

Probabilistic Coverage in Wireless Sensor Networks

Nadeem Ahmed

Salil Kanhere

Sanjay Jha


*Computer Science and Engineering,
UNSW.*

The Agenda




- Probabilistic coverage algorithm
- Mobility assisted probabilistic coverage
- ns-2 simulation script and results

The Problem

- 
- Wireless sensor network protocols often make simplifying assumptions such as the binary perfect-sensing-within-range model (unit disk)
 - Sensing capabilities are affected by environmental factors
 - Determine the *blanket coverage*, assuming randomly deployed sensor network, using a realistic sensing model
 - Degree of confidence in detection probability
 - Redundancy can be exploited for power management

The Approach

- 
- Assume that the signal propagation (from event to sensor) follows a probabilistic model
 - Valid for only certain types of sensors, e.g. acoustic, seismic etc.
 - Signal strength decays as a function of distance
 - Path loss log normal shadowing model
 - Propose a distributed computational geometry based approach
 - Only communication involved is finding the locations of nearby sensors

Log-Normal Shadowing Model

$$\overline{PL}(d) = \overline{PL}(d_0) + 10 \cdot n \cdot \log\left(\frac{d}{d_0}\right) + X_\sigma \quad (1)$$

$$Pr[Pr(d) > \gamma] = Q\left[\frac{\gamma - Pr(d)}{\sigma}\right] \quad (2)$$

- Overall detection probability of a point in the region

$$Pr = 1 - \prod_{i=1}^N (1 - Pr_i) \quad (3)$$

Log-Normal Shadowing Model

- For a given transmit power (event characteristic) and sensor receive threshold, detection probability at various distances can be pre-computed

A SAMPLE PROBABILITY TABLE (*PT*)

<i>Distance (m)</i>	<i>Probability</i>
3	0.997
6	0.90
9	0.655
12	0.41
15	0.245
18	0.135

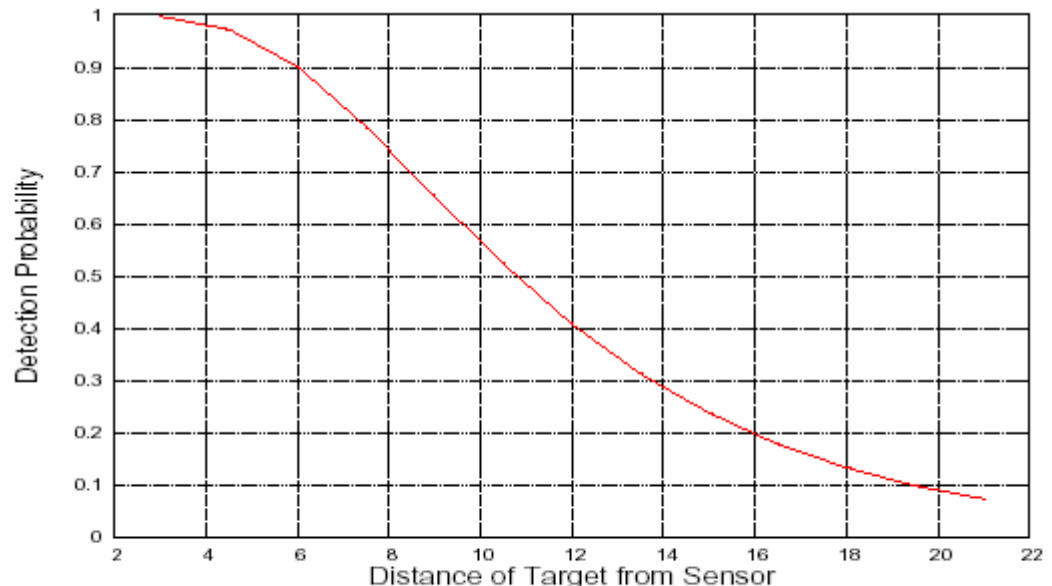
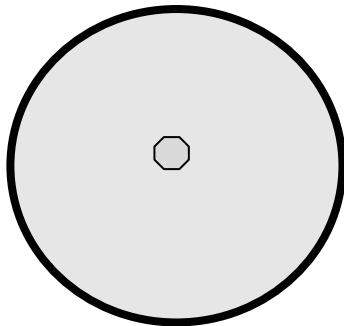


Fig. 1. Change in Detection Probability with Distance(m)

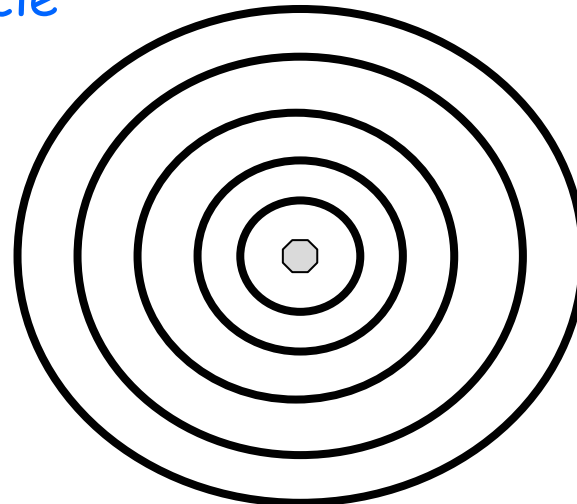
Probabilistic coverage



- The change in detection probability with distance can be represented by concentric circles drawn around the sensor location
 - Each circle represent the probability of correctly receiving a signal with strength above the receiving threshold at distance equal to the radius of the circle

