

Einführung

WAS IST DAS INTERNET?

Komponentensicht

1. *Computer* – führen Netzwerkanwendungen aus
2. *Kommunikationsmedien* – Kupferkabel, Glasfaser, Funk
3. *Zwischensysteme* – Weiterleitung durch Router und Switches

Dienstsicht

- ⇒ Infrastruktur, die Dienste bereitstellt
- *Kommunikation* (Mail, Messaging, soziale Medien)
- *Information* (Surfen)
- *Unterhaltung* (Streaming, Spiele)

RAND DES INTERNET

Geräte

- Clients
- Server

Zugangsnetze

- Heimnetz
- Mobiles Zugangsnetz
- Unternehmensnetz

KERN DES INTERNET

Pakete: voneinander unabhängige Einheiten für die Weiterleitung – werden durch Netz zur Zielanwendung geleitet

Anwendungsschicht

HISTORIE

70er/80er:

- textbasierte Anwendungen

90er:

- World Wide Web
- Instant Messaging
- P2P-Filesharing

seit 2000: steigende Vielfalt + Allgegenwärtigkeit

- Streaming (Spotify, YouTube)
- Gaming
- Soziale Netzwerke
- Smartphones

GRUNDLAGEN — SCHICHTENMODELL

Kommunikation in Schichten organisiert

Anwendungsschicht: oberste Schicht

- enthält Anwendungsprotokolle
- Anwendung kümmert sich nicht um Datentransport

Datentransport: unter Anwendungsschicht liegende Schichten

- Intern für Anwendung transparent
- Verzögerungen bleiben vor Anwendung verborgen

GRUNDLAGEN — VERZÖGERUNG

Abhängig von

- Ausbreitungsverzögerung t_a
- Sendezeit t_s
- Pufferfüllstände

Ausbreitungsverzögerung $t_a = \frac{d}{v}$

- Zeitspanne zwischen Absenden eines Signals und dessen Eintreffen am anderen Ende des Mediums
- Abhängig von:
 - Ausbreitungsgeschwindigkeit v
 - Länge des Mediums d

Sendezeit $t_s = \frac{X}{r}$

- Zeit zwischen Beginn und Abschluss der Sendung
- Abhängig von:
 - Datenmenge X
 - Datenrate des Mediums r
- **Achtung:** Nach Sendungsabschluss sind die Daten noch nicht beim Empfänger!
 - ~> Ausbreitungsverzögerung t_a

Verzögerung im Router

- Pufferung der Daten in Warteschlange
- Verarbeitung (Fehlerüberprüfung usw.)

GRUNDLAGEN — PROTOKOLLSTACK

Application: SMTP, HTTP, XMPP,...

Transport: TCP, UDP

Network: IP

Data Link: Ethernet, 802.11 (WiFi)

Physical: Bits auf Medium

GRUNDLAGEN — PROZESS UND NACHRICHT

Prozess: Programm, das im Endsystem (Anwendungsschicht) abläuft

Nachricht: Ausgetauscht zwischen Prozessen auf *unterschiedlichen* Endsystemen

GRUNDLAGEN — SOCKET UND INTERFACE

Programmierschnittstelle für verteilte Anwendungen

Von OS bereitgestellte API

Anwendungsprozess sendet/empfangt Nachrichten zum/vom Socket

Portnummern: (De-) Multiplexing auf Endsystemen

- viele Prozesse auf Endsystem kommunizieren gleichzeitig über Netzwerk
 - ~> eindeutige Socket-Identifikation über Portnummer

GRUNDLAGEN — CLIENT-SERVER-ANWENDUNGEN

Server:

- ständig in Betrieb
- permanente IP-Adresse
- häufig in Datenzentren

Clients:

- kommunizieren mit Server
- kommunizieren *nicht* direkt miteinander
- evtl. nicht immer verbunden
- evtl. dynamische IP-Adresse

GRUNDLAGEN — PEER-TO-PEER-ANWENDUNGEN

Endsysteme kommunizieren direkt miteinander

- fordern Dienste von anderen Peers an
- nicht permanent verbunden, wechseln dynamisch IP-Adressen
 - ~> komplexes Management

selbst-skalierend

- neue Peers erhöhen Kapazität, fordern aber auch selber Dienste an

WEB UND HTTP — WEB-DOKUMENTE

Webseiten bestehen aus Basis-HTML-Datei und anderen Objekten (.js, .png,...)

Jedes Objekt über URL (*uniform resource locator*) referenzierbar

HTTP — ÜBERBLICK

Protokoll der Anwendungsschicht (*hypertext transfer protocol*)

- einfaches, ASCII-basiertes Transferprotokoll

Basiert auf Client/Server-Modell

- *Client:* Browser, der Web-Objekte anfordert, empfängt und darstellt
- *Server:* sendet über HTTP angeforderte Objekte

zwei Nachrichten-Typen: *Request*, *Response*

Zustandslos:

- jeder Request wird individuell bearbeitet
- keine Zustandsinformation auf dem Server

nutzt TCP zur Kommunikation

1. Client initiiert Verbindungsaufbau
2. Server akzeptiert Verbindung
3. Austausch von HTTP-Nachrichten
4. Abbau der TCP-Verbindung

HTTP — METHODEN

HTTP-Anfragen können verschiedene Methoden nutzen

GET: Resource von Server zu Client übertragen (z.B. normale Webseite)

POST: Daten zu Ressource übertragen (z.B. Web-Formular)

Weitere Methoden:

- PUT — neue Ressource anlegen
- DELETE — Ressource löschen
- HEAD — wie GET, aber nur HTTP-Header übertragen
- ...

