Programming a Sensor Network as an Amorphous Medium

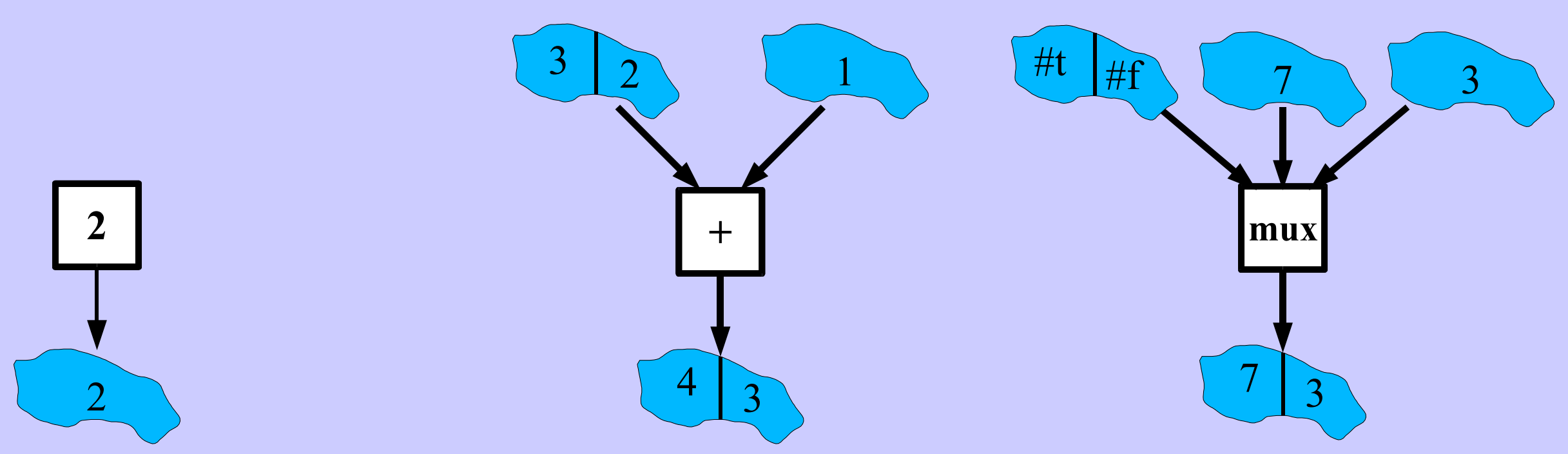
Jonathan Bachrach & Jacob Beal, MIT CSAIL

Programming in Proto

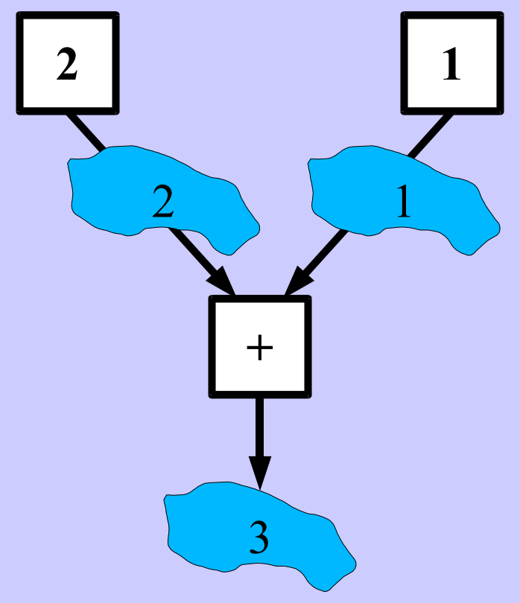
Proto is a stream processing language based on the amorphous medium abstraction. Our implementation supports over-the-air programming of Mica2 Motes.

