FEW-PNets - A Framework for Emulation of Wireless Personal Networks

R. Venkatesha Prasad, Yonghua Li, Martin Jacobsson, Anthony Lo, Ignas Niemegeers
Delft University of Technology, Delft, The Netherlands.
{Vprasad, m.jacobsson, A.Lo, i.niemegeers}@ewi.tudelft.nl

Abstract

With the advent of miniaturization and higher computing capabilities of wireless devices, there is an exponential rise in the number of devices such as personal digital assistants, etc. Application development for these devices is the main focus for many device manufacturers and software developers. The applications developed need to be tested and tried before releasing it to the market. There are cross-compilers to produce the binary files of the software applications that can run on these devices. However, it is rather difficult for developers to test it on the actual devices every time a change is made in the program. Further, a wireless network infrastructure should also be in place for testing. To expedite application development FEW-PNets -- a Framework for Emulation of Wireless Personal Networks is proposed. It is a networked application to emulate a Personal Network (PN) owned by a person. It supports emulation of the most functions of PNs. We explain our model by emulating simple PN devices. In this article we elucidate the concepts, advantages, application domains and limitations of FEW-PNets.

1. Introduction

Information and Communication Technology, ICT, industry is paying more attention to the needs of end-users. Future technologies will provide more customized services to satisfy their needs of ubiquitous access to information and communication. As a response to this trend, PDAs and many other mobile nodes are getting a facelift with many useful applications. By integrating these devices and services in different physical environments around a person the Personal Area Networks (PANs) have become the front runners in encouraging this trend.

However, it has always been a great challenge to develop and test/debug applications and protocols for wireless ad-hoc network devices. It is both expensive to test and evaluate the system during the development phase with actual wireless devices. The main challenges in developing such applications are:

1. Multitude of physical environments such as 802.11, 802.15, etc., the devices need to talk different protocols and the developers need sophisticated infrastructure for all of them during bulk development. Further, there is a tight coupling of high layer code with the hardware.
2. Building and testing multiple layers of the applications on different devices and checking their compatibility is highly difficult. However, we note that it is necessary at some point of time in the development cycle but, it can be avoided until the software is almost ready and after checking its credible implementation on the emulator on PCs.
3. Usually, for application development on wireless devices, the physical and link layers are factory built (DLLs) however, the higher layers are the ones that would be frequently developed according to the requirements and goals of the specific applications or projects (see Section VIII).
4. The code reuse from other environments such as simulators, in most of the cases, is extremely difficult.
5. During the software development cycle, it is extremely difficult to test each build on the device before the actual full blown system tests; and thus the total development time increases.
6. The emulators are provided by the manufacturers along with the cross-compilers for the devices such as PDAs or Cell Phones. However presently, it can emulate only one device using the host PC's IP address. Thus developers are not able to catch the potential bugs which can cause havoc during interaction between two or more devices.

In this paper, we try to address the above issues by proposing a Framework for Emulation of Wireless Personal Networks (FEW-PNets). FEW-PNets is designed for any general emulation of wireless nodes for testing applications. Here we concentrate on a case study of the development of PAN applications. Since we have taken a specific case of an extension of PAN applications called Personal Networks (explained below), the framework is named as such. We also note here that this framework can be extended to any other wireless system with little modification. First, we briefly explain the target system used for implementing our framework.

Personal Networks (PN) [9] is a concept related to pervasive computing with a strong user-focus. PN extends a person's PAN that surrounds him with devices and services/devices farther away. This extension will physically be made via infrastructure-based networks, vehicle area networks, a home network or mobile ad hoc networks (MANET). A person's PN is configured to support his/her applications and takes into account the context, location and communication possibilities. A PN must adapt to changes in the surroundings, be self-configurable and support many different types of networks and devices.

Thus PNs must be capable and flexible enough to make use of current and future communication networks. For direct communication between a person's devices, PNs must be able to use ad hoc wireless communication technologies such as Bluetooth, ZigBee, Wireless Local Area Networks (WLAN) in ad hoc mode, low rate WPANs, IEEE 802.15.4 and other future technologies. To interconnect personal devices in

978-1-4244-2100-8/08/$25.00 ©2008 IEEE

different locations, PNs must be able to use infrastructure-based networks such as UMTS and GPRS networks, WLAN hotspots, wired and wireless broadband access networks and other evolved future versions of these access networks.

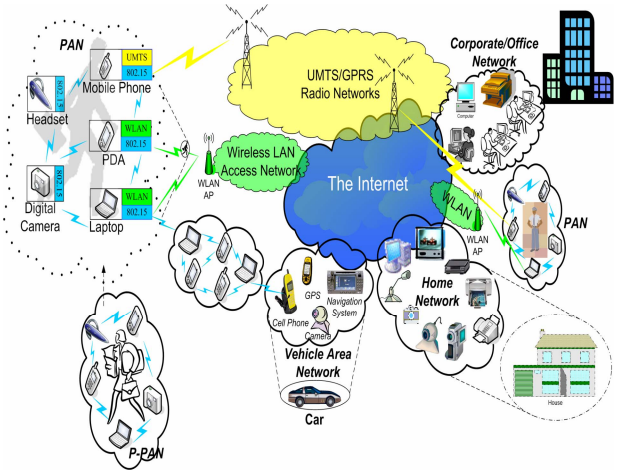

Figure. 1. An example of a Personal Network

It is a challenging to implement applications for PN nodes with wide variety of radio access technologies and devices with different capabilities. Usually the PN devices are hardware with their own processors and memory. Applications designed for these devices are usually programmed using cross compilers. To test the programs one has to download the binaries onto the devices. It is also necessary that the libraries to access lower layers should be in place. With every change in the program the total testing time of these applications increases linearly. We have attempted here to reduce this development time by segregating the higher level application software and libraries, and the low level device and physical layer dependent libraries.

