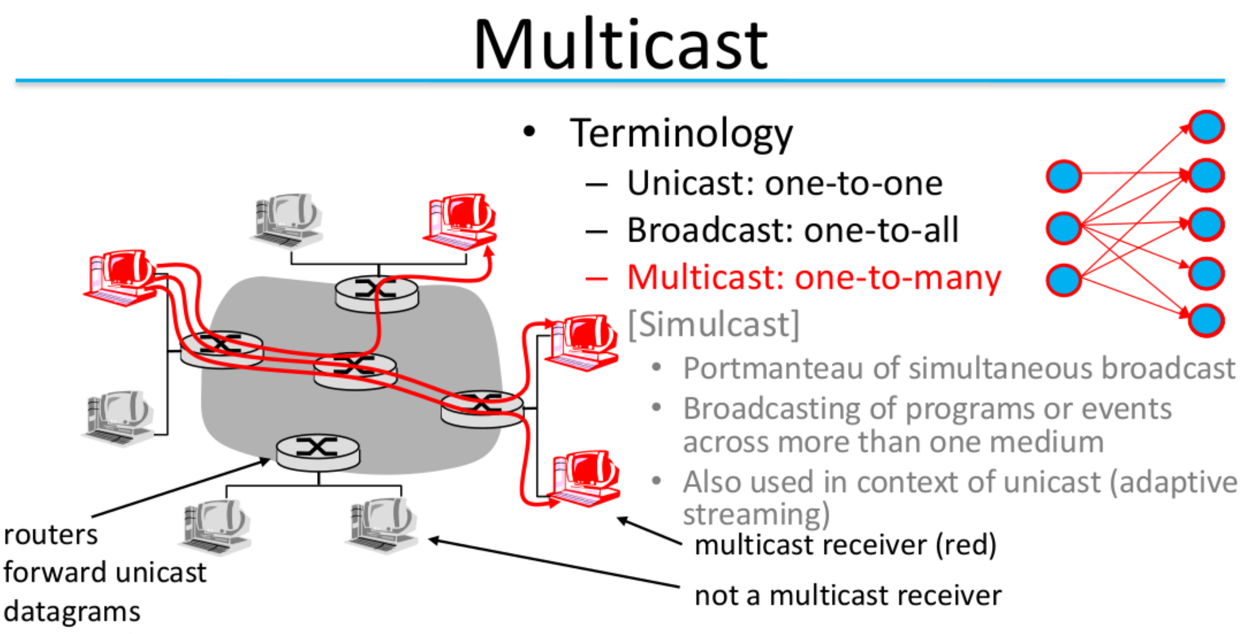
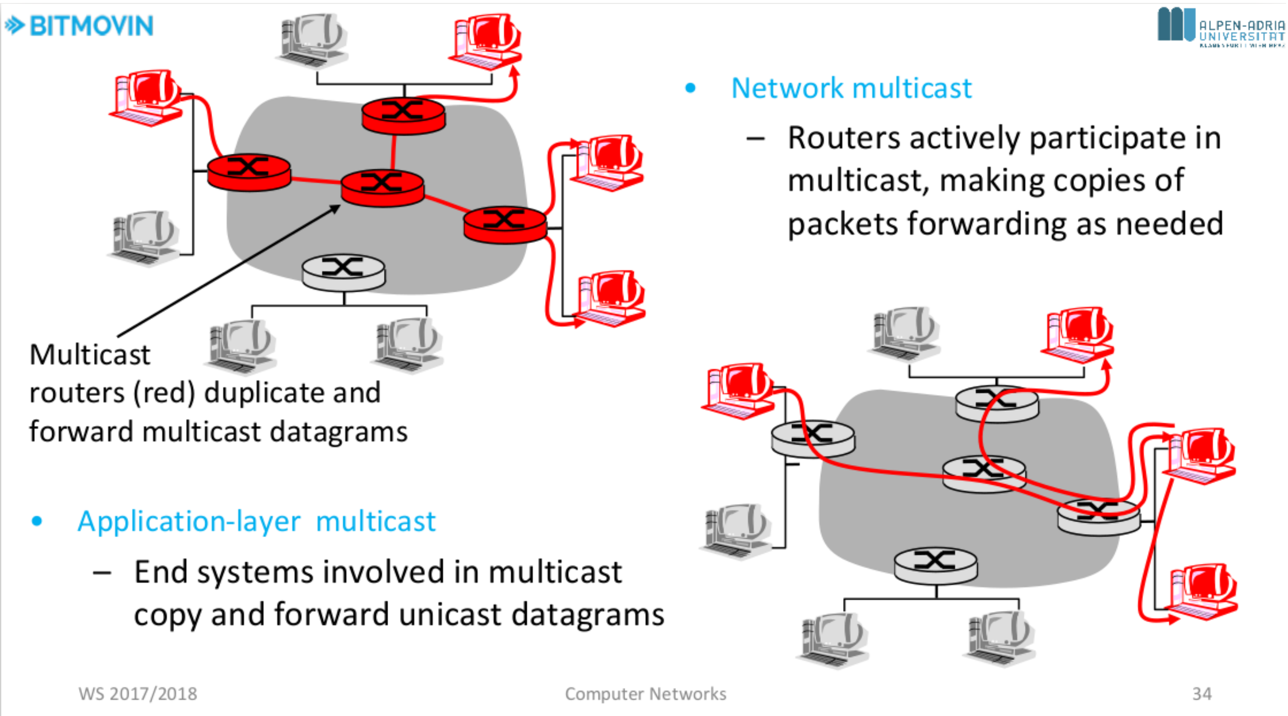


