

Beacon Vector Routing: towards scalable routing for sensor networks

NEST Retreat, January 2004

Rodrigo Fonseca

joint work with Sylvia Ratnasamy, Ion
Stoica, David Culler, Scott Shenker

Routing in Sensor Networks

- Large scale sensor networks will be deployed, and require richer inter-node communication
 - In-network storage (DCS, GHT, DIM, DIFS)
 - In-network processing
 - “Fireworks routing”
- Need point-to-point routing to scale
 - Many nodes
 - Many flows
 - Different densities

Scalable routing

- Internet scales via address aggregation
 - Can't do with sensor networks
- On demand flooding
 - Scales poorly with #nodes, #flows
- Tree-based routing (hierarchical)
 - Fragile for more general many-to-many
- Geographic Routing
 - Uses local information, reference to global space
 - Scalable

Geographic Routing

- General algorithm
 - Nodes know theirs and neighbors' geo position
 - Greedy forwarding towards destination position
 - Fallback mode if greedy fails
- Routing gradient
 - Computable for each neighbor, from coordinates
- Local routing state, local control traffic
- What if geographic information is not available?

Beacon Vector Routing

- Solution: fake geography
 - Create a routing gradient from **connectivity** information rather than geography
 - Nodes assigned positions based based on connectivity
 - Greedy forwarding on this space
- Other approaches
 - NoGeo, GEM
 - Landmark Routing, Ascent

Outline (from here on)

- Beacon-Vector Routing
 - Goals
 - Algorithm
 - Evaluation

Goals

(when we started out)

Any-to-any routing that is:

1. Scalable – low control overhead, small routing tables
2. Robust – node failure, wireless vagaries
3. Efficient – low routing stretch

Goals

(after we started on an implementation)

Any-to-any routing that is:

1. **SIMPLE** – minimum **required** state, assumptions
2. Scalable – low control overhead, small routing tables
3. Robust – node failure, wireless vagaries
4. Efficient – low routing stretch

Beacon-Vector: Algorithm

- 3 pieces
 - Deriving positions
 - Forwarding rules
 - Lookup: mapping node IDs \rightarrow positions

Beacon-Vector: deriving positions

1. K beacon nodes (B_0, B_1, \dots, B_k) flood the network; a node P 's position, $P(K)$, is its distance in hops to each beacon

$$P(K) = \{B_0:P_0, B_1:P_1, \dots, B_k:P_k\} \quad (\text{w.l.o.g assume } P_0 \leq P_1 \dots \leq P_k)$$

2. Define the distance between two nodes P and Q as

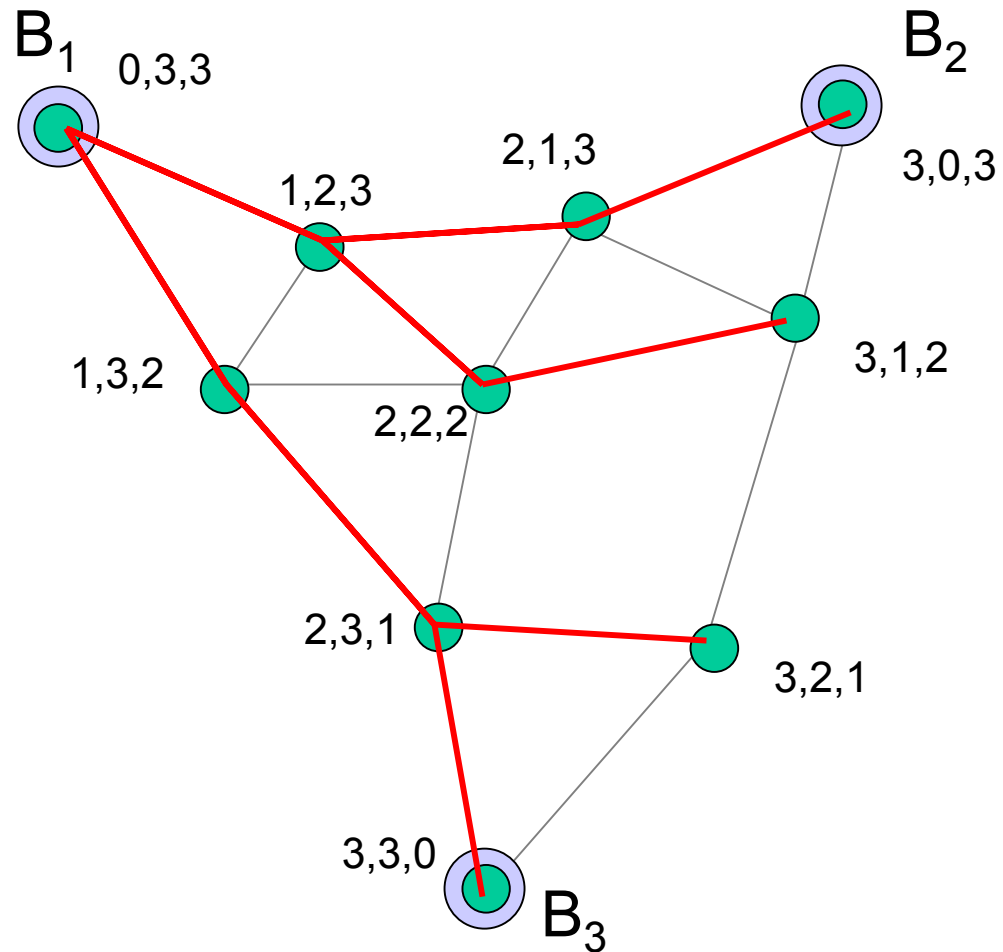
$$\text{dist}(P, Q) = \sum \omega_i |P_i - Q_i| \quad \text{where } P_i = \text{hops from beacon } i \text{ to } P$$

