Unicast Routing

Ad Hoc networks

(under standardization in the IETF MANET WG)

What is an ad hoc network

- A wireless multi-hop infrastructure-less network whose devices act as source/ destination of messages & as relay for packets generated by a node s and addressed to a node z (iff they are on a s-z route)
- Pros: No need for infrastructure → low cost, enables communication where it is usually not needed or is not viable
- Must be: Self-organizing, self-configuring, self-maintaining

Application scenarios

- Collaboration between users in office environments
- Disaster recovery applications
- Military networks
- Personal Area Networks
- Home Networking
- Wireless Sensor Networks (WSNs)
- Inter-vehicular communication

Features of ad hoc networks

- Highly dynamic networks → device mobility, energy saving sleep/awake modes
- Need for low energy/resource-consuming, simple protocols
- Bandwidth and resource constrained environment
- Traffic:
  - All-pairs in general ad hoc networks, from sensors to sink(s) in sensor networks
  - In many case not high
- Scale: Application dependent
  - 10-100 nodes in traditional ad hoc networks
  - 1000-10000 in sensor networks

Features of highly resource constrained ad hoc networks (WSNs)

- Highly dynamic networks → due to device mobility (only in some specific applications), to the fact the active node set changes in time for sake of energy saving (always to be considered)
- Need to design low energy/resource-consuming, simple protocols → very critical, energy consumption a real bottleneck
- Traffic from sensors to sink(s)
- Scalability is a major issue
- Code must be simple (small storage capability, very simple, inexpensive, resource constrained devices)
- First solutions we will see for traditional ad hoc networks do not scale to high numbers and are not energy-saving

Routing-Traditional approach (from Reti 1)

- Intra-AS routing in the Internet
  - Link State Approaches

    (info on the topology graph gathered at nodes which run shortest path algorithms- Dijkstra- to decide the routes to the different destinations –e.g. OSPF routing protocol)
  - Distance Vector approaches (e.g. RIP)

Bellman-Ford

Given a graph G=(N,A) and a node *s* find the shortest path from *s* to every node in N.

A shortest walk from *s* to i subject to the constraint that the walk contains at most h arcs and goes through node *s* only once, is denoted shortest(<=h) walk and its length is D^h_i.

<u>Bellman-Ford rule:</u>

Initiatilization $D^h_s=0$, for all h; $w_{i,k}$ = infinity if (i,k) NOT in A; $w_{k,k}=0$; D^0_i=infinity for all i!=s.

<u>Iteration:</u>

$$D^{h+1}_i=\min_k [w_{i,k} + D^h_k]$$

Assumption: non negative cycles **(this is the case in a network!!)**

The Bellman-Ford algorithm first finds the one-arc shortest walk lengths, then the two-arc shortest walk length, then the three-arc…etc. →distributed version used for routing

Bellman-Ford

$$D^{h+1}_i = \min_k [w_{i,k} + D^h_k]$$

Can be computed locally.
What do I need?

For each neighbor *k*, I need to know
-the cost of the link to it (known info)
-The cost of the best route from the neighbor *k* to the destination (←this is an info that each of my neighbor has to send to me via messages)

In the real world: I need to know the best routes among each pair of nodes → we apply distributed Bellman Ford to get the best route for each of the possible destinations

Distance Vector Routing Algorithm -Distributed Bellman Ford

iterative:

- continues until no nodes exchange info.
- *self-terminating*: no "signal" to stop

asynchronous:

- nodes need *not* exchange info/iterate in lock step!

Distributed, based on local info:

- each node communicates *only* with directly-attached neighbors

Distance Table data structure

each node has its own

- row for each possible destination
- column for each directly-attached neighbor to node
- example: in node X, for dest. Y via neighbor Z:

Cost associated to the (X,Z) link

$$D^X(Y,Z) = \text{distance } \textit{from} \text{ X } \textit{to} \text{ Y, } \textit{via} \text{ Z as next hop}$$

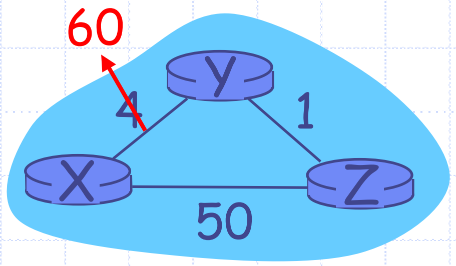
$$= c(X,Z) + \min_w \{D^Z(Y,w)\}$$

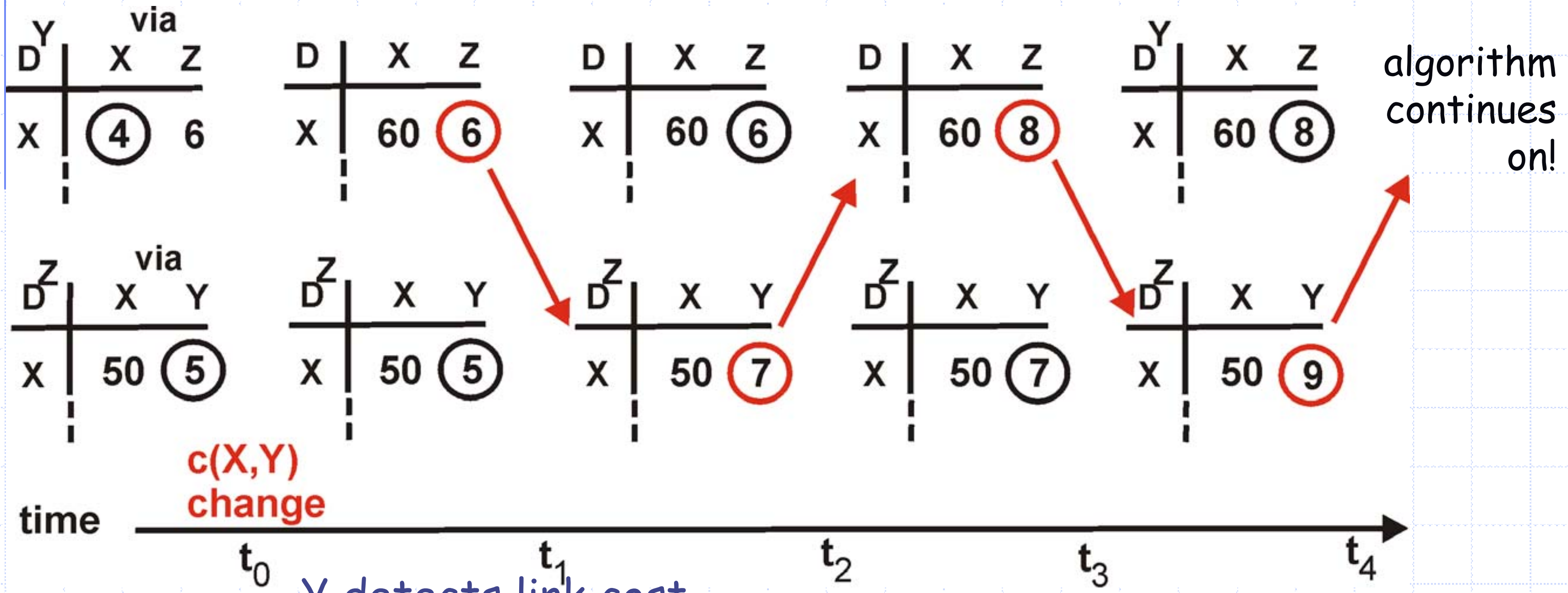
Info maintained at Z. Min must be communicated

Distance Vector: link cost changes

Link cost changes:

- good news travels fast
- ***bad news travels slow*** - "count to infinity" problem!





Y detects link cost Increase but think can Reach X through Z at a total cost of 6 (wrong!!) → **The path is Y-Z-Y-X**

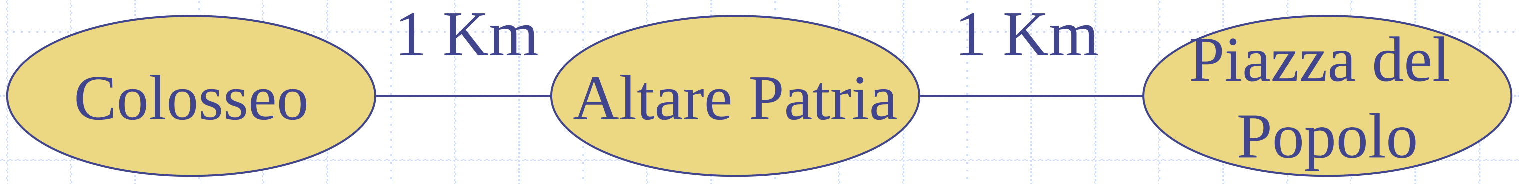
Count-to-infinity –an everyday life example

Which is the problem here?

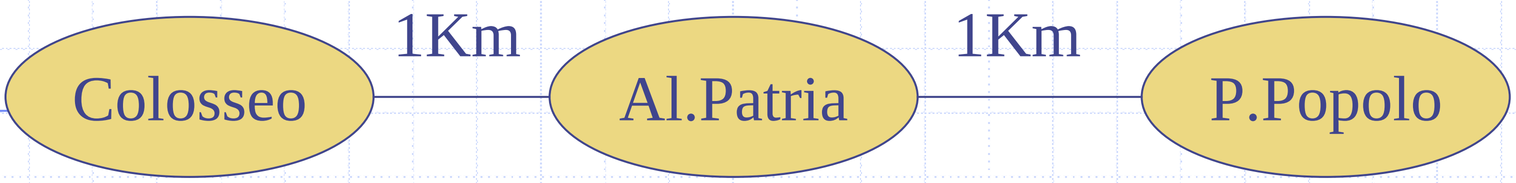
the info exchanged by the protocol!! 'the best route to X I have has the following cost…' (no additional info on the route)

A Roman example…

-assumption: there is only one route going from Colosseo to Altare della Patria: Via dei Fori Imperiali. Let us now consider a network, whose nodes are Colosseo., Altare della Patria, Piazza del Popolo



Count-to-infinity –everyday life example (2/2)



The Colosseo. and Alt. Patria nodes exchange the following info

- Colosseo says 'the shortest route from me to P. Popolo is 2 Km'
- Alt. Patria says 'the shortest path from me to P. Popolo is 1Km'

Based on this exchange from Colosseo you go to Al. Patria, and from there to Piazza del Popolo OK **<u>Now</u> due to the big dig they close Via del Corso (Al. Patria—P.Popolo)**

- Al. Patria thinks 'I have to find another route from me to P.Popolo. Look there is <u>a</u> route from Colosseo to P.Popolo that takes 2Km, I can be at Colosseo in 1Km → I have found a 3Km route from me to P.Popolo!!' Communicates the new cost to Colosseo that updates 'OK I can go to P.Popolo via Al. Patria in 4Km'

VERY WRONG!! Why is it so? I didn't know that the route from Colosseo to P.Popolo was going through Via del Corso from Al.Patria to P.Popolo (which is closed)!!

Routing in ad hoc networks- Goals

- Minimal control overhead
- Minimal processing overhead
- Multi-hop path routing capability
- Dynamic topology maintenance
- No loops
- Self-starting

2 Primary Approaches

- Proactive
  - Based on traditional distance-vector and link-state protocols
  - Each node maintains route to each other network node
  - Periodic and/or event triggered routing update exchange
  - Higher overhead in most scenarios
  - Longer route convergence time
  - Examples: DSDV, TBRPF, OLSR

Highly Dynamic Destination-Sequenced Distance-Vector (DSDV) Routing

- Proactive, distance vector approach (uses distributed asynchronous Bellman Ford). Updates on routes costs transmitted periodically or when significant new information is available.
- Difference wrt Bellman Ford: tries to avoid loops (approaches such as Poison reverse non effective in broadcast channels, we seek solutions which are simple and fully distributed)
- Metrics: fresh routes better than stale routes, number of hops used to select among the fresh routes
- How to identify fresh routes? By means of sequence numbers identifying the freshness of the communicated information. When changes occur, the sequence number increase.

Highly Dynamic Destination-Sequenced Distance-Vector (DSDV) Routing

- Periodically destination nodes transmit updates with a new sequence number (and such updates are propagated by the other nodes). Updates contain information on the costs to achieve the different destinations and the freshness of the delivered information
- Data broadcast include multiple entries each with:
  - Destination address
  - Number of hops required to reach the destination
  - Sequence number of the information received regarding that destination as originally stamped by the destination
- In the header the data broadcast also include:
  - Address (HW address/Net address) of the sender of the message
  - Sequence number created by the transmitter
- Two types of updates (full dump or incremental-only changes- to decrease bandwidth consumption.

Highly Dynamic Destination-Sequenced Distance-Vector (DSDV) Routing

- How can the costs be modified? Cost=number of hops, target: using fresh routes as short as possible → a link cost changes from 1 to inf and from inf to 1
- How do we detect that a link is 'broken'? At layer 2 (no hello messages received for some time, or attempts to retransmit a frame exceeds the MAC protocol threshold) or at layer 3 (do not receive periodic updates by a neighbor)
- Link cost increase (1→ inf):
  - The nodes incident to that link (A,B) discover it (see above)
  - Routes going through that link get assigned an inf cost in nodes A and B routing tables
  - A new sequence number is generated by the mobile node. Mobile nodes different from the destination use odd SN, the destination even SN.
  - Updates with routes with infinite cost are immediately transmitted by nodes
- Link cost decrease (inf→1):
  - Immediately transmits updates

Highly Dynamic Destination-Sequenced Distance-Vector (DSDV) Routing

- When a node receives updates it sees if costs to reach the different destinations can be improved:
  - routes with more recent sequence numbers to a given destination are used
  - if more routes available with the same SN the shortest is used
- Newly recorded routes are scheduled for immediate advertisement (inf→ finite value)
- Routes with improved metric are scheduled for advertisement at a time which depends on the estimated average settling time for routes to that particular destination (based on previous history)
- As soon as a route cost changes the node may delay informing its neighbors but immediately starts using the new information for its forwarding

Highly Dynamic Destination-Sequenced Distance-Vector (DSDV) Routing-Correctness

- Assuming routing tables are stable and a change occurs
  - let G(x) denotes the routes graph from the sources to x BEFORE the change (assume no loop)
  - change occurs at *i* when 1) the link from *i* to its parent p(*i*) in G(x) breaks → *i* sets to inf that route (no loop can occur) 2) node *i* receives from one of its neighbors k a route to x with sequence number SN^x_k and metric m which is selected to replace the current metric *i* has to reach x (this occurs only if SN^x_k greater than the previous SN I had stored Sn^x_i or if the two SN are equal but the new route has a lower hop cost → in the first case if selecting k leads to a loop then SN^x_k <= Sn^x_i which is a contradiction, in the second case comes from the observation reduction in the costs do not bring to loops).

Why traditional approaches have limits?

- Proactive protocols are costly in terms of overhead (the bandwidth and energy are critical resources)
- The cost of maintaining routes updated may not make sense in an environment in which
  - Medium-high mobility
  - Medium-high dynamicity (awake/asleep states)

  Often make the opt. Route change (requiring updates) while
  - Traffic is generally low (so the cost of maintaining always updated routes is not balanced by their use)

  If this is the scenario what can we do?

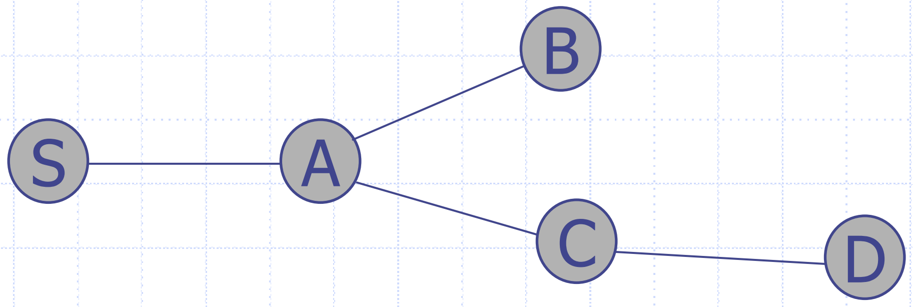
2 Primary Approaches (cont.)

- Reactive (on-demand)
  - Source build routes on-demand by “flooding”
  - Maintain only active routes
  - Route discovery cycle
  - Typically, less control overhead, better scaling properties
  - Drawback: route acquisition latency
  - Example: AODV, DSR

Ad hoc On-Demand Distance Vector (AODV) Routing

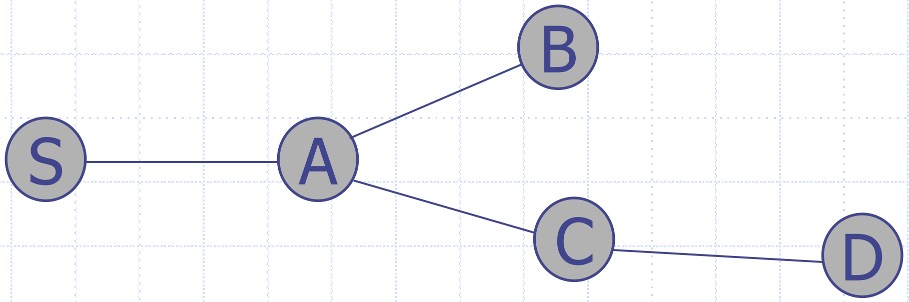
- Reactive (nodes that do not lie on active paths neither maintain any routing information nor participate in any periodic routing table exchange; a node does not have to discover/maintain a route to a destination till it is on a path to it or has to send messages to it)
- *Route discovery cycle* used for route finding
- Maintenance of *active* routes
- Sequence numbers used for loop prevention and as route freshness criteria
- Descendant of DSDV (standard distance vector approach mapped to ad hoc networks), in AODV no periodic updates but pure on-demand operation.
- Provides unicast and multicast communication

AODV: Route Discovery



1. Node *S* needs a route to *D AND does not have routing info for it in its table*

AODV: Route Discovery



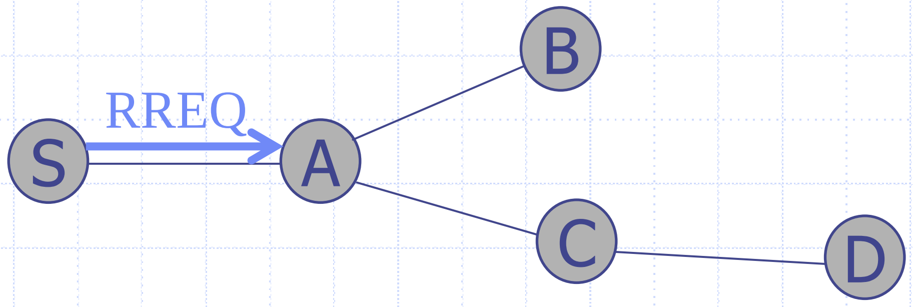
1. Node *S* needs a route to *D*
2. Creates a Route Request (RREQ)

   Enters *D*'s IP addr, seq#,

   S's IP addr, seq#

   hopcount (=0), broadcast ID

AODV: Route Discovery



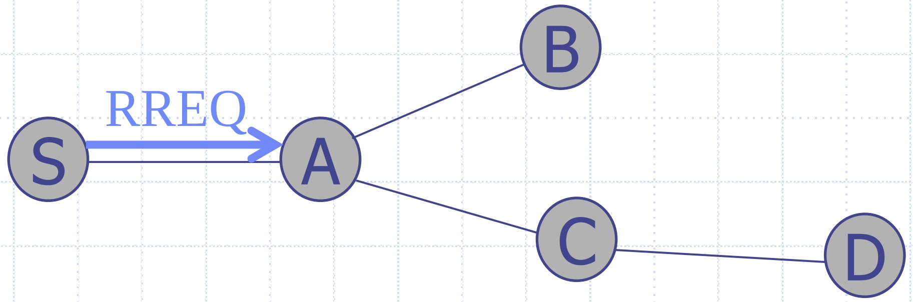
1. Node *S* needs a route to *D*
2. Creates a Route Request (RREQ)

