Open access • Journal Article • DOI:10.1007/BF01985756

Mobile wireless network system simulation — Source link

Joel E. Short, Rajive Bagrodia, Leonard Kleinrock

Institutions: University of California, Los Angeles

Published on: 01 Dec 1995 - Wireless Networks (Springer-Verlag New York, Inc.)

Topics: Network simulation, Wireless network, Network traffic simulation, Heterogeneous network and Simulation language

This paper to appear in *Wireless Networks Journal*, Vol. 1, No. 4, 1995.

Mobile Wireless Network System Simulation[1, 2]

Joel Short, Rajive Bagrodia, and Leonard Kleinrock

Computer Science Department, University of California, Los Angeles,
Los Angeles, CA 90095-1596, USA

Received August 1995

Abstract. *This paper describes an advanced simulation environment which is used to examine, validate, and predict the performance of mobile wireless network systems. This simulation environment overcomes many of the limitations found with analytical models, experimentation, and other commercial network simulators available on the market today. We identify a set of components which make up mobile wireless systems and describe a set of flexible modules which can be used to model the various components and their integration. These models are developed using the Maisie simulation language. By modeling the various components and their integration, this simulation environment is able to accurately predict the performance bottlenecks of a multimedia wireless network system being developed at UCLA, determine the trade-off point between the various bottlenecks, and provide performance measurements and validation of algorithms which are not possible through experimentation and too complex for analysis.*

1. Introduction

When developing mobile wireless network systems (i.e., wireless networking algorithms, node architectures, and network infrastructures), the designer is presented with numerous design alternatives. There are many factors which impact the analysis, performance and validation of these design alternatives. These factors range from having to support different patterns of node mobility to integrating the traffic generators, networking algorithms, and operating system capabilities.

A few operating system kernels and languages have been designed to support wireless and mobile communication [1], and a number of protocols have been devised to solve the numerous topology setup and maintenance, media access control, and transmission problems in the mobile environment [10]. Commercial radios designed to be hooked up with laptops for wireless multimedia transmissions are available in the market. Although solutions to different facets of the wireless mobile information system design are appearing, relatively little effort has been devoted to understanding the performance impact of the interactions among different components of the system.

Analysis, simulation and measurement have all been used to evaluate the performance of network protocols and multimedia systems. Measurement-based approaches are useful only after the system has been deployed. Although they offer the most accurate evaluations of performance problems, they are often inadequate because it may be infeasible to modify the deployed system to experiment with many design parameters. Even when such modifications are feasible, the cost of the necessary software and hardware modifications may be exorbitant. Analytical models offer the opportunity to quickly examine a large parameter space to identify efficient configurations; however for complex systems with many interacting components, analytical models may either be inaccurate or computationally intractable. For complex, heterogenous systems, simulations are often the only realistic alternative to performance prediction.

The primary drawback with detailed simulation models is that they are frequently slow. Experience with many existing network simulators has shown that a performance study of wireless protocols for even small networks (tens of nodes) can take many days; running such simulations for networks involving a large number of mobile elements is clearly infeasible. Recent experience with parallel execution of models for personal communication systems has shown that parallelism offers significant potential to improve the execution time for these models; it is likely that these techniques can also be exploited to improve the execution time for simulation models of wireless networks. This paper describes a simulation environment for wireless networks that is built using the Maisie [3] simulation language. Maisie has been implemented on both sequential and parallel architectures. The paper describes the environment and presents experimental results using sequential execution of the models. The environment is currently being ported to a parallel architecture.

The remainder of the paper is organized as follows: Section 2 begins with a description of the primary components which make up mobile wireless systems. Section 3 describes the new simulation environment used to analyze the performance of such systems; we see how the environment and various models of the system are built using an existing message-passing based simulation language called Maisie. Section 4 presents the results of a simulation study to evaluate the performance of a specific mobile wireless multimedia system that is being designed at UCLA. Experiments to validate the simulation are also presented. In Section 5 we see the related work in this area, and Section 6 is the conclusion.

1 This work was supported in part by the Advanced Research Projects Agency, ARPA/CSTO, under Contract J-FBI-93-112 "Computer Aided Design of High Performance Wireless Networked Systems", and by ARPA/CSTO under Contract DABT-63-94-C-0080 "Transparent Virtual Mobile Environment."

2 This paper was in part presented at the ACM Mobile Computing and Networking Conference (Mobicom '95), Berkeley, California, 14-15 November 1995.

2. Mobile Wireless Systems

We provide a common reference model for these systems by decomposing them into two primary subsystems: network and node. The node level is used to describe the hardware and software capabilities of the (possibly) mobile node including its radio characteristics and its interface to the network operating system. The network level describes the architecture of the communication network which may be wirelined, wireless or a hybrid. In this section, we describe each of the layers in detail, and develop simulation models for these layers in Section 3.

2.1 Mobile Wireless Networks

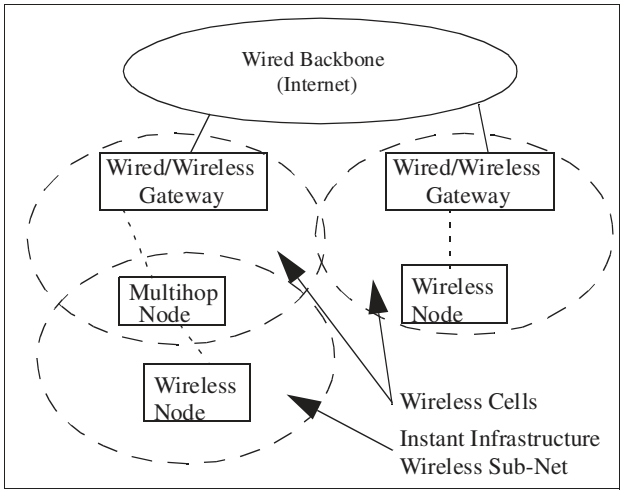


Fig. 1. Mobile Wireless Networks

Figure 1 is an example of a mobile wireless network. This network is composed not only of a static wired backbone and a few wireless cells, but also a set of nodes which are able to support instant infrastructure, self-configuring, and multi-hop functionality. We include throughout this paper the study of instant infrastructure networks [14], nodes and their algorithms since support for this architecture requires additional flexibility in the simulation environment and illustrates the complex environment in which the mobile wireless network systems can operate.

2.2 Mobile Wireless Nodes

The design of mobile wireless nodes/terminals have been studied by various groups [17][14]. In this section we describe the components which make up the node architecture and the implementation of the network control functions, multimedia support, communication substrates, and the interfaces between them. Our focus is on one of the elements shown in Figure 1, namely the wireless node; it is represented by the mobile node components shown in Figure 2. In the following subsections, we will describe various components and algorithms which make up the mobile wireless nodes.

2.2.1 Applications

The standard set of TCP/IP protocol suite applications support text based services like remote login or file transfers. New applications are now appearing which support multimedia (e.g., Netscape and video conferencing applications). Multimedia support is necessary not only for acquisition and presentation of video, speech, and data but also for coding/decoding for efficient transmission through the wireless network. To demonstrate multimedia effects over mobile wireless networks, a video conferencing application has been developed. This application (VideoTALK) brings together video, which uses UDP, and data, which uses TCP, into a single application on the laptop. To test the performance of the system for these applications, testing tools were developed to measure throughput, delay, packet loss, and to track adaptive parameters in the communication device (radio) such as code, power, and spreading factor (i.e. chips/bit). A topology analyzer program (TOPO) was developed which can be used in the simulation environment or in the implemented system to graphically display the virtual topology of the wireless multihop subnet. These tools and applications are used for experimentation and validation with simulation.

