Die LLC-Ebene

- Bei LANs unterteilt man i.a. die Schicht 2 in “Sublayers”:

3 Network Layer	
2b LLC	Logical Link Control
2a MAC	Medium Access Control (bei Ethernet: CSMA/CD)
1 Physical Layer	

- LLC bildet zur Schicht 3 die gemeinsame Schnittstelle für verschiedene LAN-Technologien

- LLC enthält klassische Funktionalität der ISO-OSI-Sicherungsschicht (framing, Fehlerbehandlung etc.)

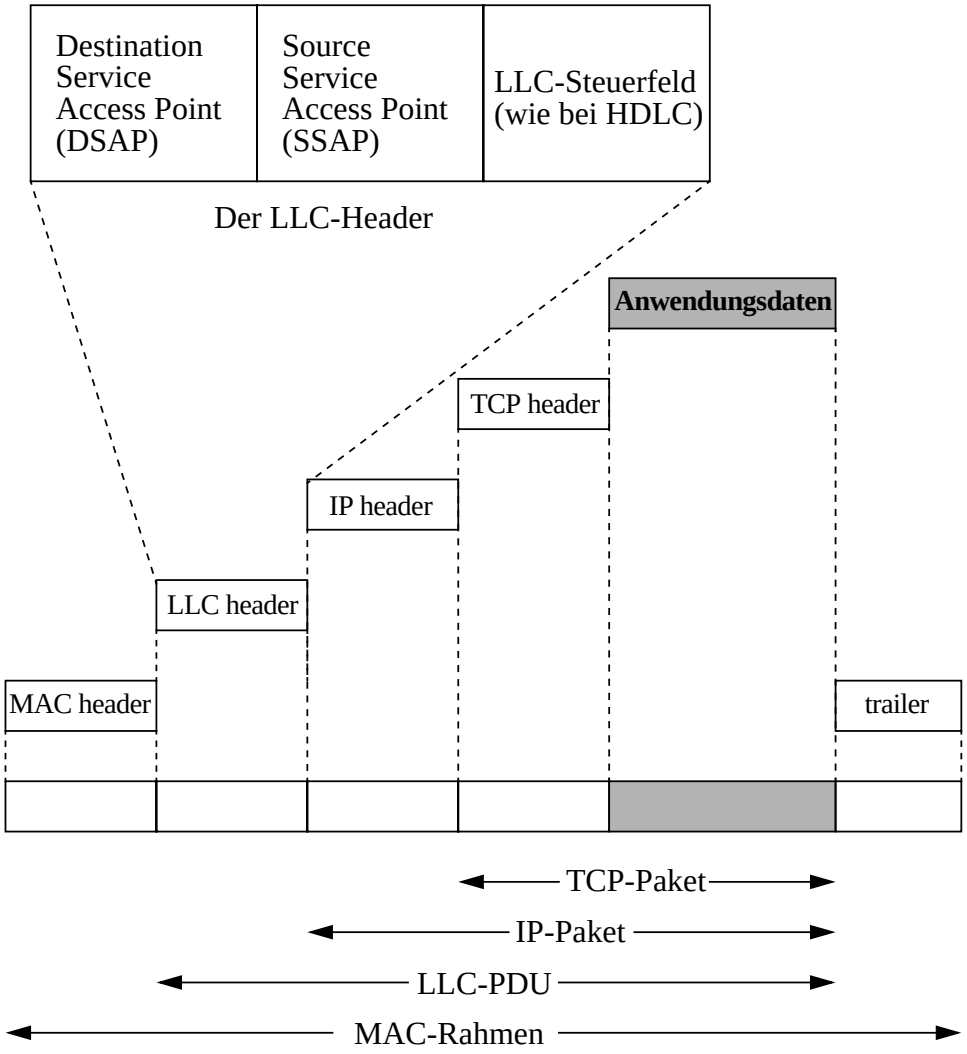
- LLC bietet prinzipiell drei verschiedene Dienste an:
  - connectionless unreliable datagram
  - connectionless acknowledged datagram
  - reliable connection (Auf- und Abbau von Verbindungen, Reihenfolgesicherung, Flusssteuerung, Empfangsbestätigung etc. analog zu HDLC)

  nicht immer sind alle implementiert!

- LLC fügt (je nach Dienst) zu einem Datenpaket der Schicht 3 Sequenznummern hinzu und verwendet ACKs

