

sQualNet version 1.0 beta: Tutorial and Programmers Manual

Maneesh Varshney, Rajive Bagrodia
Parallel Computing Lab
UCLA

Advait Dixit, Balaji Vasu, Parixit Aghera, Mani Srivastava
Network and Embedded Systems Lab
UCLA

Contents

1	Sensor Network Simulation	3
1.1	sQualnet design and architecture	3
1.2	Models for sensor network simulation	4
1.3	Installation	5
2	Running a Simple Scenario (with IP network)	6
2.1	Defining sensors	7
2.2	Defining the traffic	8
2.3	Defining Network and MAC Protocols	8
2.4	Defining the battery model	9
2.5	Defining the Processor Model	9
2.6	Hardware Power Consumption Model	10
2.7	Putting it all together	10
3	Running non-IP protocols	11
3.1	Diffusion Network Protocol	11
3.2	Defining Multiple Network Protocol Stacks	11
3.3	Multiple network protocol application	12
4	Complete APIs	13
4.1	Sensor Channel and PHY	13
4.2	Battery model	14
4.3	Diffusion Network Protocol	14
5	NesC Code Simulation	16
5.1	Installation requirement	16
5.2	Modifications to TinyOS distribution	17
5.3	How to generate a new application	17

5.4	How to add a new application	17
5.5	How to use configuration file	18
6	Future Work	20

Chapter 1

Sensor Network Simulation

Sensor networks have gathered a lot of research interest in recent years. With their unique design issues and typical deployment scenarios reaching large numbers, the need for scalable and detailed simulation framework is becoming apparent. We present sQualnet, a scalable and extensible simulation framework that offers detailed models for sensor networks. The simulator is build as extension to QualNet network simulator ([6]). This document provides a tutorial for setting up sensor network simulation and a programmer's manual for using the APIs. A basic familiarity with using QualNet for network simulations is assumed, and the readers are referred Qualnet's developers manual reviewing QualNet APIs.

Sensor network simulation is a challenging task and traditional network simulators are not suited for multiple reasons. The sensor nodes "sense" some phenomenon, a feature not modelled in current network simulators. Also the protocol models are different in sensor networks, including the trends to move away from IP based networking stack and even from layered stack. In network simulators it is not easy to define new network protocols or to have multiple running simultaneously on same node. Moreover, as these nodes are powered by battery with little possibility of recharging, there is a need for faithfully modelling the power consumed by various hardware components and also the battery. The power consumed by processor in execution the protocol and application code, hitherto not considered, can be significant proportion in sensor nodes. Additionally, the traffic models in sensor networks are different than those traditionally assumed in mobile ad-hoc networks.

All these observations highlight the need to have a next generation of network simulators that faithfully and efficiently models the specific features of sensor networks. We present the sQualNet framework to bridge this gap in the current simulators. This simulation framework is different from emulation based approaches which are not extensible and scalable, or the exclusive sensor model simulators which are not detailed enough and accurate in modelling other aspects of networking.

1.1 sQualnet design and architecture

sQualNet is build as an extension to QualNet network simulator and utilizes the various data structures and code organization defined in the simulator. The basic entity is a "node", which has data structures for the various protocol layers. We have extended this model to what is shown in fig 1.1.

The general diagram for the node architecture is as follows. Note that the are two stack with the node. One is the communication stack and the other is sensing stack. The Communication stack can be further divided into IP based and non-IP based protocol. The Power and Battery models also work with this node.

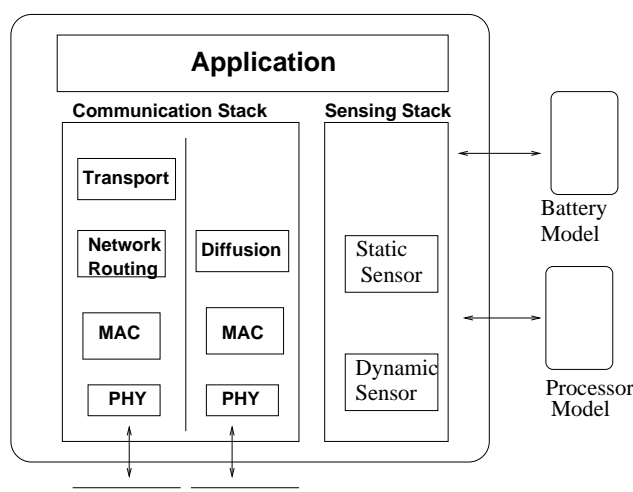


Figure 1.1: Sensor Node Architecture

1.2 Models for sensor network simulation

This version of sQualNet simulator provide following models in addition to the models provided by QualNet network simulator:

