Mobile Wireless Network System Simulation

Joel Short, Rajive Bagrodia, Leonard Kleinrock
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90024

Abstract

In this paper, we describe a 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. This paper describes the simulation language, Maisie, which is used to analyze the performance bottlenecks of a multimedia wireless network system being developed at UCLA. By modeling the various components and their integration, this simulation environment is able to accurately predict the performance bottlenecks of the UCLA wireless multimedia networking terminal, determine the trade-off point between the various bottlenecks, and provide performance measurements which are not possible through experimentation and too complex for analysis.

1. Introduction

In order to develop mobile wireless network algorithms that support multimedia communications, the designer is presented with a host of design alternatives. The problems range from having to support different patterns of mobility to integrating the traffic generators, coders and decoders for voice, data, and video traffic, each with differing computational and communication resource requirements in the communication protocol.

A few operating system kernels have been designed to support wireless and mobile communication 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. Commercial radios designed to be hooked up with laptops for wireless multimedia transmissions are available in the market. Thus 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.

Traditionally, 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 a large range of 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 such an environment.

The rest of the paper is organized as follows:
We begin with a description of the primary operating system, network, and hardware components of a mobile, wireless multimedia information system is section 2. Section 3 discusses the use of existing simulation environments to analyze the performance of such systems. Section 4 describes the new simulation environment. The environment is built using an existing simulation language called Maisie that is also described briefly in this section. Section 5 presents the results of a simulation study to evaluate the performance of a specific mobile, wireless system that is being designed at UCLA. Experiments to validate the simulation are also presented. In section 6 we conclude with ongoing research to evaluate other components of the proposed multimedia wireless network.

2. Mobile Wireless System Implementation

In this section we describe the node architecture and the implementation of the network control functions, multimedia support, communication substrates, and the interfaces between these as shown in Figure 1 for the Wireless Adaptive Mobile Information System (WAMIS) research project at UCLA [12]. Applications are needed for interaction between the system and the user. Mul-

** 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.*

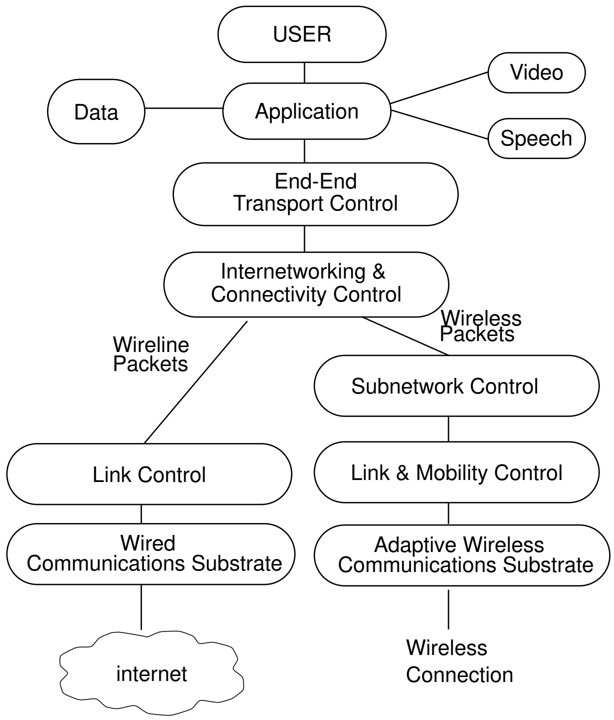


Figure 1: Node Functionality

timedia 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. The network control functions are responsible for control and interfacing between the multimedia devices and the communication substrate(s).

The networking algorithms, network operating system, and applications are implemented inside the laptop. The multimedia support (video and speech) is currently integrated into the system as ISA cards in a docking station. The docking station also contains the Packet Interface (PI) card used for connection to the radio. Technology is enabling the integration of these devices into the laptop via the PCMCIA interface. We are starting to see the emergence of camera and microphones being integrated into laptops today. Many commercially available radios and other network devices can connect directly to the laptop via a PCMCIA type interface and not require this additional interfacing hardware.