Basics

Primitives

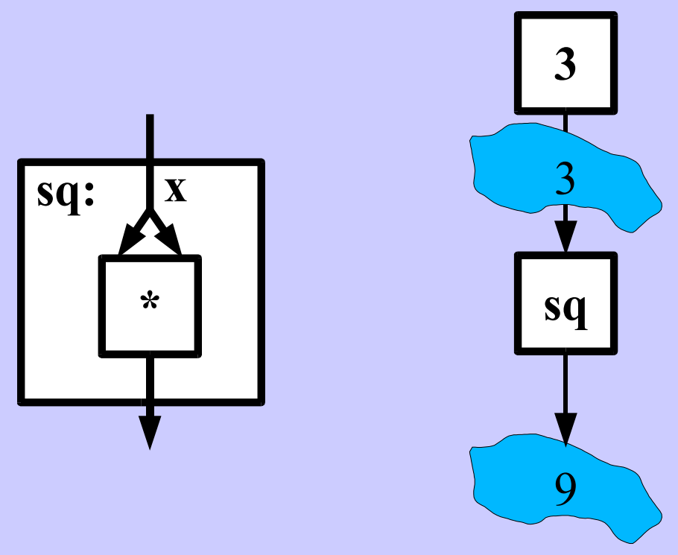


Composition



```
(+ 2 1)
```

Abstraction

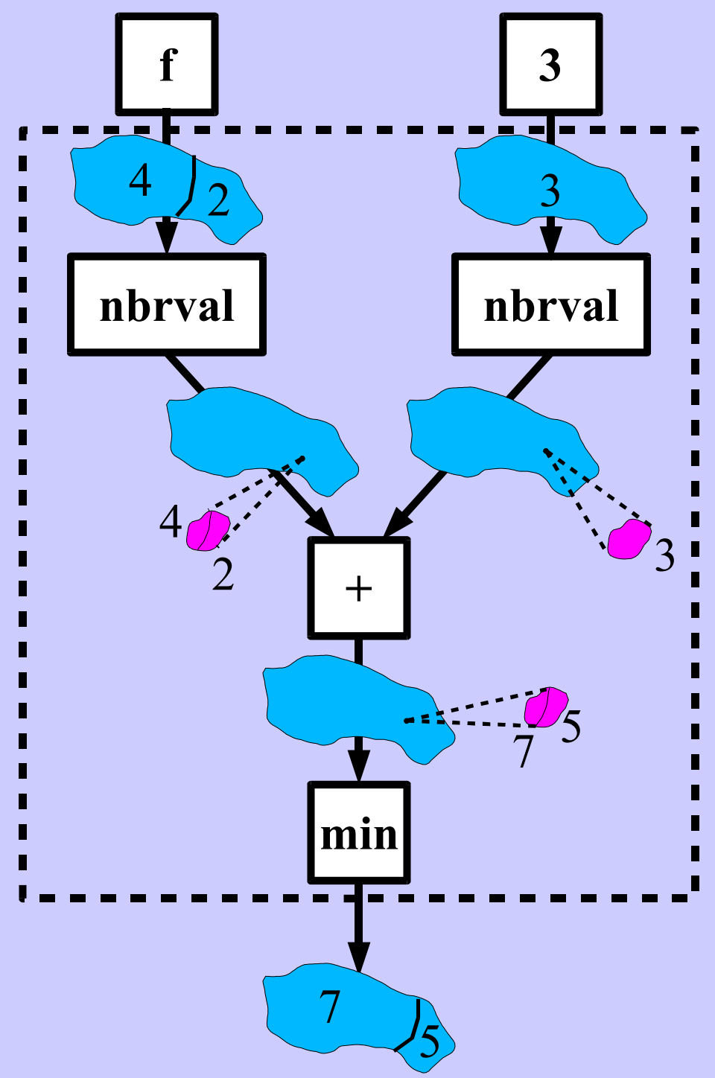


```
(def sq (x)
  (* x x))
```

```
(sq 3)
```

