Verteilte Systeme

Routing

Leon Aaron Kaplan
(Funkfeuer)

Institut für
Computertechnik

ICT

Institute of
Computer Technology

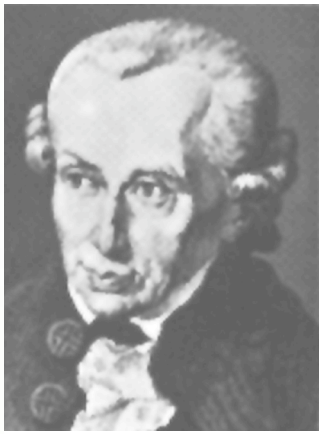
Agenda

- Was ist Routing?
- Was ist Routing NICHT?
- Übersicht Routing Probleme
- Was sagt die Graphentheorie zu Routing?
- Optima
- Distance Vector vs. Link State
- MANETs - Theorie und Bsp

Aussprache

- “Route 66” und nicht “Raut 66” [ˈraʊtɪŋ]
- Aber ist das nicht das, was Trüffelschweine suchen? ;-)
- --> viele in den USA: es heisse “Raut”. Brit. Aussprache [ˈruːtɪŋ]
- ISO sagt, es soll sogar “routeing” geschrieben werden (britische Schreibweise)
- Ich verstehe beide Aussprachen

Warum ist routing wichtig?

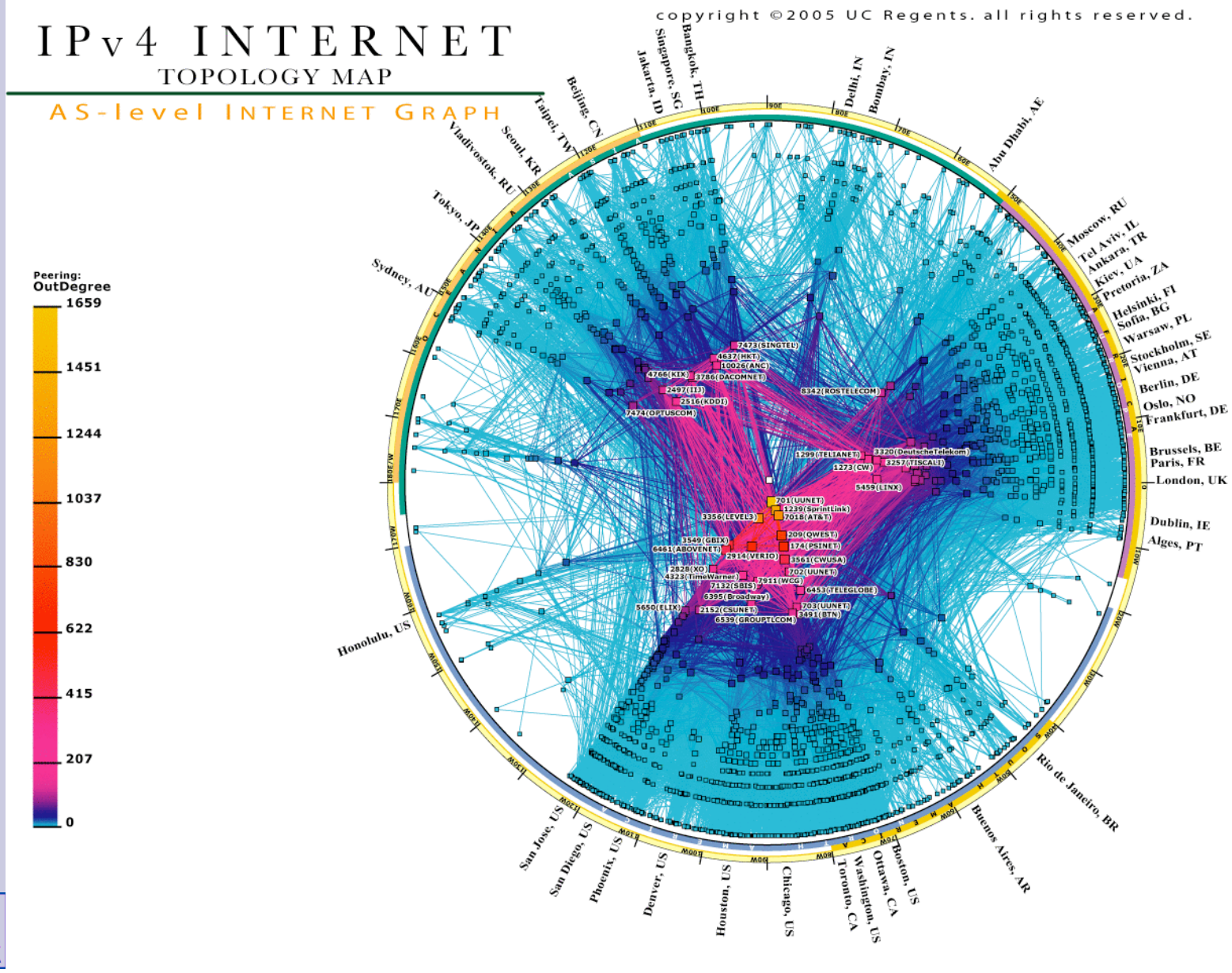


- Wo (wer) bin ich?
- + Woher komme ich?
- + Wohin (soll) ich gehen?