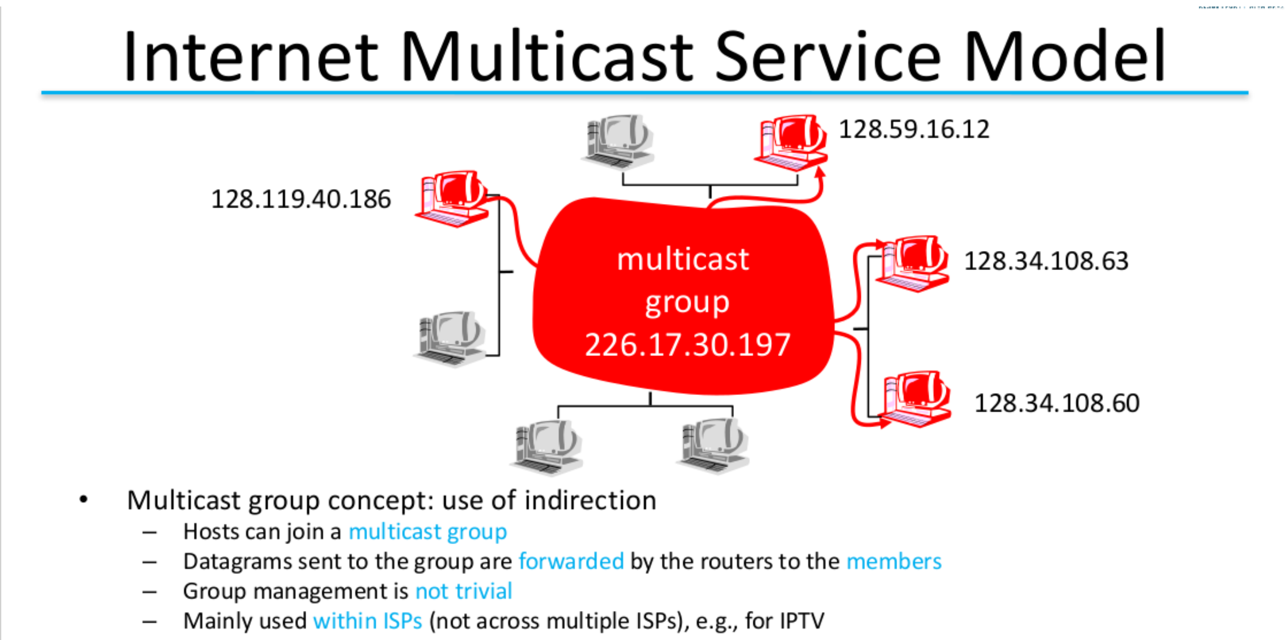
Any-Cast;Server→Klienten: z.B.: Klienten melden sich zu Stream an.

Arten von Multicast:



Network-Multicast: Auf IP-Schicht :: Im größeren Kontext nicht definiert.

Es gibt vordefinierte Multicast-Adressen:



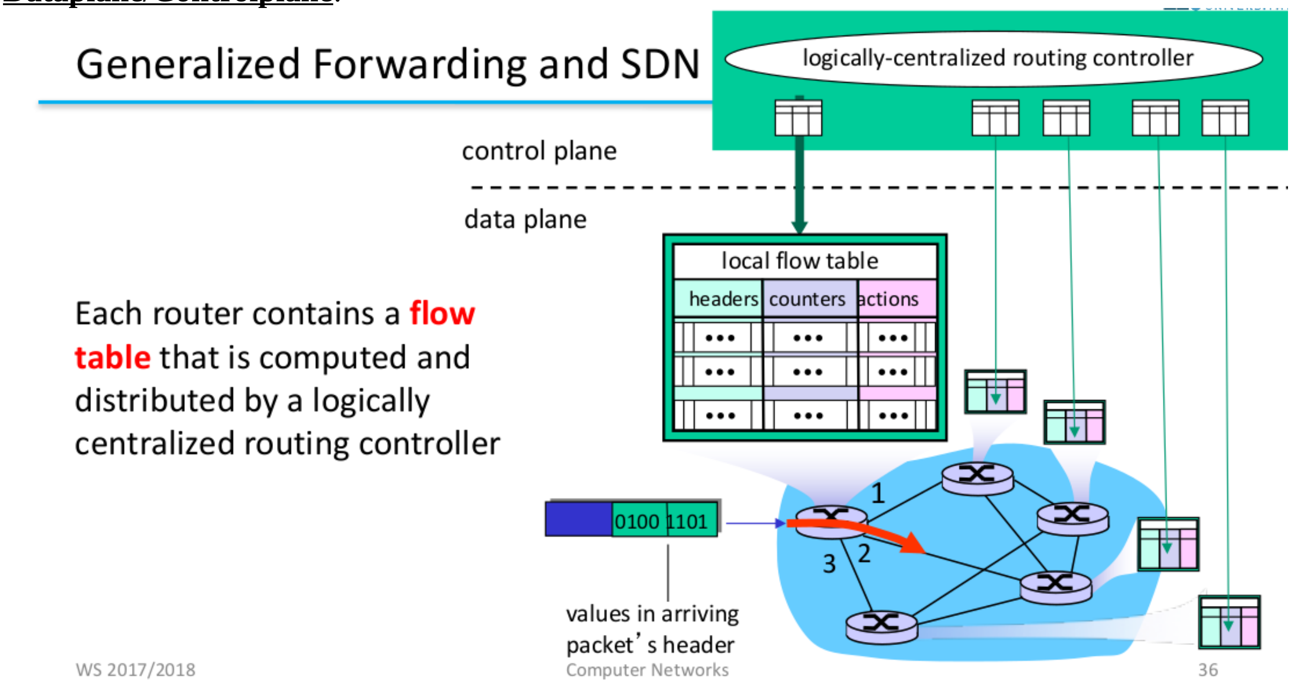
Obrige Klients haben sich zu der Mutlicast-Gruppe angemeldet & Router weiß wohins gerouted wird.