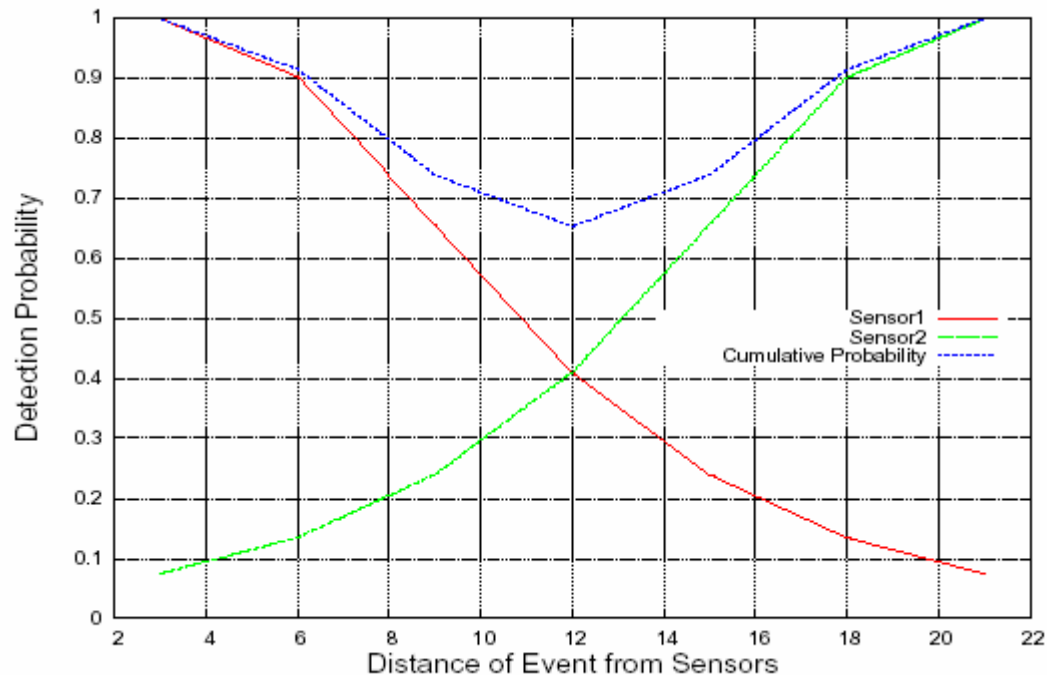


Binary, unit disc sensing model




Probabilistic sensing model

- Detection probabilities at any location is increased by neighbor contributions .



The Approach

- 
- ρ_{reqd} is the desired detection probability, d_{reqd} represent the distance from the sensor that provides ρ_{reqd} .
 - Exploit the neighbor contributions toward coverage and check whether the desired detection probability can be achieved at distances greater than d_{reqd}
 - Take the next higher distance from the Probability Table (PT).
 - Perimeter coverage as mean to ascertain area coverage [Huang et al. ACM WSNA 03]

➤ Definition 1:

Effective Coverage range, R_{effec} , of a sensor S_i is defined as distance of the target from the sensor beyond which the detection probability is negligible.

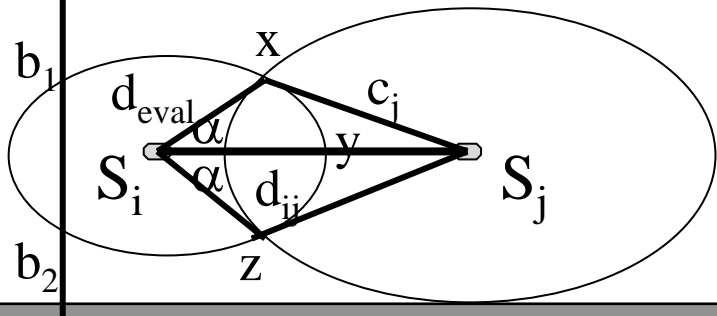
➤ Definition 2:

If the perimeter of a sensor S_i circle with radius d_{eval} is covered by cumulative detection probability, ρ , the region inside the circle is sufficiently covered with at-least ρ .

➤ The whole region, A , is sufficiently covered if all sensors in the region has perimeter of circle with radius d_{eval} , sufficiently covered with detection probability greater than or equal to the desired detection probability.

➤ Assumptions

- Random deployment
- Location information is available
- Communication range is at-least twice the effective communication range
- Sensors can detect region boundary if the boundary is within effective coverage range
- Obstacle free environment
- Mean values of path loss component, n , and shadowing deviation, σ for all the sensors.



