

# Rechnernetze: (7) Routing



**Prof. Dr. Klaus-Peter Kossakowski**



# Gliederung der Vorlesung

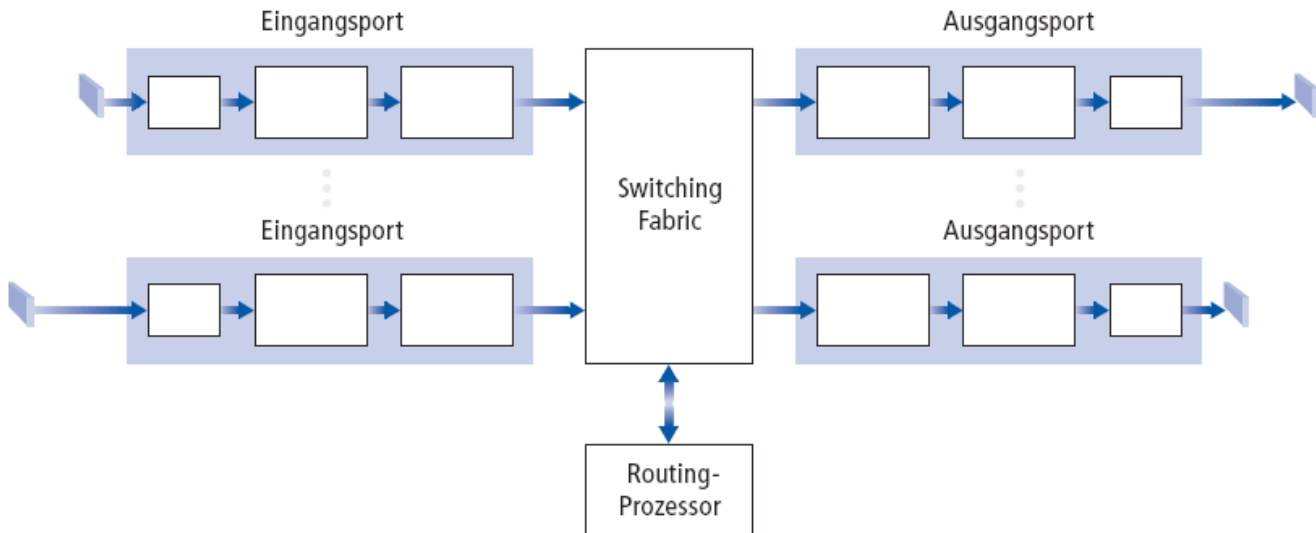
- Einführung und Historie des Internets
- Schichtenmodell
- Netzwerk als Infrastruktur
- Layer 7: Anwendungsschicht
- Layer 7/4: Socket-Programmierung
- Layer 4: Transportschicht
- Layer 3: Netzwerkschicht
  - Routing, ...
  - Firewall und Verwandtes
- Layer 2: Sicherungsschicht



# Übersicht: Routerarchitektur

## Zwei wichtige Aufgaben eines Routers:

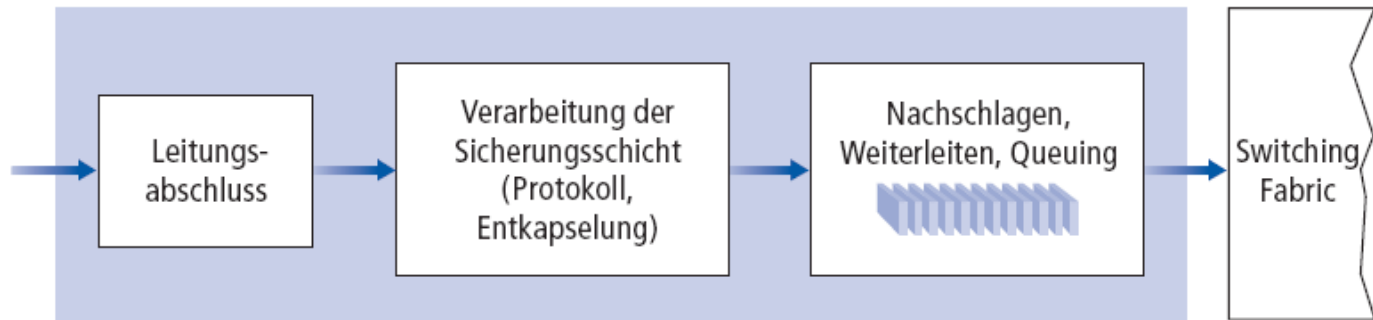
- Ausführen von Routing-Algorithmen und -Protokollen
  - RIP, OSPF, BGP
- Weiterleiten von Datagrammen von einem eingehenden zu einem ausgehenden Link





# Verarbeitung im Eingangsport

- **Leitungsabschluss:**  
physikalische Schicht, Bits empfangen
- **Sicherungsschicht:**  
z.B. Ethernet (kommt noch)





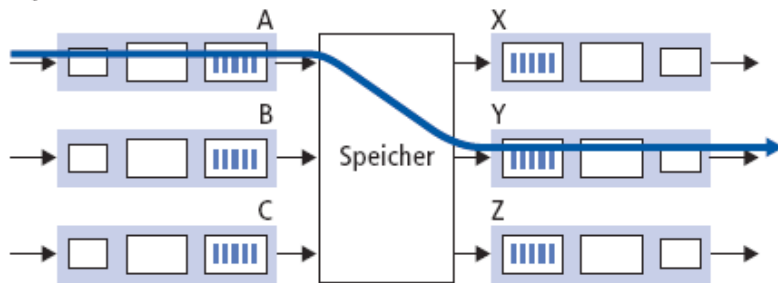
# Verarbeitung im Eingangsport (2)

- **Nachschlagen, Weiterleiten, Queuing:**
  - Suche nach einem geeigneten Ausgangsport
  - Kopie der Routing-Tabelle (oder Teile davon) notwendig, sofern dezentral
- **Ziel: Behandlung der Pakete mit „line speed“, also mit der Geschwindigkeit der Eingangsleitung des Ports**
  - Puffern von Paketen, wenn die Switching Fabric belegt ist

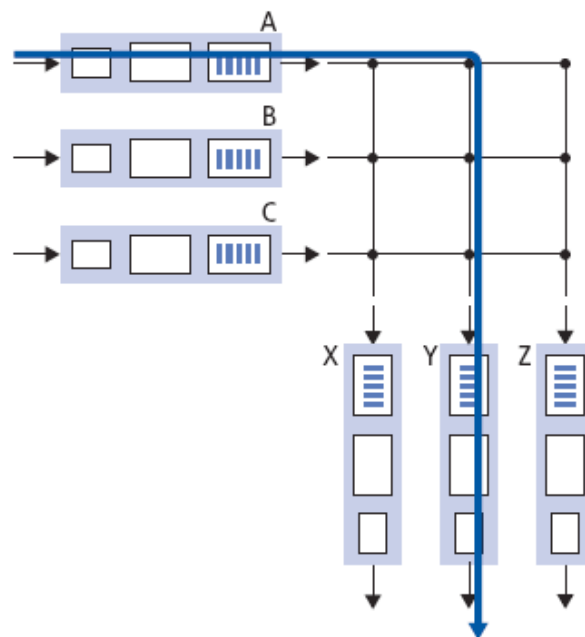


# Arten von Switching Fabrics

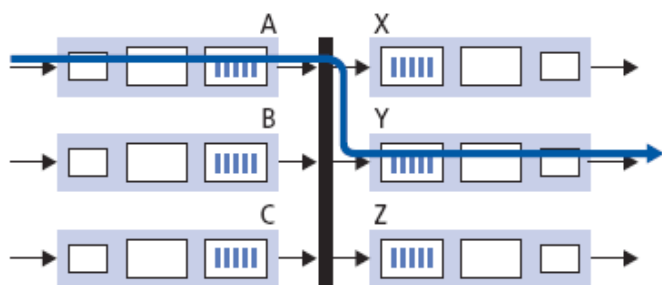
**Speicher**



**Crossbar**



**Bus**



Legende:

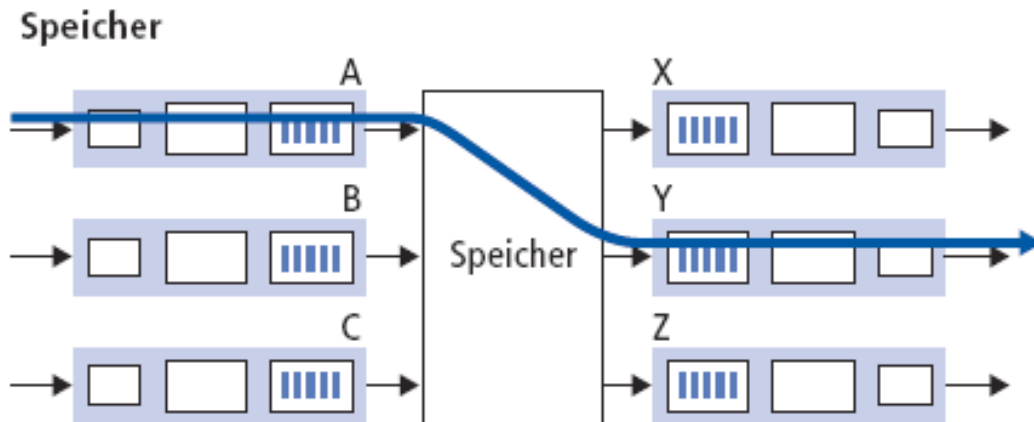




# Switching über den Speicher

## - Erste Routergeneration

- „Normale“ Rechner, Switching wird über die CPU durchgeführt
- Geschwindigkeit durch Speicherbus beschränkt!
- Zwei Speicherzugriffe: einer zum Schreiben, einer zum Lesen

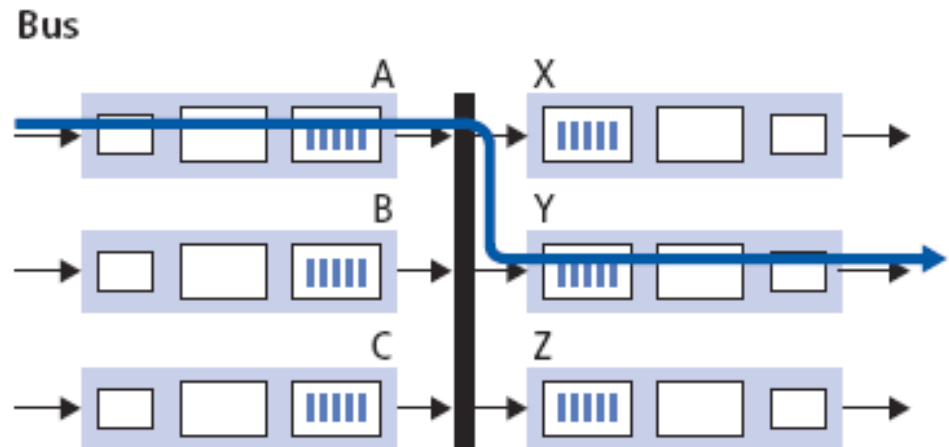


# Switching über einen Bus

## - Weiterentwicklungen



- **Bus Contention: Die gesamte Kommunikation erfolgt über den Bus**
  - dieser beschränkt die Bandbreite des Routers, allerdings auch nur eine Operation
- **Beispiel:  
32-Gbps-Bus  
Cisco 5600**