Classful IP Addressing:

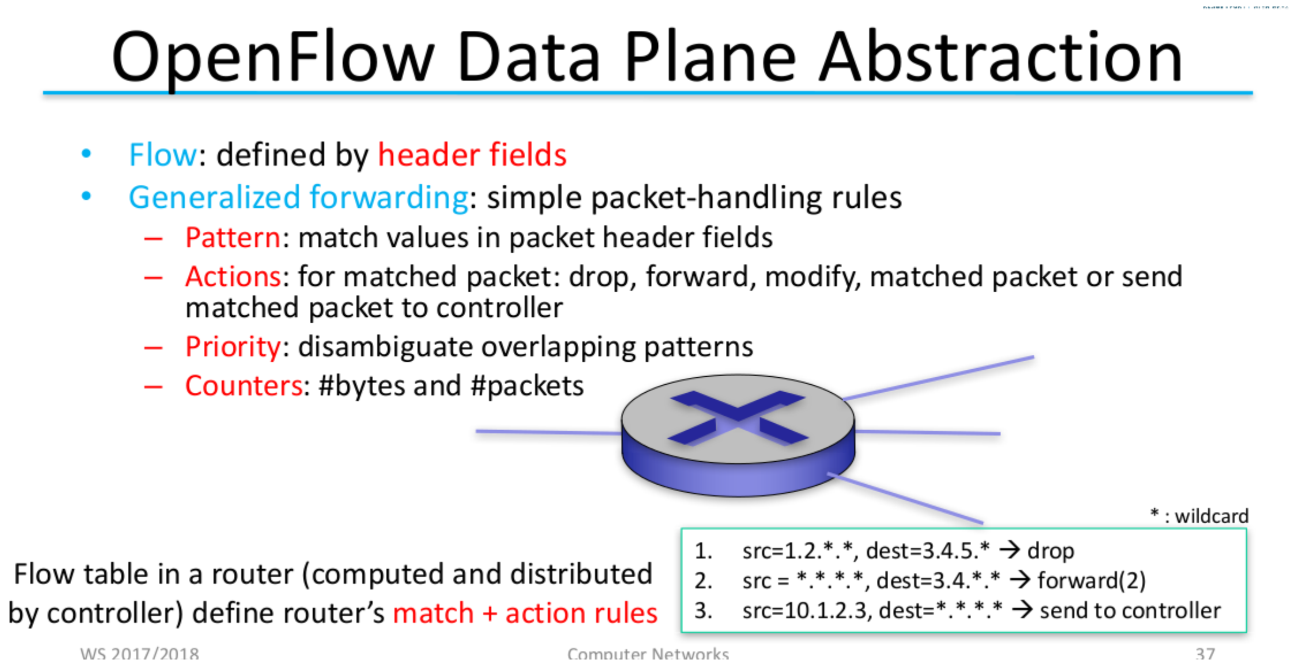
D :: |1110| Multicast Adress | ↔ 224.0.0.0 to 239.255.255.255

z.B.: Asfinag, Autobahnkameraüberwachung ist multicasted via Session Encrypted Protocol :: Kaum Sicherheit

<u>Dataplane/Controlplane</u>:

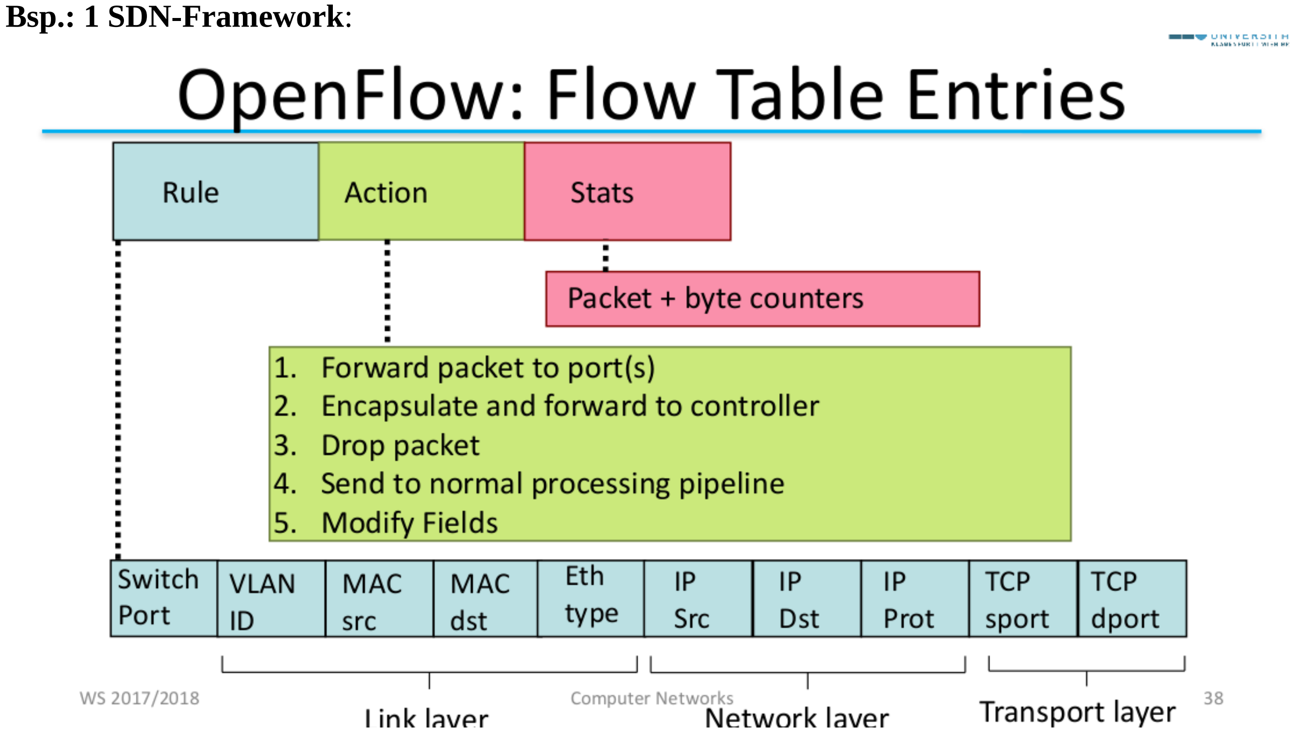


Local Flow Table :: Forwarding-Tabelle predefinierter Actions @Router



Flow: Definiert wohin was geht
Gen. Forwarding: Predefiniert im Router
- Pattern matches Header
- Actions: .. taken
- Priority: Mehrere pattern matchen auf 1 Action: Prioritätsverweisung
- Counters: #Bytes #Packets
'Match + Action Rules' – Table above :: beliebig Programmierbar

Bsp.: 1 SDN-Framework:



Frei

Destination-based forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	51.6.0.8	*	*	*	port6

IP datagrams destined to IP address 51.6.0.8 should be forwarded to router output port 6