   Enters *D*'s IP addr, seq#,

   S's IP addr, seq#

   hopcount (=0), broadcast ID
3. Node *S* broadcasts RREQ to neighbors

AODV: Route Discovery

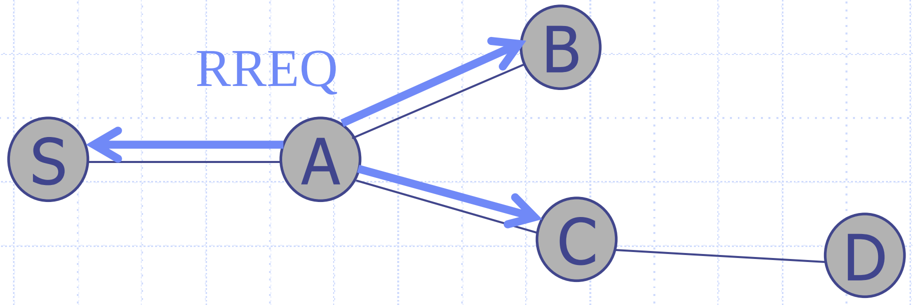


4. Node *A* receives RREQ
   - Makes reverse route entry for *S*

     dest=*S*, nexthop=*S*, hopcnt=1,expiration time for reverse path

   Source node SN,D,broadcastID also maintained
   - It has no route to *D*, so it rebroadcasts RREQ (hopcount increased)
   - If it has already received that request (same source and broadcast ID) it discards the RREQ
   - if it knows a valid path to D it will send back a reply to the source

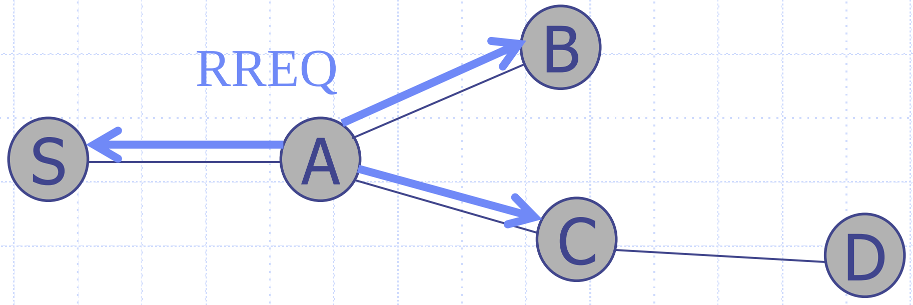
AODV: Route Discovery



4. Node *A* receives RREQ
   - Makes reverse route entry for *S*

     dest=*S*, nexthop=*S*, hopcnt=1
   - It has no route to *D*, so it rebroadcasts RREQ

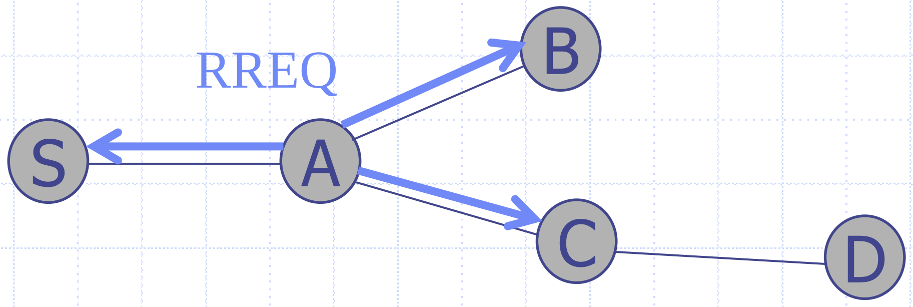
AODV: Route Discovery



5. Node *C* receives RREQ
   - Makes reverse route entry for *S*

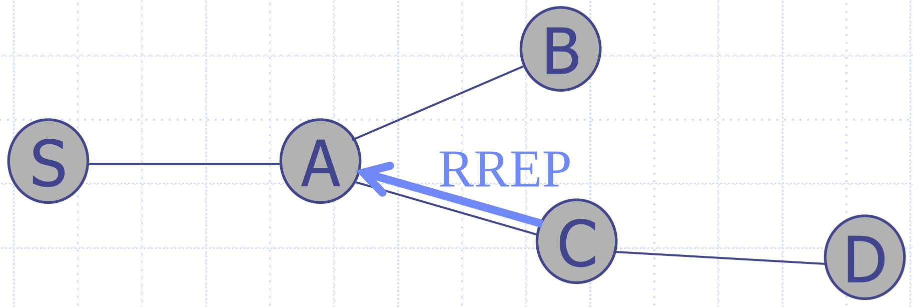
     dest=*S*, nexthop=*A*, hopcnt=2
   - It has a route to *D*, and the seq# for route to *D* is >= *D*'s seq# in RREQ

AODV: Route Discovery



5. Node *C* receives RREQ (cont.)
   - *C* creates a Route Reply (RREP)
     Enters *D*'s IP addr, seq#
     S's IP addr, hopcount to *D* (= 1), lifetime of the forward route
   - Unicasts RREP to *A*

AODV: Route Discovery

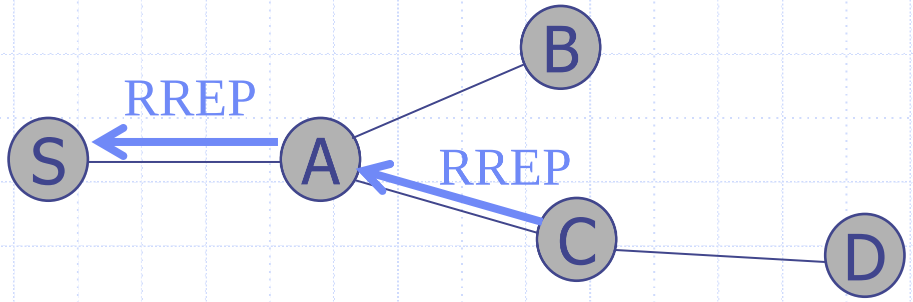


5. Node *C* receives RREQ (cont.)
   - *C* creates a Route Reply (RREP)

     Enters *D*'s IP addr, seq#

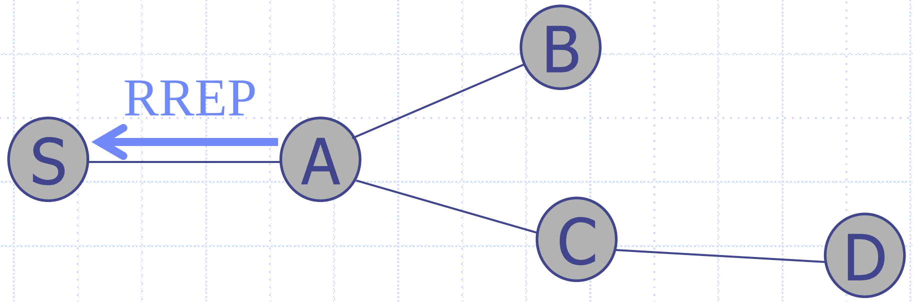
     S's IP addr, hopcount to *D* (= 1)....
   - Unicasts RREP to *A*

AODV: Route Discovery



6. Node *A* receives RREP
   - Makes forward route entry to *D*
     dest = *D*, nexthop = *C*, hopcount = 2
   - Unicasts RREP to *S*

AODV: Route Discovery



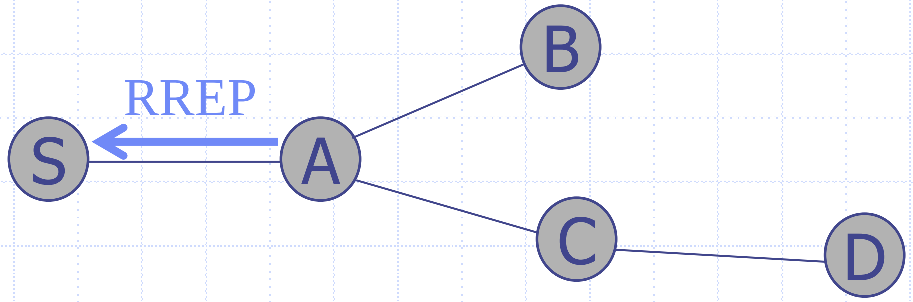
7. Node *S* receives RREP
   - Makes forward route entry to *D*

     dest = *D*, nexthop = *A*, hopcount = 3

Also the latest SN of the destination is updated when receiving the RREP

Nodes not along the path determined by the RREP will timeout after ACTIVE_ROUTE_TIMEOUT (3000ms) and will delete the reverse pointer

AODV: Route Discovery



Se è la destinazione
A rispondere con un
RREP genera un nuovo
SN

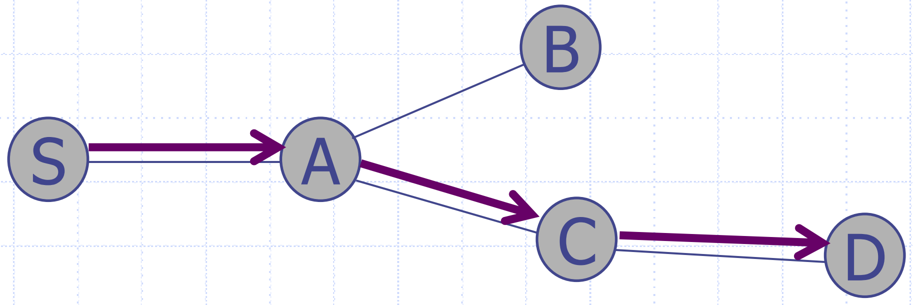
7. Node *S* receives RREP
   - Makes forward route entry to *D*

     dest = *D*, nexthop = *A*, hopcount = 3

Also the latest SN of the destination is updated when receiving the RREP

Nodes not along the path determined by the RREP will timeout after ACTIVE_ROUTE_TIMEOUT (3000ms) and will delete the reverse pointer

AODV: Route Discovery



7. Node *S* receives RREP
   - Makes forward route entry to *D*
     dest = *D*, nexthop = *A*, hopcount = 3
   - Sends data packets on route to *D*

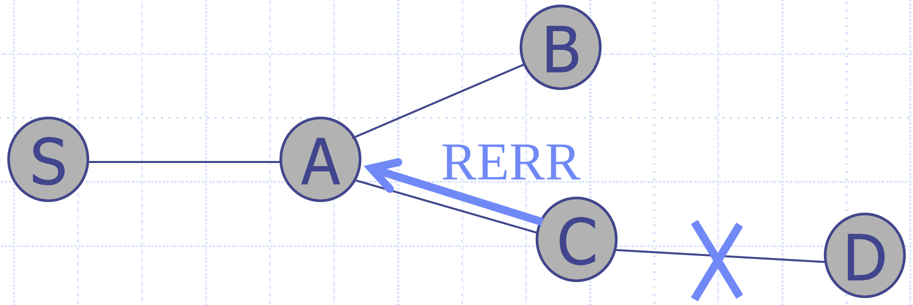
What if….

- A node receives further RREPs for the same request? (e.g. more neighbors of a node had paths to the destination in cache)?
  - upon reception of another RREP the node updates its routing information and propagates the RREP only if the RREP contains either a greater destination SN, or the same destination SN with a smaller hopcount

Other info maintained

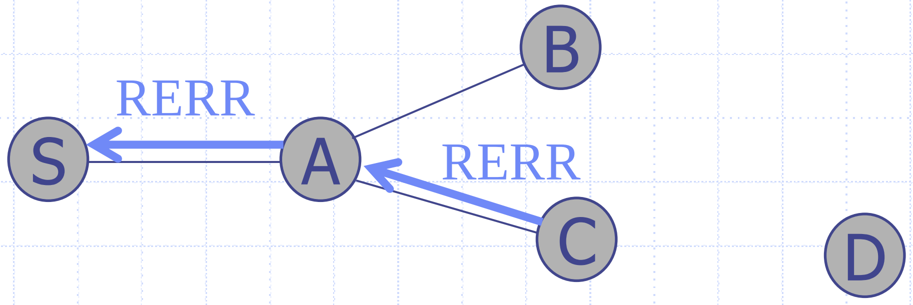
- Each node maintains the list of active neighbors, neighbors sending to a given destination through it
  - useful for route maintenance
- Routing table entries: dest,next hop, hopcount, dest SN, active neighbors for this route, expiration time for route table entry (updates each time the route is used for transmitting data → routes entries are maintained if the route is active)

AODV: Route Maintenance



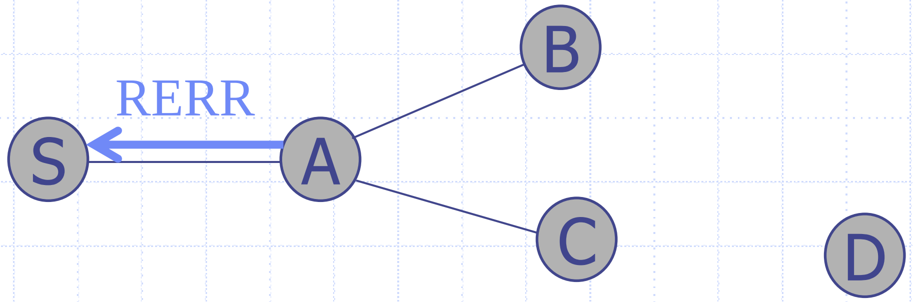
1. Link between *C* and *D* breaks
2. Node *C* invalidates route to *D* in route table
3. Node *C* creates Route Error (RERR) message
   - Lists all destinations which are now unreachable
   - Sends to upstream neighbors
   - new sequence number

AODV: Route Maintenance



4. Node *A* receives RERR
   - Checks whether *C* is its next hop on route to *D*
   - Deletes route to *D*
   - Forwards RERR to *S*

AODV: Route Maintenance



5. Node *S* receives RERR
   - Checks whether *A* is its next hop on route to *D*
   - Deletes route to *D*
   - Rediscovers route if still needed

with a new sequence number

AODV: Optimizations

- Expanding Ring Search
  - Prevents flooding of network during route discovery
  - Control Time To Live (TTL) of RREQ to search incrementally larger areas of network
  - Advantage: Less overhead when successful
  - Disadvantage: Longer delay if route not found immediately

AODV: Optimizations (cont.)

- Local Repair
  - Repair breaks in active routes locally instead of notifying source
  - Use small TTL because destination probably hasn’t moved far
  - If first repair attempt is unsuccessful, send RERR to source
  - Advantage: repair links with less overhead, delay and packet loss
  - Disadvantage: longer delay and greater packet loss when unsuccessful

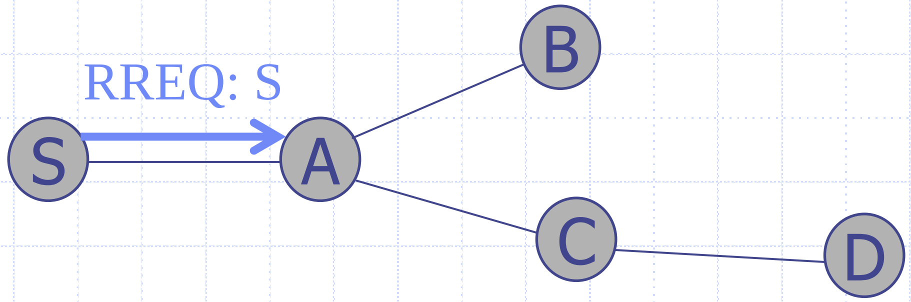
AODV: Summary

- Reactive/on-demand
- Sequence numbers used for route freshness and loop prevention
- Route discovery cycle
- Maintain only active routes
- Optimizations can be used to reduce overhead and increase scalability

Dynamic Source Routing (DSR)

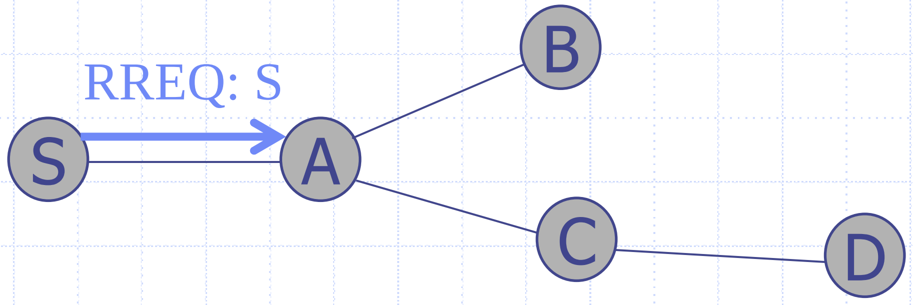
- Reactive
- *Route discovery cycle* used for route finding
- Maintenance of *active* routes
- Utilizes *source routing*

DSR: Route Discovery



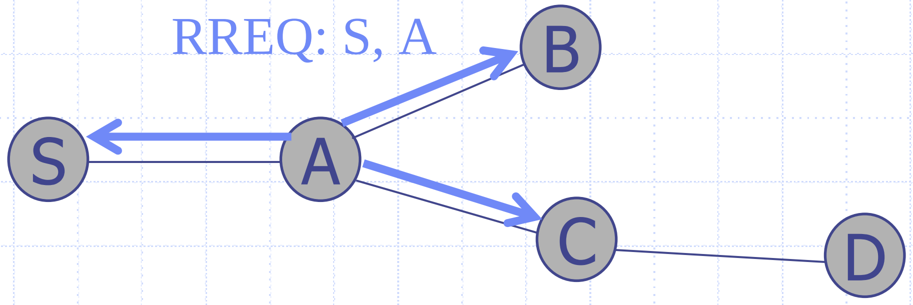
1. Node *S* needs a route to *D*
2. Broadcasts RREQ packet
    1. RREQ identifies the target of the route discovery, contains a route record in which the traversed route is accumulated, contains a pair <initiator, request id> uniquely identifying the request

DSR: Route Discovery



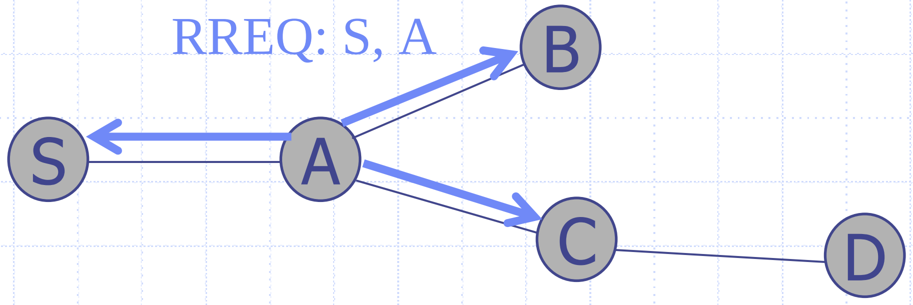
1. Node *S* needs a route to *D*
2. Broadcasts RREQ packet
3. Node *A* receives packet, has no route to *D AND is NOT D*
   - Rebroadcasts packet after adding its address to source route

DSR: Route Discovery



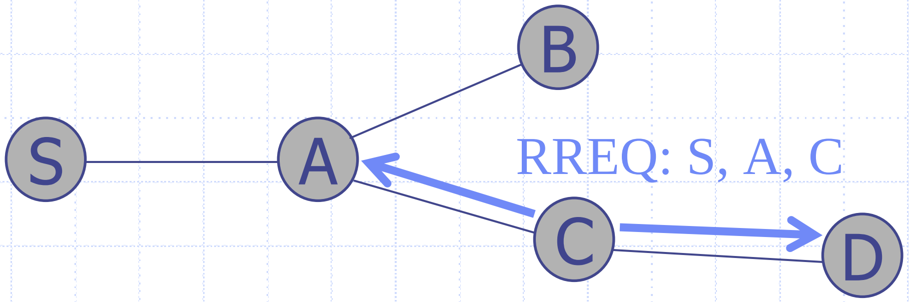
1. Node *S* needs a route to *D*
2. Broadcasts RREQ packet
3. Node *A* receives packet, has no route to *D*
    - Rebroadcasts packet after adding its address to source route

DSR: Route Discovery



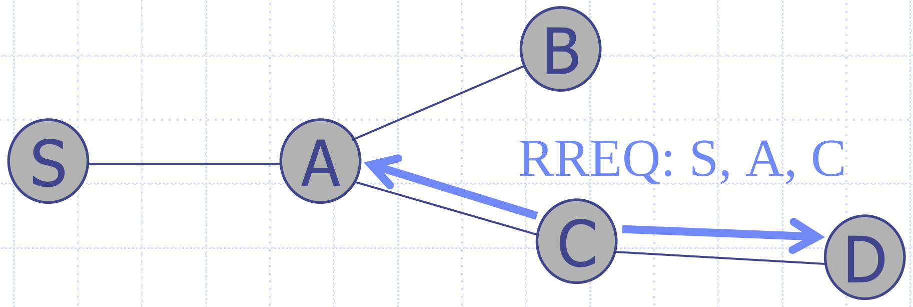
4. Node *C* receives RREQ, has no route to *D*
   - Rebroadcasts packet after adding its address to source route

