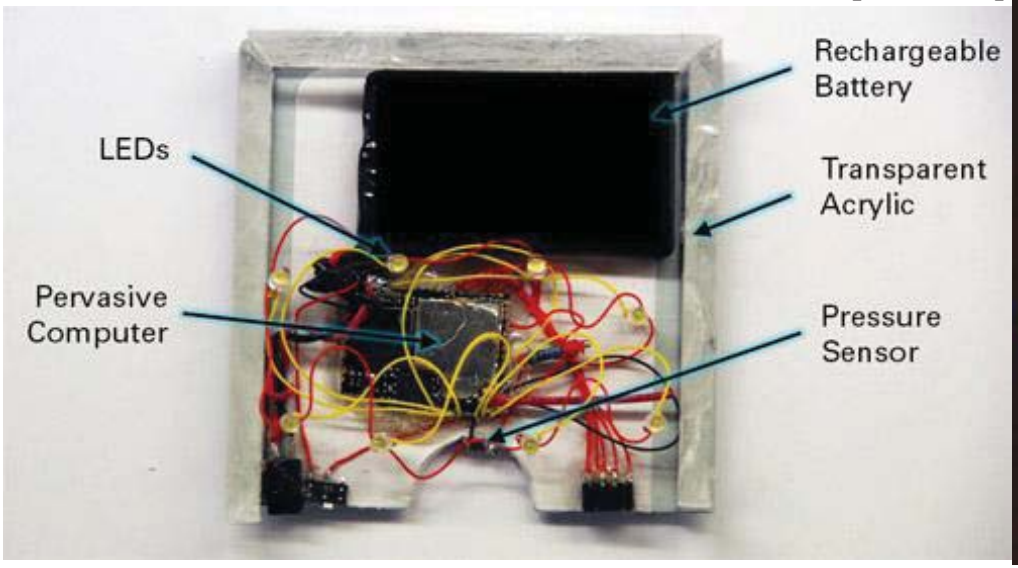
Routing

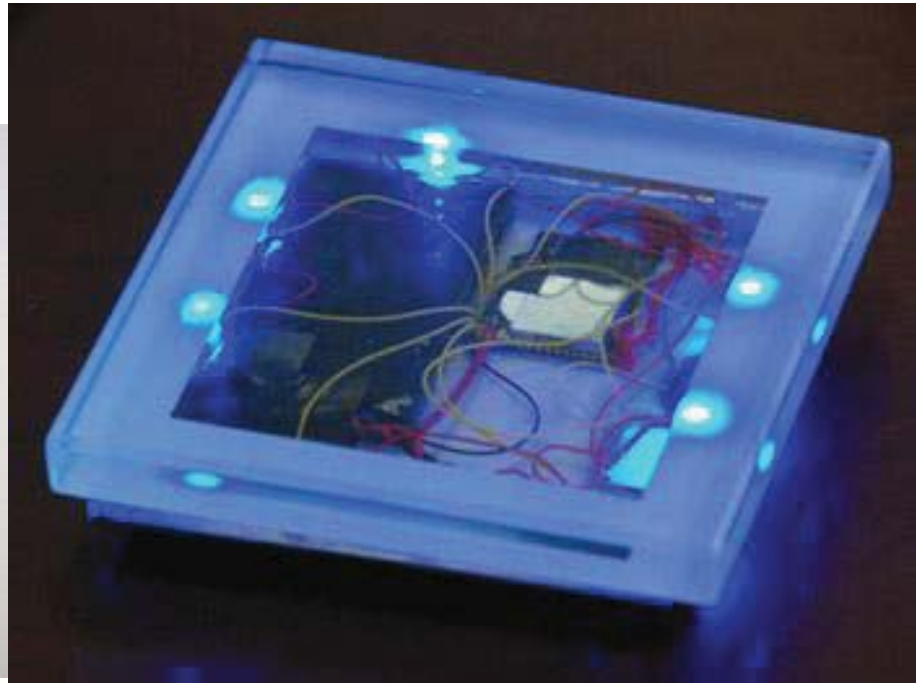
Chapter 12

Application of the Week: Games / Art

- Uncountable possibilities, below, e.g., a beer coaster that can interact with other coasters…

[sentilla]





Rating

- Area maturity

- Practical importance

- Theory appeal

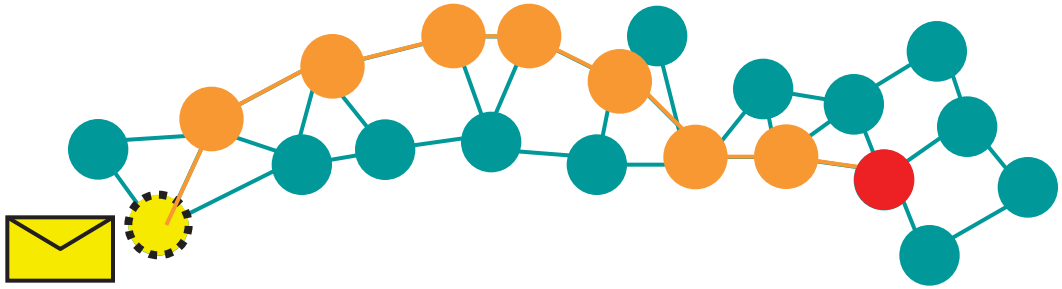
Overview

- Characteristics and models
- 10 tricks for classic routing

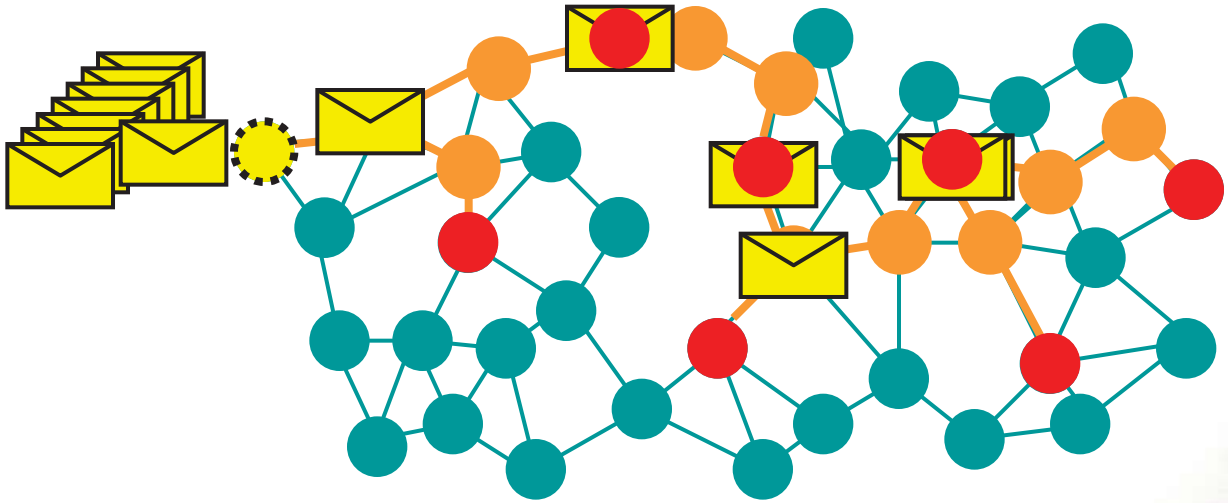
- Selected case studies
  - Link reversal routing
  - Compact routing
  - Geo-routing without geometry

Network Layer Services

- Unicast (send message to a given node)

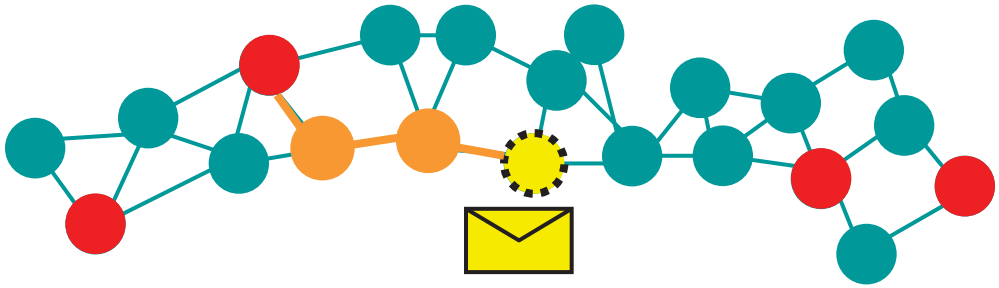


- Multicast (send message to a given set of nodes)



Network Layer Services (2)

- Anycast (send message to *any* node of a given set)



- More
  - Broadcast (special form of multicast, to everybody)
  - Convergecast (data gathering, reverse of broadcast)
  - Geocast (routing to a specific location)
  - …

Remember: Discussion of Classic Routing Protocols

- Proactive Routing Protocols
- Both link-state and distance vector are “proactive,” that is, routes are established and updated even if they are never needed.
- If there is almost no mobility, proactive algorithms are superior because they never have to exchange information and find optimal routes easily.