HTTP — STATUS-CODES

Verarbeitungsindikator (Erfolg/Fehlschlag + Gründe)

200: Erfolg; Antwort ist in dieser Nachricht

301: Angefragtes Objekt wurde verschoben (neue URL in Nachricht spezifiziert)

400: Server hat Anfrage nicht verstanden

404: Angefordertes Objekt existiert nicht

505: HTTP-Version nicht unterstützt

HTTP — VERBINDUNGEN

Non-persistent HTTP:

- höchstens ein Objekt wird über TCP-Verbindung gesendet, danach geschlossen
- ~> Herunterladen mehrerer Objekte erfordert mehrere TCP-Verbindungen

Persistent HTTP: mehrere Objekte über eine TCP-Verbindung

NON-PERSISTENT HTTP — ANTWORTZEIT

Round Trip Time (RTT): Zeit, die Paket von Sender zu Empfänger und zurück benötigt

HTTP-Antwortzeit:

- ein RTT für Verbindungsaufbau
- ein RTT für HTTP-Anfrage und erste Antwortbytes
- Zeit t_s für Senden der Datei
- ~> **Antwortzeit** $2 * RTT + t_s$

COOKIES

Speichert Nutzer-Server-Zustand

~> Server kann Inhalt abhängig von Nutzeridentifikation bereitstellen

Komponenten:

- Cookie-Information in HTTP-Response-Nachricht
- Cookie-Information wird in nachfolgenden HTTP-Requests genutzt
- Datei mit Cookies wird auf Nutzer-Endsystem vom Browser verwaltet
- Datenbank bei Webseite ~> Server muss Cookies richtig interpretieren können

COOKIES — PRIVATSPHÄRE

Webseiten unterscheiden Nutzer durch Cookies

~> Werbeanbieter können Nutzer über viele Webseiten tracken

Webseiten können durch Cookies sehr viel über Nutzer lernen

MAIL — KOMPONENTEN

User Agent (UA):

- lesen, senden, weiterleiten
- Beispiele: Outlook, Thunderbird

Mailserver:

- *mail transfer agent* (MTA)
- *mail delivery agent* (MDA)
- User-Mailboxen

simple mail transfer protocol (SMTP)

- Client/Server-Modell
- Transfer von Mails vom User Agent zum Mailserver

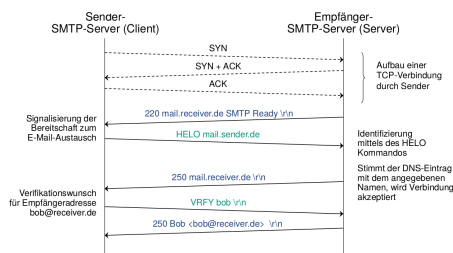
SMTP — AUFBAU

Drei Phasen:

1. Handshake
2. Nachrichtenübermittlung
3. Abschluss

Command/Response-Interaktionen

- ähnlich Request/Response bei HTTP
- Kommandos: ASCII-Text
- Antwort: Statuscode + Nachricht



MIME

Problem: SMTP kann nur ASCII-Texte versenden, keine Dateien

MIME: erweitert Kopfteil einer Nachricht um Formatinformation

- *Content-Type:* Definiert Typ des E-Mail-Inhalts

MAIL — POSTFACH-ABFRAGE

POP3 (post office protocol 3):

- Client holt von Mailserver empfangene/gespeicherte Nachrichten ab
- einfache Funktionalität
- verwaltet Nachrichten im UA, *keine* Synchronisation zwischen mehreren UAs

IMAP (*interactive mail access protocol*):

- Nachrichten werden zentral auf Mailserver verwaltet
- erweiterte Kommandos (Ordner, Filter)

XMPP

Echtzeit-XML-Streaming-Protokoll

Grundlage für Whatsapp usw.

Dezentral, ähnlich wie E-Mail

Clients: zu ihrem jeweiligen Server verbunden

Server: verbinden sich untereinander zur Nachrichtenübermittlung

Adressformat:

- *Nutzer:* Server + Username, z.B. alice@jabber.org
- *Clients:* pro Nutzer, z.B. alice@jabber.org/laptop

DNS — GRUNDLAGEN

Ziel: Verwendung von Namen statt IPs

Aufgabe: Zuordnung IP-Adresse ↔ Name

Funktionalitäten:

- *Registrierung* von Namen + IP-Adressen
- *Auflösung* von Namen in IP-Adressen

DNS — AUFBAU

Verteilte Datenbank von Name-Servern (DNS-Servern)

- Client-Server-Modell
 - Server kann Anfrage an weitere Server weiterleiten
- Protokoll der Anwendungsschicht
- Über Port 53 (UDP) realisiert

Basisdienst ~> keine Anwendung

- Komplexität am Rande des Netzes lokalisiert
- Internet-Design-Philosophie!

DNS — ANFRAGEN

Rekursiv: kennt angefragter Server Antwort nicht, fragt dieser dahinterliegende Server, bis er Antwort bekommt

Iterativ: kennt angefragter Server Antwort nicht, fragt *Client* andere Server

Üblich: Client fragt lokalen Name-Server rekursiv, dieser dann iterativ

DNS — RESOURCE RECORDS (RR)

DNS ordnet Domänen zu Einträgen zu

A / AAAA (Address): Abbildung Name auf IPv4/IPv6-Adresse

MX (Mail Exchange): Mailserver einer Domäne

NS (Name Server): Nameserver einer Domäne

CNAME (Canonical Name): Alias-Namen für Rechner/Domänen

PTR (Pointer): Abbildung IP-Adresse auf Name

Transportschicht

INTERNET-PROTOKOLLSTACK

Anwendungsschicht

Transportschicht

Vermittlungsschicht

Sicherungsschicht

Physikalische Schicht

TRANSPORTSCHICHT — ZIEL

Verbergen von Transportdetails vor höheren Schichten

- Fehlercharakteristika
- genutzte Technologien

~ ...

Bereitstellung von Transportdiensten

~> **Nutzer-zu-Nutzer-Kommunikation**

TRANSPORTPROTOKOLLE — PRINZIP

Transportprotokoll läuft auf Endsystemen

Sender:

- *Segmentieren* von Anwendungsnachrichten
- *Weiterleiten* an Vermittlungsschicht

Empfänger:

- *Reassemblieren* der Segmente in Nachrichten
- *Weiterleiten* an Anwendungsschicht

TRANSPORTSCHICHT — TRANSPORTDIENSTE

UDP (*user datagram protocol*):

bietet verbindungslosen, **unzuverlässigen** Transportdienst

TCP (*transmission control protocol*):

bietet verbindungsorientierten, **zuverlässigen** Transportdienst

TRANSPORTDIENST — UNZUVERLÄSSIG VS ZUVERLÄSSIG

Unzuverlässig:

- unklar, wieviel der gesendeten Daten heil ankommt
- keine Fehlermaßnahmen

Zuverlässig:

- *Korrektheit, Vollständigkeit, Reihenfolge* garantiert richtig
- keine Duplikate
- keine Phantom-Pakete
- Fehlermaßnahmen existieren