We have developed a modular programming environment using Microsoft Foundation Classes (MFC) on Microsoft Visual C++ platform. The program tries to emulate all the functionalities above the physical and link layers of the PNs. It simulates a link layer data transmission of the real system by using the multicasting on local LAN. We give an account of implementation of a PN on FEW-PNets and present node discovery, imprinting, and chat as a case study. The key features of FEW-PNets are: (a) User-friendly and a professional GUI environment using MFCs; (b) Extensible with open-system architecture and modularization; (c) a distributed architecture to support remote and concurrent emulations; and (d) Enables easy code reuse for building the final product. This framework can be used for application development and testing of any system with multiple wireless devices.

The organization of this article is as follows. We provide the motivation for this work in the next section. We give a general account of simulators and emulators in Section 3. We explain the raison d'être behind FEW-PNets in Section 4 along with the explanation of major modules. In Section 5 we present the major classes in our framework with class diagrams. We take a simple case study and show the screen captures of various components in Section 6. We discuss our framework vis-à-vis ns-2 in Section 7. While in Section 8 we show the applicability of our work by listing many projects that require a framework, such as FEW-PNets and we conclude in Section 9.

2. Motivation

The simple software design life cycle (SDLC) practices like waterfall model, Spiral Model, etc. [1], were religiously followed for many years. However, the software for new devices and new applications that are being developed require new approaches. These practices do not help much while building systems for knowledge workers, people at help desks, experts trying to solve problems, etc. [2]. Therefore, the software development may use combinations of many of these models. Synchronize and stabilize method [3] is one such way where many rounds of software development is done and tested.

The aim here is not to replace any of these practices and well tested procedures. We only try to enable, developers/researchers to efficiently build and test the software that goes into the target devices without an elaborate setup with all the hardware devices and infrastructure. Testing and debugging in fact takes 50% of the development time [21]. Thus emulator which can also produce reusable code is highly valuable. Succinctly our goal here is to:

1. enable the developer to feel/test how the application works with the final software and avoiding a cycle of algorithm development/testing with the simulators.
2. allow developers to reuse the code that they develop in the emulator i.e., to port almost the same code on real systems.
3. effectively use the LAN environment to emulate distributed application testing.
4. enable easy substitution of different modules e.g., routing, device discovery, service discovery, etc., for testing different approaches/algorithms.
5. enable concurrent instantiation of devices for testing on the same computer for complete testing.
6. to enable real time testing of the applications without bothering about the physical/radio characteristics while developing algorithms.
7. build a final look and feel (if the compiler environment supports it, e.g., MFC) for user trials and demonstrations.
8. allow software development to emulate heterogeneous devices to test the programs.

Before we discuss our proposal we specifically mention the differences between Simulation and Emulation – in general with respect to wireless communication applications.

3. SIMULATORS AND EMULATORS

3.1 Network Simulator

A network simulator is a piece of software that imitates a network without an actual network being present [4], and simulation is an imitation of some real devices or state of affairs. Simulation attempts to represent certain features of the behavior of the physical system abstracted by a model with some degree of details. A number of network simulators are available [5, 6], which have in common that they aim at simulation of functionalities of different layers of a network.

3.2 Network Emulator

Computer software that allows certain programs to run on a platform other than the one they are originally written for? It does this by "emulating", or reproducing, the behavior of one type of platform on another by accepting the same data, executing the same programs, and achieving the same results [7]. Unlike simulation, which only attempts to reproduce a system's behavior, emulation attempts to model to various degrees the state of the device being copied [7]. In fact, emulation does simulate some part of the system and allows the actual implementation of the remaining part to interact with the simulated part.

4. RATIONALE OF FEW-PNETS

4.1. The Foundation

The goal of FEW-PNets is to design a PN emulation system that can be used by developers with ease. In addition, it is important to avoid using actual devices for development and testing of the applications. Therefore, our framework needs to emulate all the functionalities above the link layer. The questions we consider here are: (a) How to abstract the physical (PHY) and Link layers (LL)? (b) What interfaces and functions should the lower layers provide? (c) How could the distributed architecture of FEW-PNets be designed? (d) How to modularize (objects?) the code? (e) Is it possible to produce GUI for the emulated application?

We quickly answer these questions with respect to the architecture of FEW-PNets. There is a similarity between *radio domains* (an area within where signal from this node can be heard) with multicasting. Basically a wireless node transmits a message by 'broadcasting'[1]. All the nodes that are in the vicinity can listen to that particular RF signal – however, the possibility of understanding such packets is a different aspect. Similarly, a multicast packet is also 'broadcasted' on a network, and only those who are members of the multicast group are able to use those packets. It is similar to the wireless nodes tuned that particular RF.

This is the basic principle of FEW-PNets. We simulate the PHY and LL using multicasting. Each node is tied to its own multicast group address. If we enable a node, within the reach of the RF signal of the other node to listen to multicast address of its counterpart, we have established the communication channel between those two nodes. In a nutshell:

- Multicast features a similar behavior as wireless radio communication.
- It minimizes the adaptation when code is transplanted to real devices by simply replacing multicast send() by LL send() functions on the real hardware.
- Because of the existence of a handy interface in most programming languages to send and receive multicast, it is convenient and universal for the development of software.
- The distributed testing/debugging is a step closer to reality which is straight forward since multicasting is used. Each device can be emulated on different computers on a LAN environment or even multiple instantiation of the same on the same system using different multicast address (example, PC with Multicast-3 & 5 in Fig.2).