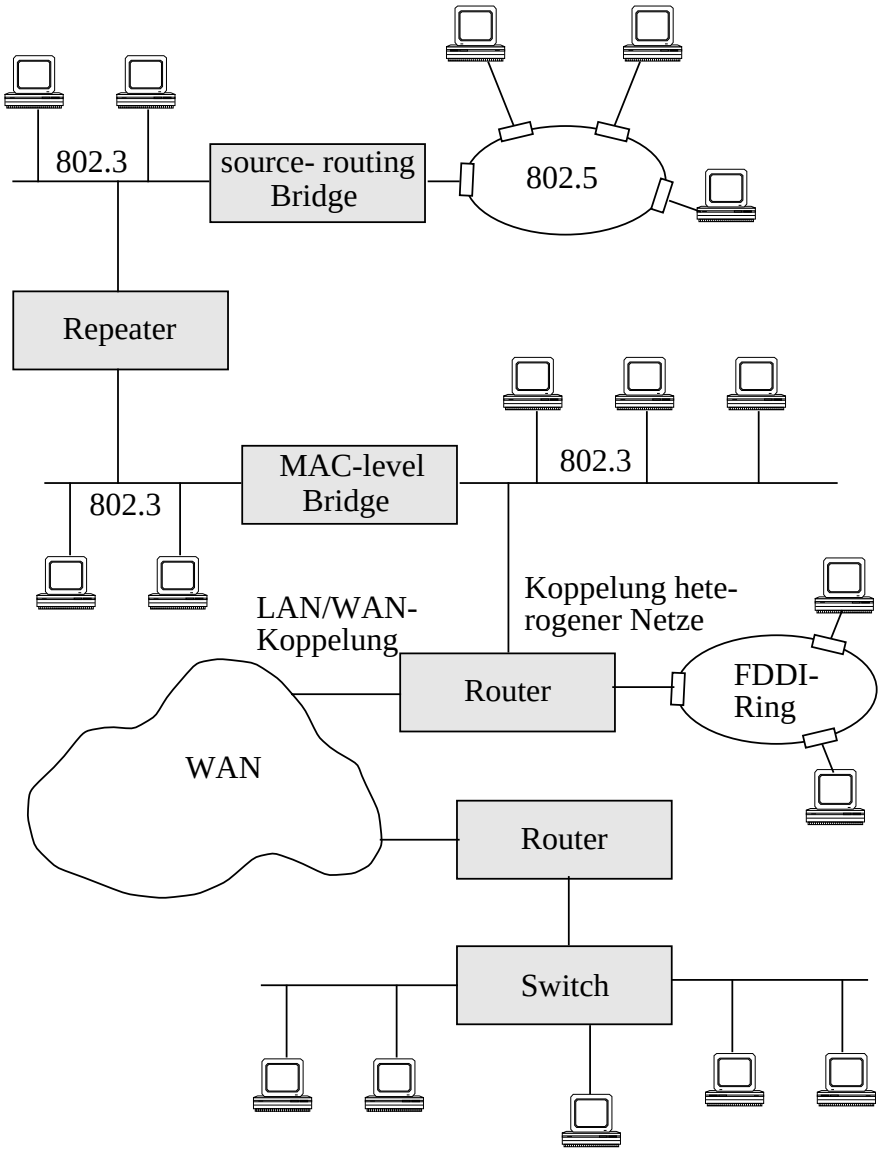
- Oft genügt “unreliable datagram”: Da bei IP der Verlust von Paketen sowieso “möglich” ist, werden z.B. IP-Pakete einfach in das Datenfeld von Ethernet-Pakten eingefügt und als “unreliable datagram” versendet
  - ist der effizienteste Dienst; Fehlerbehandlung und Flusssteuerung muss dann allerdings auf höherer Ebene gemacht werden

Paketverschachtelungen

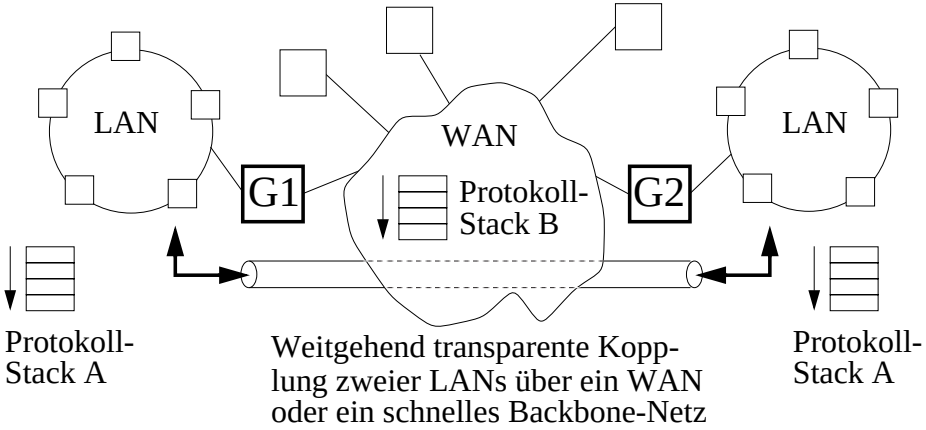


- Beachte: ggf. muss ein Paket der Ebene i in mehrere Pakete der Ebene i-1 aufgebrochen werden

Netzkoppelungen



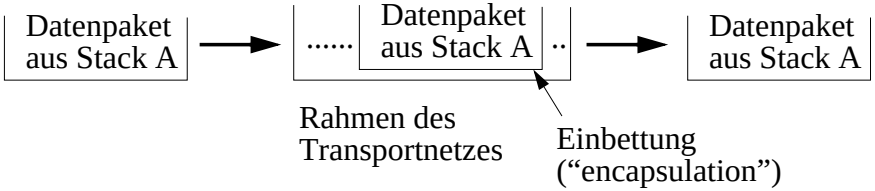
Tunneln



- *Idee:* Benutze dazwischengeschaltetes Netz nur als Transportdienst für Datenpakete von Stack A

- Gateways G1 und G2 sind über (z.B. fest eingerichtete) *virtual circuits* miteinander verbunden

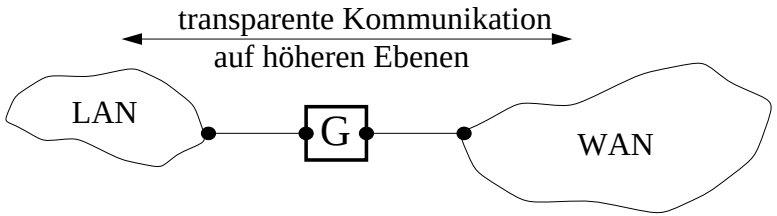
- Transportnetz ist *transparent*; es findet keine eigentliche Protokollumsetzung statt