SCHICHT VS DIENST

Schicht: Abstraktion

Dienst: Bündelung zusammengehöriger Funktionen

- Höhere Schicht nutzt Dienst darunterliegender Schicht
- Dienste werden an Dienstzugangspunkt einer Schicht bereitgestellt

PORT

= Adressen der Transportschicht

Unstrukturierte Nummer (16 Bit), 0 bis 65535

Well known ports: viele Portnummern unter 1024 für häufig benutzte Anwendungen (Telnet, HTTP, ...) reserviert

IP-ADRESSEN

= Adressen der Vermittlungsschicht

IPv4: 32 Bit (z.B. 207.142.131.235)

IPv6: 128 Bit (z.B. 2001:0db8:85a3:08d3:1319:8a2e:0370:7344)

~~ Internetweite Adressierung eines Anwendungsprozesses: IP-Adresse + Port

UDP

RFC768 — *sehr einfaches Transportprotokoll mit sehr geringem Overhead*

Eigenschaften:

- (De-) Multiplexen von Segmenten für Prozesse
- Prüfsumme für Bitfehler
- verbindungslos
- *best effort*: keine Zusagen über Auslieferung bei Empfänger
- *Unreguliertes Senden*: kann Daten so schnell senden wie von Anwendung geliefert und von Netz abgenommen
- *keine Verbindungsaufbauphase*: sofortiges Senden ~> keine weitere Verzögerung
- *kein Verbindungszustand*: keine Verbindungsinformationen im Endsystem
- ~~ skaliert z.B. für Server besser

Verwendung:

- Multimedia
- DNS

| | | |
|------------|-----------|----|
| 0 | 16 | 32 |
| Quell-Port | Ziel-Port | |
| Länge | Prüfsumme | |
| Daten ... | | |

PROTOKOLLMECHANISMEN

Ziel: Datenaustausch zwischen Anwendungen/Geräten ermöglichen

~~ Festlegen von Formaten + Regeln für Datenaustausch nötig

Problem: Fehler bei Datenübertragung möglich

BITFEHLER

Verfälschung von Bits während dem Datentransport

Ursachen:

- Dämpfung Übertragungssignal
- Übersprechen
- Synchronisationsverlust
- ...

Einzelbitfehler: ein einzelnes Bit fehlerhaft

Bündelfehler: mehrere aufeinanderfolgende Bits fehlerhaft

PAKETFEHLER

Fehlerarten:

- Verlust
- Duplizierung
- Phantom-Paket
- Reihenfolgenvertauschung

Fehlerursachen:

- Zwischennetzüberlastung
- Unterschiedliche Wege durch Netz
- Verfrühte Datenwiederholung
- ...

FEHLER — HÄUFIGKEIT UND AUSWIRKUNGEN

Bitfehlerrate: Maß für Fehlerhäufigkeit

$$\text{Bitfehlerrate} = \frac{\text{Summe gestörter Bits}}{\text{Summe übertragener Bits}}$$

Fehlerauswirkungen: 20ms Störung in

- Telex (50bit/s ~> Bitdauer 20ms): 1 Bit fehlerhaft (Einzelbitfehler)
- Gigabit-Ethernet (1Gbit/s ~> Bitdauer 1ns): 20MBit fehlerhaft (Bündelfehler)

FEHLER — GEGENMASSNAHMEN

Fehlererkennung (*error detecting code, EDC*)

- Redundanz zu Daten hinzufügen
- Ausreichend stark unterschiedliche Codewörter verwenden

Fehlerkorrektur (*forward error correction, FEC*)

- Fehler mittels Redundanz korrigieren

Wiederholungsaufforderung (*automatic repeat request, ARQ*)

- Empfänger teilt Sender Ergebnis der Fehlerkorrektur mit

FEHLERKONTROLLE — BITFEHLER

Problem: Wie Bitfehler erkennen?

Ansatz: Hinzufügen von Redundanz

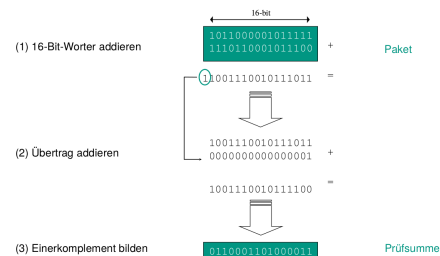
Paritätsprüfung: bekannt

BITFEHLER — INTERNET-PRÜFSUMME

Prinzip: Aufaddieren aller übertragenen Wörter (16 Bit, als Integer interpretiert)

~~ Prüfsumme

Nachteil: Falsche Reihenfolge kann nicht erkannt werden



BITFEHLER — UDP-PRÜFSUMME

Sender:

- UDP-Segment + -Kopf wird als Folge von 16 Bit-Wörtern aufgefasst
- Prüfsumme berechnen und in UDP-Kopf einfügen

Empfänger:

- Prüfsumme des UDP-Segments berechnen - Prüfsummen vergleichen

FEHLERKONTROLLE — PAKETFEHLER

Erkennung:

- Sequenznummern (*sequence number*)
- Zeitgeber (*timer*)

Behebung:

- Quittungen (*acknowledgements*)
- Sendewiederholungen (*retransmissions*)

PAKETFEHLER — SEQUENZNUMMERN

Problem: Empfänger weiß nicht, ob Pakete richtig (Reihenfolge, Duplikate, Vollständigkeit) angekommen sind

Prinzip: Pakete werden durchnummeriert

PAKETFEHLER — QUITTUNGEN

Problem: Sender erfährt nicht, ob Paket nicht (korrekt) angekommen ist

Prinzip:

- Empfänger informiert Sender über Erhalt
→ Acknowledgement (ACK)

Varianten:

- *positive Quittung:* Empfänger sagt Sender, dass er Daten erhalten hat (ACK)
- *negative Quittung:* Empfänger sagt Sender, dass er Daten *nicht* erhalten hat (NACK)
- *selektive Quittung:* Quittiert einzelnes Paket — z.B. bei Verlustvermutung (NACK)
- *kumulative Quittung:* Quittiert Paketmenge — z.B. alle Pakete bis bestimmte Sequenznummer sind ok

PAKETFEHLER — ZEITGEBER

Problem: Sender merkt nicht, wenn Paket nicht angekommen ist

Prinzip: Durch zeitliche Obergrenze wird *vermutet*, dass Paket nicht angekommen ist
→ Sendewiederholung

SENDEWIEDERHOLUNG — ARQ

= automatic repeat request

Grundlegende Sendewiederholungsvariante

Varianten:

- Wann werden Quittungen versendet?
- Wann wird eine Sendung wiederholt?