The framework consists of a (a) *Server* which helps only in housekeeping. (b) *Clients* which are actually the *emulators* have two major blocks – Base program and Node program (emulating higher layers). A client has only one base program abstracting the lower layers i.e., PHY and LL. There can be many node programs each representing a wireless device which in turn may support many applications (Fig. 2 & 3).

The main title (on the first page) should begin 1-3/8 inches (3.49 cm) from the top edge of the page, centered, and in Times 14-point, boldface type. Capitalize the first letter of nouns, pronouns, verbs, adjectives, and adverbs; do not capitalize articles, coordinate conjunctions, or prepositions (unless the title begins with such a word). Leave two 12-point blank lines after the title.

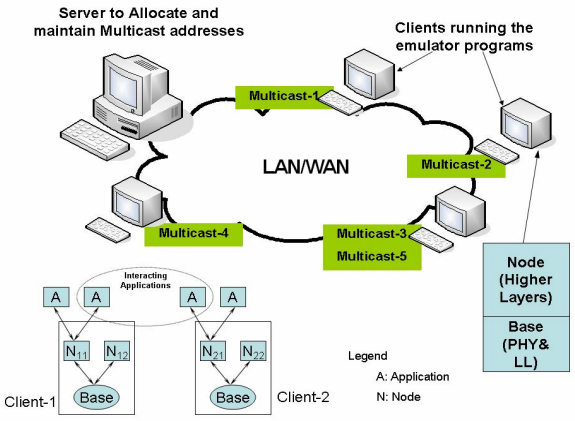

Figure 2.Emulator Schematic

4.2. Emulation of the Lower Layers using Base Program

The base program on each computer provides functions and interfaces of the PHY and LL layers while the node program is responsible for the upper layers. There are two reasons for not emulating all layers on the node program. First, the emulator imitates only the higher layers and usually, the PHY & LL code would be some sort of device drivers on which a user has least control. Thus, we keep the layers that are almost fixed separate from what an application developer can change and /or implement.

Second, when creating several nodes on a single system for emulation, this is compulsory because of the inherent multicast address collision. When a node sends packets, all nodes in the vicinity receive the packets as required by the real scenario. A single multicast address is defined with an IP address and a unique port to which a socket is bound. If a node program maintains its own socket and multicast address corresponding to its own radio domain, two nodes running on the same computer have to use different multicast addresses, which makes them unreachable by each other even if they are reachable. Now, since the base program knows that two nodes are within their reach and talking to each other, it duplicates the multicast packets and sends it to each emulated node.

[1] We do not use the term broadcast with the usual notion of broadcasting at the Network layer of OSI architecture. We use it in the sense that, at each PHY/LL one can listen to any RF signal - may be without making any meaning of it.

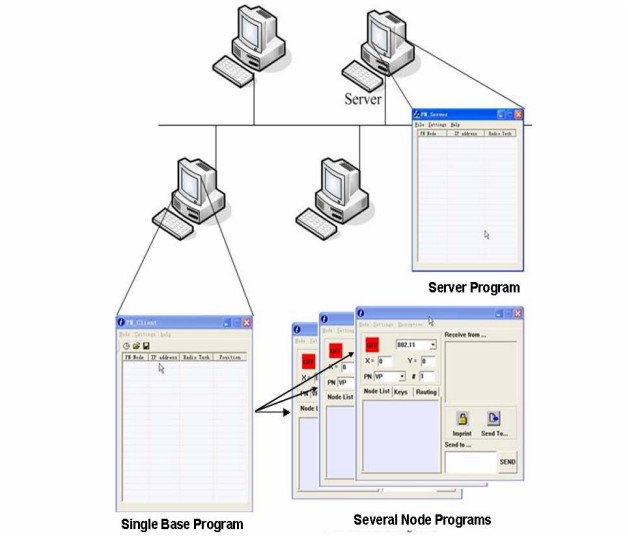

Figure 3. The System setup

4.3. Emulation of Higher Layers using Node Program

In a typical scenario, several computers running FEW-PNets base programs are connected through the LAN with multicast enabled. On each PC, a node program is created from the base program. These distributed node programs then mimic the desired wireless network topology with the radio range of each node imitated by mapping to a multicast group address. When a node is turned-on after configuration, an update packet is multicasted to all the base programs. Each base program then calculates the distances between the node that sent the message and the other nodes registered with it. After comparing the results with the fixed radius of the radio technology used by a node, the decision is made whether to transfer this message to the node or not. Thus, this mechanism takes care of distributing multiple wireless nodes on different computers or instantiating many of them on the same system. With multicasting on the network radio links are simulated.

The communication is initiated by a node application calling the send() method of the socket to multicast a "Hello" packet. The same technique as explained earlier for an update packet is applied again to select the nodes that receive this multicast Hello packet. Using Hello packets a list called active neighbors is updated. With this active neighbor list, functions of higher layers such as routing, encryption, imprinting and applications such as chat are possible.

4.4. The Server Program

A server program is necessary in FEW-PNets due to several reasons. First, if a node on another computer is named with the same name as that of an existing one, collisions are not avoidable. Second, because of the distributed architecture, a node's details such as file containing keys stored on a client computer are not accessible from another when the node is executed next time. Without a centralized repository, every time these files need to be entered manually. However, in reality this is not a problem since all local information or settings are stored in the memory on the device. So the server acts as the memory of all nodes in some sense. Third, a multicast address should not be used by two nodes. To maintain this repository the Server program is used.