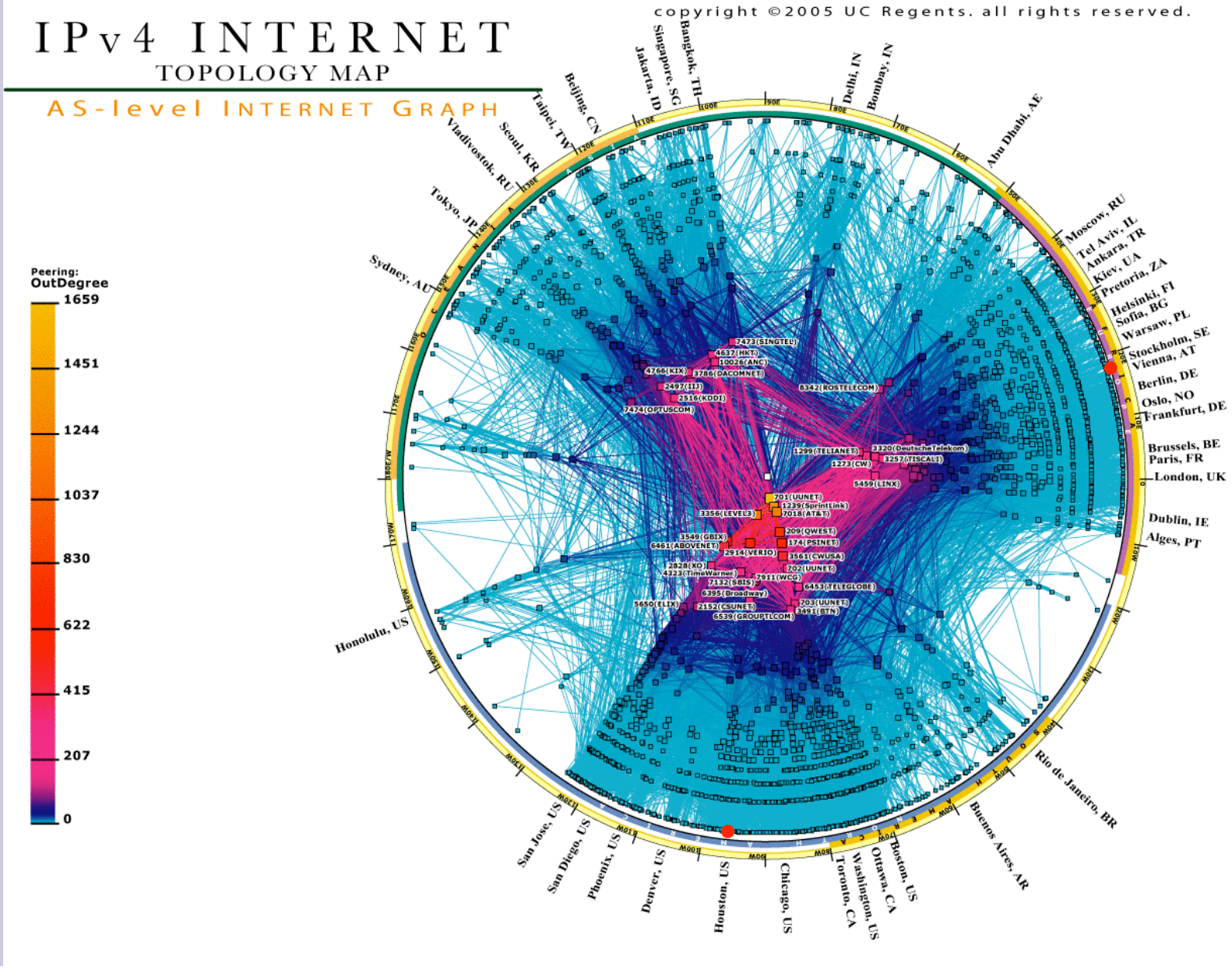
- = The eternal questions of philosophy

- Wenn das beantwortet wird, kann **Routing** sagen, **wie** es am besten geht :)

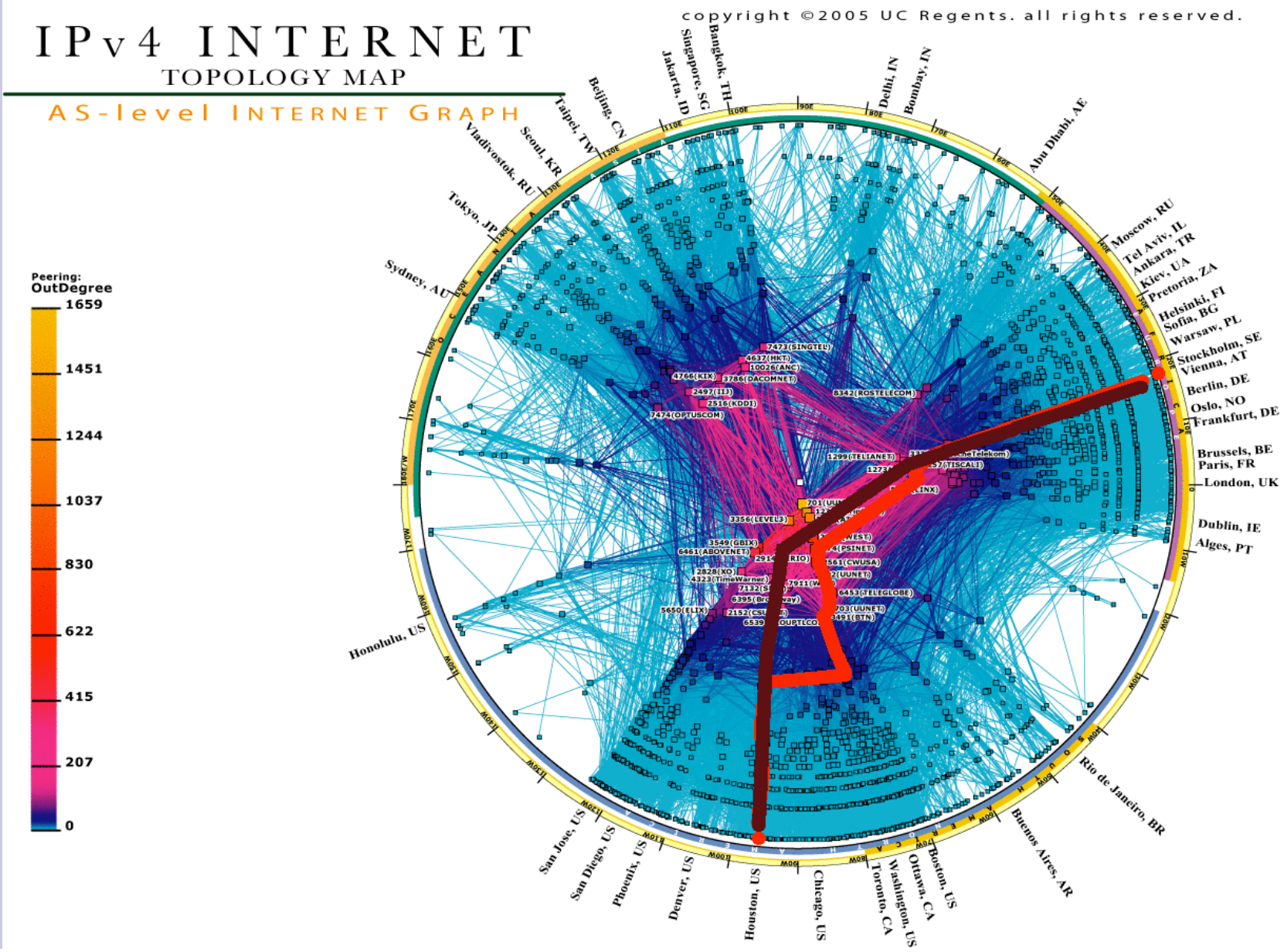
Wo bin ich?