3. Nodes know their own and neighbors' positions; to reach destination Q , forward to reduce $\text{dist}(*, Q)$

Beacon-Vector Routing: forwarding

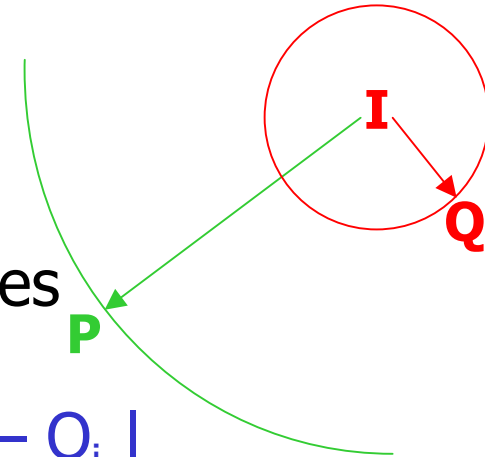
```
bvr_forward(node D, pkt P) {  
  
    // first try greedy forwarding  
    node N = neighbor with MIN dist(i,N,D)  
    if( dist(N,D) < MIN_DIST from pkt P)  
        pkt P.MIN_DIST = dist(N,D)  
    return N  
  
    // greedy failed, use fallback  
    fallback_bcn = beacon closest to D  
    if(fallback_bcn != me) return parent(fallback_bcn)  
  
    // fallback failed  
    flood with scope  $D_{\text{fallback\_bcn}}$   
}
```

Simple example



Beacon-Vector Routing: details

Observation: Beacons that “pull” are good guides



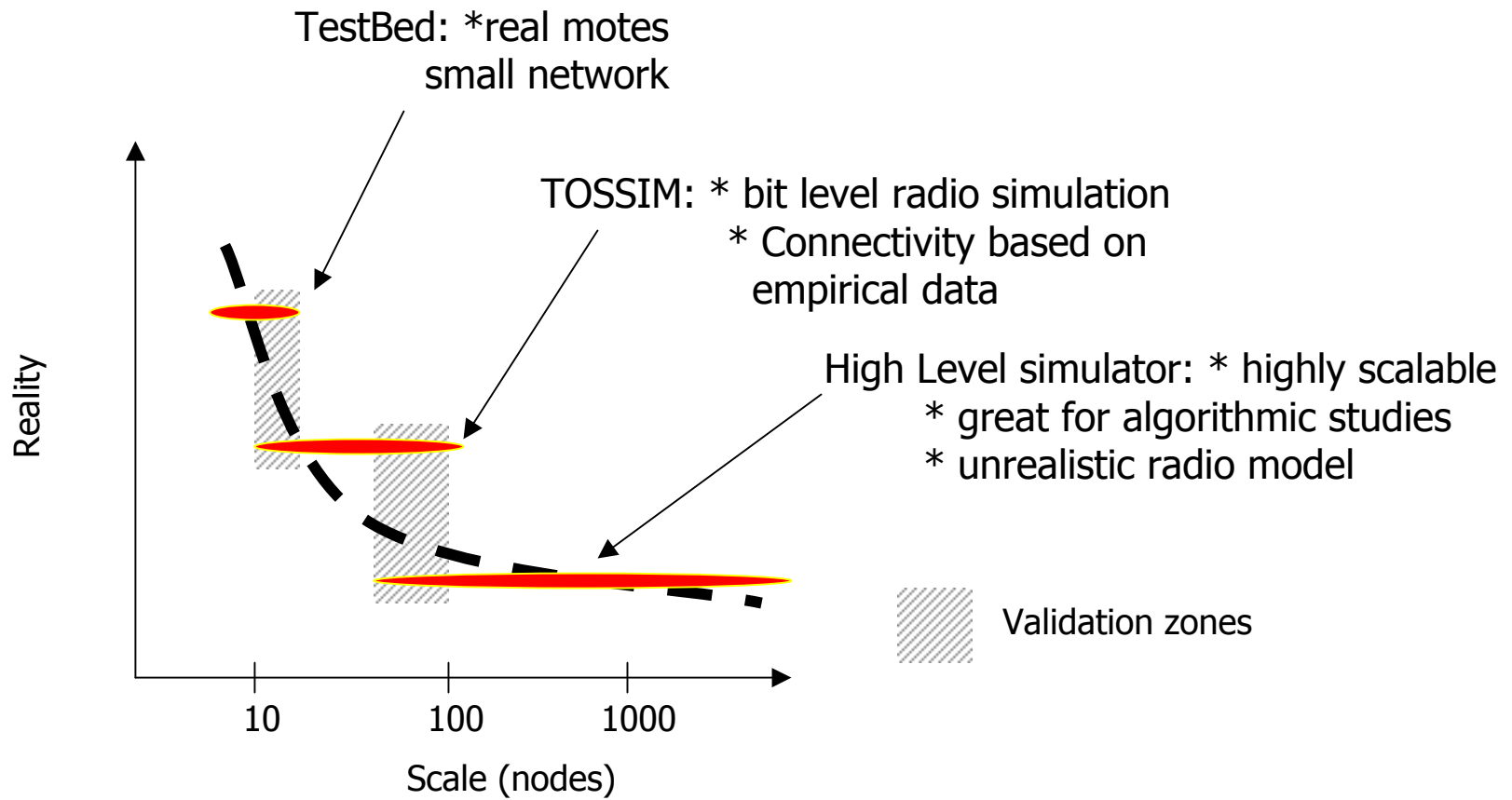
- Beacon weights in distance metric $\sum \omega_i | P_i - Q_i |$
 - $\omega_i = 10$ if $P_i > Q_i$
= 1 otherwise
- Need compact node positions to reduce per-packet overhead
 - For routing, a node's position is defined by its k ($\leq B$) closest beacons