SENDEWIEDERHOLUNG — STOP-AND-WAIT

= einfaches ARQ-Verfahren

Prinzip:

- Sender wartet auf Quittung für gesendetes Paket
- Erst *nach* Quittungserhalt wird nächstes Paket gesendet
- keine Quittung → Sendewiederholung
- Wartezeit durch Zeitgeber geregelt

STOP-AND-WAIT — SEQUENZNUMMERN

Problem: Empfänger kann Paket doppelt erhalten (nicht erkennbar)

Prinzip: Pakete werden mit Sequenznummern versehen (für Stop-and-Wait reicht 1 Bit)

Auslastung: $U = \frac{1}{1+2a}$ (mit $a = \frac{t_a}{t_s}$)

BANDBREITENVERZÖGERUNGSPRODUKT

=: $\frac{m}{v} r$ (Länge m , Ausbreitungsgeschwindigkeit v , Datenrate r)

= **Speicherkapazität des Mediums**

SENDEWIEDERHOLUNG — GO-BACK-N ARQ

Zeit: Leistungsfähigkeit von Stop-and-Wait erhöhen

Prinzip:

- Sender sendet mehrere Pakete bis Quittungspflicht
- begrenzte Anzahl an nicht quitierten Paketen (durch **Fenster** (*window*) begrenzt)
- Quittierung durch kumulative Quittungen

Fehlerfall:

1. Empfänger empfängt fehlerhaftes Paket
2. Empfänger verwirft alle nachfolgenden Pakete
3. Sender wartet auf Ablauf des Zeitgebers
4. Sender wiederholt alle nicht quitierten Pakete

Fragen:

- Wo müssen Pakete gepuffert werden?
- Wieviele Pakete müssen gepuffert werden?

Auslastung: $U = \begin{cases} 1 & W \geq 1 + 2a \\ \frac{W}{1+2a} & \text{sonst} \end{cases}$

SENDEWIEDERHOLUNG — SELECTIVE REPEAT ARQ

Ziel:

- Auslastung von Stop-and-Wait erhöhen
- Datenaufkommen von Go-Back-N reduzieren

Prinzip: Wie Go-Back-N, Empfänger quitiert selektiv

Fehlerfall:

1. Empfänger bestätigt nachfolgende, korrekt empfangene Pakete
2. Sender wiederholt nur nicht korrekt empfangene Pakete

Fragen:

- Wo müssen Pakete gepuffert werden?
- Wieviele Pakete müssen gepuffert werden?
- Vor-/Nachteile von Go-Back-N und Selective Repeat

SENDEWIEDERHOLUNG — SELECTIVE REPEAT VS. SELECTIVE REJECT

Selective Repeat:

- Fehlerhaftes Paket wird nicht bestätigt
- Sender wartet auf Timeout

Selective Reject:

- Empfänger versendet für fehlerhaftes Paket negative Quittung
- Sender wiederholt sofort und wartet nicht auf Timeout

PAKETFEHLER — VORWÄRTSFEHLERKORREKTUR

Ziel: Empfänger muss nur drei von vier Paketen korrekt empfangen um fehlendes Paket rekonstruieren zu können

Prinzip: XOR-Verknüpfung der drei Pakete → fehlendes Paket

FLUSSKONTROLLE

Problem:

- Überlastung von Empfänger durch Sender → Datenverlust - Sendet muss Empfangspuffergröße berücksichtigen

Anforderungen:

- einfach
- möglichst wenig Netzressourcen nutzen
- fair
- stabil

FLUSSKONTROLLE — HALT-UND-WEITER

Prinzip:

- Empfänger kommt nicht mehr mit → **halt**-Signal
- Empfänger wieder verfügbar → **weiter**-Signal

Bewertung:

- nur auf Vollduplex verwendbar
- nicht effektiv bei hohen Verzögerungen
- Probleme bei Verlust der **halt**-Meldung

Beispiele:

- Fast- + Gigabit-Ethernet

FLUSSKONTROLLE — STOP-AND-WAIT

Prinzip:

- Empfänger sendet Quittung erst, wenn er wieder kann
- Sender wird durch Zurückhalten gebremst

Problem:

- Sender kann nicht zwischen Datenverlust und Überlastung unterscheiden

FLUSSKONTROLLE — KREDITBASIERT

Prinzip:

- Sender kann höchstens n Pakete unquitiert senden
- n = Pufferkapazität des Senders ⇒ **Sendekredit**
- Alternativbezeichnung: Fenster (*sliding window*)
- Fenster wird durch korrekte positive Quittung weitergeschaltet
- Empfänger kann Kredit bestimmen (z.B. in TCP)

TCP — PRINZIP

Erhält Bytestrom von Anwendung, übergibt TCP-Segmente an IP

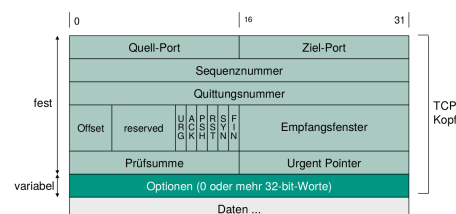
Problem: Wie Bytestrom in TCP-Segmente schnippeln?

Implementierung:

- MSS (*maximum segment size*): Anwendungsdatenlänge (z.B. 1460 Byte)
- Push (**PSH** in TCP-Segmentkopf): Sender verlangt sofortiges Versenden der Daten
- Zeitgeber: nach inaktivem Zeitintervall werden vorhandene Daten gesendet

Fehlerkontrolle: Sequenznr., Prüfsumme, Quittierungen, Sendewiederholungen

Sequenznummern: pro Byte, nicht pro Segment (erstes Byte in Segment, initiale Sequenznummer von Endsystem zufällig gewählt)



TCP — FELDER

Quell-/Ziel-Port: Identifizieren Verbindungsendpunkte

Sequenznummer: gemessen in Byte, nicht pro Segment

Quittung: nächste von Empfänger erwartete Sequenznummer

Offset: Anzahl 32 Bit-Wörter in TCP-Kopf

URG: 1, falls *urgent pointer* verwendet wird (idR leer)

SYN: Wird bei Verbindungsaufbau verwendet, um *connection request* oder *connection confirmation* anzuzeigen

ACK: unterscheidet bei gesetztem SYN-Bit zwischen Request und Confirmation; signalisiert Gültigkeit von Quittungsfeld