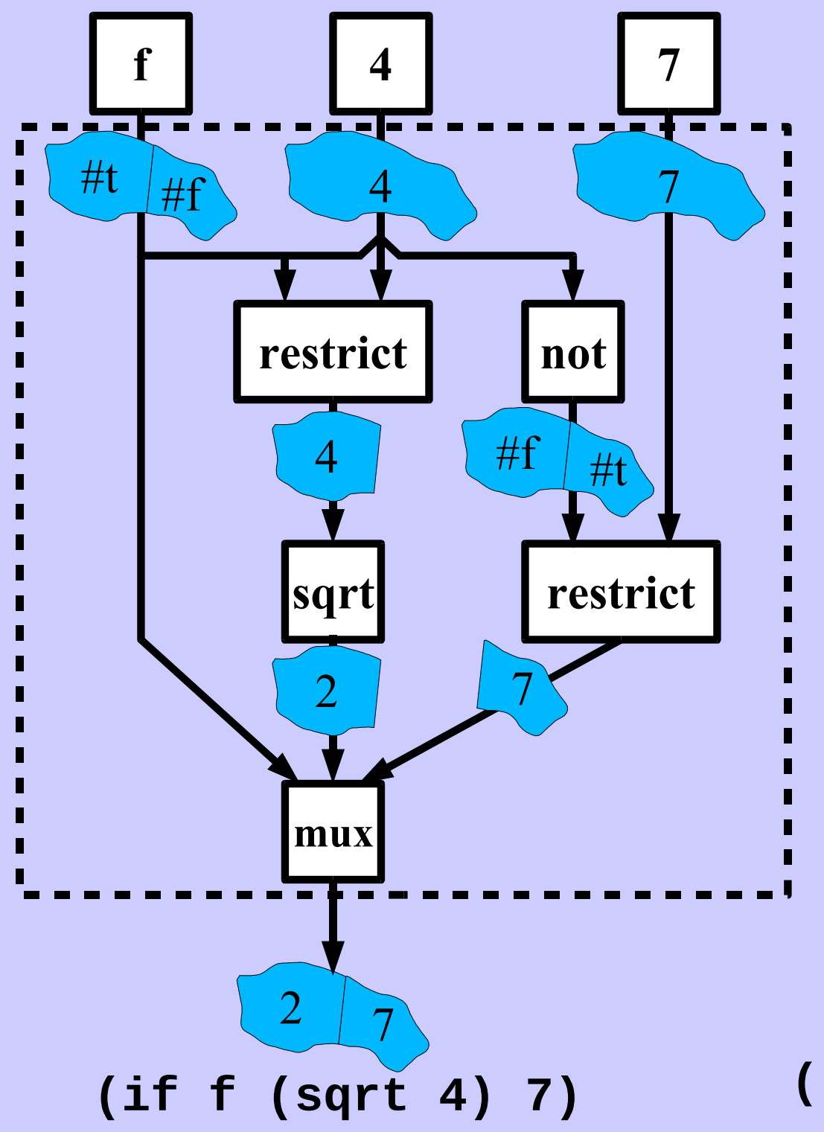
Special Operations

Communication



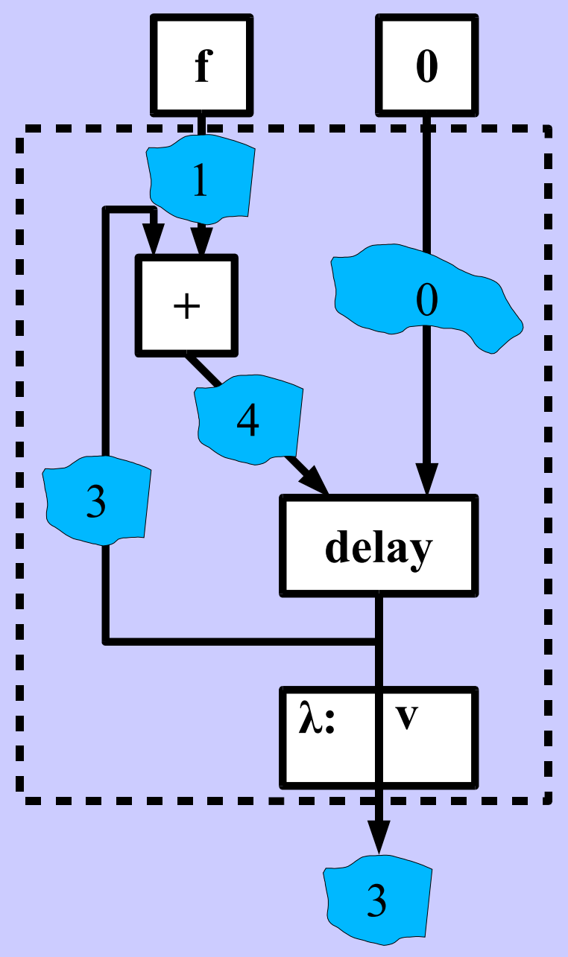
```
(reduce-nbrs (+ f 3) max)
```

Space Restriction



```
(if f (sqrt 4) 7)
```

State

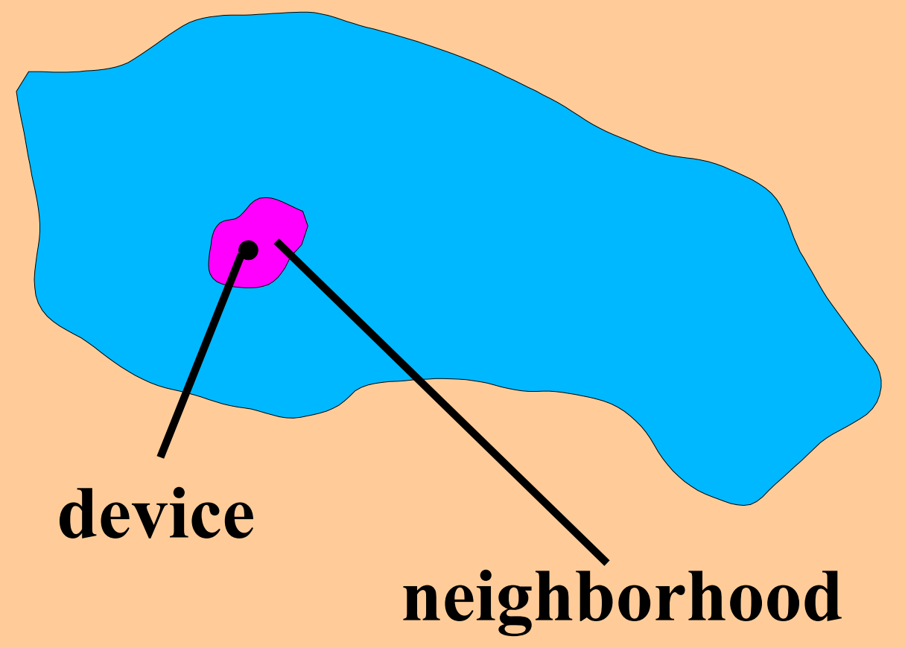


```
(letfed ((v 0 (+ f v)))
  v)
```

For more information on Proto, see *Infrastructure for Engineered Emergence on Sensor/Actuator Networks*, Jacob Beal and Jonathan Bachrach, IEEE Intelligent Systems, (Vol. 21, No. 2) pp. 10-19, March/April 2006.