2.1 Network Control Functions

The network control functions, shown in Figure 1, comprise numerous software components which can be used to support self-configuring, multihop, multimedia networking. These components can be broken down into end-to-end transport control, internetworking and connectivity control, subnetwork control, and finally link and mobility control. In order to provide flexible implementation for rapid prototyping and easy modification for experimentation, these functions are written in C and supported by the operating system to isolate the architecture and implementation details from the network algorithms. Interfacing is needed not only between the hardware and software, but also between the various network functions. The network control algorithms and protocols are responsible for making sure the data between applications are transferred via the appropriate communication channel as efficiently as possible in support of the desired quality of service. The integration of the various components fit into the standard OSI/ISO type model as illustrated in Figure 2.

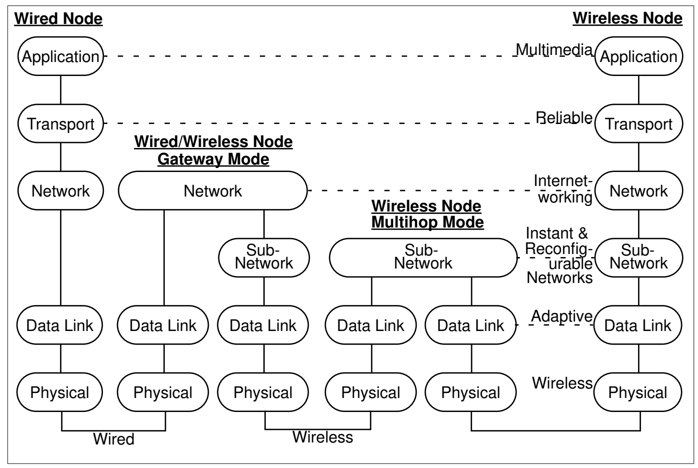


Figure 2: Network Control Functions

2.1.1 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 (Mosaic and video conferencing applications). In order see the effect and demonstrate multimedia on this new self configuring, multihop, multimedia network, a video conferencing application was developed. This application (VideoTALK) brings together video, which uses UDP, and data, which uses TCP, into a single application on the laptop. In order to test the performance of the system, testing tools were developed to measure throughput, delay, packet loss, and track adaptive parameters 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 analyze the virtual topology of the wireless multihop subnet.

2.1.2 Transport and Internetworking Control

Since internetworking requires compatibility with existing networks and protocols, 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, most of the new 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.

Although wireless communication is useful to support mobile communication, typically wired connections can support much higher bandwidth and are less prone to errors in the wireless modems. Therefore, wired connections should be utilized whenever possible. Wired connections, such as ethernet, can utilize the standard PCMCIA card for networking. In order to support a combination of wired and wireless communication, provide wireless multihop functionality, and support instant infrastructure networking, a node should be able to function in three different modes (gateway, multihop, or end node) as shown in Figure 2. A node functions as a gateway when 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 sub-network routing protocol to provide wireless multihop communication within the subnet.

2.1.3 Sub-network Control

The functionalities which support instant and reconfigurable networks are new and have been added into the TCP/IP stack (Figure 3). Since 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.

2.1.4 Link Layer Control

The Data Link layer shown in Figure 2 supports a new function unique to wireless networking for mobility control. Mobility control is used for setting appropriate hardware parameters such as the CDMA code or transmit power level. Measurements such as Signal to Interference Ratio (SIR) are fed back from the radio into the link control algorithms to do power control and minimize the power consumption of the link, reduce interference, and provide admission control such as described in [2].