IPv4 INTERNET
TOPOLOGY MAP
AS-level INTERNET GRAPH
copyright ©2005 UC Regents. all rights reserved.
Peering:
OutDegree
1659
1451
1244
1037
830
622
415
207
0
Singapore, SG
Jakarta, ID
Bangkok, TH
Delhi, IN
Bombay, IN
Abu Dhabi, AE
Moscow, RU
Tel Aviv, IL
Ankara, TR
Kiev, UA
Pretoria, ZA
Helsinki, FI
Sofia, BG
Warsaw, PL
Stockholm, SE
Vienna, AT
Berlin, DE
Oslo, NO
Frankfurt, DE
Brussels, BE
Paris, FR
London, UK
Dublin, IE
Alges, PT
Rio de Janeiro, BR
Buenos Aires, AR
Boston, US
Ottawa, CA
Washington, US
Toronto, CA
Chicago, US
Houston, US
Denver, US
Phoenix, US
San Diego, US
San Jose, US
Honolulu, US
Sydney, AU
Tokyo, JP
Vladivostok, RU
Seoul, KR
Taipei, TW
Beijing, CN
7473(SINGTEL)
4637(HKT)
10026(ANC)
4766(KIX)
3786(DACOMNET)
2497(IIJ)
2516(KDDI)
7474(OPTUSCOM)
8342(ROSTELECOM)
1299(TELIANET)
3320(DeutscheTelekom)
1273(CW)
3257(TISCALI)
5459(LINX)
701(UUNET)
1239(SprintLink)
7018(AT&T)
3356(LEVEL3)
3549(GBIX)
209(QWEST)
6461(ABOVENET)
174(PSINET)
2914(VERIO)
3561(CWUSA)
2828(XO)
702(UUNET)
4323(TimeWarner)
7911(WCG)
7132(SBIS)
6453(TELEGLOBE)
6395(Broadway)
5650(ELIX)
703(UUNET)
2152(CSUNET)
3491(BTN)
6539(GROUPTLCOM)

Wo will ich hin?



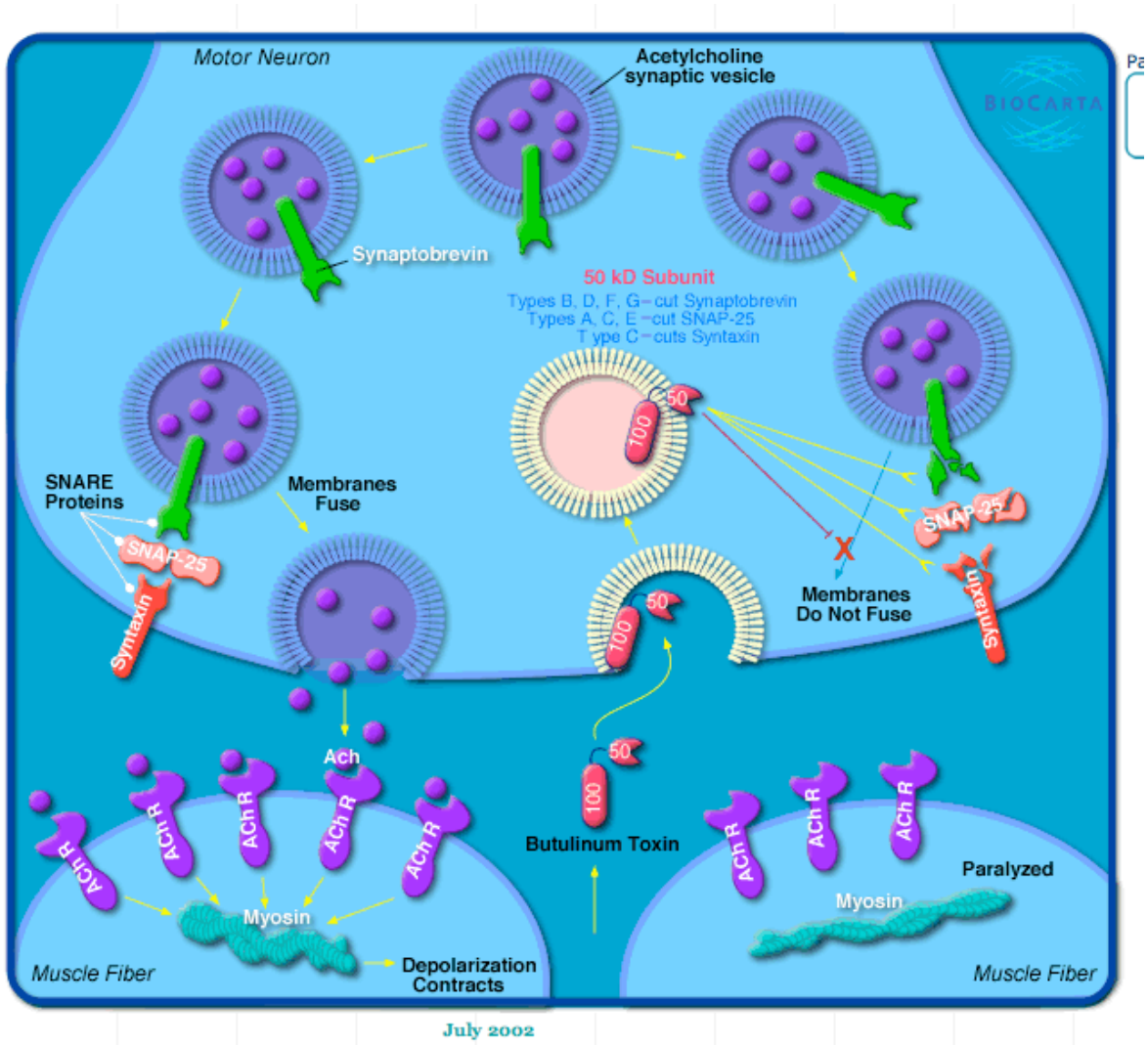
Wie komme ich (gut) dort hin?