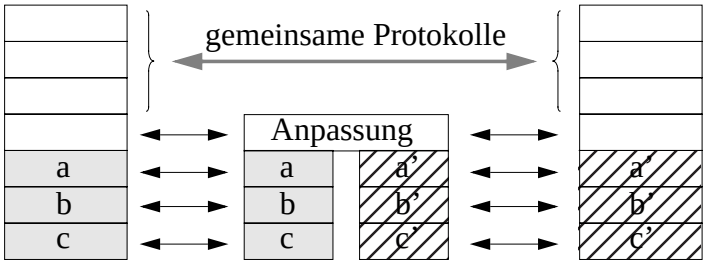
Netzverbund und Gateways

- Verbund von Netzen (ggf. mit untersch. Protokollen)
- *Gateway:*

  Vermittlungsinstanz (aus Hard- und Software), die zwei Netze verbindet und eine ggf. notwendige Protokollumsetzung vornimmt



- Gateway ist charakterisiert durch die Ebene, in der die Protokollanpassung vorgenommen wird
  - darüber: gleiche Protokolle
  - darunter: verschiedene Protokolle



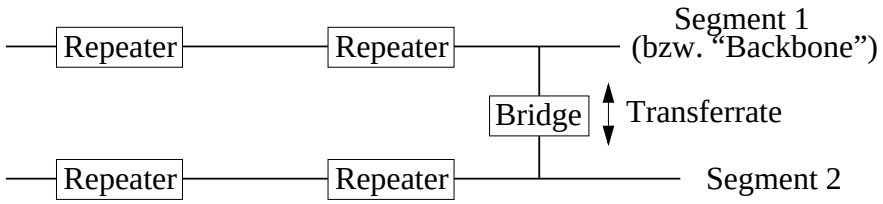
- *Anpassungsschicht* stellt sich in beiden Teilnetzen so dar, als wäre sie eine Instanz des jeweiligen Teilnetzes

Repeater: Gateway auf Schicht 1

- Aufgabe: Signalverstärkung und -aufbereitung
- Typischerweise verwendet zur Überwindung von Längenrestriktionen (z.B. bei Ethernet)
- Sieht keine Daten, sondern nur Bits
- Ist nicht zur Strukturierung oder Lasttrennung von Netzen einsetzbar
  - bei Ethernet bilden zwei durch Repeater verbundene Teile eine einzige “Kollisionsdomaine”
- Verbundene Teilnetze müssen ab Schicht 2 aufwärts identisch sein
- Spielt keine Rolle, welche Netz-Software auf den darüberliegenden Schichten verwendet wird

Brücke: Gateway auf Schicht 2

- Kopplung von zwei Subnetzen auf Ebene 2



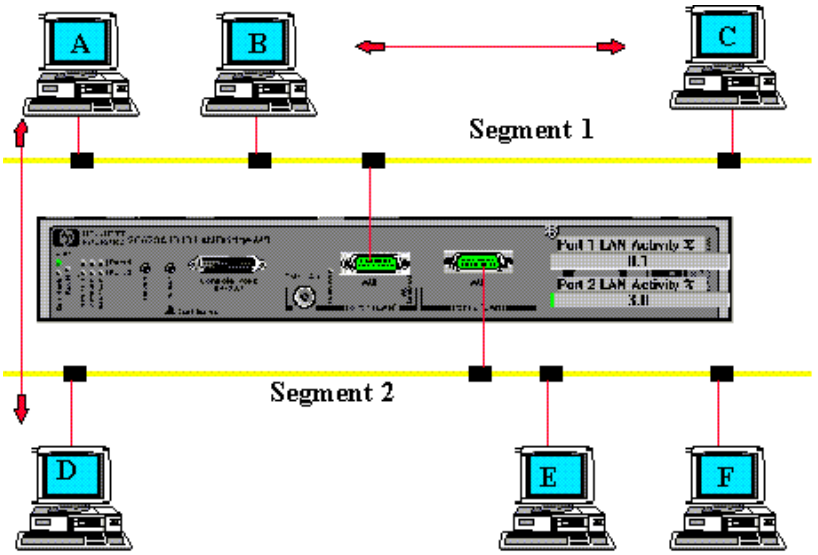
- Checksumme von Paketen wird geprüft
  - im Unterschied zu Repeatern!
  - fehlerhafte Ebene-2-Pakete werden nicht transportiert

- Lasttrennung: Pakete werden nicht transportiert, falls Empfänger auf der gleichen "Seite" wie Sender liegt
  - Broadcastpakete werden immer weitergeleitet
  - *Selbstlernalgorithmus* ("backward learning"):
    - dynamischer Aufbau einer Adresstabelle (typw. Hash-Verfahren)
    - durchleiten, wenn Empfänger (noch) unbekannt
    - Absenderadresse eines Paketes "verrät" Lage des Senders
    - Nachteil: Adressüberprüfung verzögert Datenpakete einige µs oder ms
    - um Topologieänderungen berücksichtigen zu können (z.B.: Station wandert vom einen Subnetz in das andere): Aging-Mechanismus: Eintrag wird nach einigen Minuten "vergessen"

Brücken (2)

- "MAC-level bridge": verbindet nur Subnetze mit gleichem Medienzugangsprotokoll (typw. Ethernet)