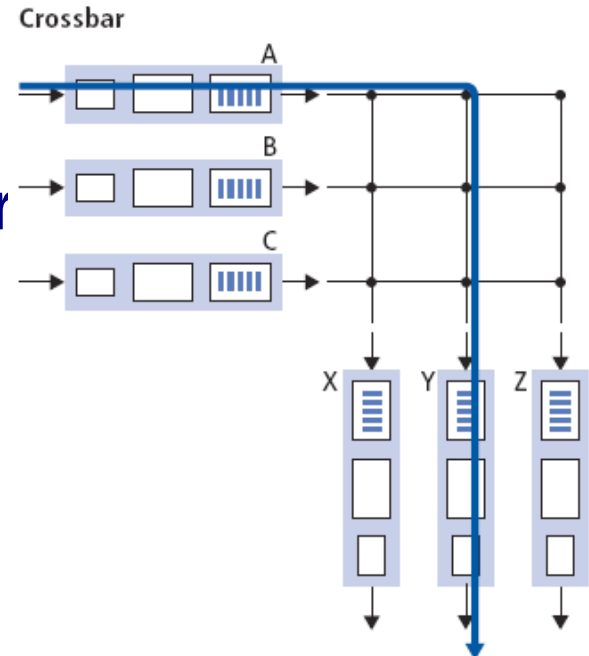


# Switching über ein Spezialnetz

## - auch im Backbone



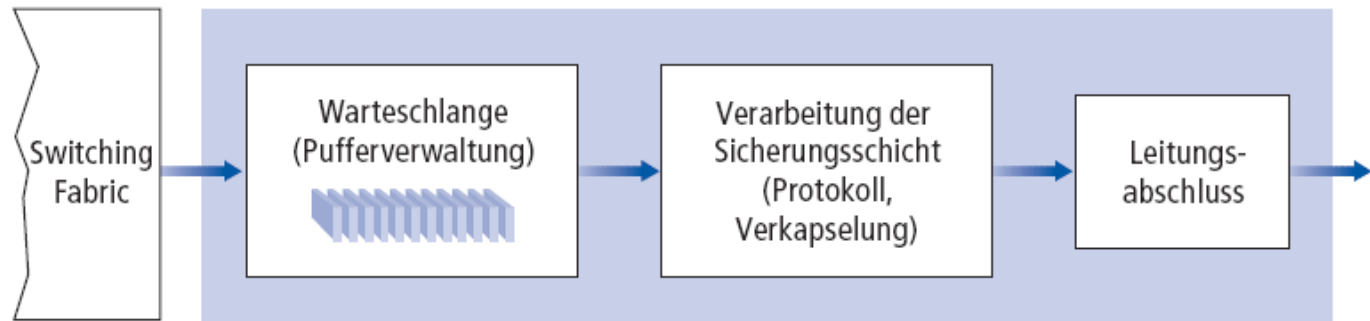
- Ports sind über ein Netzwerk miteinander verbunden
  - Beispielsweise alle Eingangsports über einen Crossbar mit allen Ausgangs-ports
- Zerlegen der Pakete in Zellen fester Größe, Zellen werden schneller durchgeleitet
- Beispiel:  
60 Gbps  
Cisco 12000





# Verarbeitung im Ausgangsport

- Prinzipiell: analog zum Eingangsport!
- Einfacher, da die Entscheidung über die Weiterleitung schon getroffen ist

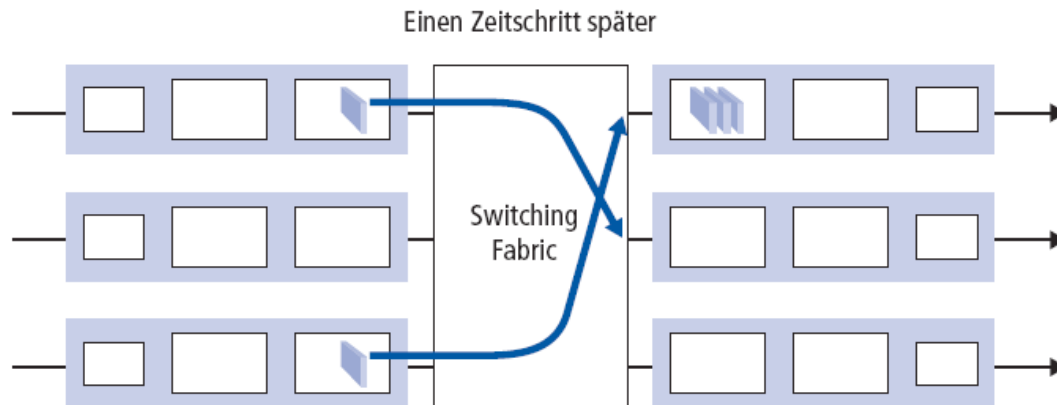
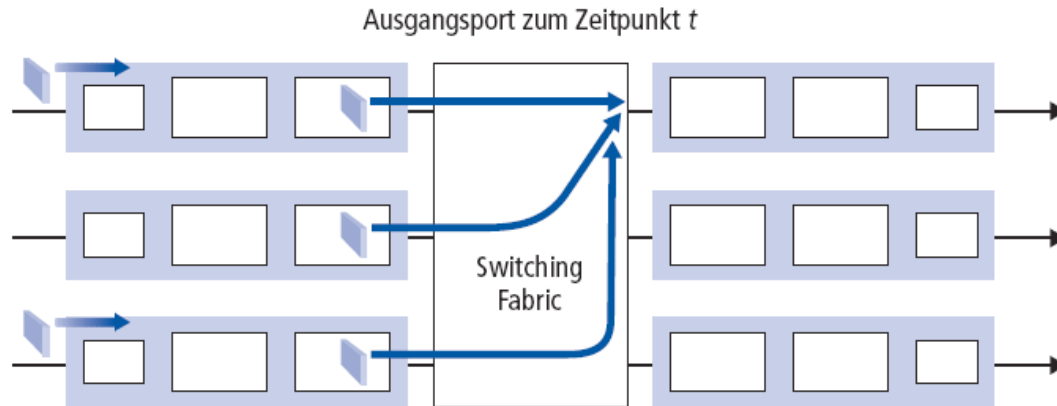




# Puffern im Ausgangsport

- **Puffern von Paketen immer dann, wenn sie schneller aus der Switching Fabric kommen, als sie auf die Leitung gelegt werden können**
- **Auswirkungen:**
  - Gepufferte Pakete werden verzögert
  - Wenn der Puffer überläuft, müssen Pakete verworfen werden
- **„Scheduling Discipline“: bestimmt die Reihenfolge, in der gepufferte Pakete auf die Leitung gelegt werden**

# Puffern im Ausgangsport





**Wieviel sollen wir puffern?**





# Wie groß sollten die Puffer sein?

- **RFC 3439 beschreibt folgende Faustregel:**

- Die Größe des Puffers sollte der Rundlaufzeit (RTT, z.B. 250 ms) multipliziert mit der Datenrate des Links entsprechen

- **Also:**

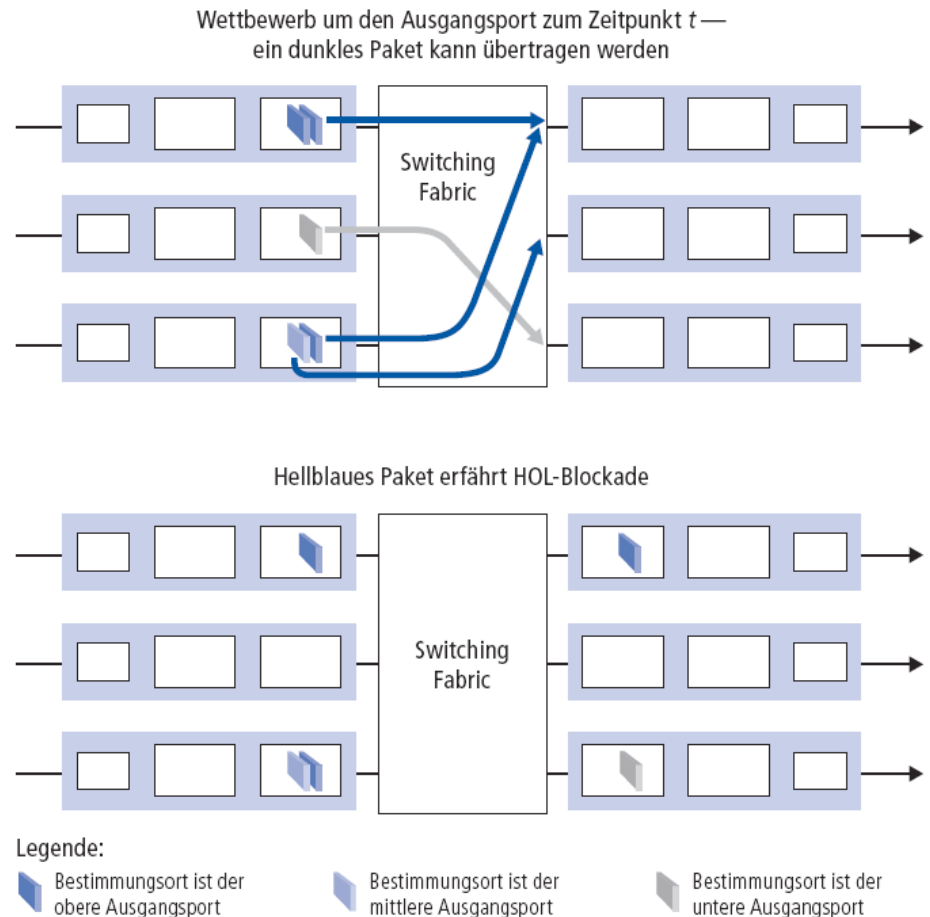
- 10 Gps Link, 250 ms RTT = 2,5 Gbit Puffer

- **Neuere Empfehlungen:**  
**bei N Datenflüssen und Datenrate C des Links gilt:**

$$\frac{RTT \cdot C}{\sqrt{N}}$$

# Puffern im Inputport

- Wenn die Switching Fabric ein Paket nicht direkt weiterleiten kann, muss dieses im Eingangsport gepuffert werden





# Puffern im Inputport

- Wenn die Switching Fabric ein Paket nicht direkt weiterleiten kann, muss dieses im Eingangsport gepuffert werden
- Dort kann es ein Paket blockieren, welches eigentlich bereits durch die Switching Fabric geleitet werden könnte
  - Head-of-Line (HOL) Blocking