- Reactive Routing Protocols
- Flooding is “reactive,” but does not scale
- If mobility is high and data transmission rare, reactive algorithms are superior; in the extreme case of almost no data and very much mobility the simple flooding protocol might be a good choice.

There is *no* “optimal” routing protocol; the choice of the routing protocol depends on the circumstances. Of particular importance is the mobility/data ratio.

Routing in Ad-Hoc Networks

- Reliability
  - Nodes in an ad-hoc network are not 100% reliable
  - Algorithms need to find alternate routes when nodes are failing

- Mobile Ad-Hoc Network (MANET)
  - It is often assumed that the nodes are mobile (“Car2Car”)

- 10 Tricks → 2^{10} routing algorithms
- Let‟s see some of these classic tricks…

Radius Growth

- Problem of flooding (and protocols using flooding): The destination is two hops away, but we flood the whole network

- Idea: Flood with growing radius; use time-to-live (TTL) tag that is decreased at every node, for the first flood initialize TTL with 1, then 2, then 4 (really?),
  - How do we stop once the destination is found?

- Alternative idea: Flood very slowly (nodes wait some time before they forward the message) – when the destination is found, it initiates a fast flood that stops the slow flood

+ Tradeoff time vs. number of messages

Source Routing

- Problem: Why should nodes store routing information for others?
- Idea: Source node stores the whole path to the destination; source stores path with every message, so nodes on the path simply chop off themselves and send the message to the next node.
- “Dynamic Source Routing” discovers a new path with flooding (message stores history, if it arrives at the destination it is sent back to the source through the same path)

\+ Nodes only store the paths that they need

– Not efficient if mobility/data ratio is high

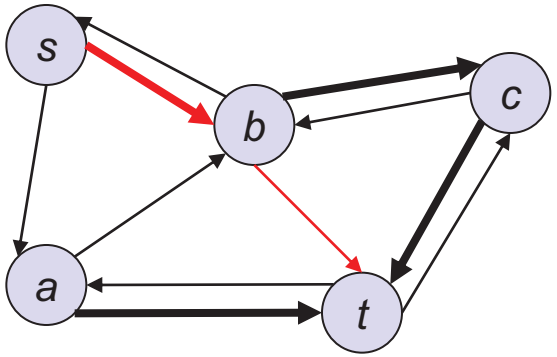
– Asymmetric Links?

Asymmetric Links

- Problem: The destination cannot send the newly found path to the source because at least one of the links used was unidirectional.

- Idea: The destination needs to find the source by flooding again, the path is attached to the flooded message. The destination has information about the source (approximate distance, maybe even direction), which can be used.

- In theory, if stations are homogeneous, the received signal strength should be equal.
  - However, noise and interference experienced might differ.
  - How can we figure out whether an asymmetric link is vital?



Re-use/cache routes

- This idea comes in many flavors
  - Clearly a source *s* that has already found a route "*s-a-b-c-t*" does not need to flood again in order to find a route to node *c*.
  - Also, if node *u* receives a flooding message that searches for node *v*, and node *u* knows how to reach *v*, *u* might answer to the flooding initiator directly.
  - If node *u* sees a message with a path (through *u*), node *u* will learn (cache) this path for future use.

\+ Without caching you might do the same work twice

– Which information is up-to-date? Sequence numbers for updates

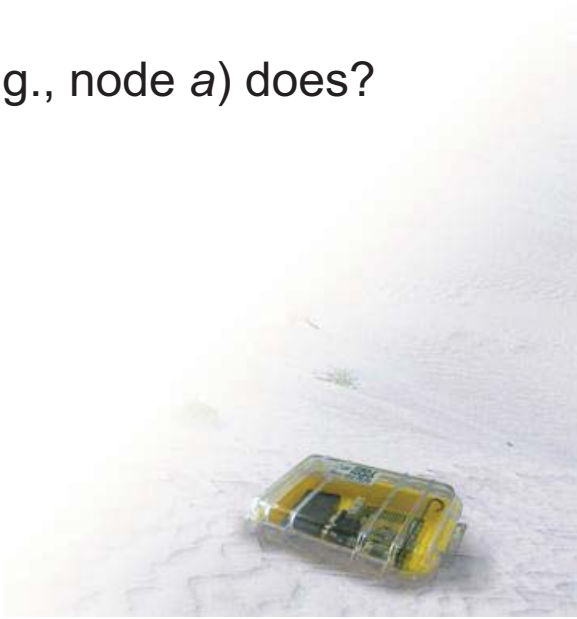
– Caching is somewhat in contradiction to the source routing philosophy, because you start building routing tables again

Local search

- Problem: When trying to forward a message on path "*s-a-u-c-t*" node *u* recognizes that node *c* is not a neighbor anymore.

- Idea: Instead of not delivering the message and sending a NAK to *s*, node *u* could try to search for *t* itself; maybe even by flooding.
  - Some algorithms hope that node *t* is still within the same distance as before, so they can do a flooding with TTL being set to the original distance (plus one)
  - If *u* does not find *t*, maybe the predecessor of *u* (e.g., node *a*) does?

– Sometimes this works, sometimes not.



Hierarchy

- Problem: Especially proactive algorithms do not scale with the number of nodes. Each node needs to store big tables

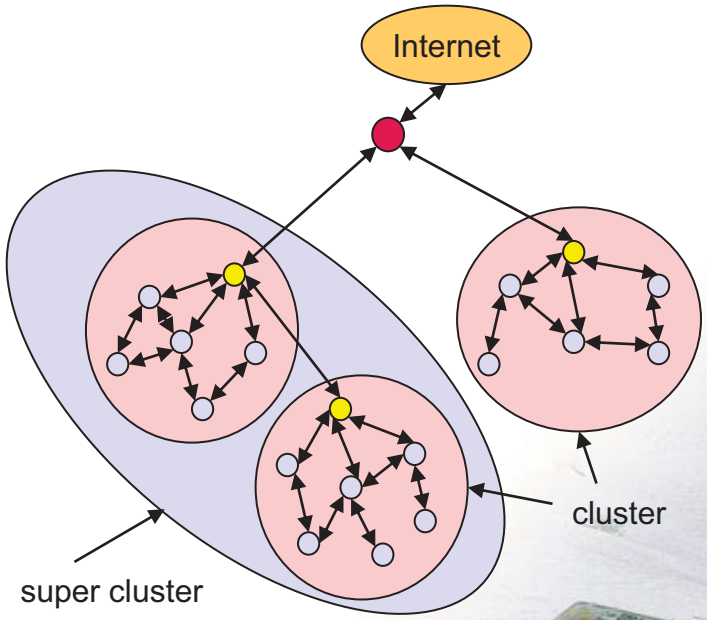
- Idea: In the Internet there is a hierarchy of nodes; i.e. all nodes with the same IP prefix are in the same direction. One could do the same trick in ad-hoc networks

+ Well, if it happens that ad hoc nodes with the same numbers are in the same area, hierarchical routing is a good idea. In static networks this is a good idea, and indeed we‟ll see an example of that later.
– In MANETs this is a problem…

More concretely: Clustering

- Idea: Group the ad hoc nodes into clusters (even hierarchically). One node is the head of the cluster. If a node in the cluster sends a message it sends it to the head which sends it to the head of the destination cluster which sends it to the destination

+ Simplifies operation for most nodes (that are not cluster heads); this is particularly useful if the nodes are heterogeneous and the cluster heads are “stronger” than others.
– A level of indirection adds overhead.
– There will be more contention at the cluster heads, but this might be solved by re-computing the cluster heads from time to time, or by computing domatic partitions…



Implicit Acknowledgement

- Problem: Node *u* only knows that neighbor node *v* has received a message if node *v* sends an acknowledgement.

- Idea: If *v* is not the destination, *v* needs to forward the message to the next node *w* on the path. If links are symmetric (and they need to be in order to send acknowledgements anyway), node *u* will automatically hear the transmission from *v* to *w* (unless node *u* has interference with another message).

- Can we set up the MAC layer such that interference is impossible?