Was ist Routing NICHT?

- Das eigentliche Übertragen der Datenpakete (*)
- Vermitteln != Übertragen ("**Forwarding**") der Datenpakete (*)
- Geographisches Routing / GPS (zumindest nicht in unserer VO)
- Platinenlayout
- Routing in Supercomputern / Parallelrechnern
  - --> aber vgl. HyperTransport Bus (hat auch Packets)
- "Route of administration" - der Weg, den ein Medikament/Gift zur Aufnahme in den Körper braucht
- Bei uns in der VO: circuit switched routing wird nicht betrachet
- (*) die Leitung kann gut sein, aber das Routing zeigt die Route nicht an!

Was ist Routing NICHT? (Signal pathways)



Was ist “gut”?

- Einfachheit!
- Wenig overhead
- Skalierbarkeit
- Optimalität
- Robustheit
- Konvergenz
- Ohne Schleifen wenn möglich ;-)
- Minimaler Hop count (internet)
- Latency
- Packet loss
- Bandbreite (BW)
- Geringer jitter (VOIP)
- Billiger ($$$) vgl. Least cost router
- Global gesehen weniger Last
- ---> wir brauchen eine **Metrik** / ein **Maß**!

Algorithmus aus der meta Perspektive

Einzelne zu optimierende Ziele

Übersicht Probleme bei Routing Algorithmen (I.e. “Wann ist etwas nicht ‘gut’?”)

- Route Flapping
- Routing Loops
- Suboptimale Routen
- Falsche / fehlende Routen
- Schlechte Abstimmung Layer 3 und Layer 2
- Starvation von gewissen Packets / Fairness
- Starvation von routing protocol packets
- Split Horizon -> Split brain problem (Bsp IPv6 + v4)

Instabilitäten (Route Flapping)

t_0: A_1 ist bester next hop

t_1: A_2 ist bester next hop --> traffic geht über A_2 --> A_2 wird jetzt überlastet.

t_2: -> kein traffic über A_1 --> A_1 ist bester next hop

etc.

Effektiv geht so sehr wenig traffic über das Netz, da beide next hops mit dem traffic überlastet sind.

Lösung 1: Hysterese Kurve

Lösung 2: multipath Routing/load balancing

Inkonsistenzen (Loops)

- A_1 -> A_2 -> .. A_{n-1} -> A_n -> A_{n-1} -> ...
- Erkennung: traceroute
- Problem: Daten bleiben “im Kreis” gefangen -> Übersättigung des Netzes in kürzester Zeit
- Ursache: entsteht häufig durch verschiedene / nicht abgestimmte Sicht der Welt/Topologie
- Lösung: TTL

Falsche / fehlende Routen

- Fall 1: unabsichtlich falsch konfiguriert
- Fall 2: Absicht
- Fall 3: buggy Protokoll (bei reifen Protokollen kaum noch der Fall)

- Lösung: alle Situationen fallen auf. Spätestens per Anruf um 2 a.m. :)
- Ad Fall 2: bei BGP wird der Knoten auf “ignore” gesetzt. Bei einem IGP ist so ein Knoten in unserer Kontrolle.

Bsp: real existierender IPv6+v4 Bug in CISCOs

- GW A_1, A_2 ist redundant (heartbeat, Linux)
- A_1 ----> Switch S ----> B
- A_2 ----> Switch S ----> B
- Switch S ----> uplink C

t_0: A_1 fällt aus

t_1: A_2 übernimmt IP (aber nicht MAC) von A_1

t_2: B lernt die neue MAC von "A", schickt an die IP(A_1) = IP(A_2), aber an die MAC von A_1 !!

A_2 erkennt dadurch seine IP packets nicht mehr.

ABER B glaubt noch immer, dass er der beste Uplink ist.

Das Problem ist ein reines dualstack (IPv4 + IPV6) Problem.