What is an Amorphous Medium?

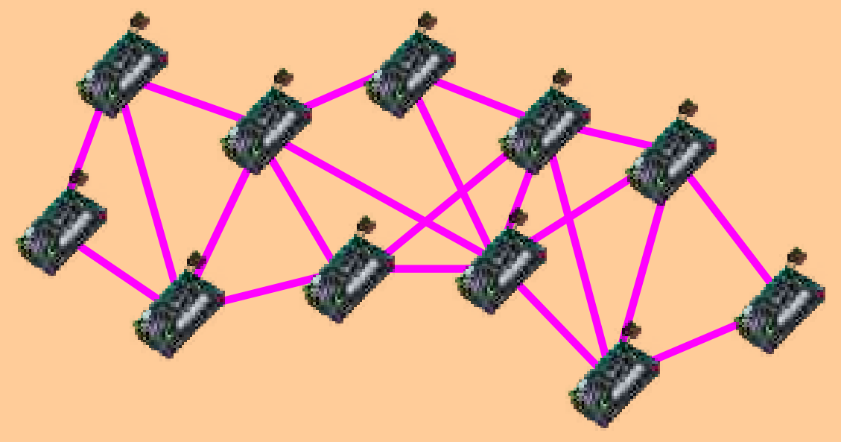
Many sensor-network applications care less about the network than the properties of the space it occupies. An *amorphous medium* program controls space explictly, and is approximated by implicit network activity.



The medium is a compact manifold with a device at every point. Devices can read time-lagged state from neaby neighbors.

