Network Access Server
  - Zugangskontrollsystem für Ressourcen und Dienste (WLAN-AP)
- Policie -> wer(Nutzer/Gruppe) darf was(Ressourcen/Dienste) wann(Zeit/Zutrittsbedingung) wielange(Zeit/Volumenbegrenzung) benutzen. . . .
- ACL -> Rechtevergabe für Ressourcen und Dienste
- **Komponenten**
  - Rule Based Engine -> Trennen von generischen und Dienstspezifischen Informationen
  - Application Specific Module -> Anpassung an konkrete Anwendung
    - Ressourcenverwaltung
    - Dienst- und Servicekonfiguration
    - Einfluss auf Autorisisierung mittels spezifischer Informationen
  - Eventlog
    - Speichern von Timestamps und Events
    - Nutzung dieser um Autorisierung für zeitliche beschränkte Zugriffe oder Event-basierte Zugriffe (Formulare ausgefüllt, Feueralarm)
  - Policie Repository
    - Datenbank für Dienst und Ressourcen
    - Zuweisung zu Namensraum
  - Service Equipment
    - Hardware mit bestimmten Service
    - von ASM gesteuert
- **Modelle**
  - Systemaufteilung:
    - Single Domain -> Service und AAA auf einem System/Gerät
    - Roaming -> Service und AAA auf verschiedenen Systemen/Geräte

  - Zugriff auf Dienste:
    * Agent Sequenz -> AAA funktioniert als Vermittler, alle Anfragen laufen über diesen
    * Pull Sequenz -> Nutzer interagiert mit Service Euipment, Service erfragt Authorisierung beim AAA
    * Push Sequenz -> Nutzer authentifiziert sich beim AAA und erhält Ticket/Zertifikat mit dem er auf den Service zugreifen kann

- **Sessionmanagement**
  - Verwendung von SessionIDs
  - gleiche SID über mehrere Server
  - Session Ende muss Service Equipment mitgeteilt werden

RADIUS

- Remote Authentication Dial In User Service
- UDP
- Client Server Modell
- speichert Nutzerdaten
- **Header**
  - Code 8bit -> Nachrichtentypen
    * Access (Request, Response, Reject, Challenge)
    * Accounting (Request, Response)
  - Identifier 8bit -> Zuordnung von Request/Response
  - Length 16bit -> Länge des kompletten Packets
  - Authenticator 16Byte -> zum Verifizieren (Zufallswert und MD5) zwischen Client/NAS und Server
  - Attributes
    * Nutzdaten (AAA Daten)
    * TLV Format
- **Authentification**
  - PAP
    * *Password Authentication Protocol*
    * Klartext
    * über das PPP Verfahren
    * Signalisierung für PAP, dass USER Name und PW gleich übertragen werden
  - CHAP
    * *Challenge Handshake Authentication Protocol*
    * Client zu NAS REQU
    * NAS schickt RESP mit Challenge

    - Client zu NAS: REQU mit CHAP-ID (Zufallswert,ID,Passwort mit MD5) und USER Name
    - NAS zu RADIUS: Access REQU (CHAP PW, UserName, CHAP Challenge)
    - RADIUS: Accept, Reject zu NAS
  - EAP -> Extensible Authentication Protocol

- **Accounting**
  - Accounting wird nach Authentifizierung gestartet von NAS
  - ACC Requ mit Status Type(Start) und Session-ID
  - Resp als Bestätigung
  - REQU Type(Interim Update, SID, Acc Daten) -> Übermittlung von Abbrechnungsdaten während der Nutzung
  - REQU(STOP, SID, Acc-Daten) -> Beenden und Übermittlung der gesammelten Daten

- **Proxy**
  - zwischengeschalteter Proxy der als Vermittler fungiert
  - fügt/entfernt Proxy Attribute

DIAMETER

- Nachfolger von RADIUS
- TCP, SCTP
- Peer to Peer Protocol (jeder kann Nachrichten senden)
- Hop by Hop und End to End Security
- **Base Protocol Funktionen**
  - Übertragung Attribut Value Paare
  - Fähigkeiten aushandeln
  - Fehler Erkennung
  - Basic Services (Session Handling, Accounting, Sicherheit, Proxy)

