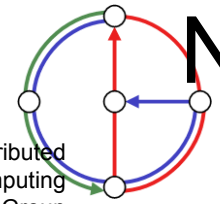
Chapter 5
AD HOC NETWORKS



Mobile Computing

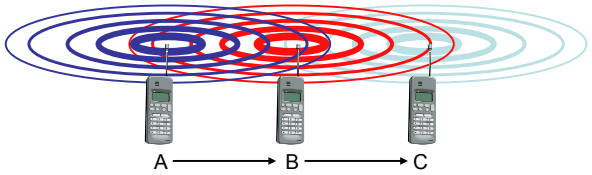
Summer 2004

Overview

- Characteristics
- Routing
- “Classic” routing
- Some basic tricks to improve routing

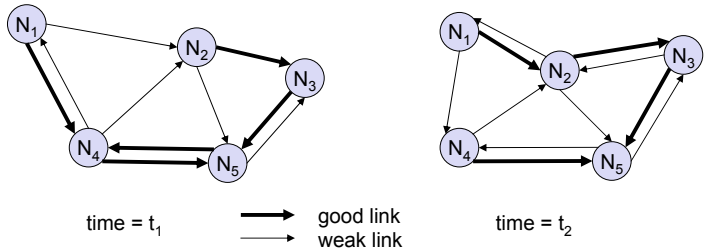
What are ad-hoc networks?

- Sometimes there is no infrastructure
  - remote areas, ad-hoc meetings, disaster areas
  - cost can also be an argument against an infrastructure
- Sometimes not every station can hear every other station
  - Data needs to be forwarded in a “multihop” manner



Mobile (and Multihop) Ad-Hoc Networks (MANET)

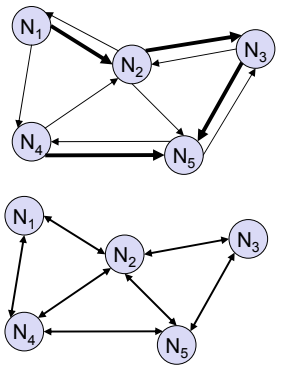
- Nodes move



- First and foremost issue: Routing

An ad-hoc network as a graph

- A node is a mobile station
- All nodes are equal (are they?)
- Iff node v can “hear” node u, the graph has an arc (u,v)
- These arcs can have weights that represent the signal strength
- Close-by nodes have MAC issues such as hidden/exposed terminal problems
- Optional: links are symmetric
- Optional: the graph is Euclidian, i.e., there is a link between two nodes iff the distance d of the nodes is less than D



Routing: Flooding

- What is Routing?
- „Routing is the act of moving information across an internetwork from a source to a destination.“ (CISCO)

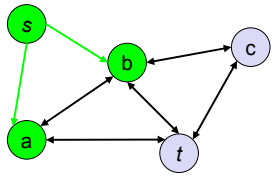
- The simplest form of routing is “flooding”: a source *s* sends the message to all its neighbors; when a node other than destination *t* receives the message the first time it re-sends it to all its neighbors.

+ simple (sequence numbers)

– a node might see the same message more than once. (How often?)

– what if the network is huge but the target *t* sits just next to the source *s?*

- We need a smarter routing algorithm



Classic Routing 1: Link-State Routing Protocols

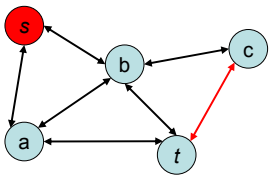
- Link-state routing protocols are a preferred iBGP method (within an autonomous system – think: service provider) in the Internet

- Idea: periodic notification of all nodes about the complete graph
- Routers then forward a message along (for example) the shortest path in the graph

+ message follows shortest path

– every node needs to store whole graph, even links that are not on any path

– every node needs to send and receive messages that describe the whole graph regularly



Classic Routing 2: Distance Vector Routing Protocols

- The predominant method for wired networks
- Idea: each node stores a routing table that has an entry to each destination (destination, distance, neighbor)
- If a router notices a change in its neighborhood or receives an update message from a neighbor, it updates its routing table accordingly and sends an update to all its neighbors

+ message follows shortest path

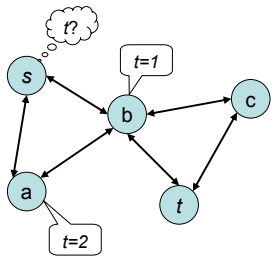
+ only send updates when topology changes

– most topology changes are irrelevant for a given source/destination pair

– every node needs to store a big table

– count-to-infinity problem

Dest	Dir	Dst
a	*a*	1
b	*b*	1
c	*b*	2
t	*b*	2



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 ("Moteran")
- 10 Tricks → 2^{10} routing algorithms
- In reality there are almost that many!
- Q: How good are these routing algorithms?!? Any hard results?
- A: Almost none! Method-of-choice is simulation…
- Perkins: "if you simulate three times, you get three different results"