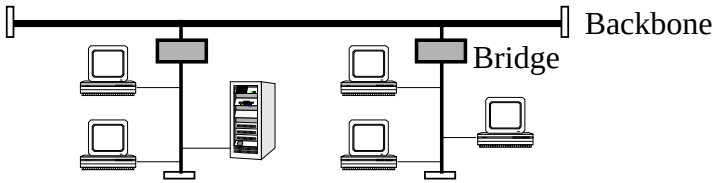
- Einige Brücken können Datenpakete *filtern*
  - String-matching bzgl. bestimmter Adressen, Protokoll-Typen etc.
  - je nach Bedingung Paket durchlassen oder zurückhalten
  - i.a. Bedingungen positiv und negativ definierbar und mit "und" bzw. "oder" kombinierbar
  - Verwendung ggf. zum Zweck der Datensicherheit oder Lastaufteilung
  - Vorsicht: Performance-Verlust bei vielen / aufwendigen Filtern!



Brücken (3)

- Brücken müssen genügend (= ?) Pufferspeicher haben, insbesondere wenn Subnetze verschieden schnell sind
  - Abfangen von Lastspitzen
  - Auch bei gleichartigen Netzen (z.B. Ethernet) notwendig, wenn Zielnetz stärker belastet ist
  - Bei Pufferüberlauf muss Paket vernichtet werden (auf dieser Ebene existiert kein Protokoll zur Wiederholung von Paketen!)
  - Selbst wenn ein Protokoll einer höheren Schicht den Verlust erkennt und behandelt: Sehr ineffizient, da dort i.a. grosse timeout-Werte und i.a. mehrere Pakete dieser Ebene wiederholt werden!

- Typischer Einsatz: Ankoppelung an ein Backbonenetz

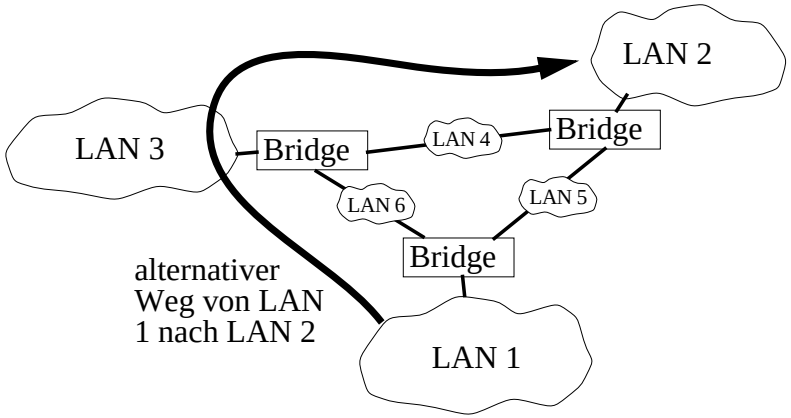


- Leistungsfähigkeit von Brücken
  - transit delay (angegeben ist meist minimaler Wert; tatsächlicher Wert hängt u.a. von den gewählten Filtern, der Stationenzahl etc. ab)
  - frames / s (berechnet i.a. für minimale Paketlängen!)
  - Byte / s (berechnet i.a. für maximale Paketlängen!)

  } sieht jeweils "besser" aus...

Brücken (4)

- Es gibt auch Multiport-Bridges (--> Switch)
  - sternförmige bzw. baumförmige Topologien
  - falls Zieladresse unbekannt oder Broadcast-Adresse: Datenpaket an *alle* anderen Ports weiterleiten ("auffächern")
  - stärkere Vermaschung von LANs (Redundanz, Lastausgleich) möglich:

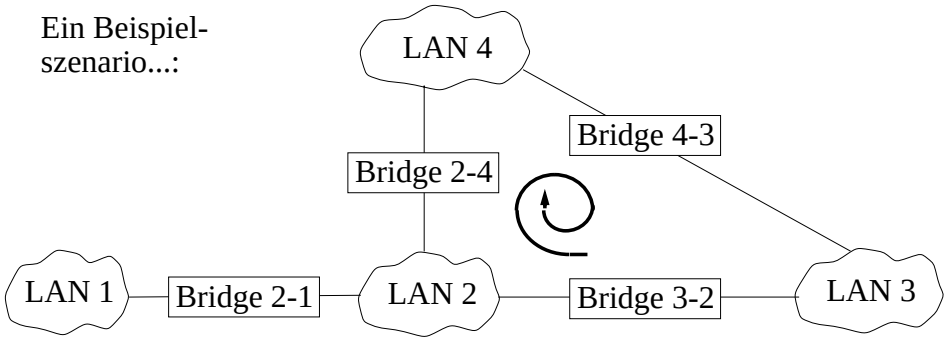


- Mächtige Bridges müssen verwaltet werden
  - Schalten von Ports / Verbindungen
  - Setzen von Filtermasken
  - ...

  } Management-Protokoll

Das Schleifenproblem bei Brücken

Ein Beispiel-szenario...:



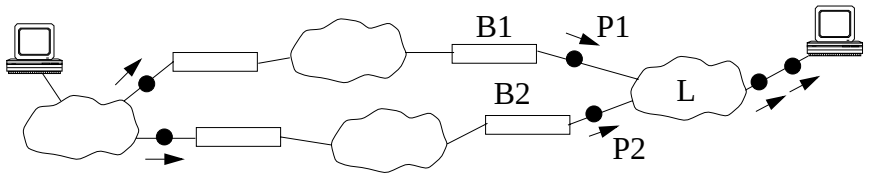
- Rechner aus LAN 3 sendet an Rechner aus LAN 1:
  - --> Bridge 3-2 leitet Paket an LAN 2 weiter
    (Bridge 3-2 weiss, dass Zielrechner “jenseits” liegt)
  - --> Bridge 2-1 *und* Bridge 2-4 leiten das Paket weiter
    (Bridge 2-4 weiss noch nicht, in welchem LAN der Zielrechner liegt)
  - --> Datenpaket kommt so nach LAN 4
  - --> Bridge 4-3 leitet aus gleichem Grund Datenpaket weiter
  - --> Datenpaket ist wieder in LAN 3
  - --> ...und wird von Bridge 3-2 erneut auf die Reise geschickt...