Notations

ρ_{reqd} = Desired detection probability

d_{reqd} = Radius of circle around S_i that provides ρ_{reqd}

ρ_{eval} = Detection probability at next circle with $\rho < \rho_{reqd}$

d_{eval} = Radius of circle around S_i providing ρ_{eval}

ρ_{cumij} = Cumulative detection probability of S_i and S_j

R_{effec} = see Definition 1

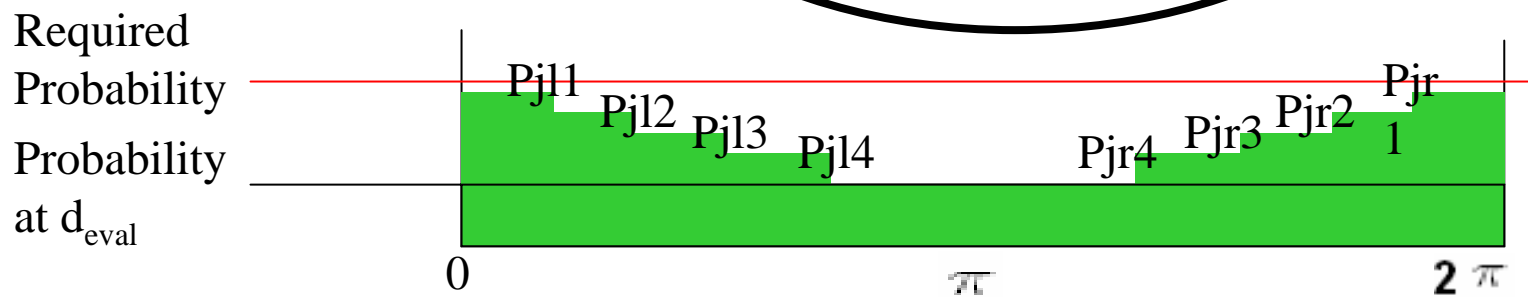
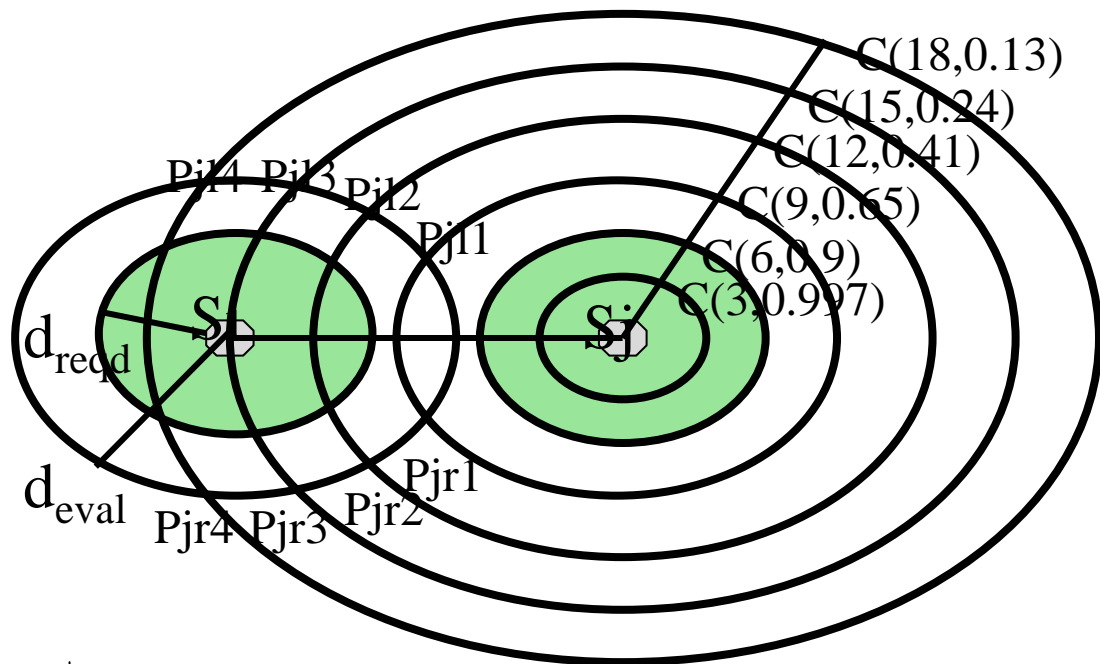
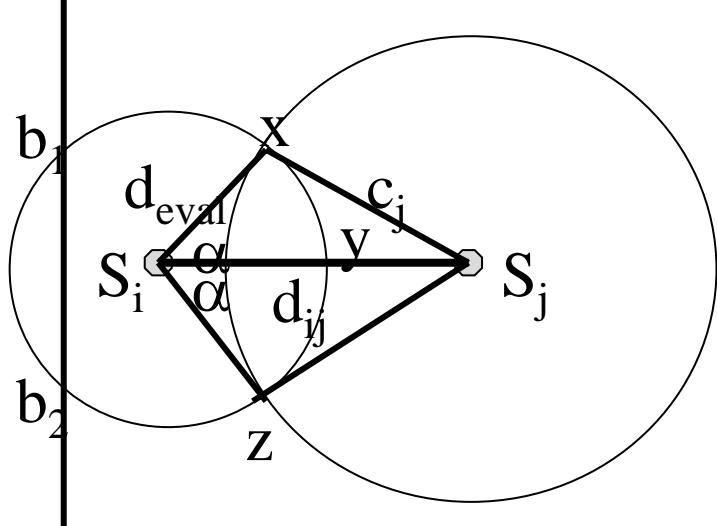
G_α = Angle subtended by the arc on perimeter of sensor S_i circle with radius d_{eval} that is covered by a neighbor

G_ρ = Cumulative probability of detection on perimeter of S_i circle with radius d_{eval}

$C_i(x)$ = Circle of S_i with radius x

- 1: ascertain ρ_{eval} and d_{eval} from PT
- 2: check boundary intersection with circle at d_{eval}
- 3: **if** $C_i(d_{eval})$ lies outside the region boundary **then**
- 4: mark segments on perimeter of $C_i(d_{eval})$ that are outside the boundary as sufficiently covered
- 5: **end if**
- 6: sort the neighbor list in ascending order of distance
- 7: **for** each neighbor j **do**
- 8: **if** $d_{ij} < d_{eval} + R_{effec}$ **then**
- 9: **for** each circle of S_j in $D(C_j(D_j))$ that intersects with $C_i(d_{eval})$ **do**
- 10: **if** $D_j < d_{eval}$ **then**
- 11: mark intersection point on perimeter of $C_i(d_{eval})$ as sufficiently covered by ρ_{reqd}
- 12: **else**
- 13: mark intersection point on perimeter of $C_i(d_{eval})$ as covered by $\rho_{cum_{ij}}$
- 14: **end if**
- 15: **end for**
- 16: update global G_α and G_ρ
- 17: sort G_α and G_ρ in ascending order on G_α
- 18: **if** G_α is all covered from 0 to 2π with $G_\rho \geq \rho_{reqd}$ **then**
- 19: declare all perimeter at $C_i(d_{eval})$ is sufficiently covered
- 20: end algorithm
- 21: **end if**
- 22: **end if**
- 23: **end for**
- 24: declare perimeter at $C_i(d_{eval})$ is not sufficiently covered

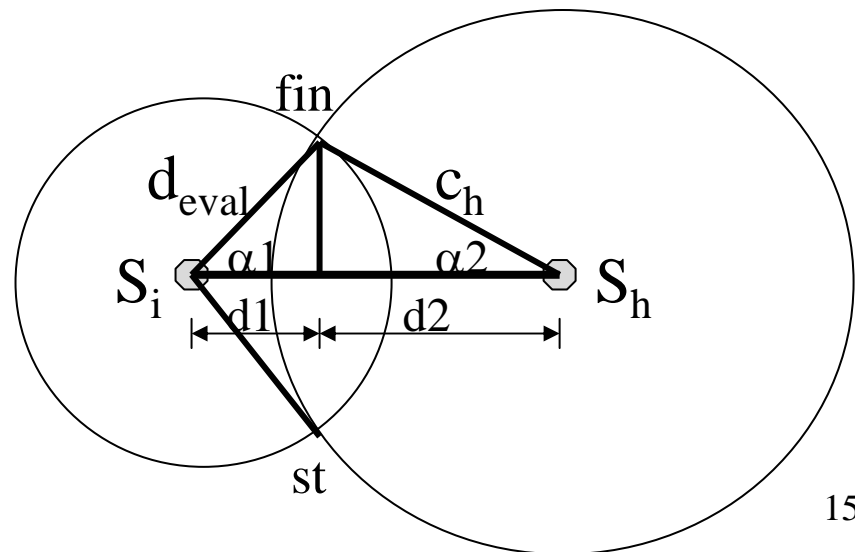




Extension

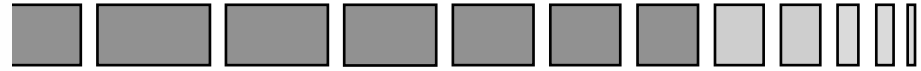
- Basic PCA gives a binary decision, a yes/no, whether the current deployment supports the required detection probability or not
 - Can be easily extended to identify the presence of *coverage holes*
 - Can also suggest possible deployment points in the region