1. **Sensor channel and PHY.** The sensor nodes have the feature to "sense" the environment. The environment is modelled as *sensor channel* while the sensing device as *sensing PHY*. Consider the analogy with radio channel and device. The wireless radio channel describes how the waves propagate in the medium under the condition of fast and slow fading, shadowing etc. The radio models the reception of these information carrying waves. The sensing channel models the propagation of events (e.g. seismic waves generated by moving vehicles, acoustic waves by animals, toxin concentration by chemical explosion etc). The sensing phy observes these phenomenon and reports values at the given sensor node location.
2. **Battery Model.** The typical model of battery as reservoir of charge fails to faithfully model the non-linear discharge and recovery effects of real battery. There are more realistic models ([5]), but execution speed prohibitively slow. We have implemented an optimized model as presented in ([7]).
3. **Processor Power Consumption.** Power consumption models in MANETs have considered radio power consumption only. However, in sensor nodes the power consumed by processor while executing protocol and application code can be substantial. In sQualNet we provide models for accounting this component.
4. **Sensor Traffic Model.** Typical traffic models used in MANETs are CBR, FTP, HTTP etc which are more or less random source-destination based communication. In sensor networks, however, the traffic is more likely to be from group of spatially close nodes to fixed gateway or monitoring stations. We provide this sensor network specific traffic model.
5. **Non-IP and Multiple Network Protocol.** In sensor networks, recent trend has been to move away from IP based stack. We provide Diffusion ([3]) protocol stack and capability to have multiple network protocols in a single node.
6. **Hardware power consumption.** Faithfully model the power consumption of commonly used sensor nodes such as Mica Motes ([2])and WINS([1]).

1.3 Installation

Copy the `squalnet.tgz` in the base `qualnet` directory (`$QUALNET_HOME`) and unzip. This will generate following new files: `sensor.cpp`, `sensor.h`, `sensor_app.cpp`, `sensor_app.h`, `power.h`, `power.cpp` (in `addons/seq` directory), and `diffusion.h`, `tinydiffusion.h`, `difflist.h`, `difflist.cpp`, `diffusion.cpp` (in `network` directory). Now run the patch file (`patch -p1 < sQualnet-3.7.patch`) from the base directory. Go to main and do *make clean; make*.

Chapter 2

Running a Simple Scenario (with IP network)

Let us consider a simple topology as shown in 2.1. There is a network of sensor nodes, some of which can detect chemicals in air, while others can detect seismic vibrations from a moving vehicle (there can be nodes which can do both). The dashed circle represents a chemical outbreak and the sensors in the vicinity can detect it. Similarly, the other kind of sensors can detect the moving vehicle when it is close to them. When the sensor nodes observe such an event, they send packets to the gateway node using the sensor network protocols. As the information reaches the gateway node (the rectangle node), it forwards to monitoring station using traditional IP based network. In simulation we are also interested in monitoring the power consumed by each node, both the radio hardware as well as processor. In this section we assume that all the nodes are running on IP network. In section ?? we look into how to configure diffusion protocol and have multiple network protocols in a single node.

In the following sections we describe step by step procedure to set up such a simulation scenario.