DSR: Route Discovery



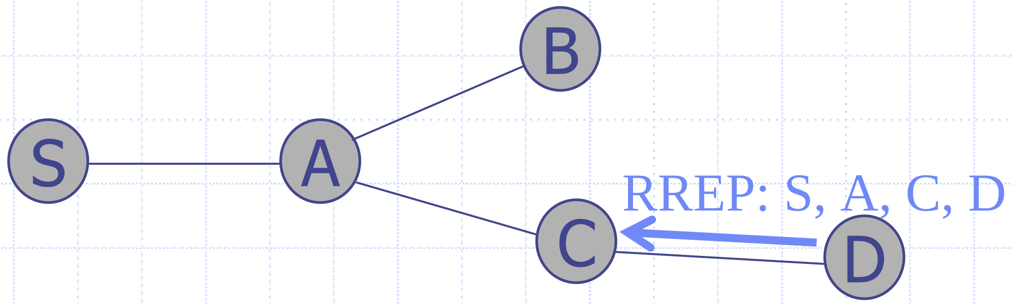
4. Node *C* receives RREQ, has no route to *D*
    - Rebroadcasts packet after adding its address to source route

DSR: Route Discovery



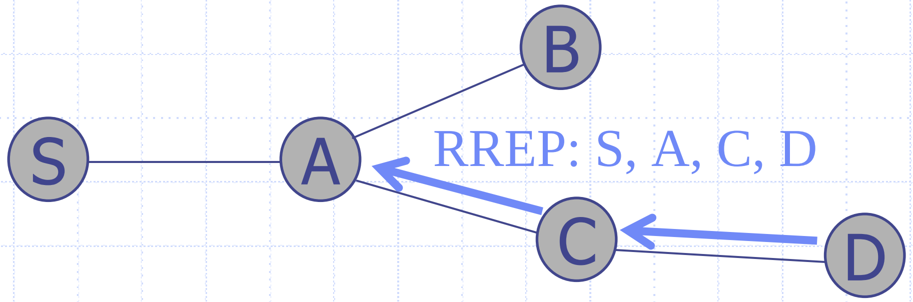
4. Node *C* receives RREQ, has no route to *D*
   - Rebroadcasts packet after adding its address to source route
5. Node *D* receives RREQ, unicasts RREP to *C*
   - Puts source route accumulated in RREQ into RREP

DSR: Route Discovery



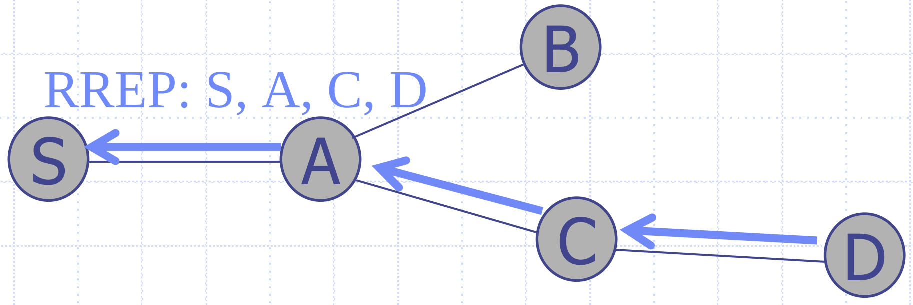
4. Node *C* receives RREQ, has no route to *D*
   - Rebroadcasts packet after adding its address to source route
5. Node *D* receives RREQ, unicasts RREP to *C*
   - Puts source route accumulated in RREQ into RREP

DSR: Route Discovery



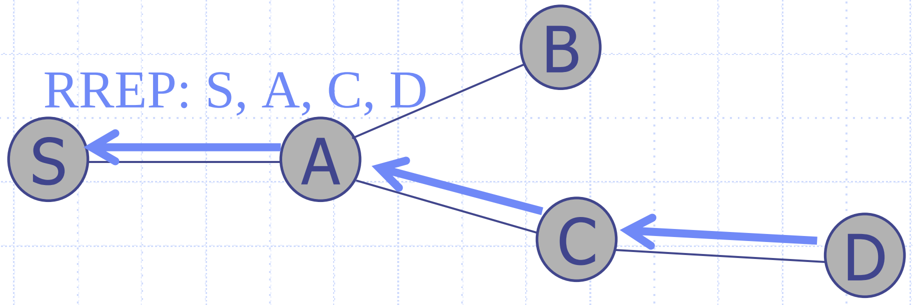
6. Node *C* receives RREP
   - Unicasts to *A*

DSR: Route Discovery



6. Node *A* receives RREP
   - Unicasts to *S*

DSR: Route Discovery



8. Node *S* receives RREP
   - Uses route for data packet transmissions

General node operation upon receiving RREQ

- If the pair <initiator address, request ID> has recently been seen, DISCARD
- If the node ID is already listed in the source route DISCARD→ avoids loops
- If I’m the destination, send a RREP
- Otrherwise, append my ID in the source route and rebroadcast (orange cases already seen in the previous slides)

Route maintenance

- The two endpoints of a failed link are transmitted to the <u>source</u> in a route error packet
- Upon a receiving a RERR packet a node invalidates all the routes going through that link
- If the route is invalidated and it is needed, a new route must be discovered

Optimizations (1)

- Extensive use of caching (caching source routes means that I already know all the routes to intermediate destinations, the discovery of a better route to an intermediate destination also brings me to improving the route to the final destination). Transmitting packets or sending back replies make me learn routes.
- A node that knows a route to a given destination (has a source route in cache) can immediately answer a RREQ
  - Broadcast storm? Each nodes waits for a time which is C*(h-1+r), r random in (0,1), h length of the route I'm advertising. Only if I haven't received other routes – listen to other routes tx in the meanwhile-I transmit mine.

Optimization (2)

- Operation in promiscuous mode (I keep discovering new routes by transmission of routes by my neighbours)
- RREQ overhead minimization: first set a TTL=1, if I do not get answer I set it to infinity
- Path shortening: if Y receives a packet by node X but in the source route we have X, B,…,C,Y, Y signals the path can be shortened (unsolicited RREP)
- What if the network is disconnected? Exponential back-off to decrease the quantity of RREQ sent

AODV and DSR Differences

- DSR uses source routing;
  AODV uses next hop entry
- DSR uses route cache;
  AODV uses route table

Geographically-Enabled Routing

- Outline
  - Problems with proactive approaches
  - Problems with reactive approaches
  - A new way of naming\locating the destination node
  - Two seminal protocols
    - DREAM & LAR, proactive and reactive
  - Geo-enable routing and the link state idea

Proactive Solutions: Drawbacks

- Updates overhead, especially in presence of high mobility
- Overhead for enforcing loop freedom
- Large routing tables
- Low *scalability*
  - Is it really necessary to maintain a consistent view of the network topology?

Reactive Protocols: Drawbacks

- The discovery phase introduces long delays
- Route discovery and maintenance is very sensitive to node mobility
- Route caching is memory greedy
- Operating in promiscuous mode is very energy cnsuming
- The size of the header of a data packet can become cumbersome (no scalability)
  - Is the dependency on the network topology avoidable?

Location-Enabled Ad Hoc Routing

- Nodes are equipped with positioning system devices (e.g., Global Positioning System receivers) that make them aware of their position
- This enables “directional” routing
- Possible solutions differ on how the location information of the destination nodes is achieved

Strengths

- No need to update big routing tables, no need to piggyback routes to the packet
- No need to know the nodes on the way to the destination: they can be moving while the packet travels

Drawbacks

- Needs extra hardware
- Depends on the extra hardware limitation (and resource requirements)
- Scalability is an issue
  - need to update nodes’ location info

DREAM

- Distance routing effect algorithm for mobility [Basagni+, 1998]
- A proactive, effective way to spread location information
- Directional routing

Disseminating Location Information: Problems

- Need to periodically update the location of a moving node.
  - Efficient broadcast of location information
  - Determining how far each location packet should travel
  - Determining how often a location packet should be sent

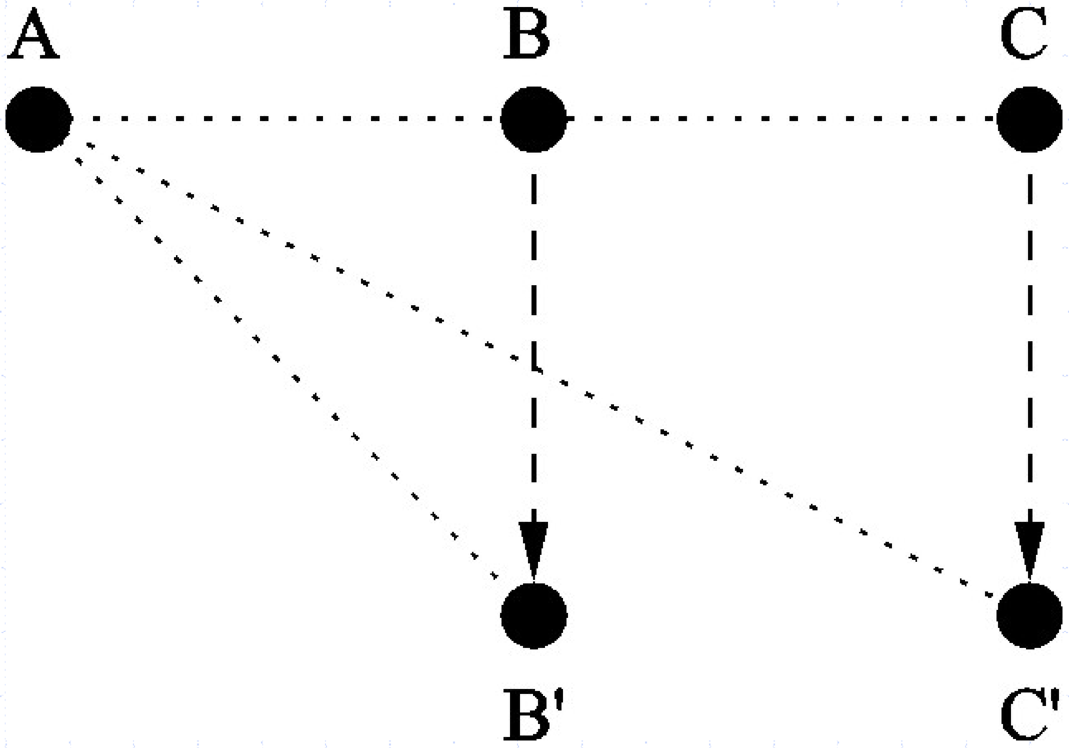
Disseminating Location Information: Solutions

- Mobility-adaptive, deterministic broadcast
- Distance effect
- Rate of updates is bound to the mobility of the node

Mobility-Adaptive Broadcast

- Deterministic solution that takes into account MAC layer characteristics
- Flooding of location packets proceeds “wave expanding” from the source to the intended destinations
- Deterministic, interference-independent delivery is obtained by using Time-Spread Multiple-Access (TSMA) protocols

The Distance Effect



The Distance Effect

- "Closer nodes look like they are moving faster"
- Need to receive more location updates from closer node
- Each packet is associated with an age that determines how far that packet must travel

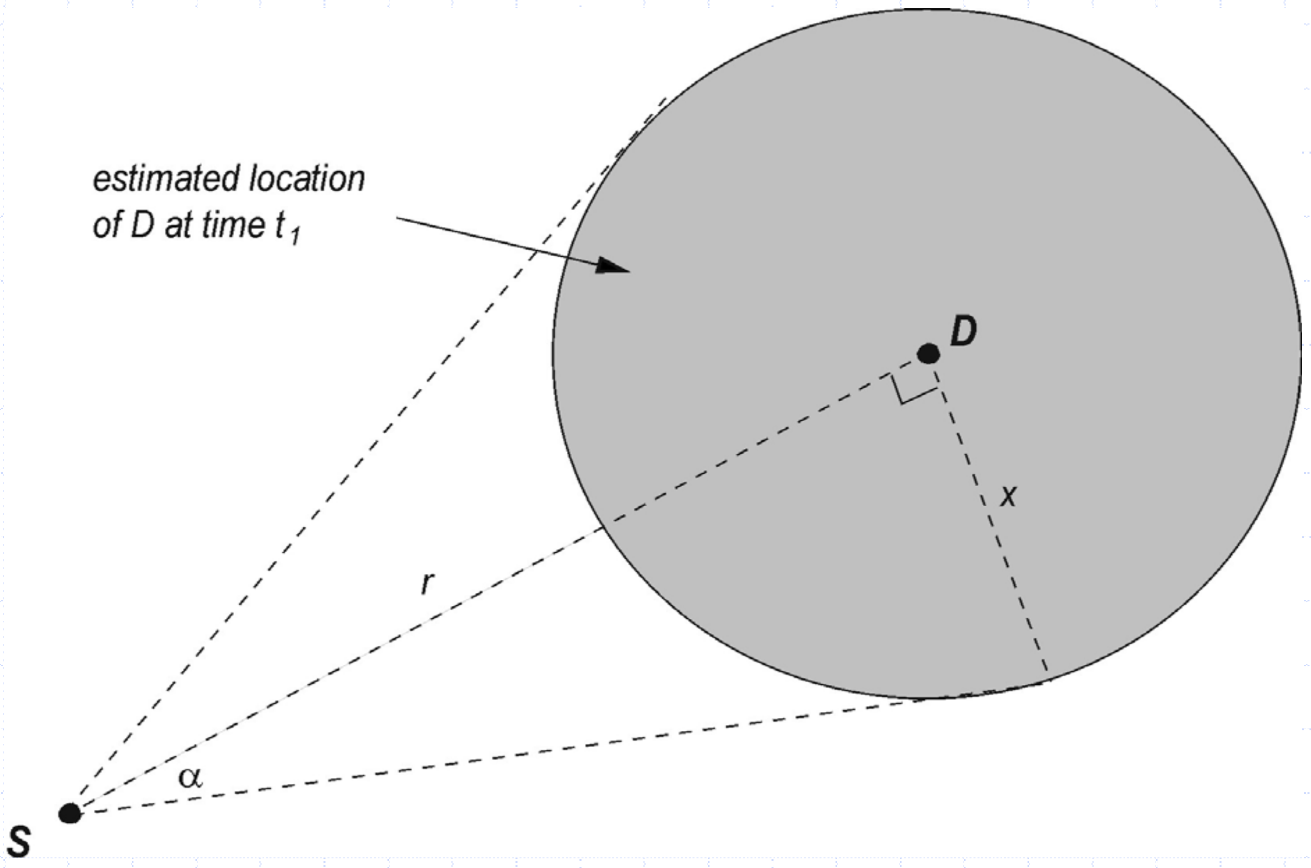
DREAM: Rate of updates

- Triggered by the mobility of the nodes
- The faster the node the more updates it sends
- A plus: slow moving nodes impose little overhead

DREAM: Directional Routing

- Source S determines the location of destination D at time t_0 based on its location table
- Based on the current time t_1 and t_0 S determines the area in which D can be found (hence, D’s direction)
- S transmits the data packet to all its neighbors in D’s direction
- Each neighbor does the same till D is reached

DREAM: Routing a Data Packet



DREAM, Strengths

- First of its kind: after that, the deluge!
- Robustness: multiple routes to the destination

DREAM, Weaknesses

- It is flooding, although only directional
- It is not that scalable, geographic info updates have to be periodically transmitted (even if mechanisms to limit such overhead are enforced)

Location-Aided Routing (LAR)

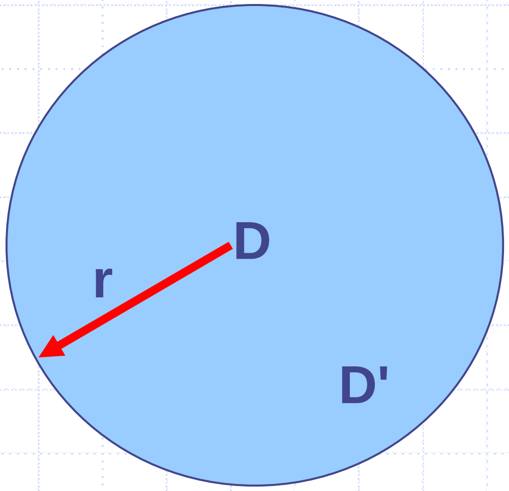
- Exploits location information to limit scope of RREQ flood
- *Expected Zone*: region that is expected to hold the current location of the destination
  - Expected region determined based on potentially old location information, and knowledge of the destination's speed
- RREQs limited to a *Request Zone* that contains the Expected Zone and location of the sender node

LAR: Expected Zone

D = last known location of node D, at time t_0

D' = location of node D at current time t_1, unknown to node S

r = $(t_1 – t_0)$ * estimate of D’s speed

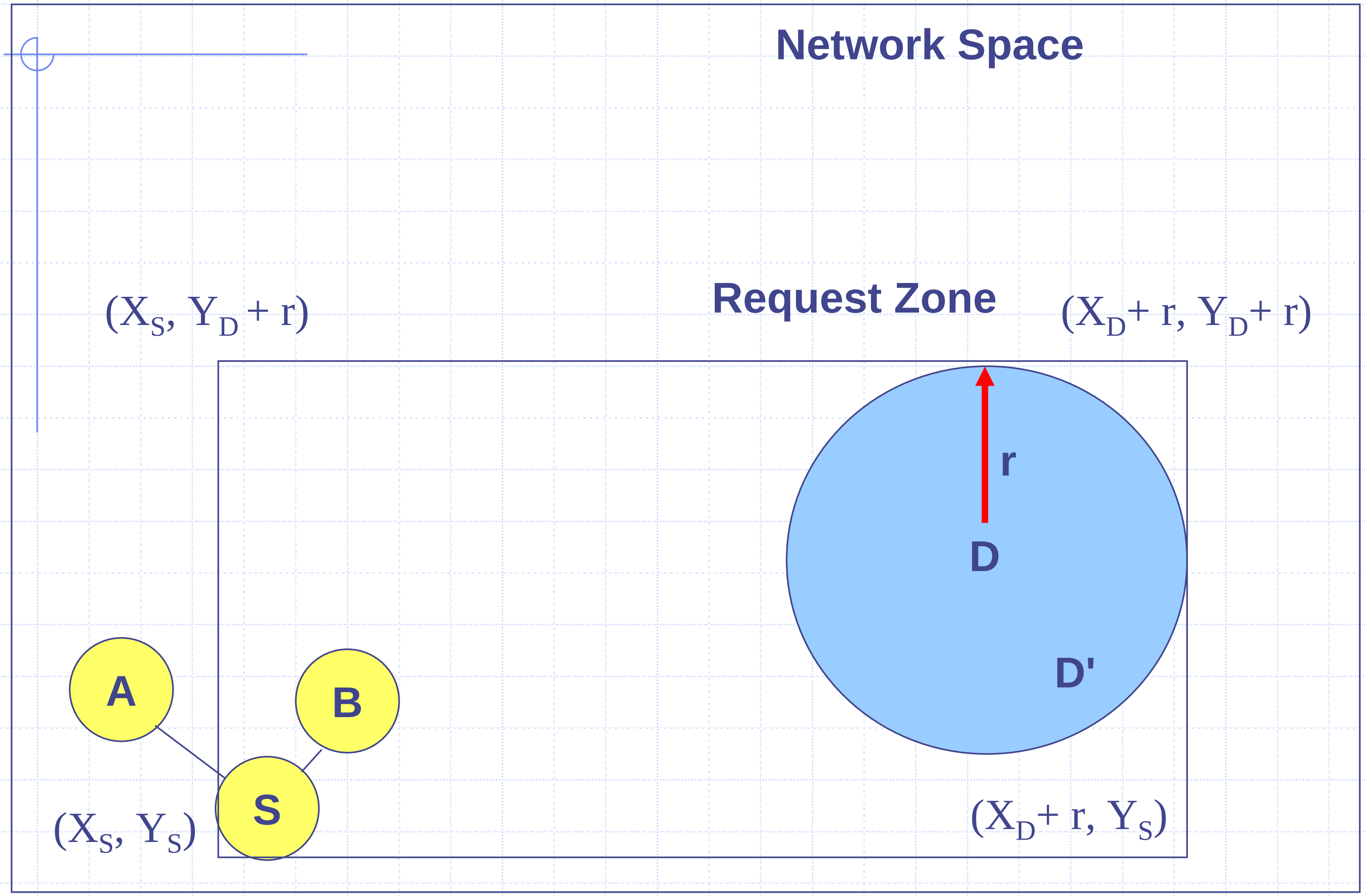


Expected Zone

LAR

- The request zone is the smallest rectangle that includes the current location of the source and the expected zone
- Only nodes within the request zone forward route requests
  - Node A does not forward RREQ, but node B does
- Request zone explicitly specified in the RREQ
- Each node must know its physical location to determine whether it is within the request zone

LAR: Request Zone



LAR, Possible Failures

- If route discovery using the smaller request zone fails to find a route, the sender initiates another route discovery (after a timeout) using a larger request zone
  - The larger request zone may be the entire network
- Rest of route discovery protocol similar to DSR

LAR, the Routing

- The basic proposal assumes that, *initially*, location information for node X becomes known to Y only during a route discovery
- This location information is used for a future route discovery

Variations

- Location information can also be piggybacked on any message from Y to X
- Y may also proactively distribute its location information

LAR, Pros and Cons

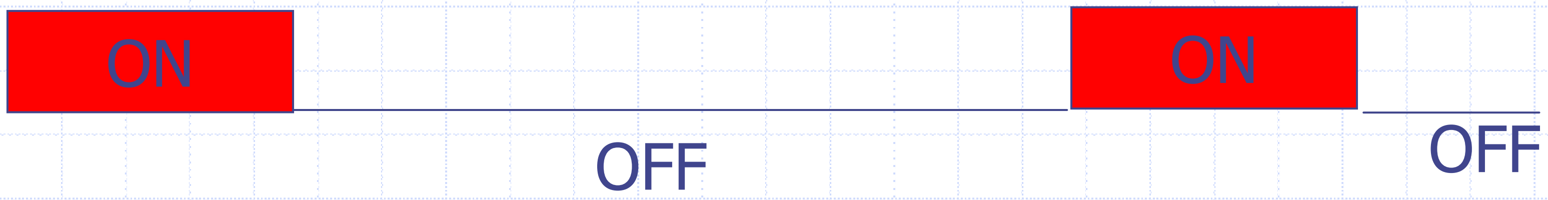
- Advantages
  - Reduces the scope of RREQ flood
  - Reduces overhead of route discovery
- Disadvantages
  - Nodes need to know their physical locations

In a sensor network we seek..