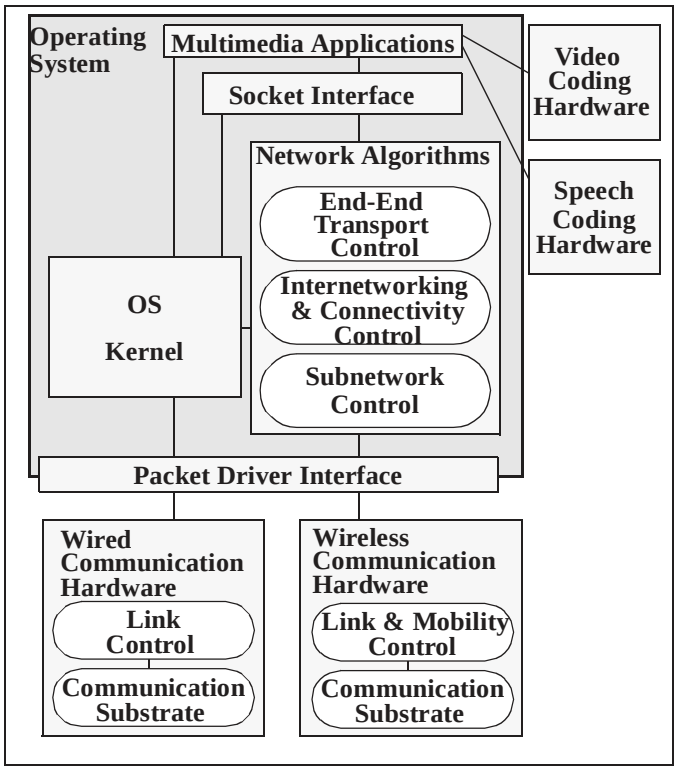


Fig. 2. Mobile Node Components

2.2.2 Operating System

The operating system is responsible for integrating all these network control components together. The choice of an operating system, such as Microsoft Windows, PC-Disk Operating System (DOS), Mac OS, or UNIX, can have significant impact on the node's capabilities and performance. However, these systems are not designed for ease of programmability or flexibility in the implementation and validation of networking algorithms and thus do not lend themselves to a flexible mobile wireless network system which can be used for experimentation or prototyping. An operating system is desired which is compatible with existing platforms (but still provides functionality such as multi-tasking and packet processing

capability useful to network control algorithms) and can be easily modeled in the simulation environment. A network operating system is able to function on a layer on top of an existing native operating system and provide the required network functionality and services. A public domain network operating system, NOS (also known as KA9Q developed by Phil Karn), has readily available source code and meets the flexibility requirements[5]. We use NOS as the operating system in our mobile wireless system (see Figure 2). It runs on top of DOS and includes its own multitasking scheduler. The benefit of this multitasking operating system is that each algorithm or protocol necessary to support this network can be developed as its own process. The multitasking kernel allows these algorithms and protocols to multitask, sharing the CPU, and yet provide semantics such as wait and signal semaphores for inter-process (inter-algorithm) communication. Time processing routines, such as TDMA, are able to sleep a process for a defined period of time, and can be used to allow other protocols and algorithms to run without halting or consuming unnecessary CPU processing time. Memory buffers (mbufs as found in BSD UNIX system buffers) are used to minimize overhead by allowing memory blocks to be linked together for performing encapsulation, packetization, etc.

The architecture of our mobile wireless system test-bench is set up to maximize the flexibility for supporting various types of mobile system components. Our current test-bench uses a NEC Versa 486 33Mhz laptop and a docking station to support the custom interfaces and hardware. The network operating system is able to run on any laptop as long as it supports DOS and the required interface cards. A Packet Interface (PI) card is used as the network interface card to integrate the wireless communication hardware into the system. To provide a standard interface to the network operating system, a packet driver interface is used, based upon FTP's packet driver specification. This interface allows interchangeability among various network interface cards (like the PI card or a PCMCIA card) without having to change the details of the network operating system to support a new or different communication substrate. A packet driver is loaded which corresponds to the correct Network Interface Card (NIC) and its capability. There are also other communication hardware drivers/interfaces such as the NDIS or ODI drivers which can be used to integrate the communication hardware with the operating system.

2.3 Mobile Wireless Algorithms

2.3.1 Transport and Internetworking Control

Since internetworking requires compatibility with existing networks and TCP/IP is so widely used through the Internet, the TCP/IP protocol suite has been implemented without need for modifications. Since the Internet Protocol can be used in conjunction with various communication substrates, much of the new mobile wireless algorithm development takes place below the network layer. The network layer is responsible for supporting various communication substrates such as internet routing, segmentation, etc. Above the network layer, the transport protocols (TCP and UDP) provide the required support for end-to-end reliability, congestion control, etc. These transport protocols interact with the applications described in the previous section by using sockets to buffer the bit stream so packetization can take place. Additional services are also being developed to support multimedia over mobile hosts [13].

Although wireless communication is useful to support mobile communication, wired connections can support a much higher bandwidth and are less prone to errors then wireless radios. Therefore, wired connections should be utilized whenever possible. Wired connections, such as ethernet, can utilize standard communication hardware, such as a PCMCIA card, for networking. To support a combination of wired and wireless communication, provide wireless multihop functionality, and support instant infrastructure networking, a node needs to be able to function in three different modes (gateway, multihop, or end node) as shown in our common reference model (Figure 3). A node functions as a gateway when

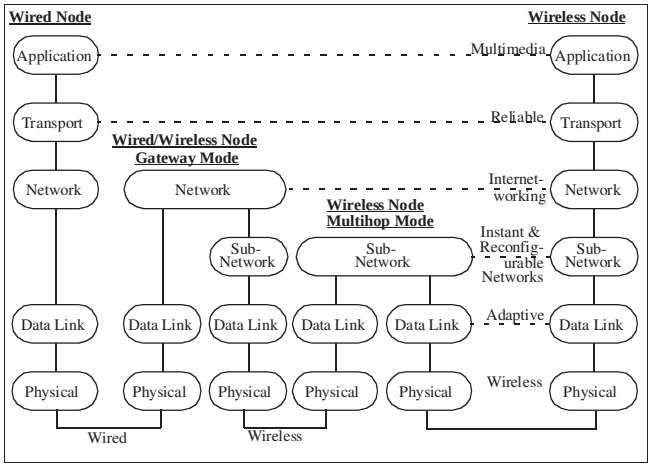


Fig. 3. Common Reference Model

both wired and wireless connections are available. In the gateway mode, it will forward packets between the wired and wireless domains as necessary. In the multihop mode, it will follow the subnetwork routing protocol to provide wireless multihop communication within the instant infrastructure subnet. Other mobile wireless network systems do not provide instant infrastructure or wireless multihop capability, but do support wireless mobility throughout an internet.

The IETF Working Group for Mobile IP has developed an Internet Draft for IP Mobility Support [12]. The primary focus of this group has been on protocol functionality and standards, rather than performance analysis. By incorporating the Mobile IP type protocol into this simulation environment, feedback can be provided to developers on its performance as a function of various mobility environments, network connectivity substrates (wireless and wired), and various traffic loads. The Mobile IP protocol can also be integrated with numerous other system components.

The analysis of the Mobile IP protocols will be useful to validate and enhance the simulation environment and can utilize the prototyping implementation path. In addition to protocol designers, the prototype can provide immediate feedback to interested groups that are developing protocols in conjunction with Mobile IP to support other network and operating system functionalities.

2.3.1 Instant Infrastructure Subnetwork Control

The functionalities which support instant and reconfigurable networks are new and have been added into the network algorithms under subnetwork control (see Figure 2). Many of the proposed schemes for supporting instant and reconfigurable network topologies are based upon TDMA to control channel contention. A clustering algorithm [10] was implemented which is heavily based on TDMA control and synchronization to test the feasibility and overhead of implementing this functionality in software.

2.4 Link Layer Control

Algorithms developed for link layer control fall into a separate category from other networking algorithms. These algorithms are not typically implemented inside the operating system, usually existing in hardware or programmable processors as part of the NIC. For maximum flexibility, simplicity of implementation, and to provide a path between simulation and implementation, these algorithms could be implemented as part of the algorithms in the operating system. To experiment and determine where an algorithm should be implemented, the simulation environment can utilize models or actual code of the link layer control algorithms.

The link layer control components typically include algorithms such as media access control (e.g., CDMA, TDMA, and CSMA/CA). The link and mobility control layer shown in Figure 2 supports a new function unique to instant infrastructure mobile wireless networking. Mobility support is provided by setting appropriate hardware parameters such as the CDMA code or transmit power level dynamically. Measurements such as Signal to Interference Ratio (SIR) are fed back from the radio into the link control algorithms for power control to minimize the power consumption, reduce interference, and provide admission control such as described in [7].

2.5 Wireless Communication Hardware

Numerous wireless radio modems are available commercially [9]. Many of the algorithms being designed for mobile wireless systems are built to support a particular device or manufacturer. Algorithms which are not designed for a specific radio face the problem of trying to predict their performance over a wide parameter space of available radio alternatives. The best way to validate over a wide parameter space of various radios is to utilize the models of the various radios in the simulation environment and experiment with actual implementation when feasible.