Smarter updates

- Sequence numbers for all routing updates

\+ Avoids loops and inconsistencies
\+ Assures in-order execution of all updates

- Decrease of update frequency
- Store time between first and best announcement of a path
- Inhibit update if it seems to be unstable (based on the stored time values)

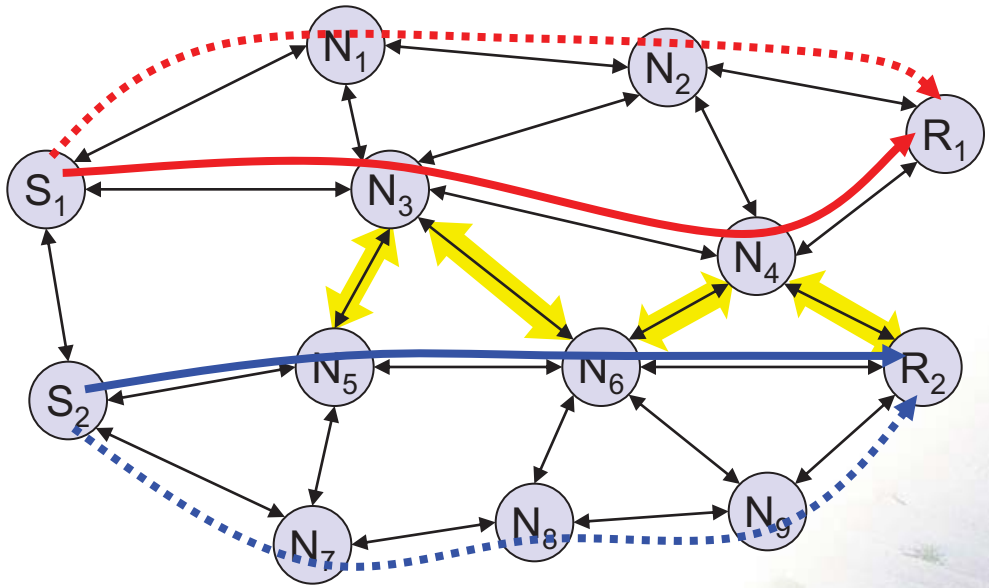
\+ Less traffic

- Implemented in Destination Sequenced Distance Vector (DSDV)

Use other distance metrics

- Problem: The number of hops is fine for the Internet, but for ad hoc networks alternative quality measures might be better, e.g., energy, congestion, successful transmission probability, interference*, etc.
  - Some of these measures may be multiplicative
  - It„s not clear how to compute some of these measures, e.g. interference, in a MANET.

    *Interference: Use one of the definitions seen in the Chapter on Capacity



Selfishness

- Problem: Why should nodes forward messages for others?

- Two ideas: Recursively encrypt source routing messages by means of public key cryptography, so that each node on the path can only see whether it is the next hop. In addition don‟t use optimal routing paths, instead add a little “noose” to the end of each path.

- This way each intermediate node on the path might worry that it is actually the destination itself, and hence each intermediate node has an incentive to forward messages.

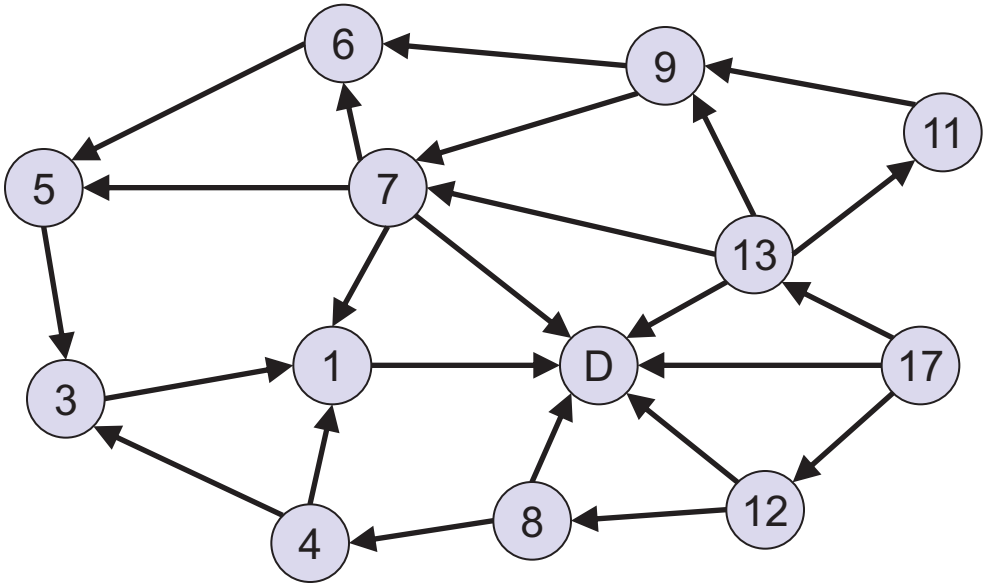
\+ This protocol is indeed incentive-compatible (in contrast to many others which only claim to be incentive-compatible)

– The overhead might be high

– More about forwarding than about routing

Case Study: Link Reversal Routing

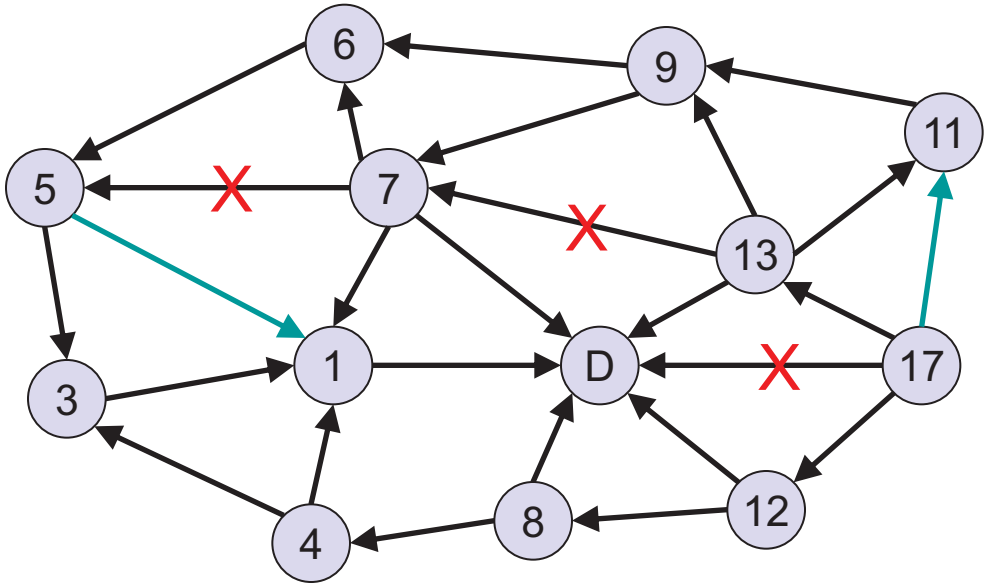
- An interesting proactive routing protocol with low overhead.
- Idea: For each destination, all communication links are directed, such that following the arrows always brings you to the destination.
- Example with only one destination *D*, i.e. sink in a sensor network:



- Note that positive labels can be chosen such that higher labels point to lower labels (destination label *D* = 0).

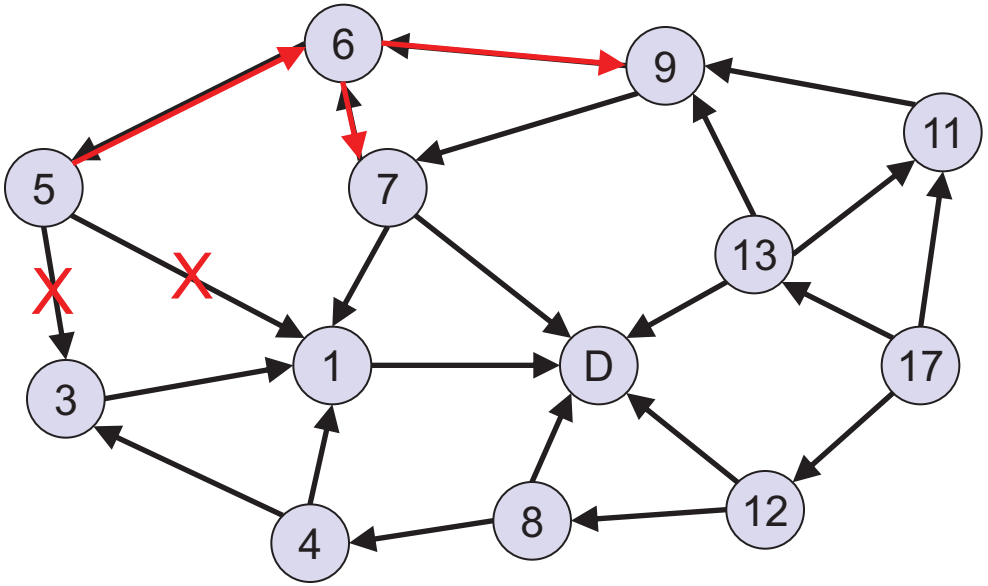
Link Reversal Routing: Mobility

- Links may fail/disappear: if nodes still have outlinks → no problem!
- New links may emerge: just insert them such that there are no loops
  - Use the labels to figure out link direction



Link Reversal Routing: Mobility

- Only problem: Non-destination becomes a sink → reverse all links!
  - Not shown in example: If you reverse all links, then increase label.
  - Recursive progress can be quite tedious… How tedious?!?