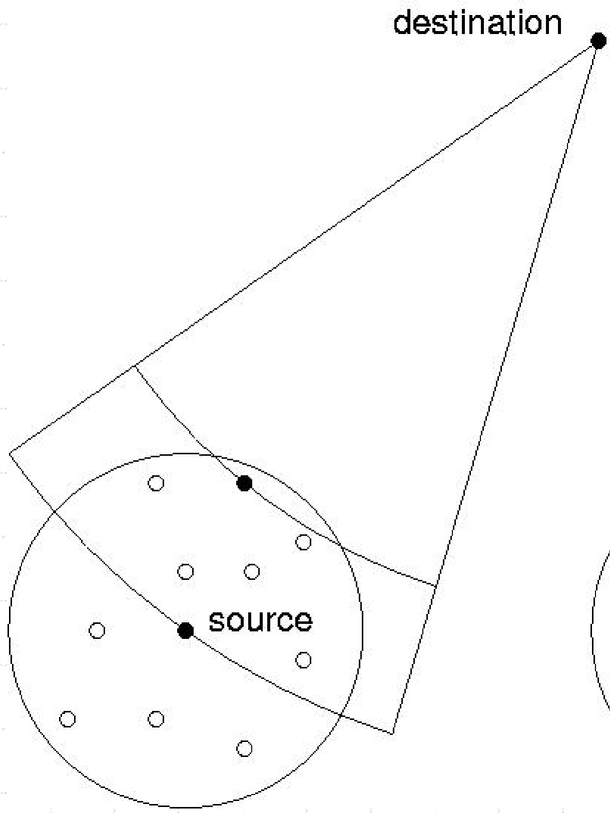
- For solutions which scale
- Which are energy saving
- Which are well integrated with awake/asleep schedules
- Which do not require to maintain routing tables
- Which are simple
- Solutions such as AODV and DSR has been proven to work well iff they exploit intensively caching and promiscous mode operation (energy inefficient← work by L. Feeney et al, 2001) and have been shown not to scale to the volumes of nodes expected in sensor networks (work by E. Belding Royer and S.J. Lee)
- What can we use?
  - communication sensors – sink
  - Info such as localization and some level of synchronization often needed by the application (if I sense an event I have to say WHERE and WHEN it occurred, otherwise the information is not very interesting)

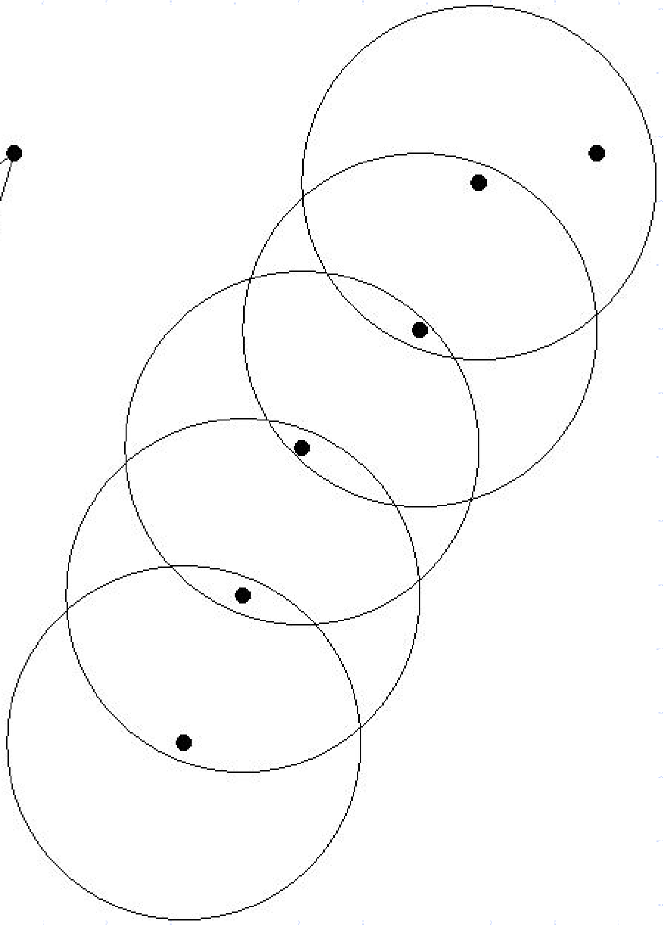
An example: GeRaF

- Integrates
  - geographic routing
  - awake/asleep schedule
  - MAC
- How do nodes alternate between awake ans asleep states? According to a duty cycle (time ON/time ON+ OFF)



GeRaF basic idea





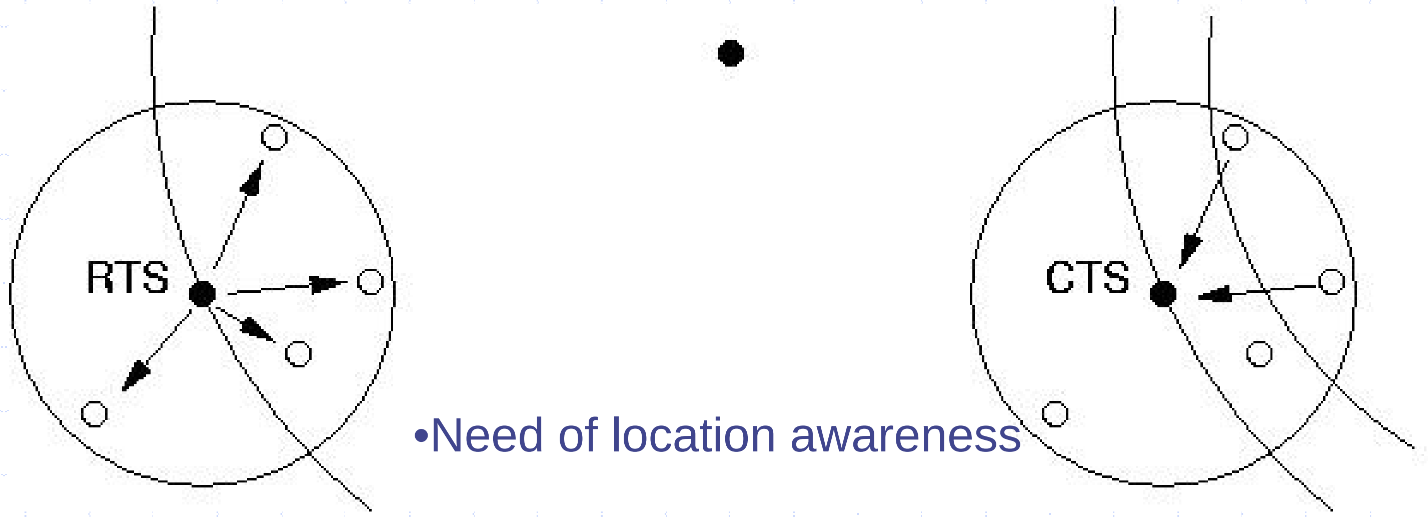
Geographic routing: each node needs to know its location, the destination (sink) location, and the location of whom is transmitting (communicated in the packet)
Greedy approach: tries to select relays so to advance as much as possible toward the destination at each hop

GeRaF: operations

- Main problem to be solved: how to make a contention-based scheme/routing work in the presence of sleep modes
  - Flat solution
  - Integrated MAC/routing/awake-asleep but awake-asleep schedule and routing decoupled → each node does not know its neighbor and their schedules → low overhead
- Tightly integrated with the routing layer (no clear separation really)
  - Without requiring routing tables/routing table updates
  - Based on the knowledge of the nodes location and on the knowledge of the sink location

Example of GeRaF operation

- RTS invites all awake neighbors to become a relay
- Nodes in best position should win
  - Nodes within tx range are divided in areas depending on how close they are to the final destination (the closest the better as relay)



•Need of location awareness

GeraF operation

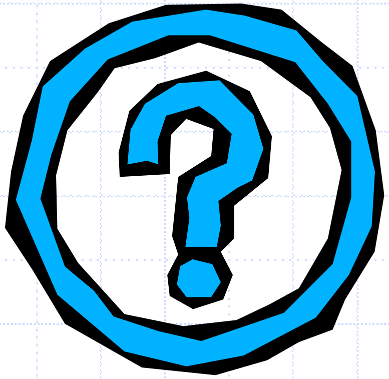
- Node i sends RTS with the identity of the area it is polling now (starting from the closer to the sink, among the slices in which its tx range has been divided)
- Each node, upon receiving the RTS, decides whether it belongs to the polled area or not (based on location info)
- Only nodes in the polled area answer with a CTS
  - No node answers → node i polls next area (no node available for forwarding in the area-there are no nodes or they are sleeping)
  - One answer, CTS correctly received, send DATA
  - Multiple answers→ COLLISION, sender sends a collision packet, MAC needed to solve collision (next slide)

GeraF operation (how to handle collision)

- 1)A node receiving a collision packet tosses a coin and with probability p transmits a CTS iff it was participating to the previous phase (it had previously sent a CTS resulting in collision)
  - if only one node answers node i sends data
  - If no node answers, node i asks these nodes to toss a coin again..
  - if more nodes answer COLLISION. Collision packet is sent. GO TO 1) (only the nodes which have lead to collision survive to the next phase)

What If

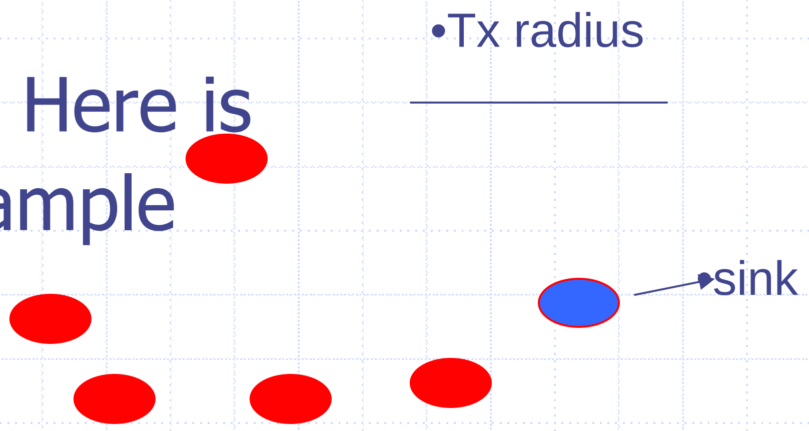
- All areas are polled unsuccessfully?
  - Try again after some time (exponential backoff)
- Can I always reach the destination in this way?



What if (answer)

- No. Here is

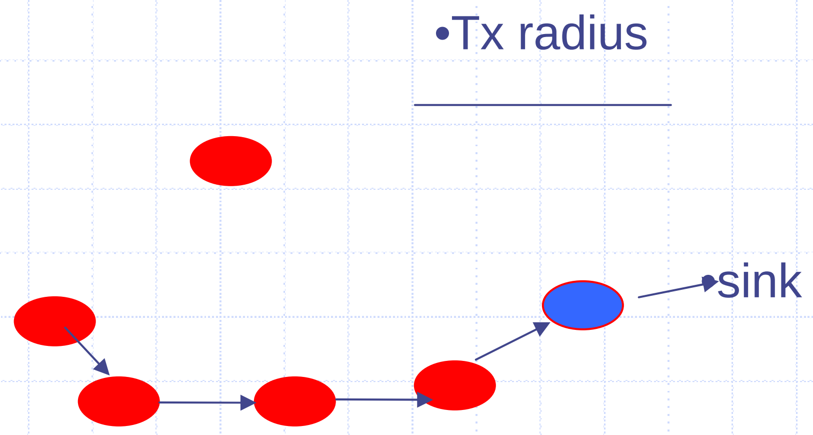
An example



- Solutions? (mechanisms have to be added to recognize the problem, do backtracking and try another route)

How to solve dead-ends

- A problem at low--medium density
- We will see a solution (ALBA-R) in the part on sensor networks



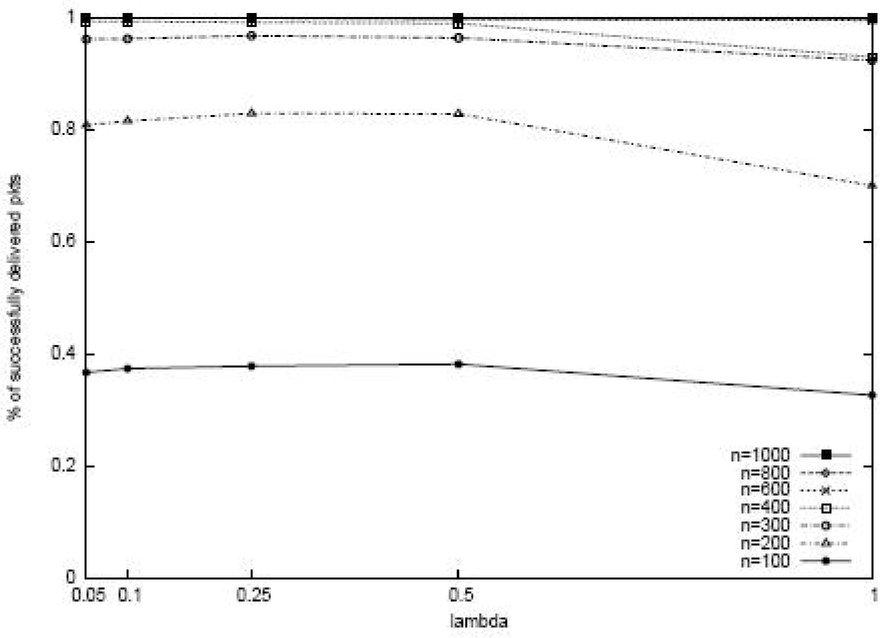
GeRaf performance

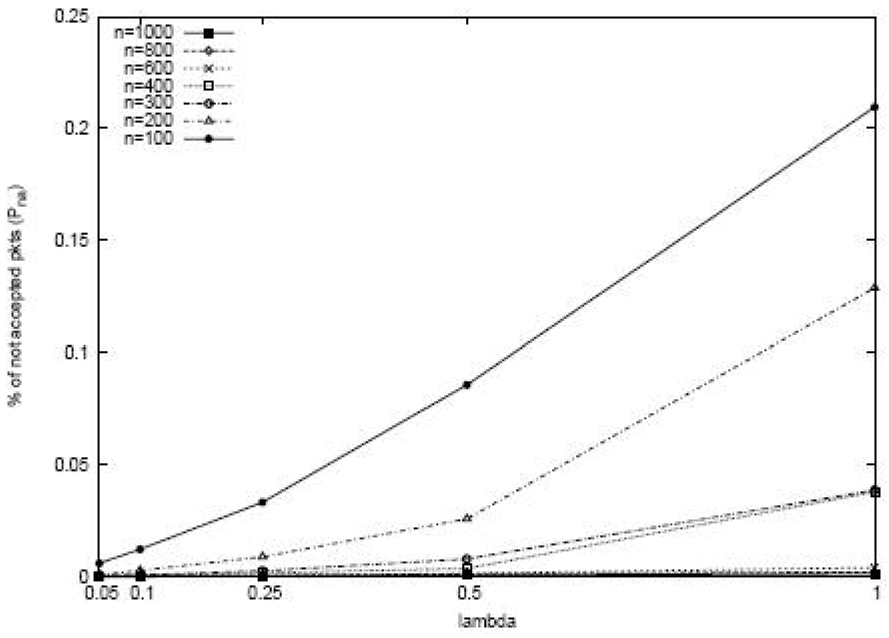
Casari, Marcucci,Nati, Petrioli, Zorzi IEEE MILCOM 2005

- square area 320m x 320m
- Transmission range=40m
- 100-1000 randomly deployed nodes (avg degree 5-50)
- Duty cycle =0.01,0.1,0.5
- Comparable costs for tx/rx/idle
- Poisson packet arrival
- Channel data rate 38Kbps

GeRaf performance, d=0.1

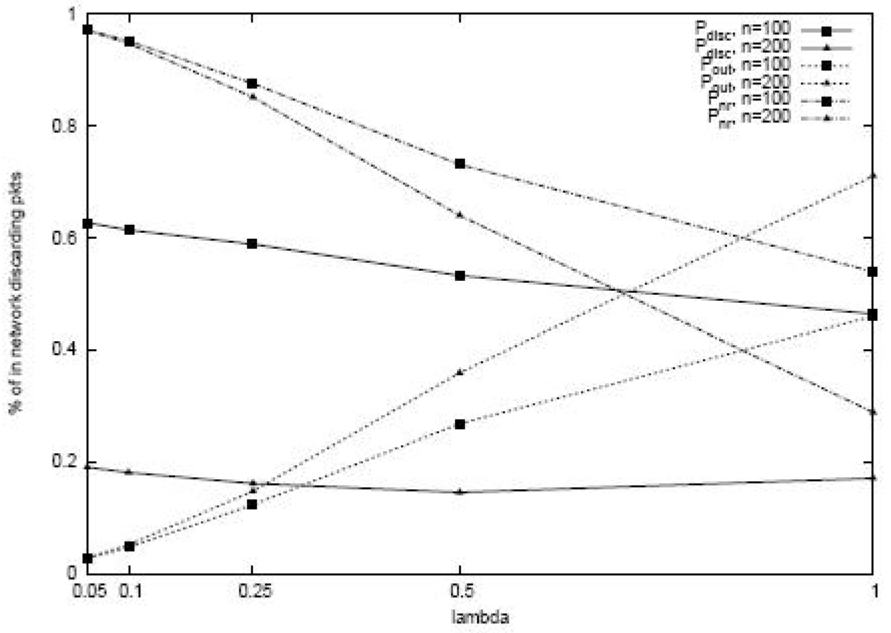
Casari, Marcucci,Nati, Petrioli, Zorzi IEEE MILCOM 2005





