In a nutshell: NetLogo is an agent-based simulation system. Decentralized algorithm specifications can be relatively directly translated into NetLogo. Simulations can help designers to understand the global, emergent behavior in the protocol, as well as to empirically test the limitations of scalability and robustness.

Understand: NetLogo is a popular free and open source agent-based simulation system. It enables rapid simulation of both decentralized algorithms and geographic environments. NetLogo documentation is online, http://ccl.northwestern.edu/netlogo. A small library of new NetLogo keywords and functions has been developed to help with implementing decentralized algorithms. This library includes *become* (change state), *broadcast* (message to all neighbors), *send* (message to specific neighbor), and routines for *network initialization*, *message receipt*, and *scalability testing*.

```
;; Run the algorithm.
to go
  ask motes [ step ]
  tick
end

;; Step through the current state.
to step
  if state = "INIT" [ step_INIT stop ]
  if state = "IDLE" [ step_IDLE stop ]
  if state = "DONE" [ step_DONE stop ]
end

;; Broadcast a message to neighbors.
to step_INIT
  broadcast ["MSGE"]
  become "DONE"
end

;; When a mote receives a message it may rebroadcast the message
;; to its neighbors based on the probability of the g value.
to step_IDLE
  if has-message "MSGE" [ ;; When the mote receives a message
    let msg received "MSGE" ;; Process the MSGE message
    let r random-float 1 ;; Generate random float [0.0,1.0]
    if r < g [
      broadcast ["MSGE"] ;; Rebroadcast MSGE with probability g
    ]
    become "DONE"
  ]
end

;; The DONE state does not respond to messages.
to step_DONE
  set color 0 ;; Nodes in the DONE state are black
end
```

Netlogo also includes the BehaviorSpace tool for automating the running of experiments on protocols. Typical scalability tests investigate the number of messages generated by the algorithm as the network size increases (e.g., 125, 250, 500, 1000, 2000, 4000, … nodes). The *overall scalability* is important, but so is the individual scalability for nodes (termed *load balance*). In addition to scalability, experiments typically also investigate correctness, especially for approximate algorithms; robustness to sensor errors and faults. Faults are usually classified into two types: *stop faults* (node or link stops functioning) and *Byzantine faults* (arbitrary failures, like message corruption or state changes).

Discuss:

- Compare the Gossiping simulation model (above) with the specification in sheet #2. What are the differences?
- Run some simulations of the Greedy Georouting protocol. Can you generate examples of the limitations identified in sheet #2?
- Experiment with the Greedy Georouting protocol. Can you determine its average case scalability?
- Investigate the operation and scalability of other protocols, such as establishing a shortest-path rooted tree (4.8), hop-count sweep (7.1), topological relations between two regions (4.15).
- Investigate the robustness of these protocols, designing and simulating some stop and Byzantine faults.

See: Chapter 7 in Duckham (2012) *Decentralized Spatial Computing*, Springer, Berlin.

GIScience 2012
Seventh International Conference on Geographic Information Science