  (2 Schleifen --> ggf. explosionsartige Vermehrung von Paketen!)

- Lösungen für das Problem?

Schleifenproblem: Lösungen?

- Erkennen von Zyklen mit einer eindeutigen “Kontroll-Nachricht”, die eine Bridge aussendet und (nach “einiger” Zeit) wieder empfängt
  - aber kann sich diese nicht selbst in anderen Schleifen vervielfältigen?
- Alterung von Datenpaketen (Hop-Zähler)
  - aber wann ist es “zu alt”?
- Datenpakete merken sich alle besuchten Bridges
  - aber: Platzbedarf in den Paketen!
- Datenpakete haben eine eindeutige Kennung und Bridge merkt sich alle “jemals” erhaltenen
  - aber: Platzbedarf in den Brücken!
- Datenpaket nicht weitergeben, wenn der Absender(!) bekanntermassen “jenseits” liegt
  - aber: darf man annehmen, dass dies immer bekannt ist?
  - vielleicht hat die Bridge den Absender wieder aus ihrer Tabelle gelöscht?

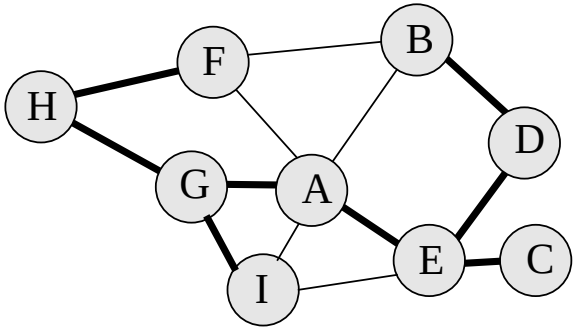


  - und löst dies (auch) das Problem paralleler Pfade (Paketverdoppelung!)?
  - vielleicht warten die Pakete P1, P2 “gleichzeitig” in einem Puffer der Netzkarten von B1 und B2 auf Versand nach L?

- Lösung in der Praxis (IEEE 802.1D): Spannbaum!
  - Aber: Einige Verbindungen bleiben dann ungenutzt, die eigentlich einen Teil der Netzlast abnehmen könnten

IEEE 802.1D: Spanning Tree Protocol

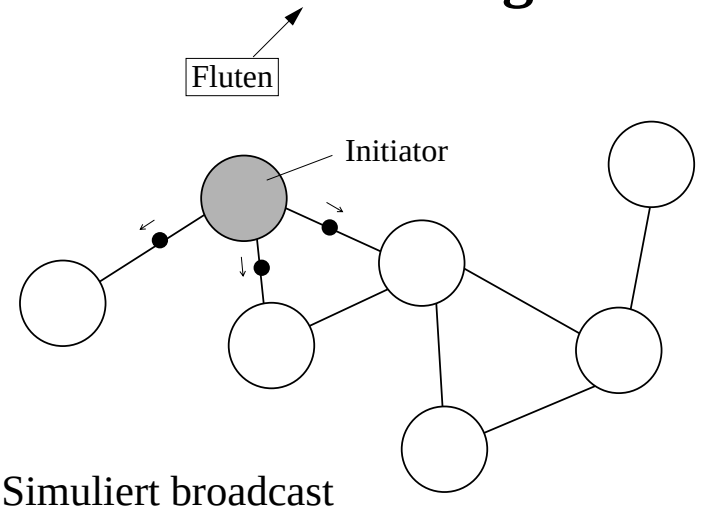
- Bestimmung eines *Spannbaumes* mit *Wurzel* (dann gibt es zwischen je zwei Knoten einen einzigen Weg!)
  - Berücksichtigung von Leitungskosten
  - Bestimmung der Wurzel durch Prioritäten beeinflussbar
  - Automatische Reinitialisierung bei Fehlern (timeout-gesteuert)



- Die Idee des Verfahrens (“HELLO-Protokoll”) basiert auf einem verteilten *Routing-Algorithmus*

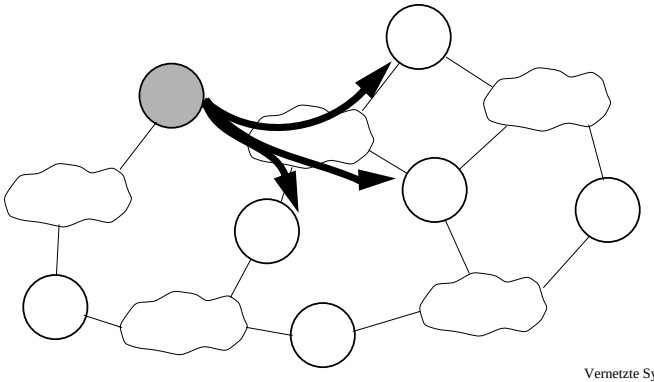
- Wir beschreiben zunächst diesen Routing-Algorithmus, der nach dem Prinzipien des *Flutens* und der verteilten Approximation arbeitet