Firewall:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Forward
*	*	*	*	*	*	*	*	*	22	drop

do not forward (block) all datagrams destined to TCP port 22

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Forward
*	*	*	*	*	128.119.1.1	*	*	*	*	drop

do not forward (block) all datagrams sent by host 128.119.1.1

Examples

Destination-based layer 2 (switch) forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	22:A7:23:11:E1:02	*	*	*	*	*	*	*	*	port3

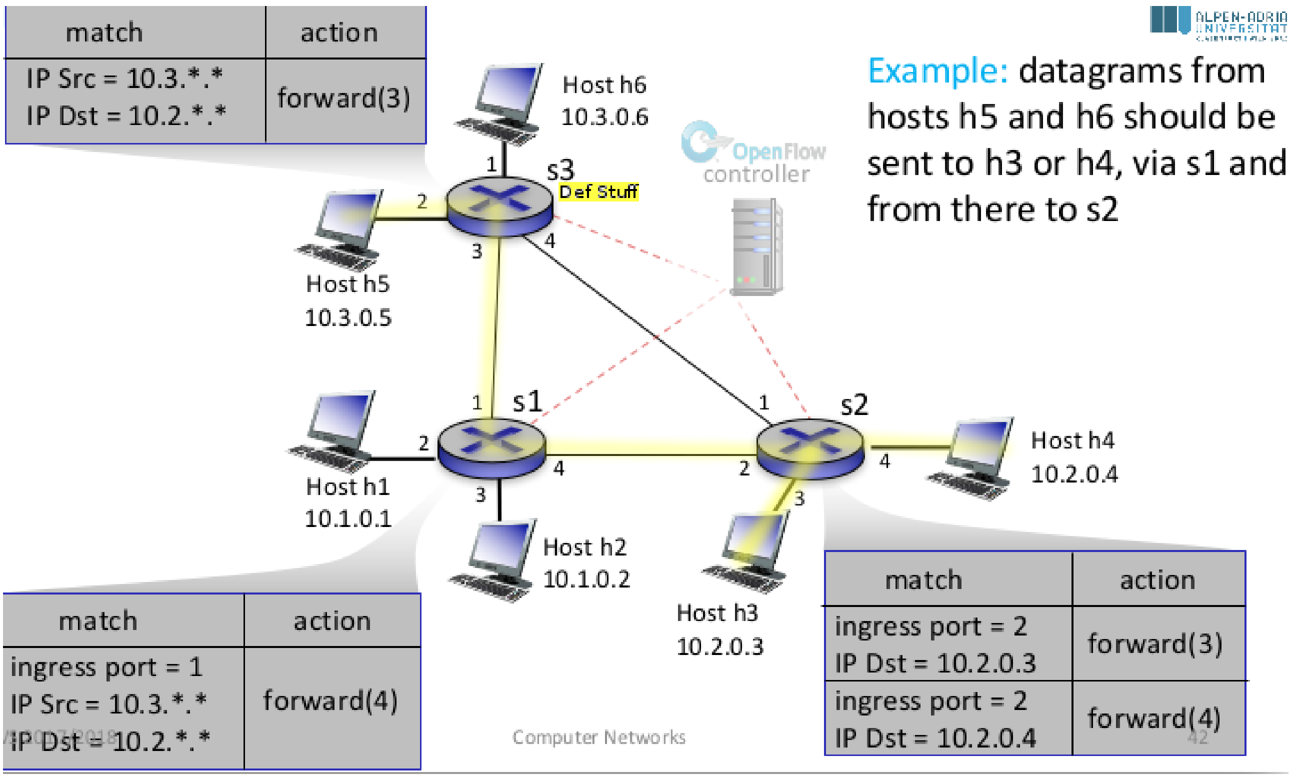
layer 2 frames from MAC address 22:A7:23:11:E1:02 should be forwarded to output port 3

definierbar in der OpenFlow Abstraction:

OpenFlow Abstraction

Match + action: unifies different kinds of devices

- Router
  - Match: longest destination IP prefix
  - Action: forward to a link
- Switch
  - Match: destination MAC address
  - Action: forward or flood
- Firewall
  - Match: IP addresses and TCP/UDP port numbers
  - Action: permit or deny
- NAT
  - Match: IP address and port
  - Action: rewrite address and port



<u>Network Layer – Control Plane</u>

Network Layer Functions

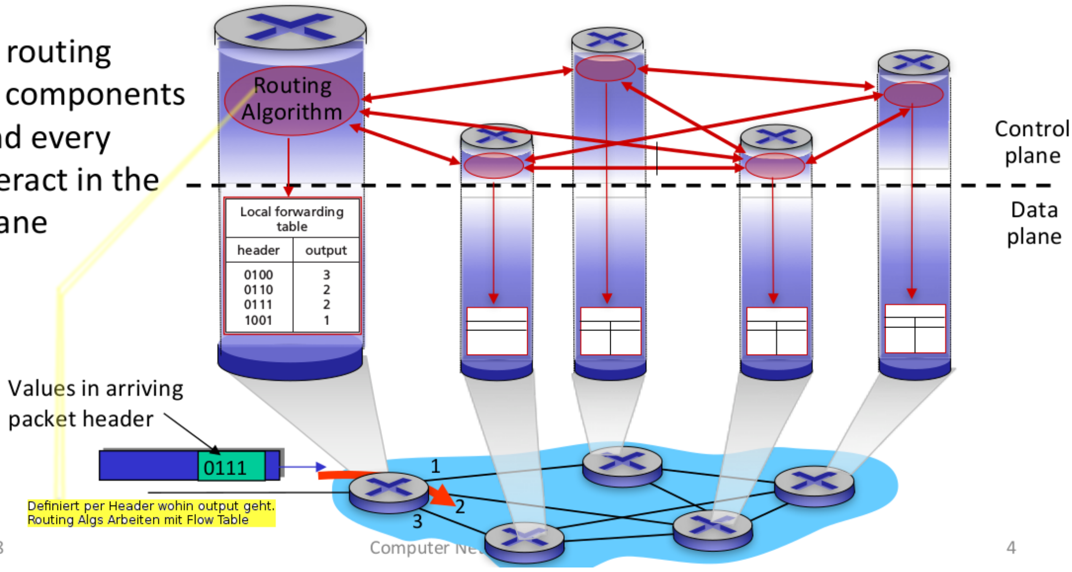
Recall: two network-layer functions

- Forwarding: move packets from router's input to appropriate router output — **data plane**
- Routing: determine route taken by packets from source to destination — **control plane**
- Two approaches to structuring network control plane:
  - Per-router control (traditional)
  - Logically centralized control (software defined networking)