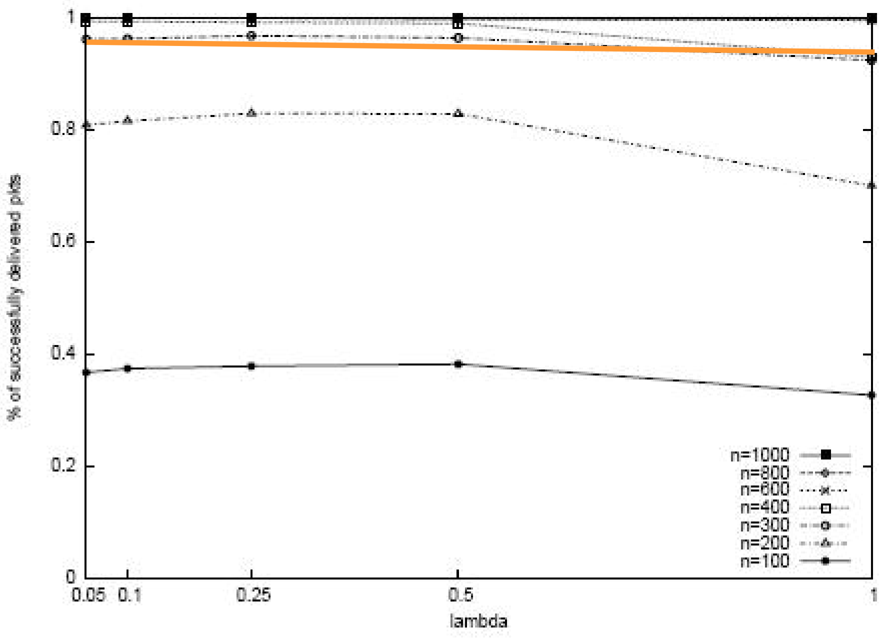
GeRaf performance, d=0.1

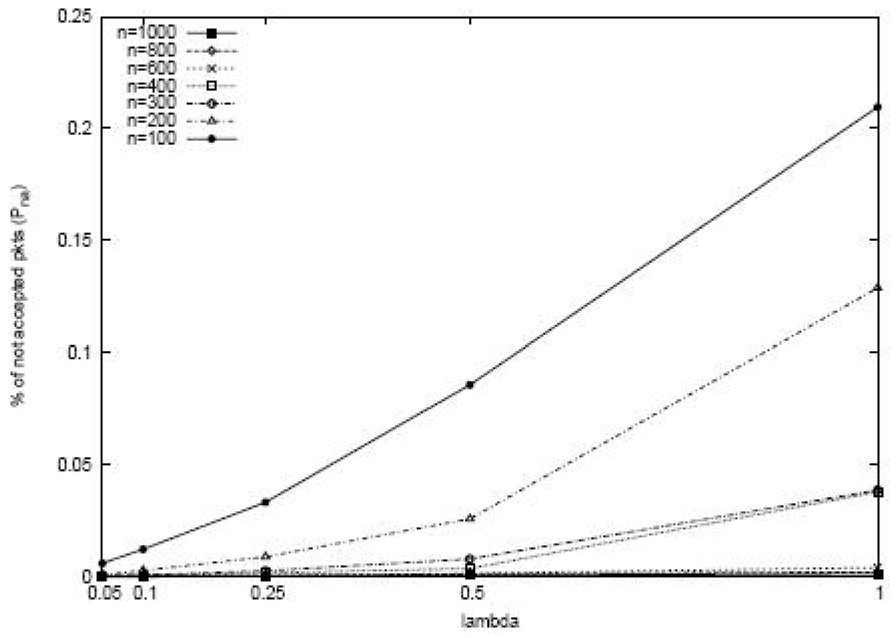
Casari, Marcucci,Nati, Petrioli, Zorzi IEEE MILCOM 2005



GeRaf performance, d=0.1

Casari, Marcucci,Nati, Petrioli, Zorzi IEEE MILCOM 2005





Con i meccanismi per evitare dead end si sale a % di successfully Delivered packets nel caso di 200 nodi pari a 93-97%

Localization in sensor networks

Thanks to Prof. Mani Srivastava
These slides have been derived
From his tutorial on sensor networks
Given at Rome Un. On July 2003

Localization

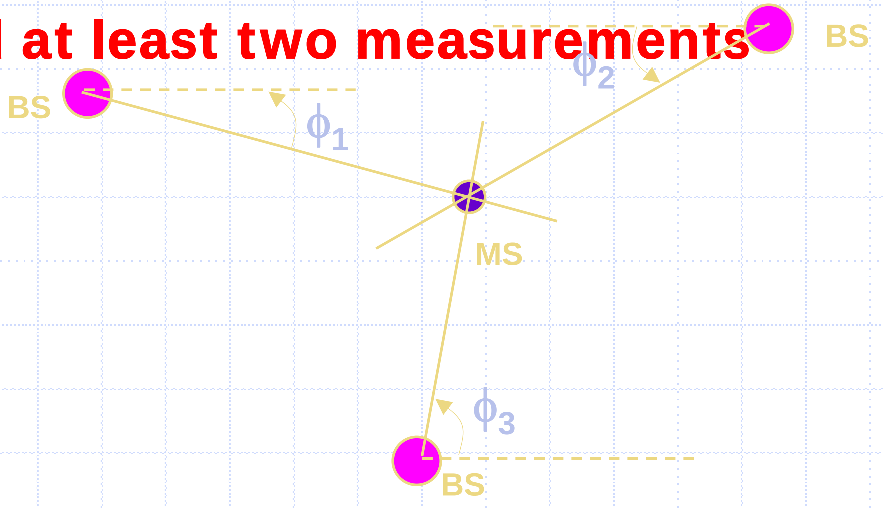
- Useful info
  - Helps with some protocols (e.g. GeraF)
  - Needed for being able to identify where events occur
- Why not just GPS (Global Positioning System) at every node?
  - Large size
  - High power consumption
  - Works only when LOS to satellites (not in indoor, heavy foliage…)
  - Over kill – often only relative position is needed (e.g. enough to know that relative to a coordinate system centered in the sink the event occurred in a position (x,y). Starting from relative info if some nodes have global coordinates global coordinates of events can be inferred.
  - Works only on earth ☺

Localization

- Basic step is to evaluate distance between two nodes (ranging). Different techniques depending on the available HW:
  - AoA (e.g. directional antennas)
  - RSS
  - ToA
- Range free approaches (number of hops between nodes used to estimate the distance between them without using any extra HW)

Techniques for Location Sensing (AoA)

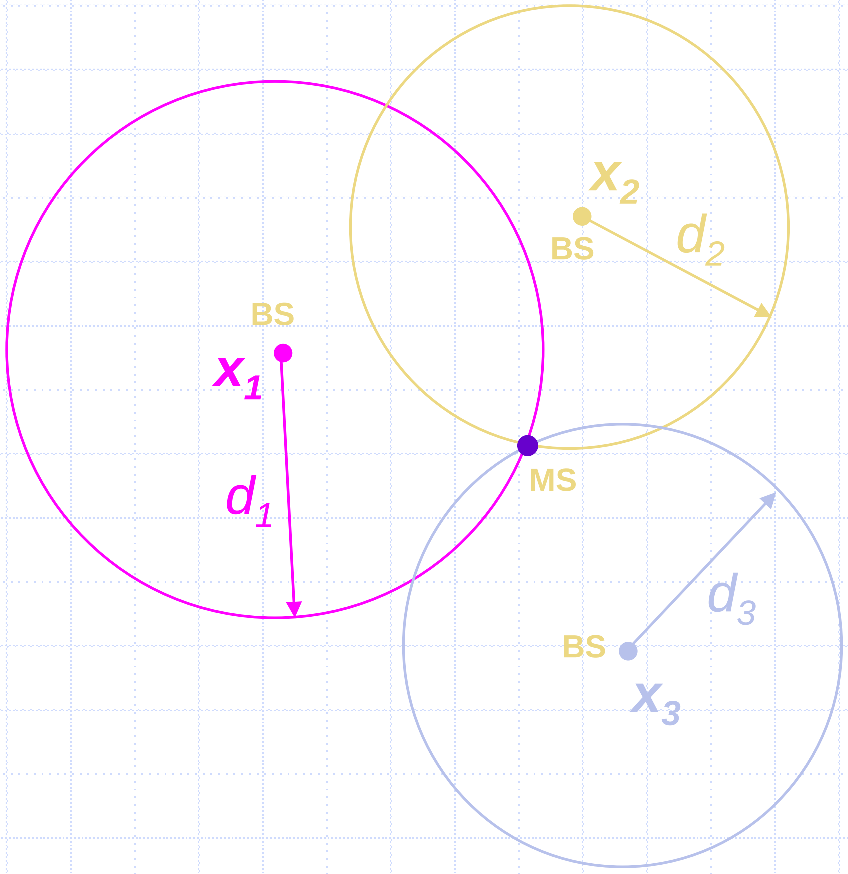
- Measure direction of landmarks
  - Simple geometric relationships can be used to determine the location by finding the intersections of the lines-of-position
  - e.g. Radiolocation based on angle of arrival (AoA) measurements of beacon nodes (e.g. base stations)
    - can be done using directive antennas + a compass
    - **need at least two measurements**



Techniques for Location Sensing (RSS or ToA)

- Measure distance to landmarks, or Ranging
  - e.g. Radiolocation using signal-strength or time-of-flight
    - also done with optical and acoustic signals
  - Distance via received signal strength
    - use a mathematical model that describes the path loss attenuation with distance
      - each measurement gives a circle on which the MS must lie
    - use pre-measured signal strength contours around fixed basestation (beacon) nodes
      - can combat shadowing
      - location obtained by overlaying contours for each BS
  - Distance via Time-of-arrival (ToA)
    - distance measured by the propagation time
      - distance = time * c
    - each measurement gives a circle on which the MS must lie
    - active vs. passive
      - active: receiver sends a signal that is bounced back so that the receiver knows the round-trip time
      - passive: receiver and transmitter are separate
        - time of signal transmission needs to be known
  - N+1 BSs give N+1 distance measurements to locate in N dimensions

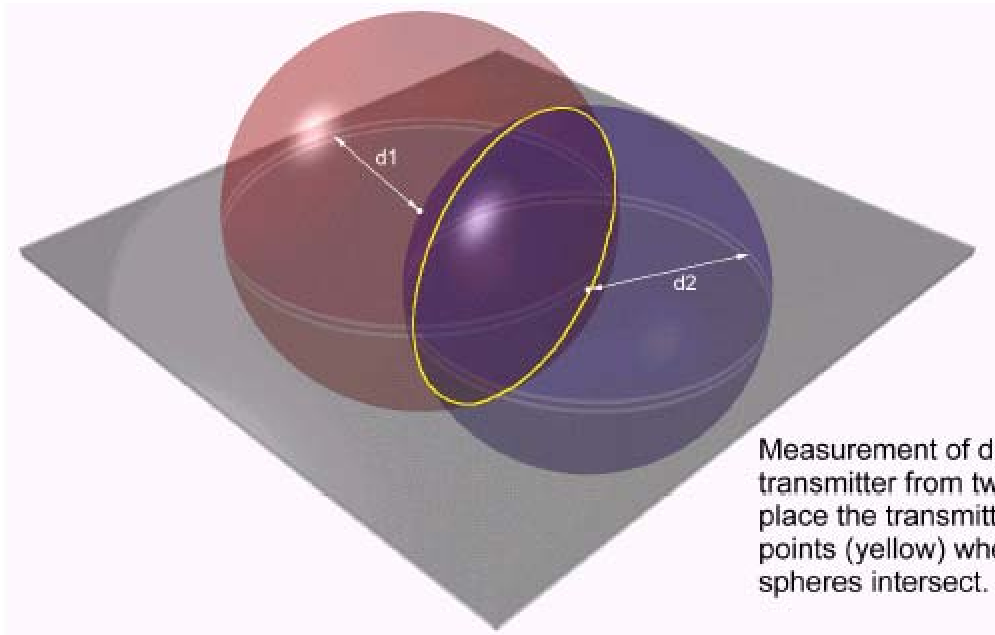
Radiolocation via ToA and RSSI



- But what if the circles do not intersect due to measurement errors (e.g. due to fading etc.)?

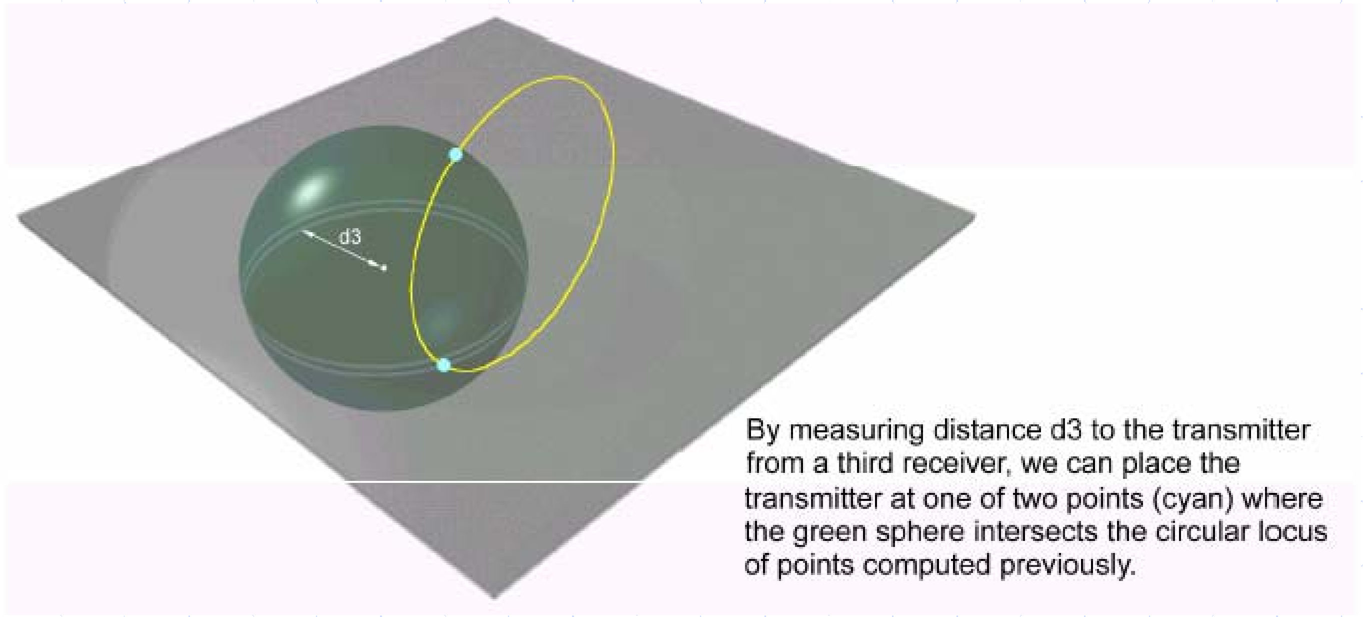
→will have to identify the best ‘guess’ given errors

Location in 3D



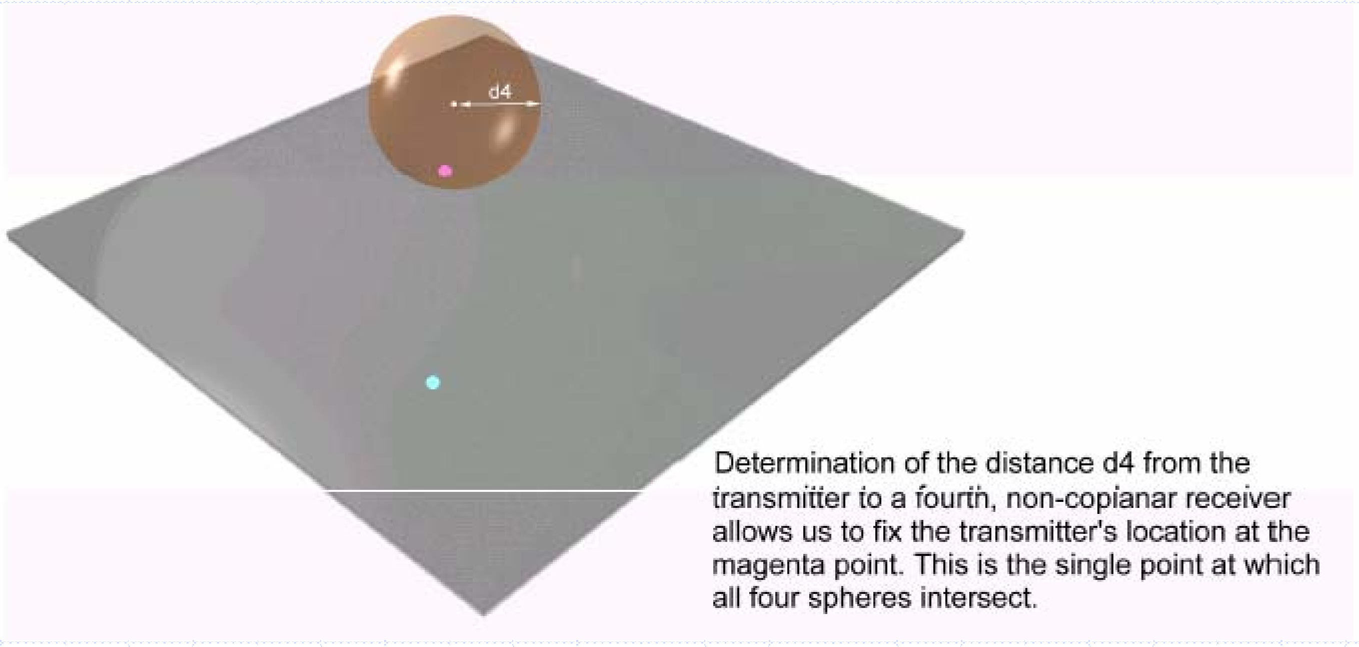
Measurement of distances d1 & d2 to the transmitter from two receivers allows us to place the transmitter on a circular locus of points (yellow) where the red and blue spheres intersect.

Location in 3D



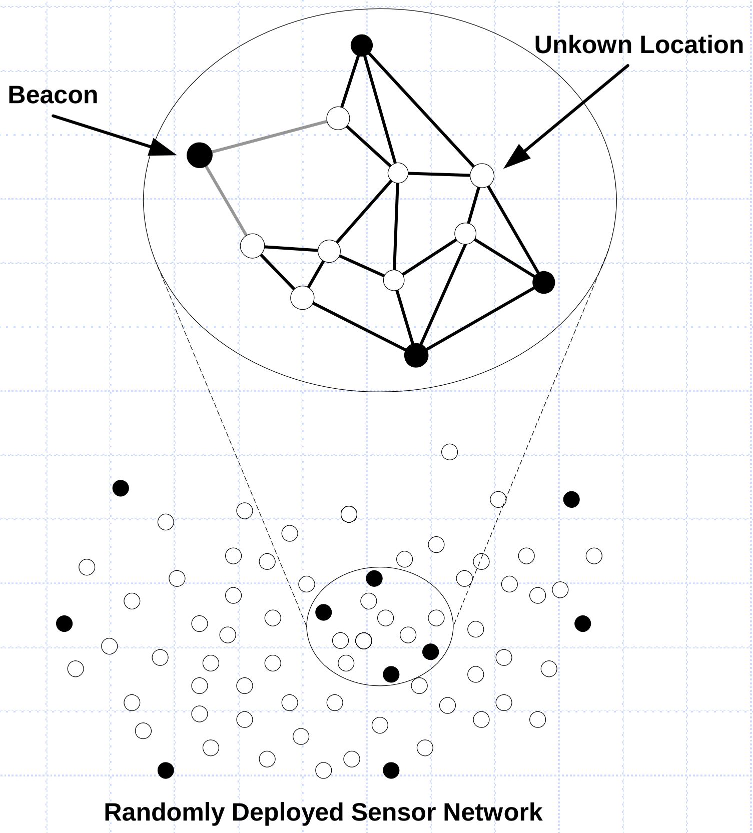
By measuring distance d3 to the transmitter from a third receiver, we can place the transmitter at one of two points (cyan) where the green sphere intersects the circular locus of points computed previously.

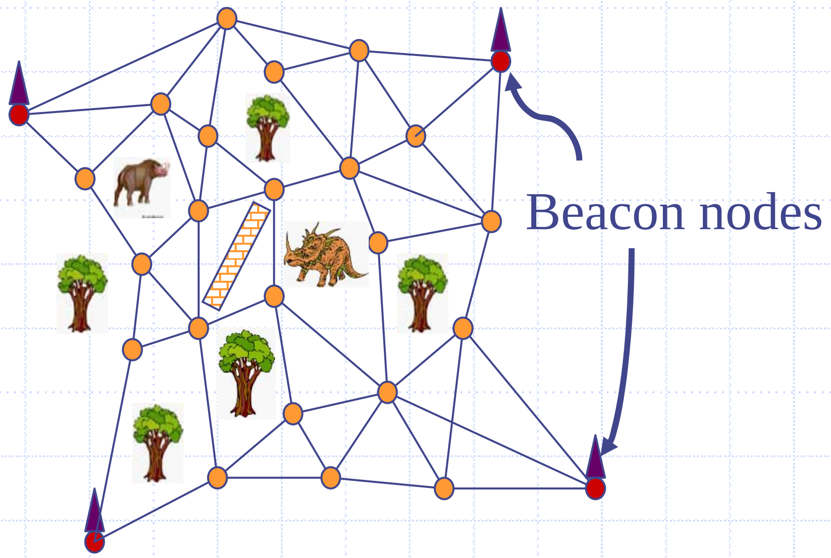
Location in 3D



Determination of the distance d4 from the transmitter to a fourth, non-copianar receiver allows us to fix the transmitter's location at the magenta point. This is the single point at which all four spheres intersect.