Das Flooding-Prinzip



- Simuliert broadcast
- Voraussetzung: zusammenhängende Topologie
- Prinzip: jeder erzählt *neues* Gerücht allen anderen Freunden
- Kein Routing etc. notwendig

- Prinzip funktioniert auch noch, wenn die Nachrichten über ein *broadcastfähiges Teilnetz* an mehrere Knoten “gleichzeitig” verteilt werden:



Flooding: Bemerkungen

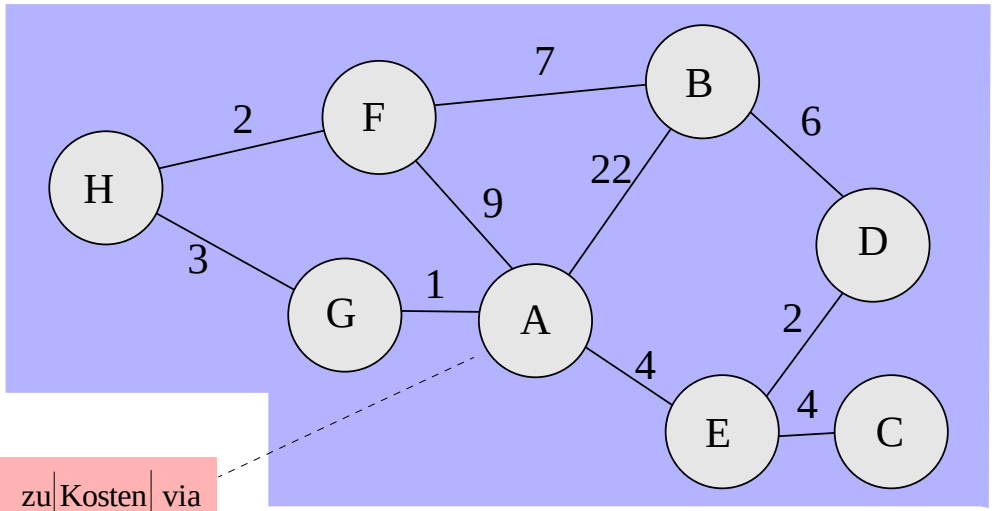
- Begriff “flooding” wird oft auch in einem etwas allgemeineren Sinne für ein naives Routing-Prinzip gebraucht:
  - ein so verbreitetes Datenpaket erreicht “garantiert” den Empfänger, wenn es überhaupt einen Weg dorthin gibt
  - oft: ein empfangenes Datenpaket “öfters” oder “immer wieder” an alle Nachbarn weiterleiten
  - Strategie ist ggf. sinnvoll, wenn hohe Robustheit (Fehlertoleranz) gefordert wird (z.B. militärische Anwendungen)
  - oft auch eine Information so lange wiederholt verbreiten, bis eine neuere Information vorliegt, dann nur noch diese wiederholt verbreiten...
- Typischerweise existiert dann ein Alterungsmechanismus (hop counter, time to live) in den Datenpaketen
  - hop counter mit einem sinnvollen Wert initialisieren (≥ max. Pfadlänge)
  - bei jedem Passieren eines Knotens dekrementieren
  - Datenpaket vernichten, wenn hop counter = 0
- Alternativ: Lebensdauer in Zeiteinheiten
  - gemeinsame Zeitvorstellung notwendig
- Vorteil von flooding: Keine Topologiekenntnis notwendig (kein explizites Routing)
- Nachteil: Sehr aufwendig!

Dezentrale Berechnung von Routingtabellen für kürzeste Wege

Gegeben: Graph mit bewerteten Kanten (ungerichtet)

Kosten: z.B. $, ms, km,... (oder auch "Weglänge")

“Distance Vector Routing”



zu	Kosten	via
A	0	-
B	22	B
C	∞	?
D	∞	?
E	4	E
F	9	F
G	1	G
H	∞	?

Anfangstabelle für Knoten A

- Jeder kennt anfangs die Kosten zu seinen Nachbarn
- "Spontanstart": Sende eigene Tabelle an Nachbarn
- Bei Empfang einer Tabelle über Verbindung mit Kosten g:
  Für alle Zeilen i der Tabelle:
  Falls Nachricht.Kosten[i]+g < Knoten.Kosten[i]:
  ersetze Zeile (Kosten := Kosten+g; via := Absender)
- Falls sich Tabelle verändert hat:
  Neue Tabelle an Nachbarn senden (Ausnahme: Sender)
- Wie Terminierung feststellen? (Ist das notwendig?)

Distance Vector Routing

- Ist eine verteilte Version des Bellman-Ford-Algorithmus
  - ähnlich dem bekannten Dijkstra-Algorithmus für kürzeste Wege
  - " Relexationsprinzip" (Bellman 1958, Dijkstra 1959, Ford 1962)

- Wurde im ARPANET bis 1979 eingesetzt
  - darauf beruhendes RIP-Verfahren (Routing Information Protocol) wird in Teilbereichen des Internet ("Domains") noch oft verwendet; vgl. RFC 1058 (http://ds0.internic.net/rfc/rfc1058.txt)