Forwarding: Packet kommt → Wohin gehts?Routing: Packet: Definiere Wohin. (Flow table)

Per-Router Control Plane

Individual routing algorithm components in each and every router interact in the control plane

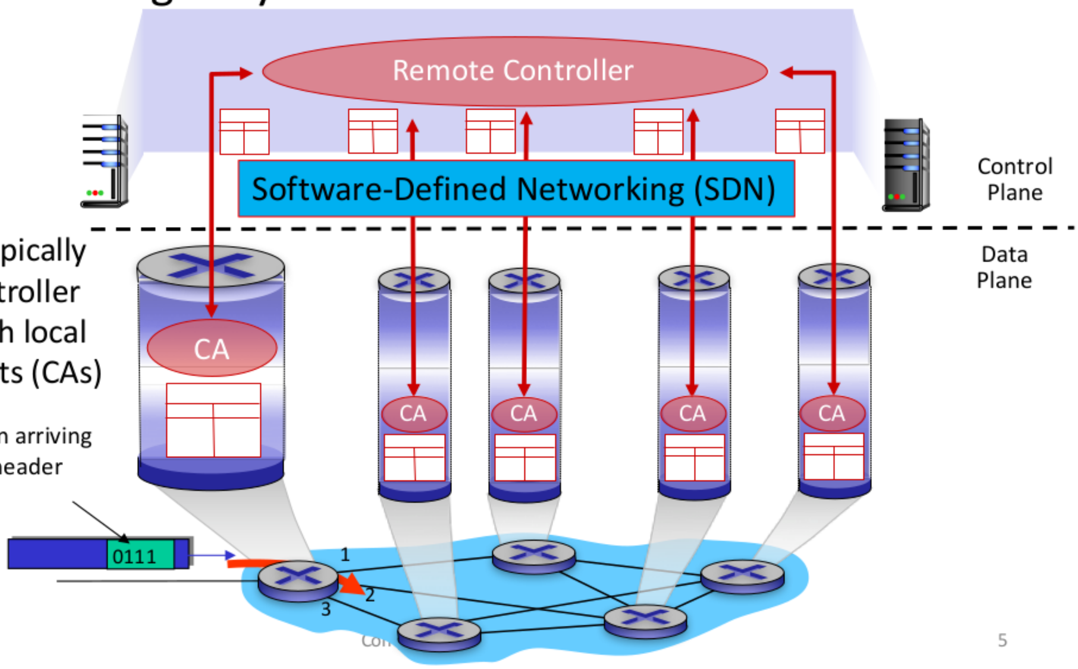


Definiert per Header wohin output geht.
Routing Algs Arbeiten mit Flow Table

BITMOVIN

Logically Centralized Control Plane

A distinct (typically remote) controller interacts with local control agents (CAs)



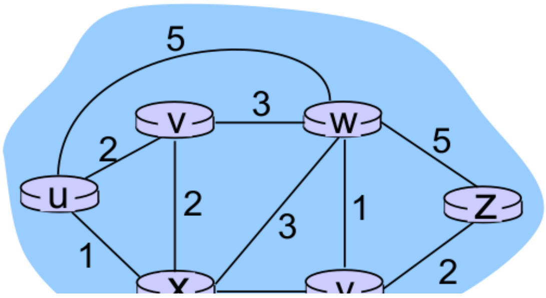
→ Data Flow via Control Agents definiert via Remote Controller.
→ Kann physisch verteilt sein
→ Einfach implementierbar.

Routing Protocols

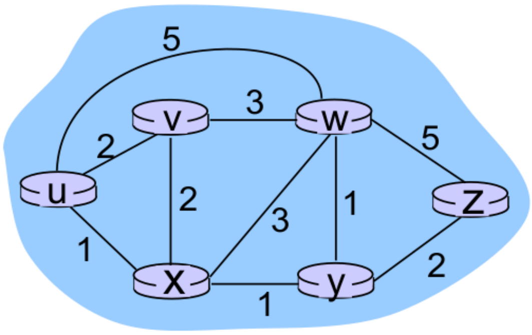
Routing protocol goal: determine "good" paths (equivalently, routes), from sending hosts to receiving host, through network of routers

- **Path**: sequence of routers packets will traverse in going from given initial source host to given final destination host
- **"Good"**: least "cost", "fastest", "least congested"
- Routing: a "top-10" networking challenge!

Graph Abstraction of the Network



- graph: G = (N,E)



c(x,x') = cost of link (x,x')
e.g., c(w,z) = 5

Cost could always be 1, or inversely related to bandwidth, or inversely related to congestion

Cost of path $(x_1, x_2, x_3,\dots, x_p) = c(x_1,x_2) + c(x_2,x_3) + \dots + c(x_{p-1},x_p)$

Key question: what is the least-cost path between u and z?
Routing algorithm: algorithm that finds that least cost path

Routing Algorithm Classification

Q: global or decentralized info?

Global:
- All routers have complete topology, link cost info
- "Link state" algorithms

Decentralized:
- Router knows physically-connected neighbors, link costs to neighbors
- Iterative process of computation, exchange of info with neighbors
- "Distance vector" algorithms

Local:
- Router knows only itself
- Hot potato, flooding, …

Q: static or dynamic?