FIN: gibt an, dass Sender nichts mehr senden möchte

RST: Verbindung zurücksetzen

PSH: übergebene Daten sollen sofort weitergeleitet werden (idR leer)

Empfangsfenster: für Flusskontrolle

Prüfsumme: Prüfsumme über TCP-Kopf, Daten und Pseudoheader

Urgent-Zeiger: relativer Zeiger auf wichtige Daten

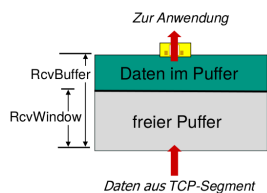
Optionen-Feld: kann Optionen variabler Länge aufnehmen

TCP — FLUSSKONTROLLE

Ziel: Empfängerüberlastung vermeide

Prinzip:

- Empfänger: reserviert Pufferplatz pro Verbindung (explizite Kreditvergabe)
- **RcvBuffer:** gesamter Pufferplatz (default 4096 Byte)
- **RcvWindow:** freier Pufferplatz (Empfangsfenster)
- Sender sendet nicht mehr unbestätigt als in **RcvWindow** passt



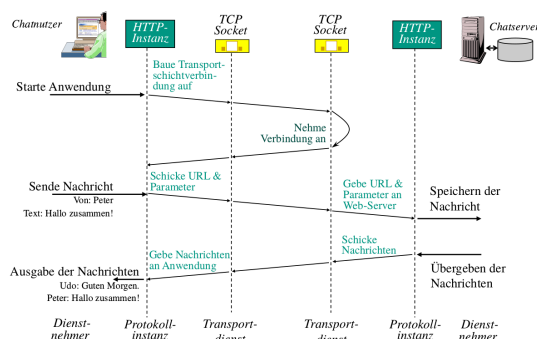
VERBUNDINGSVERWALTUNG — VERBUNDINGSLOS VS. VERBUNDINGSORIENTIERT

Verbindungslos:

- Daten werden ohne vorherigen Handshake gesendet
- **Vorteil:** schnelle Datenversendung möglich
- **Nachteil:** kein Feedback, keine Bestätigung

Verbindungsorientiert:

- Verbindungsaufbau vor Datenaustausch, Verbindungsabbau danach
- **Vorteil:** Kommunikationsparameter können ausgehandelt werden
- **Nachteile:** Verzögerter Datenaustausch, Overhead ggf größer als Daten



TCP — ZUSAMMENSPIEL MIT HTTP

STAUKONTROLLE

Ziel: Netzüberlastungssituationen vermeide

Prinzip:

- Staukontrollfenster (*congestion window*, **CWnd**) beim Sender beeinflusst maximal zu sendende Datenmenge:
- $\text{LastByteSent} - \text{LastByteAcked} \leq \min\{\text{CWnd}, \text{RcvWindow}\}$
- Schwellenwert (**SSTresh**)

Stauerkennung:

- Nutzung von Zeitgebern
- Vermutung einer Stausituation bei ausbleibender Quittung

Staubehhebung:

- Reduktion von **CWnd**
- Langsames Erhöhen von **CWnd** \leadsto herantasten an Netzkapazität

STAUKONTROLLE — TCP

Start: **CWnd** = 1 MSS (*maximum segment size*)

Slow-Start, falls **CWnd** \leq **SSTresh** & Quittungen rechtzeitig da

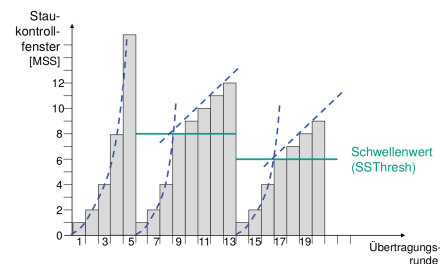
- Exponentielles Erhöhen **CWnd** (**CWnd** += 1 bei jeder empfangenen Quittung)

Congestion Avoidance, falls **CWnd** > **SSTresh** & Quittungen rechtzeitig da

- Lineares Erhöhen **CWnd** (**CWnd** += $\frac{1}{\text{CWnd}}$ bei jeder empfangenen Quittung)

Congestion, falls Quittung nicht rechtzeitig da

- Stau vermutet
- **SSTresh** = $\max(\frac{\text{FlightSize}}{2}, 2 * \text{MSS})$ (FlightSize = unquitierte, gesendete Daten)
- **CWnd** zurücksetzen (neue Slow-Start-Phase): **CWnd** = 1 MSS



Vermittlung

VERMITTLUNGSTECHNIKEN — LEITUNGSVERMITTLUNG

Prinzip: Verbindung erhält *durchgehenden Kanal* mit konstanter Bandbreite für exklusive Nutzung

Multiplexing: *starr*es Multiplexing möglich

- **Frequenzmultiplex:** Feste Zuweisung von Übertragungskanal + Frequenzabschnitt
- **Zeitmultiplex:** Feste Zuweisung von Übertragungskanal + Zeitschlitz (*time slot*)

Eigenschaften:

- Aufbau eines durchgehenden Übertragungskanals zwischen Endsystemen
- keine Adressinformation nötig, dafür Zustandshaltung
- zugesicherte, feste Datenrate \leadsto ungenutzte Ressourcen bei Nichtverwendung
- Übertragungsverzögerungen nur physikalisch bedingt
- \leadsto keine Schwankungen durch Puffer
- Reihenfolgentreue Bitfolgenübertragung

Einsatzgebiete: Telefonnetze, GSM

VERMITTLUNGSTECHNIKEN — PAKETVERMITTLUNG

Prinzip: Weiterleitung anhand von Kontrollinformation in Paket

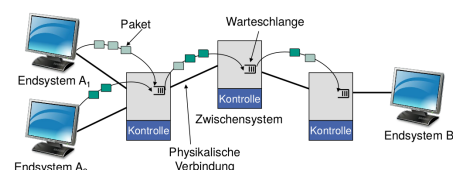
- Zieladresse in Datagrammen
- lokale Kennung bei virtuellen Verbindungen
- Zwischensysteme speichern Pakete in *Warteschlangen* \leadsto Paketverlust möglich

Eigenschaften:

- Wechselnde Paketwege möglich \leadsto Reihenfolgevertauschung möglich
- Üblicherweise Zeitmultiplex

Varianten:

- *verbindungslos:* Datagramme
- *verbindungsorientiert:* virtuelle Verbindungen



PAKETVERMITTLUNG — DATAGRAMME

Pakete (= Datagramme) werden als isolierte Einheiten betrachtet

Ziel: In jedem Datagramm enthalten → keine Verbindungsverwaltung nötig

Routing: können Netz unterschiedlich durchlaufen
→ Datagramme können bei Empfänger unsortiert ankommen

PAKETVERMITTLUNG — VIRTUELLE VERBINDUNGEN

fester Übertragungsweg zwischen zwei Endsystemen

Reihenfolgetreue: Alle Pakete gehen selben Weg

Kennungen: *virtual circuit identifier* (VCI) kennzeichnen Pakete

Ziel: Zieladresse nur bei Verbindungsaufbau nötig

Prozess: Verbindungsaufbau → Datenübertragung → Verbindungsabbau

VERMITTLUNGSTECHNIKEN — NACHRICHTENVERMITTLUNG

Vermittlung von Anwendungsnachrichten

Vermittlung üblicherweise mittels mehrerer Pakete

→ Segmentierung und Reassemblierung

Zwischensysteme: Reassemblierung nötig, da alle Teile in selbe Richtung weitergeleitet werden müssen

→ Ende-zu-Ende-Verzögerung wesentlich höher als bei Paketvermittlung

VERMITTLUNGSSCHICHT — ÜBERBLICK

Ende-zu-Ende: transportiert Segmente zwischen Endsystemen

Sender: kapselt Segmente in Datagramme

Empfänger: Segmente werden an Transportschicht ausgeliefert

Protokolle: in *allen* Endsystemen und Routern

Router: werten Felder im Kopf aller Datagramme aus, die sie passieren

VERMITTLUNGSSCHICHT — AUFGABEN

Weiterleitung (*forwarding*): Datenebene

- Pakete werden von Routerzugang an geeigneten Ausgabe weitergeleitet

Wegwahl (*routing*): Kontrollebene

- ermittelt Weg, den Pakete zurücklegen (sollen)
- erfordert Routingalgorithmus + -protokoll

VERMITTLUNGSSCHICHT — KONTROLL- UND DATENEBENE

Kontrollebene:

- betrachtet gesamtes Netz
- bestimmt wie Datagramm über Router von Quelle zu Ziel geroutet wird
- *Konzepte:*
 - traditionelle Routingalgorithmen: in jedem Router implementiert
 - *software defined networking* (SDN): in logisch zentralen Servern implementiert

Datenebene:

- Funktionen lokal in Router
- Bestimmt wie Datagramm von Eingangs- zu Ausgangsport geleitet wird

VERMITTLUNGSSCHICHT — BEGRIFFE

Router:

- auf Vermittlungsschicht operierendes Zwischensystem
- leitet Datagramme mithilfe von Weiterleitungstabelle weiter
- tauscht über Routingprotokolle Informationen mit anderen Routern aus

Route: Weg eines Datagramms von Start zu Ziel

Link: Übertragungsabschnitt zwischen 2 Routern (kann z.B. auch Brücken enthalten)

Port: Eingangs-/Ausgangs-Netzwerkschnittstelle

VERMITTLUNGSSCHICHT — PROTOKOLLE

IP (*internet protocol*):

unzuverlässige Datagrammübertragung

ICMP (*internet control message protocol*):

Kontrollinformationsaustausch innerhalb der Vermittlungsschicht

ARP (*address resolution protocol*):

Zuordnung von IP-Adressen zu Adressen der Sicherungsschicht

RARP *reverse ARP*:

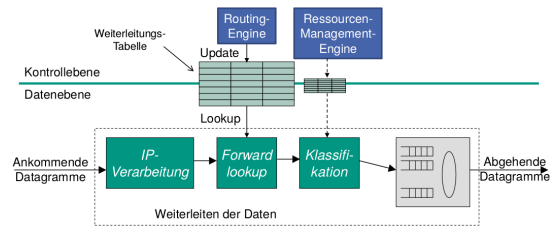
Umkehrfunktionen von ARP

BGP (*border gateway protocol*), **RIP** (*routing information protocol*), **OSPF** (*open shortest path first*): Routingprotokolle

VERMITTLUNG — IP

IP macht die ganze Vermittlung → nur ein großes Vermittlungsprotokoll

- Interoperabilität erhöhen
- Anzahl unterschiedlicher Interfaces erniedrigen
- kleinster gemeinsamer Nenner
- Anzahl nutzbarer Netze maximieren



IP — FRAGMENTIEREN + REASSEMBLIEREN

Anpassung an Maximallänge unterliegender Netze (**MTU**: *maximum transfer unit*)

Flag-Felder IP-Kopf:

- *Bit 0*: reserviert, muss 0 sein
- *Bit 1*: 0 = darf fragmentiert werden, 1 = nicht
- *Bit 2*: 0 = letztes Fragment, 1 = nicht

IP — WEITERLEITUNG

Endsystem:

- Rechner mit Zieladresse direkt verbunden → Datagramm direkt zustellen
- Sonst: Datagramm-Übergabe an Standardrouter

Router: Verwendung *Weiterleitungstabelle*

- Zieladresse
- *Next-Hop-Router*
- *Flags*, die Start und Ziel genauer klassifizieren
- *Netzschnittstelle*, auf die bei Endsystem das Datagramm ausgegeben werden soll

Weiterleitungstabelle: Identifikation des nächsten Systems auf Weg zum Ziel

IP — EMPFANGSPROZESS

Überprüfen:

- *Kopflänge*
- *Datagrammlänge*
- *Versionsnummer* IP
- *Prüfsumme*
- *Lebenszeit*
- *Protokoll-Identifikation*
- *Adressklassen*

Fehlerfall: Benachrichtigung ICMP (*internet control message protocol*) — möglicherweise wird ICMP-Paket ausgesendet

IP — ADRESSIERUNG

Ziel: Eindeutige Identifizierung aller angeschlossenen Systemschnittstellen

IP-Adressen: Kennungen für Interfaces von Routern/Endsystemen

- IPv4: 32 Bit-Adressen
- IPv6: 128 Bit-Adressen

IP — SUBNETZE

Gliederung: IP-Adresse unterteilt in

- *Subnetz-Teil*: high order bits
- *Endsystem-Teil*: low order bits

Subnetz: Interfaces mit selbem Subnetz-Teil, können ohne Router kommunizieren

ADRESSZUTEILUNG

Manuell: Durch Systemadministrator

Dynamisch: *dynamic host configuration protocol* (DHCP)