Amorphous Medium

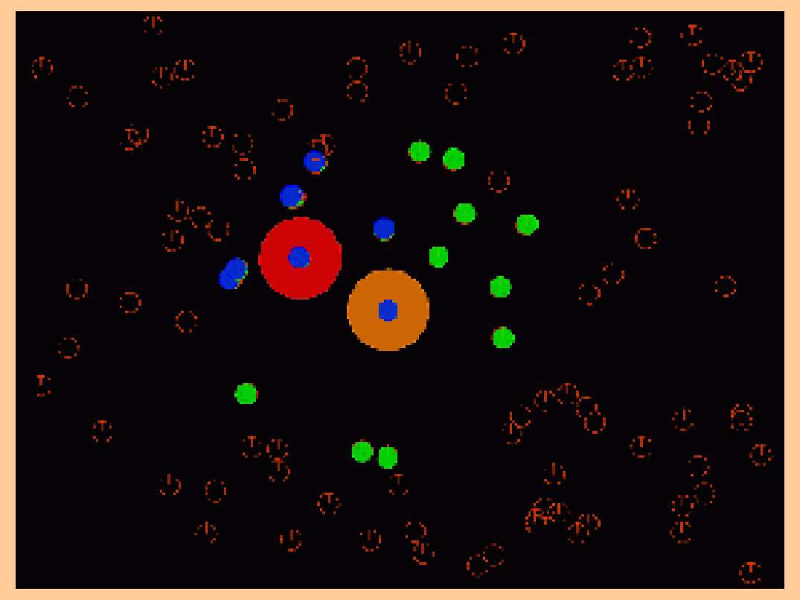
Discrete Network



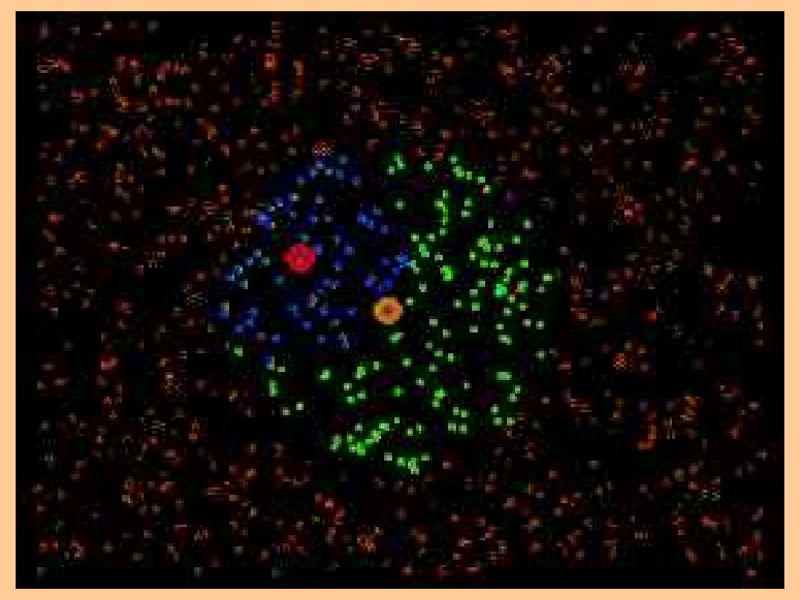
The network is a sample of the amorphous medium and simulates it approximately.