Static:
- Routes change slowly over time

Dynamic:
- Routes change more quickly
  - Periodic update
  - In response to link cost changes

Wie: Protokollimplementierung
→ Decentral: Distance Vector Algorithm (Know your neighbour); Link State Algorithm(Dijkstra)
→ Lokal: Hot potatoe, Flooding (Lenkt an alle weiter == Broadcasting im THCP)
→ Static: Linkweiterleitungen ändern sich nicht
→ Dynamisch: Linkweiterleitungen ändern sich oft ↔ Kostenneuberechnung

A Link-State Routing Algorithm

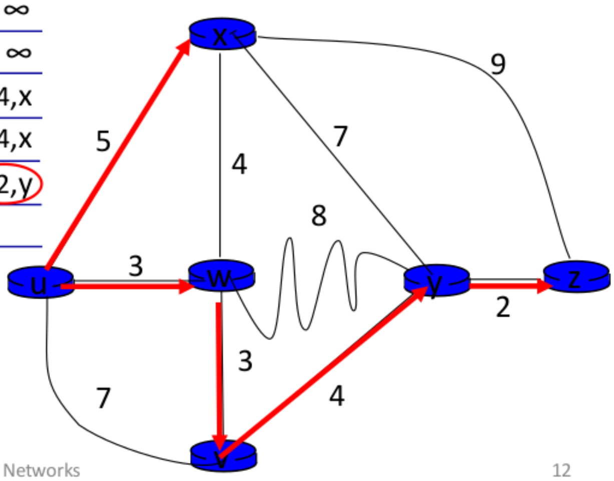
Dijkstra's algorithm

- Network topology, link costs known to all nodes
  - Accomplished via "link state broadcast"
  - All nodes have same info
- Computes least cost paths from one node (source) to all other nodes
  - Gives forwarding table for that node
- Iterative
  - After k iterations, know least cost path to k destinations

Alle Knoten kennen ihre Flows.

Dijkstra's Algorithm: Example

Step	N'	D(v) p(v)	D(w) p(w)	D(x) p(x)	D(y) p(y)	D(z) p(z)
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3	uwxv				10,v	14,x
4	uwxvy					12,y
5	uwxvyz					



Notes:

Construct shortest path tree by tracing predecessor nodes

Ties can exist (can be broken arbitrarily)

Bei Gleichheit : Wurscht, beide Varianten möglich; Dennoch definiert Router:
- Gibts zusätzliche Kriterien: #Hops, Zufall, Alphabetisch, ...

Dijkstra’s Algorithm

Notation

c(x,y): link cost from node x to y; = ∞ if not direct neighbors

D(v): current value of cost of path from source to destination v

p(v): predecessor node along path from source to v

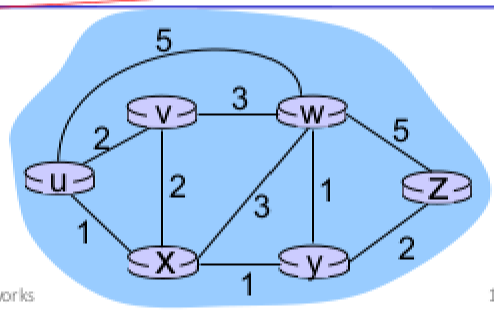
N': set of nodes whose least cost path definitively known

```
Initialization:
  N' = {u}
  for all nodes v
    if v adjacent to u
        then D(v) = c(u,v)
    else D(v) = ∞

Loop
  find w not in N' such that D(w) is a minimum
  add w to N'
  update D(v) for all v adjacent to w and not in N' :
      D(v) = min( D(v), D(w) + c(w,v) )
  /* new cost to v is either old cost to v or known
   shortest path cost to w plus cost from w to v */
until all nodes in N'
```

Dijkstra's Algorithm: Another Example

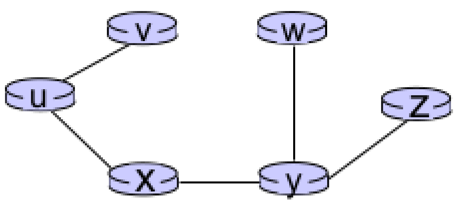
Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

Dijkstra's Algorithm: Example (2)

Resulting shortest-path tree from u:



Resulting forwarding table in u:

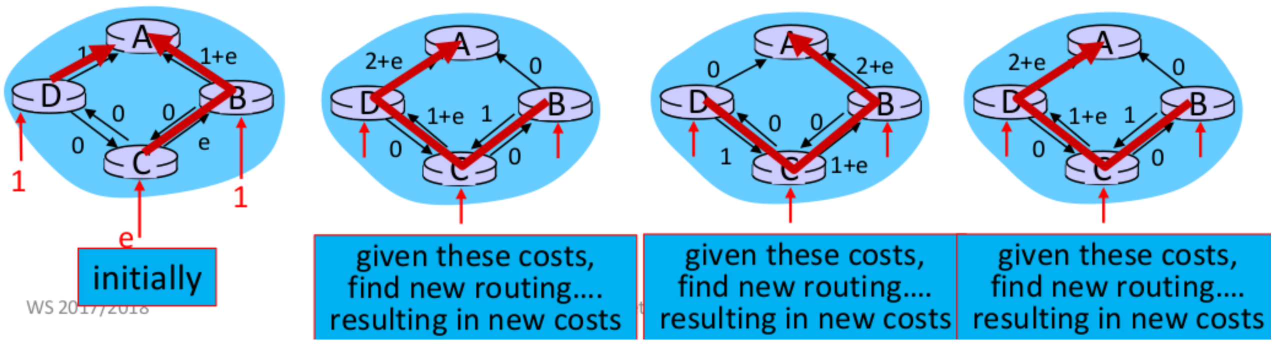
destination	link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
z	(u,x)

→ Mögl. Wie man an Forwarding-Table kommt.

<u>Komplexität</u>:

Dijkstra's Algorithm: Discussion

- Algorithm complexity: n nodes
  - Each iteration: need to check all nodes, w, not in N
  - n(n+1)/2 comparisons: $O(n^2)$
  - More efficient implementations possible: O(nlogn)
- Oscillations possible:
  - E.g., support link cost equals amount of carried traffic:



Oszillationen möglich :: Wenn Kostenmenge = Datenmenge:: 1 für D, e für C, 1 für B
→ Loops. Möglichkeit: Keine Abhängigkeit von Datenvolumen ↔ Widerspruch zur Annahme
→ Dezentraler Algorithmusverlaufs – Jeder Router für sich.