Open questions: optimal distance metric, weights, beacon placement and number, ...

Routing to Node Identifiers

- Beacon-Vector routes to node positions; need a **lookup** mechanism to map node identifiers to positions [GLS, GHT]
- Our solution: Use beacons to store mapping
 - Given a node identifier, use consistent hashing to determine which beacon stores its position
 - Simple, but imposes additional load on beacons

Evaluation Methodology



Evaluation: Metrics

1. Performance metrics

- success rate without flooding
- path stretch

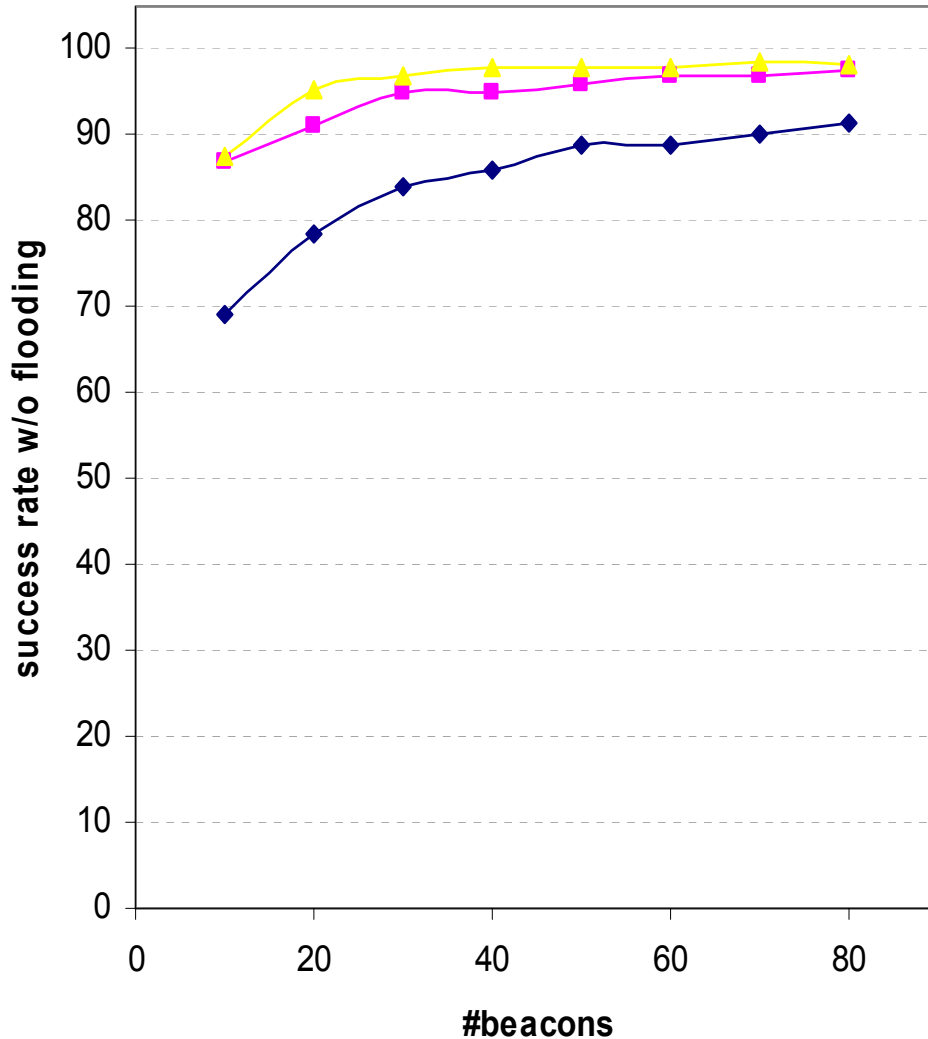
2. Overhead

- total #beacons needed (flooding overhead)
- #beacons used for routing (per-packet overhead)
- #neighbors (per-node routing table)