2.2 Operating System

To integrate all these network control components together, an operating system is desired such as we see in Figure 3. There are numerous operating systems available today such as Microsoft Windows, PC-Disk Operating System, and UNIX, but these systems aren't designed for ease of programmability or flexibility for the types of functionality described above. An operating system is desired which is compatible with existing platforms today but still provide functionality such as multi-tasking and packet processing capability useful to network control algorithms. 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) designed to support packet radio meets the flexibility

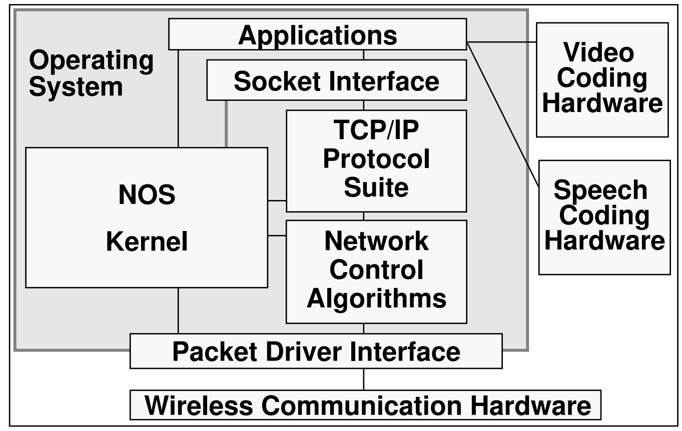


Figure 3 Wireless Adaptive Mobile Information System Network Operating System Components

requirements[4]. 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 interprocess (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 current implementation, uses a NEC Versa 486 33Mhz laptop and docking station. The WAMIS Network Operating System is able to run on any laptop as long as it supports DOS and the required interface cards. The Packet Interface (PI) card is used as the network interface card to integrate the wireless modem into the system. In order to provide a standard interface to the network operating system, a packet driver interface is used. The packet driver interface is based upon FTP's packet driver specification. This interface allows various network interface cards (like the PI card or a PCMCIA card) to be used in place of one another without having to change the details of the network operating system in order 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.

2.3 Multimedia Support

To support various multimedia aspects with regard to wireless communication substrates, a full frame wavelet transform based technique has been developed by members of the UCLA WAMIS project [14] [15] and is used instead of the common wireline techniques such as motion compensation, and block based compression. The video subsystem is built on top of a Data Translation (DT) card which is responsible for the input (digitization) from the camera. It is able to digitize up to 30 frames per second of 256 greyscale video. This card has a piggyback ability so a custom card can be added on. This is used to integrate the video coding which exists of a Xilinx FPGA. The current FPGA implementation is able to operate around 8 frames per second but is capable of operating around 10 frames per second if a faster clock (crystal) is used. For Speech coding, an I/O processing card with programmable coding is desired for speed and flexibility. A programmable processor (DSP) implementation of a speech coding algorithm performs the adaptive speech coding.

2.4 Radio Modem

The radio is a direct sequence spread spectrum radio designed and implemented at UCLA [8][9]. This radio was designed using CAD tools (VANDA) in order to allow adaptive design of the radio. The radio is currently able to operate at speeds from 7 to 32 Kbps depending on the spreading factor desired. A unique ability of this radio is the control of various 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-off. It should take anywhere from 500

chips /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

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, then we are able to achieve the various data rates as described above. The reason one would not always necessarily want to use the fastest data rate is that the lower the spreading, the less resilience to noise and interference. By using more chips/bit (slower data rate) we are able to have more capacity of the network and less interference. It is up to the network control algorithms to dynamically determine what these parameters should be set at for optimum network efficiency.

Numerous wireless radio modems are commercially available, such as Proxim's RangeLAN2, but they lack the ability to dynamically adapt to various power levels and codes. They also do not provide accurate measurements such as SIR, necessitating the use of the custom radio that has been developed at UCLA.

3. Commercial Simulators

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.

OPNET by Mil3 is a useful commercially available network simulators that was examined as a potential for developing a simulation and implementation environment for the wireless network algorithms. It follows a 3 level model (Network, Node, and Algorithm) and has a nice graphical user interface where algorithms can be described using Finite State Machines. There are, however, several pitfalls with OPNET that prevented it from being the network simulation environment of choice.

As far as mobility is concerned, we can specify the path a node will follow and how fast it traverses that path, but OPNET does not allow the nodes to move in a random fashion. The channel model follows a rigid 14 stage pipeline and requires significant understanding of the OPNET kernel in order to modify or enhance the channel model. It currently takes into consideration the distance, attenuation, and collisions of packets from interfering transmissions. The primary problems that prevent us from using OPNET for this project are its limited support for node mobility patterns, restricted channel models, rigid modeling methodology, and long execution times for even moderate sized models.