$$\rho_{help} = 1 - \frac{(1 - \rho_{reqd})}{(1 - \rho_{exist})}$$

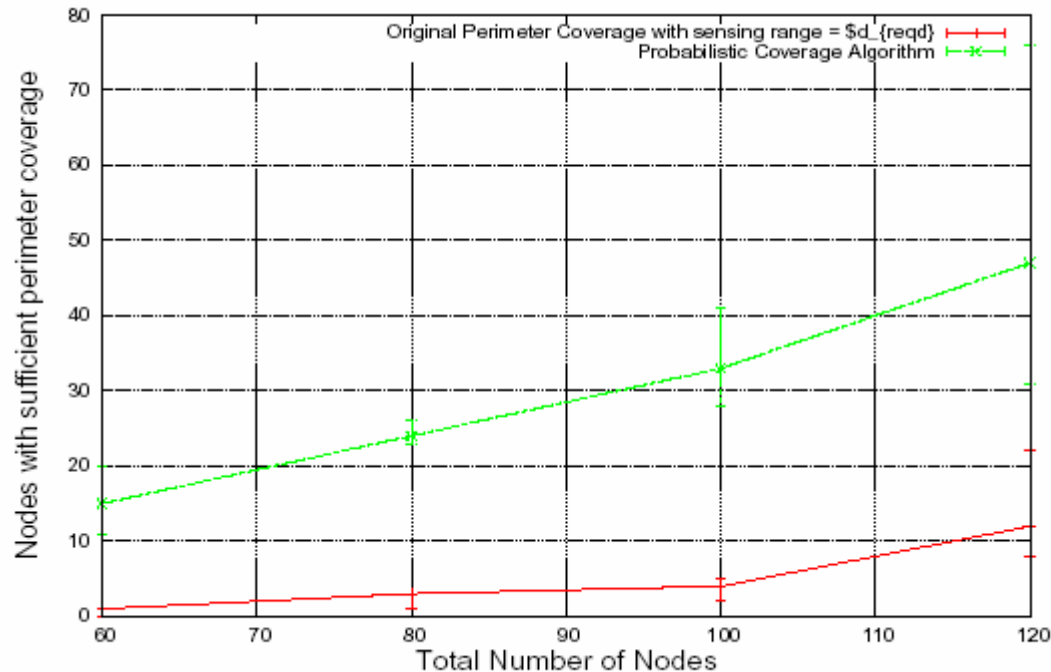


Parameter	Value
Transmit power P_t (Target)	24.5 dBm
Receiving threshold (γ) at sensor	-27.85 dBm
Path loss exponent n (free space)	2
σ	4 dBm
Effective coverage range	20m
Communication range	40m
Region (A)	100m x 100m
Number of nodes	60,80,100,120


Simulation



- Binary detection model (with sensing range set to d_{reqd}) underestimates the coverage.



Mobility Assisted Probabilistic Coverage

- 
- Hybrid network
 - Combination of mobile and static sensor nodes
 - All randomly deployed
 - Mobile nodes spread out executing Virtual Force Algorithm (Phase-I)
 - Attraction and repulsion forces
 - Boundary also exerts forces
 - Oscillation control mechanism
 - Static nodes run the PCA and determine the perimeter coverage (Phase-II)
 - Mobile nodes respond to help messages from the static sensor nodes (Phase-III)