3. Scalability

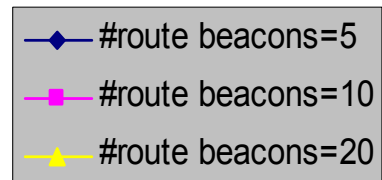
- network size
- network density

Beaconing overhead

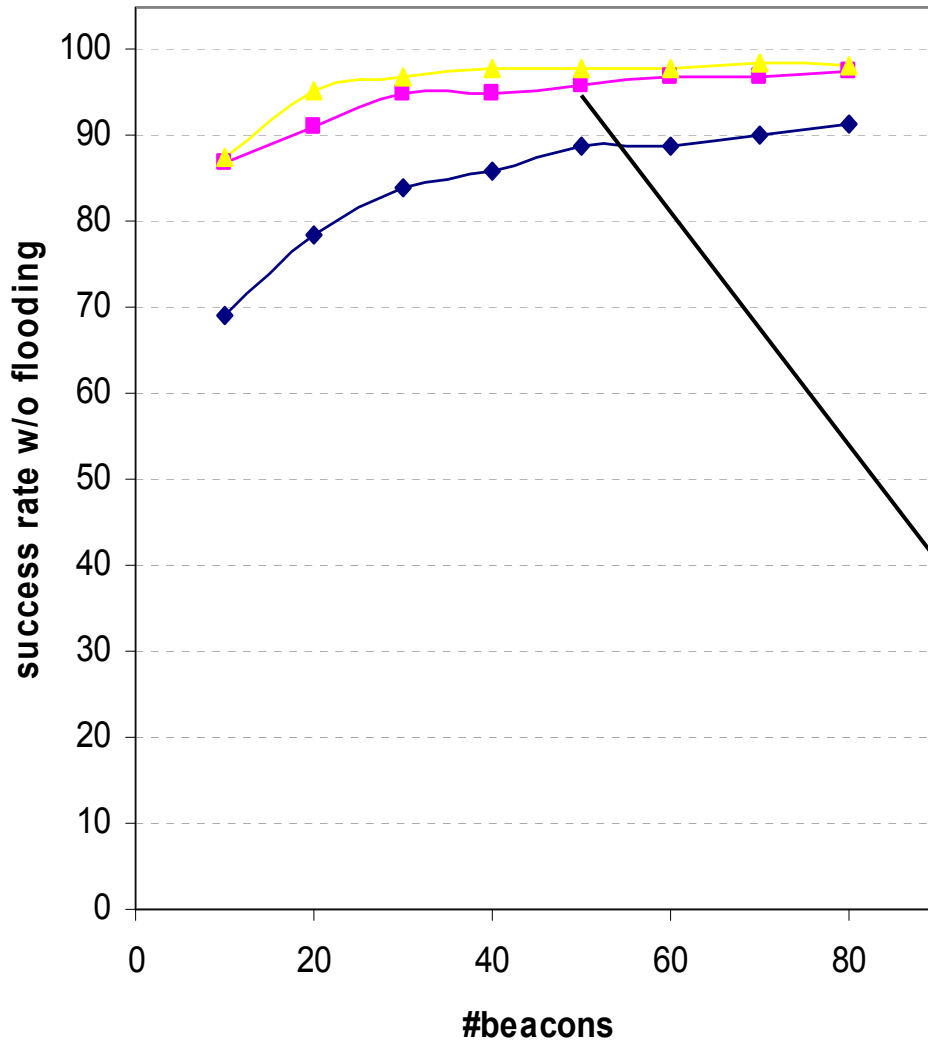


3200 nodes u.a.r in grid

Success rate with
true positions = 96.3%



Beaconing overhead



3200 nodes u.a.r. in grid

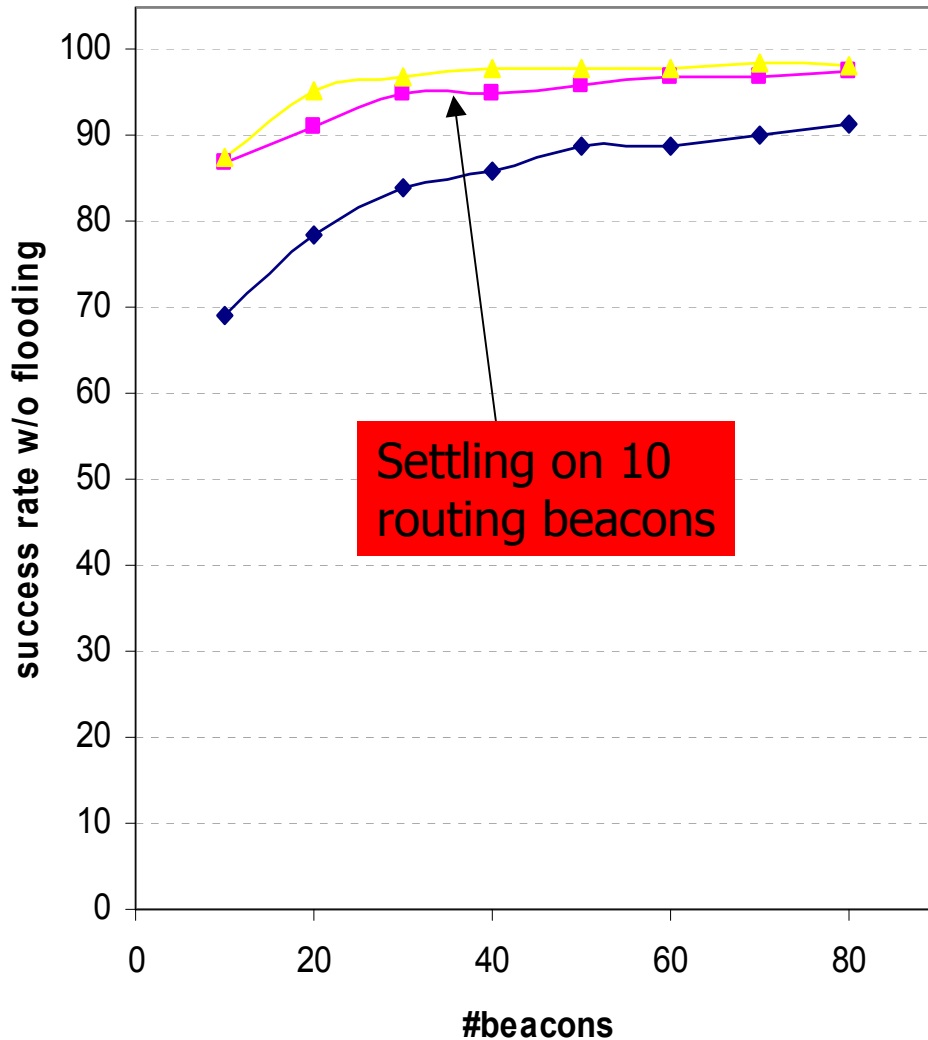
Success rate with
true positions = 96.3%