A possible solution: Absolute Localization

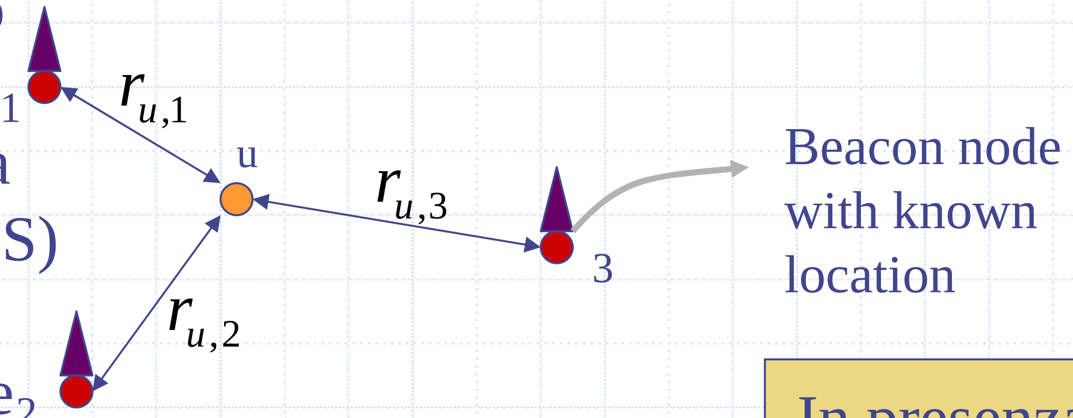




- A small fraction of the nodes is aware of their locations
- Rest need to collaborate to estimate their locations

Atomic Multilateration

Nodi che hanno almeno 3 vicini (in 2D, se si usa Ad esempio RSS) beacon possono stimare la loro posizione (triangolarization)



In presenza di errori

Metrica di interesse

Errore quadratico medio

$$f_{u,i} = r_{u,i} - \sqrt{(x_i - \hat{x}_u)^2 + (y_i - \hat{y}_u)^2}$$

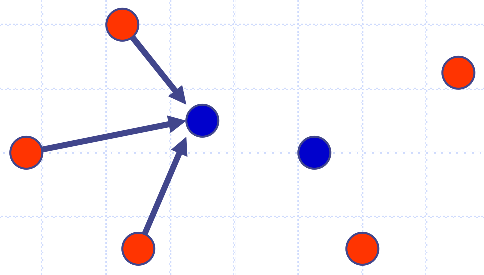
$(\hat{x}_u, \hat{y}_u)$ - initial position estimate for node *u*

Our objective function is:

$$F(x_u, y_u) = \min \sum f_{u,i}^2$$

Iterative Multilateration

- Nodes that estimate their locations can become beacons and help other nodes discover their locations.



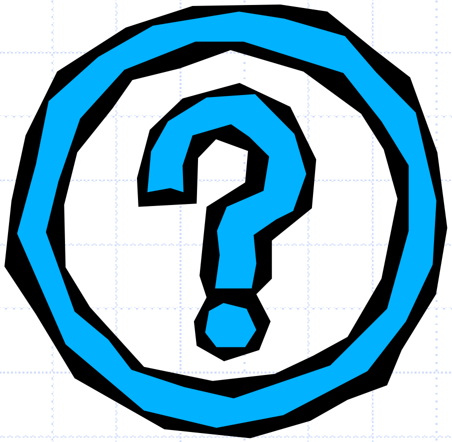
- Some observations:
  - Can work for small networks, if ranging is accurate
  - Energy efficient
  - Still requires quite a lot of initial beacons
  - Suffers from error accumulation
  - **Bad geometry yields bad results => unpredictable performance**
  - Still a useful primitive for Distributed Collaborative Multilateration

  Primo approccio semplice, vasta letteratura

Ref: based on slides by Andreas Savvides

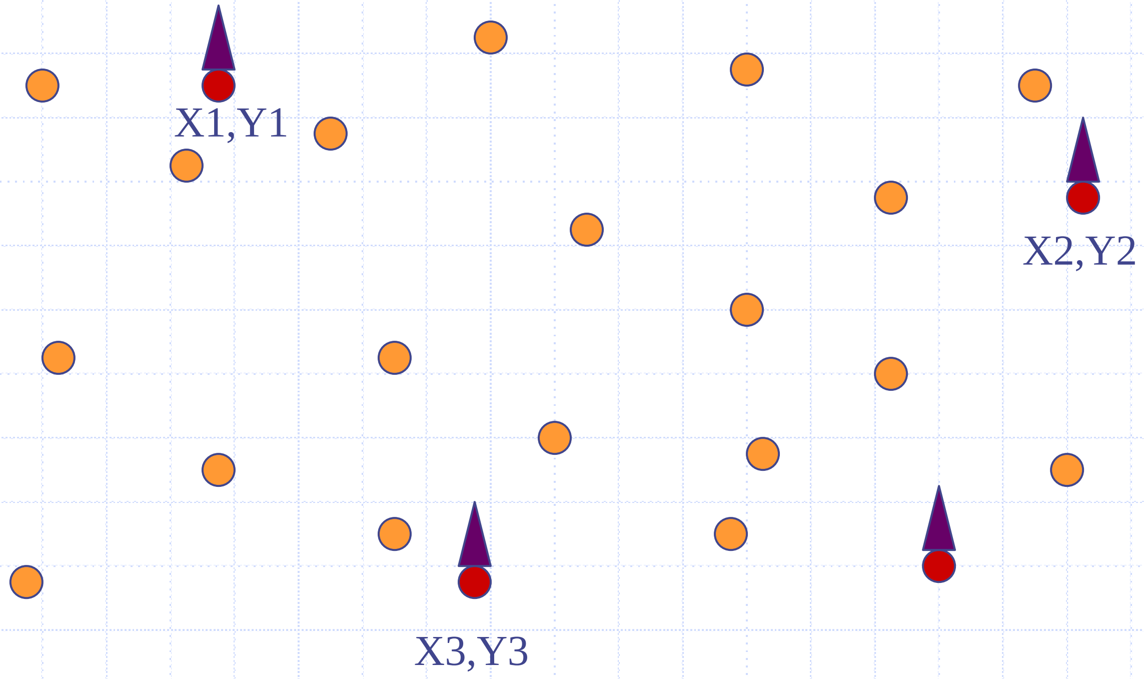
Range free localization

- Non usa ranging, ma solo informazioni che si possono ottenere tramite algo di routing tradizionali
- Idee su come possa funzionare?



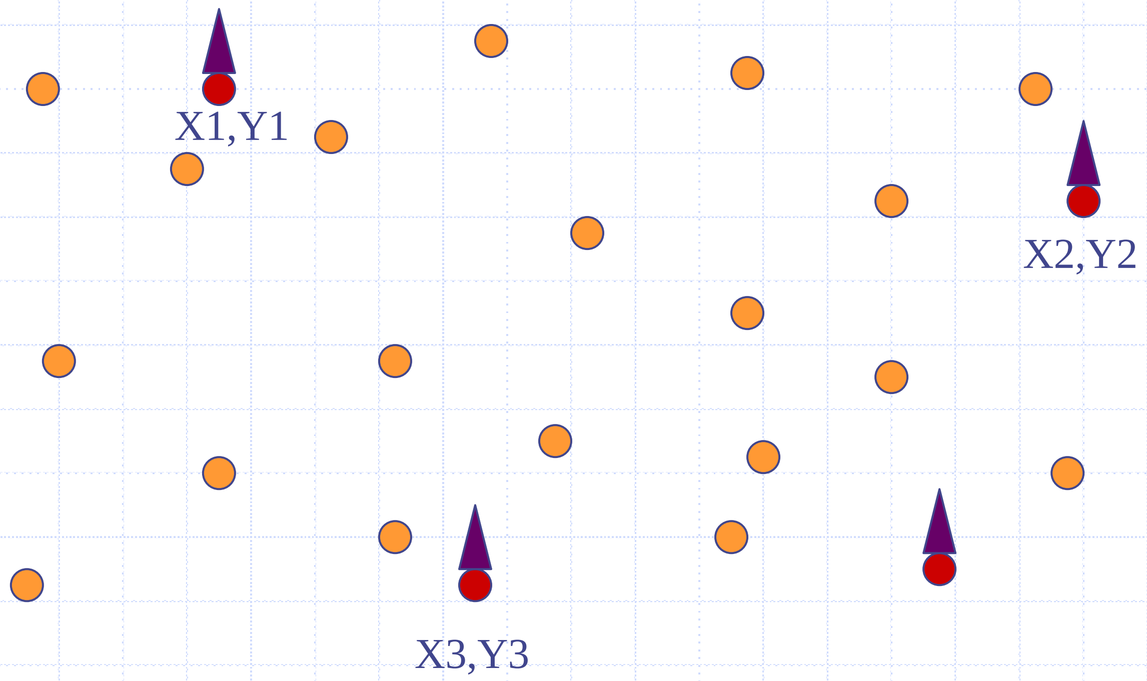
Qualche idea sull'approccio …

- Servono degli anchor, nodi che conoscono la propria posizione in un sistema di coordinate comune



Qualche idea sull'approccio …

- Tutti I nodi calcolano il numero min. di hop tra loro e gli anchor
- Anche gli anchor lo fanno tra loro



Qualche idea sull'approccio …

- Anchor A: conosco la posizione esatta mia e degli altri anchor, il num. di hop, posso stimare la 'lunghezza media di un hop'
- Questa informazione e' usata per stimare le distanze da tutti I nodi agli anchor. Sulla base di tali distanze, le corrette coordinate degli anchor, per triangolarizzazione ciascun nodo stima le proprie coordinate

- Pro: Non serve extra HW
- Cons: si perde in precisione

Scalability Problems and Clustering

- What happens to protocols when the number of network nodes grows?
  - Especially crucial in WSNs
- A traditional networking solution: Hierarchical organization of the nodes
- Network nodes are grouped into clusters
- Some nodes, locally the “best,” are selected to coordinate the clustering process: Clusterheads

How to Select the Best Nodes

- Independence of the clusterheads
- Dominance of the clusterheads
- Possibility to express “preferences”
- Distributed operations
- Fast and simple implementation

Previous Approaches

- Heuristics based on Independent Sets
  - Minimum ID approach (Gerla & al.)
  - Maximum degree (Ephremides & al.)
- Heuristics based on Dominating Sets
  - The concept of “spine”
  - Minimum connected dominating set

Previous Approaches: Drawbacks

- No preferences
- Clustering “set up” differs from clustering maintenance
- One *and* two hops neighbors have to be known at each node
- Problems with nodes mobility
- No analytical results

Maximum Independent Set (MIS)

- A subset V′ of the vertices V of a graph G=(V,E) is independent when for each u,v $\in$ V′ the edge {u,v} $\notin$ E
- MIS is an Optimization Problem
- Input: A Graph G=(V,E) with n vertices
- Output: A subset V′ of V that is independent and has maximum size

MIS: Hardness

- No known algorithm computer a MIS in polynomial time
- Need for approximate solutions
- And approximation algorithm is an algorithm that produces a solution that is not optimal, but that approximates it
- We sacrifice optimality in favor of a “good” solution that can be computed efficiently

MIS is HARD to Approximate

- Bad news
  - Not only MIS is computationally hard
  - It is also hard to approximate:
    - Approximate solutions are not so good
    - They are “unboundedly” far from the optimum
- We consider the simple greedy heuristic for the MIS

Greedy Heuristic for MIS, 1

- Select the vertex with minimum degree and put it in the MIS
  - The degree of a vertex is the number of its neighbors
    - Cardinality of its adjacency list
  - Keep going till all the vertices are either in the MIS or COVERED by a vertices in the MIS

Greedy Heuristic for MIS, 2

```
MIS(V,E,d) // d is the vector of degrees
  mis = Ø
  while V ≠ Ø do
    v = vertex with min degree
    mis = mis U {v}
    V = V – {{v} U  N(v)}
  return mis
```

Greedy MIS: Maximal Solution

- The greedy solution provides a maximal independent set
  - An independent set is maximal when, if you add a vertex, the set is no longer independent
    - You cannot make an maximal independent set bigger
- This solution is also a minimal dominating set
  - A dominating set D subset of V is a set such that a vertex $v \in V$ is either in D or it has a neighbor in D

Using Greedy MIS on UDG to Computer a DS

- Bad news: Still computationally hard
- Better news: Minimum DS It is approximable “up to a constant”
  - It means that the ratio between the size of a DS computed by MIS greedy on UDGs and the size of a MDS is < c, c a constant
- This constant is 5

Greedy MIS for MDS on UDG is 5-approximable, 1

- Key fact: In a UDG disk (radius 1) there are at most 5 independent nodes
- Consider an Optimal solution and a Greedy solution
- Since Opt is dominant, it dominates Greedy
- Assign every vertex of Greedy to one dominator in Opt (choose one if more)

Greedy MIS for MDS on UDG is 5-approximable, 2

- For each u in Opt consider its assigned vertices $v_1(u)$, $v_2(u)$, …, $v_k(u)$ of Greedy
- How big is k?
- Well, all $v_i(u)$ must be distant 1 from u and they also have to be independent
- Greedy: at most 5 times bigger than Opt

MIS and Dominating Sets and Wireless Networks

- UDGs model ad hoc networks
- IS and DS are useful for clustering ad hoc networks
  - Gives the network a hierarchical organization
  - Decreases the amount of information at each node
  - Enhances scalability
  - Helps in "resource assignment"

MWIS-Based Clustering

- MWIS = Maximal Weight Independent Set
- Clustering selection based on generic **weights** (real numbers > 0)
  - Mobility/node related parameters
  - Generalizes previous “Independent Set” solutions

Two Protocols

- Distributed Clustering Algorithm (DCA)
  - Quasi-mobile networks, periodical reclustering. Allow complexity analysis, fast and simple
- Distributed and Mobility-Adaptive Clustering (DMAC) Algorithm
  - Same rules/procedures for clustering set up and maintenance, adaptive to nodes mobility and node/link failures

DCA: Distributed Clustering Algorithm, 1

- Assumptions
  - Knowledge of IDs and weights of one-hop neighbors
  - Broadcast transmission of a packet in finite time (a “step”)
  - Nodes do not move during clustering

DCA, 2

- (Only) Two messages:
  - CH(v): Sent by a clusterhead v
  - JOIN(u,t): Sent by ordinary node u when it joins the cluster of clusterhead t
- Three (simple) procedures:
  - Init (start up)
  - OnReceivingCH(v), OnReceivingJOIN(u,v) (message triggered)

DCA

- Ogni nodo conosce i suoi vicini ed il loro peso
- Un nodo è init se ha il peso più grande dei pesi dei suoi nodi vicini
- Gli init node diventano clusterhead e invitano i loro vicini a far parte del loro cluster
- Un nodo x aspetta di ricevere messaggi dai vicini di peso maggiore prima di prendere una decisione
  - Se un vicino di peso maggiore lo invita a far parte del suo cluster allora x entra a far parte del cluster del vicino di peso maggiore che lo contatta (inviando un messaggio di Join) → nodo ordinario
  - Altrimenti diventa clusterhead lui stesso e invia un messaggio di CH

DCA

- Due tipi di messaggi
  - CH(v) è usato da un nodo v per rendere consapevoli i suoi vicini del fatto che ha assunto il ruolo di clusterhead
  - JOIN(v,u) è usato dal nodo v per comunicare ai suoi vicini che sarà parte di un cluster il cui clusterhead è il nodo u

DCA

- Variabili
  - Cluster(v) indica l’insieme dei nodi che vanno parte del cluster di cui è clusterhead v
  - Clusterhead è una variabile che identifica il clusterhead del mio cluster
  - Ch(u) è vero quando o ha mandato un messaggio CH (u==v) oppure quando ha ricevuto un messaggio di CH dal nodo u
  - La variabile booleana Join (u,t) è vera se il nodo v ha ricevuto un JOIN(u,v) dal nodo u

DCA-Procedure (eseguite dal nodo v)

- Init

```
Se tutti i nodi vicini hanno un peso
   minore di v
   invia CH(v);
   Cluster(v)=Cluster(v)U{v};
   Ch(v)=true;
   Clusterhead=v;
```

DCA-Procedure (eseguite dal nodo v)

- On receiving CH(u)

```
Ch(u)=true;
Se i vicini di peso maggiore di v con
peso maggiore di u hanno tutti
mandato un Join, allora
    Clusterhead=u;
    invia JOIN(v,Clusterhead);
```

DCA-Procedure (eseguite dal nodo v)

- **On receiving JOIN(u,t)**
  Join(u,t)=true;
  Se v è un clusterhead allora se t==v
  Cluster(v)=Cluster(v)U{u};
  Se ho ricevuto Join da tutti i vicini più piccoli EXIT

Altrimenti si verifica se tutti i vicini di peso maggiore hanno preso una decisione sul ruolo.

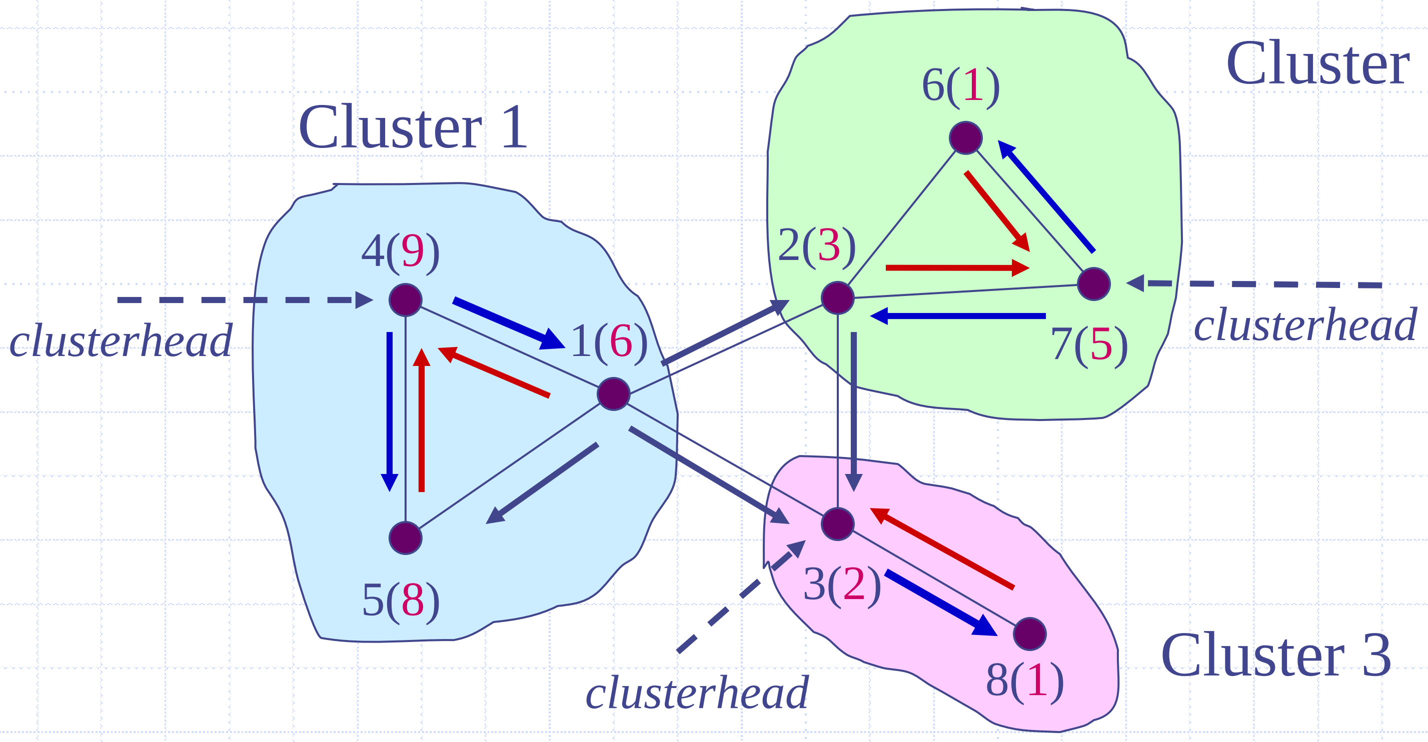
Se questo è il caso e tutti i vicini di peso maggiore hanno mandato JOIN

mandiamo un CH(v);
Cluster(v)=Cluster(v)U{v};
Clusterhead =v;
Se si è ricevuto JOIN da tutti i vicini minori EXIT.