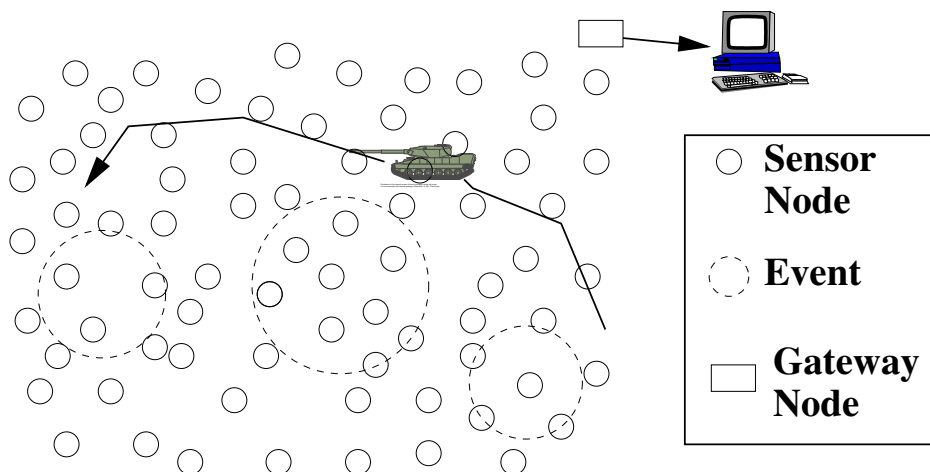


Figure 2.1: Sensor Network Scenario

2.1 Defining sensors

As mentioned previously a new feature added in the sQualnet framework is the capability to *sense* an event. Each sensor node is equipped with "Sensor PHY" which monitors the environment (the "Sensor Channel") and reports the magnitude of phenomenon observed.

Defining Sensors. We distinguish between two sensor types: Static and Dynamic sensing channels. In the former case, the events happen at fixed locations in space and are propagated to nearby regions. An example is chemical outbreak, where the "events" are originated as some fixed locations and are spread following some propagation model. In the dynamic sensing channel, on the other hand, the "events" are generated by an object that is moving. An example is moving vehicle that is generating seismic waves. These two types of sensors can be defined as

```
[1 thru 30]   SENSOR-DEVICE[0]   STATIC
[20 thru 50]  SENSOR-DEVICE[1]   DYNAMIC
```

The above lines configures sensing device for nodes 1 through 30 that is of static type. It also configures a dynamic phy for nodes 20 through 50 of dynamic type (node 20-30 have both kinds of sensors). Note the use of instance identifiers to define multiple sensor devices in simulation. These instance IDs should always be different from others, even if the nodes they are configured in are different.

Defining Events. Having defined the channel type we have to next define how and where the events are generated. For static channel type we have to describe the location where events occur, their start time and finish time. This can be either defined manually or randomly. For manual configuration:

```
SENSOR-DEVICE-EVENTS[0]           FILE
SENSOR-DEVICE-EVENTS-FILE[0]      ./squalnet.events
----- squalnet.events -----
# <event-id> {coordinates of event} start_time finish_time
0 (100, 100, 0) 2S 5M
1 (400, 200, 0) 1M 10M
```

The first two lines define that the events are generated as described in the file. The *.events* file define two events: first at (100, 100) which starts at 2S and finish at 5M, and second at (400,400) starting at 1M and finishing at 10M. In our example these are location at which the chemical outbreak occurs and also at what time it starts and ends. We can also define these events randomly, as follows:

```
SENSOR-DEVICE-EVENTS[0]           RANDOM
SENSOR-DEVICE-NUM-EVENTS[0]       10
SENSOR-DEVICE-EVENTS-DURATION[0]  4M
```

The above lines define 10 random events, with duration exponentially distributed with mean 4M.

For dynamic sensing channel, we have to define node(s) that are moving and generating events. This is a regular QualNet node and its mobility pattern is defined in the standard manner as the node mobility is defined in qualnet.

```
[55] SENSOR-DEVICE[3]           DYNAMIC-SOURCE
[55] MOBILITY                    WAYPOINT-RANDOM
```

Defining Propagation Parameters. We have defined when and where the events occurs, we now have to define how the events are propagated in space. There are two models for propagation: wave propagation and diffusion. In the former case, the propagation follow the inverse power distance pathloss model (i.e.

$P_{recv} = P_{transmit}(\frac{d_0}{d})^\alpha$. In diffusion model, the "waves" propagate following the Ficks' model of mass diffusion. In this version, we provide the wave propagation model only.