As an example of the complexity and trade-offs associated with developing wireless networking algorithms with wireless communication hardware, we experiment with two wireless communication hardware devices. We use the Proxim RangeLAN2 wireless frequency hop spread spectrum radio, which is commercially available, and a custom direct sequence spread spectrum radio, designed and implemented at UCLA [8]. The UCLA radio is used to support instant infrastructure networking through adaptive hardware control and feedback with the networking algorithms. This radio is currently able to operate at speeds from 7 to 32 Kbps depending on the desired spreading factor. Although other radios are able to support higher data rates, this radio provides unique control over various hardware parameters such as the spreading (chips/bit), code, power, and even acquisition time. In Table 1 we can see the spreading factor (chips/bit), data rate, and acquisition time trade-offs. It should take anywhere from 500 to 1000 data bits to acquire the signal so a preamble is sent before each packet according to the desired acquisition time. Since the radio transmits at a fixed rate of 1 Mchips/sec, and we are able to vary the number of chips/bit, we are able to achieve the various data rates described above. In order to increase the resilience to noise and interference, a parameter can be set on the radio to increase the spreading factor (chips/bit) at a cost of decreasing the data rate. By using more chips/bit (slower data rate) we are also able to potentially have higher node capacity in the wireless sub-net. It is up to the network control algorithms, with development and analysis support from the simulation environment, to dynamically determine what these parameters should be set at for optimum network efficiency.

chips per bit	Data Rate (kbps)	Optimistic ACQ Time	Conservative ACQ Time
31	32.258	15.5 ms	31 ms
63	15.873	31.5 ms	63 ms
127	7.824	63.5 ms	127 ms

Table 1: UCLA Radio Parameters

3. Simulation Environment

We have designed a general purpose parallel environment for the simulation of mobile wireless network systems and to provide an implementation path for networking algorithms. The simulation environment can be used to evaluate the effectiveness and performance of algorithms as a function of the application requirements, mobility patterns, and radio characteristics. The simulator is being built on top of an existing message-passing based parallel simulation language called Maisie [3]. Maisie is a message-based discrete event simulation language that provides a rich set of modeling constructs to facilitate the design of concise network models. The Maisie simulation environment has been implemented on a variety of workstations, networks of workstations and distributed memory multicomputers (like the IBM SP1) and on a shared memory Sparc 1000. In the following sections we will see how the Maisie constructs are used to develop various modules in the network simulation environment.

The proposed simulation environment has a number of unique features: first, it is being designed to make effective use of the facility for parallel model execution that is supported by Maisie. This potential for parallel execution of the models will allow us to investigate much larger networks than would otherwise be feasible. Second, the environment will support automatic migration of the simulation models to operational code by providing a common set of interfaces to widely used network operating systems and their models. Third, the simulation environment includes a facility for interactive control of key model parameters like mobility, transmission power, etc. This facility will allow an analyst to interactively

evaluate the impact of various changes to the hardware and protocol parameters. Lastly, the environment is modular and extensible, in the sense that different components of the mobile network can be modeled at different levels of detail. Thus, it can be useful to develop in a simulation which utilizes several different levels of detail [2].

The modeling environment is designed to allow the primary components of the wireless network system to be simulated at different levels of detail. Thus, it might be useful to initially have an approximate but fast model of all components and then refine the details of some of the components that appear to be the primary bottleneck(s). Our aim is to decompose the model in order to allow maximum flexibility in experimentation with alternative implementations of a given functionality (e.g. mobility patterns of the node) as well as to support a ''plug and play'' capability that generates composite models constructed from pieces that model system components at widely differing levels of detail.

3.1 Mobile System Simulation Modules

Our model of the mobile, wireless network system is broken down into two levels with the following primary components:

Network Level
- Node Mobility Models (MOM)
- Channel Models (CHM)

Node Level
- Wireless Radio Models (RFM)
- Operating System Models (OSM)
- Application-specific traffic models (SOURCEM)
- Network Algorithm Models (NAM)

The MOM components are responsible for movement patterns of the nodes, such as the speed in which the nodes move, and their motion pattern, such as Brownian random motion or drift. The CHM components are responsible for the transmission media including the range in which two nodes are able to communicate with each other, and environmental effects such as multi-path fading, shadowing, and interference.

The RFM components are responsible for the physical layer modeling of the radio frequency modem and includes the raw channel bandwidth, modulation techniques, and acquisition delays. The OSM simulates the relevant portion of the operating system, such as the WAMIS Network Operating System (WAMISNOS) kernel, and is involved in interfacing with the application (e.g. delivery of incoming messages) or with the network (e.g. transmission of remote messages). The OSM components include multi-tasking process scheduling, packet manipulation routines, time control, and interfacing such as between the SOURCEM and NAM and between NAM and RFM.

The SOURCEM components can be broken down into the source and destination streams (e.g., hard disk, keyboard, camera, screen, microphone, or speaker) corresponding to the voice, video and data traffic, control of these streams via the application, and the transport mechanism (e.g., TCP, UDP, or Virtual Circuits) which the application chooses to use. The NAM components are broken down into internetwork models such as IP, instant infrastructure subnetwork control (such as clustering), and mobility control (such as power control, logical link control, and media access control).

Figure 4 illustrates how the mobile wireless system simulation

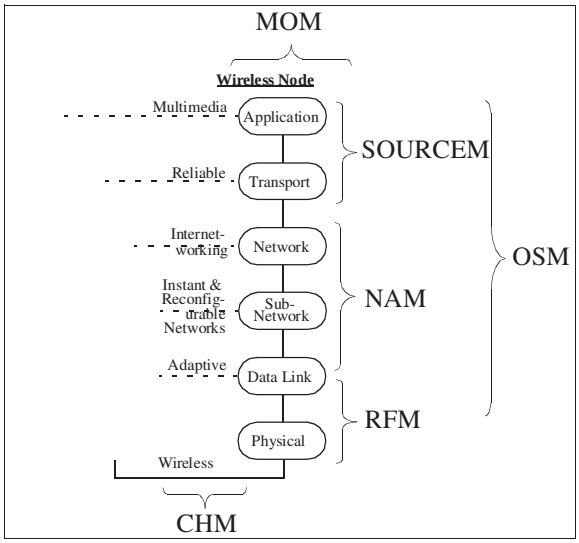


Fig. 4. Mobile System Simulation Models

models fit in with the common reference model described earlier (Figure 3).

Before providing examples of how the various models can be developed using Maisie in Section 3.3, a brief overview of the Maisie simulation language is provided in the following section.

3.2 The Maisie Language

A Maisie program is a collection of entity definitions and C functions. An entity definition (or an entity type) describes a class of objects. An entity instance, henceforth referred to simply as an entity, represents a specific object in the physical system and may be created and destroyed dynamically. An entity is created by the execution of a new statement and is automatically assigned a unique identifier on creation. For instance, the following statement creates a new instance of a manager entity and stores its identifier in variable *r1*.

r1 = **new** *manager*{N};

An entity can reference its own identifier using the keyword **self.** Entities communicate with each other using buffered message-passing. Maisie defines a type called **message**, which is used to define the types of messages that may be received by an entity. Definition of a message-type is similar to a struct; the following declares a message-type called *req* with one parameter (or field) called *count*.

message *req* {int *count*; };

Every entity is associated with a unique message-buffer. A message is deposited in the message buffer of an entity by executing an **invoke** statement. The following statement will deposit a message of type *req* in the message buffer of entity *m1*. The message will have time stamp *clock*+*t*, where *clock* is the current value of the simulation clock.

invoke *m1* **with** *req*(2) **after** t

If the **after** clause is omitted, the message is time stamped with the current simulation time. If required, an appropriate hold statements (described subsequently) may be executed to model message transmission times or a separate entity may be defined to simulate the transmission medium. An entity accepts messages from its message-buffer by executing a **wait** statement. The wait statement has two components: an optional wait-time (t_c) and a required resume-block. If t_c is omitted, it is set to an arbitrarily large value. The resume-block is a set of resume statements, each of which has the following form:

mtype(m_i) [**st** b_i] statement$_i$;

where m_i is a message-type, b_i an optional boolean expression referred to as a guard, and statement$_i$ is any C or Maisie statement. The guard is a side-effect free boolean expression that may reference local variables or message parameters. If omitted, the guard is assumed to be *true*. The message-type and guard are together referred to as a *resume* condition. A *resume* condition with message-type m_i and guard b_i is said to be enabled if the message buffer contains a message of type m_i, which if delivered to the entity would cause b_i to evaluate to true; the corresponding message is called an enabling message.

With the wait-time omitted, the **wait** statement is essentially a selective *receive* command that allows an entity to accept a particular message only when it is ready to process the message. For instance, the following **wait** statement consists of two *resume* statements. The *resume* condition in the first statement ensures that a *req* message is accepted only if the requested number of units are currently available (the keyword *msg* refers to the message that was most recently removed from the message buffer of the entity.) The second *resume* statement accepts a message of type *free* from its buffer:

```
wait until
{  mtype(req) st (units >= msg.req.count)
  /* signal requester that request is granted */
  or  mtype(free) /* return units to the pool */
}
```

Maisie also provides a number of pre-defined functions that may be used by an entity to inspect its message buffer. For instance, the function **qsize**(m_t) returns the number of messages of type m_t in the buffer. A special form of this function called **qempty**(m_t) is defined, which returns true if the buffer does not contain any messages of type m_t, and returns false otherwise. In general, the *resume* condition in a **wait** statement may include multiple message-types, each with its own boolean expression. This allows many complex enabling conditions to be expressed directly, without requiring the programmer to describe the buffering explicitly.