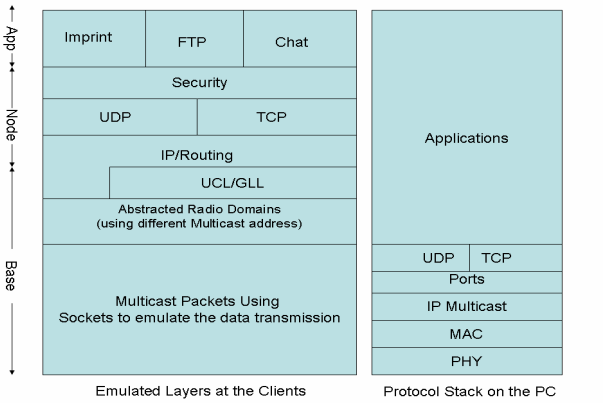

Figure 4. Layers in the Emulator vis-a-vis the Protocol Stack in the PC

4.5. Layers of the Emulator

FEW-PNets has been completely implemented in the application layer where the entire user written executables are stored. Fig.4 shows the various layers of the emulator (left side) with respect to the usual Internet stack (right side). The programs used for various tasks in the emulator can be moved on to the devices at a later stage easily without many changes. This is the important gain of using the emulator. This flexibility is crucial when designing new protocols as it allows the designer to rapidly try out new designs almost in the actual scenario.

The base program needs to know the location of every other node, as they use such information to determine the dynamic connectivity in real time. Based on the packets received by the lower layers, this sub-layer calculates distances between all pairs of nodes and decides all members of a certain radio domain in the range of the individual nodes. Universal Convergence Layer (UCL) or Generic Link Layer (GLL) [13] is used to abstract multiple radio links; however they are not used in the current version. The base program actually corresponds to the radio domain and UCL/GLL layers as shown in Fig.4. The node program (emulator) consists of all the layers above this UCL/GLL.

In the IP and routing sub-layer, the packet header is analyzed to get the destination addresses and checks whether it is an intermediate routing packet. If it is an intermediate routing packet, the node program adds its address to the packet and forwards it without passing it to higher layers. Otherwise, the packet is further processed in the layer above it. Since, currently implemented applications use UDP, the TCP/UDP layer just passes the packet to the security layer. The data packet is examined by applying a set of rules in the security sub-layer before it is passed on for further processing. In the application layer, the data is processed appropriately which is same as implementing any stack.

5. CLASS DIAGRAMS

Modeling is the central part of all the activities that leads up to the development of proper software. Modeling helps in communicating the desired structure and behavior of our

system. In addition, it enables visualization of the system architecture. The important aspect with respect to this work is that it enables us to explain what has been done and how it can be replicated as well as improved.

The Unified Modeling Language (UML) [10] is the standard for writing software blueprints. UML may be used to visualize, specify, construct and document the artifacts of a software-intensive system. UML includes many kinds of diagrams, where class diagrams are the most commonly used in modeling object-oriented systems.

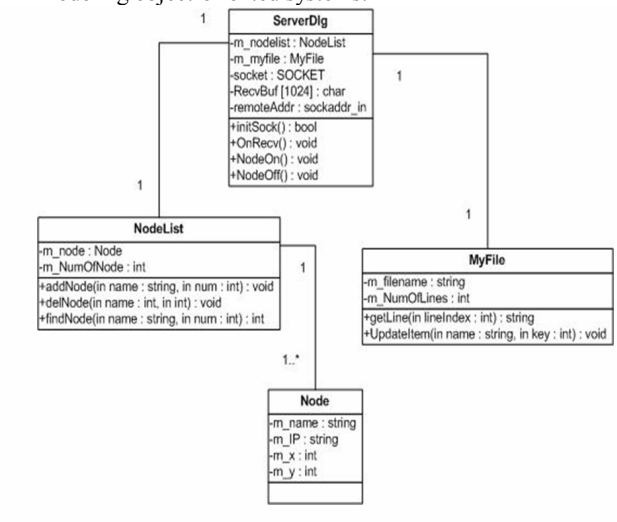

Figure 5. Class Diagram for Server

We use class diagrams here to explain our framework implementation. We implemented FEW-PNets on MS-Windows using MS Visual C++ and MFC [8]. Since functions using Windows API are not compatible with other OSs and real devices. To make the code reusable, we have segregated all important classes and functions in such a way that all Windows APIs are grouped together in a class. The emulator code is not mixed with the above class. Thus we can later translate the emulator code onto devices without any change also onto other OSs. The classes that end with "Dlg" are specific to Windows-API functions and variables which are the base class for GUI. For programming the devices with MS Windows Mobile OS, similar constructs can be used with slight variations. The idea is to abstract all the common functions in objects that remain the same on PCs as well as the handheld devices.

5.1. Server Program

Fig.5 shows the class diagram of the server program with only the important members and functions. It has the following objects:

MyFile: The ServerDlg contains an object of MyFile, which is used for file operations.

NodeList: This class is constructed to manage the list of active nodes. addNode(), delNode() and findNode() are used to add, delete and search a node in the list of active neighbors respectively.

Node: This class is used for keeping all information of a node. m_name is the node's name, m_IP represents its IP address. m_x and m_y are its position defined in x and y-axes.

5.2. Class Diagram of a Node

Fig. 6 shows the class diagram of an emulated node in FEW-PNets. In general, PN_ClientDlg is the class that provides functions of the base program, while NodeDlg and Node together implement the node emulator program. The functions in NodeDlg may use Win-APIs, which are to be appropriately translated when they run on real devices. However, the code that implements the functions of Node can be directly reused. The following are the important classes that are used here:

PN_ClientDlg: This is the class that provides GUI and functions of the base program. It contains a list of objects of the RD class with the name m_RDList. An important function of this class is CreateNode() which is used to create an application dialogue for the node program. CalculateRD() is used to compare distances between two nodes and determine to which radio domains each node belongs to. The radii for Bluetooth and 802.11 are set to 10m and 100m respectively. OnRecv() informs the base program that a packet has arrived and makes proper decisions depending on different packet types. Send() is the fundamental function to send all packets. It sits in the lower layer and uses Windows API. This function needs to be replaced with appropriate interfaces provided by the DLLs or System calls while porting onto real devices.