Link Reversal Routing: Analysis

- In a ring network with n nodes, a deletion of a single link (close to the sink) makes the algorithm reverse like crazy: Indeed a single link failure may start a reversal process that takes n rounds, and n links reverse themselves n^2 times!

- That‟s why some researchers proposed partial link reversal, where nodes only reverse links that were not reversed before.

- However, it was shown by Busch et al. that in the extreme case also partial link reversal is not efficient, it may in fact even worse be than regular link reversal.

- Still, some protocols (TORA) are based on link reversal.

Case Study: Compact Routing

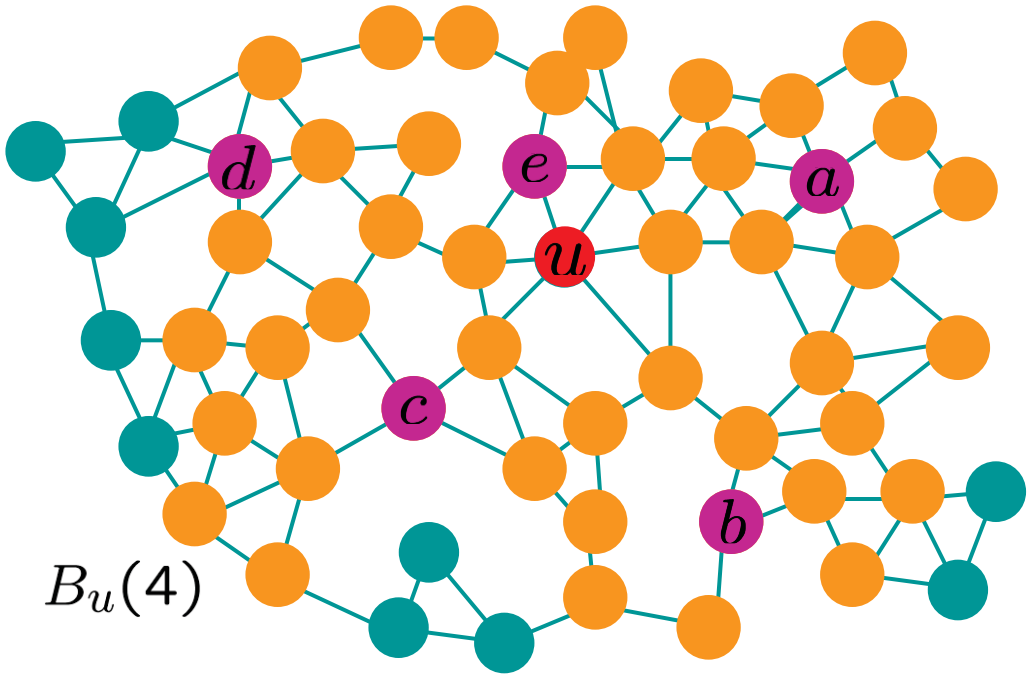
- Generally, there are too many aspects/tradeoffs. Some of these tradeoffs are quite well understood in theory, others not at all.
- A trade-off which is well understood is known as compact routing:

Routing Table Size vs. Routing Stretch

- Remember: The stretch is the ratio of the route length divided by the length of the shortest path, over all routes
- For general (static) graphs it was shown that a stretch strictly below 3 cannot be achieved unless routing tables are at least linear in the number of nodes (in the worst case).
- So what about more realistic graphs (BIG, UBG, etc.)?!?

Reminder: Constant Doubling Metric

- Metric (V,d) with constant doubling dimension
  - Metric: distances between all pairs, non-negative, triangle inequality
  - Ball $B_u(r) := \{ v \mid v \in V \text{ and } \text{dist}(u, v) \leq r \}$
  - Doubling dimension α = log(#balls of radius r/2 to cover ball of radius r)
  - α is constant



What can be achieved?

- Labeled routing scheme → nodes can choose an ID
- (1+ε)-approximation for **Unicast Routing**
- Constant approximations for **Multicast** and **Anycast**

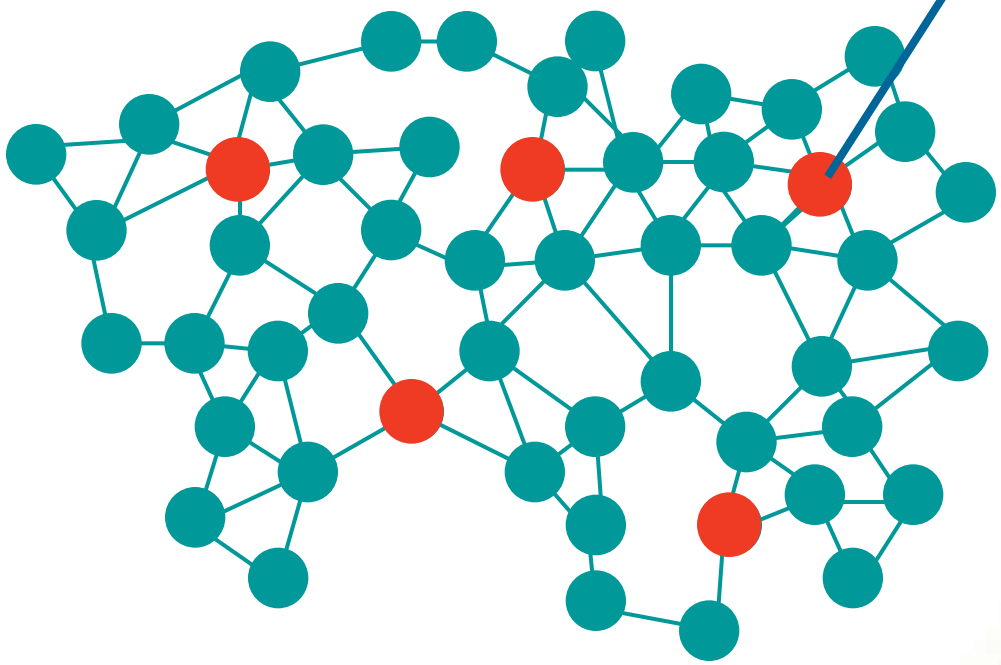
- **Label size: $O(\alpha \log D)$** (bits)
  - D is the diameter of the network

- **Routing table size: $O(1/\varepsilon)^{\alpha} (\log D) (O(\alpha) + \log \Delta)$** (bits)
  - α is the doubling dimension of the graph (small constant, 3-5)
  - Δ is the max degree of any node

- There are so-called name-independent compact routing algorithms which can almost achieve the same bounds as their labeled counterparts by adding a “peer-to-peer” technique.

