Routing

Sensor Networks

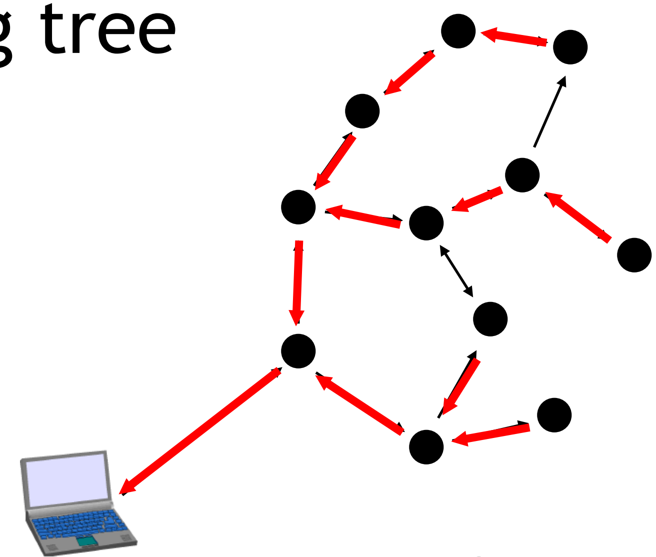
Prof. Dr. Kay Römer

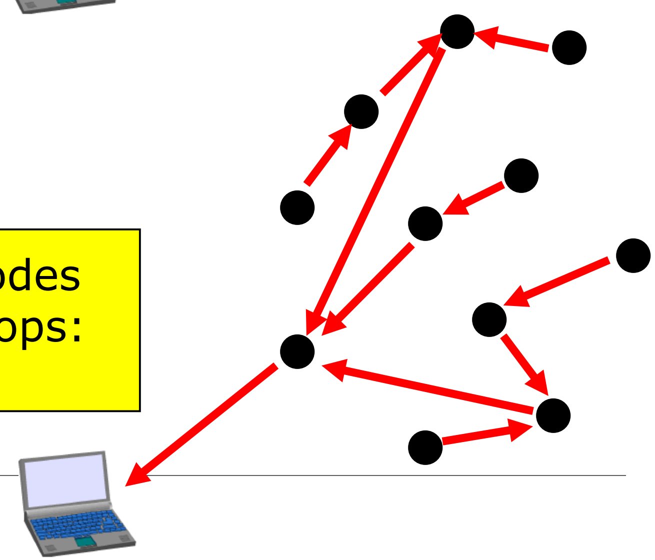
Routing in Sensor Networks

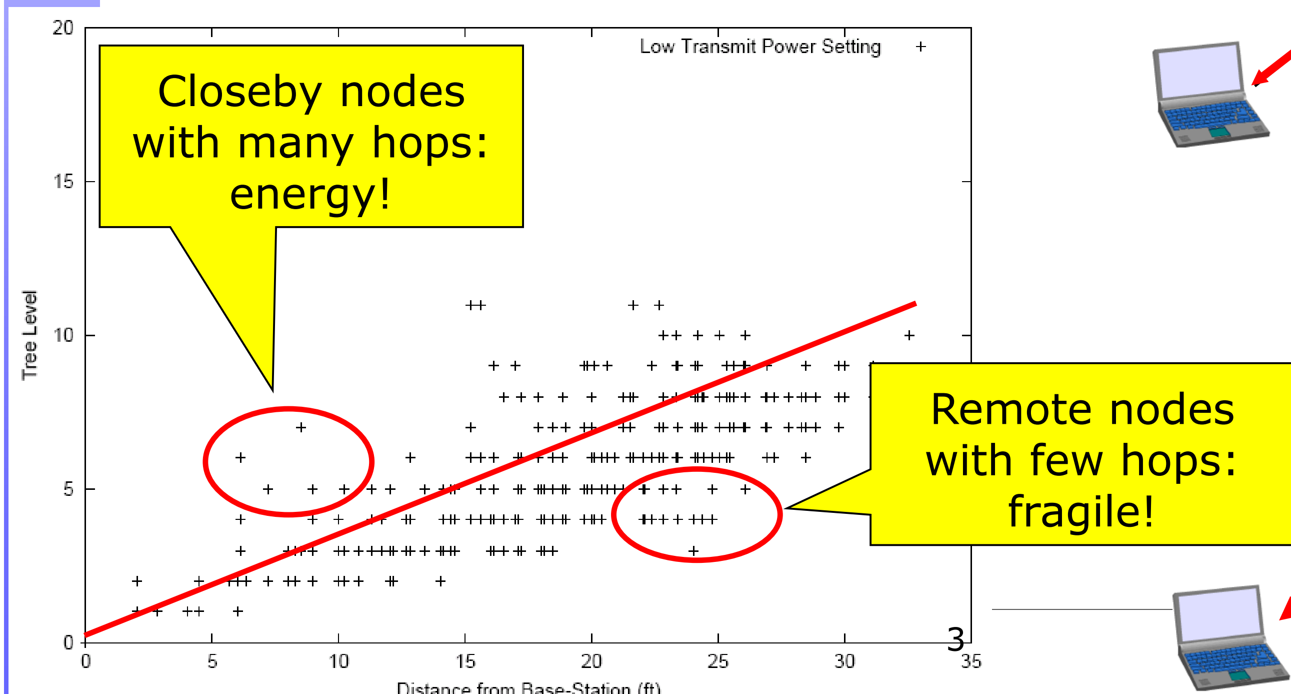
- Traditional Networks
  - Typically based on addresses
  - Unicast, multicast
- Sensor Networks
  - Convergecast (all nodes to sink)
    - Data collection
  - Local interaction
  - Flooding (sink to all nodes)
    - Code/task distribution
  - Geo routing