ns-2 Simulator




➤ Discrete Event Simulator

- Simulator models world as events
- Single thread of control

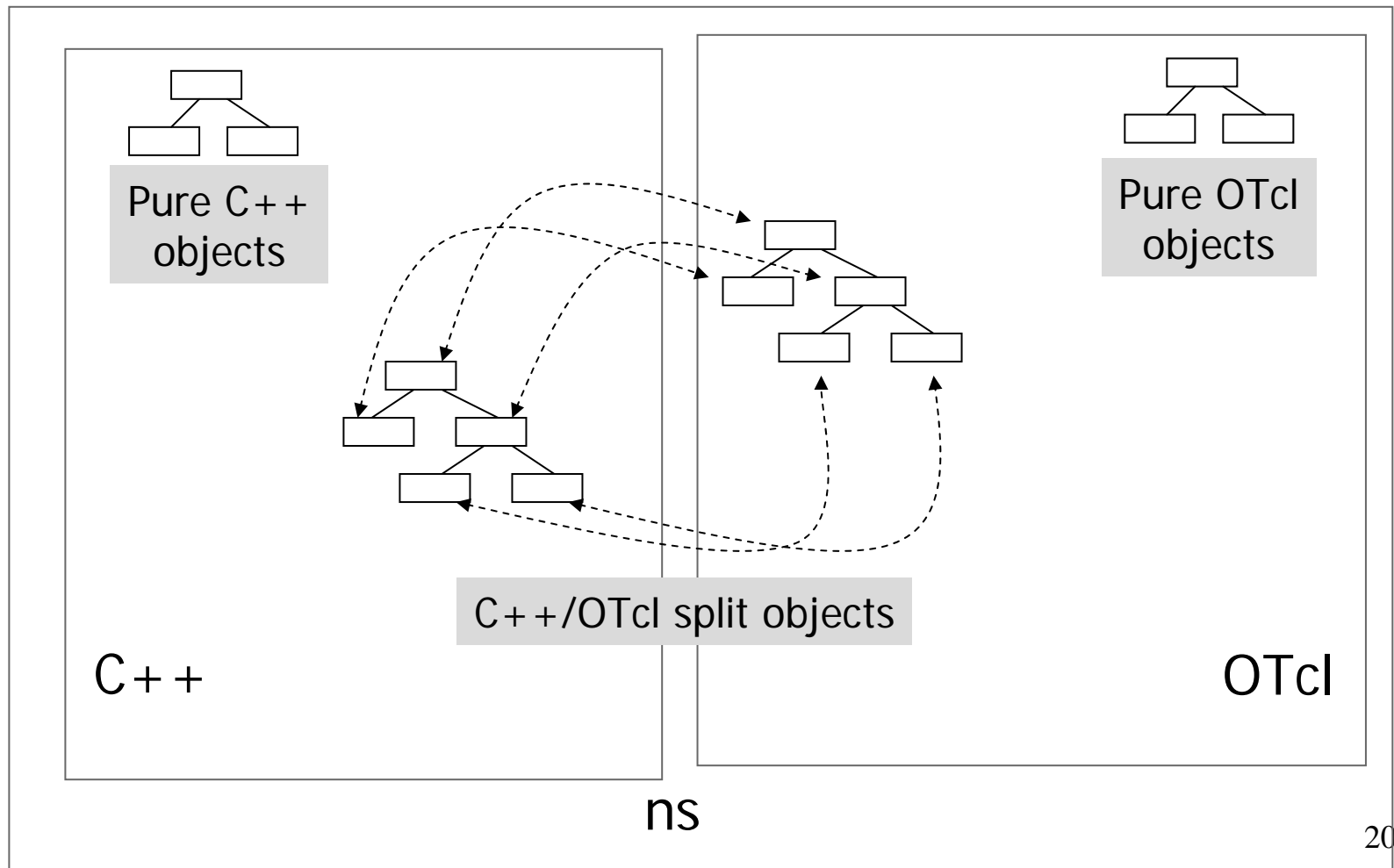
➤ Modeling network protocols

- Wired, wireless, satellite
- Ad-hoc, sensor networks
- Provides comparison of similar protocols


ns-2 Software Structure

- 
- Object-oriented
 - Uses two languages
 - C++ for "data"
 - » Per packet action.
 - » Fast to run
 - Otcl for control
 - » Simulation setup, configuration, triggered action
 - » Fast to write and change
 - Split C++/OTcl Objects


OTcl and C++: The Duality



ns-2 components

- 
- **ns, the simulator**
 - www.isi.edu/nsnam/ns
 - **nam, the Network AniMator**
 - Visualize ns output
 - **Pre-processing**
 - Topology and traffic generators
 - `~ns/ns2.27/indep-utils/`
 - **Post-processing**
 - Trace analysis, in Awk/Perl etc.

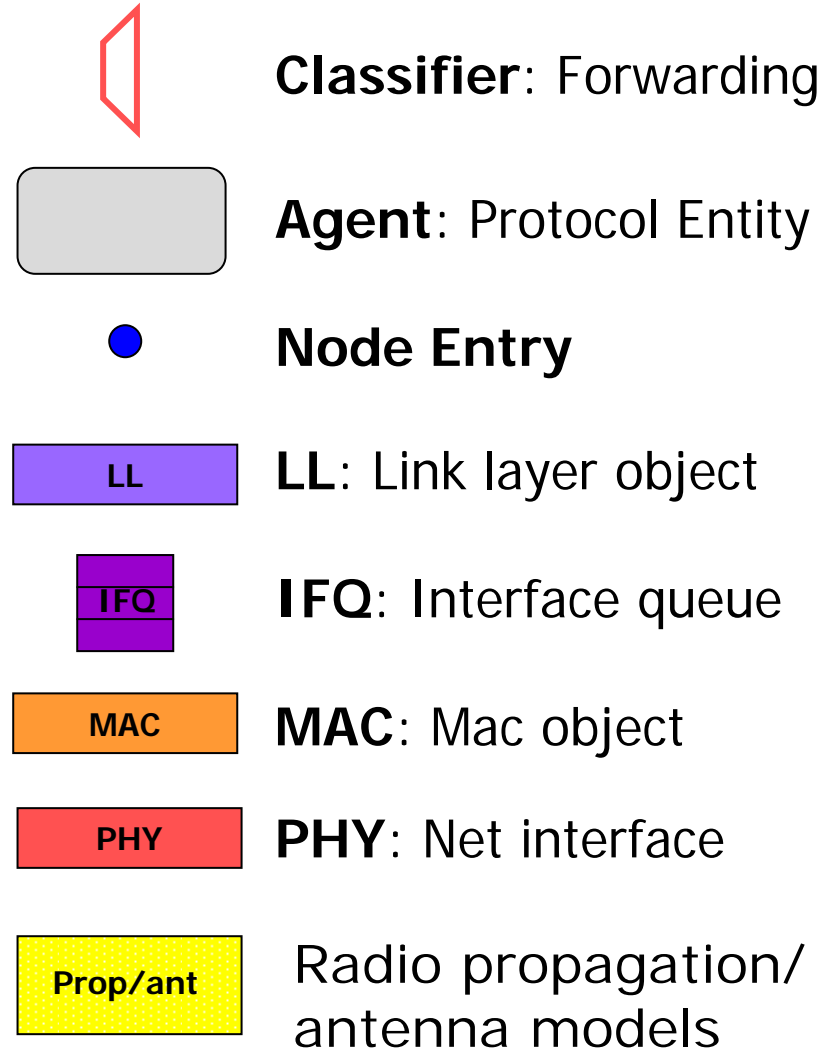
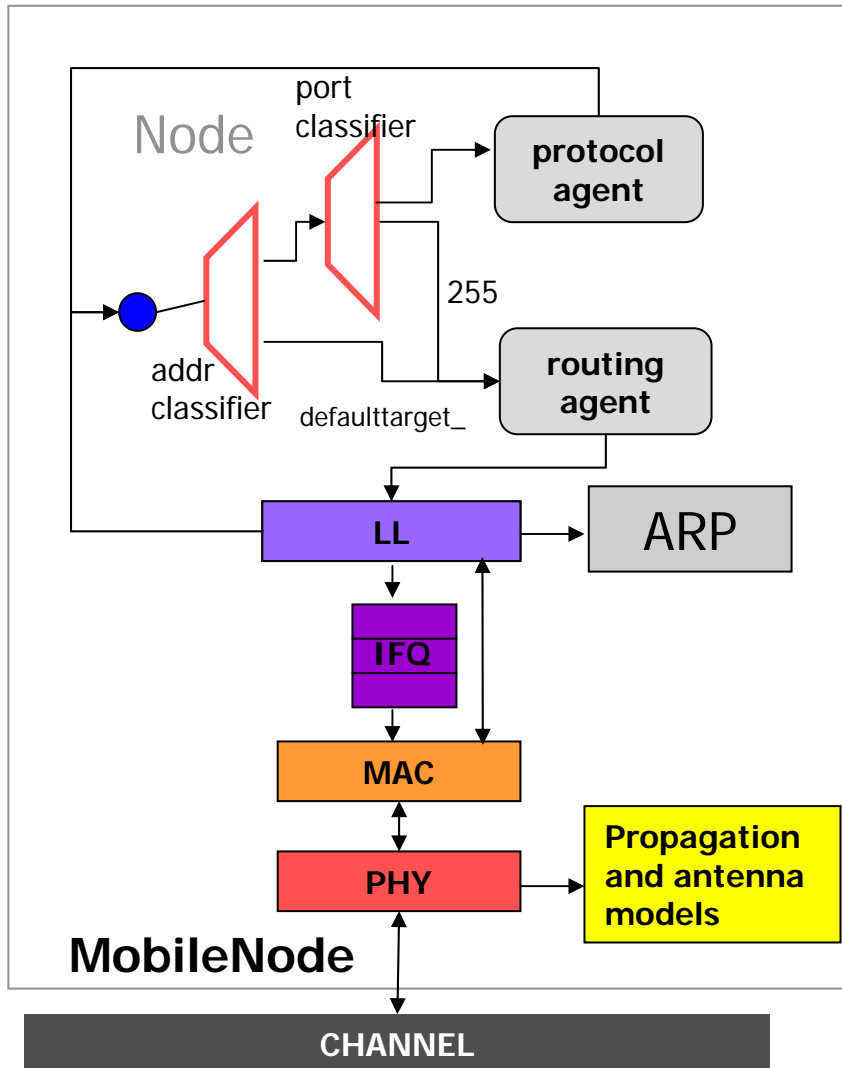
Elements of ns-2