◆ #route beacons=5
■ #route beacons=10
▲ #route beacons=20

success rate = 96% (w/o fallback=91%)
avg(ngbrs) = 15.0
avg(hops) = 17.5, path stretch = 1.05
avg(flood hops)=3.7

Beaconing overhead

Can achieve performance comparable to that using true positions



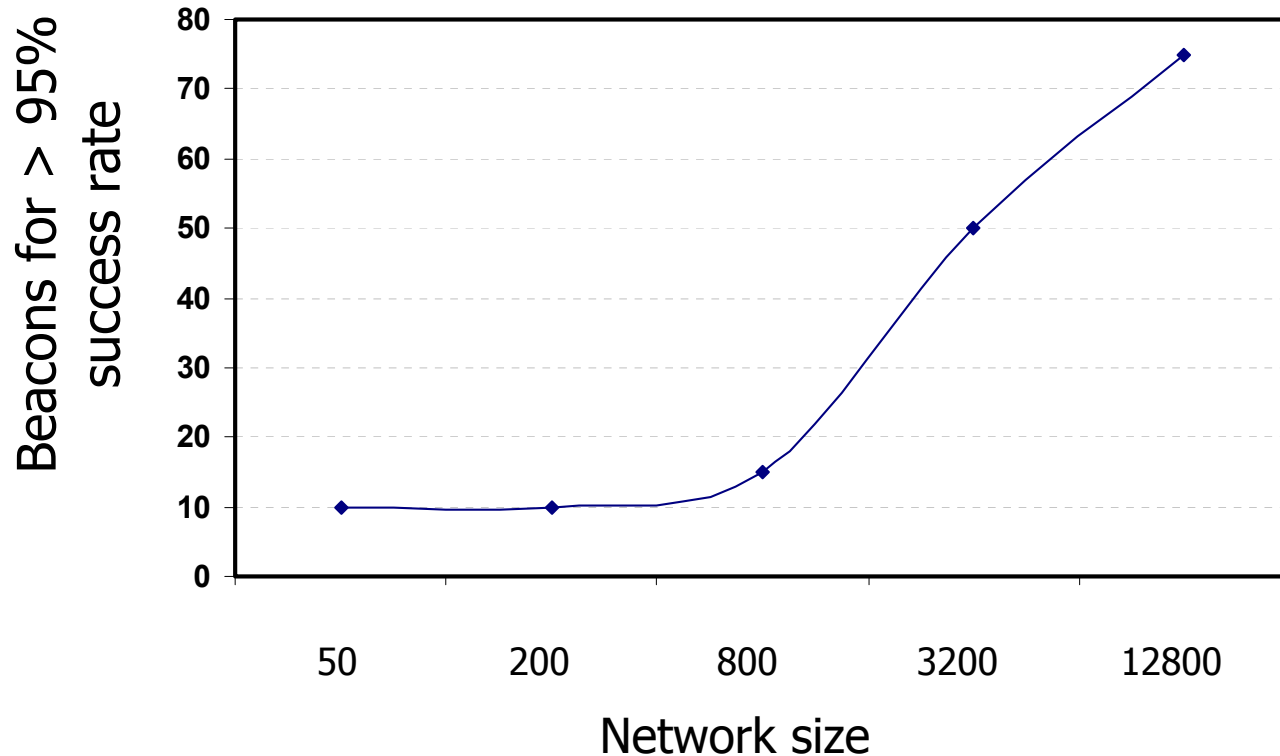
3200 nodes u.a.r. in grid

Success rate with true positions = 96.3%

- ◆ #route beacons=5
- #route beacons=10
- ▲ #route beacons=20

Scaling Network Size