Programs scale gracefully across a wide range

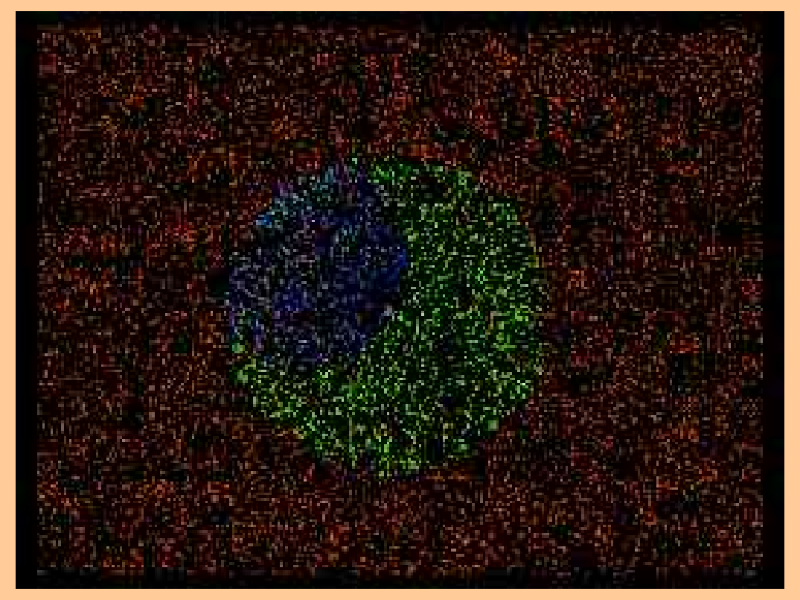
100 nodes



1,000 nodes



10,000 nodes

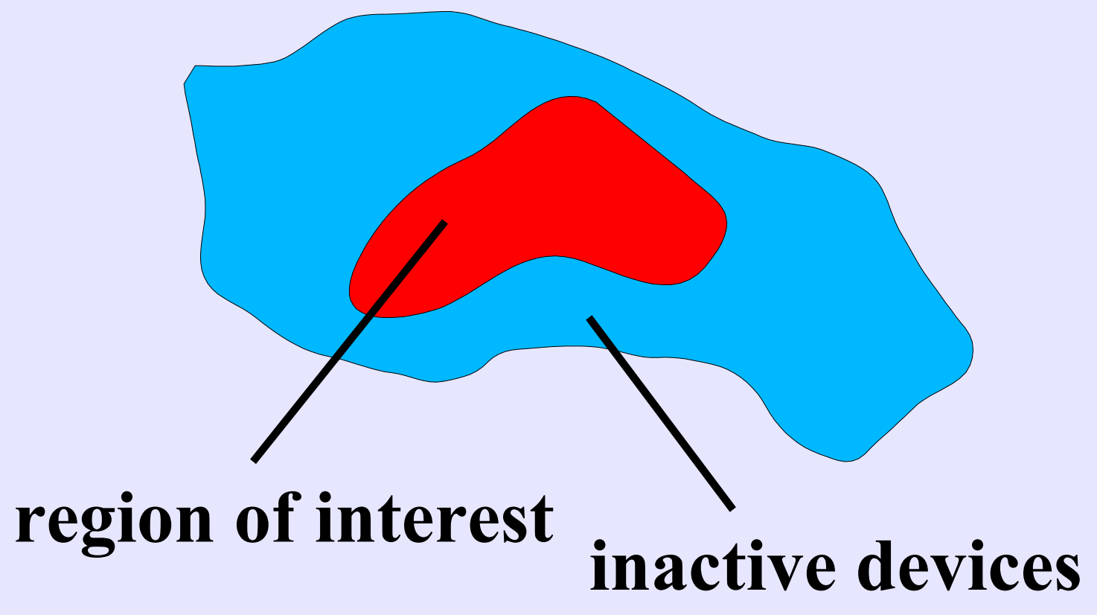


```
(and (green (dilate (sense 1) 30)) (blue (dilate (sense 2) 20)))
```