- 
- Create the event scheduler
 - Turn on tracing
 - Create the network
 - Create traffic
 - End of simulation

Sample Script for MAPC

- 
- Create event scheduler
 - set ns_ [new Simulator]
 - Tracing
 - set tracefd [open mytrace.tr w]
\$ns_ trace-all \$tracefd
set nf [open out.nam w]
\$ns_ namtrace-all-wireless \$nf \$val(x) \$val(y)
 - Set up topography object
 - set topo [new Topography]
\$topo load_flatgrid \$val(x) \$val(y)


Portrait of A Mobile Node



Mobile Node Configuration

```
$ns node-config \  
-adhocRouting AODV \  
-llType LL \  
-macType Mac/802_11 \  
-ifqLen 50 \  
-ifqType Queue/DropTail/PriQueue \  
-antType Antenna/OmniAntenna \  
-propType Propagation/Shadowing \  
-phyType Phy/WirelessPhy \  
-channelType Channel/WirelessChannel \  
-topoInstance $topo \  
-energyModel EnergyModel \  
-initialEnergy 18.8 \  
-txPower 0.0891 \  
-rxPower 0.033 \  
-idlePower 0.001 \  
-agentTrace OFF \  
-routerTrace OFF \  
-macTrace ON
```

Setting the Parameters

- 
- Antenna/OmniAntenna set X_ 0
Antenna/OmniAntenna set Y_ 0
Antenna/OmniAntenna set Z_ 1.0
 - Propagation/Shadowing set pathlossExp_ 2.0
Propagation/Shadowing set std_db_ 4.0
Propagation/Shadowing set dist0_ 1.0
Propagation/Shadowing set seed_ 0
 - Phy/WirelessPhy set CPTresh_ 10.0
Phy/WirelessPhy set CSTresh_ 4.62633e-08
Phy/WirelessPhy set RXThresh_ 1.20174e-07
Phy/WirelessPhy set Pt_ 0.28183815
Phy/WirelessPhy set freq_ 914e+6

Create nodes

➤ Create mobile nodes

- for {set i 0} {\$i < \$val(nn)} {incr i} {
 set node_(\$i) [\$ns_ node \$i]
 \$node_(\$i) color blue
 \$god_ new_node \$node_(\$i) }

➤ Provide initial node positions

- \$node_(0) set X_ 10.0
 \$node_(0) set Y_ 10.0
 \$node_(0) set Z_ 0.0 OR
 set val(move) "~/simulations/topo/scenario-80-01"
 source \$val(move)

➤ Create agents and attach to node

- set p0 [new Agent/Move]
 \$ns_ attach-agent \$node_(0) \$p0

```
static class MoveClass : public TclClass {
public:
    MoveClass() : TclClass("Agent/Move") {}
    TclObject* create(int, const char*const*) {
        return (new MoveAgent());
    }
} class_move;

class MoveAgent : public Agent {
struct mLocation locs;
struct moveTo mt;
int flag;
struct mHelp help;
    MoveAgent();
    int command(int argc, const char*const* argv);
    void recv(Packet*, Handler*);
};
```

Schedule Events

➤ Schedule events

- `$ns_ at 1.0 "$p0 hello"`
`$ns_ at 5.0 "$p13 shello"`



➤ When to stop

- `for {set i 0} {$i < $val(nn)} {incr i} {`
`$ns_ at 500.0 "$node_($i) reset"; }`

`$ns_ at 500.01 "puts \"NS EXITING...\" ; $ns_ halt"`

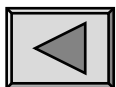
➤ Start simulation

- `puts "Starting Simulation..."`
`$ns_ run`

```

int MoveAgent::command(int argc, const char*const* argv)
{
    if (argc == 2) {
        if (strcmp(argv[1], "hello") == 0) {
            // get my own location
            double x_=0.0,y_=0.0,z_=0.0;
            Node *thisnode=Node::get_node_by_address(here_.addr_);
            ((MobileNode *)thisnode)->getLoc(&x_,&y_,&z_);
            //create a packet to send
            Packet* pkt = allocpkt();
            hdr_cmn *ch=HDR_CMN(pkt);
            hdr_move* hdr = hdr_move::access(pkt);
            hdr_ip* iph = HDR_IP(pkt);
            iph->daddr() = IP_BROADCAST; iph->dport() = iph->sport();
            hdr->type="HELLO";
            hdr->fx=x_;    hdr->fy=y_;
            // Send the packet
            printf("%f  Node %d sending packet type %s\n",
                Scheduler::instance().clock(),iph->saddr(),hdr->type);
            send(pkt, (Handler*) 0);
            return (TCL_OK);
        } .....
    }
}