Convergecast

- Typically based on spanning tree rooted at the sink

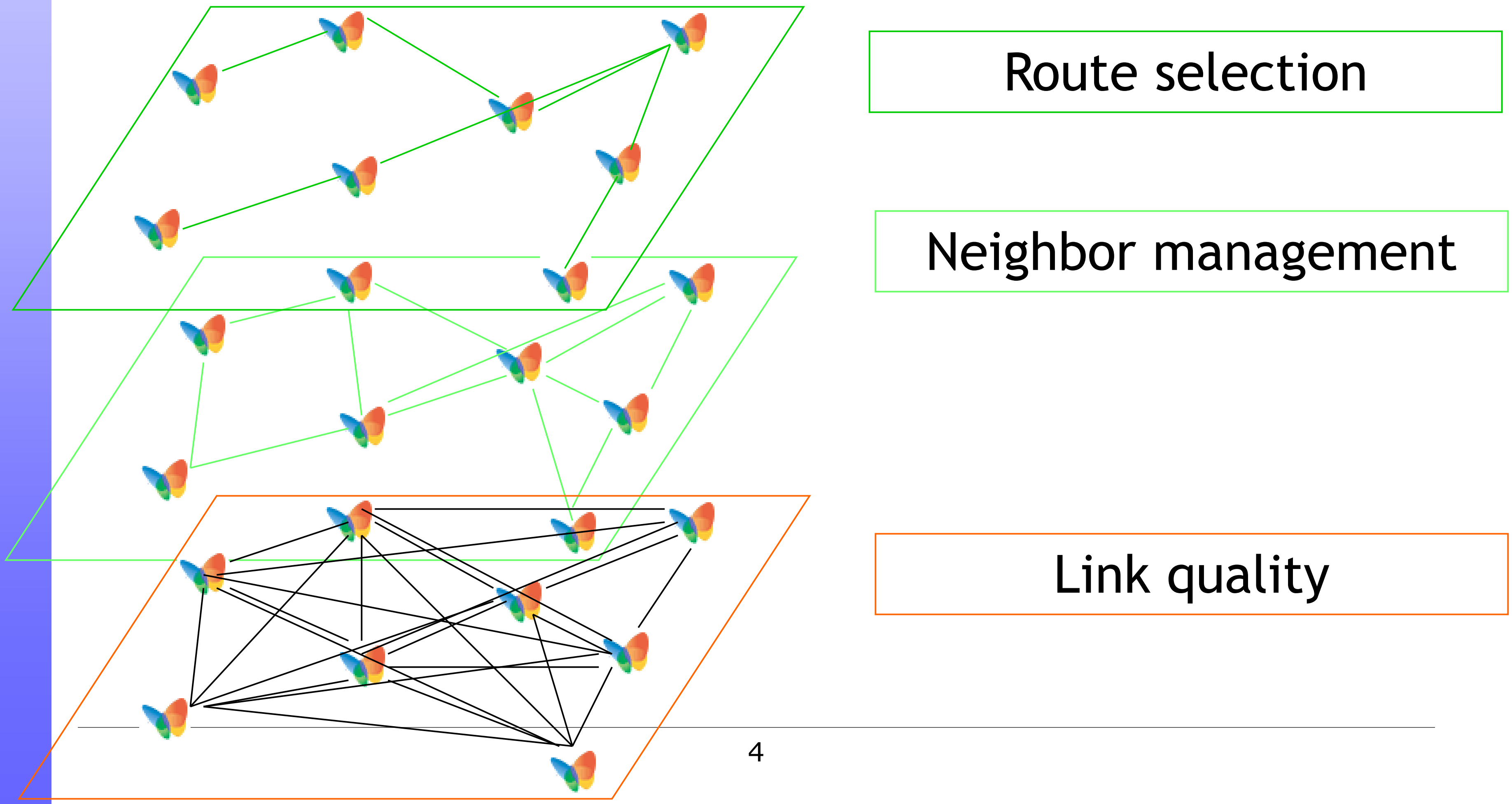






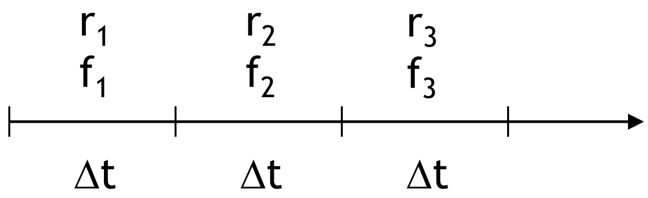
Good Spanning Trees

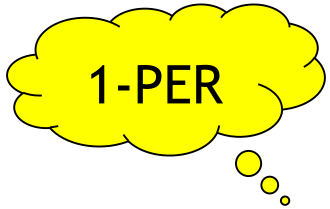
- How to build a good spanning tree?
  - Good neighbors: low packet loss
  - Stable neighbors: infrequent changes



Link Quality

- Estimation of packet delivery rate
  - Cf. Chapter „Physical Layer“





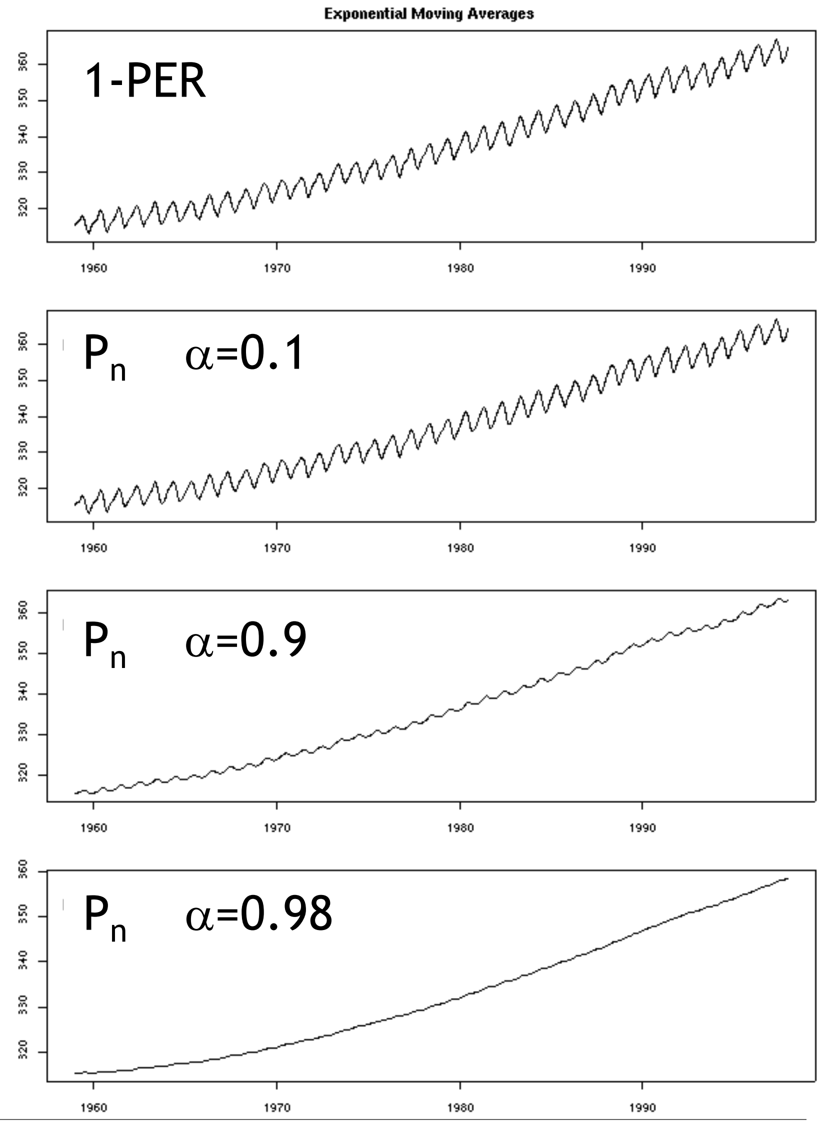
$$P_n = \alpha P_{n-1} + (1-\alpha)\frac{r_n}{r_n + f_n}$$

r_n: received packets in interval

f_n: packets identified as lost

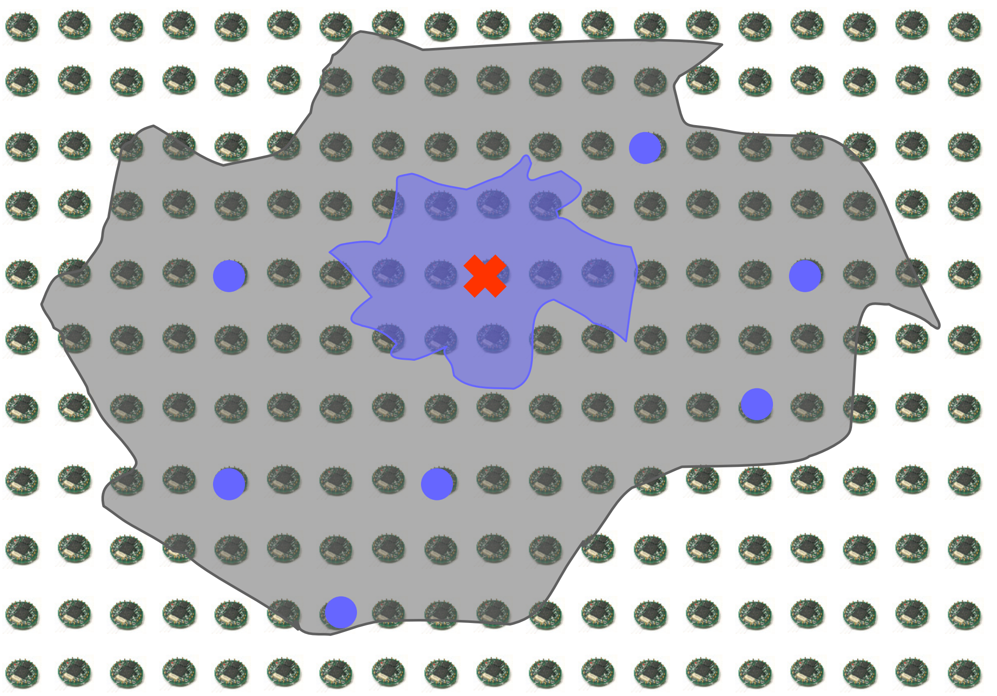
EWMA

$$P_n(t) = \alpha\, P_{n-1} + (1-\alpha)\,(1 - PER)$$



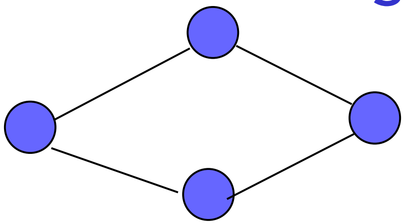
Neighbor Management

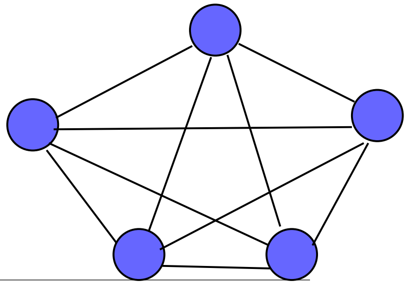
- Dense sensor networks
  - Many neighbors (>200)
  - Many bad (grey)
  - Few good (blue)
- How to pick out good neighbors?
  - Appears to require state information for each neighbor
    - Memory!
  - Typically neighbor table with fixed size T
- How to efficiently find best T neighbors with small memory?