If two or more *resume* conditions in a **wait** statement are enabled, the time stamps on the corresponding enabling messages are compared and the message with the earliest time stamp is removed and delivered to the entity. If no resume condition is enabled, a timeout message is scheduled for the entity t_c time units in the future. The timeout message is canceled if the entity receives an enabling message prior to expiration of t_c; otherwise, the timeout message is sent to the entity on expiration of interval t_c. Thus the wait statement can be used to schedule conditional events. A **hold** statement is provided to unconditionally delay an entity for a specified simulation time. For instance, the statement **hold**(t) will suspend the corresponding entity for t units in simulation time.

3.3 Simulation Environment Modules

The simulation environment models are broken down into two categories: global and local. The global models are responsible for modeling the interaction among the nodes at the network level. The global models include the mobility (MOM) and channel (CHM) models. The local models are responsible for modeling the functionality inside a node. Local models inside a node (inter-node) can be highly integrated. The global models are used for all inter-node communication.

Let us now look at the details of some of the responsibility of each module is and an example model of one of the components.

3.3.1 MOM

The mobility models include, but are not limited to, the following components:

- tracking location of the nodes
- speed of the nodes
- direction of motion

In order for the channel model to track the location, and thus have the channel model be able to determine which nodes are able to send packets to each other, the x and y coordinates of each node are tracked. Since the mobility model is responsible for tracking the location of each node, a node can not simply update its position locally but must send a message to the mobility model to have it update the node's new location so the channel model which is coordinating communication in that area can utilize the node's location information.

In order to model speed (such as stationary, walking speed, running speed, driving speed, or even flying speed) and direction of motion (such as drift or a semi-random walk), the channel model can select a random step size which it is able to move within. Once the new position is selected and forced to remain within the space (grid) of the simulation, its new position is updated.

To get a feeling for an example model of mobility, the following Maisie fragment shows how the speed of a mobile can be modeled with a semi-random direction.

```
entity mom{}
{
 for (;;)
 {
  wait until
    mtype(move)
    {
     id=msg.move.id;
     position[id].x =
          position[id].x-(int)lrand48()%(SPEED*2+1)+SPEED;
     position[id].y =
          position[id].y-(int)lrand48()%(SPEED*2+1)+SPEED;
    if(position[id].x<0)  position[id].x=0
      else if(position[id].x>max_x) position[id].x=max_x;
    if(position[id].y<0)  position[id].y=0
      else if(position[id].y>max_y)  position[id].y=max_y;
```

```
    }
   or mtype(done)
   {
    break;
    }
  }
}
```

3.3.2 CHM

The channel model is responsible for determining which nodes are able to communicate with each other and what the received information or quality of information should look like. The CHM components can include, but are not limited to, the following:

- Distance/Range
- Shadowing (such as Log-normal)
- Attenuation (such as Free-Space)
- Multi-path (such as Raleigh Fading)

Once the channel models determine the effects of transmitting data through the wireless channel, the radio RFM models can interact in a realistic manner.

The following Maisie fragment represents a portion of code in the channel model to model radio broadcasts. The model uses the transmit power and current location of each node to determine which nodes receive an incoming packet. The actual packet (message) is sent to the appropriate node via the **invoke** statement.

```
entity chm{}
{
  ...
   wait until  mtype(broadcast)
  {
   b = msg.broadcast;
   for (i=1; i<=num_nodes; i++)
      if (i != b.id)
         if (sqrt(pow((double)(position[b.id].x - position[i].x), 2.0) +
pow((double)(position[b.id].y-position[i].y), 2.0)) < (double)
COMM_RANGE)
            invoke pktdrvr[i] with pktin{b.id,b.info};
   }
   ...
}
```

In order to simulate the packet transmission time through the channel, the transmitter should advance the simulation clock by the time needed to send the bits out the radio. This can be done using the Maisie **hold** statement.

hold(TXTIME);

The transmit time (TXTIME) is determined by the RFM and is based upon factors such as the bandwidth, packet size, and physical layer headers.

3.3.3 RFM

The RFM module is a local model which is responsible for the data link and physical layer modeling inside the node layer of the radio frequency modem, and includes, but not limited to, the following components:

- link level media access control algorithm
- NIC interfacing overhead
- acquisition delays
- raw bandwidth (data rate)
- modulation techniques (spread spectrum direct sequence or spread spectrum frequency hop)

Any time a packet is to be sent over the wireless channel, the media access control algorithm is responsible for determining if or when that packet can be transmitted. A common media access control algorithm is the Carrier Sense Multiple Access/Collision Avoidance algorithm such as found in the IEEE 802.11 specification. This will impose a delay and bandwidth overhead for every packet sent. This algorithm can be modeled inside the simulation environment to not only test feasibility and performance but also to see the implication on other aspects of the node and network. The analyst could also choose not to model the CSMA/CA algorithm itself but simply provide a metric in the RFM as the setup time before a packet can be transmitted and include this as part of the signal acquisition time (pre-amble).

Other link level control algorithms such as CRC checking, preamble, bit stuffing, etc. can be modeled at various levels of details. The model can include the details of the bits being transmitted or model this overhead by holding the RFM from being able to transmit for the period of time it would take to do such link level control processing.

The raw bandwidth affects how long it takes for a packet or bits in the packet to propagate to the next node dependent on certain parameters of the radio being used. Given the packet size, we can use the data rate to model how long it will take for the packet to be transmitted through the wireless channel.

For the UCLA radio described, the RFM parameters include 50ms for acquisition (pre-amble) of each packet, 10ms for tail processing (post-amble) on each packet, and a raw channel rate of 32 Kbps. The actual transmission time through the air can be determined in conjunction with the channel model since the transmission time (TXTIME) can be calculated as follows:

$$TXTIME = AcqTime + \frac{PktSize \bullet 8}{DataRate} + TailTime \qquad (1)$$

The modulation technique used, whether it be DSSS or Frequency Hop Spread Spectrum (FHSS), both effect the simulation environment. In a DSSS modem, the amount of spreading (chips/bit) of the original signal, or in a FHSS modem, the number of frequency bands which overlap, and thus the number of available (CDMA) codes affect the usefulness and reliability of the wireless channel (CHM) and simulation as a whole.

The most critical part of the simulation environment which integrates the various components and can have a significant impact on the simulation as a whole is the operating system.

3.3.4 OSM

The Operating System Model has three primary components:

- kernel model
- application interface model
- network interface model

The kernel model provides the basic functionality needed to simulate a multi-tasking OS kernel. It models a (dynamic) set of interacting processes, where each process is simulated by a Maisie entity and the inter-process communication and synchronization is simulated by appropriate message communication among the corresponding entities. Henceforth, we use the term ''kernel entity'' to mean a Maisie entity that is simulating a NOS kernel process.

The KA9Q kernel uses interrupts to interface with many of its drivers; hence the kernel entity used in the simulation environment models also supports interrupts. The entity may (dynamically) specify the set of enabled interrupts. A common source of interrupts in the kernel is the arrival of a packet for the corresponding entity. We present a short Maisie fragment to illustrate the handling of an interrupt called ''pktin'' by a kernel entity called ''wproc''. The **wait** statement on the following fragment models an interruptible activity. The time specified in the wait statement is initially set to t_c, which models its execution time in the absence of any interrupts. If an interrupt *(pktin)* is received during this interval, the entity suspends normal operation, executes a pre-specified routine to handle the interrupt, and suspends itself for t_i time units, where t_i models the time taken to execute the interrupt handling routine in the physical kernel. Note that this model assumes interrupts cannot be nested, because a **hold** statement is used to simulate service of the interrupt. It is possible to instead use an interruptible **wait** statement to model nested interrupts. After executing the **hold** statement, the entity again executes the **wait** statement with an updated wait-time to complete the simulation of the original activity. For simplicity all time units are expressed as integers in this fragment. The function **clock()** returns the current value of the simulation clock.

```
entity wproc{id,pktdrvr,ipalgptr,tc,ti}
int id; /* My Node ID */
ename pktdrvr; /* Pointer to Packet Driver Interface */
ename ipalgptr; /* Pointer to IP Protocol Processing Routine */
int tc; /* Initial Execution Time */
int ti; /* Interupt Handling  Routine Time */
{
 message pktin{int pkttype; int len; int id; int info;} pkt;
 int newlen, remtime, endtime;

 for (;;)
 { endtime=clock()+tc;
   remtime=tc;
   for(;;)
    wait remtime until
    { mtype(pktin)
      { pkt=msg.pktin;

        if (pkt.pkttype==clust_type)
          clust_got_pkt(id,neighbor,I_am_ch, pkt.id,pkt.info)

        else if (pkt.pkttype==ip_type)
          ip_got_pkt(id,newlen,pkt.id,pkt.info, pktdrvr,ipalgptr);

        remtime=endtime-clock();
        hold(t1);
      }
      or mtype(timeout)   break;
    }
 }
}
```

The application interface model interacts with the SOURCEM model to both accept a message for delivery to another node and also to deliver an incoming message. In either case, the kernel provides the interface needed by the application to the network and simulates the software delays that are typically suffered by the message as it passes through the kernel of an operational OS. This delay can be simulated either by doing a detailed (and hence time-consuming) simulation of the various kernel modules, or approximated by simply delaying the message by a randomly distributed value, where the distribution is chosen to reflect the aggregated behavior of various kernel modules.