In order to define this pathloss model, we have to provide transmit power, pathloss exponent, propagation speed and sensing threshold. These can be defined as follows:

```
SENSOR-DEVICE-PATHLOSS-EXP[0]      2
SENSOR-DEVICE-TX-POWER[0]          0      # dBm
SENSOR-DEVICE-THRESHOLD[0]         -100   # dBm
SENSOR-DEVICE-PROP-SPEED[0]        100    # m/s
```

Thus the "event" will be transmitted at 0 dBm and suffers pathloss with exponent equal to 2. Note that the sensing threshold value effectively restricts the region in which this event can be sensed. Also the events are propagated with speed of 100 m/s, which models the propagation delay. However, if we are interested in a propagation model, in which the events are propagated instantaneously, that is, at infinite speed, then we can define such a model as follows

```
SENSOR-DEVICE-PROP_SPEED[1]         0      # m/s
SENSOR-DEVICE-RADIUS[1]             100    # m
```

Here we have configured the second sensor channel to be have *infinite* propagation speed. We have also defined a "radius", which restricts the region in which this event can be observed.

2.2 Defining the traffic

So far we have configured the sensing device in each node and defined when and where the events are generated and how are they propagated. Let us now set up a traffic model for this scenario. The typical traffic models used in wired and wireless networks are CBR, FTP, HTTP etc. All of them are point-to-point random source-destination pair based traffic models. In sensor networks, however, we observe that the traffic is typically from spatially close group of nodes (nodes that sense the same event) to a fixed destination (the gateway node). We provide this sensor network specific traffic model: SENSOR-APP and DIFFUSION-APP. The difference between the two is that the former uses the IP protocol stack (UDP and IP+routing protocol), while the latter uses the diffusion protocol stack. We defer the discussion on the latter until we have discussed the diffusion protocol and multi-protocol stack in sec ??.

The traffic model is described in `.app` file, and the format is

```
# SENSOR-APP  sensorId  numServers  <server-list>  packet-size  rate  threshold
SENSOR-APP  0  2  110  111  64  250MS  -40
SENSOR-APP  1  1  110  200MS  -35
```

The first parameter is the sensor Id for which this application is running. Following it we define the gateways or the base stations that receive the sensing data. There can be more than one gateways (for load balancing etc), and each node randomly pick one destination from the list. The *threshold* value defines the limit, that is, if the sensor value is greater then this threshold then the node will send a packet of size *packet-size* at the *packet-rate* to its chosen destination. For example, in the second line, for each node having sensing channel type "0" (the STATIC sensor in our running example), if they observe value greater then -40dBm they will send a packet of size 64B to either node 110 or 111 every 250MS.

2.3 Defining Network and MAC Protocols

SMAC ([8]): This medium access protocol puts a node in alternate states of sleep and listening mode to

conserve energy. "SYNC" packets are used to synchronize the sleep schedule of neighbor nodes. It can be defined as:

```
MAC-PROTOCOL S-MAC
SMAC-SYNC-FLAG YES/NO # default NO
```

Note: This MAC protocol does not work properly with AODV, DSR etc ad-hoc routing protocols with their typical settings. The SMAC has higher latency for packet delivery, which the routing protocol mistakes as destination unreachable. The users are advised to use modify the ad-hoc protocol timer settings or use static routes or use sensor network specific routing protocols.

2.4 Defining the battery model

Battery is the reservoir of energy in each node. The simulator monitors how much energy is consumed by each node, and if the capacity goes down to zero, it turns off the node. The battery model is defined by BATTERY-MODEL parameter, and it can take following three values:

- NONE. This is the default value. Consumption of energy by node is not monitored.
- SIMPLE. The battery is modeled as reservoir of some given capacity (defined by SIMPLE-BATTERY-CAPACITY parameter). After any event that consumes power (e.g. radio transmission, reception, cpu processing etc) some amount is decremented from remaining capacity. When this value goes down to zero, then we assume the node is dead. Note that this is a simple model and does not exhibit the recovery and non-linear discharge observed in actual battery ([4]).
- ACCURATE. This is a more realistic model proposed in [5] with implementation optimizations described in [7]. We have calibrated the model parameters for typical batteries used commonly.