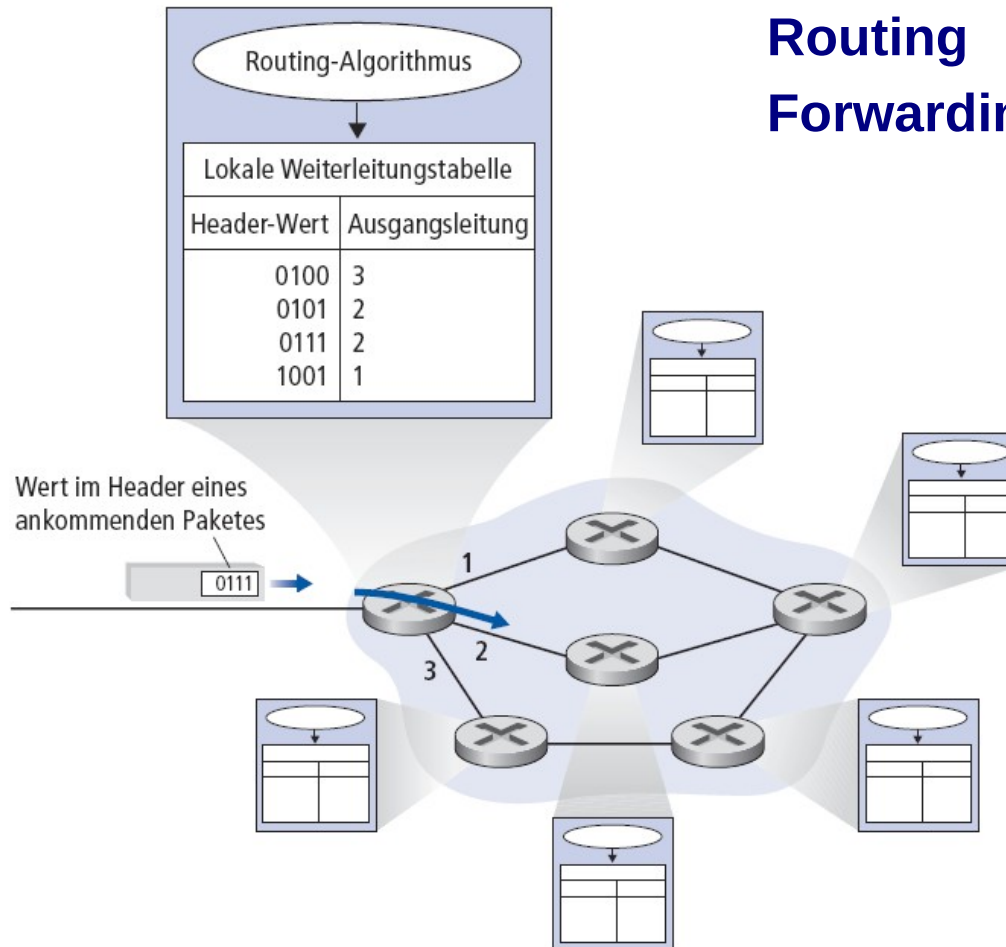


# Routing-Algorithmen

# Routing und Weiterleitung



**Routing = Wegewahl**  
**Forwarding = Weiterleitung**



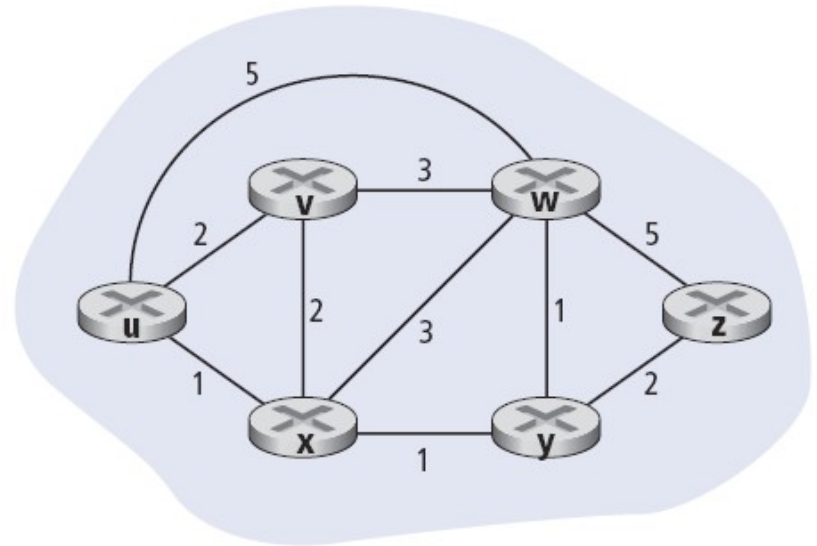


# Ein Netzwerk als Graph

Graph:  $G = (N, E)$

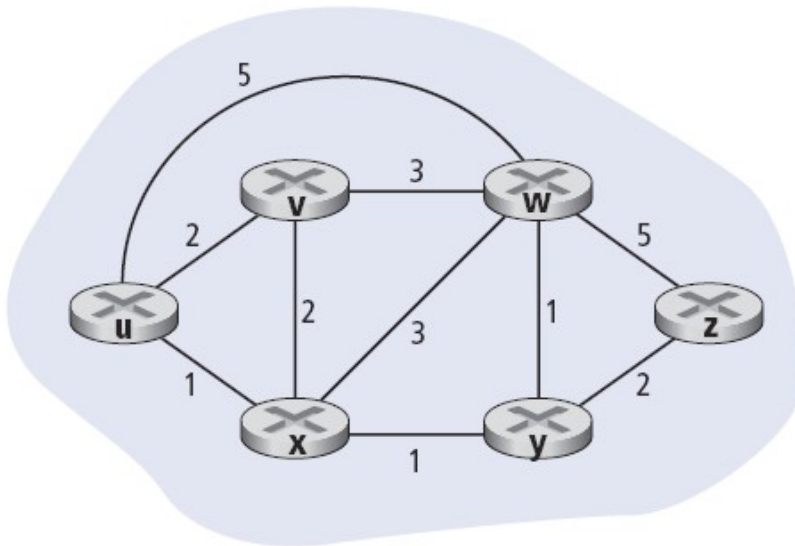
$N$  = Menge von Routern =  
 $\{ u, v, w, x, y, z \}$

$E$  = Menge von Links =  
 $\{ (u,v), (u,x), (u,w), (v,x), (v,w),$   
 $(x,w), (x,y), (w,y), (w,z), (y,z) \}$



## Was ist der günstigste Weg von U nach Z?

# Kosten

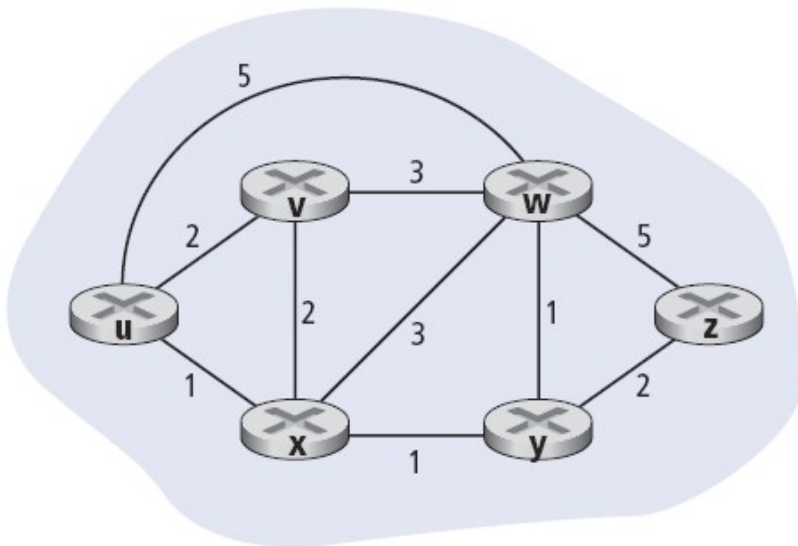


- $c(x,x')$  = Kosten von Link  $(x,x')$ 
  - z.B.  $c(w,z) = 5$
- Kosten können:
  - immer 1 sein
  - invers proportional zur Datenrate sein
  - invers proportional zum Verkehrsaufkommen sein
  - ...

Kosten eines Pfades  $(x_1, x_2, x_3, \dots, x_p)$   
 $= c(x_1, x_2) + c(x_2, x_3) + \dots + c(x_{p-1}, x_p)$



# Kosten



- $c(x,x')$  = Kosten von Link  $(x,x')$ 
  - z.B.  $c(w,z) = 5$
- Kosten können:
  - immer 1 sein
  - invers proportional zur Datenrate sein
  - invers proportional zum Verkehrsaufkommen sein
  - ...

Kosten eines Pfades  $(x_1, x_2, x_3, \dots, x_p)$   
 $= c(x_1, x_2) + c(x_2, x_3) + \dots + c(x_{p-1}, x_p)$

## Routing-Algorithmen finden den günstigsten Weg!

# Routingstabelle

## statisch oder dynamisch?



### Statisch:

- Routen ändern sich langsam/selten
- Einmaliges Ausführen des Algorithmus OK
- Manuelle Konfiguration (manchmal) OK

### Dynamisch:

- Routen ändern sich schnell/ständig
- Periodisches Ausführen des Algorithmus notwendig
- Als Reaktion auf Änderungen von Links
- **Manuelle Konfiguration nicht möglich!**

# Routinginformationen global oder dezentral



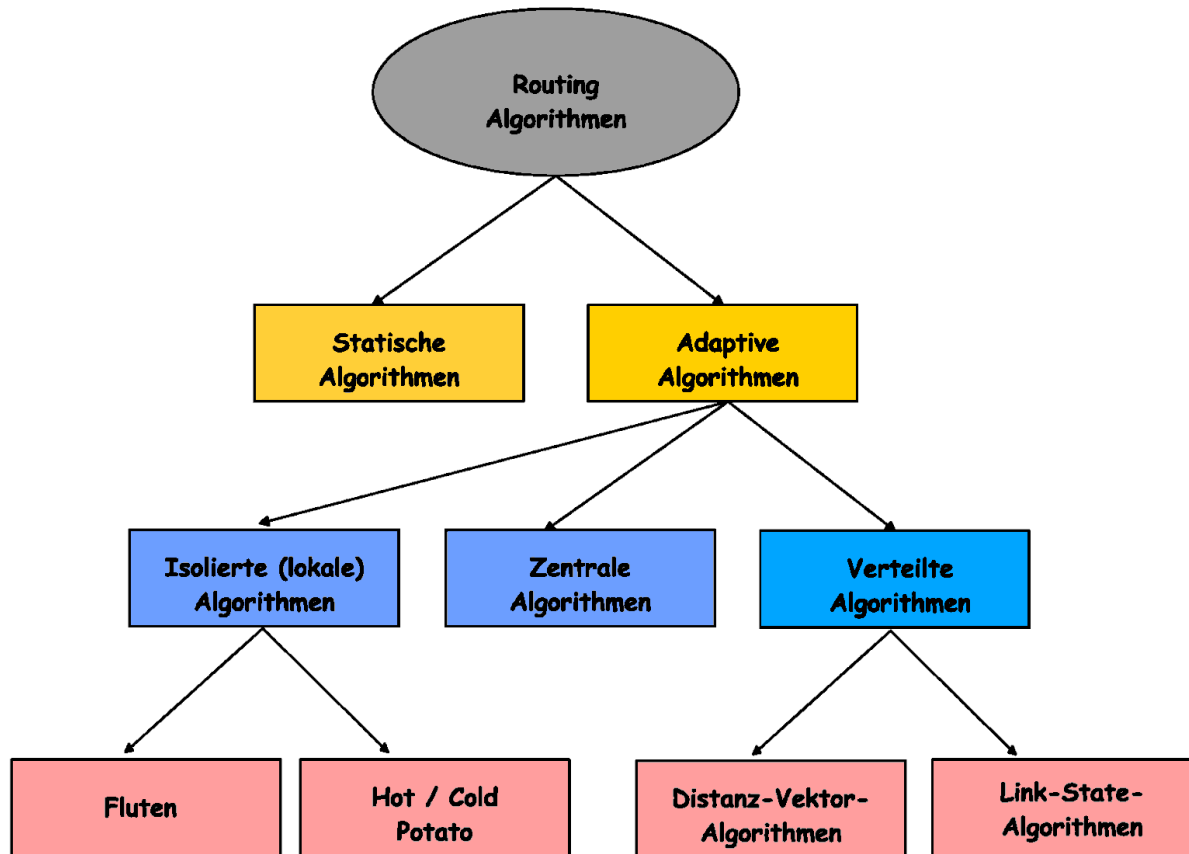
## Globale verfügbar:

- Die vollständige Topologie des Graphen (alle Knoten, alle Kanten, alle Kosten) ist bekannt
  - **Link-State-Routing**

## Dezentrale Informationen:

- Router kennt die Kanten und Kosten zu seinen direkten Nachbarn
- Router tauschen sich mit Nachbarn aus
  - **Distance-Vector-Routing**

# Übersicht der Routing-Algorithmen





# Link-State-Routing-Algorithmus

## → Dijkstras-Algorithmus



- **Alle Knoten, Kanten und Kosten sind in allen Knoten bekannt**
  - Durch Fluten von Link-State-Informationen im Netz haben alle das gleiche Wissen
- **Mit dem Algorithmus wird der günstigste Pfad von dem betrachteten Router (der Quelle) zu allen anderen Knoten bestimmt**
- **Iteratives Vorgehen:**  
Nach k Iterationen sind die günstigsten Pfade zu den k am günstigsten zu erreichenden Zielen bekannt

# Link-State-Routing-Algorithmus

## → Dijkstras-Algorithmus



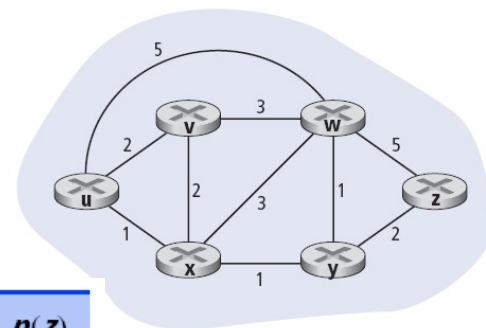
### ■ Details zum Algorithmus

- $N$ : Menge aller Knoten
- $c(x,y)$ : Kosten des Links von  $x$  nach  $y$ 
  - ist  $\infty$  wenn  $x$  und  $y$  keine Nachbarn sind
- $D(v)$ : Kosten des günstigsten derzeit bekannten Pfades von der Quelle zu  $v$
- $p(v)$ : Vorgängerknoten entlang des Pfades von der Quelle nach  $v$
- $N'$ : Menge der Knoten, für die der günstigste Pfad definitiv feststeht



# Dijkstra-Algorithmus (u = Quelle)

```
1 initialisiere:
2    $N' = \{u\}$ 
3   für alle Knoten v aus N
4     wenn v ein Nachbar von u ist
5       dann  $D(v) = c(u,v)$ 
6     sonst  $D(v) = \infty$ 
7
```



Schritt	$N'$	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	2, u	5, u	1, u	$\infty$	$\infty$



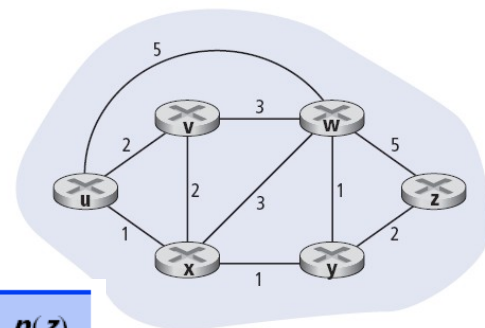
# Dijkstra-Algorithmus (u = Quelle)

```
1 initialisiere:
2   N' = {u}
3   für alle Knoten v aus N
4     wenn v ein Nachbar von u ist
5       dann  $D(v) = c(u,v)$ 
6     sonst  $D(v) = \infty$ 
7
8 wiederhole:
9   finde ein w aus N welches nicht in N' ist, so dass D(w) minimal ist
10  füge w zu N' hinzu
11  Berechne D(v) neu für jeden Nachbarn v von w der nicht in N' ist:
12     $D(v) = \min(D(v), D(w) + c(w,v))$ 
13  /* die neuen Kosten nach v sind entweder die alten Kosten
14  oder die Kosten nach w plus die Kosten von w nach v */
15 bis N' = N
```



# Dijkstra-Algorithmus (u = Quelle)

```
8  wiederhole:
9    finde ein w aus N welches nicht in N' ist, so dass D(w) minimal ist
10   füge w zu N' hinzu
11   Berechne D(v) neu für jeden Nachbarn v von w der nicht in N' ist:
12      $D(v) = \min(D(v), D(w) + c(w,v))$ 
13   /* die neuen Kosten nach v sind entweder die alten Kosten
14   oder die Kosten nach w plus die Kosten von w nach v */
15 bis N' = N
```

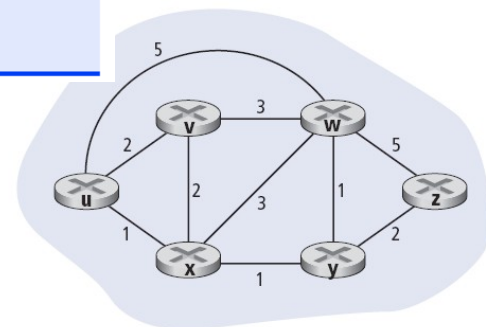


Schritt	$N'$	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	$u$	$2, u$	$5, u$	$1, u$	$\infty$	$\infty$
1	$ux$	$2, u$	$4, x$		$2, x$	$\infty$



# Dijkstra-Algorithmus (u = Quelle)

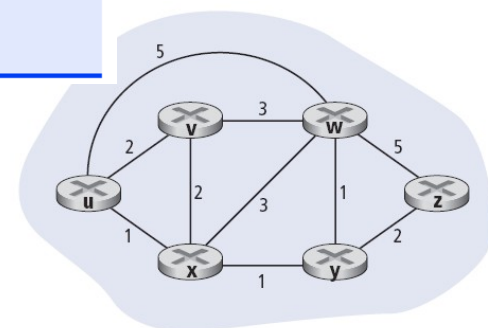
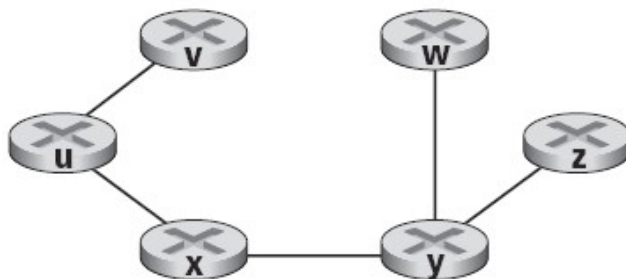
Schritt	$N'$	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	$u$	2, $u$	5, $u$	1, $u$	$\infty$	$\infty$
1	$ux$	2, $u$	4, $x$		2, $x$	$\infty$
2	$uxy$	2, $u$	3, $y$			4, $y$
3	$uxyv$		3, $y$			4, $y$
4	$uxyvw$					4, $y$
5	$uxyvwz$					





# Dijkstra-Algorithmus (u = Quelle)

Schritt	$N'$	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	$u$	2, $u$	5, $u$	1, $u$	$\infty$	$\infty$
1	$ux$	2, $u$	4, $x$		2, $x$	$\infty$
2	$uxy$	2, $u$	3, $y$			4, $y$
3	$uxyv$		3, $y$			4, $y$
4	$uxyvw$					4, $y$
5	$uxyvwz$					



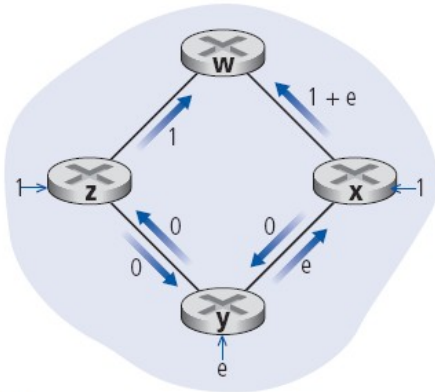
# Wie gut ist der Dijkstras-Algorithmus?



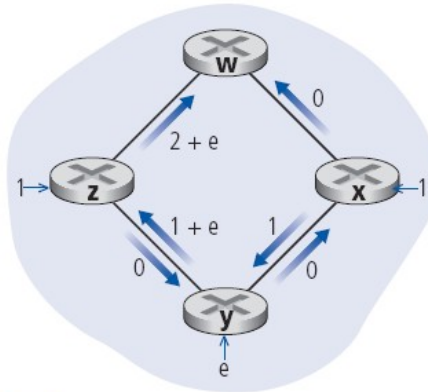
- Rechenkomplexität für  $n$  Knoten wird bestimmt durch die Iteration:  
alle Knoten, die nicht in  $N'$  sind, überprüfen  
 $n * (n+1) / 2 \rightarrow$  Vergleich:  $O(n^2)$
- Effizientere Implementierung möglich, so  
das die Komplexität sinkt:  $O(n \log n)$



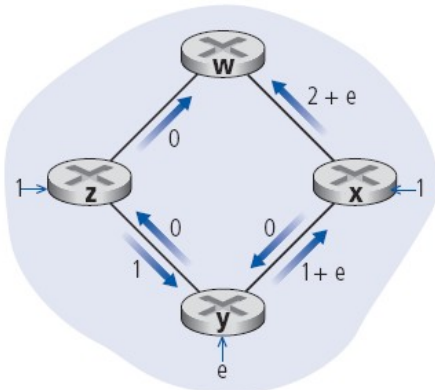
# Wie gut ist der Dijkstras-Algorithmus?