RD: It is a data structure including the node's name, the multicast address for the radio domain and the socket used for communication by the node. Their relationships are shown in Fig.7.

NodeDlg and ***Node*****:** These two classes together emulate a node program and they are segregated into two with all the Windows-APIs into the "Dlg" class. There are various functions in *NodeDlg*. *PowerOn()* and *PowerOff()* defines the steps the program does if a user clicks the power button of the node.

OnTime() sets a few timers for FEW-PNets, for example, a timer of 1sec for sending "Hello" packets using the function *SendHelloPkt()*. Similarly many other functions are used here. In FEW-PNets, *Node* contains 2 lists: one for neighbor nodes called *m_NbNodeList*, and another for imprinted nodes and keys (if security is part of the implementation). This list is called *m_KeyList* consisting of *NODE_KEY* structure variables. Functions in *Node* class are used to search, add or delete any data, which can be easily recognized by their names.

Encryption**:** At present, there are only two major functions in this class, *Encrypt1()* and *Decrypt1()*. The digit "1" represents Algorithm 1. Later more encryption and decryption algorithms could be implemented and tested without any other changes.

ImprintDlg**:** Imprinting is the process by which two nodes in a PN permanently know each other by generating a shared key. This key is used to encrypt the data exchanged between them. This class is designed for node imprinting using Diffie-Hellman key exchange protocol [11]. *m_base* is a base number randomly generated from {2,5} and *m_prime* is a prime number from {7,11,13}. *m_SelNum* is the number input

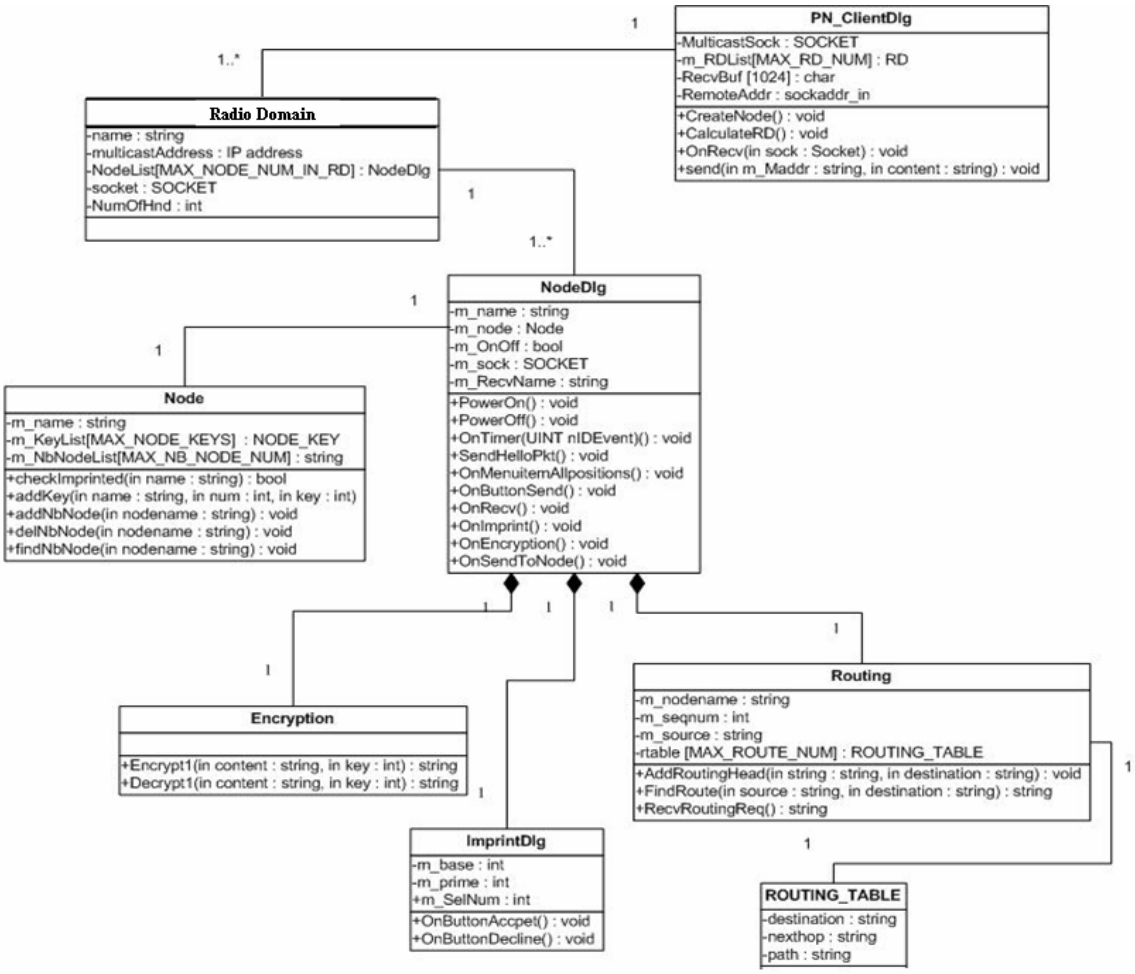

Figure 6. Class Diagram of a Node

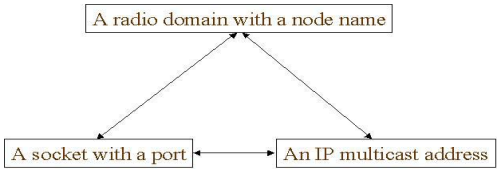

Figure 7. Mapping between parameters

by the user. *OnButtonAccept()* and *OnButtonDecline()* are functions called when the user accepts or declines the imprinting request.