T?

- What should be the size of the neighbor table?
  - Connected network!
- Xue and Kumar (2002 und 2004)
  - For almost certainly connected network Θ(log n) neighbors necessary and sufficient
    - T < 0.074 log n: almost certainly not connected
    - T > 5.1774 log n: almost certainly connected
  - In practical networks (n < 1000): T = 6-10
- Penrose (1999)
  - With T neighbors, there are T disjoint paths between any pair of nodes with high probability
    - P -> 1 for n -> ∞





Picking Good Neighbors

- Assumptions
  - Unknown number N of neighbors
  - Neighbor table with size T
  - Nodes periodically broadcast «hello» beacons with sender address
- Approach
  - Table should contain nodes from which most «hellos» have been received
- Upon reception of «hello» from node n
  - n already in table?
    - Reinforce
  - Else, should we insert n?
    - Insertion criteria
  - If yes, which other node should be removed?
    - Removal criteria
  - Cf. cache management
    - FIFO, LRU, ...

Insertion, Removal, Reinforcement

- Goal: table should always contain nodes from which most «hellos» have been received
  - Doesn't this require O(N) memory?!
  - How to pick the T most frequent senders with memory O(T)?

Picking Frequent Neighbors

- Candidate n, counter C=0
- Upon reception of «hello» from “sender“
  - If C>0 and n=sender
    - C++

    Reinforce
  - If C=0
    - n := sender; C := 1

    Insertion
  - Else
    - C--

    Removal
- Result: Majority candidate
  - C=0: For each increment there is a decrement – there is no majority element with frequency > ½
  - C>0: n only majority candidate (!)
- Works only if one node dominates all others!
  - Practically n is a good approximation of the most frequent element

Picking Frequent Neighbors

- T counters <n, C>, initially <0, 0>
- Upon reception of «hello» from «sender»
  - Does counter <sender, C> exist with C>0?
    - Increment C by 1
  - Otherwise, free counter <x, 0>?
    - Set to <sender, 1>
  - Else
    - Decrement ALL counters by 1
- Result: All candidates for > P/(T+1) received «hellos» out of P «hellos»
  - All entries <n, C> with C>0
  - Cf. „Frequency Estimation of Internet Packet Streams with Limited Space“, E.D. Demaine et al

Stable Neighbors

- Table does now contain at any point in time neighbors with many received «hellos»
  - These neighbors are probably good
  - But: neighbors may change frequently -> not stable
- Modified insertion
  - Insert new neighbor only with probability P = T/N
    - Why does this help?
  - N is unknown!
    - Counting appears to require memory O(N)?!

Estimating Number of Neighbors

- Algorithm
  - Stream of hellos with sender s
  - Uniform hash function h: s -> [1, M]
  - r(i) = Number of 0's at end of bin(i)
  - R = max { r(h(s)) }
  - $N = 2^{R+1}$
- Why does this work?
  - r(h(n)) = k expected for $1/2^{k+1}$ of all neighbors
    - $Prob[r(h(n))=k] = 1/2^{k+1}$
  - As R is the maximum of all k, we can expect 2^{R+1} neighbors
  - It can be shown that
    - $E[1.2928... \times 2^{R+1}]$ = true number of neighbors
  - Cf. „Probabilistic Counting Algorithms for Data Base Applications“, P. Flajolet et al

Prob[h(s)=i] = 1/M

...1	– ½ of all integers
...10	– ¼ of all integers
...100	– 1/8 of all integers
...	– ...

Stable Neighbors

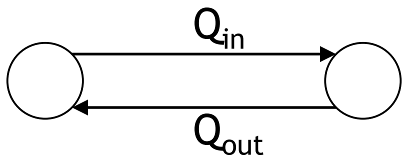
- We can decide with memory O(1) if a new neighbor should be inserted
  - Throw asymmetric coin with P[heads] = T/N

Conclusion

- Each node does now have a stable set of good neighbors
  - Note: the link quality (packet reception rate) is only estimated for the nodes in the table
- How to construct a spanning tree?

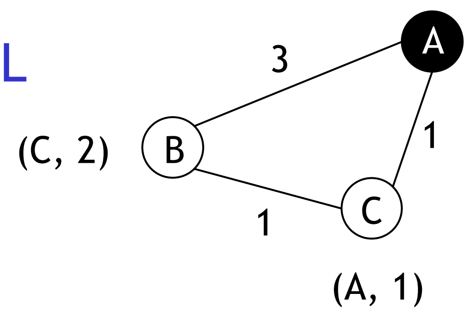
Routing: Good Links

- Foundation for Routing: good links
  - „good“ link = link with low packet loss
  - Both directions relevant: packet + ACK!
  - Routing metric: $m(L) = 1 / Q_{in}(L) \times 1 / Q_{out}(L)$
    - Number of expected transmissions (ETX)
    - Small values are better
- Links often asymmetric: $Q_{in}(L) \,!= Q_{out}(L)$
  - Each node only knows quality of incoming links
  - Broadcast link qualities to neighbors periodically