- **Header**
  - Version 8bit -> zwangsweise 1
  - Msg Length 24bit -> Länge Header + Body
  - Flags 8bit
    - REQU/RESP -> 1/0
    - Proxyable -> Proxy, Redirect, Relay Agent erlauben
    - Error Bit -> Protokollfehler anzeigen
    - T-Bit -> retransmit bei failover ??
    - reserve 4bit
  - Command Code 24bit -> Kodierter Typ für einzelne Aktionen
  - Application ID 32bit -> Zuordnung auf Anwendung

  - Hop by Hop ID 32bit -> wird durch Proxy etc geändert
  - End to End ID 32bit -> eindeutig bis zum bittere ende

- **Peer Connection**
  - zwischen 2 Peers genau eine permanente TCP oder SCTP Verbindung
  - mehrere Sessions möglich

- **Capabilities Exchange**
  - erfolgt nach Verbindungsaufbau
  - Identitäten der anderen Peers feststellen
  - austausch der Fähigkeiten: Protokollversion, Diameter Applicationen, Sicherheitsmechanismen
  - Verbindungsaufbau wird abgebrochen, falls Applicationen und Sicherheitsmechanismen nicht unterstützt werden

- **Watchdog**
  - Heartbeat zur aktiven Überwachung der Connectivity
  - mittels Device Watchdog Request DWR
  - bleibt Device Watchdog Answer DWA aus, wird auf secondary Peer umgeschaltet
  - DWR Zeitintervall wird manuell festgelegt

- **Agenten**
  - Relay Agent -> routen von Diameter Nachrichten zwischen Peers
  - Proxy -> routen und verändern anhand von Policies
  - Redirect Agent -> zentraler Informationsverwalter bzgl Routinginformationen
  - Translation Agent -> zusammenführen verschiedener Applicationen und Protokolle, z.B. Datenbanken zweier Domaenen mit ihren AAA oder RADIUS/Diameter

Vergleich Radius mit Diameter

Eigenschaft	**Radius**	**Diameter**
Header	8 bit	32 bit
Attribut Laenge	8 bit	24 bit
Flusssteuerung	UDP-keine	TCP, SCTP
Architektur	Client-Server	Peer
nicht lesbare Nachricht	verwerfen	Error-FLag
Server Status	keine Info	Watchdog
Vendor Codes	X	OK
Failover	X	OK
Roaming	OK	OK
Sicherheit	Hop, Shared Secret	Hop, End, IPSec, TLS

EAP

- Extensible Authentication Protocol
- Authentication Framework auf Layer 2(Ethernet, WLAN, PPP) -> da Supplicant an Netzzugangspunkten noch keine IP besitzt
- Funktionsprimitive:
  - Datenübertragung (Timer, Retransmission)
  - Nutzer Identifikation
  - Auswahl Authentifizierungsverfahren
- **Header**
  - Code 8bit -> Nachrichten Msg (Requ, Resp, Success, Failure)
  - Identifier 8bit
  - Length 32bit
  - Type 8bit -> gibt die Unterabfrage an, was gerade gefordert/geschickt wird (Identity, Notification, NACK, MD5 Challenge...)
- **Authentifizierung**
  - Supplicant startet mit EAPOL-Start
  - Authenticator erfragt Identitaet (Nutzernamen) -> Type 1
  - Supplicant schickt Identitaet
  - Authenticator startet Authentifizierung und schickt MD5 Challenge
  - Supplicant erfüllt Challenge oder schickt NACK mit anderem Verfahrensvorschlag
  - Authenticator wählt aus und startet Authentifizierung erneut
- **PEAP**
  - Protected EAP
  - TLS Tunnel zu beginn -> Identitaet wird geschützt
  - danach EAP Ablauf
- **EAP-TLS**
  - Zertifikatsbasierte Authentifizierung
  - kein Identitätsschutz
- **EAP-TTLS**
  - EAP -Tunneling TLS
  - Authentifizierung erfolgt dann mit EAP oder PAP/CHAP/MSCHAP
  - bei Verwendung von EAP ist es gleich dem PEAP