Routing and ***ROUTING_TABLE*****:** *Routing* is the class used to find a route to any other node. *m_nodename* is the name of node to which the object is linked. *m_seqnum* and *m_source* is useful to record the sequence number and source node of any routing packet depending on which, the node decides to react to a routing request or not. *rtable* is the data structure to keep discovered routes. It contains 3 members: *destination*, *nexthop* and the whole path - *path*, which is used to compare alternative paths to find the shortest one.

AddRoutingHead() is called when a packet is passed down to the IP layer, which then adds address and routing information as the packet header. *FindRoute()* initializes a routing request, while the *RecvRoutingReq()* defines operations when a routing request packet arrives. These are the generic functions and any specific routing algorithm can be implemented with a little modification in the framework.

6. IMPLEMENTATION - A CASE STUDY

We emulated a PN with devices having two types of radio technology, Bluetooth and 802.11 (we show in diagrams only 802.11 devices). We present now some of the screen captures of the emulator software. We also implemented key exchanges (as a separate class). Fig. 8 shows front end of the server program, which allocates the IP number to each emulated node. The PN owner's name is 'VP' and we create multiple nodes for 'VP' now. In the base there are limited menu items through which we can create nodes, and set some of the parameters such as Radio, range, position, etc., for all the nodes when we create it. However, the individual nodes can also change their positions and characteristics. Fig. 8 also shows the list of the IP numbers allocated to each node at the node program (we have allowed arbitrary numbers). A multicast address tied to each node in VP, however it is not shown here.

We show in Fig. 9 a node emulated with some limited capabilities. We have implemented at present imprinting, DSR routing, sending/receiving text chat with encryption. One can observe that VP-3 cannot interact with VP-6 but can interact with VP-2. We can also change the position of the node VP-3 by changing the X-Y co-ordinates.

At present we have not automated the mobility where X-Y co-ordinates can change based on a mobility model. Fig. 10 show the panel where all the neighbors of the node VP-1 can be seen in one place.

Further, VP-1 node can be dragged to a new position. Similarly, we can change the positions of all the nodes. This is a limited way of emulating the mobility which can be enhanced later to automate it. When the position of node changes it should send information about its new position to the server, which in turn sends it to all the nodes. Otherwise send it on a separate multicast address to which all the nodes are listening to. This helps in proper selection of multicast group membership and to find the neighbors.

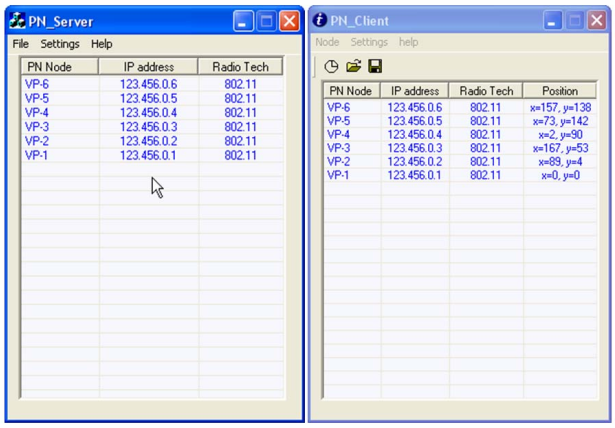

Figure 8. Server and Client Base Program Frontends

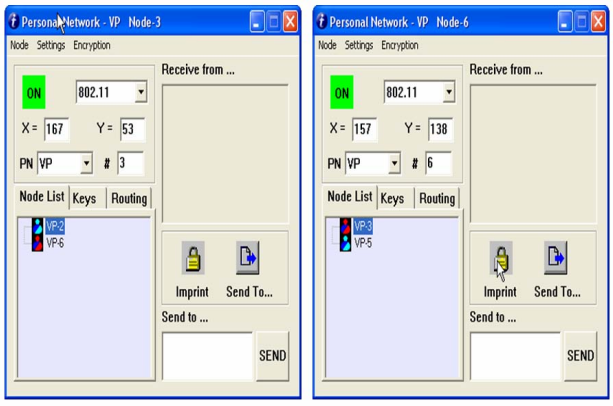

Figure 9. Front-end of Two Nodes of PN-VP, trying to connect to each other

7. DISCUSSION

FEW-PNets enables realtime user interactions while developing the applications. It is designed in such a way that we can reuse most of the code. Since, it uses multicasting distributed application testing/debugging/implementation can be effectively carried out in a LAN. Since the speed of the LANs is of the order of Gigabits, the speed is never the bottleneck. Given that FEW-PNets uses layered and Object Oriented architecture, modularizing each task is inherent. Similarly, it can also be invoked concurrently. It gives an abstract view of the PHY and LL which in most cases are embedded into the devices that the application developers do not alter.