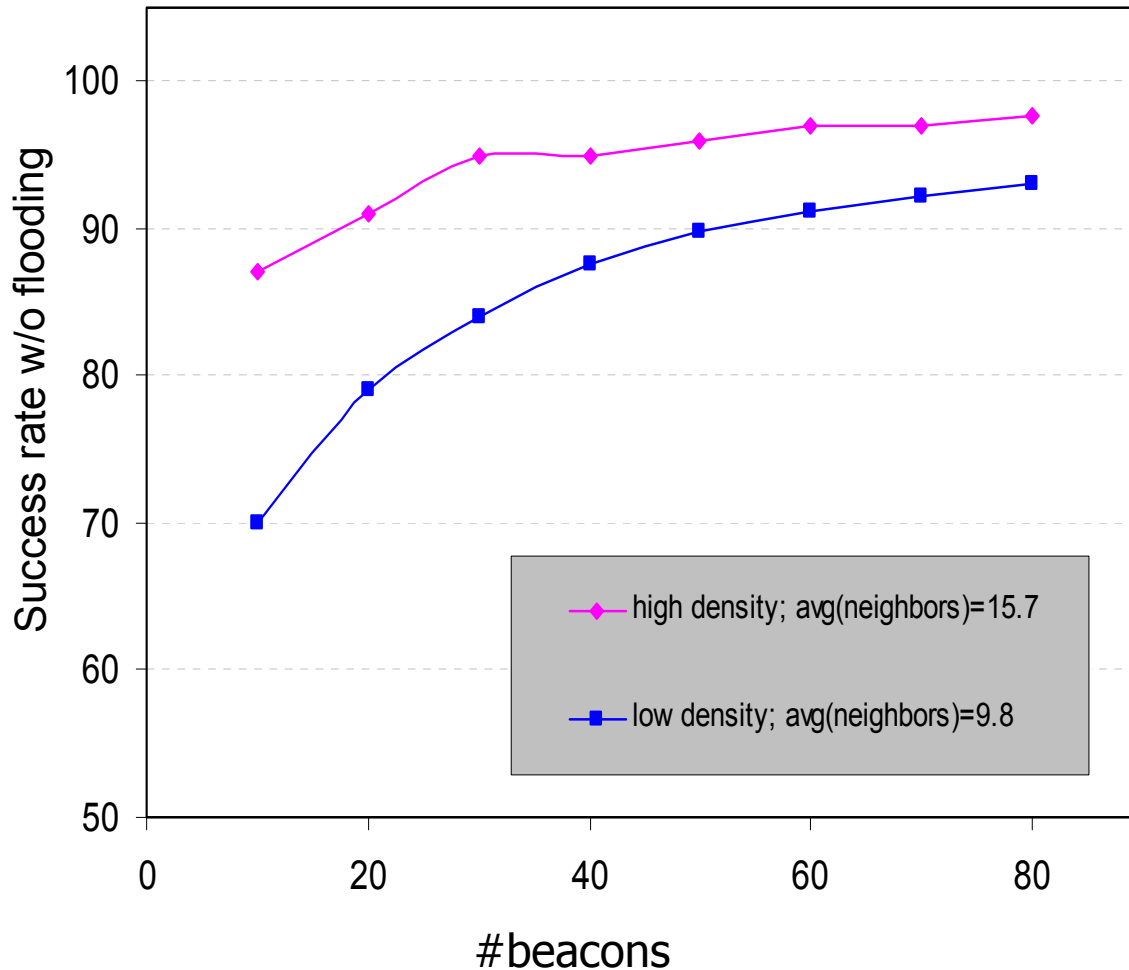
3200 nodes, 10 routing beacons



Beaconing overhead grows slowly with network size

Scaling Network Density

3200 nodes, 10 routing beacons



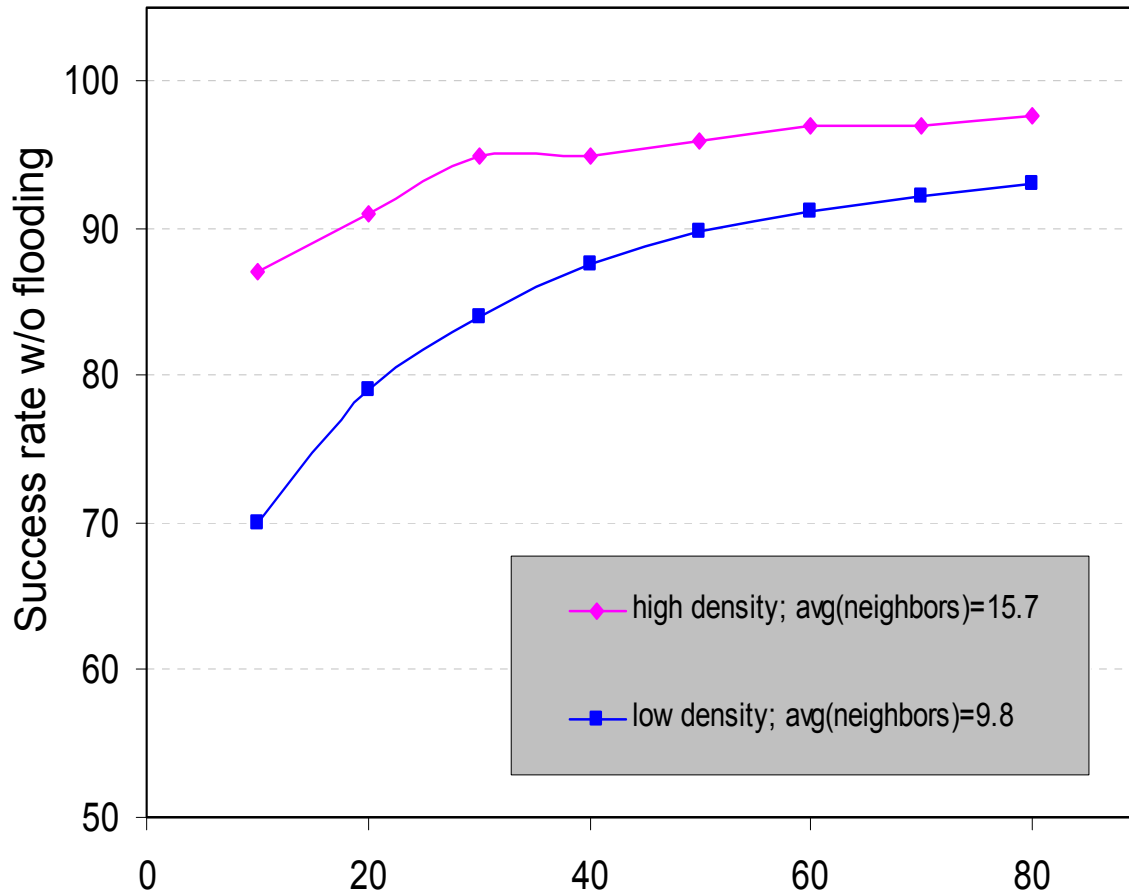
1-hop neighbors

True postns. (high dens) = 96.3%

True postns. (lo density) = 61.0%

Scaling Network Density

3200 nodes, 10 routing beacons



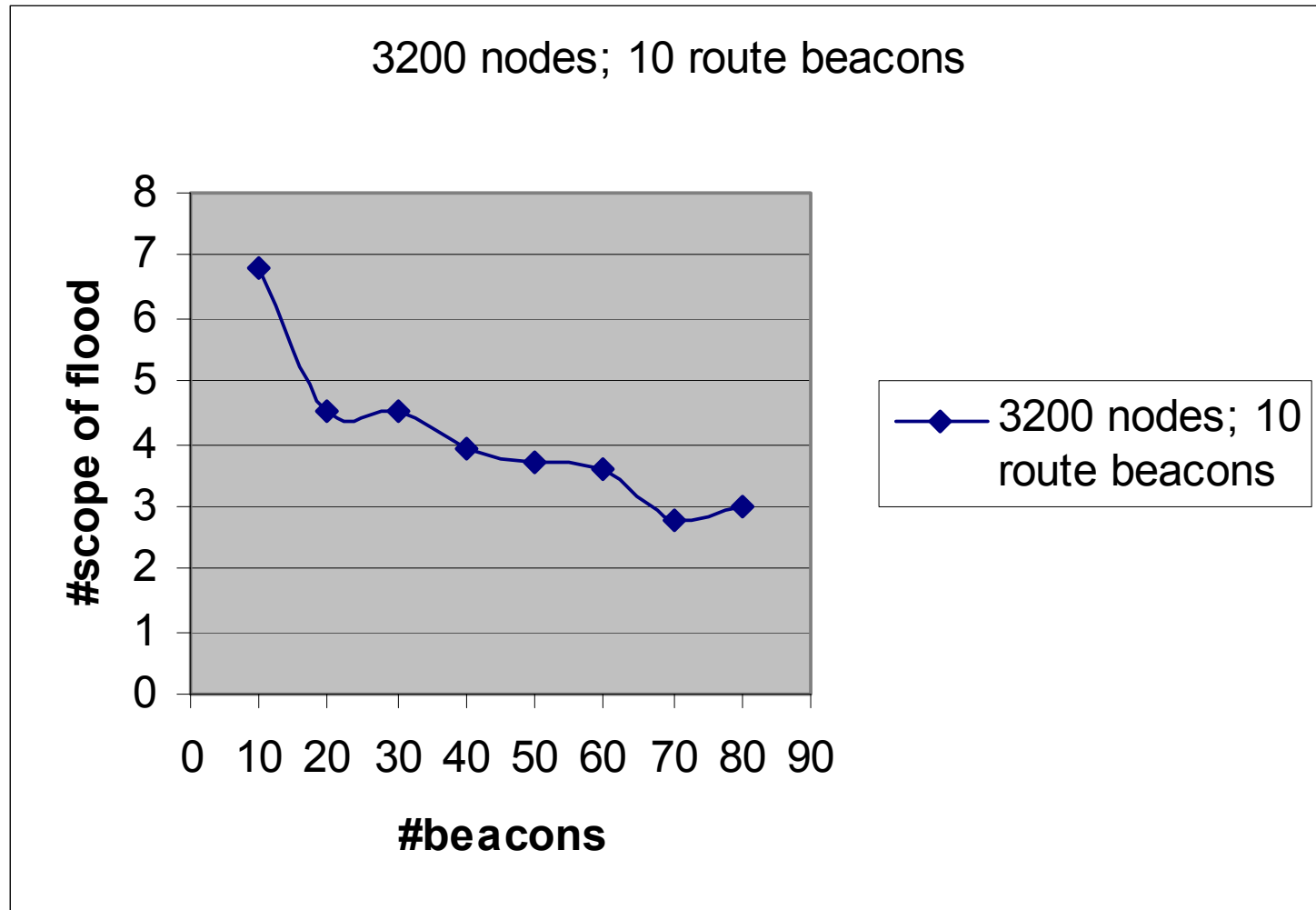
1-hop neighbors

True postns. (high dens) = 96.3%

True postns. (lo density) = 61.0%

Inherent tradeoff between routing state and the efficacy of greedy routing

Scope of flood for failed routes



Implementation: current status

Prototype in TinyOS for the Mica2 motes

- exports a “route-to-coordinates” interface

Routing state maintenance

- Assumes preconfigured and perennial beacons
- Bidirectional loss-driven link estimation
- Beacons flood periodically; tree constructed using link estimates
- Nodes periodically advertise their coordinates (1-hop)