Spanning Tree

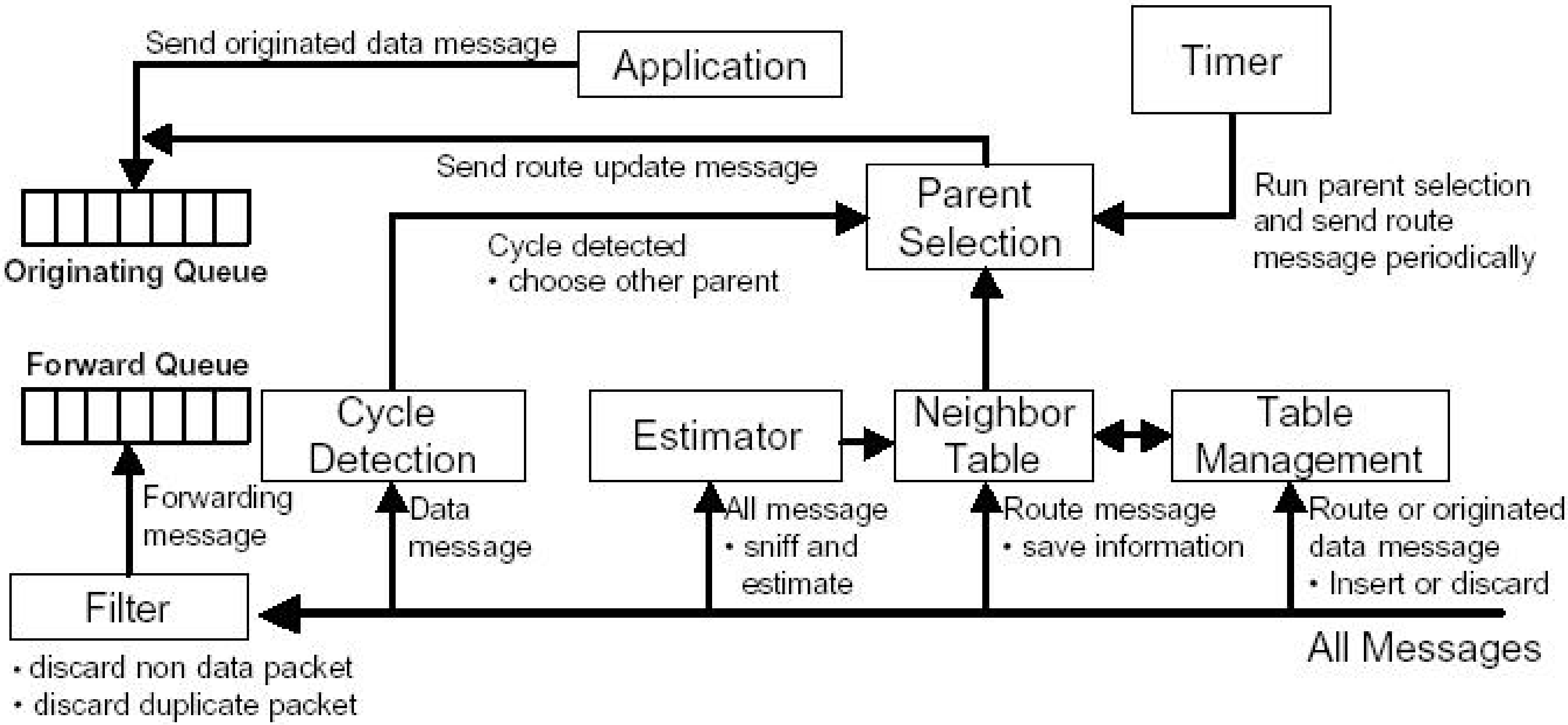
- Good tree = good path from each node to sink
  - „Good“ path = sequence of good links $L_1, \ldots, L_i$
  - Formally: find shortest path w.r.t routing metric
    - min $\sum m(L_i)$
- Approach: Distance Vector Routing
  - Each node records shortest distance D to sink and current parent V
    - D: At sink initially 0, otherwise ∞
    - V: Initially «-»
  - Update: Nodes periodically broadcast beacon P containing their distance to sink
    - Also sink with D=0
  - Neighbor receives P from node S via link L
  - If P.D + m(L) < D
    - D = P.D + m(L)
    - V = S
    - Broadcast update



Stability, Cycles, Fairness

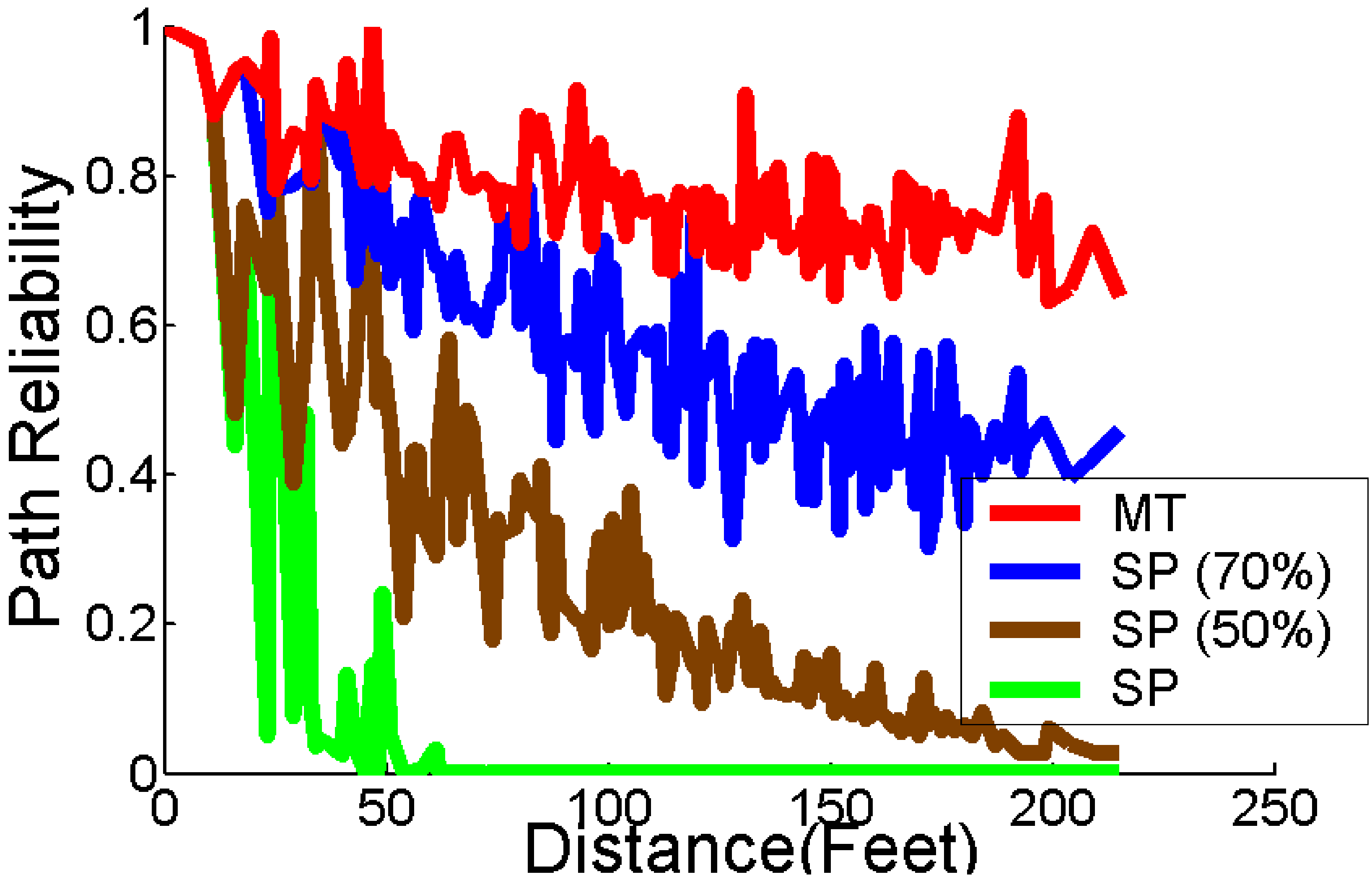
- Tree should be stable -> change parent infrequently
  - Periodic updates rather than immediately after receiving new information
- Cycle detection
  - Node receives packet it sent earlier
  - Change parent
- Fairness
  - Separation of locally generated and forwarded packets
  - Locally generated packets have priority