Asymetrisches Routing

- Typischer Fall:

- PC mit privater IP (zB chello home network) bekommt eine private IP
- Problem: manche Services (H.232) hätten gerne öffentliche IP

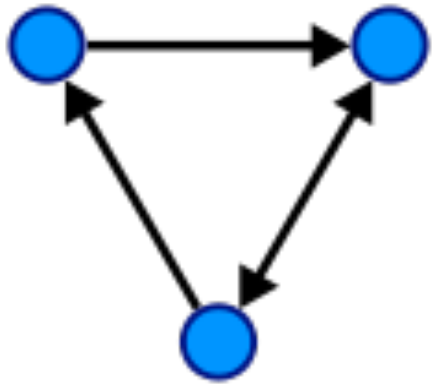
- Lösung:
- Reinrouten über einen Tunnel
- Rausrouten über NAT (Network address translation)

Wichtige Kategorien/Dimensionen von Routingalgorithmen

- Dynamisches Routing
  - IGP
    - OSPF (open shortest path first)
    - RIP (routing information protocol)
  - Mobile Ad-Hoc routing (MANETs)
    - OLSR (optimized link state routing)
    - AODV (Ad-hoc On Demand Distance Vector)
    - HSLS (hazy sighted link state routing)
  - EGP
    - BGP (border gateway protocol)
- Statisches Routing
- Source Routing
- MPLS (multi protocol label switching), vgl. ATM
- Link State vs. Distance Vector

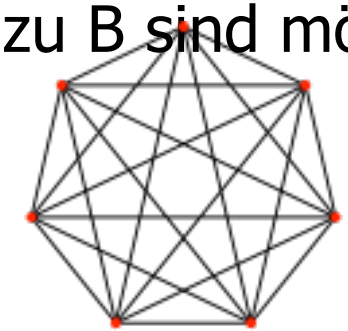
Mathematisches (Wiederholung)

- G(V,E) Graph, Knoten (vertices V), Kanten (edges E)
- V= {v_1, v_2, …, v_n}
- E = {(v_1,v_2, m_1), (v_2, v_3, m_2), (v3, v2, m_3), (v_3, v_1, m_4), …}
- Knotengrad
- m_x = metrik Wert
- Directed Graph
- Vs. Undirected Graph



Mathematisches 2

- Pfad = eine Folge von Kanten (die verschieden ist)
- Weg = eine Folge von Kanten
- Zyklus = Weg, sodass Start = Ende
- Kreis = Weg, sodass *nur* Start = Ende
- TSP
- Multigraph = mehrere Kanten von A zu B sind möglich (evtl. mit verschiedenen Metriken)
- Vollständiger Graph K_7:



- Graphentheorie is ***sehr*** reichhaltig! Eigene VO

Kürzester Weg (Dijkstra)

- Algorithmus von Edsger **Dijkstra**
- Geht nur für $m_x >= 0$
- $m_x = w((u,v))$: w: E -> [0, ∞)
- Input: s Startknoten, G graph, v Ziel
- Output: d[v] = kürzeste distance zwischen s und v
- Idee: edge relaxation: Wenn (s…u) ein kürzester Weg ist und u eine Kante nach v hat , dann is d[u] + w(u,v) ein Weg. Wenn der kürzer als der bisher kürzeste Weg nach v is, übernehme dies (d[v] = d[u]+w(u,v))
- Breitensuche

Dijkstra Algo

```
1  function Dijkstra(G, w, s)
2   for each vertex v in V[G]                  // Initializations
3         d[v] := infinity                     // Known distance function from s to v
4         previous[v] := undefined
5   d[s] := 0                                  // Distance from s to s
6   S := empty set                             // Set of all visited vertices
7   Q := V[G]                                  // Set of all unvisited vertices
8   while Q is not an empty set                // The algorithm itself
9     u:= Extract_Min(Q)                       // Remove best vertex from priority queue
10    S := S union {u}                         // Mark it 'visited'
11    for each edge (u,v) outgoing from u
12      if d[u] + w(u,v) < d[v]                // Relax (u,v)
13            d[v] := d[u] + w(u,v)
14            previous[v] := u
```

wobei

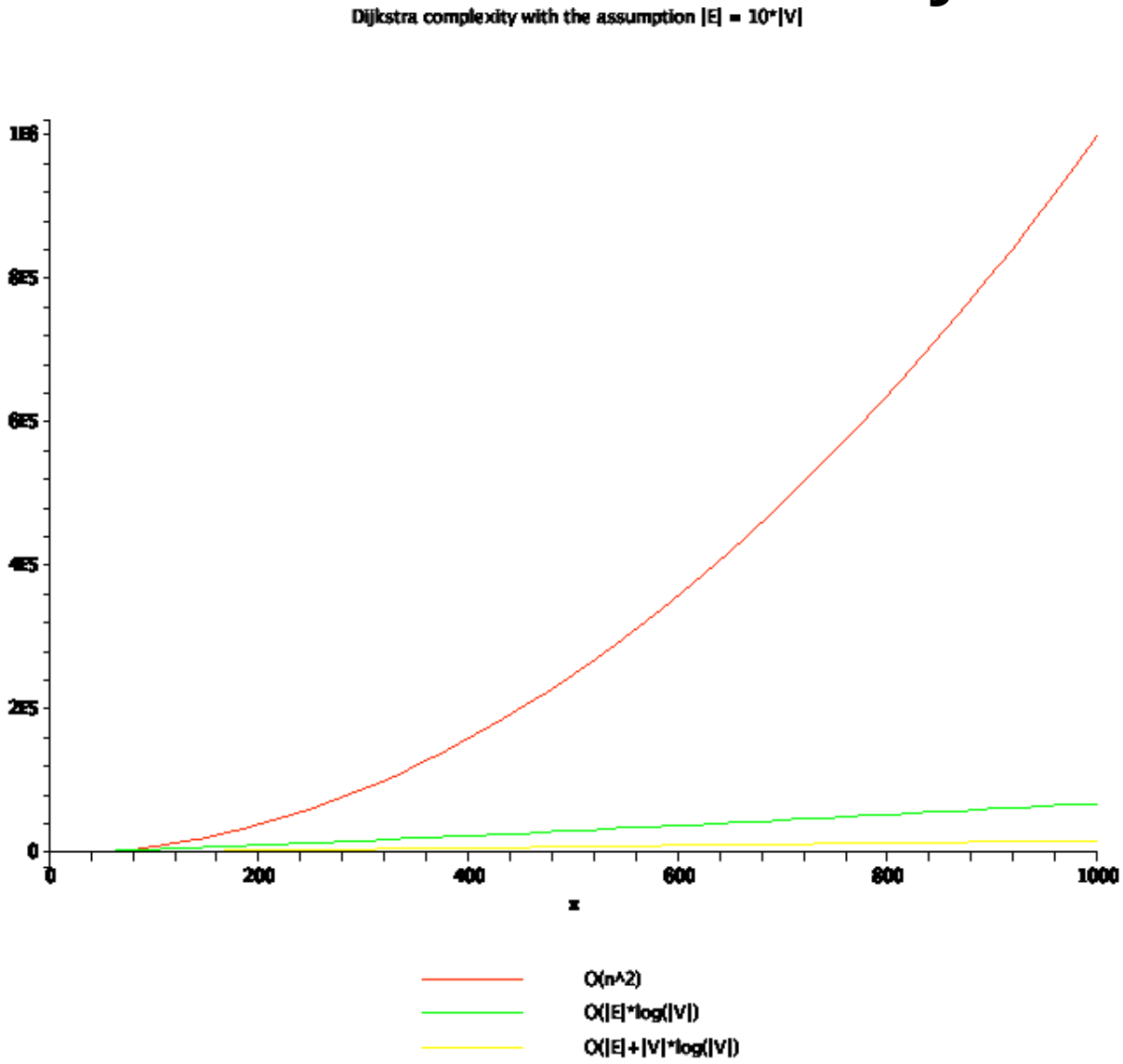
u:= Extract_Min(Q) ….. Suche u in Q, sodas d[u] is minimal, u wird von Q herausgenommen (Greedy)