Trick 1: Radius Growth

- Problem of flooding (and similarly other algorithms): The destination is in two hops 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 3 (really?), …when destination is found, how do we stop?
- Alternative idea: Flood very slowly (nodes wait some time before they forward the message) – when the destination is found a quick flood is initiated that stops the previous flood

\+ Tradeoff time vs. number of messages

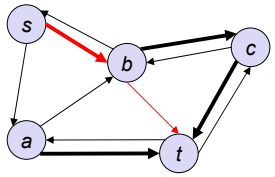
Trick 2: Source Routing

- Problem: nodes have to 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 they need
– Not efficient if mobility/data ratio is high
– Asymmetric Links?

Trick 3: 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.



Trick 4: 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 in contradiction to the source routing philosophy

Trick 5: 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* (*a*) does?

– One can construct examples where this works, but of course also examples where this does not work.

Trick 6: Hierarchy

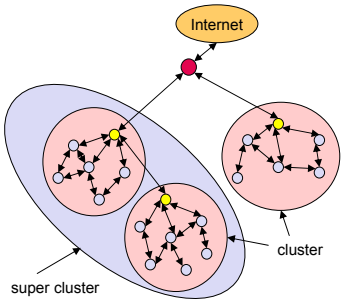
- Problem: Especially proactive algorithms do not scale with the number of nodes. Each node needs to store big tables
- Idea: In the Internet there is a hierarchy of nodes; i.e. all nodes with the same IP prefix are in the same direction. One could do the same trick in ad-hoc networks

\+ Well, if it happens that the ad-hoc nodes with the same numbers are in the same area are together, hierarchical routing is a good idea.
– There are not too many applications where this is the case. Nodes are mobile after all.

Trick 7: Clustering

- Idea: Group the ad-hoc nodes into clusters (if you want 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.



Trick 8: 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?

+ Finally a good trick

Trick 9: 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

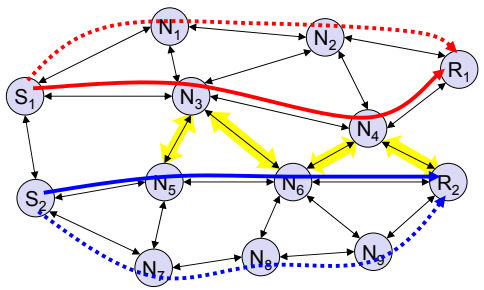
- Implemented in Destination Sequenced Distance Vector (DSDV)

Trick 10: Use other distance metrics

- Problem: The number of hops is fine for the Internet, but for ad-hoc networks other metrics might be better, for example: Energy, Congestion, Successful transmission probability, Interference*, etc.

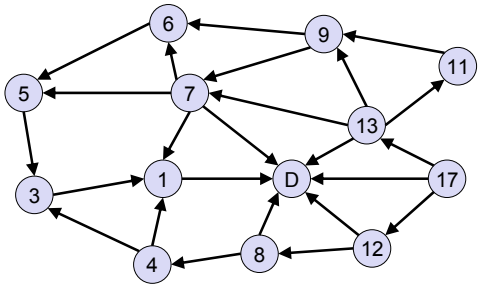
– How do we compute interference in an online manner?

*Interference: a receiving node is also in the receiving area of another transmission.

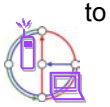


Link Reversal Routing

- 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):

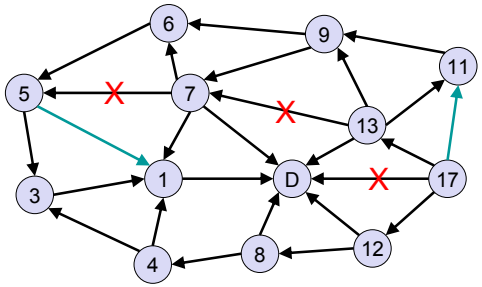


- Note that positive labels can be chosen such that higher labels point to lower labels (and the destination label D = 0).



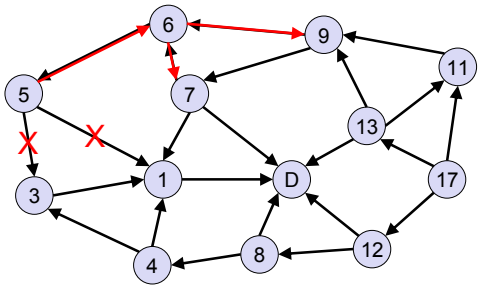
Link Reversal Routing: Mobility

- 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 that out)



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…



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.