```



Output

➤ Trace file

- Mytrace.tr

```
M 1.00000 11 (66.00, 36.00, 0.00), (0.00, 0.00), 0.00
s 1.000118396 _0_ MAC --- 0 PCSN 136 [0 ffffffff 0 800] -----[0:0 -1:0 32 0]
r 1.001206455 _58_ MAC --- 0 PCSN 84 [0 ffffffff 0 800] ----- [0:0 -1:0 32 0]
D 1.004733843 _32_ MAC COL 10 PCSN 136 [0 ffffffff a 800] ----- [10:0 -1:0 32 0]
M 2.00434 0 (10.00, 10.00, 0.00), (22.00, 29.44), 3.00
```

➤ nam output file format

- out.nam

```
+ -t 1.000118396 -s 0 -d -1 -p PCSN -e 136 -c 2 -a 0 -i 0 -k MAC
- -t 1.000118396 -s 0 -d -1 -p PCSN -e 136 -c 2 -a 0 -i 0 -k MAC
h -t 1.000118396 -s 0 -d -1 -p PCSN -e 136 -c 2 -a 0 -i 0 -k MAC
r -t 1.001206419 -s 1 -d -1 -p PCSN -e 84 -c 2 -a 0 -i 0 -k MAC
```

Phase-I

*****This is Mobile Node 7

12.102530 Received packet type HELLO

From node 3.0 at Location 31.624344 36.967522 ,

My location is 27.528023 33.387974

*****Neighbor list for mobile Node 7

0 27.528023 33.387974

1 49.952408 20.000000

2 31.242449 51.616062

3 20.000000 53.680698

4 21.997828 29.444267

5 31.624344 36.967522

6 50.000000 50.000000

Distance is 26.116869 offset is 4.441566

force -3.813603 2.276826

Distance is 18.602692 offset is 8.198654

force -1.637037 -8.033557

Distance is 21.644072 offset is 6.677964

force 2.322662 -6.261026

Distance is 6.792340 offset is 14.103830

force 11.483073 8.188837

Distance is 5.439946 offset is 14.780027

force -11.129475 -9.725430

Distance is 27.945467 offset is 3.527267

force -2.836405 -2.096764

Fx -5.610785 Fy -15.651114

Old location 27.528023 33.387974 New location 21.917238 20.000000

Custom tracing,
using printf / cout from C++ code

Phase-II

*****Neighbor list for static Node 14

0 1.000000 54.000000
1 11.000000 55.000000
2 12.000000 44.000000
3 22.000000 54.000000
4 29.000000 55.000000
5 20.000000 75.000000
6 14.000000 80.000000
7 32.000000 45.000000
8 33.000000 46.000000
9 36.000000 55.000000
10 32.000000 37.000000
11 19.000000 21.000000