Dijkstra Aufwand

- O(|V|*|E|) …. k*|E| ~ |V| bei scale free networks (sehr vereinfacht), wenn ineffizient implementiert
- ➔ O(|V|*|E|) ~ O($|V|^2$)…
- Internet: V > 10^6 :) -> 10^12 Operationen ??
- Folge: nicht jeder ist Router im Internet. Das ist die Begründung für IGP. Kleine Netze!!

- Optimiertere Implementation: Priority Queue
- Extract-Min wird dadurch optimiert.
- ➔ O(|E| * log(|V|)) oder sogar (Fibonacci Heap) O(|E| + |V|*log(|V|))

Dijkstra Aufwand



Dijkstra Anwendungen

- OSPF
- IS-IS
- OLSR (TODO: Dijkstra optimieren!)
- Logistikunternehmen
- etc.

Bellman Ford

- 1958 R. Bellman, “On a routing Problem”
- Vorteil: negative Gewichte sind möglich
- Problem: Zyklen mit nur negativen Gewichten gehen gegen -∞. Muss berücksichtigt werden.
- Nachteil: Komplexität ist höher O(|V|*|E|). Es wird jeder Knoten durchwandert und für den nochmals alle Kanten.

Bellman Ford Algo

```
function BellmanFord(list vertices, list edges, vertex source)

    // This implementation takes in a graph, represented as lists of
    vertices
    // and edges, and modifies the vertices so that their distance and
    // predecessor attributes store the shortest paths.
    // Step 1: Initialize graph
    for each vertex v in vertices:
     if v is source then v.distance := 0
     else v.distance := infinity
     v.predecessor := null

    // Step 2: relax edges repeatedly
    for i from 1 to size(vertices):
     for each edge uv in edges:
         u := uv.source
         v := uv.destination              // uv is the edge from u to v
         if v.distance > u.distance + uv.weight:
             v.distance := u.distance + uv.weight
             v.predecessor := u

    // Step 3: check for negative-weight cycles
    for each edge uv in edges:
     u := uv.source
     v := uv.destination
     if v.distance > u.distance + uv.weight:
         error "Graph contains a negative-weight cycle"
```

Bellman-Ford Anwendungen

- RIP
- BGP
- Generell DV Algos (AODV)

Optima: Max Flow

- Sei c(u,v) die capacity der Kante (u,v) (zB bits/sec)
- Max flow := ein flow durch G, der maximal ist
- Flow (s..t) durch G :=
  - 1. f(u,v) <= c(v,u)
  - 2. f(u,v) = f(v,u)
  - 3. $\Sigma_{w \in V}$ f(u,w) = 0, ausser u=s oder u=t (was bei u reingeht, muss rausgehen.. Vgl. Kirchhoff)
- Algorithmus: Ford-Fulkerson Max Flow

  O(|E| * f) … f ist max flow

Optima: Wardrop Equilibrium

- Transportnetzwerke - John Glen Wardrop 1952
- Vgl. Nash equilibrium in Gametheory
- Wardrop 1st Principle:
  jedes Fahrzeug / Agent soll den Weg finden, der optimal ist (eine Änderung der Route würde mehr Kosten verursachen) (UE)
- Wardrop 2nd Principle:
  At equilibrium the average journey time is minimum. (System optimal flows)
- B.A.T.M.A.N (http://www.open-mesh.net) ?

Link State Routing

- Sage der Welt, wer deine Nachbarn sind
- Vorteil: Redundanz -> weniger loops, skaliert besser, schnelle Konvergenz
- Typisch für LS:
  - HELLO packets
  - Updates des LS durch flooding
  - Dijkstra
  - Berechnung des kürzesten Weges zu jedem anderen Router
  - Schnelle Konvergenz

Jeder router weiss alles

Distance Vector

- Teile **nur** deinen Nachbarn mit, wie gut die links sind. Bsp: RIP
- Typisch für DV:
  - Bellman Ford (-> skalierbarkeit)
  - Schlechte Konvergenz (temporäre Loops!)