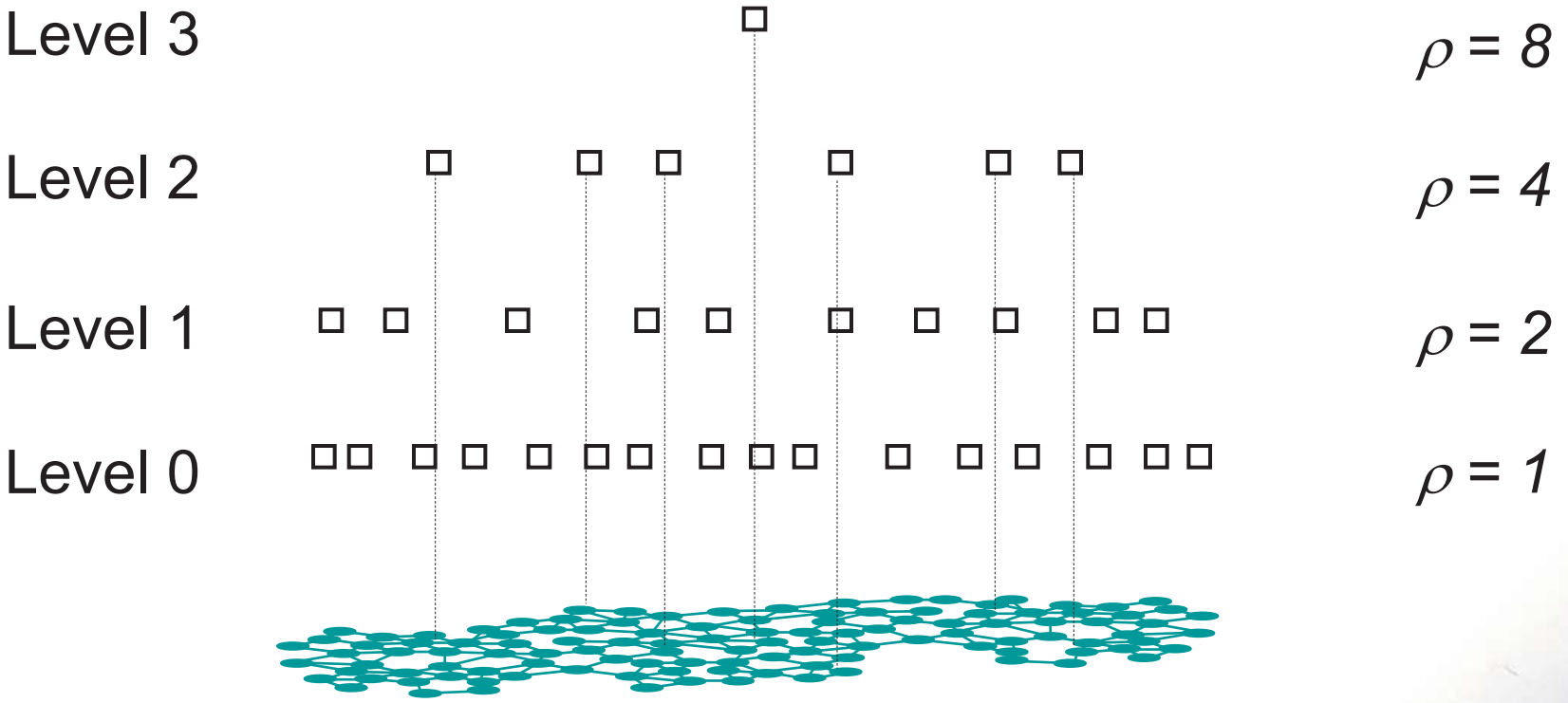
Node Labeling: ρ-net

- Given a graph G=(V,E)
- **U** $\subset$ V is a ρ-net if
  - a) $\forall\ v \in V: \quad \exists\, u \in U : d(u,v) \leq \rho$
  - b) $\forall\, u_1, u_2 \in U : d(u_1, u_2) > \rho$



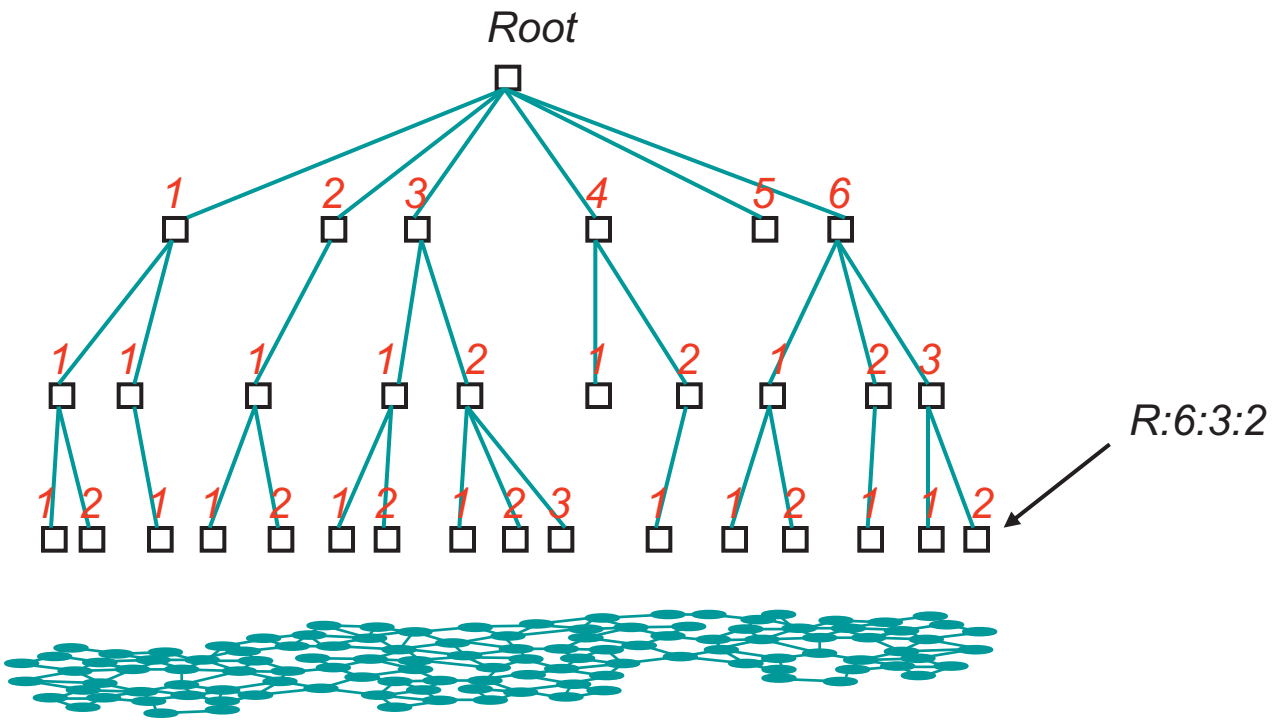
Dominance Net Hierarchy

- Build ρ-nets for $\rho \in \{1, 2, 4, \ldots, 2^{\lceil \log D \rceil}\}$



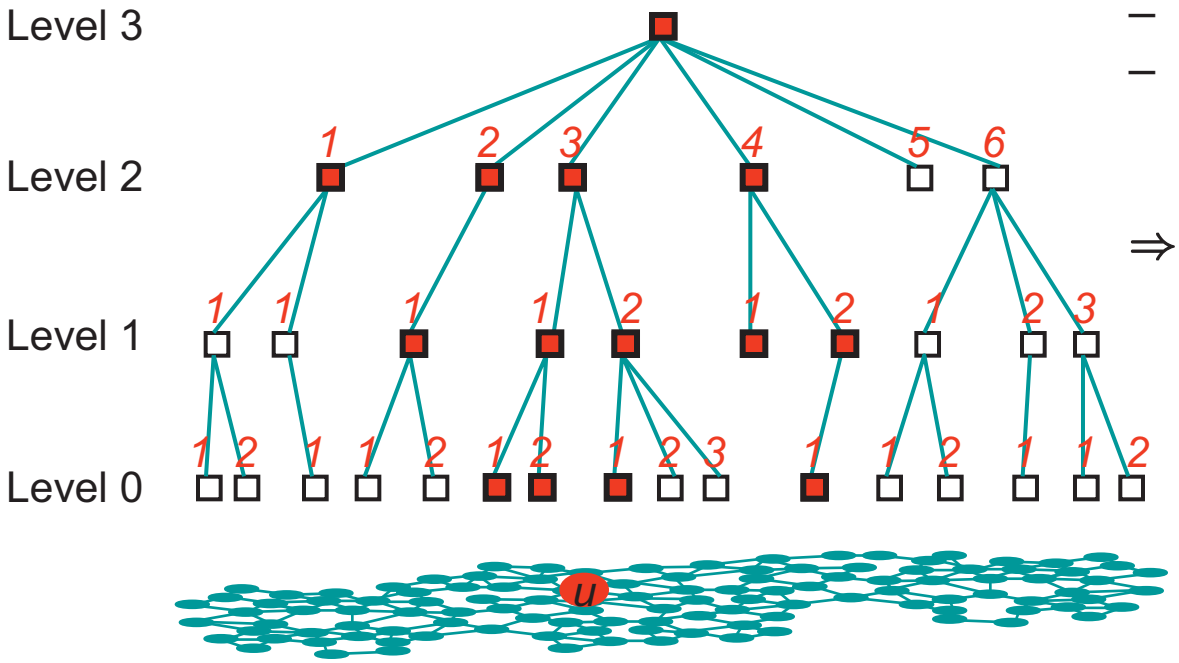
Naming Scheme

- Select parent from next higher level
- Parent enumerates all of its children
  - At most $2^{2\alpha}$ children
  - 2α bits are sufficient for the enumeration
- Name of net-center obtained by concatenation of enum values
  - Name at most $2\alpha \log D$ bits long



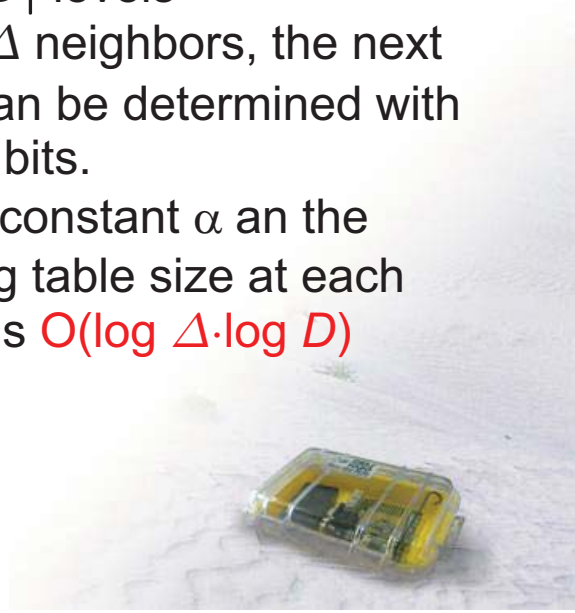
Node Labeling

- Each net-center c of a ρ-net advertises itself to $B_c(2\rho)$
- Any node u stores ID of all net-centers from which it receives advertisements
  - Per level at most $2^{2\alpha}$ net-centers to store
  - If net-center c covers u, then also the parent of c covers u
  - The set of net-centers to store form a **tree**



- Per level at most $2^{2\alpha} \cdot 2\alpha$ bits
- $\lceil \log D \rceil$ levels
- With Δ neighbors, the next hop can be determined with $\log \Delta$ bits.