Similarly, the network interface model will determine the transmission mode of the message (e.g., datagram or bit stream) and provide the message to the NAM in an appropriate format from the network interface. A driver such as the packet interface driver is typically used as the NIC interface. Note that the kernel delays can be simulated either in the application or the network interface model (or both), depending on the analyst and the application being simulated. The applications are represented in the system as the source model.

3.3.5 SOURCEM

The SOURCE models are composed of, but not limited to, the following components:

- source & destination streams
- application control
- end-to-end transport mechanisms

One of the primary uses of the mobile wireless network nodes are to exchange data, voice, or video. The input or source of the data, voice, or video usually comes from either the hard disk, memory, keyboard, microphone, or camera. The output or destination usually goes to either the hard disk, memory, screen, or speaker. Depending upon the analyst's need, it is typically not required that the actual data, voice, or video images be sent from one source stream to the destination but rather modeled based upon certain characteristics. The characteristics modeled for the hard drive and memory include read and write access time, models of the voice streams include the rate and silence characteristics, and models of the video stream usually include the frame size, frame rate, and other control information such as frame delimiters.

The application control component is responsible for controlling the source and destination streams in conjunction with the transport protocols. The application affects the environment such as by determining if, when, and what data, video, or speech should be sent. Typical applications used in the mobile wireless system implemented include the standard TCP/IP applications such as FTP and telnet along with custom multimedia applications such as a video conferencing (VTALK) application.

In order to deliver the streams of data, video, and speech an end-to-end transport mechanism is used. These protocols typically include TCP and UDP for data and usually virtual circuits for multimedia in order to provide bandwidth allocation. Typical functionality of the transport protocols include providing flow control, error detection and possible retransmission of lost or corrupted data, and

acknowledgment of data received. As an example, we can see in the following Maisie fragment the functionality of TCP and FTP used in a file transfer to send data, check for acknowledgments of sent data, and retransmit lost packets upon a time-out.

```
for (i=MSS;i<FILE_SIZE-MSS;i=i+MSS)
{
  wait RTO until /* RTO = Round-trip TimeOut */
  {
    mtype(ack); /* Packet Received */
    or mtype(timeout) /* Pkt or ACK Lost */
      i=i-MSS; /* Resend last packet */
  }
  /* Generate ftp packet */
  sendpacket(pktdrvr, ftp_type, id, 0, i, MSS);
              /* Type, From, To, Info, Len */
  num_pkts_out[id]++;
}
```

In order to model the source and destination streams, application control, and transport mechanism, traffic generators are used to generate the data streams corresponding to the voice, video, or data traffic expected to be generated by the different types of applications. Table 2 lists a set of example applications. For each application, the transport protocol that is commonly used, typical packet size, traffic type, and metric to be optimized is listed.

Appl.	Trans.	Pkt. Size	Traffic Burstyness	Goal
FTP	TCP	Large	Low	Max. Throughput
telnet	TCP	Small	High	Min. Delay
vtalk: data & video	TCP UDP	Small Large	High Low	Min Delay Max Throughput
video	V.C.	Large	Low	Max. throughput
speech	V.C.	Small	High	Delay & Throughput

Table 2: SOURCEM Characteristics

3.3.6 NAM

The network algorithm model components are the focus for those developing wireless and mobile networking algorithms. We break the Network Algorithms Models into the following layers:

- network layer
- sub-network layer
- data link layer

The network layer components include the internetworking functionality. The Internet Protocol is commonly used either in its entirety or just a model of IP to provide functions such as domain addressing, routing, segmentation, and reassembly. Other protocols modeled in this layer include the ICMP for control messages and Mobile IP [12] for mobility tracking and support of roaming through the internet.

The wireless subnet, whether it be a base station and its clients or a wireless multihop cluster are found in the sub-network layer. The subnetwork layer models are used to model the topology creation (instant infrastructure), reconfigurability, adaptive channel assignment (CDMA), and wireless multihop routing.

As an example of a NAM, below is a Maisie fragment for the clusterhead election algorithm found in [10]. The basic idea of the algorithm is that between any two nodes that can communicate, the node with the lowest ID should become the clusterhead with the restriction that two clusterheads can not communicate directly; however, they can communicate via a gateway by multi-hopping between the two clusters.

```
entity clust_proc{id,pktdrvr,neighbor,I_am_ch}
int id;
ename pktdrvr; /* From OSM */
int *neighbor;
int *I_am_ch;
{
  for (;;)
  {
    hold(RESET_TIMEOUT);

    /* Reset neighbor and clusterhead tables */
    for (i=1; i<=N; i++)
    {
      neighbor[i]=-1;
      I_am_ch[i]=0;
    }

    /* Send "I'm here" msg to all neighbors */
    invoke pktdrvr with broadcast{id, 0};

    /* Wait to hear responses from neighbors */
    hold(RESPONSE_TIME);

    /* Run the Clusterhead election alg. */
    I_am_ch[id] = 1;
    for(i=1;i<id;i++)
      if ((I_am_ch[i]==1)&&(neighbor[i]==1))
      {   I_am_ch[id] = 0; break; }

    /* Broadcast Clustering Packet Update */
    /* I_am_ch (1) = Not CH;    I_am_ch(2) = CH */
    invoke pktdrvr with
          broadcast{id,I_am_ch[id]+1};
  }
}
```

The *clust_proc* Maisie fragment first initializes the neighbor status upon a reset timeout. Then each node broadcasts a message using the **invoke** statement in order to determine connectivity and find out who their neighbors are. The **hold** statement is used to wait for responses from other nodes. Starting with the lowest ID, the algorithm iteratively determines who can be clusterheads. Finally, the neighbors are told whether or not each node thinks it is a clusterhead.

The data link layer models are used to provide mobility and link level control such as power control [7] (utilizing various power levels available on the radio and adapting the SIR measurement), media access control via a TDMA based time frame[10], error control such as the spreading factor which the radio transmits on, the CRC functions, and possibly even the Reed-Solomon forward error correction, and lastly the logical link control such as providing a hop by hop based acknowledgment scheme such as described in [16]. These models can be refined or simplified as desired.

4. Example Study

In this section, we provide results and comparisons from experimentation and simulation of a point to point file transfer over a wireless network to determine where the bottlenecks lie in the node performance. Our example study uses the FTP application, which is built upon TCP, to determine file transfer throughput. The TCP parameters were customized to maximize the possible efficiency and surface node performance limitations. Admittedly these are very elementary models for the general purpose simulation environment that has been described, but it allows us to illustrate the interaction of the various models in the simulation environment.

In order to validate the network algorithms being developed for mobile wireless systems, a prototyping test-bench is set up to test the instant infrastructure networking capabilities of the Wireless Adaptive Mobile Information System (WAMIS) research project at UCLA [14]. These nodes are also used as a test-bench for experimentation and validation of multimedia coding algorithms and prototype wireless communication hardware.

To validate the simulation models, actual measurements were done using 2 486-based laptops hooked up with the UCLA designed radios running WAMISNOS to provide a point to point wireless link. WAMISNOS is built upon KA9Q NOS, which includes the complete TCP/IP protocol suite. WAMISNOS provides several custom protocols and algorithms for adaptive instant infrastructure wireless networking, customizable parameters for the various algorithms, and performance hooks and measurement tools for analysis.

4.1 Simulation Models

We have developed several simple modules in this simulation environment to model the functionality and performance of the various components including the network operating system (OSM), FTP application and TCP transport protocol (SOURCEM), network algorithm header effect and Maximum Transmission Unit (MTU) limitations (NAM), two wireless radio modems (RFM), and the reliability of the wireless channel (CHM).

4.1.1 OSM

In order to model the performance of the WAMIS Network Operating System (WAMISNOS) running on the 486 laptop, experimentation was done to find out the average processing time for incoming and outgoing packets. In Section 4.3.4 we will examine how the measurements were done in more detail and their effect. We found that the average time for the transmitter to transmit the next packets once it received the ACK was around 8ms, whereas the response time from when a packet arrived into WAMISNOS on the receiver side until an ACK could be generated averaged around 37ms. Since the source had to receive the ACK and transmit the packet, in order to estimate the input processing time of a packet for the OSM, we found the average processing time to be 23ms ((37+8)/2).

For every packet received we would enforce a Maisie **hold** of 23ms for WAMISNOS processing and similarly we would hold for 23ms for every packet sent out through WAMISNOS.

4.1.2 SOURCEM & NAM

The modeling of the file transfer application and TCP protocol are done in the SOURCEM module as we saw in Section 3.3.5 and the various parameters are shown in Table 3.