- DHCP-Server liefert IP auf Anfrage

ADRESSBLOCKZUTEILUNG

Provider: Erhält Block von ICANN (*internet corporation for assigned names/numbers*)

- ICANN allokiert Adressen
- ICANN verwaltet DNS
- ICANN weist Domainnamen zu

INTERNET CONTROL MESSAGE PROTOCOL (ICMP)

Einzelne Datagrammverluste: meldet IP nicht (unzuverlässiger Dienst)

Schwerwiegende Probleme (z.B. Verbindungsabbruch): Mitteilung an Kommunikationspartner via ICMP

⇒ ICMP tauscht Fehlernachrichten, Statusanfragen und Zustandsinformationen aus

ICMP — STATUSANFRAGEN

Echo + Antwort (*echo and echo reply*):

- Aktivitätsüberprüfung von Kommunikationssystemen
- Empfänger von Echo-Anfrage sendet erhaltene Daten in Echo-Antwort zurück

Zeitstempel + Antwort (*timestamp and timestamp reply*):

- Bestimmung von Umlaufzeiten (*round trip time*, RTT)

IPv6

Problem:

- Adressraum von IPv4 geht aus
- Kopfformat IPv4 nicht optimal

Lösung:

- Erhöhung Adresslänge von 32 auf 128 Bit
- feste Kopflänge (40 Byte)
- keine Unterstützung von Fragmentierung
- keine Prüfsumme
- Optionen: als Erweiterungsköpfe (*next header*)
- ICMPv6: neue Version von ICMP

ROUTING — PRINZIPIEN

Ziel: guten Weg durch Netz finden

Weg: Sequenz von Routern von Start- zu Ziel-Endsystem

Netzgraph: Netz wird als Graph verstanden

- *Knoten:* Router
- *Kanten:* Übertragungsabschnitte (Kantenkosten z.B. Verzögerung, Preis,...)

Pfad: Knotenfolge (meist Pfad mit kürzester Länge gesucht)

ROUTING-VERFAHREN — DYNAMIK

Frage: Wie dynamisch ist Routing-Verfahren?

Nicht adaptiv: Routen ändern sich sehr selten, wesentlich seltener als Verkehrsänderungen

Adaptiv:

- Routen ändern sich abhängig von Verkehr und Topologie
- Routenänderungen periodisch oder reaktiv
- *Zielkonflikt:* Systeme haben ggf kein Live-Abbild des Netzes
ggf hohe Netzbelastung durch Routing-Informationsaustausch

ROUTING-VERFAHREN — STATISCHES ROUTING

Beispiel:

- Zufallszahl $1 < x \leq 0$
- Falls $x < 0.6$ weiterleiten nach B
- Falls $0.9 \geq x \geq 0.6$ weiterleiten nach C
- Sonst D

ROUTING-VERFAHREN — ZENTRALISIERT

Adaptives Verfahren

Routing Control Center: Für Berechnung/Verteilung der optimalen Pfade

- Systeme senden periodisch Zustand an RCC
(aktive Nachbarn, Warteschlangenlängen,...)

Vorteile:

- RCC hat alle Informationen ⇒ kann perfekte Entscheidungen treffen
- Systeme müssen kein Routing betreiben

Nachteile:

- Berechnungsdauer für große Netze ggf sehr lang
- Ausfall RCC lähmt ganzes Netz
- Inkonsistenzen möglich, da Systeme nah an RCC Tabellen schneller erhalten
- starke Belastung des RCC

ROUTING-VERFAHREN — ISOLIERT

Prinzip: Jedes System entscheidet anhand selbstgesamelter Information

- kein Routinginformationsaustausch zwischen Systemen

ISOLIERTES ROUTING — FLUTEN

Prinzip: Jedes eingehende Datagramm wird auf jeder Übertragungsleitung weitergeleitet

Fluteindämmung:

- *Sequenznummern* für Duplikaterkennung
- *Lebensdauerkontrolle* durch Zählen der Übertragungsabschnitte (*hops*)

Varianten:

- *selektives Fluten:* Weiterleitung nicht auf allen Übertragungsabschnitten
- *random walk:* Zufällige Auswahl eines Übertragungsabschnittes

ISOLIERTES ROUTING — HOT POTATO

Prinzip: Jeder versucht Datagramme so schnell wie möglich weiterzuleiten

⇒ Übertragungsabschnitt mit kürzester Warteschlange wählen

Varianten:

- nie an Herkunftsleitung weiterleiten
- Kombination mit statischem Routing: statisches Verfahren zur Auswahl von Leitung mit Warteschlangenlänge unter Schwellwert

ROUTING-VERFAHREN — VERTEILTES ADAPTIVES ROUTING

Prinzip: Systeme tauschen Routing-Informationen mit Nachbarn aus

- jedes System unterhält Routing-Tabelle

Varianten:

- periodischer Informationsaustausch
- Austausch nur bei größeren Änderungen

ROUTING-ALGORITHMEN — ÜBERSICHT

Distanz-Vektor-Algorithmen: Distanz als Routing-Metrik

- jeder Router kennt Distanz zu allen anderen Systemen im Netz
- *Problem:* kürzerer langsamer Weg wird längerem schnelleren vorgezogen

Link-State-Algorithmen: Unterschiedliche Routing-Metriken

- berücksichtigt aktuelle Zustände der Netzanschlüsse
- jeder Router kennt ganze Netztopologie
- Link-State- konvergieren schneller als Distanz-Vektor-Algorithmen

ROUTING-ALGORITHMEN — DISTANZ-VEKTOR

Eigenschaften:

- *verteilt:* jeder Router erhält Infos von direkten Nachbarn, führt Berechnung durch und verteilt dann neue Informationen an direkte Nachbarn
- *iterativ:* Verteilen + Berechnen von Informationen so lange, bis keine Information mehr ausgetauscht wird

Distanz-Vektor-Tabelle:

- Grundlegende, in jedem System vorhandene Datenstruktur
- Zeile für jedes mögliche Ziel
- Spalte für direkte Nachbarn

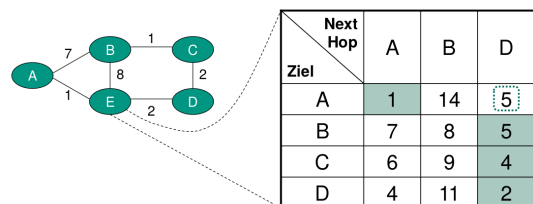
DISTANZ-VEKTOR-ROUTING — DISTANZ-VEKTOR-TABELLE