$\Rightarrow$ For a constant α an the routing table size at each node is $O(\log \Delta \cdot \log D)$



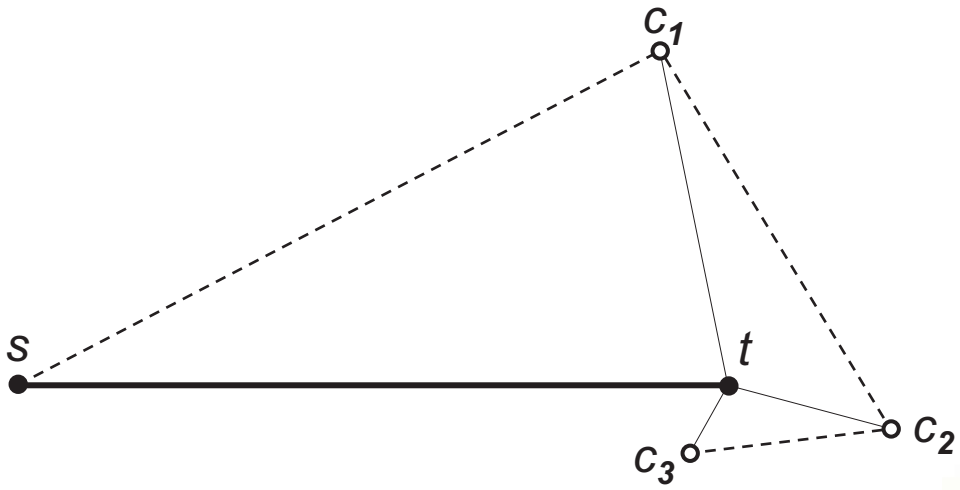
Unicast Routing

Problem: From a sender node *s*, send a message to a target node *t*, given the ID and label L(*t*).

Algorithm: From all net-centers listed in L(t), *s* picks the net-center *c* on the lowest level to which it has routing information and forwards the message towards *c*.

Idea: Once we reach a first net-center of t, we are sure to find a closer net-center on a lower layer.

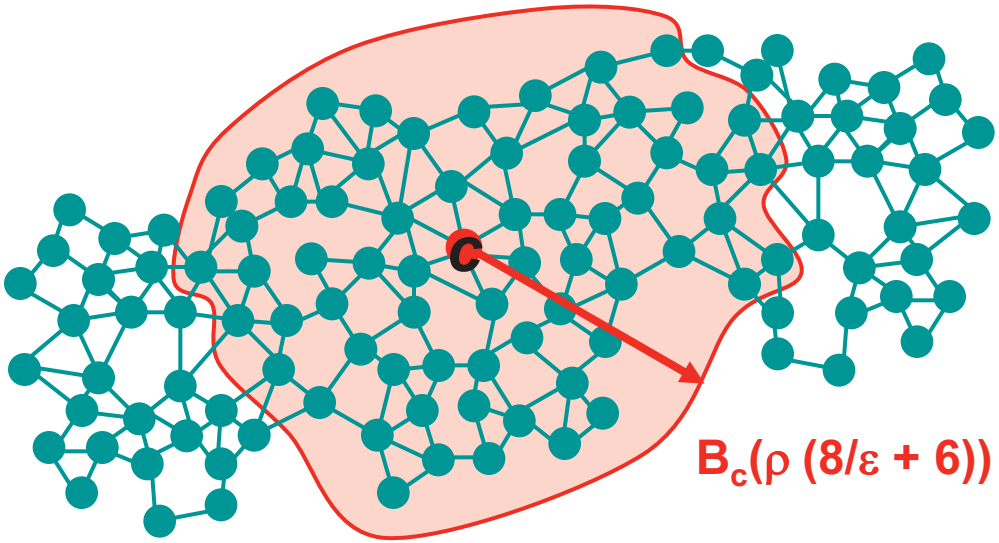
The path to the first net-center causes only little overhead as the net-centers advertise themselves quite far.



A more detailed analysis

- Routing tables to support (1+ε) stretch routing
  Recall: Routing table size of $O(1/\varepsilon)^{\alpha}$ (log D) ($O(\alpha)$ + log Δ) bits

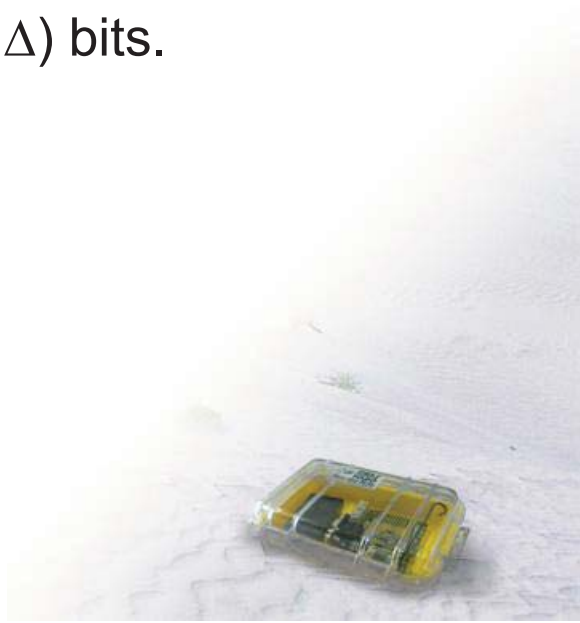
- Every net-center $c \in \rho$-net of the dominance net advertises itself to $B_c(\rho\,(8/\varepsilon + 6))$
- Every node stores direction to reach all advertising net-centers



A more detailed analysis (2)

- Each node needs to store direction for at most $2^{2\alpha}(8/\varepsilon + 6)^{\alpha}$ net-centers per level
- If a node needs to store a routing entry for net-center *c*, then it also needs to store a routing entry for the parent of *c.*
  - Again, the routing table can be stored as a **tree**
- For each net-center, we need to store its enumeration value, and the next-hop information, which takes at most $2\alpha + \log \Delta$ bits
- Total storage cost is $2^{2\alpha}(8/\varepsilon + 6)^{\alpha} \log D\ (2\alpha + \log \Delta)$ bits.

- Similar bounds hold for multicast and anycast.

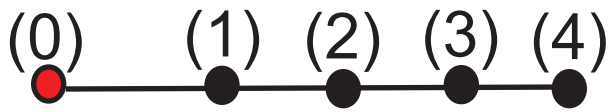


Case Study: Geo-Routing without Geometry

- For many applications, like routing, finding a realization of a UDG is not mandatory
- Virtual coordinates merely as infrastructure for geometric routing