Parameters in TCP which are customizable or tunable include: the backoff algorithm (exponential or linear), initial round trip time (IRTT), maximum segment size (MSS), and the window size (WINDOW). The backoff algorithm is designed to provide congestion control throughout the network. The most fair algorithm used is an exponential backoff algorithm. However, since congestion would not occur in a point to point file transfer (only 1 link) this backoff algorithm was replaced with a linear backoff algorithm. The round trip time is used for determining what the time-out should be for retransmitting lost packets. This round trip time is based upon an adaptive algorithm which is constantly measuring and adapting to the current round trip time. A stability parameter is specified which weights the current round trip time with the average round trip time. Since TCP is responsible for packetizing the data bit stream, the maximum segment size specifies the maximum packet (segment) size which TCP can generate. IP uses a MTU which specifies the largest packet that can be sent over a particular network or link. If the segment size is larger then the packet size then IP does segmentation and reassembly of the packet. So, we set the MSS to be 40 bytes less (to compensate for headers) then the MTU. Finally, the window size specifies how much data can be outstanding before an acknowledgment is required. The benefit of having a large window is to handle the case when the latency of the path is significant compared to the bandwidth. That is, if you can fit more than 1 packet on the path at a time, then it is useful to have a window so the bit pipe can be filled. For our wireless radios, the latency is insignificant compared to the bandwidth so the window should be set to equal the MSS.

Description	Value
SOURCEM TCP Backoff Algorithm	Linear
SOURCEM File Size	1751560 Bytes
SOURCEM MSS	3960 Bytes
NAM MTU	4000 Bytes
NAM Header Size	71 Bytes

Table 3: SOURCEM & NAM Parameters

The effects of customization on the performance is significant. With standard parameters used on most TCP/IP implementations, the overhead with UCLA's Radio approaches 99% (depending upon link errors, back-off algorithm, etc.) Given that customization can be achieved through better integration of the protocols and link level implementation. This paper attempts to identify the remaining bottlenecks.

4.1.3 RFM

The UCLA Direct Sequence Spread Spectrum Modem/Radio used in experimentation and simulation operates at a fixed chip rate of 1.032Mchips/sec. With a spreading factor of 32chips/bit, it is able to achieve a data rate of 32.258 Kbits/sec. A packet interface card is used to connect the radio with the computer and a packet driver is used to connect the packet interface card with the WAMIS Network Operating System. The various rates and customized parameters for this experiment are shown in table 4.

Description	Value
Raw Channel Rate	32.258 Kbits/sec
Maximum Trans. Unit	4000 Bytes
Acquisition Time	200ms
Tail Time	10ms
Media Access Control	CSMA
CHM Packet Loss Rate	.15

Table 4: UCLA Radio & WAMISNOS Parameters

Based upon the radio experiments with indoor channel models, we found the average packet loss to be around 0.15. A packet is lost any time the CRC checksum fails, the radio fails to acquire the signal in time, or there is data corruption such as from interference or background noise.

4.2 Validation

Table 5 compares the performance of the FTP application as predicted by the simulation model with actual measurements. We find

	Sim.	Exper.
Data Bytes	1751560	1751560
Packets In	444	589
Packets Out	443	569
Time (ms)	823023	942710

Table 5: Simulation & Experimentation Comparison

the simulation results come close to those found in experimentation. The majority of the difference lies in the accuracy of the TCP model. A fixed RTO (Retransmission Timeout) was used for every packet that was lost whereas in the experiment, TCP determines this parameter dynamically. We also found that through experimentation the packets were not always filled as was the case in the simulation. This can probably be attributed to the stream processing functions in WAMISNOS which could be modeled in the simulation environment as part of the SOURCEM.

4.3 Performance Breakdown Analysis

Table 6 presents a breakdown of the various sources of overhead in the FTP application as determined by experimental measurements. We first examine the sources for each component and subsequently compare the experimental results with the simulation results.

Description	%
1. User Data Transmission (Efficiency)	46.0
2. Acquisition Time	24.6
3. Time-outs (Packet Loss)	19.8
4. CPU Processing (Rx + Tx)	2.8
5. TCP/IP/WAMIS Headers	2.2
6. Tail Time	1.2
7. Misc. (H/W Proc, CRC Checking, Bit Stuffing...)	3.4

Table 6: Performance Breakdown

4.3.1 SOURCEM Efficiency

When using the UCLA Radio for the file transfer of the 1.7Megabyte file it took 942.71 seconds (as reported by the application), with an overall throughput of 1,858 Bytes/Sec. or 14,864 bits/sec. This means that the efficiency of the file transfer was about 46%. We use the following calculation to determine the efficiency:

$$\frac{\frac{FileSize}{ChannelRate} \times \frac{Bits}{Byte}}{TotalTime} = Efficiency \qquad (2)$$

We see that the largest percentage of our breakdown is the user data (efficiency) which is 46%. At first this seems very good that the user is able to achieve 46% utilization of the link bandwidth, however the link bandwidth is only 32Kbits/sec so the user is able to achieve 14.7Kbps. If we were able to increase the channel rate, even at the cost of decreasing the link efficiency, we could achieve a better user throughput. This means that the largest bottleneck in getting better performance is the limitation in the transmission rate (raw channel rate) of the radio (32Kbits/sec.).

4.3.2 RFM Acquisition Time

The second major bottleneck is the acquisition (25%). Each time a packet is transmitted the radio has to go through an acquisition of the channel which is done by adding on 200ms worth of preamble data to the beginning of each packet. It is possible to shorten this preamble time but the error rates and thus retransmission of the data increase dramatically causing overall poorer performance. Besides modifying the required time to acquire the channel, this overhead can be reduced by decreasing the number of packets transmitted.

The larger the packet size, the lower the number of packets, and thus the less overhead for acquiring all the packets. One of the major factors enforcing the packet size is the bandwidth-delay trade-off. By increasing the packet size, we can reduce overhead and increase bandwidth but at the cost of delays (and having to retransmit more data). To keep the delays and memory requirements for storing packets to a minimum, the packet size (MTU) is constrained to 4K in the current UCLA Radio-WAMISNOS implementation.

The overhead for acquisition was calculated using the following:

$$\frac{\text{TotalNumPkts}(Tx+Rx) \times \frac{AcqTime}{Pkt}}{TotalTime} = AcqOverHead \quad (3$$

WAMISNOS includes the ability to monitor and the number of WAMIS Packets, IP Packets, and TCP segments sent and received at each node. The numbers of TCP segments sent and received make up the TotalNumPkts since no segmentation was necessary in IP (which would cause generation of more packets), and there were not any WAMIS control algorithms running which would generate additional packets to the radio. There were 569 data packets sent and 589 acknowledgment packets sent. Each packet had a 200ms header and the total time for the file transfer was 942.71 seconds or 24.6%.

Tail time is similar to acquisition time; it is the amount of postamble used on each packet. This is required to ensure that the packet is completely sent out before the carrier signal is dropped. Experimentation shows that 10ms is an adequate tail time. The tail time overhead can be calculated similar to the acquisition time and is found to be 0.012 of the total raw bandwidth.

4.3.3 SOURCEM Time-outs & CHM Packet Loss

Note that 19.8% of the throughput is lost due to time-outs. Time-outs occur when a packet is lost (the receiver fails to lock onto the packet or one or more bit errors occur causing the CRC check to fail and the packet to be discarded) and then the sender must wait for the time-out period to occur (failure to get an acknowledgment) before the packet is retransmitted. The variable time-out period is called the RTO and varies based upon the measured round trip time of data flowing across the path and then a weighting is done for stabilization. The base RTO varies around 2200 milliseconds. When a packet loss does occur, the linear backoff algorithm would increase in the time before the next packet is transmitted. The time-out starts increasing linearly as several packet losses occur in a row. If an exponential backoff algorithm were used, the RTO would have grow exponentially at this point rather than linearly, making the throughput dramatically worse.

The following calculation was used as an estimation of the time-out overhead.

$$\frac{NumPktsLost \times RTO}{TotalTime(ms)} = TimeoutOverHead$$

During this test, there were 85 packets that had to be retransmitted and the average base RTO was around 2200ms so we find TimeoutOverHead to be 19.8%.

4.3.4 OSM Processing

Not as significant as the first three overheads, CPU Processing does make an impact on the performance using the UCLA Radio.

The Transmitter (Tx) is responsible for taking the bit stream and forming the packets, and putting the header information on it. We use a hook in the WAMISNOS system which allows us to watch at what time (in milliseconds) when an acknowledgment of a packet comes in from the packet driver into WAMISNOS and until the next packet is transmitted from WAMISNOS to the packet driver. We found that the average time is 8ms. If we multiply the number of packets sent (569) by the amount of processing time (8ms) per packets, we find the transmitter CPU processing overhead to be 4.5seconds or 0.5% of the total overhead.,

The receiver (Rx) has to check and remove all the header information from the packet and verify that the data is correct (passing the CRC check) and create a response (acknowledgment) to the sender informing that the data was received correctly. It was measured using the trace facility built into WAMISNOS that the time from when a packet first arrives in WAMISNOS from the packet driver until the acknowledgment goes out WAMISNOS back to the packet driver around 37ms. Since the TCP/IP protocols and the WAMIS Network Operating System are both competing for CPU time, along with other applications, protocols, etc., this number can have a high variance, so much that it would impact the performance of any time critical algorithms which needed to run at a particular time, such as TDMA. Since 589 packets were received and each had to be processed (37ms/pkt) the total overhead imposed by the receiver CPU processing was 21.8sec or 2.3% of the overall bandwidth.

