Simulating Routing over Virtual Nodes

Mike Spindel

Role of Simulation

- Repeatability
- Detailed control
- Detailed examination
- Large scale experiments
- Rapid development and testing

A Typical Simulator

- For example, ns-2
- Discrete event simulator
- Based around a scheduler and event queue
- Provides mobile node structure
- Network stack includes netif, mac, ifq, ll, antenna, etc.

Irritations

- Lots of custom interfaces
- Generally asynchronous
- Obtuse configuration
- Manual plumbing
- Inconvenient languages

```
$ns_ node-config -adhocRouting $opt(adhocRouting)
                 -llType $opt(ll)
                 -macType $opt(mac)
                 -ifqType $opt(ifq)
                 -ifqLen $opt(ifqlen)
                 -antType $opt(ant)
                 -propInstance [new $opt(prop)]
                 -phyType $opt(netif)
                 -channel [new $opt(chan)]
                 -topoInstance $topo
                 -wiredRouting OFF
                 -agentTrace ON
                 -routerTrace OFF
                 -macTrace OFF
```

```
void ECHO_Agent::timeout(int)
{
        sendit();
        echo_timer_.resched(interval_);
}

void ECHO_Agent::sendit()
{
        Packet* p = allocpkt();
        ECHOHeader *eh = ECHOHeader::access(p-\>bits());
        eh-\>timestamp() = Scheduler::instance().clock();
        send(p, 0);     // Connector::send()
}

void ECHO_Timer::expire(Event *e)
{
        a_-\>timeout(0);
}
```

A Slightly Different Approach...

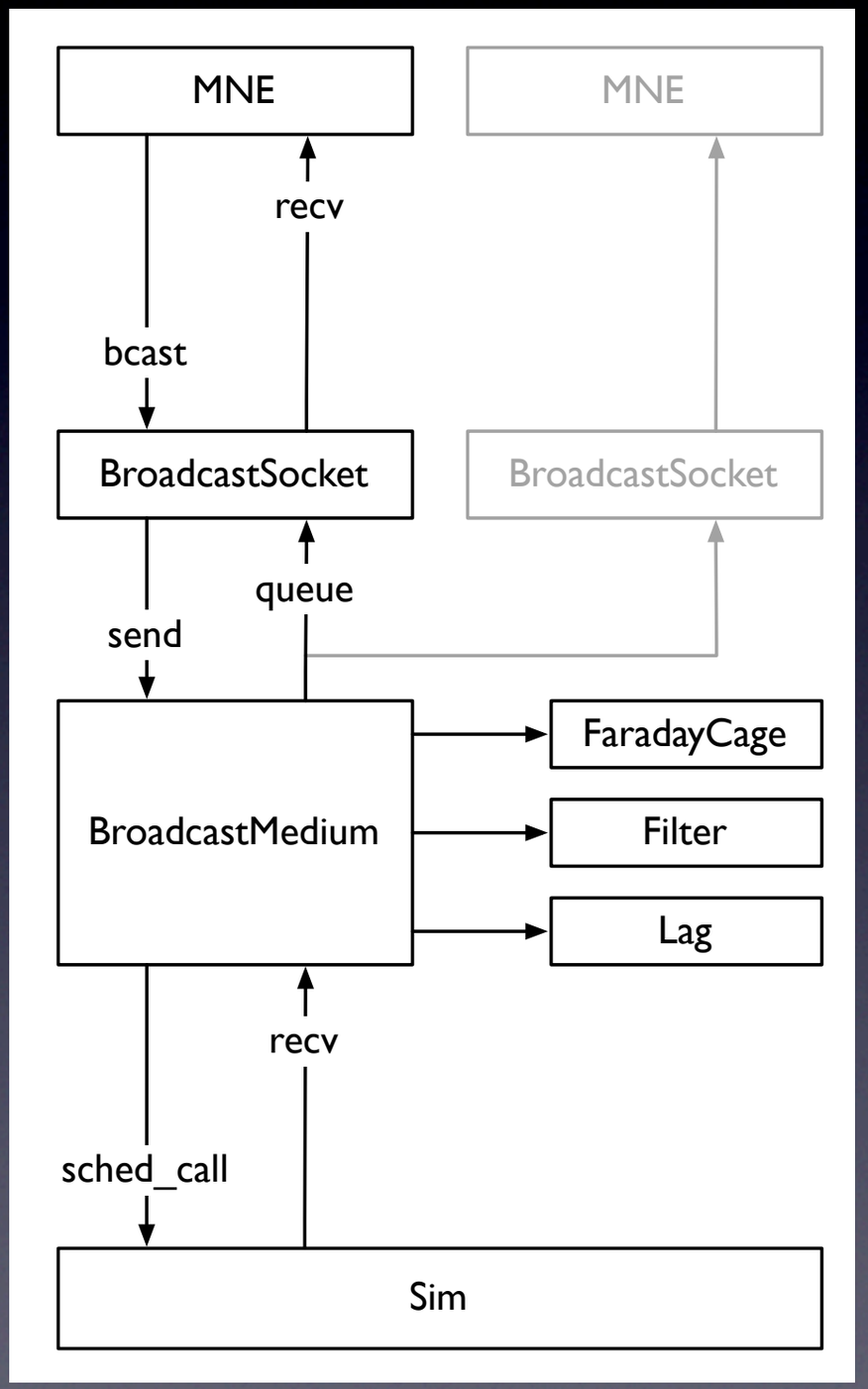
- Still event based
- Let people write simulator code that resembles real code
- Implemented in Stackless Python
- Shadow common system methods
  - e.g. time(), sleep(), select()
- Maintain thread of execute with tasklets