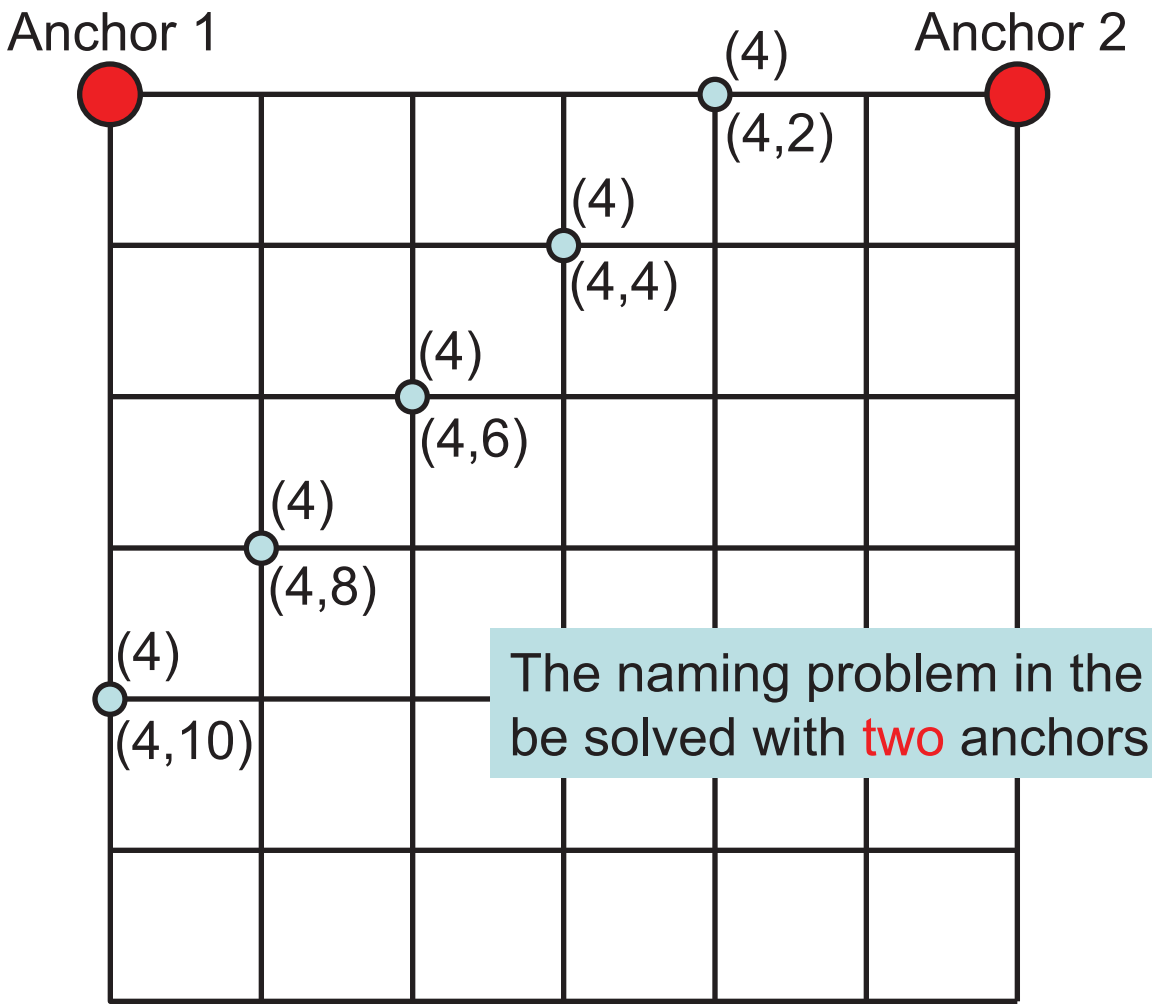
→ Pseudo geometric coordinates:

  - Select some nodes as anchors: $a_1, a_2, \ldots, a_k$
  - Coordinate of each node *u* is its hop-distance to all anchors: $(d(u,a_1), d(u,a_2), \ldots, d(u,a_k))$



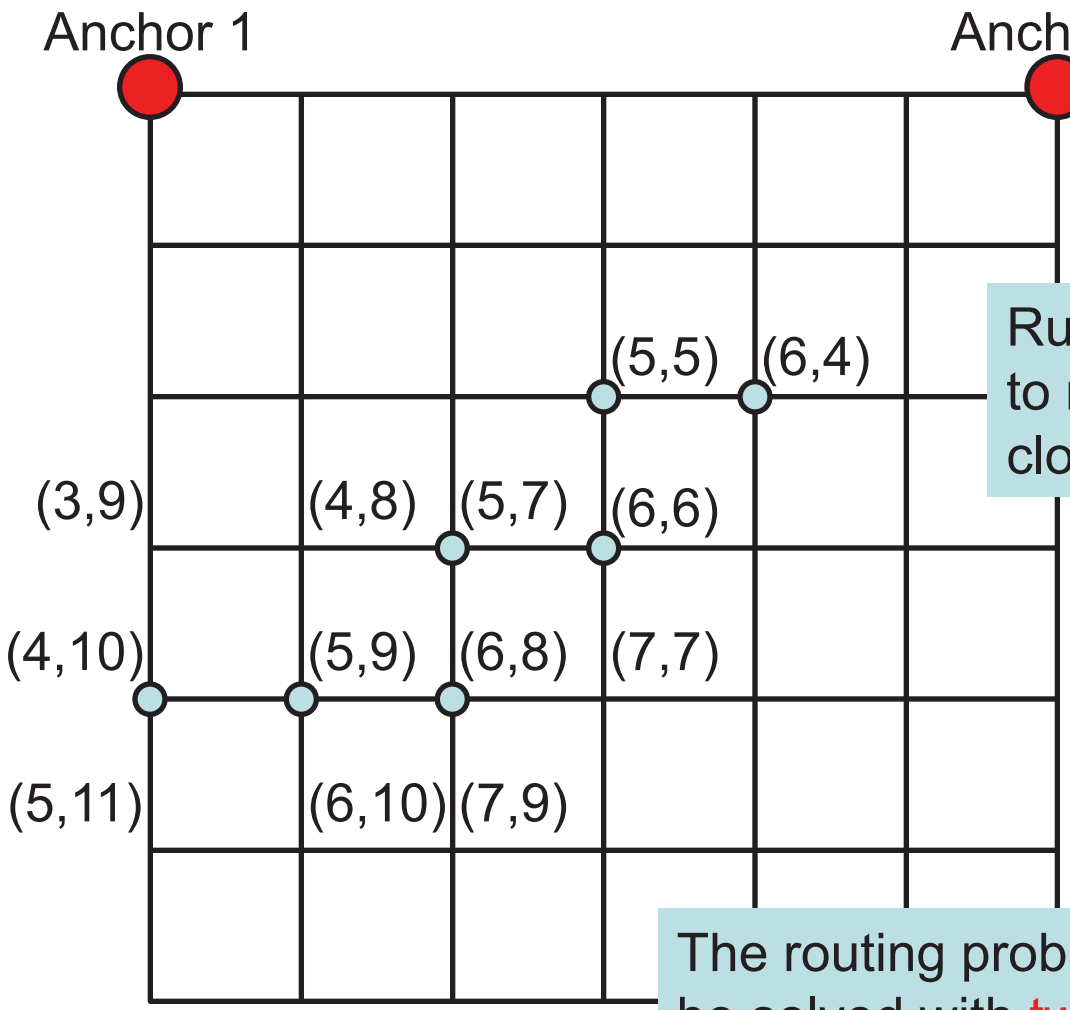
- Requirements:
  - each node uniquely identified: Naming Problem
  - routing based on (pseudo geometric) coordinates possible: Routing Problem

Pseudo-geo-routing on the grid: Naming



The naming problem in the grid can be solved with two anchors.