Altrimenti se uno o più vicini di peso maggiore hanno mandato un CH

Clusterhead=il vicino di peso maggiroe con peso più grande tra quelli che sono diventati clusterhead e mi hanno invitato.
manda JOIN(v,Clusterhead);
EXIT;

Example



I Step　　II Step　　III Step　　IV Step　　V Step

DCA: Provable Properties

- Consider

$$\tau: V \rightarrow \{1,2,3, \ldots , 2k\}$$

V = set of network nodes, k = number of clusters

- **Proposition**: Each node v in V sends exactly one message by $\tau(v)$ steps
- **Corollary 1**: DCA message complexity is $n = |V|$
- **Corollary 2**: DCA terminates correctly in at most 2k steps ($<= 2n$)

A Note on the Average Time Complexity

- We notice that

$$k <= \alpha(G)$$

G = topology graph, $\alpha(G)$ = G's *stability number*

- We see the network as a *random graph*, for which

$$(2k <=) \; 2\,\alpha(G) = \text{circa } O(\log n)$$

Log's base is a function of n and the number of the network links

Adapting to Mobility and Node/Link Failures: DMAC

- DMAC is for clustering set up AND maintenance
- Nodes can move during the clustering
- Each node reacts to
  - Reception of a message
  - Presence of a new link
  - Link failure
- Same assumptions of DCA, plus knowledge of neighbors’ roles (no role = ordinary role)

DMAC: The Procedures

- INIT
- Link-dependent procedures:
  - Link_Failure
  - New_Link
- Message-triggered procedures:
  - OnReceivingCH(v)
  - OnReceivingJOIN(u,t)

Joining Clusterheads: Dynamic Backbone

- A theorem from Chlamtac and Farago:

  If a network is connected, and DCA is used, then if and only if each clusterhead is linked to all the clusterheads at most three hops away, the resulting backbone network is connected
- Inherently mobility adaptive and stateless
- Good if the random graph model could be used

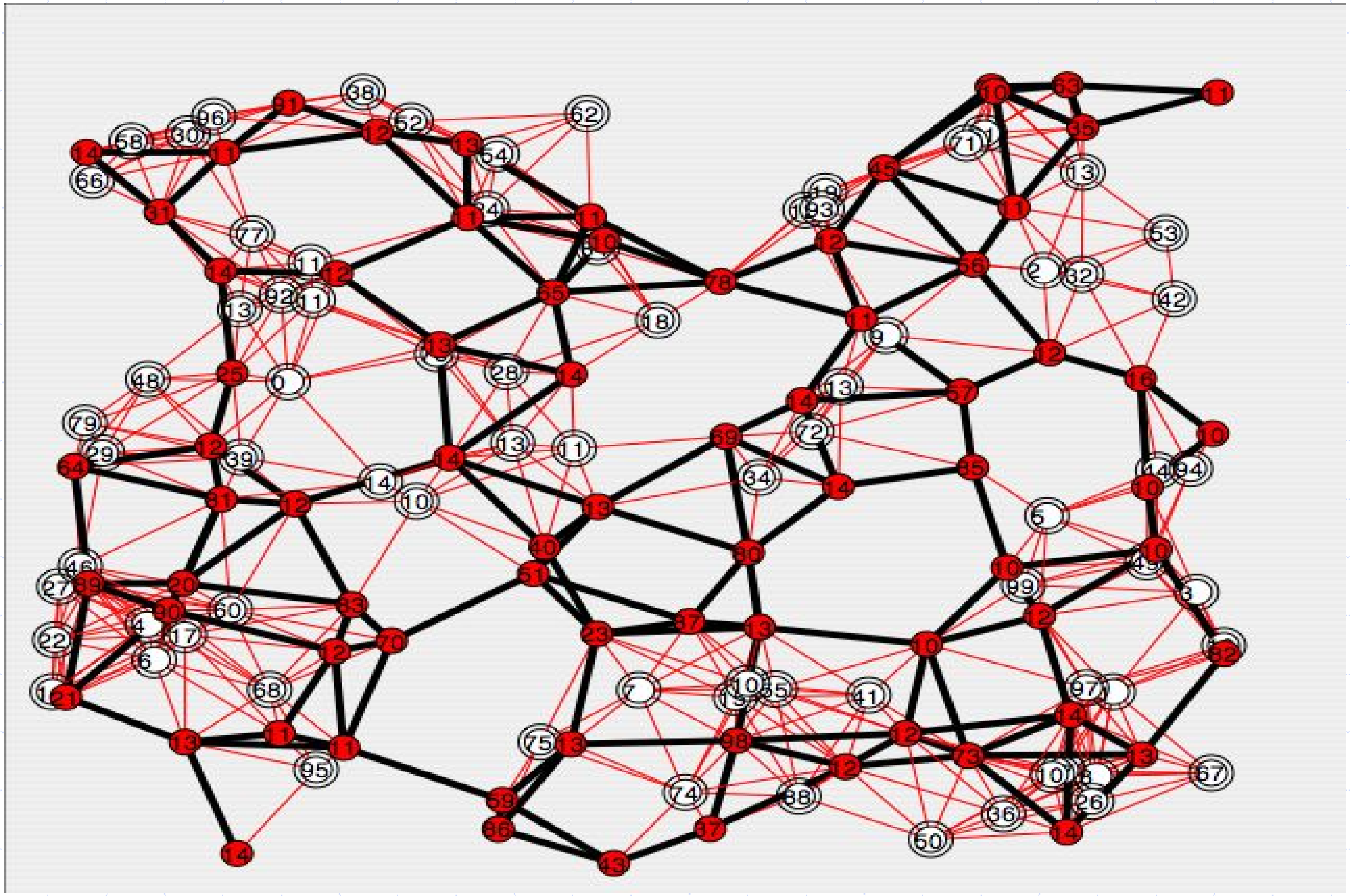
4 Backbone Formation Protocols

- 3 representatives of major approaches
  - Selection of independent set of nodes and backbone construction (DCA)
  - Rich dominating set formation and pruning (WuLi)
  - Two-phase algorithm with theoretical guarantees (WAF)
- 1 proposal after the performance comparison (DCA-S)

Distributed Clustering Algorithm (DCA)

- Distributed and localized implementation of the greedy for independent set
- Takes node status into account for node selection
- Independent nodes are joined into a connected backbone (connectivity is guaranteed) via gateways
- Low degree of parallelism ("dependency chains")

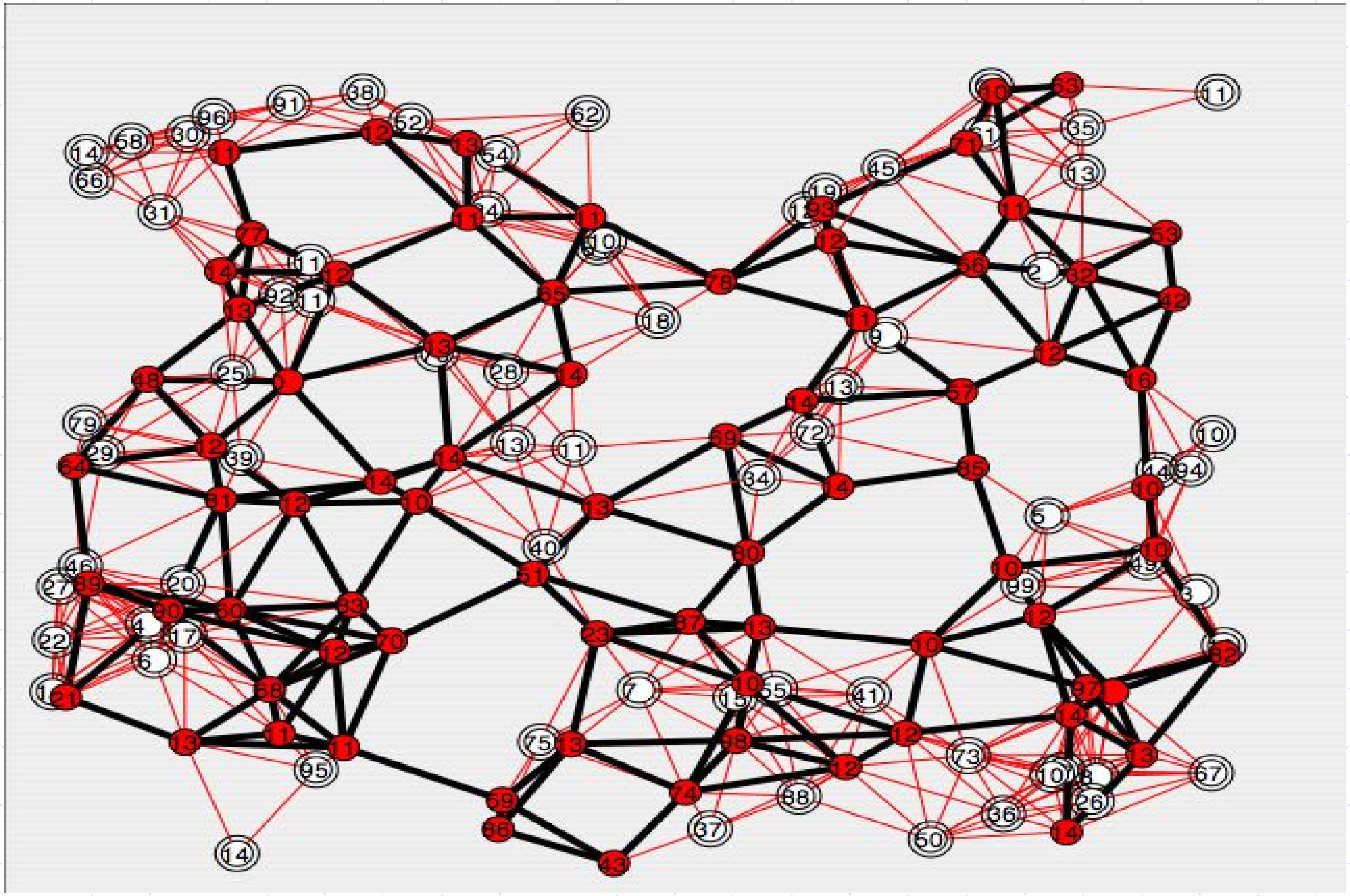
A DCA Backbone



WuLi: Wu and Li protocol

- Distributed and localized protocols for forming a connected dominating set
- Build a rich connected dominating set
- Applies localized rules for pruning unnecessary nodes/links
- High degree of parallelism ("all localized")

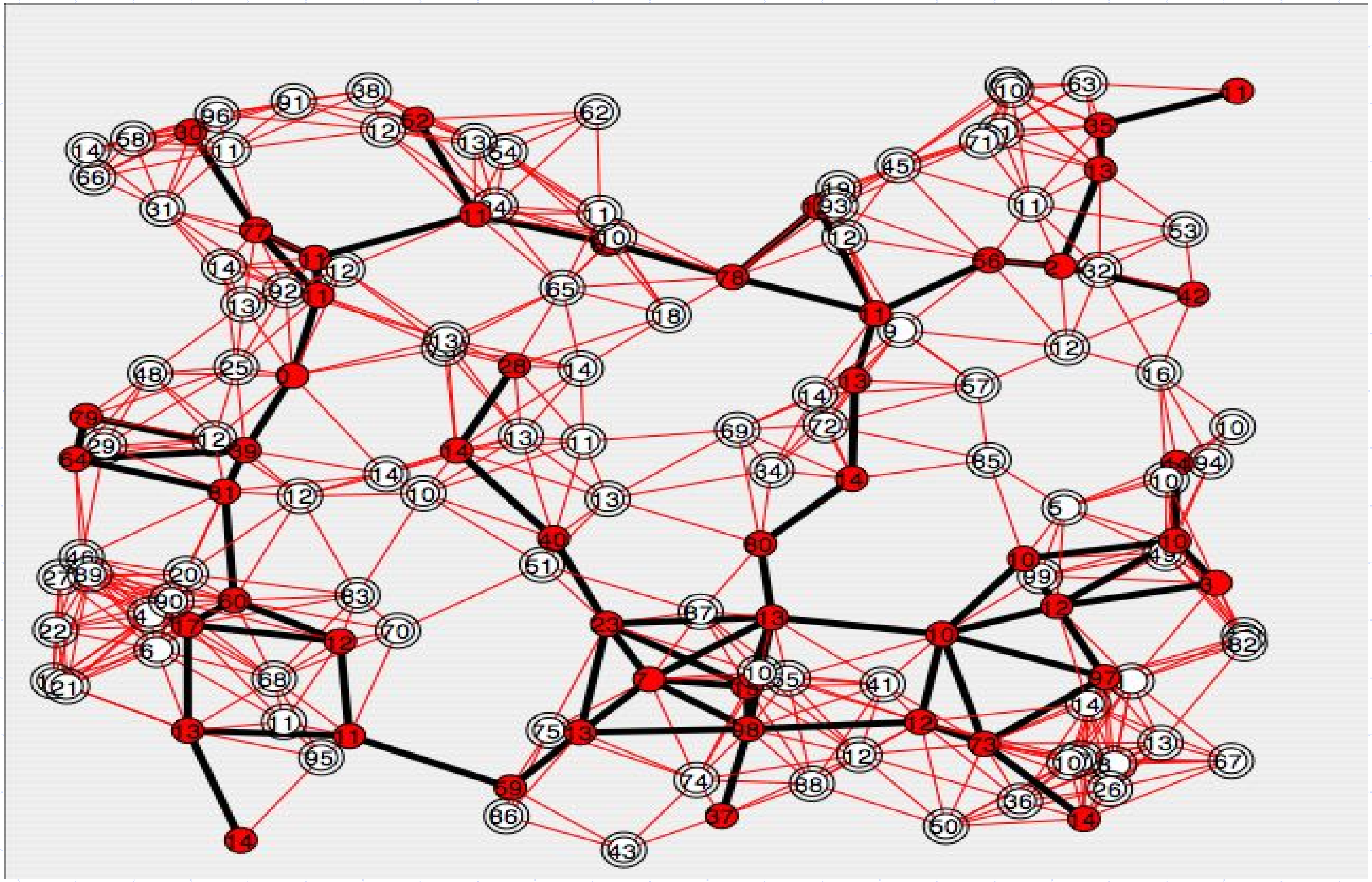
A WuLi Backbone



WAF: Wan, Alzoubi and Frieder

- Two phases
  - Leader election: One node is chosen among all network nodes to be the root of a tree
  - Nodes at different levels of the trees can be chosen to form a connected dominating set
- The “leader election tree” is quite expensive
- Very low degree of parallelism

A WAF Backbone



DCA-S: DCA Sparsified

- Build a connected dominating set (say, with DCA) and consider its spanned sub-graph H (include gateways)
- Erdös: If a graph does not have small cycles then it is sparse
- Find and break small cycles (small=log n)
  - In practice we search and break cycles with 3 and 4 links
- Breaking cycles does not compromise connectivity

Simulation Results

- Metrics (all averages)

1. Protocol duration
2. Operation overhead (in bytes)
3. Energy consumption (per node)
4. Backbone size
5. Route length
6. Backbone robustness (node deaths for disconnections)

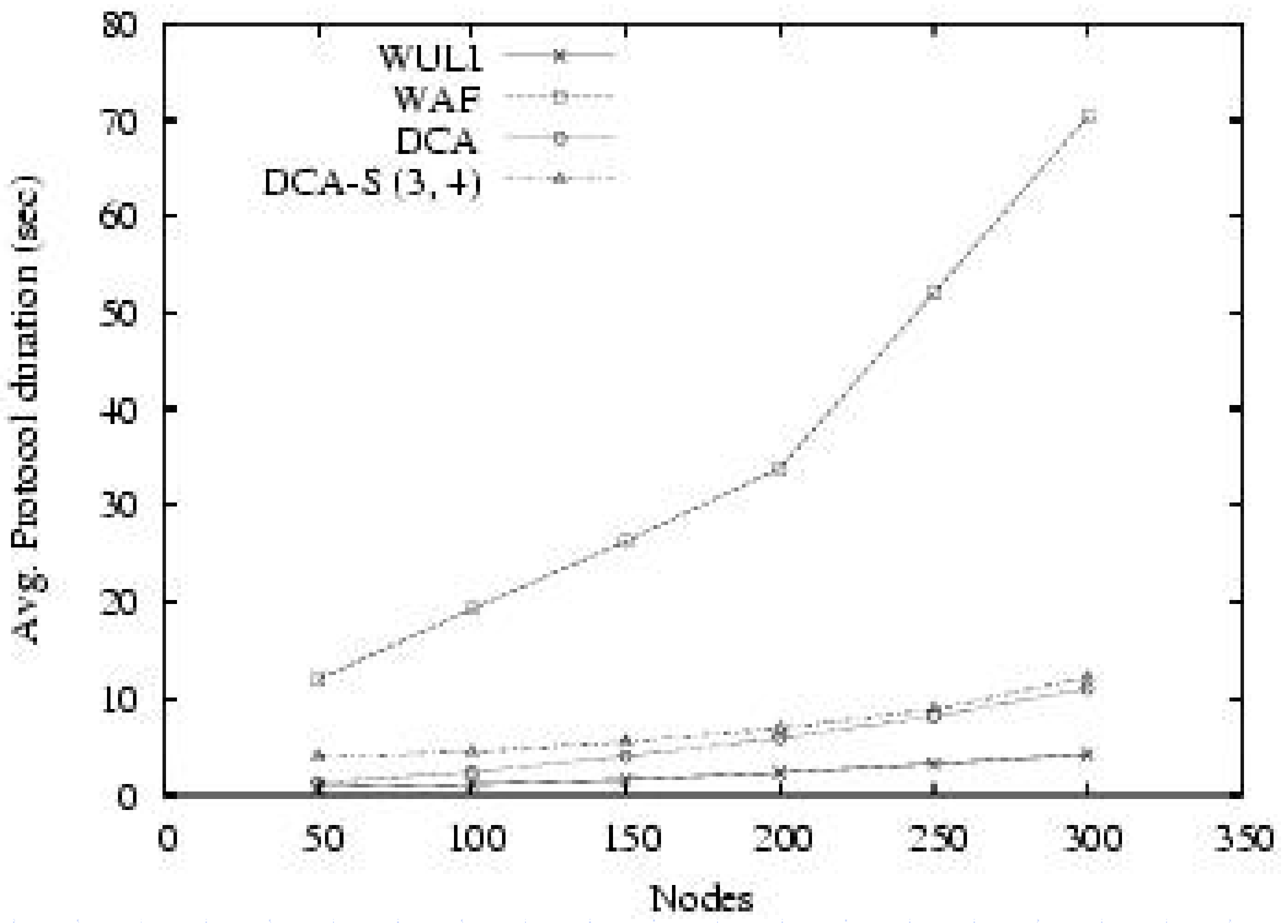
Simulation Results, 2

- Parameters of ns2-based simulations
  - Nodes: ≤ 300, IST EYES prototype
    - Tx range: 30m
    - Initial (residual) energy: 1J
    - Tx, Rx, idle power: 24, 14.4, 0.015 (mW)
  - Area: 200 x 200m
  - Six scenarios with increasing densities (avg. degrees: 3.5 to 20)