System Architecture

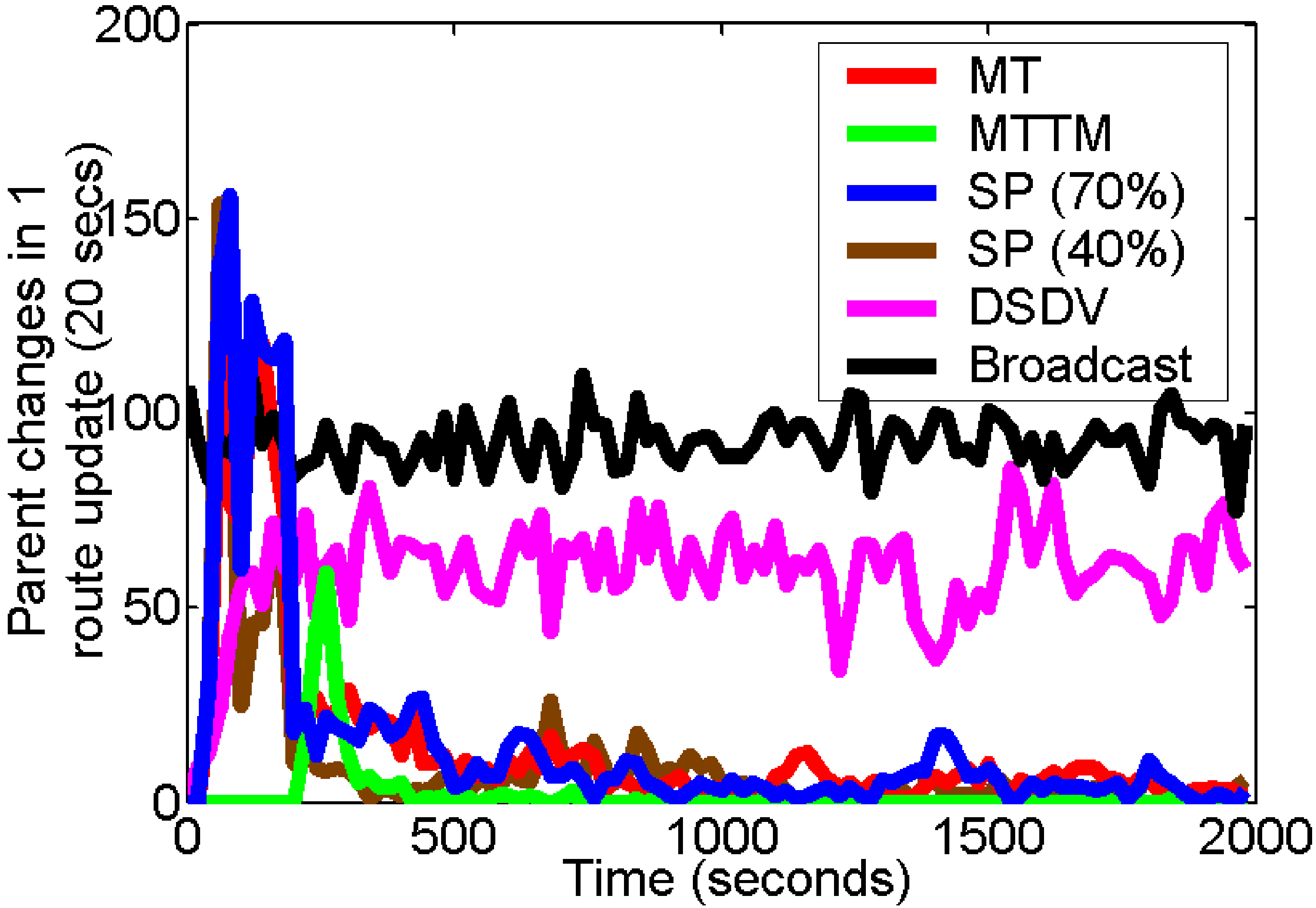


- Good spanning trees are hard to obtain!

Path Quality



Path Stability

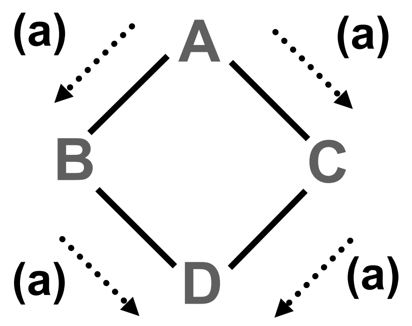


Local Interaction

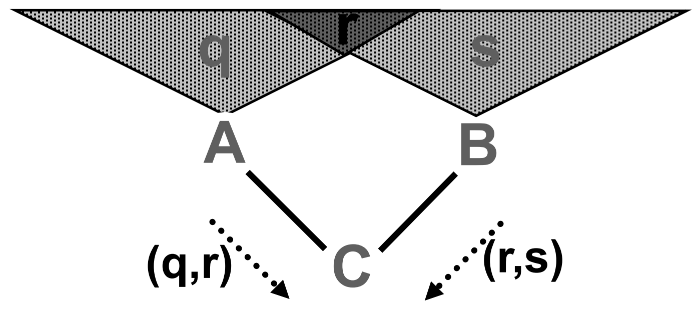
- Flooding with limited hop distance r
  - Sender: broadcast packet with distance r
  - receiver: If r > 0 and message not forwarded earlier:
    - Rebroadcast with distance r-1

Flooding: Problems

- Implosion
  - Same message received over multiple paths

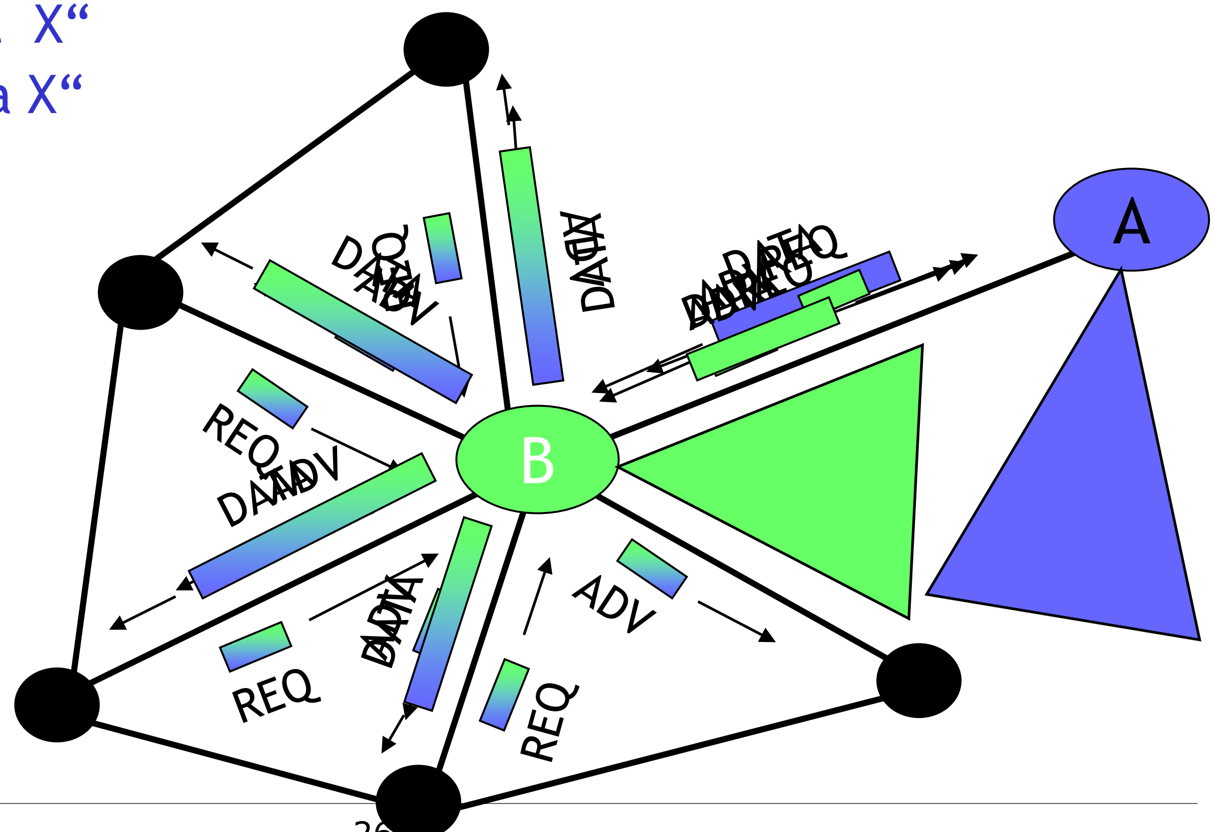


- Overlap
  - Different messages containing overlapping sensor data (multiple nodes observing same phenomenon)