Timers in ns-2

```
void HelloTimer::handle(Event*) {
    agent->sendHello();
    double wait = min_wait +
      ((max_wait - min_wait)*Random::uniform());
    Scheduler::instance().schedule(this, &intr, wait);
}
```

```
def hello_timer():
    while 1:
        send_hello()
        wait = min_wait +
          (max_wait - min_wait)*random.random()
        time.sleep(wait)
```

Architecture



Configuration

- Two files
  - scenario files describe simulation
    - config. files describe parameters
- Change config values on the fly
- Example

Trivial Scenario

sim.cfg

```
medium:
 recv lag:
  bcast: 0.02
  local: 0.001
 filter:
   max_r: 20
rng:
 seed: 584456
```

scenario.py

```
import sim, random, math

config = sim.parseConfig("sim.cfg")
sim.config(config)
  ...
sim.run()
```

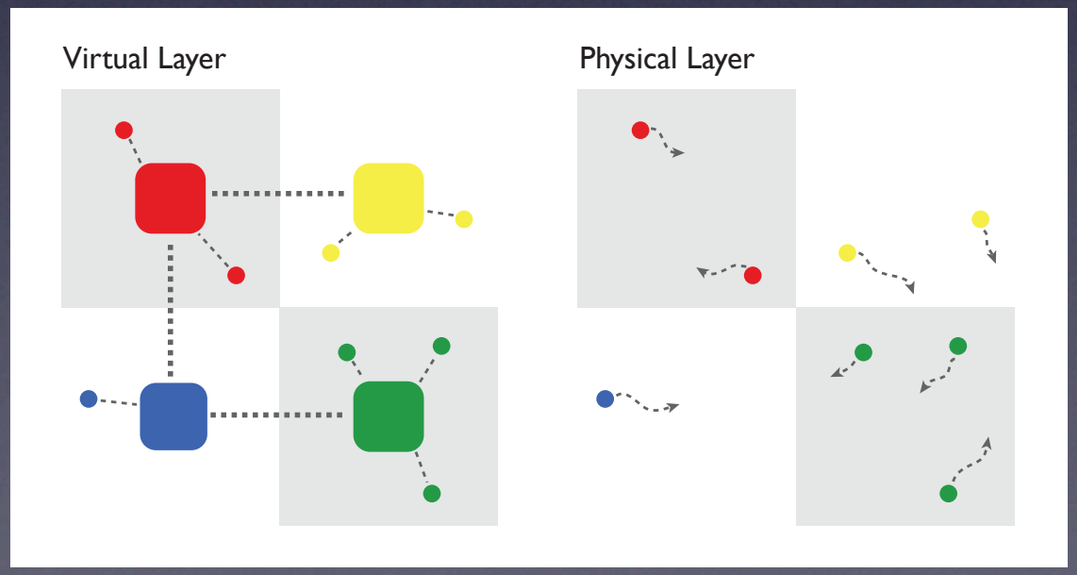
What's Missing...