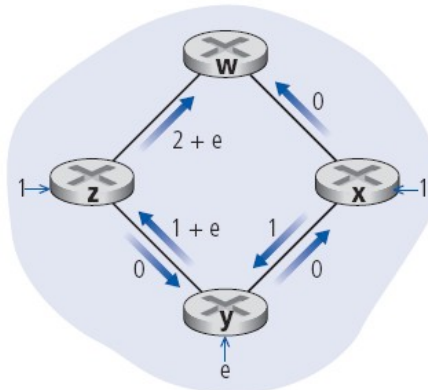
**a** Anfängliches Routing



**b** x, y entdecken den im Uhrzeigersinn verlaufenden besseren Pfad nach w



**c** x, y, z entdecken den gegen den Uhrzeigersinn verlaufenden besseren Pfad nach w



**d** x, y, z entdecken den im Uhrzeigersinn verlaufenden besseren Pfad nach w

**Oszillationen sind möglich, zum Beispiel, wenn die Metrik für die Kosten der Pfade von der Netzwerklast abhängt**



# Distance-Vector-Algorithmus

- Prinzipielle Idee basiert auf der Bellman-Ford-Gleichung
- Sei  $d_x(y)$  der billigste Pfad von x nach y
- Dann gilt:

$$d_x(y) = \min \{ c(x,v) + d_v(y) \}$$

- Wobei das Minimum über alle Nachbarn v von x gebildet wird

# Distance-Vector-Algorithmus

## → Bellman-Ford-Algorithmus



□ Offensichtlich gilt:

□  $d_v(z)=5, d_x(z) = 3, d_w(z) = 3$

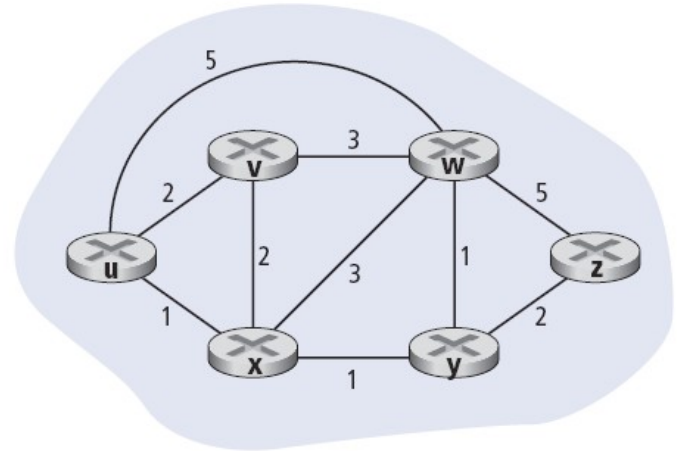
□ Die Aussage der Bellman-Ford-Gleichung ist dann:

$$d_u(z) = \min \{ c(u,v) + d_v(z), \\ c(u,x) + d_x(z), \\ c(u,w) + d_w(z) \}$$

$$= \min \{ 2 + 5, 1 + 3, 5 + 3 \} = 4$$

□ Der Nachbar, über den das Minimum erreicht wird, wird für das betrachtete Ziel in die Routing-Tabelle eingetragen

□ z wird von u aus am besten über x erreicht



# Distance-Vector-Algorithmus

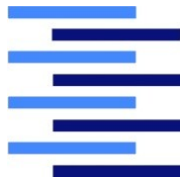
## → Bellman-Ford-Algorithmus



- $D_x(y)$  schätzt die günstigsten Kosten für einen Pfad von  $x$  nach  $y$
- Knoten  $x$  kennt die Kosten zu jedem Nachbarn  $v$ :  $c(x,v)$
- Knoten  $x$  führt einen Distanzvektor  
 $D_x = [D_x(y): y \in N]$
- Knoten  $x$  merkt sich die Distanzvektoren der Nachbarn
  - Für jeden Nachbarn  $v$  merkt sich  $x$   
 $D_v = [D_v(y): y \in N]$

# **Distance-Vector-Algorithmus**

## **→ Bellman-Ford-Algorithmus**



- Wenn ein Knoten hochgefahren wird, dann sendet dieser seinen eigenen Distanzvektor an alle seine Nachbarn
- Immer wenn sich der eigene Distanzvektor verändert, wird dieser an alle Nachbarn gesendet
- Erhält ein Knoten einen Distanzvektor von einem Nachbarn, überprüft er seinen eigenen Distanzvektor

# Bellman-Ford- und Dijkstras-Algorithmus im Vergleich



## ■ Anzahl von Nachrichten:

### ■ Dijkstras / Link State:

- n Knoten, E Kanten:  
O( $n * E$ ) Nachrichten im gesamten Netz

### ■ Bellman-Ford / Distance Vector:

- Nachrichten werden nur mit Nachbarn ausgetauscht
- Zeit bis zur Konvergenz variiert

# Bellman-Ford- und Dijkstras-Algorithmus im Vergleich



## ■ Geschwindigkeit der Konvergenz

### ■ Dijkstras / Link State:

- Fluten der Zustände der Links
- Kann oszillieren

### ■ Bellman-Ford /Distance Vector:

- variiert stark
- Temporäre Routing-Schleifen sind möglich
- Count-to-infinity-Problem

# Bellman-Ford- und Dijkstras-Algorithmus im Vergleich



- **Was passiert, wenn ein Router fehlerhaft ist? (Robustheit)**
  - Dijkstras / Link State:
    - Knoten kann falsche Kosten für einen Link fluten
    - Pfade möglicherweise nicht mehr optimal
  - Bellman-Ford /Distance Vector:
    - Router kann falsche Kosten für einen ganzen Pfad ankündigen
    - Fehler propagiert durch das ganze Netzwerk, insbesondere wenn die (falschen) Kosten klein sind mit u. U. Katastrophalen Folgen





# Routing im Netz der Netze



# Aber die Welt routet anders ...

## ■ Wunschdenken:

- Alle Router sind gleich
- Das Netzwerk ist „flach“ / keine Hierarchien

## ■ Warum ist die Welt anders?

- 200 Millionen Ziernetzwerke
- Platz in Routing-Tabelle reicht nicht
- Routing-Protokolle würden alles überlasten

## ■ Außerdem ...

- Jede Organisation hat eigene Präferenzen...



# Idee der Autonomen Systeme

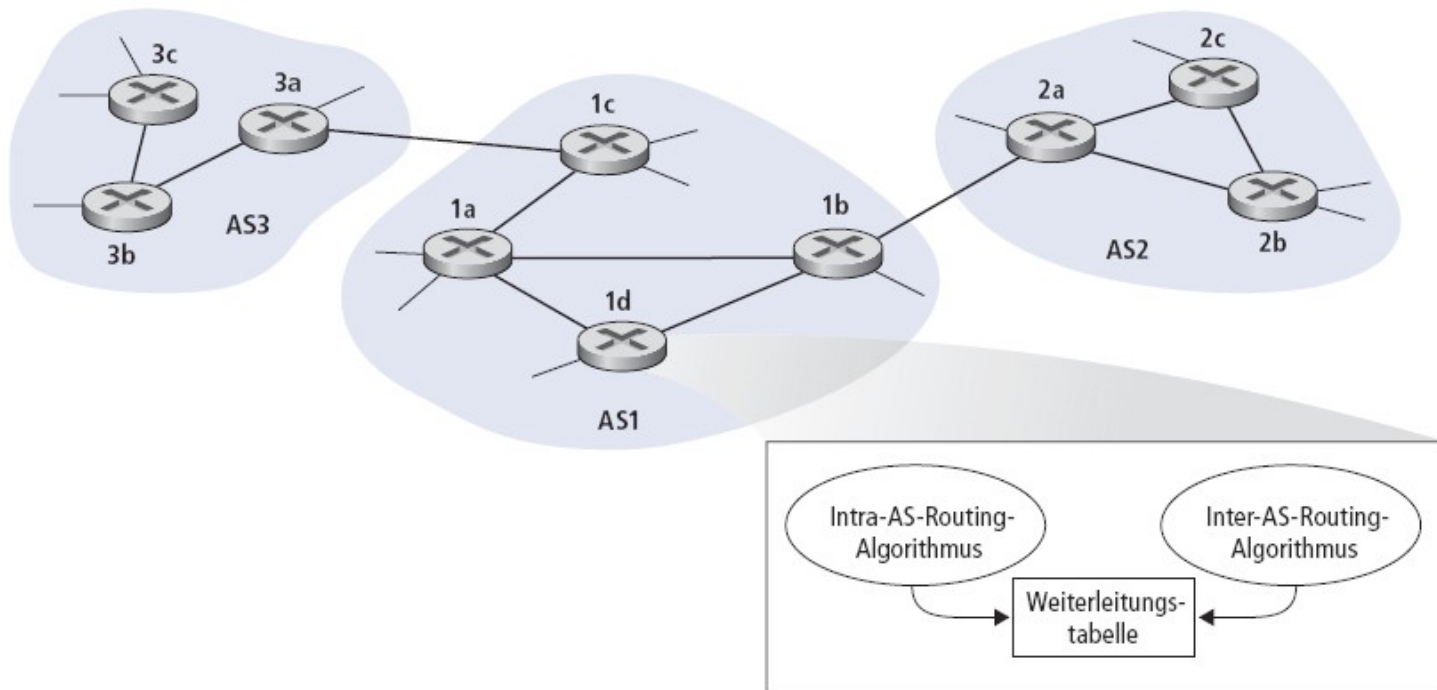
- Router werden Regionen zugeordnet, die wir autonome Systeme (AS) nennen
- Für jedes AS ist eine Organisation zuständig
- Router innerhalb eines AS verwenden eines der vorgestellten Routing-Protokolle
  - “Intra-AS”-Routing-Protokoll
- Verbindung zwischen unterschiedliche AS
  - durch sogenannte Gateway-Router
  - „Inter-AS“-Routing-Protokoll



# Verbundene Autonome Systeme

## ■ Routing-Tabelle wird gefüllt

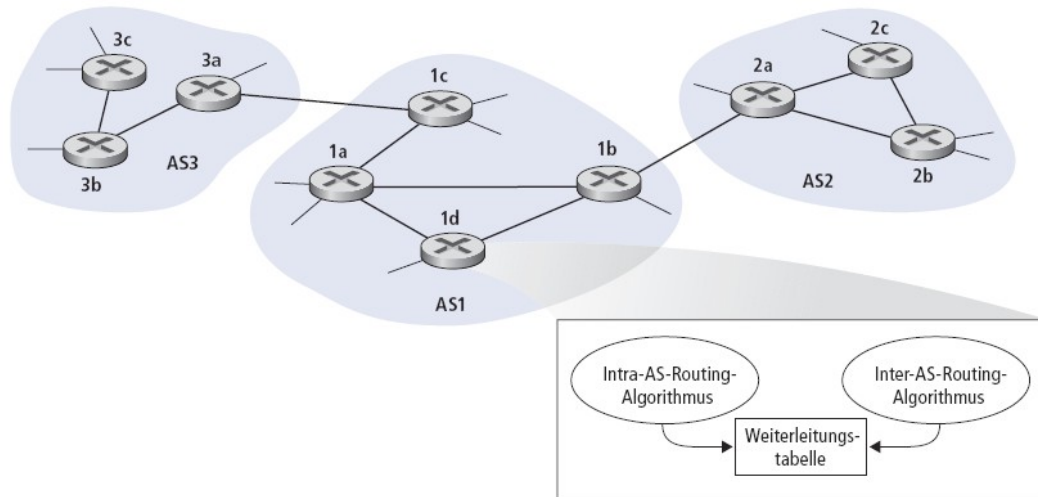
- Intra-AS-Einträge für interne Ziele
- Inter-AS- & Intra-AS-Einträge für externe Ziele





# Aufgaben des Inter-AS-Routing

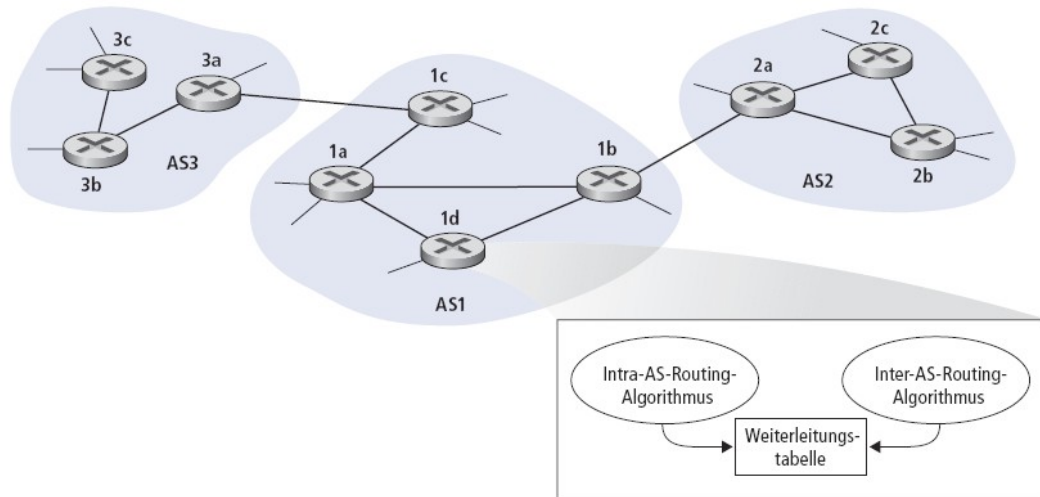
- Wenn ein Router in AS1 ein Paket für ein Ziel außerhalb von AS1 erhält:
- Router sollte das Paket zu einem der Gateway-Router in AS1 weiterleiten  
Aber zu welchem?