An example script to show the above is

```
[101, 110, 111] BATTERY-MODEL    NONE

[1 thru 50] BATTERY-MODEL        SIMPLE
SIMPLE-BATTERY-MODEL-CAPACITY    10000

[51 thru 100] BATTERY-MODEL      ACCURATE
ACCURATE-BATTERY-TYPE            DURACELL-AA
# ACCURATE-BATTERY-TYPE          DURACELL-AAA, DURACELL-9V, ITSY (other parameters)
```

2.5 Defining the Processor Model

A new feature in the sQualnet framework is to account for the power consumed by the processor when executing the protocol and application code. For sensor nodes that run on low-end microprocessors or microcontrollers this can be a relatively high proportion. The technique for modelling processor power consumption is described in [7]. We have calibrated only a part of simulator framework. Currently we are modelling power consumed in executing MAC 802.11 protocol, IP and AODV. In next releases, and as the community interest suggests, we will release model for other protocol models. The processor type is StrongARM SA1100, used in WINS ([1]) sensor node.

```
[110, 111] PROCESSOR-MODEL      NONE
[1 thru 100] PROCESSOR-MODEL    WINS
```

2.6 Hardware Power Consumption Model

There is wide range in sensor hardware regarding the capability they provide and the power they consume. We have provided three sample hardware types, for purposed of power consumption that represent the various points in the spectrum of sensor hardware. These are Mica Motes ([2]), a very low end sensor node, WINS ([1]) a more powerful node and also draining more power, and WaveLAN card. These harware types can be defined as:

```
[1 thru 20]    HARDWARE-TYPE    WINS
[21 thru 100] HARDWARE-TYPE    MICA-MOTES
[101,110,111] HARDWARE-TYPE    WAVELAN
```

2.7 Putting it all together

We describe here the complete config file for our scenario.

```
VERSION      3.7
EXPERIMENT-NAME    sQualnet
SIMULATION-TIME    15M
SEED          1
COORDINATE-SYSTEM  CARTESIAN
TERRAIN-DIMENSIONS (1500, 1500)
SUBNET N8-0.1 {1 thru 100}
```

Chapter 3

Running non-IP protocols

Qualnet's communication stack assumes IP as default network protocol and hard-codes this assumption into the MAC layer. The simulator framework eases this assumption and allows non-ip protocols to run in Qualnet. Diffusion at the network layer is an example of a non-IP protocol running on a node's interface. Before describing how to configure these protocols and multiple network protocols for a node, let us first how to set up the diffusion network protocol.

3.1 Diffusion Network Protocol

Diffusion protocol simulated at the network layer follows TinyDiffusion implementation in TOSSIM closely. It simulates 1-sink, multiple source scenarios.

Configuration parameters for Diffusion are:

```
DIFFUSION-INTEREST-RESUBSCRIBE-PERIOD <time in seconds>  
DIFFUSION-INTEREST-EXPIRE-PERIOD <time in seconds> DIFFUSION-TTL <hops>
```

The application can be defined as:

```
DIFFUSION-SINK-APPLICATION <node> <start time> <stop time>  
DIFFUSION-SOURCE-APPLICATION <node> <start time> <stop time>
```

Note: While the diffusion implementation works with multiple sinks, duplicate data suppression via negative reinforcements has not been implemented. This will be done soon. The implementation works well for 1-sink, multiple source scenarios.

3.2 Defining Multiple Network Protocol Stacks

The simulator framework removes the Qualnet limitation of running only one network protocol across all interfaces of a node. Multiple network protocols can concurrently run on a node, each on a different interface.

If a node is part of multiple subnets, and thus has multiple interfaces, it can run different network protocols on each of its interfaces. Qualnet's current implementation allows only IP to be running on both the interfaces.

```
SUBNET N8-1.0 { 1 thru 50}
```

```
SUBNET N8-2.0 { 50, 51}
[1 thru 50] NETWORK-PROTOCOL  DIFFUSION
[50, 51] NETWORK-PROTOCOL    IP
[N8-1.0] USE-NETWORK-PROTOCOL DIFFUSION
[N8-2.0] USE-NETWORK-PROTOCOL IP
```

The above configuration parameters indicate the two subnets N8-1.0 and N8-2.0 exist with a set of nodes belonging to the subnets. The NETWORK-PROTOCOL indicates that the nodes in the list run the specified network protocol. Any nodes in the intersection of the node lists run both network protocols. The USE-NETWORK-PROTOCOL key is used to specify what network protocol is being used in that subnet/interface. Each interface needs to be on a different frequency to ensure that simultaneous transfers on all interfaces can occur.