- Radio and antenna models
- MAC layer implementation
  - Accurate collision modeling isn't possible yet

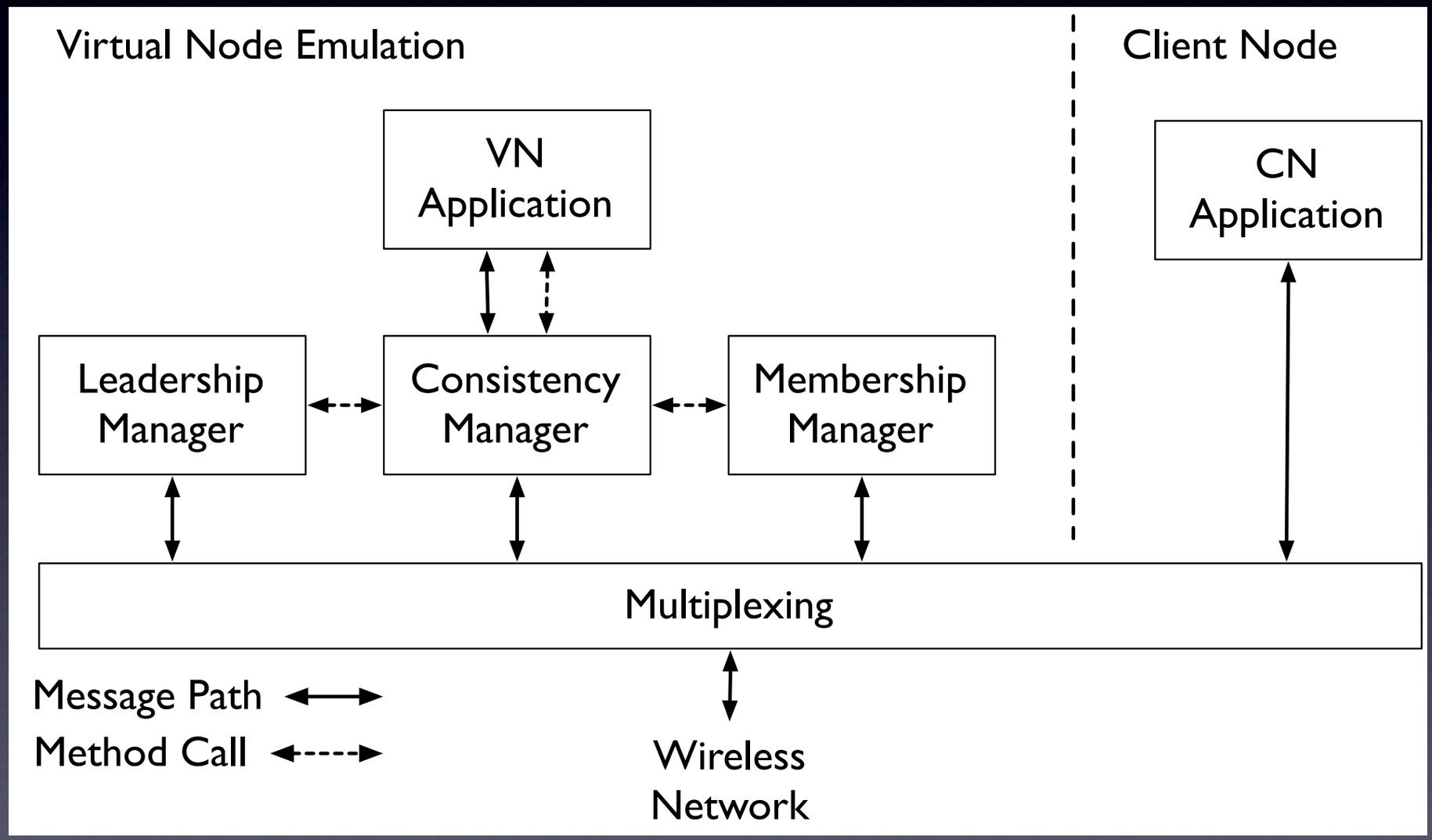
Simulating Virtual Nodes

- Implemented as a layer on top of the sim
- `import vnesim`

- Provides a variety of helpers



Virtual Nodes



vnodes.cfg

```
mobile nodes:
 number: 50
 client: path/to/client.MyClient
 paths: mobility.trace

virtual nodes:
 applications:
  - name: geocast
    location: path/to/vnapp.MyVnApp
 regions:
  - spec: grid [(0, 25), (25, 25), (5, 5)]
    neighbors: []
    apps: geocast

vne:
 services:
  ...
```

VN Scenario

```
import sim, vnesim, random, math

config = sim.parseConfig("vnodes.cfg")
vnesim.init(config)
sim_length = 3600
sim.run(sim_length)
```

VN Scenario

```
import sim, vnesim, random, math

config = sim.parseConfig("vnodes.cfg")
vnesim.init(config)
sim_length = 3600

for node in range(1, 4):
    t_stop, t_start = 0, 0
    while t_start < sim_length:
        t_stop = t_start - math.log( random.random() ) * 900
        t_start = t_stop - math.log( random.random() ) * 900

        vnesim.at(t_stop, "stopMobileNode(nodes['%d'])" % node)
        vnesim.at(t_start, "startMobileNode(nodes['%d'])" % node)

sim.run(sim_length)
```

Writing VN Apps

- Virtual node applications are reactive
- Clients are not
- Both app and client have interfaces
- Examples

Client Interface

```
class UpdateClient(vnclient.VNClient):
    def __init__(self, name, vn_map, mnode):
        vnclient.VNClient.__init__(self, name, vn_map, mnode)
        self.sock = vnesockets.ClientSocket(self)

def run(self):
    pass
```