# Aufgaben des Inter-AS-Routing (2)

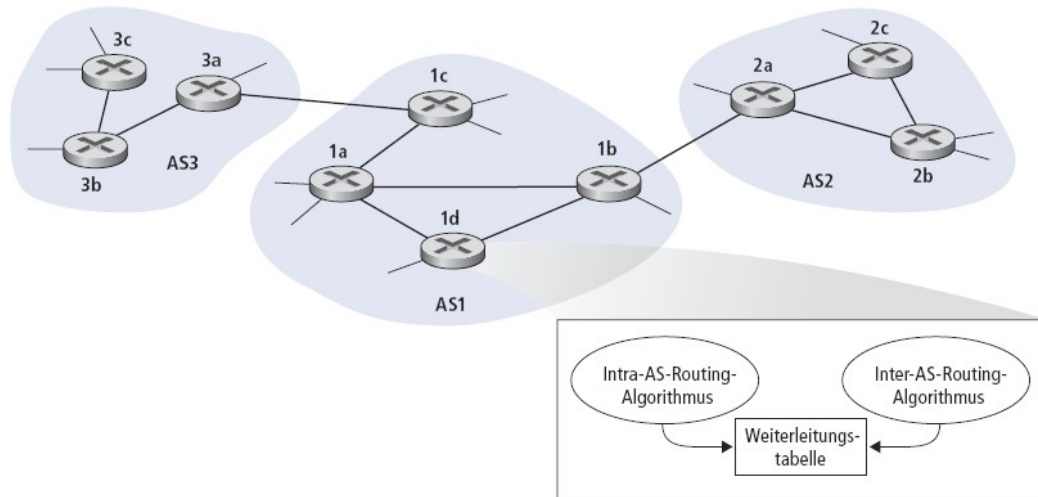
- Lernen, welche Ziele über die Autonomen Systeme AS2 und AS3 erreichbar sind
- Verteilen dieser Informationen an alle Router in AS1





# Aufgaben des Inter-AS-Routing (3)

- Netzwerk X ist sowohl über AS2 als auch über AS3 zu erreichen
- Router 1d muss sich entscheiden
- Hot Potato-Routing: Schicke das Paket einfach an den nächsten Gateway-Router





# Intra-AS-Routing

- **Die bekanntesten Intra-AS-Routing-Protokolle:**
  - RIP: Routing Information Protocol  
verwendet Distance-Vector
  - OSPF: Open Shortest Path First  
verwendet Link-State
  - Außerdem:
    - IGRP: Interior Gateway Routing Protocol  
→ CISCO proprietär



# RIP (Routing Information Protocol)



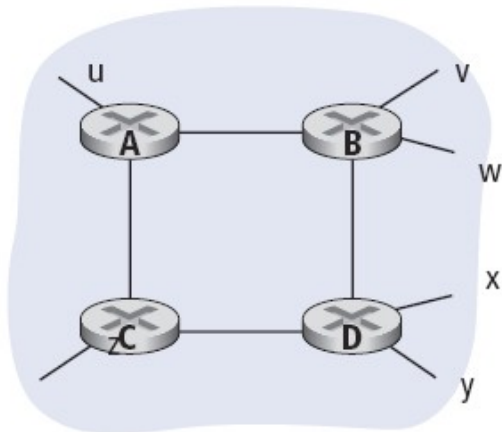
## ■ Distance-Vector-Algorithmus

### ■ RFC 1058: Version 1

- War bereits in der BSD-UNIX-Distribution von 1982 enthalten

### ■ RFC 2453: Version 2 (abwärtskompatibel)

## ■ Metrik: Anzahl der Hops ( $\leq 15$ Hops)



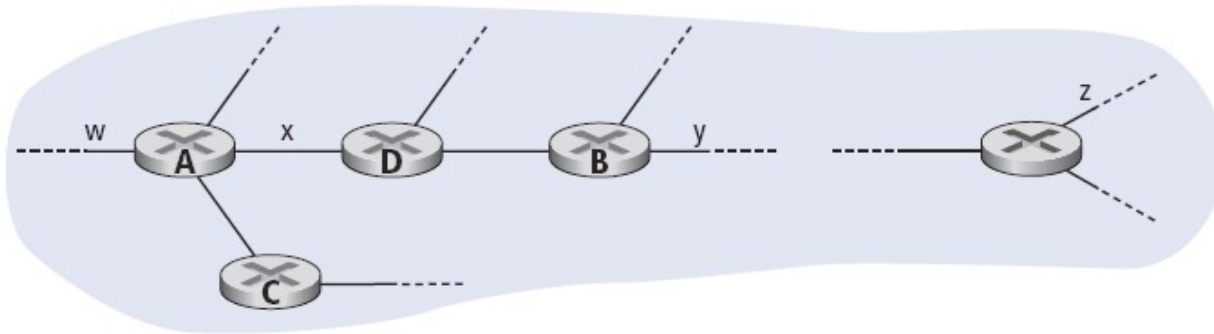
Ziel	Hops
u	1
v	2
w	2
x	3
y	3
z	2



# RIP-Advertisements

- **Distanzvektoren werden alle 30 Sekunden zwischen den Nachbarn per RIP-Advertisement ausgetauscht**
- **Jedes RIP-Advertisement enthält eine Liste**
  - bis zu 25 Ziernetzwerken
  - aus dem Inneren des Autonomen Systems

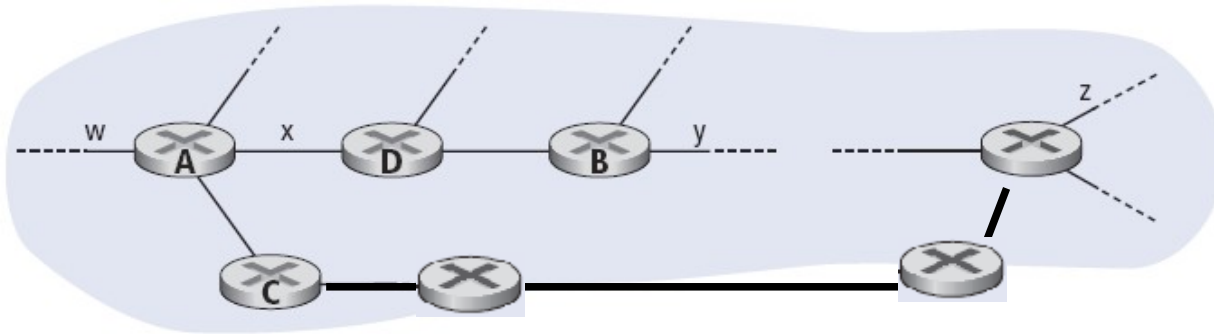
# RIP: Beispiel - Ausgangszustand



Routing-Tabelle in D:

Zielsubnetz	Nächster Router	Anzahl von Hops zur Zieladresse
w	A	2
y	B	2
z	B	7
x	—	1
...	...	...

# RIP: Beispiel - Advertisement von A



Advertisement an D:

Zielsubnetz	Nächster Router	Anzahl von Hops zur Zieladresse
z	C	4
w	–	1

Neue  
Routing-  
Tabelle in D:

Zielsubnetz	Nächster Router	Anzahl von Hops zur Zieladresse
w	A	2
y	B	2
z	A	5
...	...	...



**Router sind immer da,  
richtig?**





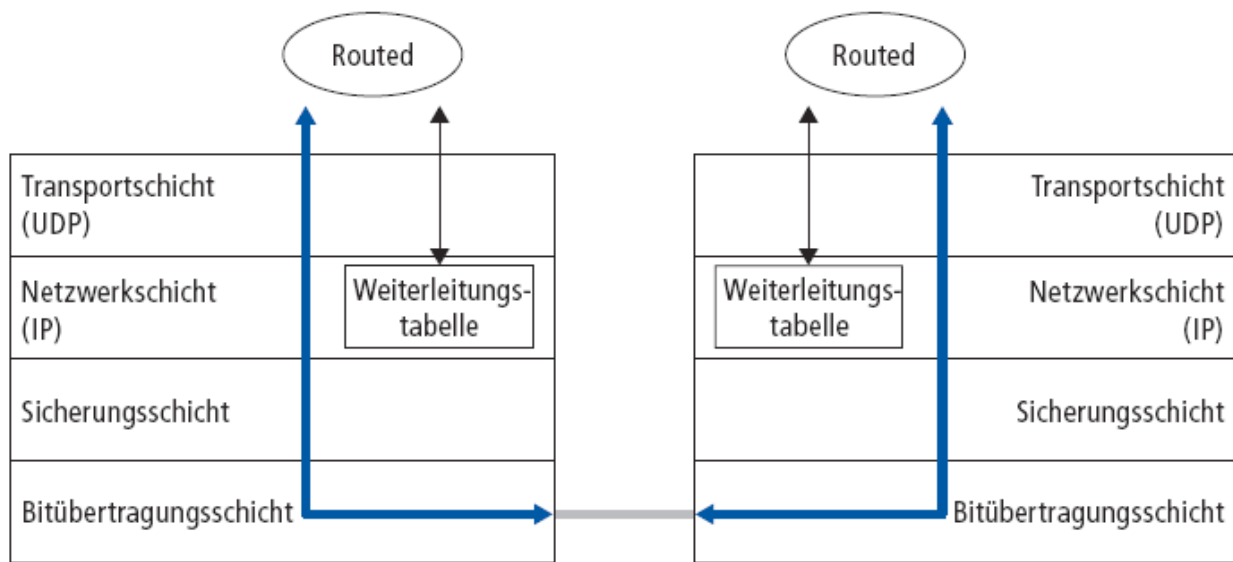
# RIP: Brechen von Links

- **Werden nach spätestens 180 Sekunden kein neues Advertisement mehr empfangen, wird der Nachbar gestrichen!**
  - Alle Routen über diesen Nachbarn werden ungültig
  - Neuberechnung des lokalen Distanzvektors
  - Verschicken des neuen Distanzvektors
  - Nachbarn bestimmen ihren Distanzvektor neu und verschicken ihn gegebenenfalls



# RIP-Architektur

- Die Routing-Tabelle wird in einem Prozess auf Anwendungsebene gepflegt, z.B.
  - routed (für „route daemon“)