Distance Vector Algorithm

Bellman-Ford equation (dynamic programming)

let

$d_x(y)$:= cost of least-cost path from x to y

then

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

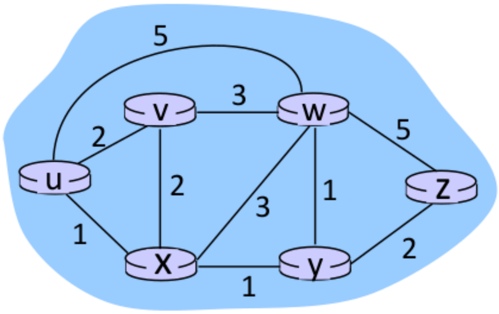
cost from neighbor v to destination y

cost to neighbor v

min taken over all neighbors v of x

→ "Know your neighbour" - Bellman-Ford equation
→ Suche Minimale Kosten zu allen Nachbarn + Kosten vom Nachbarn zur Dest.

Distance Vector Algorithm: Example



Clearly, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

Bellman-Ford equation says:

$$d_u(z) = \min \{ c(u,v) + d_v(z), \; c(u,x) + d_x(z), \; c(u,w) + d_w(z) \}$$
$$= \min \{2 + 5, \; 1 + 3, \; 5 + 3\} = 4$$

Node achieving minimum is next hop in shortest path, used in forwarding table

d_v(z) = 2+1+2
d_x(z) = 1+2
d_w(z) = 1+2
c(u,v) = 2
c(u, x) = 1

Distance Vector Algorithm

- $D_x(y)$ = estimate of least cost from x to y
  - x maintains distance vector $\mathbf{D}_x = [D_x(y): y \in N]$
- Node x:
  - Knows cost to each neighbor v: $c(x,v)$
  - Maintains its neighbors' distance vectors. For each neighbor v, x maintains $\mathbf{D}_v = [D_v(y): y \in N]$

Key idea:

- From time-to-time, each node sends its own distance vector estimate to neighbors
- When x receives new DV estimate from neighbor, it updates its own DV using B-F equation:

$$D_x(y) \leftarrow \min_v\{c(x,v) + D_v(y)\} \text{ for each node } y \in N$$

- Under minor, natural conditions, the estimate $D_x(y)$ converge to the actual least cost $d_x(y)$

… Maintain routes logical costs. Update periodically

Iterative, asynchronous: each local iteration caused by:

- Local **link cost change**
- DV **update message** from neighbor

Distributed:

- Each node **notifies neighbors only** when its DV changes
  - Neighbors then notify their neighbors if necessary

- **Each node**

  Wait for (change in local link cost or msg from neighbor)

  ↓

  Recompute estimates

  ↓

  If DV to any destination has changed, ***notify*** neighbors

→ Bei Änderung innerhalb eines Knotens, so benachrichtige alle Nachbarn → Welche wiederum Nachbarn benachrichtigen.

$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\}$
$= \min\{2+0 , 7+1\} = 2$

$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\}$
$= \min\{2+1 , 7+0\} = 3$

node x table

from \ cost to	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

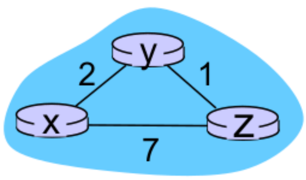
node y table

from \ cost to	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

node z table

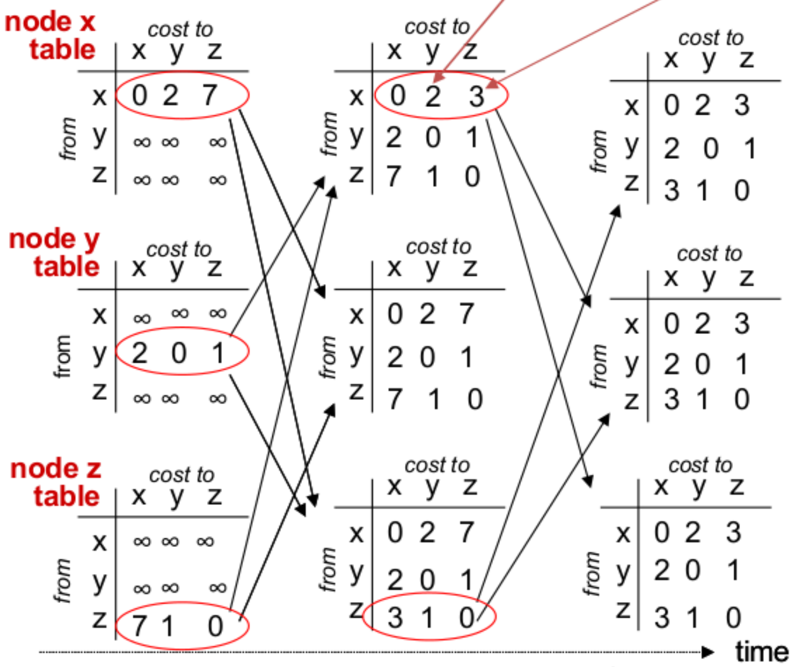
from \ cost to	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

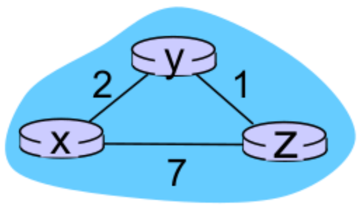
time



$$D_x(y) = \min\{c(x,y) + D_y(y),\ c(x,z) + D_z(y)\} = \min\{2+0\ ,\ 7+1\} = 2$$

$$D_x(z) = \min\{c(x,y) + D_y(z),\ c(x,z) + D_z(z)\} = \min\{2+1\ ,\ 7+0\} = 3$$

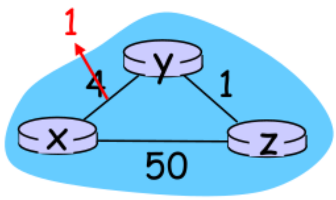




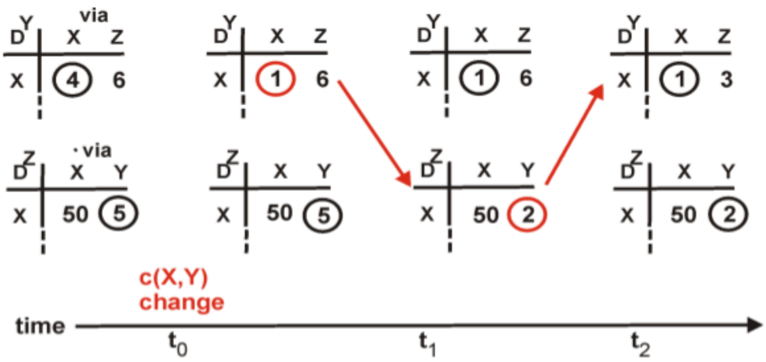
Distance Vector: Link Cost Changes

Link cost changes

- Node detects local link cost change
- Updates routing info, recalculates distance vector
- If DV changes, notify neighbors