Custom tracing,
using printf / cout from C++ code

Check Boundary =====1.682135 4.601045

Check Boundary re-oriented =====2.180559 5.099469

Distance between nodes 1.000000 54.000000 and 11.000000 55.000000 is 10.049876

Distance between nodes 1.000000 54.000000 and 12.000000 44.000000 is 14.866069

-- -- -- -- -- -- -- --

Distance between nodes 1.000000 54.000000 and 19.000000 21.000000 is 37.580000

Max Uncovered from 0.000000 to 0.127170 difference 24.500000

Centre point 0.063585

Required probability is 71.014493 use circle 6 m

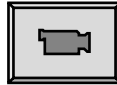
*****HELP needed 7.385930 40.477543

Node 14 sending HELP packets

Output

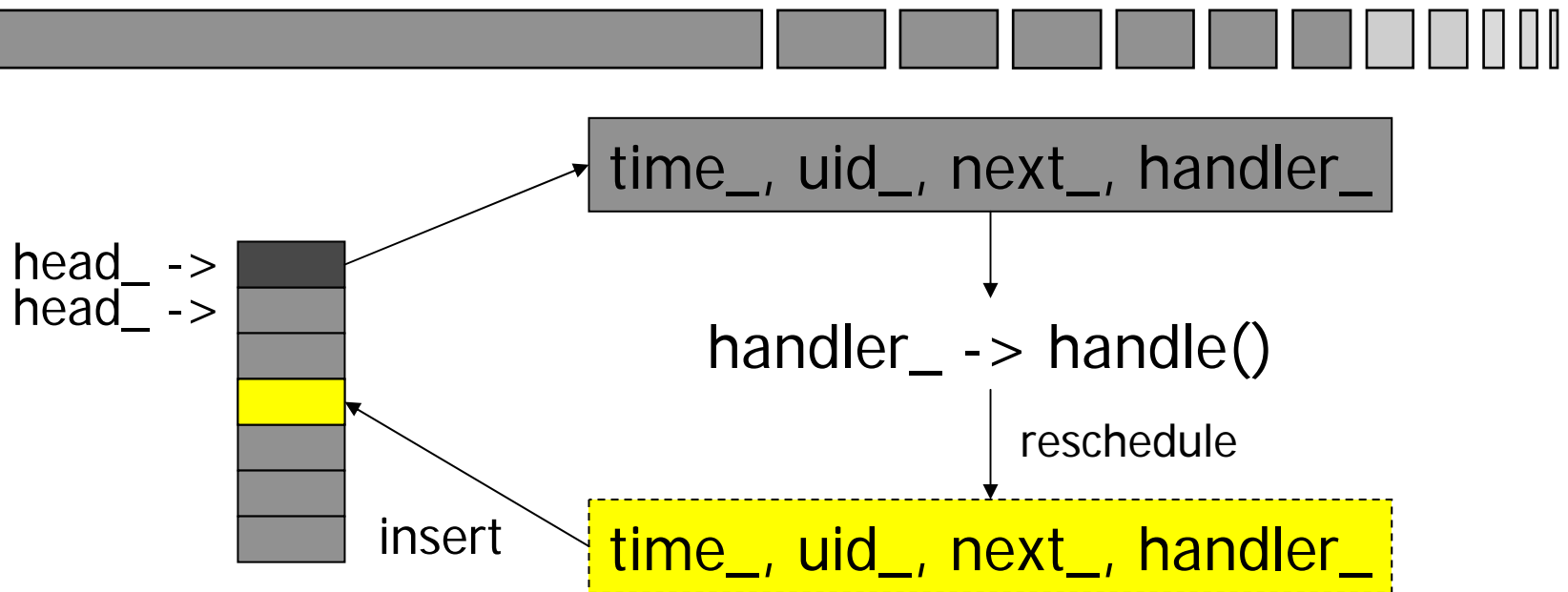


➤ `nam out.nam &`





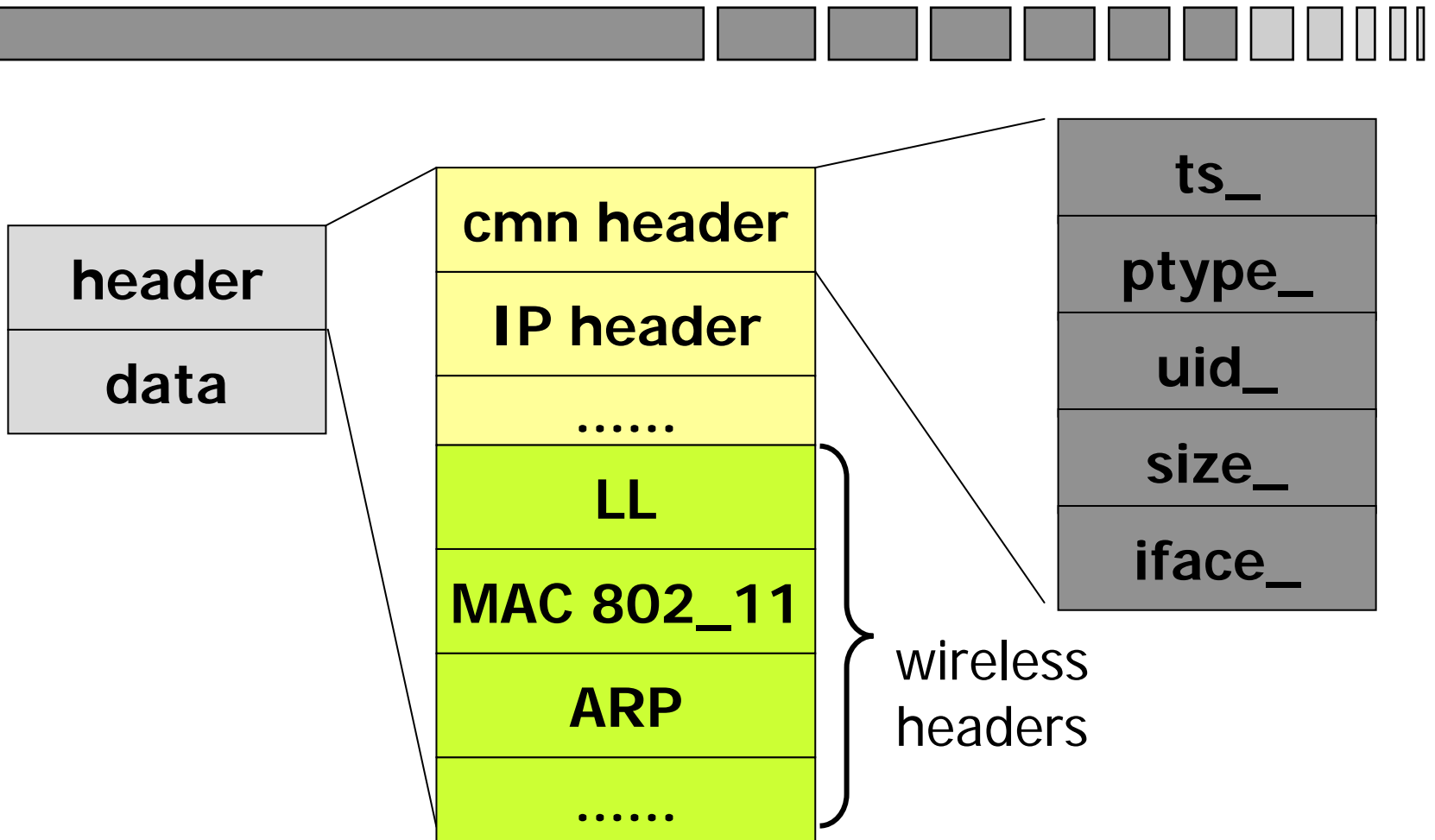
Discrete Event Scheduler



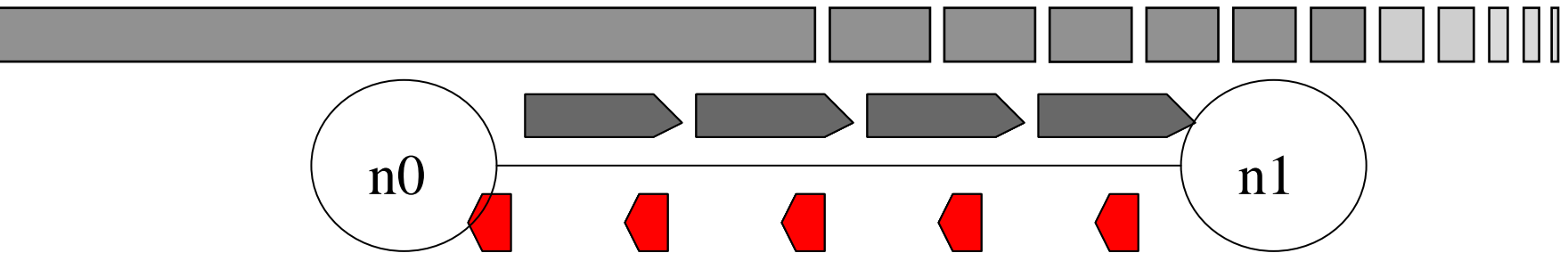
➤ Four types of schedulers

- List: simple linked list, order-preserving, $O(N)$
- Heap: $O(\log N)$
- Calendar: hash-based, fastest, $O(1)$
- Realtime

Headers



Example : TCP



```
set ns [new Simulator]
set n0 [$ns node]
set n1 [$ns node]
$ns duplex-link $n0 $n1 1.5Mb 10ms
    DropTail
set tcp [new Agent/TCP]
set tcpsink [new Agent/TCPSink]
$ns attach-agent $n0 $tcp
$ns attach-agent $n1 $tcpsink
$ns connect $tcp $tcpsink
```

```
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 0.2 "$ftp start"
$ns at 1.2 "exit"
$ns run
```