# OSPF (Open Shortest Path First)

- **Link-State Routing / Dijkstras-Algorithmus**
  - RFC 2328: Version 2
  - “Open”: frei verfügbar
- **Periodisches Fluten von Link-State-Paketen**
  - Jeder Router kündigt seine Links an
    - Diese Ankündigungen werden geflutet
    - OSPF wird direkt in IP-Pakete eingepackt
  - Topologie ist jedem Router bekannt
    - Wird in einer sogenannten „Netzwerkkarte“ oder „Topology Map“ gespeichert





# Wo ist OSPF besser als RIP?

## ■ Sicherheit

- Nachrichten können authentifiziert werden

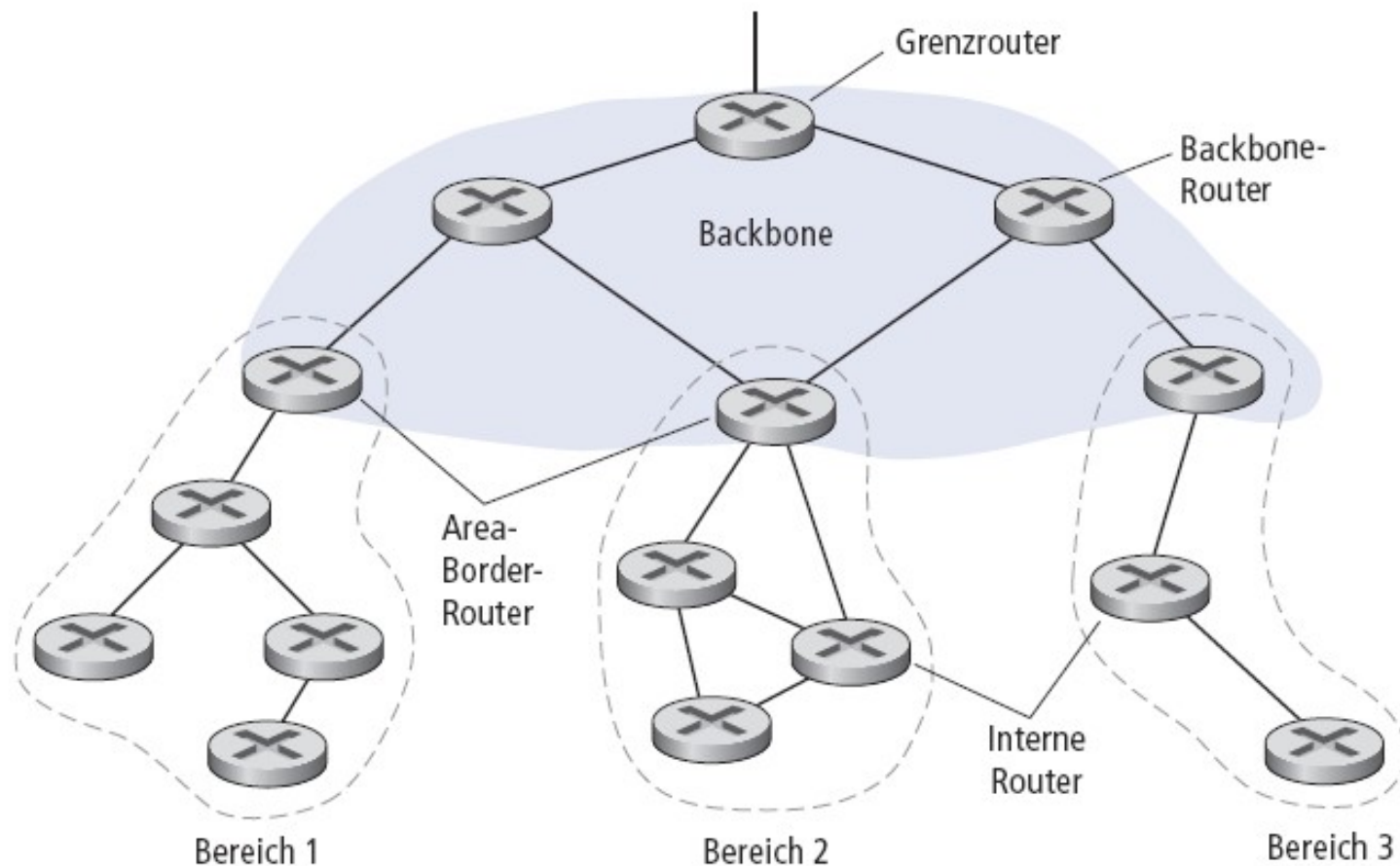
## ■ Mehrere alternative Pfade

- auf Basis verschiedener Metriken, pro Link können Gewichte angegeben werden
  - Kosten für einen Satellitenlink gering (pro Paket), aber Latenzzeit hoch

## ■ Hierarchisches OSPF in größeren autonomen Systemen



# Hierarchisches OSPF





# Hierarchisches OSPF

- **Zweistufige Hierarchie:  
Local Area versus Backbone**
  - Ein Backbone pro AS
  - Link-State-Advertisements werden nur in einer Local Area geflutet
  - Jeder Router in einer Local Area kennt deren detaillierte Topologie **und**
  - die Richtung zu den Netzwerken der anderen Local Areas



# Hierarchisches OSPF (2)

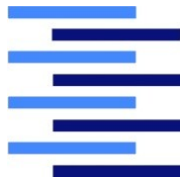
## ■ Area-Border-Router:

- Zusammenfassen der Distanzen zu den Netzwerken in der eigenen Local Area
- Ankündigen
  - der eigenen Zusammenfassung an die anderen Area-Border-Router
  - der Zusammenfassungen der anderen Area-Border-Router in der Local Area

## ■ Backbone-Router:

**führe OSPF-Routing im Backbone durch**

# Inter-AS-Routing beruht auf: BGP (Border Gateway Protocol)

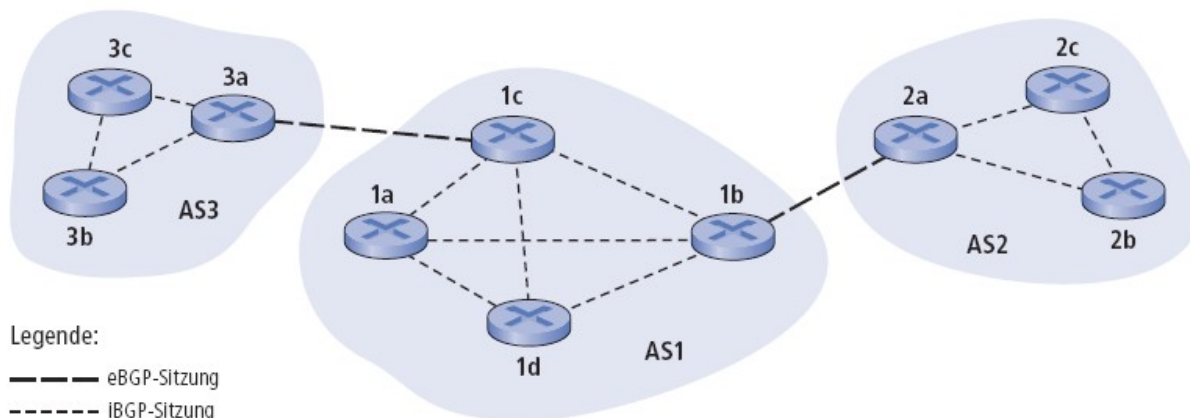


- **De-facto-Standard im Internet**
  - RFC 4271: Version 4
- **BGP erlaubt:**
  - Informationen über die Erreichbarkeit von Netzen von seinen benachbarten AS
  - Weiterleitung im eigenen AS
  - „Gute“ Routen zu bestimmten Zielnetzwerken festzulegen
- **Außerdem ermöglicht BGP es einem AS, seine eigene Erreichbarkeit anzukündigen**



# BGP -Grundlagen

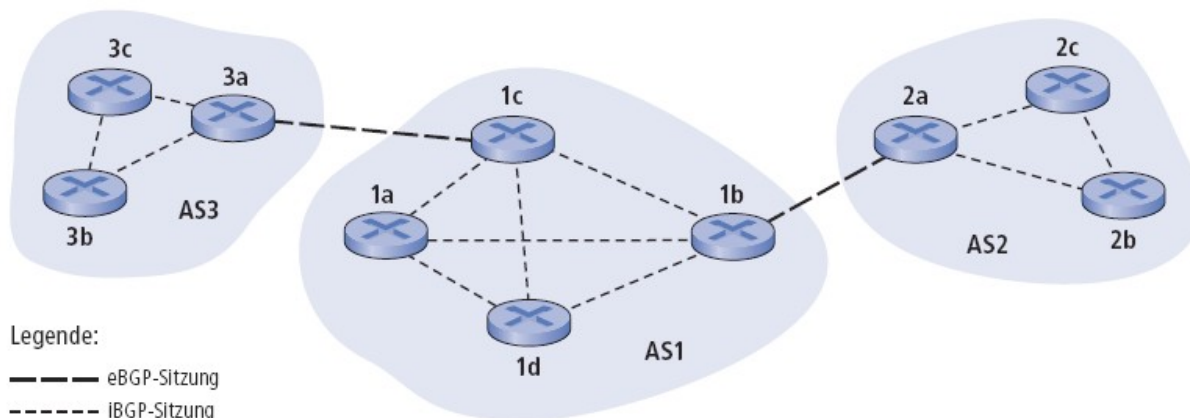
- **BGP-Peers tauschen Informationen über semi-permanente TCP-Verbindungen aus**
  - BGP-Sitzung (engl. BGP session)
  - Beide Router im selben AS : interne BGP-S.
  - in verschiedenen AS: externe BGP-Sitzung





# BGP -Grundlagen

- Wenn AS2 einen Präfix an AS1 meldet (z.B. 167.3/16), dann verspricht AS2, dass es alle Datagramme weiterleitet, deren Zieladressen zu diesem Präfix passen
- AS2 kann Präfixe in seinen Ankündigungen aggregieren





# Pfadattribute und BGP-Routen

- **Präfix-Ankündigungen beinhalten auch Pfadattribute (in BGP-Terminologie)**
  - Route = Präfix + Attribute
- **Zwei wichtige Attribute:**
  - AS-PATH: eine Liste aller AS, durch welche die Ankündigung weitergeleitet wurde
  - NEXT-HOP: Um Ankündigungen zuordnen zu können, schickt der Router seine IP-Adresse als NEXT-HOP-Attribut mit
- **Empfänger entscheiden über Annahme!**





# BGP-Routenwahl

- **Router können mehr als eine Route zum Ziel angeboten bekommen.**
  - Ein Router muss entscheiden, welche Route verwendet wird.
- **Regeln:**
  - Kürzester AS-PATH
  - Dichtester NEXT-HOP-Router:  
Hot Potato Routing
  - Weitere Kriterien, wohl der häufigste Fall –  
Routing ist eben Politik