In this scenario, nodes 50 and 51 run IP, while nodes 1 to 50 run diffusion. Multi-tiered networks can be simulated using this: Node 51 functions as a laptop tier node, a final destination for data. Node 50 acts a stargate tier node, with two NICs (a 802.11b wireless card and a mote radio). Node 1 through 49 act a mote tier node, with sensors on board to sense the environment.

A data query can be initiated by node 51 which causes node 50 to flood diffusion interest packets to the mote tier nodes. The mote tier nodes respond with data to node 51, which then forwards data back to node 50.

3.3 Multiple network protocol application

A simple application to illustrate the use of multiple network protocols is one of multi-tiered networks. A sample application set to illustrate the multi-tiered networks is:

CBR-DIFF-CLIENT : Sends a packet to CBR-DIFF-SERVER and waits for data.

CBR-DIFF-SERVER: Receives trigger (query) packet from client and floods interest packet.

DIFFUSIONAPP: Source receives interests and responds with data.

These applications can be modified to better simulate multi-tiered networks; for example, data aggregation can be added to the stargate tier node (CBR-DIFF-SERVER)

Chapter 4

Complete APIs

4.1 Sensor Channel and PHY

Important Data Structures. The data structure that stores information relevant to sensing capability is `SensorData` and defined in `addons/seq/sensor.h`. A list of this data structure is stored in Node data structure. Note that we can have multiple sensors per node (of same or different type), hence we have a list of structure. The `SensorData` stores the pointer to node for which it is initialized, the `sensorId` which is the id given to this sensor in config file, the sensor type and the value it is reading.

```
struct struct_node_str {
    ....
    #ifdef SENSOR_MODEL
    int numSensors;
    SensorData **sensorData;
    #endif
};

typedef struct {
    Node *nodeData; // Pointer to node
    int sensorId; // The sensor Id for which this struct is defined
    SensorPhyModel sensorPhyModel; // Sensor type
    float value; // Current value observed by sensor

    // Parameters defined in config file
    double prop_speed, loss_exp, tx_power;
    double threshold, radius;
    ...
} SensorData;

typedef enum {
    SENSOR_STATIC,
    SENSOR_DYNAMIC,
    SENSOR_DYNAMIC_SOURCE
} SensorPhyModel;
```

Important API. A node can call the following function to read the current value observed by the sensor.

```
SENSOR_GetCurrentValue(Node *node, float *value, int phyIndex);
```

We should note here the distinction between `phyIndex` and `sensorId`. The `sensorId` is the instance id of this sensor *as defined in the config file*. The `phyIndex` is the position in the Node structure `sensorData` list for this sensor.

4.2 Battery model

Important Data Structures.

```
typedef struct {
    int model;           // Simple or accurate model
    int batteryId;      // If we have multiple batteries per node
    double remaining;   // Remaining capacity
    BOOL dead;
    AccurateBatteryData *batData;
} Battery;

typedef struct {
    float usage[BATTERY_PROFILE_LEN];
    float *precomputed;
    float alpha;
    int index;
    float cummulative;
} AccurateBatteryData;
```

Important API. Wherever in the simulation model, the node executes event that consumes energy, it should call the following function to add load to the battery.

```
BATTERY_AddLoad(Node *node, double current, double duration);
BATTERY_GetRemainingCapacity(Node *node);
```

In the `AddLoad()` function, the current is in milli-Ampere while the duration is in seconds. Typically the places where this function is called are: transmitting or receiving packets, cpu processing etc. The `GetRemainingCapacity()` returns the remaining capacity. Note that the simulator already checks the battery status periodically, and shuts off the radio if the capacity goes to zero. The simulator checks every `BATTERY_CHECK_TIMER` interval (default at 1 minute). If the simulation is running slow or if periodic checking is not required, then this value can be changed.

4.3 Diffusion Network Protocol

Adding a non-ip protocol: Adding a non-ip protocol involves placing the protocol functions in the correct places. For example, a call to `NewProtocolInit` needs to be placed in `NETWORK_InitProtocol()` function.

The APIs available to the application layer are:

```
void NETWORK_Subscribe(Node *node, Message *msg, NodeAddress sourceAddress, NetworkRoutingProtocolType networkProtocol)
void NETWORK_Publish(Node *node, Message *msg, NodeAddress sourceAddress, NetworkRoutingProtocolType networkProtocol)
```

NETWORK_Subscribe takes in the node, message to be sent and the node's sourceaddress and forwards the message. The message to be sent contains the interests/attributes. The networkProtocol indicates which of the

NETWORK_Publish takes the same parameters, and forwards a data message to the next hop based on the one phase pull algorithm.