<quote Wikipedia>

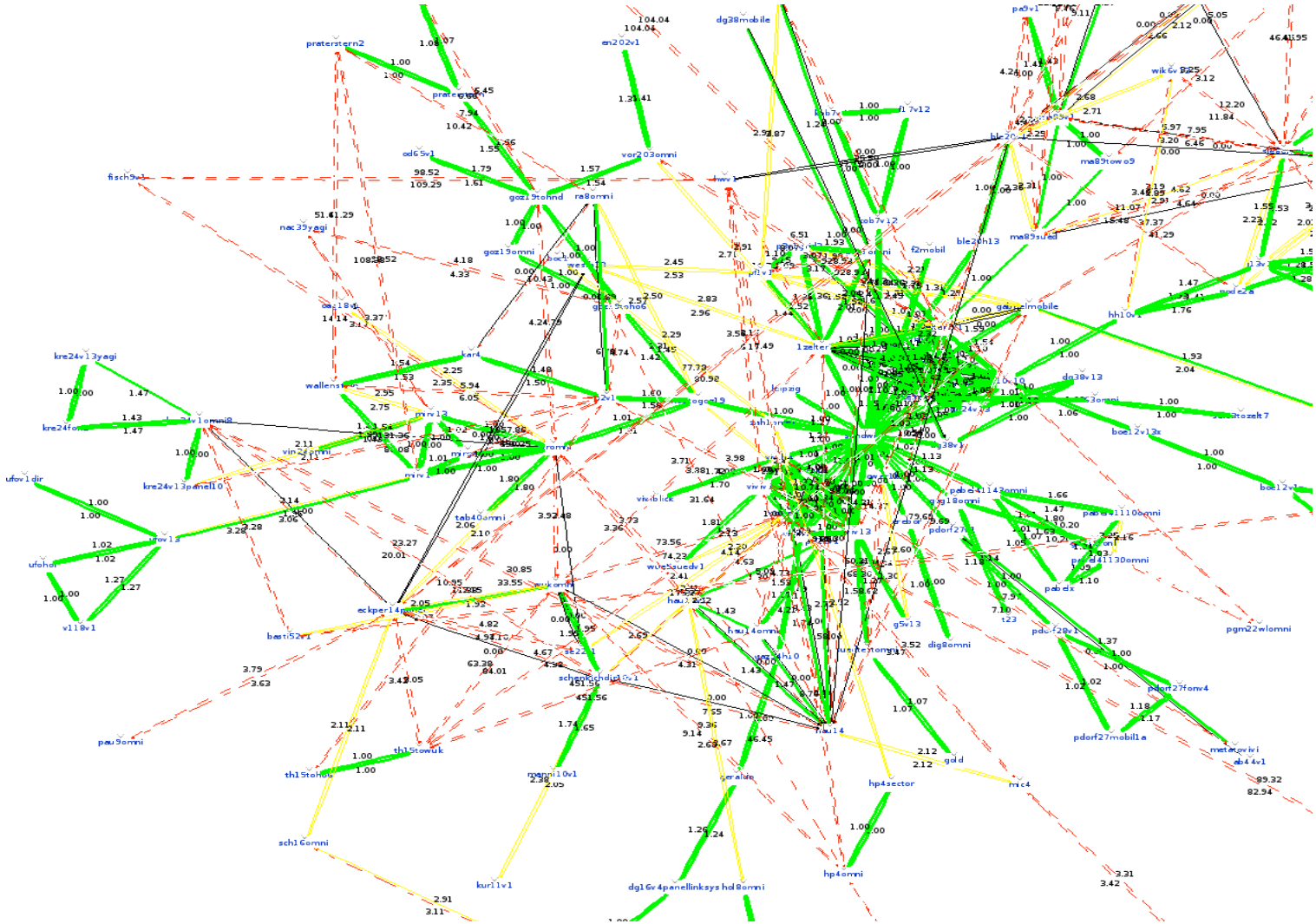
1. Erzeuge eine Kostenmatrix, welche Router über welche Nachbarn und zu welchen Kosten erreichbar sind. - Diese Matrix enthät anfangs nur die (bekannten) Kosten zu direkten Nachbarn.
2. Erzeuge eine Aufstellung mit Informationen, welche Router wir zu welchen Kosten am besten erreichen könen und schicke sie an alle Nachbarn.
3. Warte auf Aufstellungen dieser Art von anderen Routern, rechne diese dann in die eigene Kostenmatrix ein.
4. Ändern sich dadurch die minimalen Kosten, zu denen wir einen Router erreichen können: fahre mit Schritt 2 fort, sonst mit Schritt 3.

</quote>

MANETs Übersicht

- MANET = **m**obile **a**d-hoc **net**work
- Das internet ist so gesehen ein sehr langsames MANET
- -> wichtig ist der Grad der mobilität
- Ad-hoc: Wir müssen keine grosse Infrastruktur aufbauen
- Unterscheidung: Proaktiv vs. reactiv vs. hybrid
- Meist wireless, muss aber nicht sein