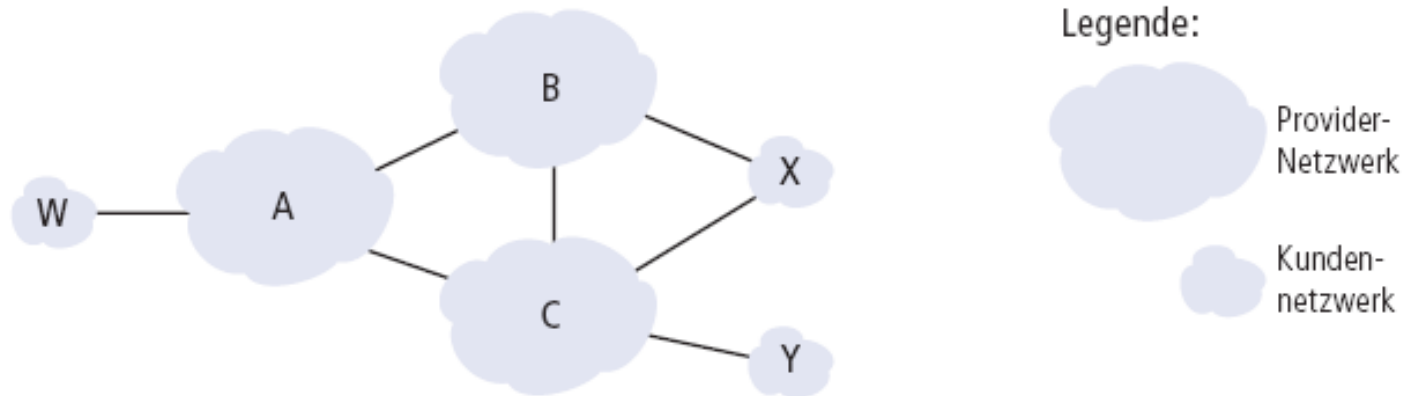
# BGP-Nachrichten

- **OPEN:** Öffnen einer BGP-Sitzung, Authentifizierung
- **UPDATE:** neue Route ankündigen oder Ankündigung zurückziehen
- **KEEPALIVE:** Sitzung aufrechterhalten, wenn keine UPDATES geschickt werden müssen (wird auch als ACK auf eine OPEN-Nachricht verschickt)
- **NOTIFICATION:** Mitteilen von Fehlern, Schließen einer Session



# BGP-Routing-Politiken

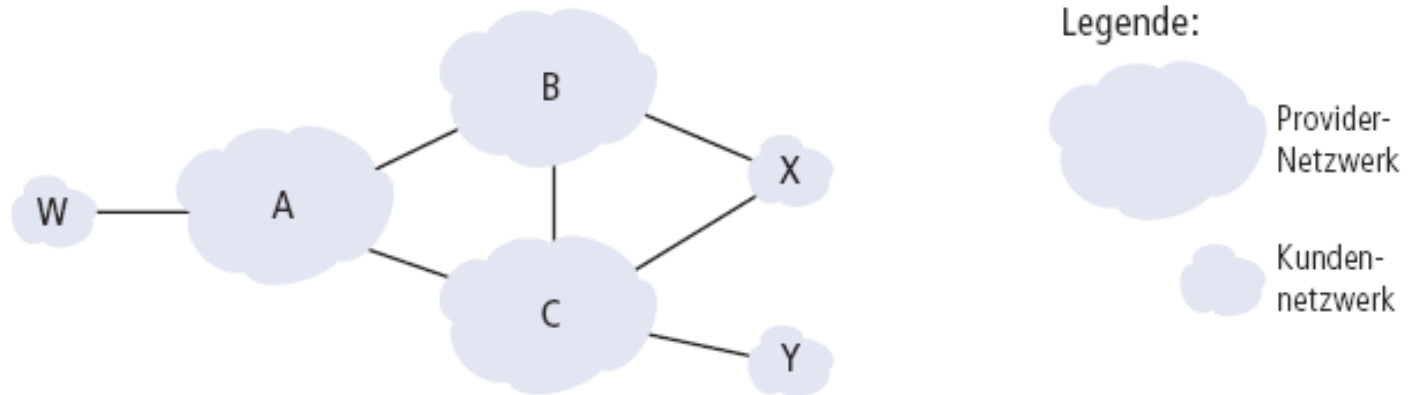
- $A \rightarrow B$  : Pfad AW und  $B \rightarrow X$  : Pfad BAW
- Soll B den Pfad BAW auch C ankündigen?





## BGP-Routing-Politiken (2)

- **A → B : Pfad AW und B → X : Pfad BAW**
- **Soll B den Pfad BAW auch C ankündigen?**
  - Nein! B hätte nichts davon, da weder W noch C Kunden von B sind





## BGP-Routing-Politiken (3)

- **A → B : Pfad AW und B → X : Pfad BAW**
- **Soll B den Pfad BAW auch C ankündigen?**
  - Nein! B hätte nichts davon, da weder W noch C Kunden von B sind
  - B möchte, dass C Datenpakete zu W über A leitet
  - B möchte nur Verkehr an oder von seinen Kunden weiterleiten

# Unterschiede bei Intra-AS- und Inter-AS-Routing



## ■ Politiken:

### ■ Inter-AS:

Eine Organisation möchte kontrollieren, wie (und ob) der Verkehr anderer Organisationen durch das eigene Netzwerk geleitet wird

### ■ Intra-AS:

eigener Verkehr, eigene Administration, hier sind keine Politiken nötig

# Unterschiede bei Intra-AS- und Inter-AS-Routing (2)



## ■ Größenordnung:

### ■ Inter-AS:

Hierarchisches Routing reduziert die Größe der Routing-Tabellen und reduziert den Netzwerkverkehr für Routing-Updates

### ■ Intra-AS:

Größe der Tabellen kein Problem

# Unterschiede bei Intra-AS- und Inter-AS-Routing (3)



## ■ Performance:

### ■ Inter-AS:

Politiken können wichtiger sein als Performanz

### ■ Intra-AS:

kann sich auf Performanz konzentrieren





# **Neues Routingparadigma auf Basis virtueller Kanäle**



# Dienstmodell der Netzwerkschicht

Wie darf man sich den Kanal vorstellen?

- garantierte Bandbreite?
- Erhaltung des zeitlichen Abstandes?
- Verlustfreiheit?
- Gleiche Reihenfolge?
- Stau-Rückmeldung an den Sender?

? ?  
Virtueller Kanal  
oder  
Datagramm?  
?



# Internet im Vergleich

Netzwerk- architektur	Dienst- modell	Band- breiten- garantien	Garantie der Ver- lustfreiheit	Reihen- folge	Zeit- garantien	Hinweis auf Überlast
Internet	Best Effort	keine	nein	beliebige möglich	nicht unterstützt	keiner
ATM	CBR	garantiert eine kon- stante Rate	ja	in korrekter Reihenfolge	unterstützt	Überlast tritt nicht auf
ATM	ABR	garantier- tes Mini- mum	nein	in korrekter Reihenfolge	nicht unterstützt	Überlasthin- weise werden verwendet



# **Datagram: Klassisches Dienstmodell**

- **Kein Verbindungsaufbau auf der Netzwerkebene**
- **Router kennen keinen “Zustand” über die Ende-zu-Ende-Verbindungen**
  - Pakete werden typischerweise allein anhand der Zieladresse geroutet
  - Pakete zwischen den gleichen Rechnern können unterschiedliche Wege nehmen
- **Beispiele:**
  - IPv4
  - IPv6 (teilweise)

# **Virtueller Kanal: Neues Dienstmodell**



**Es werden weiter Pakete über das Netz  
gesendet, aber der Pfad vom Sender zum  
Empfänger verhält sich wie eine klassische  
Telefonleitung**

- In Bezug auf die Performanz
- und auf die Netzwerkaktionen auf dem Pfad  
vom Sender zum Empfänger

**Deswegen verbindungsorientiert**

- mit Verbindungsaufbau für jede Verbindung  
vor dem Transport der Daten
- und Verbindungsabbau

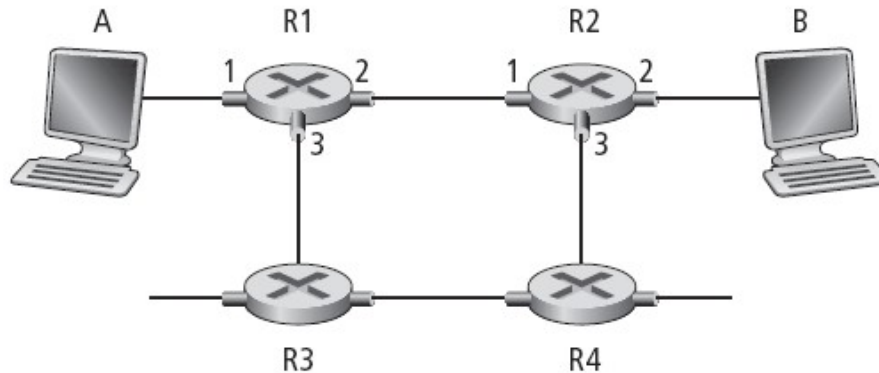
# Virtueller Kanal: Neues Dienstmodell (2)



**Wie gelingt dies?**

- **Jedes Paket trägt den Bezeichner eines virtuellen Kanals (VC) – und nicht die Adresse des Zielrechners**
- **Jeder Router auf dem „geschalteten“ Pfad speichert nunmehr einen “Zustand” für jede durch ihn laufende Verbindung**
  - **Erinnerung!**  
Transportschicht-Verbindungen sind nur in den Endgeräten existent

# Und so sieht das aus:



## Weiterleitungstabelle in R1:

Eingehende Schnittstelle	Eingehende VC-Nummer	Ausgehende Schnittstelle	Ausgehende VC-Nummer
1	12	2	22
2	63	1	18
3	7	2	17
1	97	3	87
...	...	...	...

# Virtueller Kanal: Neues Dienstmodell (3)



## Wie gelingt dies?

- **Ressourcen der Verbindung (Übertragungskapazität, Puffer) können für einen virtuellen Kanal reserviert werden**
  - Damit wird es möglich, ein Verhalten zu gewährleisten, das dem einer festen Leitung entspricht
- **Beispiele:**
  - IPv6 (teilweise)
  - ATM
  - MPLS (zwischen Schicht 2 und 3)





# Vergleich Datagramm / VC

	Datagramm-Netzwerk	VC-Netzwerk
<i>Verbindungsaufbau</i>	Nicht erforderlich	Erforderlich
<i>Adressierung</i>	Jedes Paket enthält die volle Quell- und Zieladresse	Jedes Paket enthält eine kurze VC-Nummer
<i>Zustandsinformation</i>	Router führen keine Zustands-informationen	Jede virtuelle Verbindung hat einen Tabelleneintrag
<i>Routing</i>	Jedes Paket wird unabhängig befördert	alle Pakete folgen der anfänglich gewählten Route
<i>Wirkung von Routerfehlern</i>	Nur Verlust einzelner Pakete	Alle virtuellen Verbindungen über den ausgefallenen Router werden beendet
<i>Dienstgüte-Garantie und Überlastkontrolle</i>	Schwierig	Einfach, wenn ausreichende Ressourcen reserviert sind
<i>Flexibilität</i>	sehr hoch	gering (hoher Verwaltungs- und Abstimmungsaufwand)



# Kontakt

**Prof. Dr. Klaus-Peter Kossakowski**

**Email: klaus-peter.kossakowski  
@haw-hamburg.de**

**Mobil: +49 171 5767010**

**<https://users.informatik.haw-hamburg.de/~kpk/>**