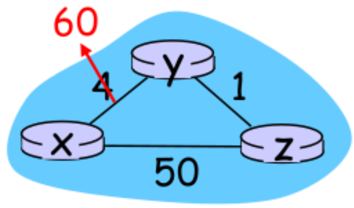


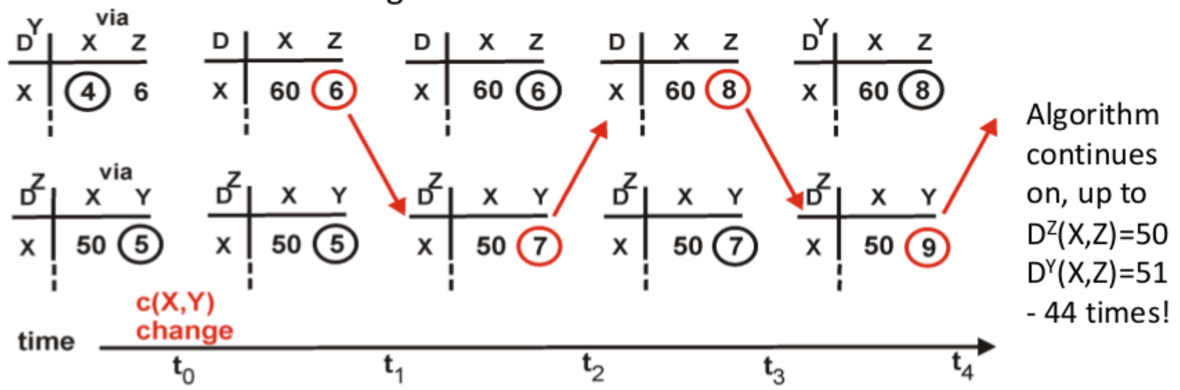
"Good news travel fast"



Link cost changes

- Node detects local link cost change
- Bad news travels slow – "count to infinity" problem!
- 44 iterations before algorithm stabilizes

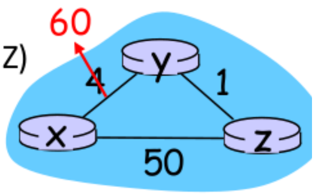


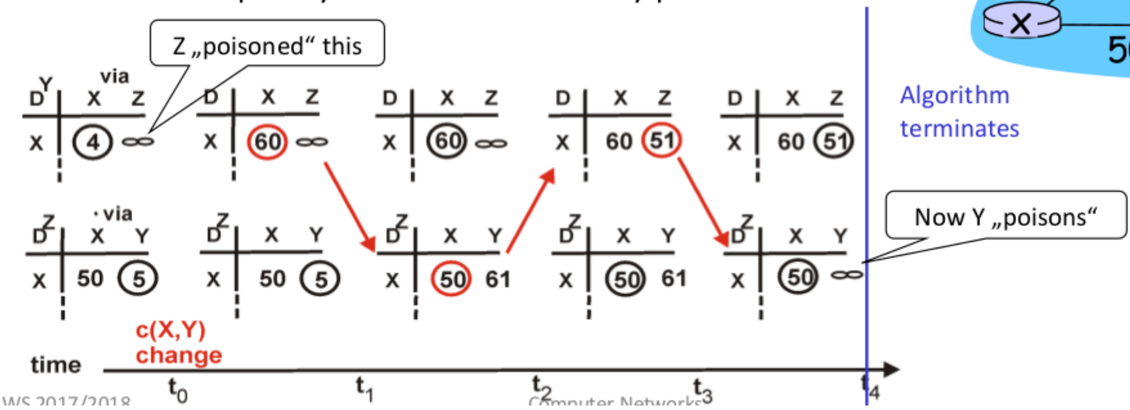


^ Umgehbar mit Poisoned Reverse:

Poisoned reverse

- If Z routes through Y to get to X:
- Z tells Y its (Z's) distance to X is infinite (so Y won't route to X via Z)
- Will this completely solve count to infinity problem?





Comparison of LS and DV algorithms

Message complexity

- LS: with n nodes, E links, O(nE) messages sent
- DV: exchange between neighbors only
  - Convergence time varies

Speed of convergence

- LS: $O(n^2)$ algorithm requires O(nE) messages
  - May have oscillations
- DV: convergence time varies
  - May be routing loops
  - Count-to-infinity problem

Robustness: what happens if router malfunctions?

- LS:
  - Node can advertise incorrect link cost
  - Each node computes only its own table
- DV:
  - DV node can advertise incorrect path cost
  - Each node's table used by others
  - Error propagate thru network

LS: Kennt alle Kosten (?)
DV: Kennt nur Nachbar
→ Beide Arten sind 'idealisiert'

Making Routing Scalable

- Our routing study thus far - idealized
  - all routers identical & network "flat"

  ... not true in practice

- Scale: with billions of destinations:
  - Can't store all destinations in routing tables!
  - Routing table exchange would swamp links!
- Administrative autonomy
  - Internet = network of networks
  - Each network admin may want to control routing in its own network

Aggregate routers into regions known as "autonomous systems" (AS) (a.k.a. "domains")

Intra-AS Routing

- Routing among hosts, routers in same AS ("network")
- All routers in AS must run same intra-domain protocol
- Routers in different AS can run different intra-domain routing protocol
- Gateway router: at "edge" of its own AS, has link(s) to router(s) in other AS'es

Inter-AS Routing

- Routing among AS'es
- Gateways perform inter-domain routing (as well as intra-domain routing)