IEEE 802.1X

- zur Authentifizierung in Layer 2 Netzen
- portbasierte Authentifizierung
- EAP Authentifizierung vorgeschrieben
- **Sicherheitsanforderungen**
  - Integrität
  - Replayschutz

  - Wörterbuchattacken
  - Session Unabhängigkeit
  - uvm.
- **Port Access Entity**
- Authentifizierung am Netzwerkport
- besitzt Lower/Upper Layer
- Supplicant/Authenticator auf upper layer
- Port Access Control Protocol -> Verbindung zum upper Layer
- **Port-Types**
  - uncontrolled Port -> empfangen und senden von Auth.-Frames
  - controlled Port
    * Zutrittskontrolle durch PAE im uncontrolled Port
    * Stati -> authorized(802.1X deaktiviert), unauthorized(Port deaktiviert), auto(PAE)

Sockets

- standardisierte Schnittstelle zwischen Anwendung und Transportschicht
- aus BSD
- Kommunikation immer mit Socket Paar (jede Anwendung hat eigenen Socket)

Arten

- **Datagram Socket**
  - verbindungslos, Paketorientiert
  - keine Verbindungsaufbau/abbau
  - UDP, DCCP, ICMP
  - Reihenfolge und Fehlerkorrektur durch Anwendung
  - **Ablauf Server**
    * socket
    * bind
    * recvfrom
    * sendto
    * close
  - **Ablauf Client**
    * socket()
    * sendto
    * recvfrom
    * close
- **streaming Socket**
  - verbindungsorientiert, Bytestrom
  - Auf-/Abbau der Verbindung

  - Daten nach Verbindungsaufbau
  - Fehlerbehandlung, Reihenfolge in Tansportschicht
  - TCP, SCTP

Funktionen für Ablauf

- **socket erzeugen**
  - int socket(int family, int type, int protocol)
  - family: Protokollfamilie Layer 4 (IP, IPv6, IPX, RAW Socket...)
  - type: Socketart -> `SOCK_STREAM oder SOCK_DGRAM`
  - protocol: welches Transportprotokol -> `IPPROTO_TCP, IPPROTO_UDP...`
- **connect() für StreamSockets**
  - `int connect(int sockfd, struct sockaddr* serv_addr, int addrlen)`
  - sockfd: Deskriptor aus socket()
  - servaddr: Adresse mit der sich verbunden wird
  - addrlen: länge des struct sockaddr
- **bind() des Servers**
  - `int bind(int sockfd, struct sockaddre *myaddr, int addrlen)`
- **listen()**
  - warteschlange für die eingehenden Verbindungen
  - int listen(int sockfd, int backlog);
  - backlog: länge der wateschlange
- **accept()**
  - `int accept(int sockfd, struct sockaddr *addr, int addrlen)`
  - addr: addresse des clients
  - wenn liste leer, dann blockiert accept()
- **write() send()**
  - benötigen connect
  - senden kann blockieren
- **sendto()**
  - sendet an adresse
  - für Datagram
  - blockierend
- **read(), recv()**
  - benötigt vorhergehende Bindung an Adresse

Optionen

- `int setsockop(int sockfd, int level, int optname, const void *optval, socklen optlen)`
- level: Schicht -> `IPPROTO, IPPROTOV6`
- optname: Option die gesetzt wird
- optval: Wert der Option
- optlen: länge des Wertes
- Optionen:
  - `SO_REUSEADDR` -> wiederverwenden einer aktuellen Adresse
  - `SO_BROADCAST` -> Datagram an Broadcast senden
  - `SO_RCVBUF` -> Empfangsbuffergröße
  - `SO_SNDBUF` -> Sendebuffergröße

Blocking / Non-Blocking

Also darüber kannst du mal schön selbst nachdenken :p