Energy Management

Many existing energy management techniques can be confined to one side of the abstraction barrier.

Space-Centric

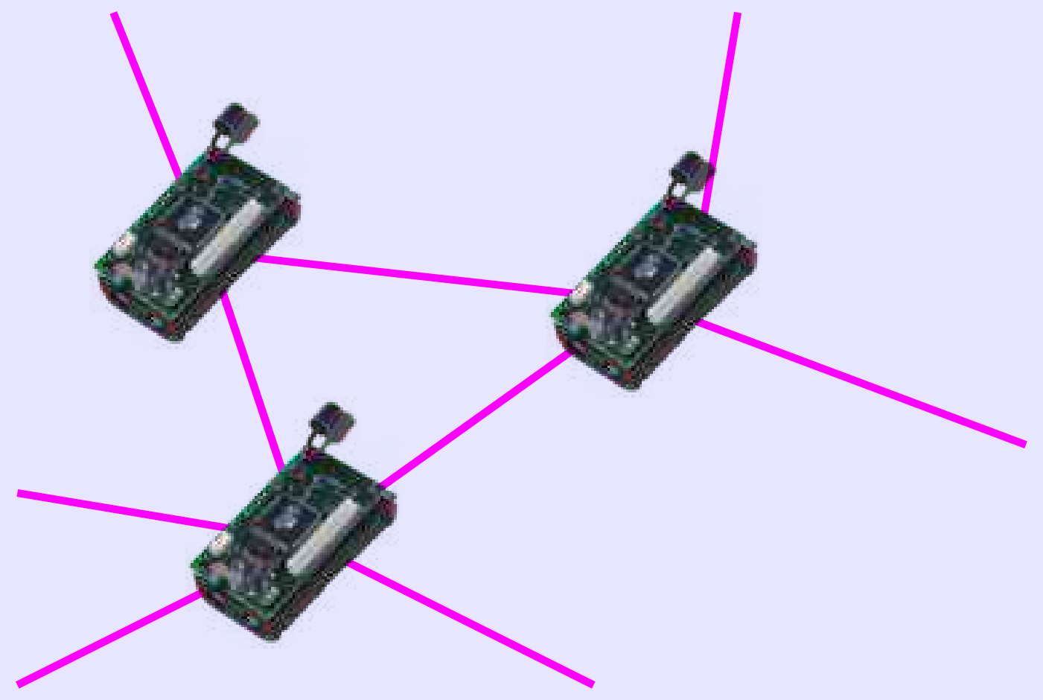


Examples: Energy Aware Routing, Directed Diffusion

Amorphous Medium

Discrete Network

Local Communication



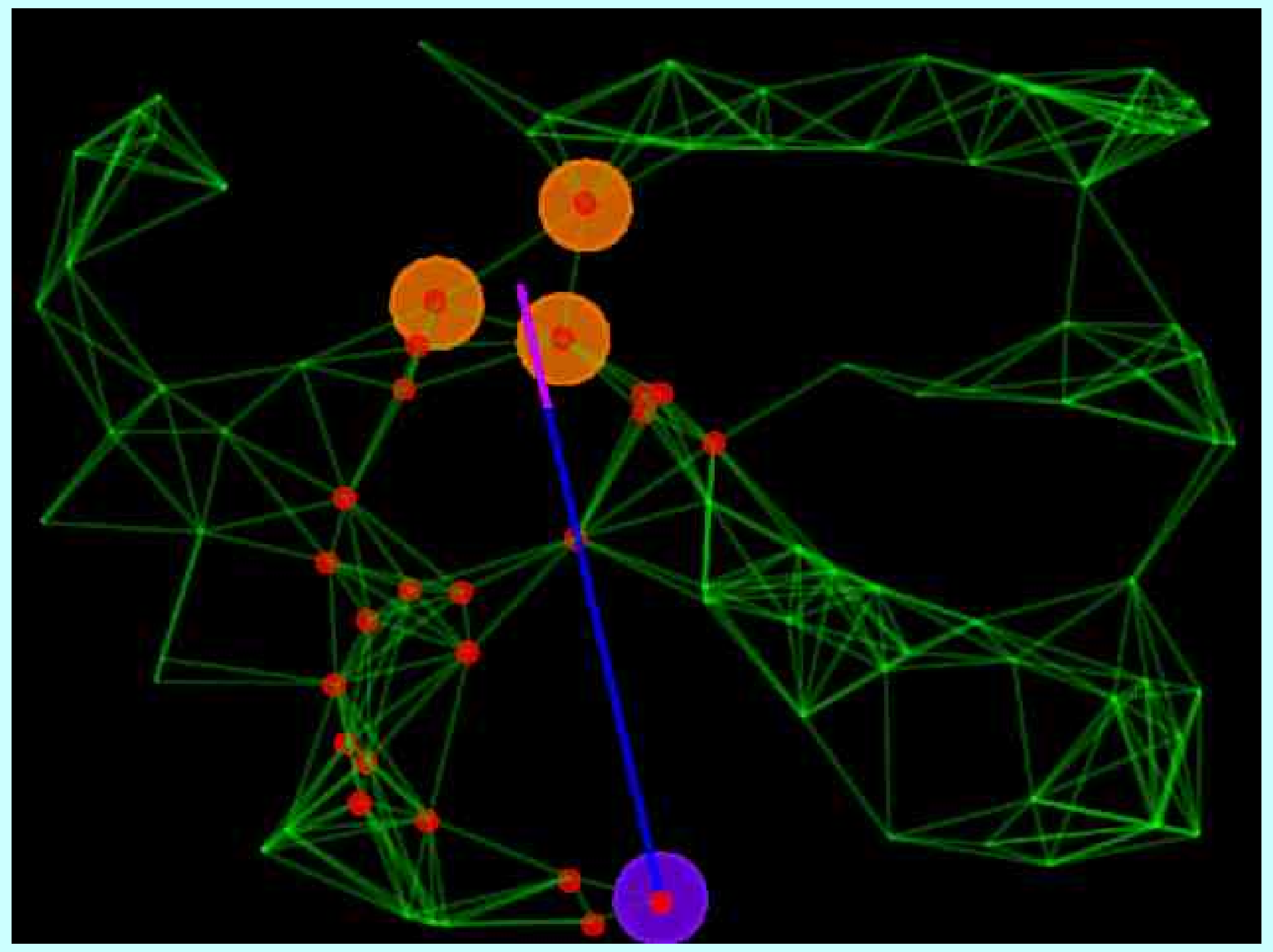
- reduce collisions
- only transmit changes
- directional transmission

etc.