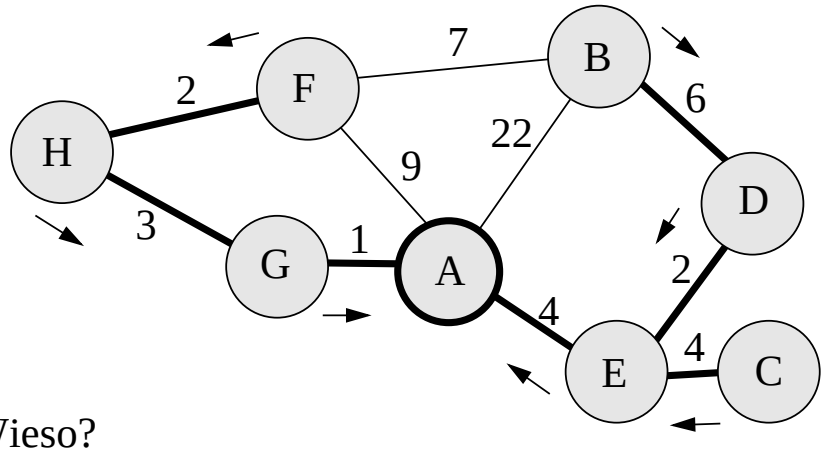
- Oft als dynamischer ("adaptiver") Routing-Algorithmus verwendet mit Neuberechnung und Austausch der Tabellen
  - in regelmässigen Zeitabständen (z.B. Austausch alle 30s bei RIP)
  - wenn sich etwas ändert (Kosten einer Verbindung, z.B. Ausfall einer Leitung oder Änderung der Lastsituation)

- Metrik für die Kosten z.B.:
  - Anzahl der hops (z.B. bei RIP, wobei ≥16 als "∞" gilt)
  - gewichtete Anzahl von hops
  - Bandbreite einer Verbindung
  - Verzögerung einer Verbindung (z.B. gemessen mit Testpaketen)
  - Länge der Paketwarteschlange einer Verbindung; ggf. gemittelt über eine längere Zeit
  - ... (z.B. komplexes Kostenmass aus verschiedenen Faktoren)

Spannbaum aus den Routing-Tabellen

Annahme hier: zwischen je 2 Knoten existiert genau ein kürzester Weg (das lässt sich notfalls durch "Tricks" bei den Kantenkosten erreichen)

- Wähle einen Knoten als *Wurzel* (hier z.B. A). Dann bilden die kürzesten Wege zur Wurzel einen Spannbaum



- Wieso?
  - man überlege sich: die zugehörige Kantenmenge ist *zusammenhängend* und besteht (bei n Knoten) aus *n-1 Kanten*

- Jeder Knoten (≠Wurzel) hat *eine* Kante, die zum Nachbarknoten führt, der am nächsten an der Wurzel liegt
  - diese Kante lässt sich direkt aus der lokalen Routingtabelle ermitteln
  - die Routingtabelle muss dann aber "stabil" sein, d.h. der Routingalgorithmus muss terminiert (konvergiert) sein
  - die Menge dieser Kanten bildet die Kanten des Spannbaumes

Das IEEE 802.1D-Protokoll

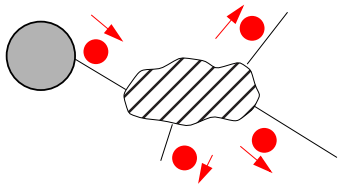
- Das 802.1D-Spannbaumprotokoll lässt sich aus dem Distance Vector Routing-Algorithmus ableiten
  - 802.1D-Protokoll funktioniert in der Praxis ein klein wenig anders, die Unterschiede sind für unsere Zwecke aber unerheblich
  - beachte: es entsteht nicht unbedingt ein (kosten)minimaler Baum!

1) Auf den Knoten (= Brücken) ist eine *Ordnung* festgelegt
   - im Bsp. etwa A > B > C...
   - in der Praxis gegeben durch manuell einstellbare *Prioritäten* (bei gleicher Priorität entscheidet die Hardwareadresse der Bridge)

2) Beim Austausch der Routingtabellen findet *gleichzeitig* eine *Election* mit message-extinction statt:
   - nur der jeweils “höchste” Knoten ist noch Kandidat für die Wurzel
   - bei den Routing-Tabellen interessiert auch nur die Zeile für den Wurzel-Kandidaten; alle anderen Zeilen werden nicht realisiert

3) Die “Entfernungskosten” einer Kante können manuell festgelegt werden
   - defaultmässig durch die Kapazität der Leitung bestimmt

4) Eine Kontrollnachricht einer Brücke wird per multicast an alle Brücken des einen LANs verteilt
   - ein Knoten muss seine Information sowieso an alle Nachbarn senden
   - die Brücken besitzen dazu eine eigene *Gruppenadresse*



multicastfähiges LAN (z.B. Ethernet)

IEEE 802.1D-Protokoll (2)

5) Ende des Algorithmus wird über einen Timer erkannt
   - hoffentlich ist bei Ablauf des Timers bereits “Stagnation” eingetreten!
   - im Bsp. würde dann D wissen, dass die Wurzel A über den Port zu Knoten E ”billigst” erreichbar ist und würde die Kante zu B stilllegen

6) Die Wurzel ist die einzige Brücke (root bridge), die nicht durch höherwertige Nachrichten stillgelegt wurde
   - sie sendet weiterhin im Abstand von ca. 1 - 4 Sekunden ihre Kontrollnachrichten (“HELLO-Pakete” bzw. “Bridge Protocol Data Units” = BPDUs) aus, die über den Spannbaum an alle anderen Brücken verteilt werden

7) Wenn eine andere Brücke längere Zeit keine solche Nachricht erhalten hat, beginnt sie selbst wieder...
   - ggf. wird daraufhin ein etwas anderer Spannbaum ermittelt
   - ggf. wird auch eine neue Wurzel gewählt (wenn die alte ausfiel)