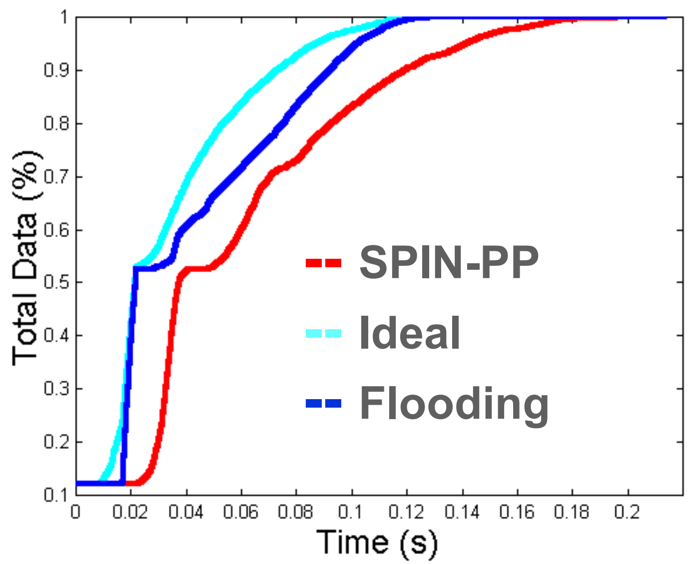
SPIN

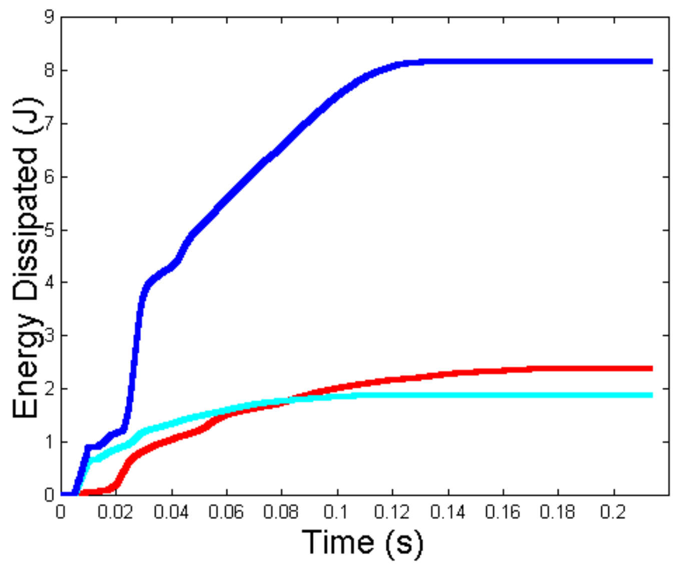
- Assumption
  - Large payload
- Advertisements
  - ADV: „Have X“
  - REQ: „Want X“
  - DATA: „Data X“
- Variants
  - SPIN-PP: Point-to-Point
  - SPIN-BC: Broadcast
  - SPIN-EC: Energy-aware



SPIN Performance

- Setup
  - 25 nodes
  - Every node has 3 data items, randomly chosen from 25 possible items
  - ADV/REQ: 16 Bytes
  - DATA: 500 Bytes



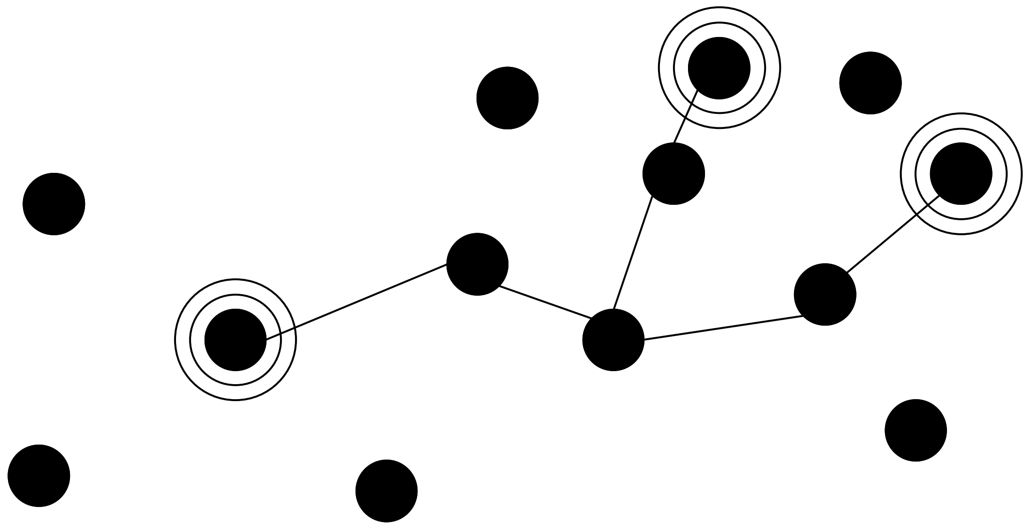


Network Flooding

- Sink to all nodes
  - New task / program
- Multiple options
  - Reverse spanning tree
    - Reliability?
  - Global flooding
    - Efficiency?

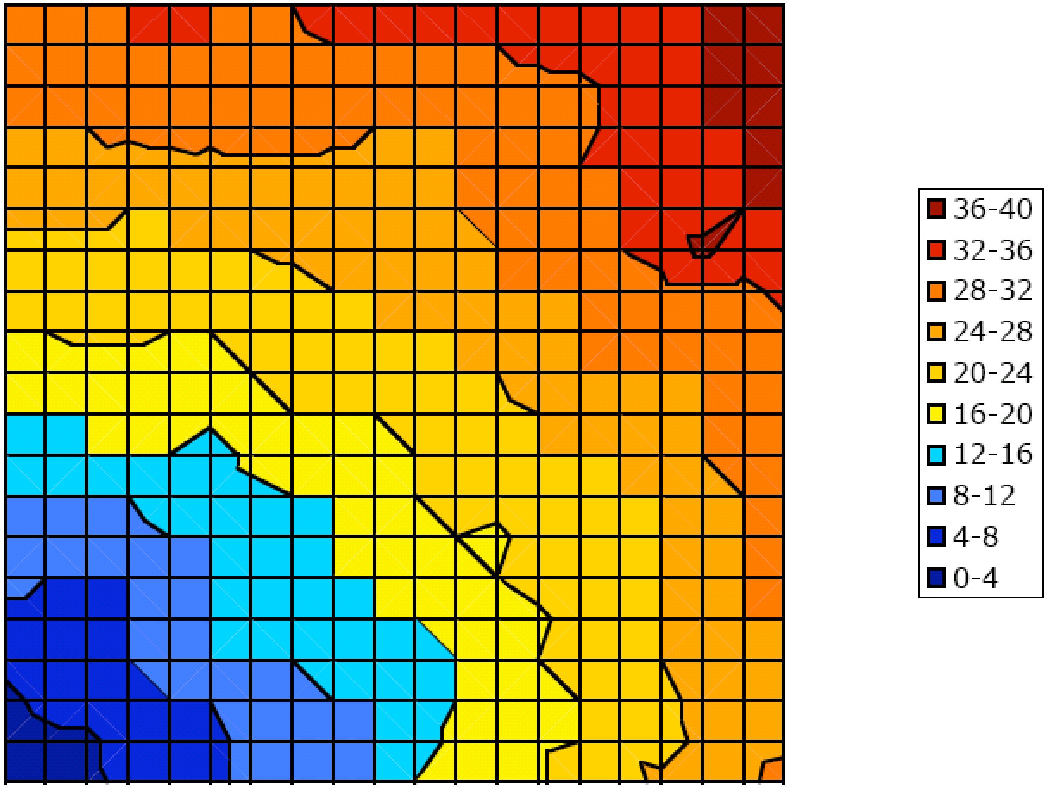
Fire Cracker

- Combination of spanning tree and flooding
  - Route message to some (remote) nodes
  - Flood from there
- Efficiency of spanning tree and reliability of flooding