Pseudo-geo-routing on the grid: Routing

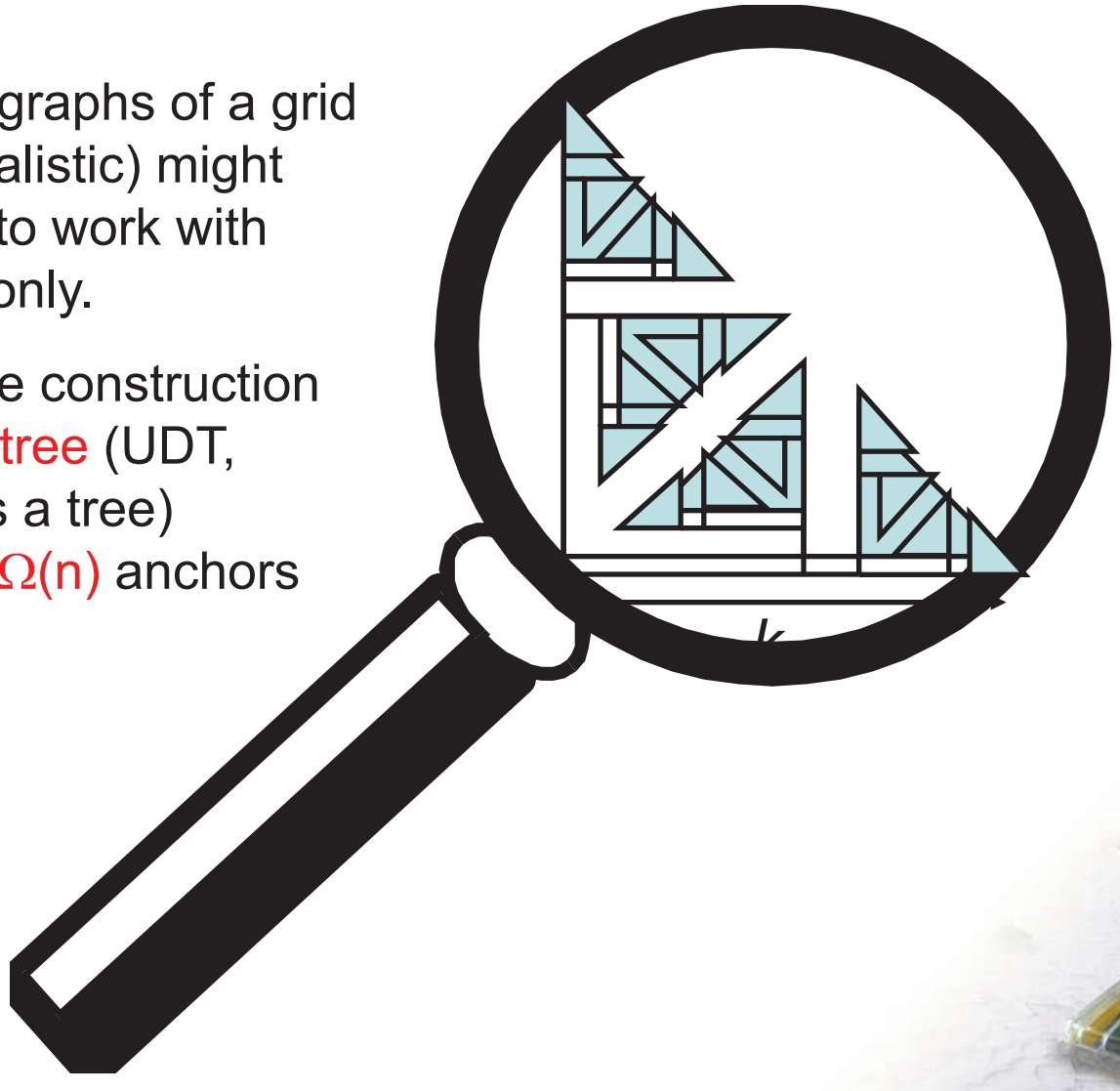


Rule: pass message to neighbor which is closest to destination.

The routing problem in the grid can be solved with two anchors.

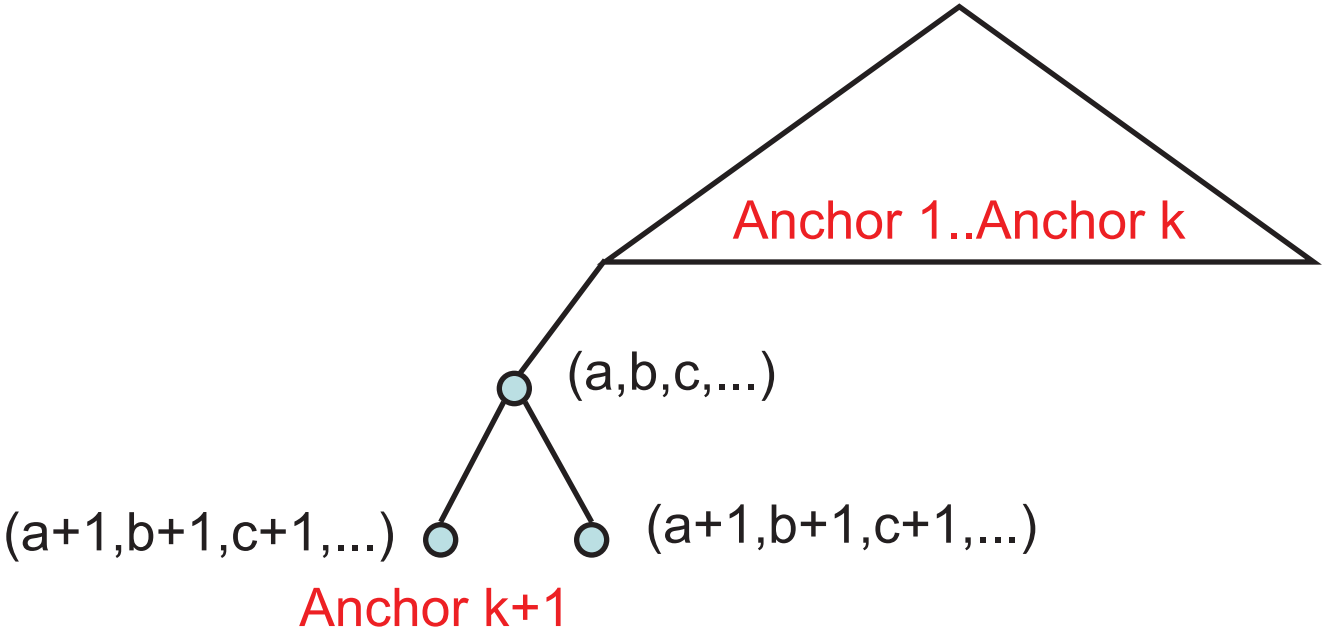
Problem: Even a UDG is not a grid

- But even subgraphs of a grid (which are realistic) might have trouble to work with few anchors only.
- E.g., recursive construction of a unit disk tree (UDT, UDG which is a tree) which needs $\Omega(n)$ anchors



Pseudo-geo-routing in the UDT: Naming

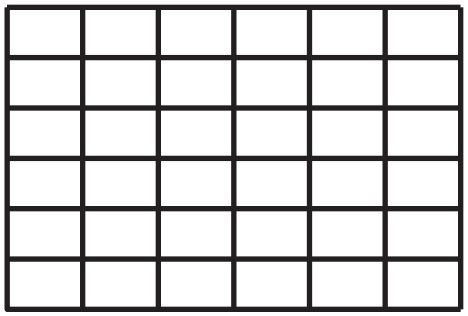
- Leaf-siblings can only be distinguished if one of them is an anchor:

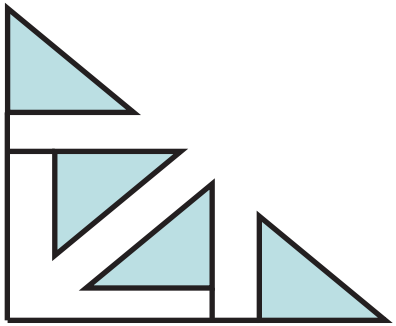


In a unit disk tree with n nodes there are up to $\Theta(n)$ leaf-siblings. That is, we need $\Theta(n)$ anchors.

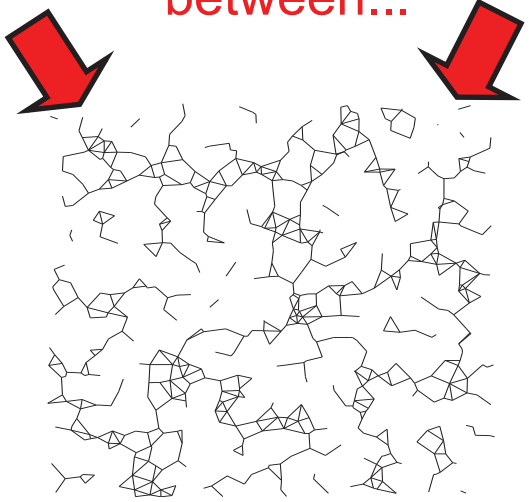
Pseudo-geo-routing in ad hoc networks

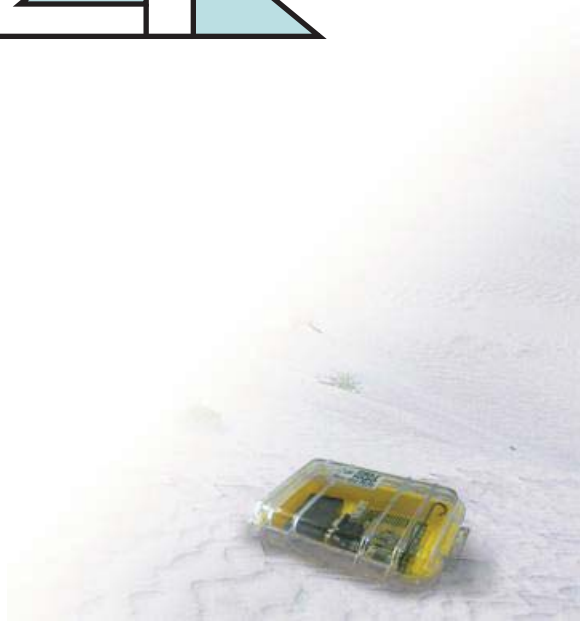
- Naming and routing in grid quite good, in previous UDT example very bad
- Real-world ad hoc networks are very probable neither perfect grids nor naughty unit disk trees





Truth is somewhere in between...





Open Problem

- Apart from compact routing, almost nothing “interesting” is known in the routing domain. From a theoretical point of view, classic ad hoc routing protocols such as AODV or DSR perform very poorly.

- An interesting open question is whether a provably efficient routing algorithm for MANETs (truly dynamic) can be constructed.

- Mind however that one has to carefully argue around too simple cooked-up worst-case examples (e.g. messages may be stuck on “mobile islands”).