The application that functions as sink/source is diffusionapp.cpp

Chapter 5

NesC Code Simulation

The simulation of NesC code uses the architecture as shown in 5.1. The Mica2 code is inserted above the MAC layer and above the sensor layer.

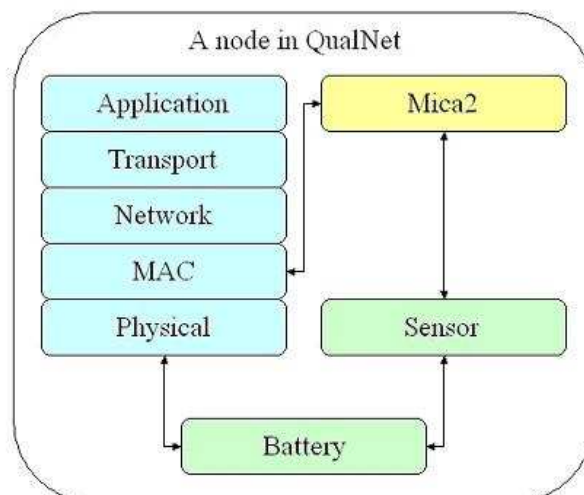


Figure 5.1: Architecture for Real Code Simulation

The Mica2 layer implements the hardware abstraction layer, with the interfaces implemented in NesC inside the qualnet directory in a tinyos distribution.

5.1 Installation requirement

The following tools are required and are available in the sQualnetTools.tar file.

Ncp.tar: Preprocessor requires the ncp directory and the ncp.sh file. Untar this file (`tar -xvf ncp.tar`).

NesCGen.sh: Script to run preprocessor + generate C code from NesC code.

Bagheera.jar: compiler to translate from NesC code to C code.

These above files should be in the same directory, but can be placed in different directories if the path variables are set properly.

The important path variables are:

- NCPATH: points to the ncp directory that was untarred from ncp.tar.
- QUALNET_HOME: required to run qualnet properly, should point to the Qualnet directory.
- TOSDIR: Needs to point to tinyos-1.x/tos.

Running NesC code in Qualnet requires the following:

5.2 Modifications to TinyOS distribution

TinyOS distribution contains a platform directory which contains platform specific files. To compile NesC to Qualnet, a qualnet directory needs to be added to the platform directory. P-Types UART uses named pipes to simulate read/write data. Ptypes has been included in the CVS, but needs to be built. This just requires a make inside \$QUALNET_HOME/ptypes-2.0.2

5.3 How to generate a new application

The first step to simulating your NesC code is to generate a cpp file for it and placing it in the correct directories. The NesC application should be placed in \$TOSDIR/../apps/ABC. Once you have placed it in the tinyos structure, you can now generate the cpp file by running NesCGen.sh. To Run the script, you also need ncp, a NesC preprocessor. The script file does the following:

1. Run a preprocessor to generate a ABC.ncpp file.
2. Compile the ABC.ncpp file to generate a ABC.cpp file. This ABC.cpp file is ready to be added to the Qualnet framework, as described below.
3. The ABC.cpp file then needs to be moved to \$QUALNET_HOME/mica2/. This file need not be added to the Qualnet Makefile.

5.4 How to add a new application

The .cpp contains Qualnet ready code for the NesC application. Now it needs to be added to the Qualnet framework. The following are the steps to add a new application. These steps are similar to adding a new application at the application layer in the Qualnet framework.

1. In \$QUALNET_HOME/include/mica2.h. Add the new application name. TINYOS_APPLICATION_ABC is what the example application name would be (to keep with the naming scheme). It can be anything.
2. In \$QUALNET_HOME/mica2/mica2.cpp. In MICA2.Initialize(), the new application needs to be initialized.

```
else if (strcmp (appStr, "ABC") == 0)
{
    printf ("%u: Running ABC.\n", node->nodeId);
    mica2Data->tinyosApplication = TINYOS\_APPLICATION\_ABC;
}
```

3. In `$QUALNET_HOME/mica2/tinyos.cpp`.

```
case TINYOS_APPLICATION_TOSBASE: {
    mica2Data->applicationData = new ABC();
    ((TOSBase*) (mica2Data->applicationData))->node = node;
    ((TOSBase*) (mica2Data->applicationData))->TOS_LOCAL_ADDRESS =
        mica2Data->interfaceAddress;
    ((TOSBase*) (node->mica2Data.applicationData))->TOS_AM_GROUP = 0x7d;
    break;
}
```

The new application needs to be added to `tinyos.cpp`; this procedure is much like adding a new application to application layer in the Qualnet framework.