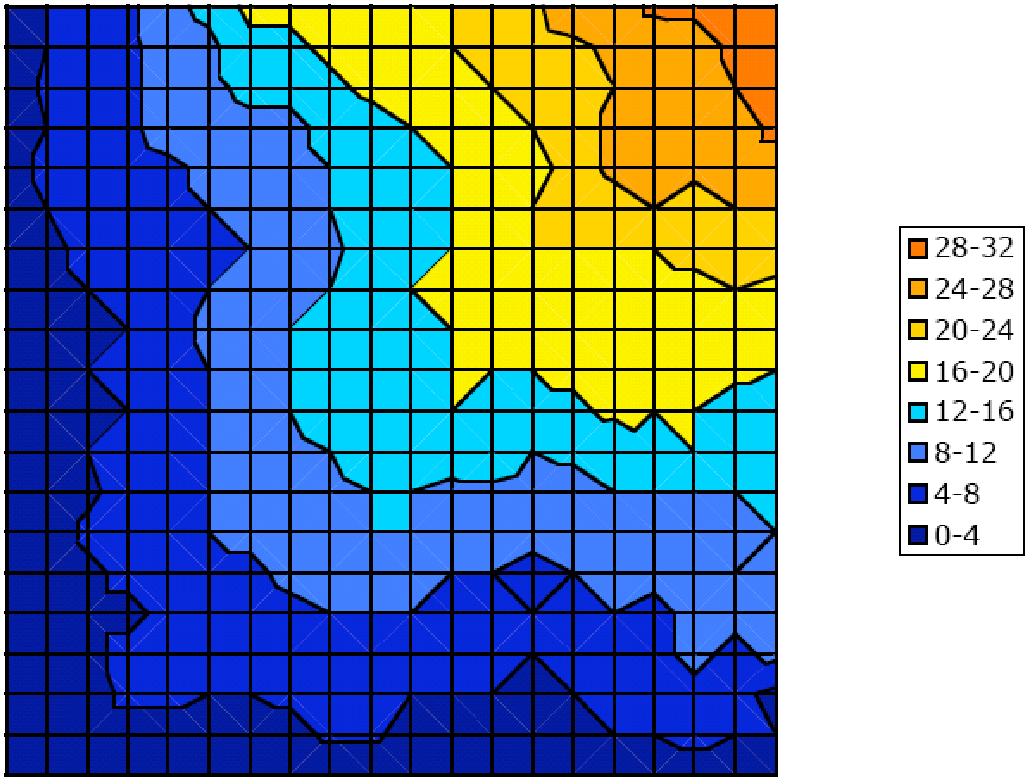
Flooding

- Trickle
  - Flooding with advertisements (cf. SPIN)
  - CSMA + BEB

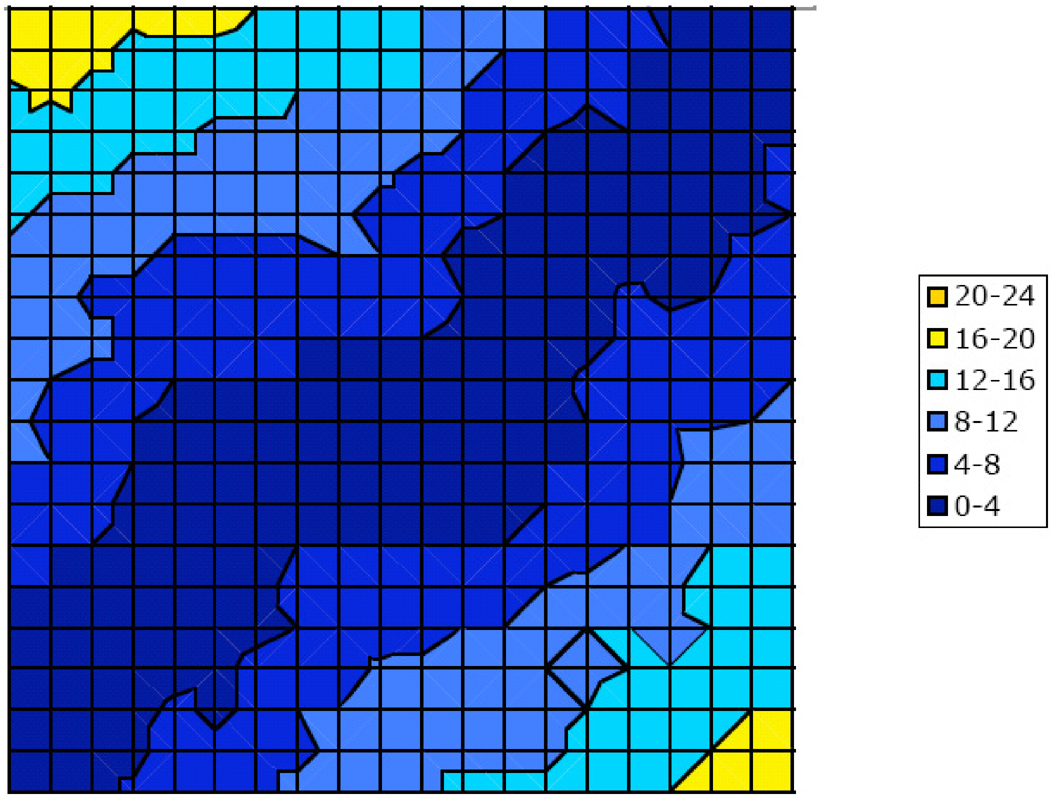


Flooding from 3 Corners

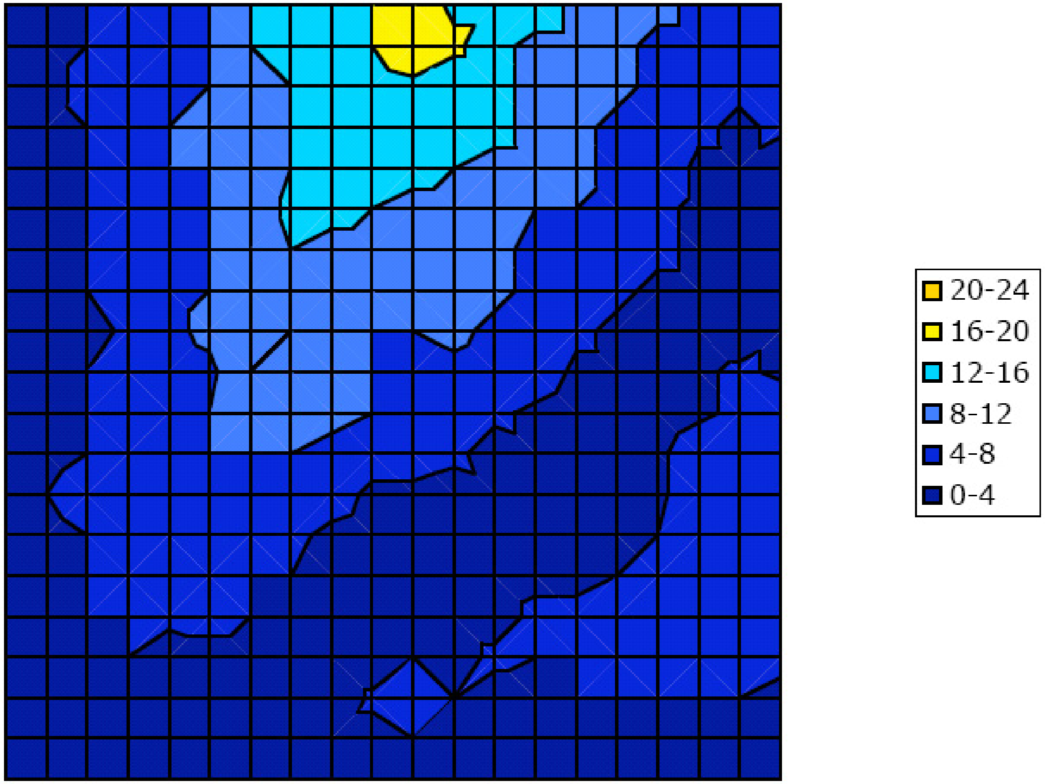
- Including routing to the corners!
- Nodes overhearing packet during routing start flooding after fixed time