The total CPU Processing time is the sender's overhead (0.5%) plus the receiver's overhead (2.3%) which totals 2.8%, as is found in Table 6.

4.3.5 NAM Headers

The application, FTP, uses TCP as its reliable connection oriented transport protocol. The TCP protocol packetizes the bit stream into segments and encapsulates it with a TCP control header. This TCP header is usually around 20 bytes. The TCP header contains information such as the source and destination port (application), the sequence and acknowledgment number, a 16 bit checksum, and some miscellaneous flags. TCP sends the segment down to the IP protocol which encapsulates the segment into a

packet and puts on its own header of approximately 20 bytes. The IP header contains information such as the total length of the packet, a 16-bit checksum, an identification field, and source and destination IP addresses. From here, IP sends the packet down to the WAMIS algorithms which puts on an additional 31 byte header which contains information such as the source and destination hardware node address, code and power control information, SIR control information, etc. The total TCP/IP/WAMIS headers are usually around 71 bytes.

There were 569 data packets sent and 589 acknowledgment packets sent and at 71 bytes per packet, the total time used up (overhead) in transmitting header information was about 20.4 seconds (2.2%).

4.3.6 Miscellaneous

There are a number of other miscellaneous factors which add to the total overhead (3.4%). It was not possible using the current analysis and software tools to determine the exact processing time by the software below the WAMIS Network Operating System. This includes the time for the packet driver to activate, calculation of a CRC check for the packet, and Carrier Sense Multiple Access. The packet has a start of packet (STX) and end of packet (ETX) marker so the receiver will know the exact beginning and ending of the packet. Bit stuffing is used to ensure that none of the data inside the packet would look like one of these delimiters. Then the data has to be sent out of the packet interface card to the modem and from there the processing can take place to send it out to the transmitter. The opposite process has to take place on the receiving end.

This overhead was not measured but is the remaining of the overheads which had not been compensated for in the analysis above.

4.3.7 Performance Breakdown Validation

We found that through customization of TCP parameters, we were able to achieve a link efficiency of 46% (14.8Kbps) using UCLA's Radio with WAMISNOS. As shown in Table 7, the three largest bottlenecks in this system are 1) the raw channel rate, 2) acquisition delays, and 3) time-outs in TCP caused by bit errors and packet losses in the link. The magnitude and order of the performance bottlenecks are the same in simulation as found through experimentation with the UCLA radio and test-bench.

Description	% of Raw Channel Rate (Experiments)	% of Raw Channel Rate (Simulation)
1. Data Bandwidth	46.0	52.7
2. Acquisition Time	24.6	21.0
3. Packet Loss & Time-outs	19.8	17.0
4. CPU Processing	2.8	2.0
5. Protocol Headers	2.2	1.8
6. Tail Time	1.2	1.0
8. Other	3.4	4.4

Table 7: Performance Comparison & Validation

In order to evaluate the performance bottlenecks using a different wireless communication hardware and driver, we used a commercial radio on our test-bench to see where the bottlenecks lie on a higher speed wireless radio using different link level networking protocols.

4.4 Extending the Example Study

In this section, we examine the performance breakdown of a higher speed wireless radio by repeating the experiment using a Proxim RangeLAN2 wireless radio. This experiment is able to validate the performance bottlenecks which we examine under various parameter spaces to determine the trade-off point between the various bottlenecks.

We repeated the experiment using the Proxim RangeLAN2/ PCMCIA Wireless LAN Adapter [18]. This radio uses Frequency Hop Spread Spectrum, operates at a raw channel rate of 1.6 Mbps, and uses the RangeLAN2 CSMA/CA media access protocol. By using the packet driver for this radio, we were able to utilize this radio as part of the test-bench in place of the UCLA radio [Figure 2].

The similar set of statistics were tracked for the Proxim radio as was used in the UCLA radio performance breakdown. The breakdown analysis and comparison between the UCLA radio and Proxim radio are shown in Table 8.

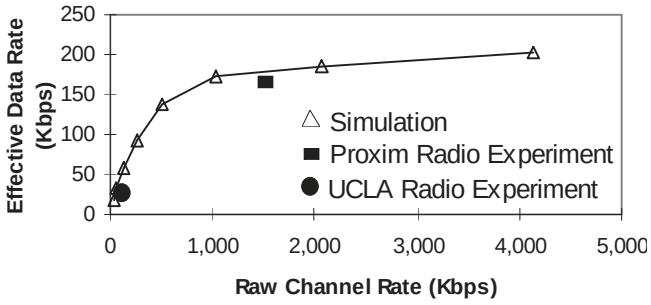
Description	UCLA Radio (32Kbps)	Proxim Radio (1.6Mbps)
1. Data Bandwidth	46.0	10.1
2. Acquisition & Tail Time	25.8	0.0
3. Packet Loss & Time-outs	19.8	0.0
4. CPU Processing	2.8	63.9
5. Protocol Headers	2.2	1.0
6. Miscellaneous (H/W Proc.,Media Access,etc.)	3.4	24.9
7. Total	100	100

Table 8: Experimental Comparison (UCLA vs. Proxim) [Percentage of Raw Channel Rate]

We found that the performance bottlenecks were significantly different between the UCLA and Proxim radios. The data bandwidth seen by the application decreased from 46% of the raw channel rate to 10.1%. Therefore, the Proxim radio, which is rated at 50 times faster (1.6Mbps) then the UCLA radio (32Kbps) in raw chan-

nel rate, is able to achieve only a 11 times increase (161.6Kbps) in effective (user) data bandwidth over the UCLA radio (14.7Kbps).

In order to examine how the effective data rate changes as a function of the raw channel rate, we are able to utilize the simulation models and simulation environment, described earlier. The results are shown in Graph 1. We see a very close correlation between the

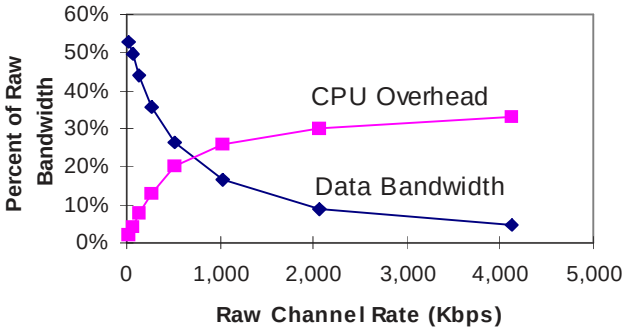


Graph 1: Effective Data Rate vs. Raw Channel Rate

simulation results shown with those found through experimentation with the UCLA and Proxim radios.

As the raw channel rate increases, we find that the data bandwidth bottleneck decreases, as a percentage of the raw channel rate. We see in Table 8 that there is a significant increase in CPU Processing overhead (from 2.8% to 63.9%) even though the exact same laptops were used due to the increased raw channel rate (32Kbps vs. 1600Kbps). Now the CPU Processing became the largest bottleneck for the Proxim radio.

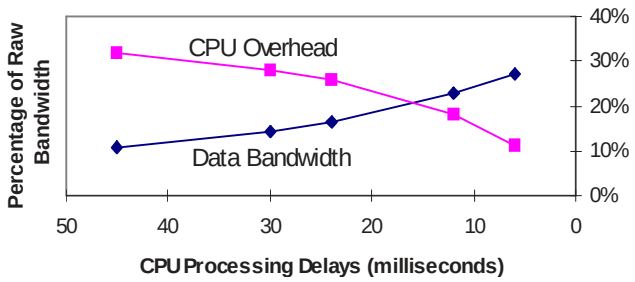
In order to evaluate the CPU Overhead versus Data Bandwidth for a larger parameter space of various raw channel rates, we used the simulation models again to obtain Graph 2.



Graph 2: Bottleneck vs. Channel Rate
(15% Packet Loss Rate)

We see how the CPU Overhead becomes a dominant factor of the raw channel rate around 700Kbps and thus the decline in the data bandwidth efficiency when the CPU processing time is fixed at around 23ms per packet. In this graph, we see that the CPU Overhead attributes to approximately 30% of the loss in bandwidth. However, in Table 8 we found through experimentation that the CPU Processing overhead for the 1,600Kbps Proxim was 63.9% of the raw channel rate. The difference between simulation and experimentation is that in the simulation models, we assume TCP sees a packet loss rate of 15%, and in our experiments, TCP saw no packet loss. We attribute this to a link level acknowledgment scheme as part of the link level protocol used inside the Proxim RangeLAN2.