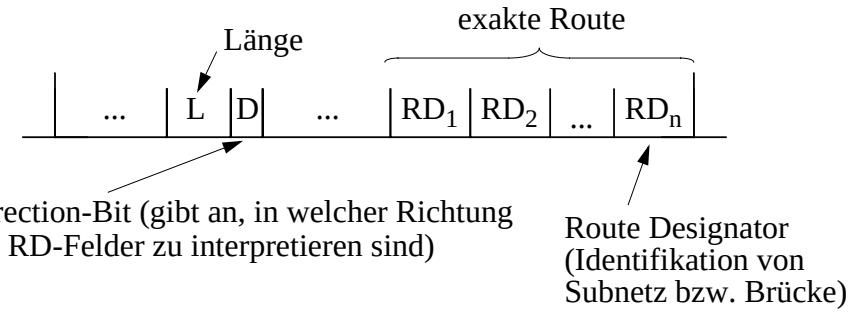
8) Bei einer solchen Rekonfiguration werden alle Ports der Brücken gesperrt und Adresseinträge gelöscht
   - vermeidet so inkonsistente Zwischenzustände (abgehängte Teilnetze, Schleifen...)

Source-Routing-Brücken

- Alternative zum Spannbaum-Prinzip; verwendet vor allem bei gekoppelten Netzen aus Token-Ringen

- Nicht die Brücken halten die Information über Routen, sondern der jeweilige Absender eines Datenpakets

- Datenpakete enthalten dazu ein *Routing-Indicator-Feld*:



- Bridge sucht jeweils das nächste RD-Feld neben ihrem eigenen und leitet Paket entsprechend weiter

- Der Absender eines Paketes muss daher die RD-Felder vollständig füllen

- Wie lernt ein Sender die Route zum Empfänger?

Folge von Subnetzen und Brücken

Route-Discovery

- Falls einem Sender die Route zum Empfänger noch unbekannt ist, tut er folgendes:
  - Senden eines speziellen Testpakets (“discovery frame”)
  - dies wird von allen Brücken in alle anderen Richtungen weiterverteilt
  - dabei werden sukzessive die RD-Felder mit der durchlaufenen Route gefüllt
  - Zyklen werden erkannt: ein Testpaket wird nur weitergereicht, wenn die nächste Etappe noch nicht in einem RD-Feld steht

- Der Empfänger sendet jedes so erhaltene Paket zurück
  - und zwar auf der gleichen Route (Direction-Bit ändern!)

- Falls es verschiedene Wege zwischen Sender und Empfänger gibt, erhält der Sender mehrere Testpakete zurück
  - wieviele Pakete zurückkommen sollen, weiss der Sender i.a. nicht
  - Sender kann sich eine Route auswählen (z.B. erste Antwort mit weniger als x hops; kürzeste Route innerhalb eines Zeitintervalls...)
  - potentielles Problem: Explosion der Zahl der Testpakete ! (Abhilfe: Vorgabe eines Spannbaums, den Testpakete ausschliesslich nutzen)

- Vergleich Source-Routing <--> Spannbaum-Verfahren:
  - Source-Routing kann ggf. redundante Verbindungen und parallele Pfade nutzen (z.B. zur Lastverteilung)
  - Brücken für Source-Routing haben weniger Arbeit
  - dafür muss der Sender das Routing übernehmen (Adresstabellen führen)
  - Source-Routing kann sehr hohe Zahl von Testpaketen verursachen
  - Spannbaum-Verfahren aus Sicht der Stationen transparenter

Internetworking

- Es gibt viele verschiedene Netze (LAN, MAN, WAN,...)
- Es gibt verschiedene Protokollstandards
  - verschiedene Protokolle auf allen Ebenen...
  - z.B. TCP/IP, SNA, IPX, Appletalk...
- Es gibt verschiedene Techniken mit unterschiedlichen Eigenschaften (Funk, Glasfaser, Satelliten...)
- Es gibt unterschiedliche Bedürfnisse, Marktsegmente...

- Dennoch sollen verschiedene Netze miteinander verbunden werden und “so gut es geht” eine Einheit bilden
  - hinsichtlich interessanter Dienste (z.B. E-mail, Dateitransfer...)
  - Menge gekoppelter Netze bilden dann ein “internet” !
- Koppelung auf unterschiedlichen Ebenen möglich, z.B.:
  - Router (Ebene 3)
  - Transport Gateway (Ebene 4)
  - Application Gateway (Ebene 7)
- Problem: Netze sind unterschiedlich
  - z.B. bzgl. Adressierungskonventionen, Paketgrössen, Flusssteuerung...
  - dadurch Probleme mit der *Transparenz*
- Mit IP (“Internet Protocol”) soll eine Koppelung von Rechnern verschiedener Netze erreicht werden

Router

- Koppelung von Netzen auf Ebene 3
  - Verbindet ggf. unterschiedliche Ebene 2 - Protokolle
- Wesentliche Aufgabe: Routing
  - Wegewahl über grössere Distanzen und mehrere “hops”
  - hierfür Kooperation verschiedener Router notwendig! --> Protokoll
- Kein “Plug and Play”
  - Router müssen konfiguriert werden
  - sind oft echte, “spezialisierte” Rechner
- Pakete werden nur weitergereicht, wenn Zieladresse dem Router im Prinzip bekannt ist (--> Routingtabelle)
- Broadcasts aus LANs werden nicht weitergereicht
- Typische Anwendung: Koppelung von LAN und WAN