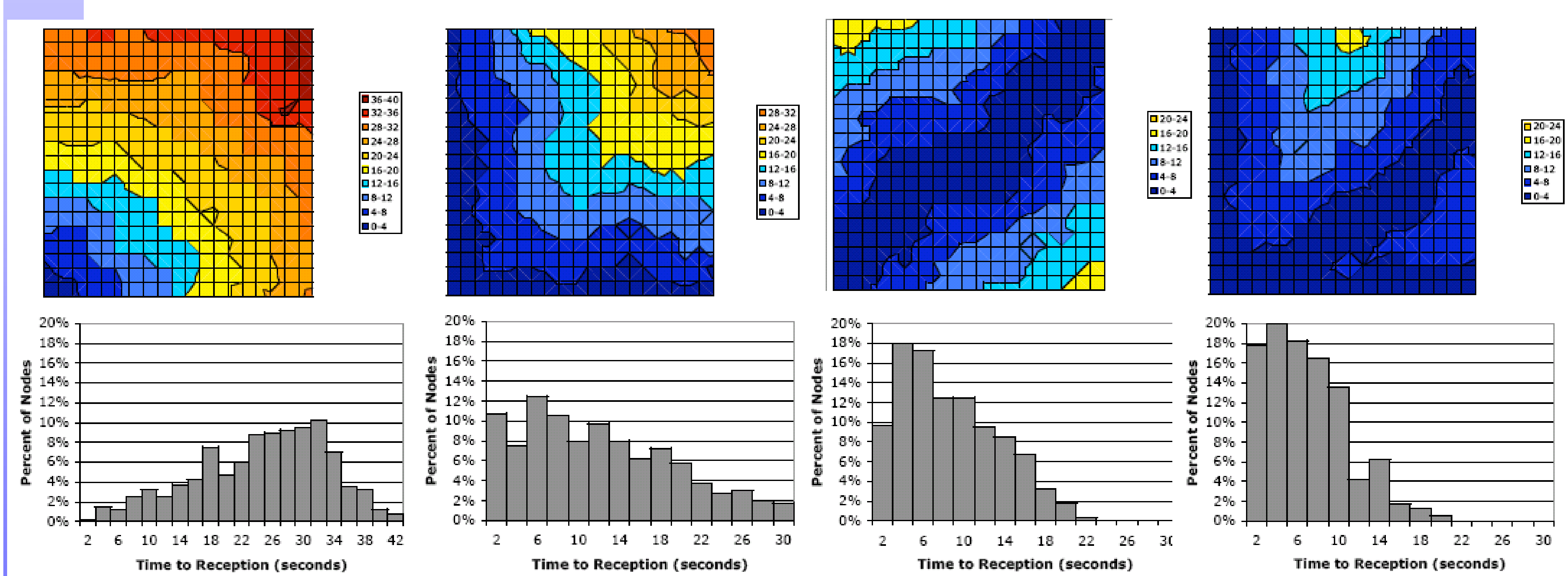
Opposite Corners



All Corners

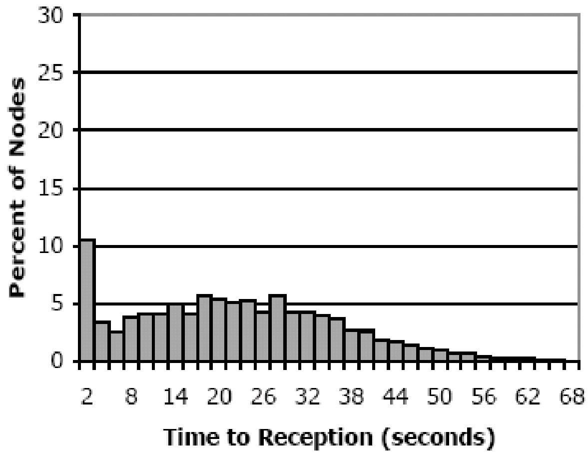


Latency / Transmissions

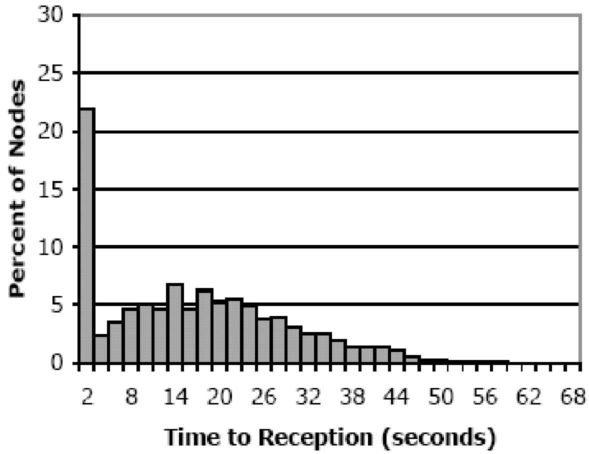


20835 19544 18275 6665

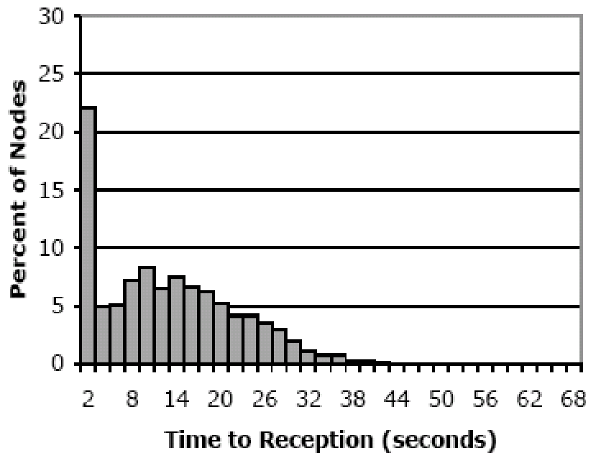
Random Nodes



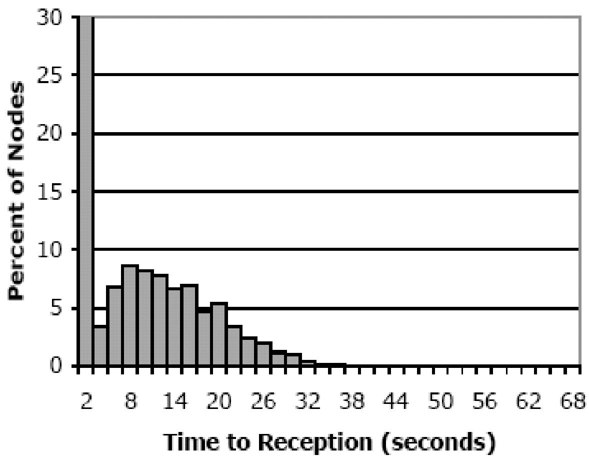
One from corner



Three from corner



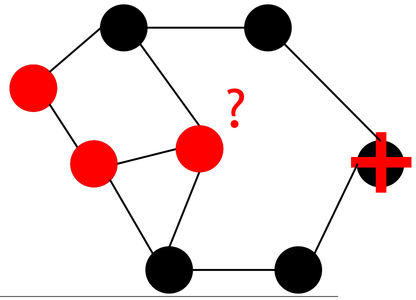
Three from center



Three remote from center

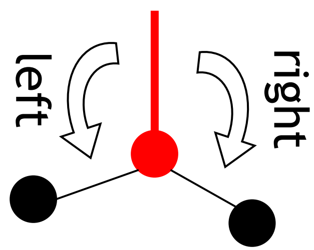
Geo Routing

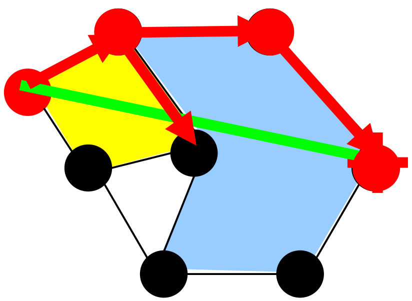
- Send to node at position (x,y)
  - Avoiding keeping state in nodes
  - Few bytes in message headers
- Greedy Routing
  - Send to neighbors closest to (x,y)
  - Problem: holes in the networks



Face Routing

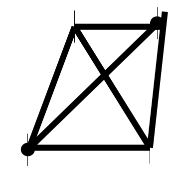
- Walk along polygons („Face“) crossed by line L between start and dest position
  - Select first edge left of L
  - If edge crosses L
    - Select first edge left of edge
  - Traverse edge
  - Stop if destination reached
  - Select first edge left of old edge

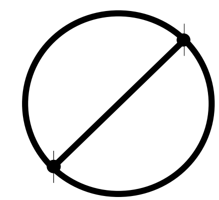




Face Routing

- Requires planar network graph
  - No crossing edges in 2D
  - Example: Gabriel Graph
    - Two nodes are connected only if enclosing circle does not contain other node





- Many possible improvements
  - GPSR: Greedy + Face Routing
- Addressing variants
  - Node close(st) to destination position
  - All nodes in region

References

- Slides contain material by the following authors
  - Prabal Dutta, Alec Woo – UC Berkeley
  - Phil Levis – Stanford
  - Li Huan, Junning Liu – Amherst
  - Ten-Hwang Lai – Ohio
  - Roger Wattenhofer – ETH Zurich