As technology advances the raw channel rate in which these wireless radios are able to operate, so will advance the processing speed of the laptops. To see the effect of the CPU Overhead and Data Bandwidth as a function of CPU Process delays, we used modified the simulation parameters in the models described earlier to obtain the performance bottlenecks at various CPU processing delays. The results are shown in Graph 3. Here we see trade-off



Graph 3: Bottleneck vs. Processing Delay

between the CPU Overhead and Data Bandwidth efficiency of the channel as CPU processing delays (per packet) decrease and the channel rate remains fixed at 1,000 Kbps. When the CPU processing delays fall below 17ms per packet, the channel rate starts becoming the larger bottleneck.

5. Related Work

There are several different network simulators currently on the market. These simulators have primarily been used for design and performance evaluation of networking algorithms. The problem with these simulators is the lack of full flexibility for customization such as modeling the operating system kernel or system interfacing found in the implemented system.

Many existing commercial network evaluation tools suffer from the following limitations which are addressed in this simulation & prototyping environment:

- Most tools are not tailored for wireless protocols and have awkward and inadequate interfaces for specifying wireless and mobility related parameters.
- Models generated by existing tools are often of little use in generating working implementations of the protocols. For instance, the finite state machines used to specify the protocols in OPNET must be manually re-coded to design an operational prototype. This leads to unnecessary duplication of resources and is also error prone.
- As yet, no common reference model exists for most mobile and wireless parameters such as performance measurements.
- Existing prototyping tools do not provide a way to incorporate operational protocols into the modeling environment. An important component of the simulation environment is *backporting* of existing software and

protocols into the simulation environment for scaling studies and validation as well as for testing inter-operation with novel protocols.

- Existing simulation tools are extremely slow. Models with even a relatively small number of mobile devices (e.g., personal communication systems) can take hours of execution time on contemporary workstations. Scalability studies involving hundreds, and perhaps thousands of these devices, are practically impossible using these tools.

In addition to the related work with commercial products, various research projects at other universities are working on specific simulation and implementation environments for mobile, wireless, and networking protocols. A simulation environment was developed specially for the x-kernel [11] which successfully analyzes the performance of the new protocol based upon various simulation model parameters. In order to support implementation, the approach used in the x-kernel and Scout projects at the University of Arizona is to develop an operating system which can support the implementation of networking protocols [1]. Of the related work, few address the development of a simulation environment to model the various components used in mobile wireless network systems, and none of them support parallel simulation and a direct path between implementation and simulation for validation and experimentation.

6. Conclusion

This paper described a software architecture for a simulation environment for mobile wireless network systems. The environment provides clearly delineated modules to model each of the primary components of the system: network operating system, traffic models, protocol models of the network, data, and physical link levels, radio models, and mobility patterns. Each of these models can be as simplistic or detailed as desired. The environment has been used to perform a number of studies: this paper described only one simple study that used the NOS, radio, and channel models to evaluate a point-to-point file transfer protocol over a wireless network. The test-bench not only provides a path for implementation from the simulation environment, but also validates the simulation results. The test-bench and simulation environment are flexible enough to be used for various aspects of simulating mobile wireless network system components and their integration. The time to achieve analysis results have been significant reduced by being able to write models which transparently run in a parallel simulation language and supports refinement of the models as desired.

The experiments reported in this study used only the sequential Maisie implementations. Parallel Maisie implementations have yielded significant performance improvements for a number of example studies[4]. We intend to explore the viability of the parallel implementation in improving the performance of simulation models for wireless networks such as those described in this paper. We are also extending this simulation environment and test-bench to support nomadic computing issues [15] and various transparent nomadic networking protocols.

7. Acknowledgments

We would like to thank Jack Tsai, Mario Gerla, and Eric Wu for their support in developing some of the models used in this simulation environment, as well as everyone in the WAMIS project for their contribution in development of the wireless multimedia system test-bench.

8. References

[1] Abbott, M. and L. Peterson, A language-based approach to protocol implementation, Technical Report # 92-2, University of Arizona CSD, July, (1992).

[2] Bagrodia, R., M. Gerla, L. Kleinrock, J. Short, T.-C. Tsai, A Hierarchical Simulation Environment for Wireless Networking Algorithms, Winter Simulation Conference, (1995).

[3] Bagrodia, R. and W-L. Liao, Maisie: A language for design of Efficient Discrete-Event Simulations, *IEEE Transactions on Software Engineering*, April (1994).

[4] Bagrodia, R., Z. Li, V. Jha, Y. Chen, and J. Cong, Parallel Logic-level simulation of VLSI circuits, *Winter Simulation Conference*, Orlando, FL, December (1994).

[5] Boring, W. and J. Short, UCLA WAMISNOS Network Protocol Programmers Guide, UCLA Computer Science Department Technical Report # 950010, April (1995).

[6] Chen, S., N. Bambos, and G. Pottie. Admission control schemes for wireless communication networks with adjustable transmitter powers, *INFOCOM 94*, Toronto, Canada, (1994).

[7] Chen, S., N. Bambos, and G. Pottie, On Distributed Power Control for Radio Networks, *IEEE International Conference on Communication,* New Orleans, LA, pp. 1281-1285, (1994).

[8] Chien C., et al., A 12.7 Mchips/sec All-Digital BPSK Direct-Sequence Spread Spectrum IF Transceiver, *IEEE Journal of Solid-State Circuits*, Vol. 29, No. 12, December (1994).

[9] Cox, D., Wireless Personal Communications: What Is It?, *IEEE Personal Communications*, Vol. 2, No. 2, pp. 20-35, April (1995).

[10] Gerla, M. and J. Tsai, Multicluster, mobile, multimedia radio Network, accepted for publication in *Wireless Networks Journal* (1995).

[11] Hutchinson, N. C. and L. L. Peterson, The x-Kernel: An architecture for implementing network protcols. *IEEE Transactions on Software Engineering* 17(1):64-76 (1991).

[12] IP Mobility Working Group, Routing Support for IP Mobile Hosts, Internet Engineering Task Force, Internet Draft, (1995).

[13] Mah, B., S. Seshan, K. Keeton, R. Katz, and D. Ferrari. Providing Network Video Service to Mobile Clients, *Proceedings of the Fourth Workshop on Workstation Operating Systems*, Napa, CA, October (1993).

[14] Jain, R., J. Short, S. Nazareth, L. Kleinrock, and J. Villasenor, PC-notebook based mobile networking: Algorithms, Architectures and Implementation, *IEEE International Conference on Communication*, Seattle, WA, (1995).

[15] Kleinrock, L., Nomadic Computing - An Opportunity, *ACM SIGCOMM*, Vol. 25, No. 1, pp. 36-40, January (1995).

[16] Lin, C. and M. Gerla, A Distributed Control Scheme in Multi-hop Packet Radio Networks for supporting Voice/Data Traffic, *IEEE International Conference on Communication,* Seattle, WA, (1995).

[17] Long, A. C. Jr., S. Narayanaswamy, A. Burstein, R. Han, K. Lutz, B. Richards, S. Sheng, R. W. Brodersen, and J. Rabaey, A Prototype User Interface for a Mobile Multimedia Terminal, *Proceedings of the 1995 Computer Human Interface Conference*, May (1995).

[18] Proxim, Inc., RangeLAN2/PCMCIA User's Guide, Mountain View, CA, (1994).

RAJIVE L. BAGRODIA received the B.Tech. degree in Electrical Engineering from the Indian Institute of Technology, Bombay in 1981 and the M.A. and Ph.D. degrees in Computer Science from the University of Texas at Austin in 1983 and 1987, respectively. He is currently an Associate Professor in the Computer Science Dept. at UCLA. His research interests include parallel languages, parallel simulation, distributed algorithms, and software design methodologies. He was selected as a 1991 Presidential Young Investigator by NSF.
E-mail: rajive@cs.ucla.edu

LEONARD KLEINROCK received the B.S. degree in Electrical Engineering from the City College of New York in 1957 and the M.S.E.E. and Ph.D.E.E. degrees from the Massachusetts Institute of Technology in 1959 and 1963, respectively. He is currently a Professor in the Computer Science Department at UCLA. His research interests focus on performance evaluation and design of many kinds of networks (e.g., packet switching networks, packet radio networks, local area networks, metropolitan area networks broadband and gigabit networks) and of parallel and distributed systems. Dr. Kleinrock is a member of the National Academy of Engineering, a Guggenheim Fellow, and an IEEE Fellow.
E-mail: lk@cs.ucla.edu

JOEL E. SHORT received the B.S. degree in Computer Science from California State University, Chico in 1992 and the M.S. degree in Computer Network Modeling and Analysis from the University of California, Los Angeles in 1995. He is currently a Ph.D. candidate in the Computer Science Department at UCLA studying wireless and nomadic system simulation and implementation.
E-mail: jshort@cs.ucla.edu