Examples: S-MAC, TDMA

Example Applications

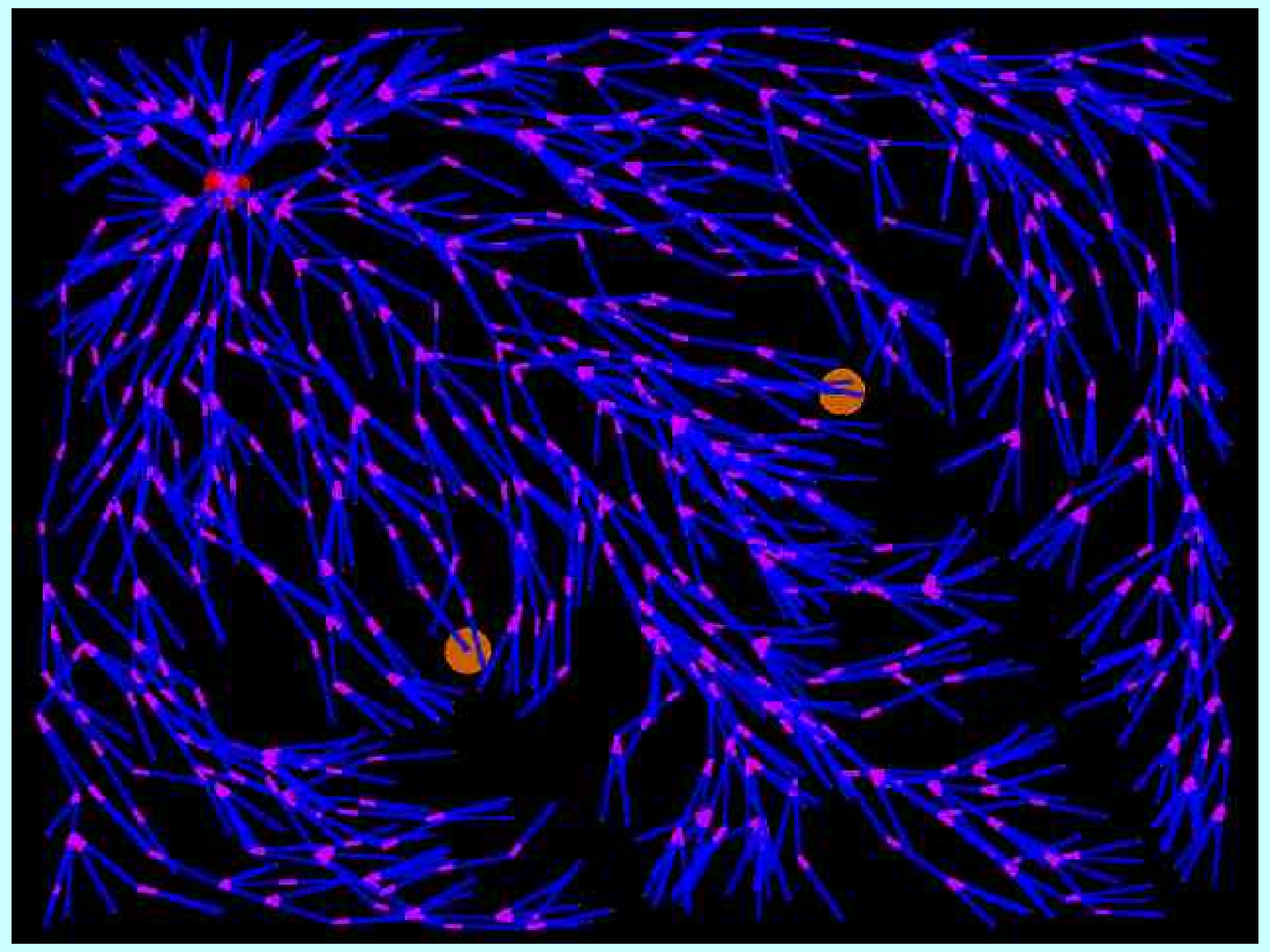
Target Tracking



Complete Code:

```
(def local-average-tup (x)
  (vmul (/ 1 (fold-hood + 0 (* (infinitesimal) 1)))
        (fold-hood vadd (tup 0 0) (vmul (infinitesimal) x))))
(def gradient (src)
  (letfed ((n (inf) (+ 1 (if src 0 (fold-hood (fun (r x) (min r (+ x (nbr-range)))) (inf) n)))))
    (- n 1)))
(def grad-value (src f)
  (let ((d (gradient src)))
    (letfed ((v f (mux src f (2nd (fold-hood (fun (r x) (if (< (1st x) (1st r)) x r))
                                             (tup (inf) f) (tup d v))))))
      v)))
(def distance (p1 p2)
  (let ((gv (gradient p2))) (grad-value p1 gv)))
(def dilate (src n) (<= (gradient src) n))
(def channel (src dst width)
  (let* ((d (+ (distance src dst) 1))
         (trail (<= (+ (gradient src) (gradient dst)) d)))
    (dilate trail width)))
(def track (target dst coord)
  (let ((point
         (if (channel target dst 10)
             (all (red 1) (grad-value target (mux target (local-average-tup coord) (tup 0 0))))
             (tup 0 0))))
    (mux dst (vsub point coord) (tup 0 0))))
(track (sense 1) (sense 2) (coord))
```

Threat Avoidance



Complete Code:

```
(def sqr (x) (* x x))
(def dist (p1 p2)
  (sqrt (+ (sqr (- (1st p1) (1st p2))) (sqr (- (2nd p1) (2nd p2))))))
(def li (p1 v1 p2 v2)
  (pow (/ (- 2 (+ v1 v2)) 2) (* 0.01 (+ 1 (dist p1 p2)))))
(def max-survival (dst v p)
  (letfed ((ps 0 (fold-hood (fun (r n) (max r (* (li (1st n) (2nd n) p v) (3rd n))))
                            (if dst 1 0) (tup p v ps))))
    ps))
(def exp-gradient (src d)
  (letfed ((n 0 (max (* d (fold-hood max 0 n)) src))) n))
(def greedy-ascent (v c)
  (vsub (2nd (fold-hood (fun (r p) (if (< (1st r) (1st p)) p r)) (tup v c) (tup v c))) c))
(def avoid-threats (src dst)
  (greedy-ascent (max-survival dst (exp-gradient src 0.8) (coord)) (coord)))
(avoid-threats (sense 1) (sense 2))
```