Currently testing on the Intel Lab testbed and under TOSSIM

- small scale (~15 motes)
- complete, ordered logging via Ethernet backend

Implementation

- First experimental results, understanding imp. Behavior
- Mica2 motes, CC1000 radio @ 433Mhz, Intel Lab Testbed
- 4 beacons in the corners
- All route to all, no link retransmission

	Testbed (16 nodes)			TOSSIM (20 nodes)		
Route	Total	%	%C	Total	%	%C
Started	408			740		
Dropped	29	7.1	-	235	31.8	-
Successful	343	84.1	90.5	436	58.9	86.3
Same Coords	0	0	0	0	0	0
At Beacon	36	8.8	9.5	69	9.3	13.7

Conclusion

- **Beacon-Vector performance**
 - overhead scales well with network size and density
 - outperforms true geography at lower densities
- **Other results**
 - On demand 2-hop neighbors is a big win, specially on lower densities
 - obstacles: up to 40% improvement relative to true positions
- **Many open questions remain**
 - Further prototype evaluation (Intel testbed & TOSSIM)
 - Robustness under failure (TOSSIM)
 - Evaluate DHT on top of BVR
 - Compare with other approaches

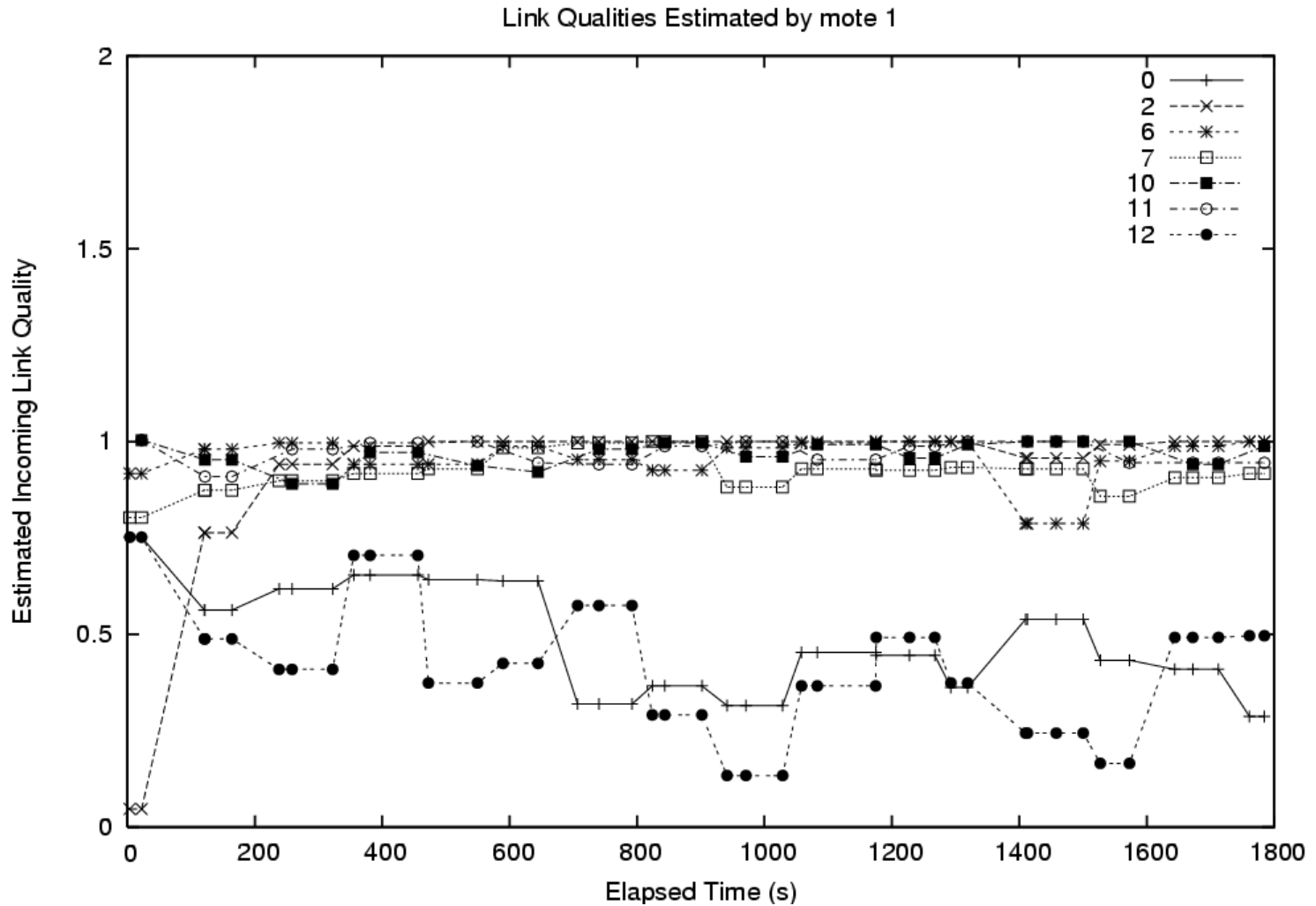
Thank you

Backup Slides

Beacon-Vector *vs.* NoGeo/GEM: summary

- NoGeo/GEM: embed nodes in a virtual coordinate space
 - Forming this space is non-trivial
 - However, given this space, supporting a DHT is simple
- Beacon-Vector: landmarks, but no global coordinate space
 - Routing is easy
 - Building a DHT is more complex
- Open question: Is there a middle ground? Define a global frame of reference based on beacon positions?

Link Stability



Tree Stability

Distances from beacon 13