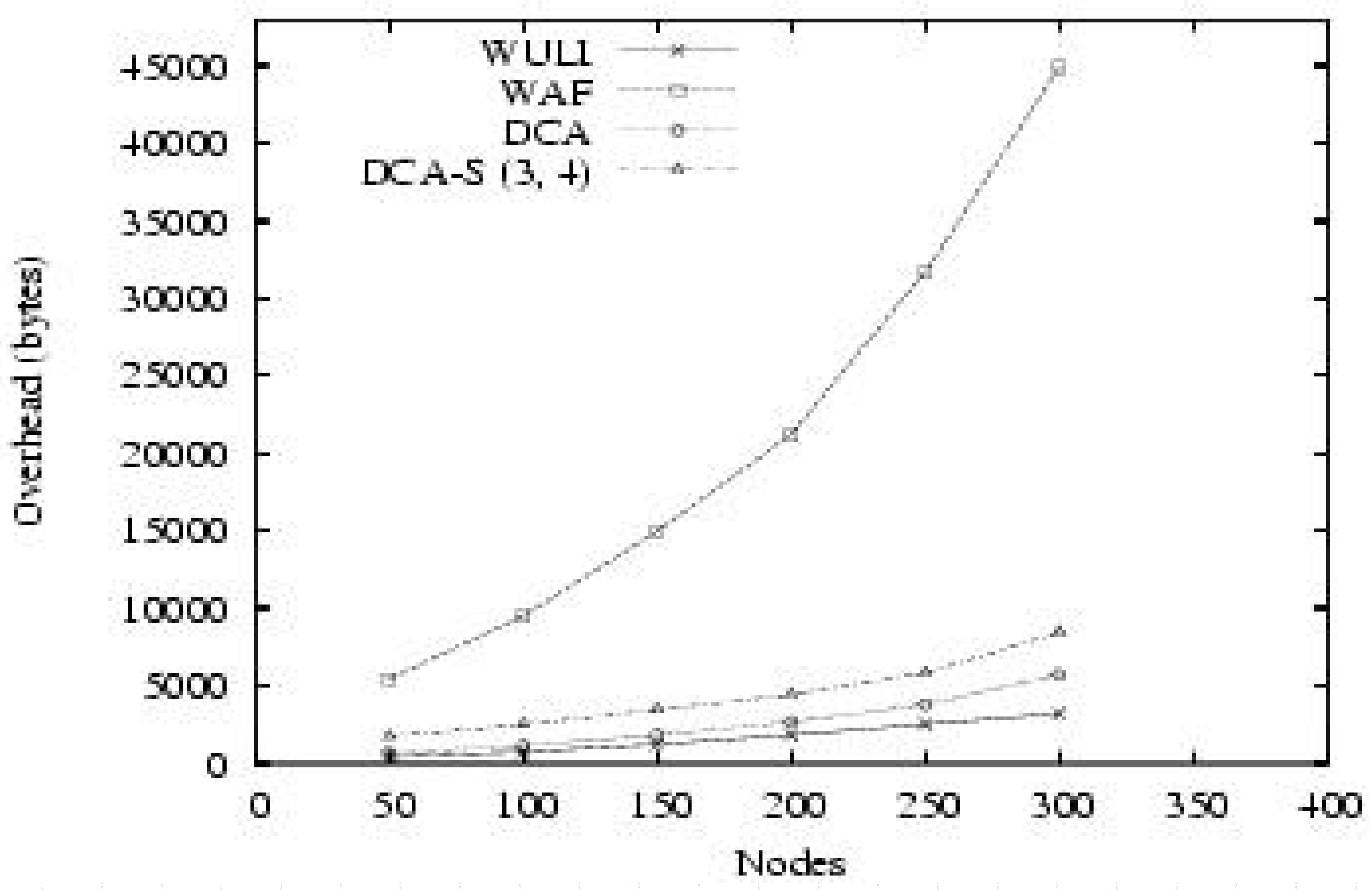
Protocol Duration

- WuLi is fastest
  - Simple operation; parallelism
- DCA: Reasonably fast
  - Possible dependencies and gateway selection
- DCA-S: As DCA
  - The sparsification phase is executed by fewer nodes and requires little info exchange
- WAF: Slower
  - Non-trivial leader election

Protocol Duration, 2



Protocol Overhead



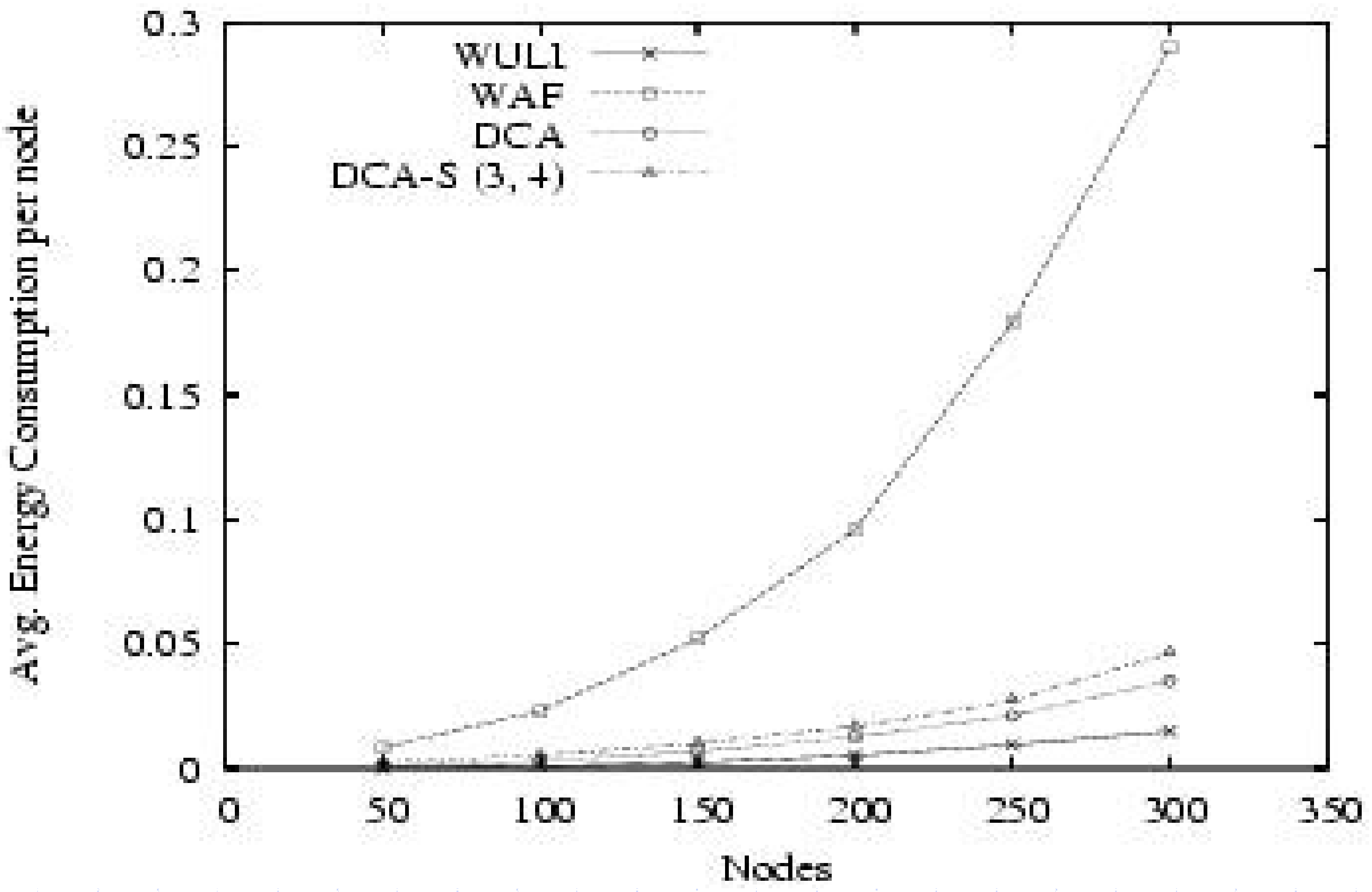
Protocol Overhead, 2

- Average number of protocol bytes per node
- WuLi: Best performing
  - Simple list exchange
- DCA(-S): Almost twice as much
  - Bit more info needed (weight, IDs, …)
- WAF
  - Leader election complexity

Energy Consumption

- Important metric per backbone set up and maintenance
- Similar to overhead results
- WuLi and DCA perform quite well
- DCA-S performs similarly: No difference in breaking cycles with 3 or 4 links
- WAF: High consumption due to first phase

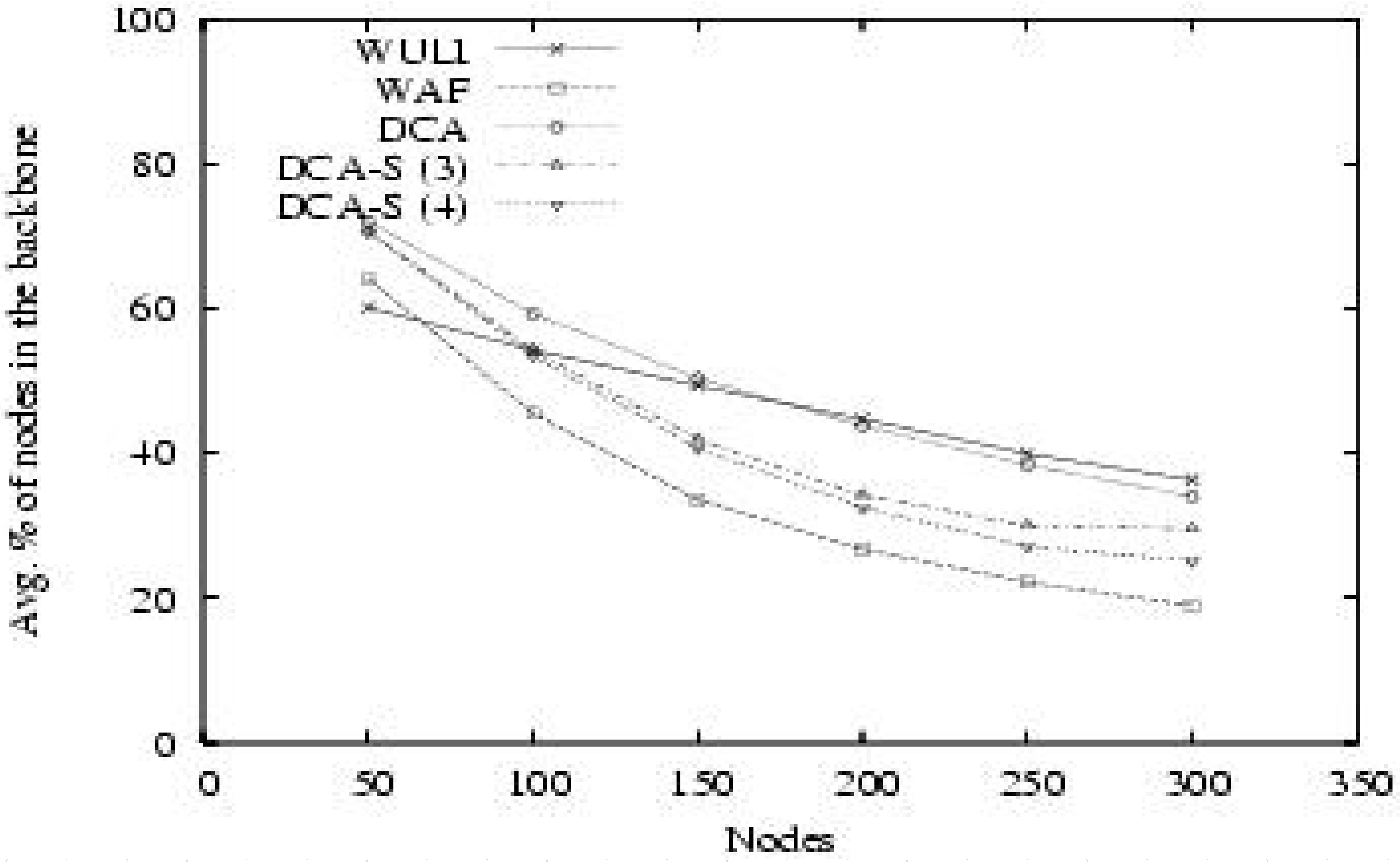
Energy Consumption, 2



Backbone Size

- Important metric: Aggregation and awake/asleep cycles
  - Small backbone + role rotation: key for WSNs
- Decrease with n increasing (bigger clusters)
- WAF: “Slimmer” backbone (tree like)
- DCA-S, 4 > DCA-S, 3 > DCA > WuLi

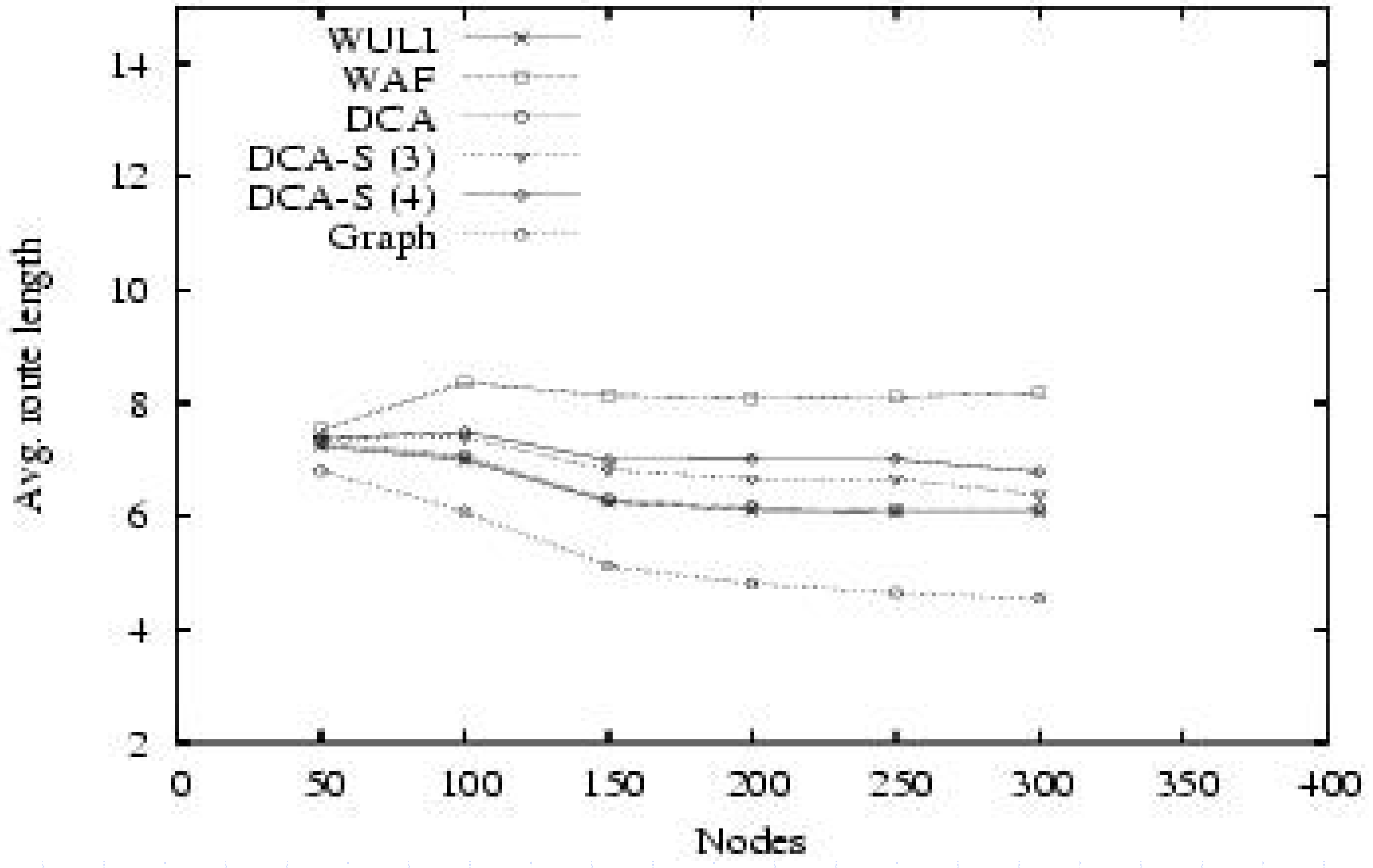
Backbone Size, 2



Route Length

- Flat topology (“visibility graph”) as a base
- Expected increase: Hierarchy routes are longer
- DCA & WuLi: 7 to 34.7% longer routes
- DCA-S: Up to 9% more than DCA
- WAF: Up to 33.4% longer than DCA

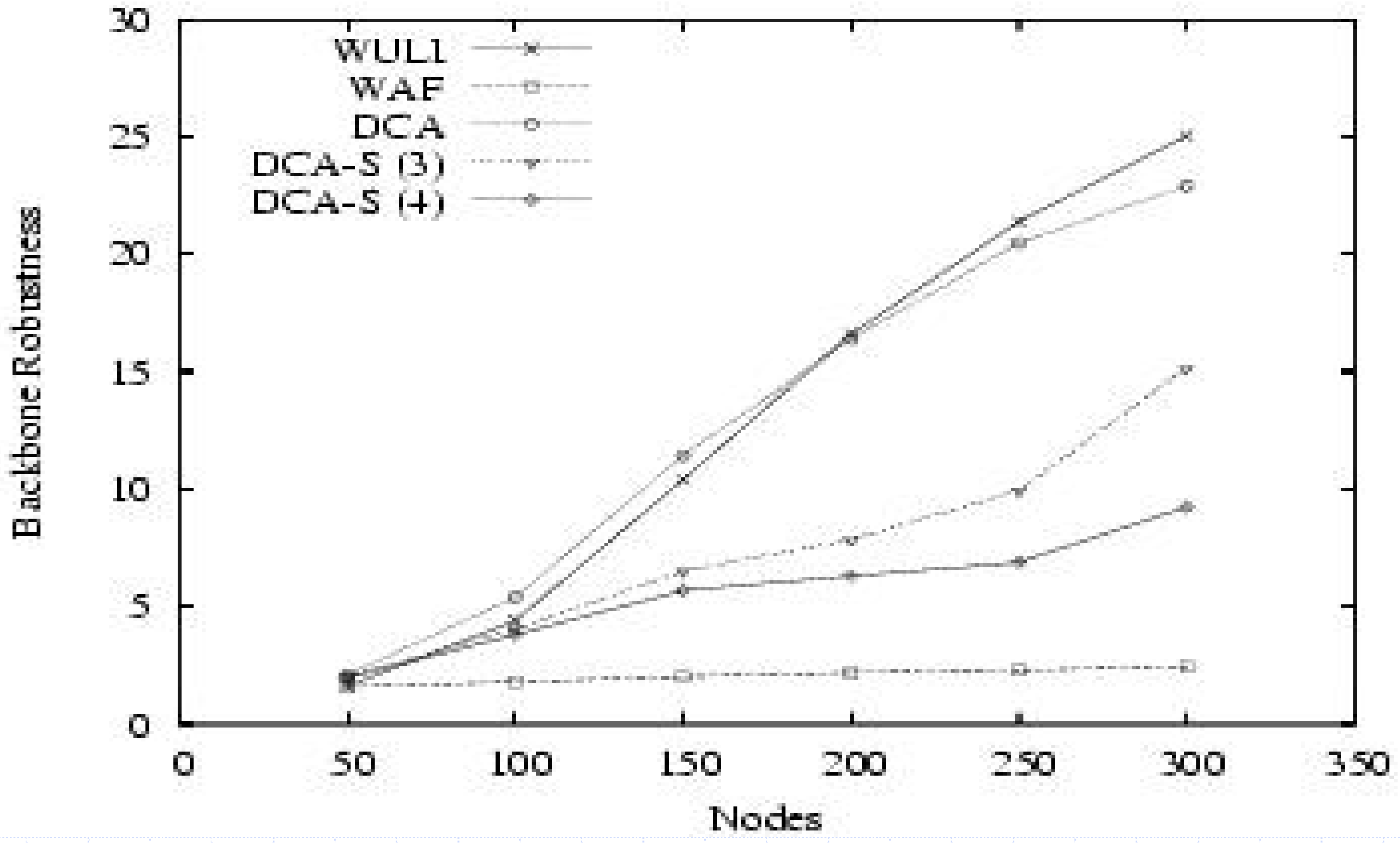
Route Length, 2



Backbone Robustness

- Number of nodes needed to disconnect the backbone
- Useful for planning backbone re-orgs
- Increases with network density
- WuLi and DCA: More robust
  - Resilient to up to 25 “death” when n = 300
- WAF: Quite a disaster (tree-like topologies)
- DCA-S: In the middle

Backbone Robustness, 2



"To Go"

- Hierarchical organization is effective for prolonging network lifetime
- Four protocols for backbone formation:
  - DCA, WuLi, WAF and DCA-S
- Nice theoretical features → hard to implement
- Simple solutions (WuLi, DCA): Good starting point for efficient implementations
- DCA-S: "Slimmer" backbone at a reasonable cost