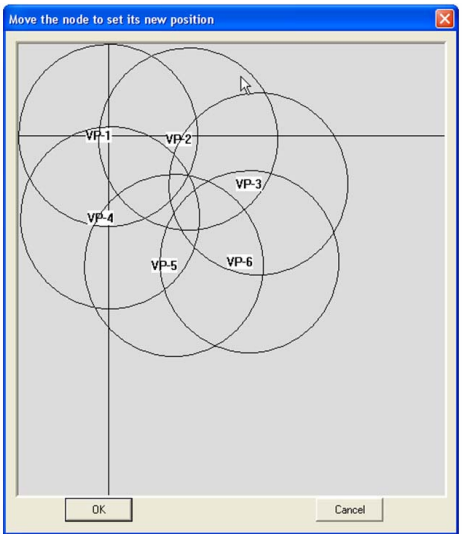

Figure 10. The Spread of Nodes; Position Control Panel

By changing the node characteristics with different RDs heterogeneous devices can be emulated. Thus FEW-PNets accomplishes all the goals that are listed in Section II. Time and again, such works triggers the debate 'which is effective – simulator or emulators?' There is no clear winner in such debates. However, depending on the objective one might suggest a particular method. If it is to understand and model a real-world system or an academic endeavor simulators are preferred. If the implementation takes the front seat then emulators come in handy. Any such study is incomplete without discussing about the network simulator ns-2[5].

ns-2, the *de facto* simulator, is a discrete event simulator targeted to networking research. It offers full network simulation on all layers and can be used to run customized experiments that can lead to important insights into the behavior of protocols and algorithms. ns-2 provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks. However, many of our goals set in Section-II cannot be achieved by simulators.

Apart from the usual differences between simulators and emulators, we can notice that the message handling inside the code can be different. In FEW-PNets, since packets are always sent on multicast address, only the members of that multicast group, usually, only the sender's neighboring nodes receive the packets. In contrast, ns-2 sends packets to all nodes, which

later determines to process or drop the packets based on the calculated power of the received signal. Thus, if there are *N* nodes in total in a given scenario, when a node with *M* neighboring nodes sends a packet, FEW-PNets delivers *M* packets while ns-2 delivers *N*. In a large network with $N >> M$, FEW-PNets could provide efficient emulation. However, FEW-PNets requires additional operations when topology changes. When a node moves, the new position of that node is multicast to all the base programs of FEW-PNets. The distances between this node and the other nodes are then recalculated. If the distance is less than the radius defined by a radio technology (100m for 802.11 and 10m for Bluetooth), both nodes are added to each other's neighboring nodes' list. Thus in a single machine, the distance is calculated *N* times, and several nodes might be registered and deregistered for multicast groups. While in ns-2, only the position of a node is changed, other related calculation would be executed when packet delivery occurs, which potentially reduces unnecessary computation. Before we close, it is mandatory to mention that ns-2 is also capable of carrying out emulation by coupling it to the live network for emulation [12]. Special objects within ns-2 are capable of introducing live traffic into the simulator and injecting traffic from the simulator into the live network.

Considering the differences in the way ns-2 and FEW-PNets are implemented it is a challenge to design an interface between them. But if later version of FEW-PNets could provide an interface of the live network, it could integrate ns-2 into a new emulation system, which is able to adopt advantages of both FEW-PNets and ns-2. As a result, the new system can use all the advanced models of ns-2 and enhance its usefulness.

8. RELATED PROJECTS

Multicast has been used in distributed simulation settings [22][23] to aid in simulations. However, to the best of our knowledge it is not been used in device emulation as has been proposed here.

There are many projects that aim at providing solutions for implementation of the future applications based on PANs. We briefly mention them here to position the usefulness of our proposed framework. This work was executed as part of a bigger project that is similar to many of the following ventures. Our work can directly help in testing and realizing these projects. The intention of mentioning these projects here is to show that our framework can expedite the development of these projects. However, the framework presented here is completely general thus it can also be applied to any networking scenarios.

8.1 Personal Distributed Environment

Mobile VCE [15] project defines the concept called Personal Distributed Environment (PDE). PDE has a very similar vision to PN, with a focus on delivering services to the user. Local and remote devices and services form a user's PDE sentence incomplete. A PDE consists of isolated devices and a collection of sub networks, including PANs, BANs and home networks. Subnetworks can be connected to each other through a core network using various access technologies.

8.2. MOPED

Current technology and communication support provide connectivity between devices, but do not enable cooperation between devices. The goal of the MOPED Project [16] is to bridge this gap from communication to cooperation. The project presents a networking model that treats a user's set of personal devices as a MOPED -- an autonomous set of MObile grouPEd Devices, which appears as a single entity to the rest of the Internet. All traffic for a MOPED user is delivered to the MOPED, where the final disposition of traffic is determined. To the outside world, this MOPED appears as a single device with a single interface or identifier.

8.3. Ambient Networks

The Ambient Networks (AN) [17] is another integrated project that intends to develop a new vision of network, which is an open and scalable homogenous overlay of heterogeneous devices networks that facilitates ambient networking. ANs allow efficient use of resources and enables increased competition and dynamic cooperation. It tries to connect different kinds of networks such as inter-vehicle networks, BANs, and sensor networks using IP as the basic network protocol. AN aspires three important features such as Composability, Mobility, and Heterogeneity.

8.4. MyNet / User Information Architecture / Unmanaged Internet Protocol

The User Information Architecture (UIA) [18] and the Unmanaged Internet Protocol (UIP) [19] are being investigated at MIT, while MyNet [20] is a collaboration project between Nokia research and MIT based on these projects. UIP combines the self-management of ad-hoc networks with the scalability of IP by creating some sort of self-organized overlay network for personal devices. UIA, on the other hand, is intended to allow global interaction and sharing among the devices between persons. The UIA protocols are the foundation upon which the rest of the MyNet project work is layered. The UIA is based on two principles: (a) security is decoupled from physical connectivity; and (b) establishment of trust is based on social connectivity.

8.5. IST Magnet Beyond

IST MAGNET Beyond [14] has the vision that future users will be supported in their professional and private activities by their PNs consisting of PANs extended with clusters of remote devices, which could be private, shared, or public. The users will have continuous access to personal resources and connect through personal devices wherever they are located. It is developing PN that can satisfy the user requirements and integrate PNs with existing technology. Issues both at the PN and PAN level, in particular service and context discovery, self-organization, mobility management, addressing, routing, and co-operation with outside world are being attempted.