MANETs

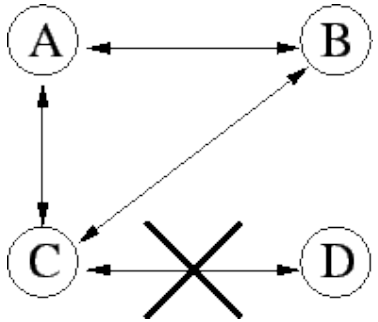


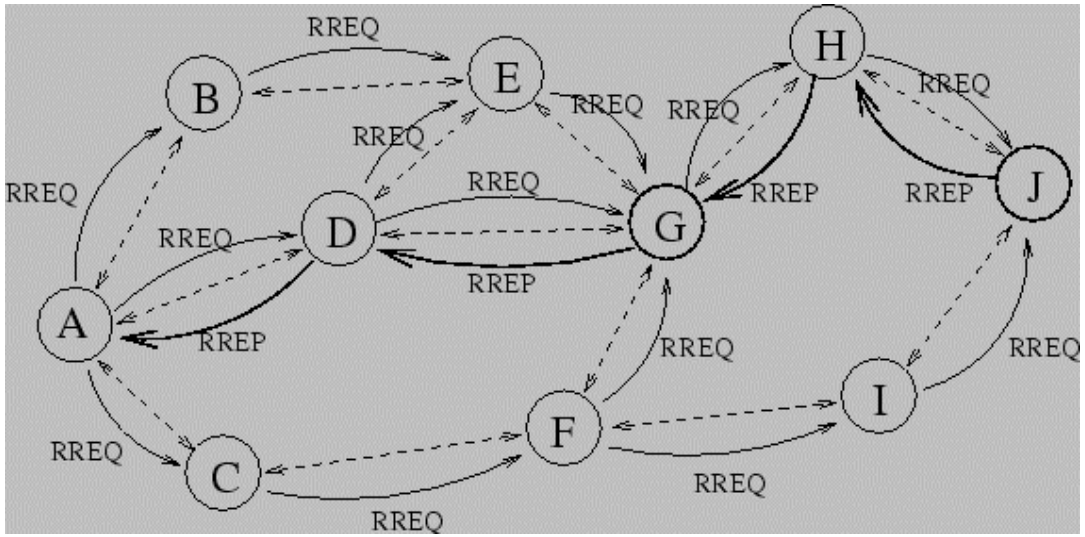
MANETs

- Notwendige MANET komplementär Technologien:
- IP autoconfig
- Security (trust)
- Service autodiscovery
- Power adaption

AODV

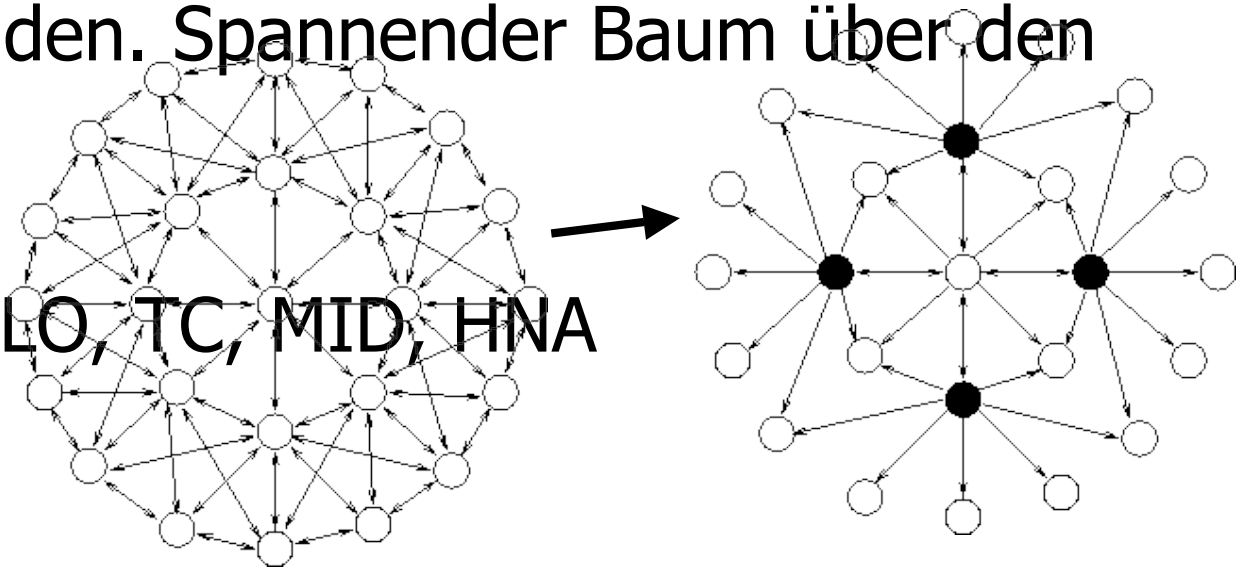
- Distance vector protocol, reaktiv
- RREQ, RREP, RERR packets
- Broadcasts
- Counting to infinity problem -> SEQ number
- RFC 3561





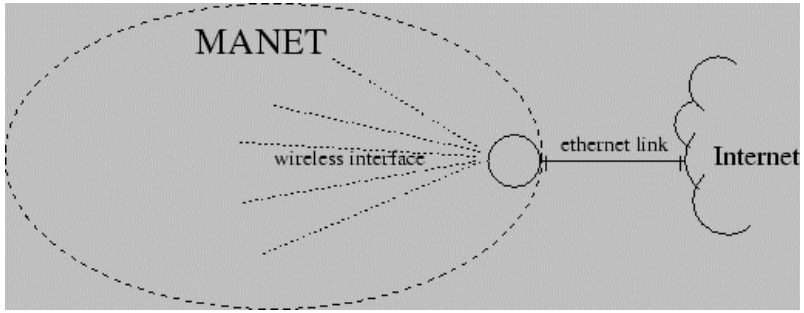
OLSR

- Optimized Link State Routing
- RFC 3626
- Beste Implementierung: www.olsr.org
- Diplomarbeit Andreas Tønnesen
- Optimized -> es werden MPRs (multi point relays) gefunden. Spannender Baum über den Graphen
- Flooding
- Packets: HELLO, TC, MID, HNA



OLSR

- Hello packet -> ETX (expected transmission count). Vgl. MIT roofnet (freifunk erweiterung)
- MID packet = alias interfaces
- HNA packet = host and network association
- TC packet = topology control



- Per plugins erweiterbar. Daten können automatisch geflooded werden. Praktisch!
- TC redundancy ist wichtig
- Freifunk Erweiterung: fisheye: TTLs variieren (vgl. HSLS)
- Im weltweiten Einsatz bei freien Netzen. Berlin ~ 600 Knoten

OLSR-NG

- http://olsr.funkfeuer.at
- Ziel: OLSR von olsr.org skalierbarer zu machen
- Open Source Projekt
- --> mailto:aaron@lo-res.org oder bernd@firmix.at
- Thx 2 IPA / nic.at

OLPC

- OLPC = one laptop per child (150$ laptop)
- Inkludiert mesh 802.11s (layer 2 mesh)
- Arbeitet auch, wenn zugeklappt
- Marvell chipsatz, 802.11s
  stack im chip, unfreie FW :(
- Verbrauch ~ 600mW
- Reichweite: bis zu 1.4km

Aktuelle Trends

- Skalierbarkeit von MANETs
- Kombination routing + andere layer
- Power aware routing (MOTEs)
- Ant based / Pheromone based routing
- Gametheory für routing