The variables that need to be initialized include:

```
mica2Data->applicationData = new ABC();
```

This variable needs to be initialized to a new `ABC()`.

```
((ABC*) (mica2Data->applicationData))->node = node;
```

The new object needs to point to the current node.

```
((ABC*) (mica2Data->applicationData))->TOS_LOCAL_ADDRESS = mica2Data->interfaceAddress;
```

This initializes the local address of the node.

```
((ABC*)(node->mica2Data.applicationData))->TOS_AM_GROUP = 0x7d;
```

This initializes the group for the Node. This parameter should be modified only when the application uses the radio module. The rest of the variables should be initialized as shown above.

5.5 How to use configuration file

To Run NesC applications, the following need to be added to the main configuration file.

```
[node-list] NETWORK-PROTOCOL MICA2
[subnet] USE-NETWORK-PROTOCOL MICA2
```

```
TINYOS-APP-CONFIG-FILE ./mica2.app
MICA2-HW-CONFIG-FILE ./mica2.hw
```

This configures sQualnet to use the Mica2 stack, rather than the networking stack already present. `TINYOS-APP-CONFIG-FILE` is the application file. `MICA2-HW-CONFIG-FILE` specifies the hardware details.

To specify which node id runs which application, `TINYOS-APP-CONFIG-FILE` needs to be modified. The format is:

```
[NodeId] ApplicationName or [NodeId thru NodeId] ApplicationName
```

This application name needs to correspond to the Application Name in `mica2.cpp`.

If the UART is being simulated as well, the details are specified in the `MICA2-HW-CONFIG-FILE`.

[NodeIds] UARTx-INPUT FILE FILE-NAME.

To specify the input file for UART0 or UART1, the format of the command is above. The FILE-NAME is then read as input to the UART.

[NodeIds] UARTx-OUTPUT FILE FILE-NAME

This specifies the file to which the output of the UART is directed to. For example, [1 thru 3] UART0-OUTPUT FILE uart0.out specifies that the output from the UART 0 of nodes 1 through 3 should be directed to uart0.out

[NodeIds] ADC-INPUT adcPort sensorID

This is used to specify the port to which the sensor layer is attached to the ADC. Here sensorID is connected to adcPort. In the nodes, there must be a sensor with a sensor id equal to sensorID.

For example,

[1 thru 3] ADC-INPUT 0 0

specifies that sensor id 0 is connected to port 0 for nodes 1 through 3. In the main configuration file, sensors with sensor id of 0 must be defined for nodes 1 through 3.

Chapter 6

Future Work

Sensing channel. This release does not support diffusion based propagation channel models.

Processor Model. In this release, the processor power consumption model is not included.

Diffusion network protocol: Capability to specify which laptop tier node a stargate tier node responds to. b. Buffering of data at the stargate tier node/data aggregation.

Bibliography

- [1] <http://www.janet.ucla.edu/WINS/>. Wins.
- [2] <http://www.xbow.com/>. Mica notes.
- [3] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. *International Conference on Mobile Computing and Networking (MobiCOM)*, August 2000.
- [4] D. Linden and T. B. Reddy. *Handbook of Batteries*, volume Third edition.
- [5] D. Rakhmatov and S. B. K. Vrudhula. Energy management for battery-powered embedded systems. *ACM Transactions on Embedded Computing Systems*, 2002.
- [6] Scalable Network Technology. *QualNet*. <http://www.scalable-networks.com>.
- [7] Maneesh Varshney and Rajive Bagrodia. Detailed models for sensor network simulation and their impact on network. *Proceedings of ACM MSWIM 2004, Venice, Italy*, Oct 4-6 2004.
- [8] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. *INFOCOM*, June 2002.