9. LIMITATIONS AND CONCLUSIONS

9.1. Limitations

We have outlined an emulation framework with which one can develop and test higher layer software/firmware that sits inside the wireless devices. This study at this stage is still in its

infancy. It requires major restructuring of the classes to cover a broader range of applications/scenarios to make it more generic. Implementation as of now is not clearly user friendly to emulate any network. We have not yet incorporated any models for mobility, and packet loss models to evaluate a system. The movement at this point of time is not automated – a user needs to drag the nodes on the GUI from one position to the other. Received signal strength variations based on the distance or due to environment is not considered yet. The emulator software has a strong allegiance Microsoft VC++ and MFCs thus a trifle dependent on the OS. However, one can use the software in the application space could use it.

9.2. Conclusions

We have presented an ongoing work FEW-PNets – a framework for emulating the PANs and in general any wireless device or networks. We associated the emulation software with multicasting technique such that we can effectively simulate the radio domains without any need for hardware devices. With the use of multicasting, we can distribute the nodes on multiple computers and a team can be involved in testing without any additional infrastructure. We can also emulate these nodes on a single system by concurrently invoking several instances of the emulator. We built a low-cost and user-friendly emulator using VC++ and MFCs, which supports flexible, dynamic and distributed test environment. We gave a sneak peek into our software development with class diagrams to enable the readers how we have implemented the system. We have also shown that the emulator framework can be decoupled from the OS on which it is implemented.

A case study of a PAN with screen captures has been shown to give a full picture of the framework. Though we have only shown some vistas for a possible future at this juncture, we believe this framework has a high potential for large scale use if designed carefully. We have also shown many avenues where this framework can be used immediately. A few limitations in the current implementation, like lack of a mobility model have to be addressed in the near future. This is the first step in designing a software framework that helps in faster development cycle. A user friendly GUI, a clear bifurcation of the software at higher and lower layers are immediate responsibility. An interesting step would be to connect ns-2 with FEW-PNets. We opine that many miles are to be covered before the release of a stable version of the software for a generic use.

ACKNOWLEDGMENT

The authors were supported by the Freeband PNP2008 research project during this work, as well as IST MAGNET Beyond research project. The authors would like to thank all the partners from these projects for their fruitful discussions on the topic of Personal Networks, its implementation and helping us to crystallize our ideas.

REFERENCES

[1] Roger S Pressman, Software Engineering: A Practitioner's Approach, 6th edition, Tata Mc Graw Hill, 2005.

[2] Russell Key, “System Development Life Cycle”, http://www.computerworld.com/developmenttopics/development/story/0,10801,71151,00.html

[3] Cusumano, M. A., and D. B. Yoffie., Competing on Internet Time: Lessons from Netscape and Its Battle with Microsoft, New York: Free Press, 1998.

[4] http://en.wikipedia.org/wiki/Simulator

[5] ns-2-Network Simulator, website: http://www.isi.edu/nsnam/ns

[6] GloMoSim simulator, website: http://pcl.cs.ucla.edu/projects/glomosim

[7] http://en.wikipedia.org/wiki/Emulator

[8] Website: msdn.microsoft.com/visualc/

[9] Ignas G. M. M. Niemegeers, Sonia M. Heemstra de Groot, "Research Issues in Ad-Hoc Distributed Personal Networking", Wireless Personal Communications: An International Journal, Volume 26, Issue 2-3, Pages 149-167, Kluwer Academic Publishers, August, 2003.

[10] Grady Booch, James Rumbaugh, Ivar Jacobson, “The Unified Modeling Language User Guide”, Addison-Wesley, Second edition, 2005.

[11] E. Rescorla, “Diffie-Hellman Key Agreement Method”, RFC2631, June 1999.

[12] Network Emulation with the NS Simulator, http://www.isi.edu/nsnam/ns/ns-emulation.html

[13] Sachs, J., “A generic link layer for future generation wireless networking”, ICC, May 2003, Vol. 2, pp. 834- 838

[14] IST MAGNET Project, http://www.ist-magnet.org/

[15] Mobile Virtual Centre of Excellence, http://www.mobilevce.com/

[16] Robin Kravets, Casey Carter, Luiz Magalhaes, “A Cooperative Approach to User Mobility”, ACM Computer Communications Review, Volume 31, Pages 57-69, 2001.

[17] IST Ambient Networks, http://www.ambient-networks.org/

[18] Frans Kaashoek, Robert Morris, “User-Relative Names for Globally Connected Personal Devices”, In the 5th International Workshop on Peer-to-Peer Systems IPTPS'06, Santa Barbara, CA, USA, February 27-28, 2006.

[19] Bryan Ford, “Unmanaged Internet Protocol: Taming the Edge Network Management Crisis”, In the 2nd Workshop on Hot Topics in Networks (HotNets-II), Cambridge, MA, USA, November 20-21, 2003.

[20] MyNet/UIA, http://research.nokia.com/research/projects/mynet-uia/index.html.

[21] Jiantao Pan, Software Testing, CMU, USA, 1999,http://www.ece.cmu.edu/~koopman/des_s99/sw_testing

[22] Smith, W. G., Koifman, A. “A Distributed Interactive Simulation Intranet Using RAMP, a Reliable Adaptive Multicast Protocol”, In Proc. of the 14th Workshop on Standards for the Interoperability of Distributed Simulations, Orlando, FL (Mar. 1996).

[23] Dennis M. Moen and J. Mark Pullen, “Enabling Real-Time Distributed Virtual Simulation over the Internet Using Host-based Overlay Multicast” Workshop on Distributed Simulation and Real Time Applications, 2003.