Beispiel: X will Daten über direkten Nachbar Z an Y weiterleiten

$$D^X(Y, Z) = c(X, Z) + \min_w \{D^Z(Y, w)\}$$

Beispiel: $D^E(A, D)$

- erster Übertragungsabschnitt: $E \rightarrow D$
- Tabelleneintrag: Kosten $E \rightarrow D (= 2)$ + minimale Kosten $D \rightarrow A (= 3)$
⇒ **minimale Kosten von D nach A über Nachbar von D**



DISTANZ-VEKTOR — DISTANZ-VEKTOR-ALGORITHMUS (BELLMAN-FORD)

Initialisierung:

- für alle Nachbarn v : $D^X(*, v) = \infty$, $D^X(v, v) = c(X, v)$
- für alle Ziele y : sende $\min_w D^w(y, w)$ allen Nachbarn (w enthält alle Nachbarn)

Schleife:

- geänderte Abschnittskosten: für alle Ziele y : $D^X(y, V) := D^X(y, V) + d$
- Update von Nachbarn: kürzester Pfad von V zu Ziel Y hat sich um α geändert
 $\rightsquigarrow D^X(Y, V) := c(X, V) + \alpha$
- \rightsquigarrow Falls neuer $\min_w D^w(Y, w)$ für Ziel Y existiert, dann sende allen direkten Nachbarn diesen Wert

Komplexität: $O(n^2 k)$ für n Knoten und k Kanten

DISTANZ-VEKTOR-ALGORITHMUS — UPDATEAUSBREITUNG

Ausbreitung good news: schnelle Ausbreitung

Ausbreitung bad news: langsame Ausbreitung, ggf Routing-Schleifen

\rightsquigarrow **Count to Infinity-Problem**

DISTANZ-VEKTOR-ALGORITHMUS — POISONED REVERSE

Ziel: Vermeidung von Routing-Schleifen

Prinzip: Routing-Information wird Y vorenthalten, wenn Weg über Y kürzer

ROUTING-ALGORITHMEN — LINK-STATE-ROUTING**Prinzip:**

- Systeme müssen zu Beginn nur direkte Nachbarn kennen
- Entdecken von neuen Nachbarn zB mittels HELLO-Pakete
- *link state broadcast*: Identität + Kosten zu Nachbarn werden allen Routern im Netz weitergeleitet (Fluten)
- Systeme lernen Topologie durch LSBs der anderen Systeme
- *Ergebnis*: Alle Systeme haben *identisches* Wissen über Netz

Implementierung: Dijkstra-Algorithmus

LINK-STATE VS. DISTANZ-VEKTOR**Komplexität Kontroll-Pakete:**

- *link-state*: jedes System muss Kosten aller Links kennen, Änderungen müssen an alle Systeme geschickt werden
- *Distanz-Vektor*: Änderungen nur an benachbarte Systeme weitergegeben

Konvergenzgeschwindigkeit:

- *link-state*: $O(n^2)$, $O(nE)$ Pakete \rightsquigarrow schnelle Konvergenz, danach schleifenfrei, aber Oszillationen möglich
- *Distanz-Vektor*: langsame Konvergenz, Schleifen möglich, Count-to-Infinity kann auftreten

Robustheit:

- *link-state*: Routenberechnungen separiert \rightsquigarrow Robustheit
- *Distanz-Vektor*: ein System kann inkorrekte Pfade zu allen Zielen verbreiten

Gewinner?

- *link-state*: Konvergiert schneller, ist robuster
- *Distanz-Vektor*: einfacher zu implementieren

SOFTWARE DEFINED NETWORKING**Eigenschaften:**

- **E1**: Separierung von Kontroll- und Datenebene
- **E2**: Flow-basierte Paketweiterleitung
- **E3**: Log an externen Controller ausgelagert
- **E3**: Netzwerk programmierbar

Umsetzung: open flow-Protokoll

- regelt Kommunikation zwischen Controller und Switch
- OpenFlow quasi-Standard, Alternativen existieren

TRADITIONELLES IP-ROUTING

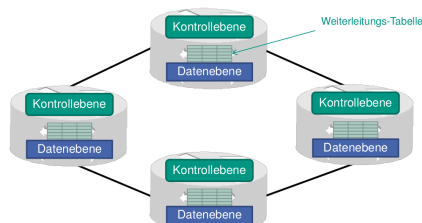
Kontroll- und Datenebene in jedem Router

Vorteile:

- Ausfallsicherheit (selbstorganisiert, verteilte Kontrolle, hohe Redundanz)
- Schnelle Reaktion (optimierte Routing-Hardware, lokale Routingentscheidung)
- Bewährtes Konzept

Nachteile:

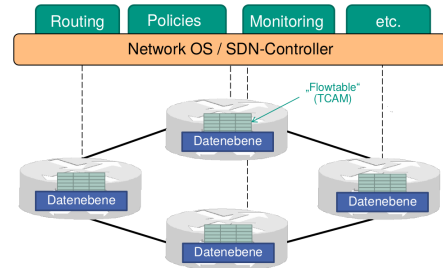
- proprietäre Management-Schnittstellen (Mischbetrieb schwierig)
- unflexibel (neue Funktionen hinzufügen schwierig, aufwändige Standardisierung)
- teuer (hochqualifiziertes Personal + Overprovisioning benötigt)

**SDN-ROUTING****Vorteile:**

- logisch zentralisierte Sicht (Controller hat Netzüberblick, einfacher Einsatz von Graphalgorithmen)
- neue Funktionalität in Software (als App im Controller)
 \rightsquigarrow kürzere Entwicklungszyklen
- Trennung von Kontroll- und Datenebene (Innovationen unabhängig möglich, herstellerunabhängig durch offene Schnittstellen)

Nachteile:

- Skalierbarkeit
- *single point of failure*
- Kommunikation mit Controller langsam

**SDN — TRADITIONELLER ROUTER VS. SDN-SWITCH****Traditioneller IP-Router:**

- kennt keine Flows
- Weiterleitung über Ziel-IP-Adresse

SDN-Switch:

- Weiterleitung über Flowtable
- nutzt verschiedene IP-Kopf-Felder
- speichert Zustand pro Flow