App. Interface

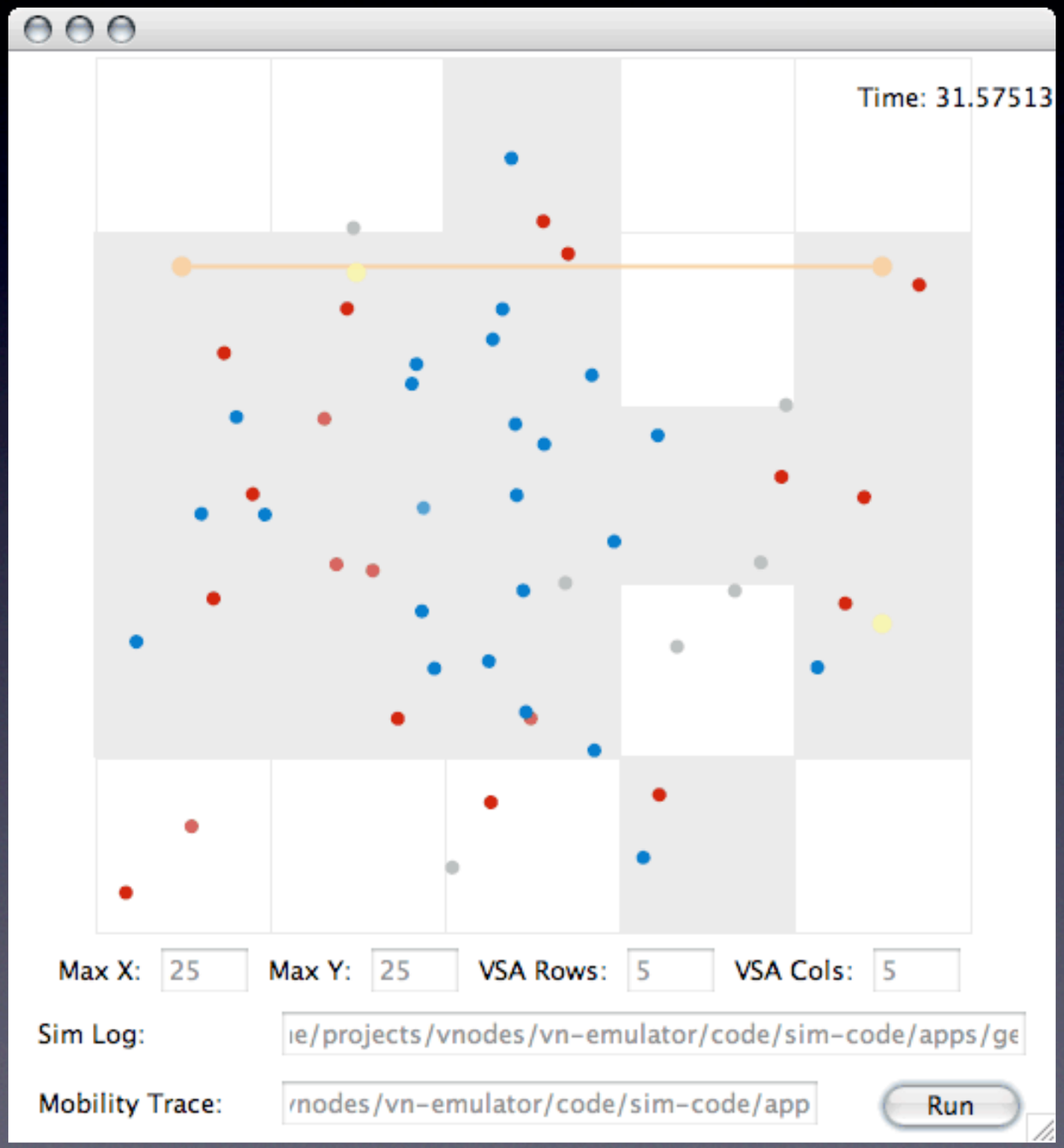
```
class MyApp:
    def __init__(self, region, vnmap, state=None, mnode=None):
        pass

    def getState(self):
        pass

    def loadState(self):
        pass

    def reset(self):
        pass

    def regionChanged(self, new_region):
        pass

    def msgReceived(self, message):
        return [replies]
```

Logging

- Lots of it
- Example:

```
mn:2   10.2   LeaderElect   hls,action Claiming leadership in (2,0)
mn:2   10.2   ConsistencyMgr   hls,action Attempting to synchronize
mn:2   10.2   ConsistencyMgr   hls,action Leading - performing a reset
mn:2   10.2   ConsistencyMgr   hls,info   synchronized
```

Visualization



VN Routing

- Implemented as three services
  - VN to VN communication
  - Node Location Tracking
  - End to End Routing

VN to VN

- Greedy geographic DFS
  - Messages are forwarded to the VN nearest the destination
  - End to end ack
  - Attempt second nearest VN after timeout
  - Timestamps used to filter old attempts

Node Location

- Each mobile node has some *home locations*
- Home locations are found by hashing the node's name
- VNs can find a client's region by querying its home
- End to end routing uses node location and VN to VN routing

Implementation

- VN based routing schemes implemented inside VN app / client scheme.
  - Traffic sources are just clients
- Other routing schemes done at lower sim level

Points of Comparison

- Other routing protocols
- Non-geographic
  - AODV, DSR
- Geographic
  - Naive, Greedy
  - GPSR

Calibrating Sim

- Seems worthwhile to repeat published tests
- Gauge simulator implementation
- Establish reference point for VN routing
- AODV seems a good candidate

Metrics

- Packet delivery fraction
- End to end delay
- Routing packets per delivered data packet
- Throughput

- vs. mobility
  - Pause time, speed, density.