As far as the node layer layout/simulation of OPNET, it has the ability to graphically connect various algorithm or protocol modules together as seen in Figure 4. After the simulation has run,

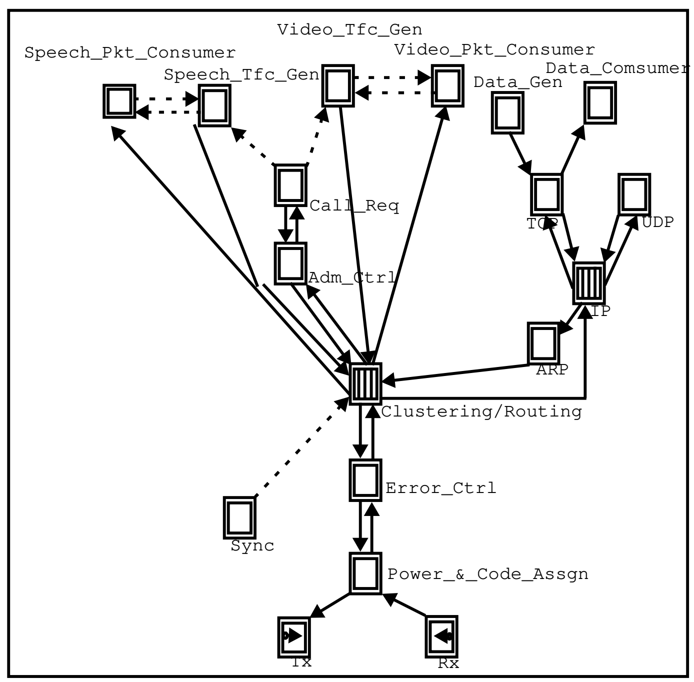


Figure 4: OPNET Node Level Layout Simulation

you can watch an animation of the flow of packets across the specified links to see packets (solid lines) or messages (dashed lines) flowing from one module to another. The problem with the node layer layout in OPNET is its rigid modeling methodology.

Only sequential model execution is supported and mixed domain simulation is impossible with various components that make up the node. All models must be put into a discrete event simulation that fits into this specified layout. Moreover, and importantly, once the algorithm and protocol stack is working in OPNET, there is no path to implementation[1]. The Finite State Machines and C code which represent the algorithm or protocol must be re-coded and that can easily introduce errors and inconsistencies which weren't modeled in the simulation. Implementation can be a very time consuming and non-productive process even when the simulation works perfectly.

4. Simulation Environment

We have designed a general purpose, parallel environment for the simulation and implementation of network algorithms. The 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 parallel simulation language called Maisie. The Maisie simulation environment has been implemented on a variety of workstations, on networks of workstations and on distributed memory multicomputers like the IBM SP1 and on a shared memory Sparc 1000.

4.1 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* with time stamp *clock*()+*t*, where *clock* is the current value of the simulation clock, in the message buffer of entity $m1$.

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 the constant *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 requests are serviced in first-fit manner). The second resume statement accepts a *free* message:

```
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

1. The simulation environment we have developed allows a direct path from simulation into implementation without modification of the algorithm; however, this is not the focus of this paper.

hold(t) will suspend the corresponding entity for t units in simulation time.

4.2 Mobile System Modeling Environment

The modeling environment is designed to allow the primary components of the wireless network system to be simulated at different levels of details. 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.

Our model of the mobile, wireless network system consists of the following primary components:

- Operating System Models (OSM)
- Application-specific traffic models (SOURCEM)
- Network Algorithm Models (NAM)
- Wireless Radio Models (RFM)
- Channel Models (CHM)
- Mobility Models (MOM)

The OSM simulates the relevant portion of the WAMIS Network Operating System (WAMISNOS) kernel that is involved in interfacing with the application (e.g. delivery of incoming messages) or with the network (e.g. transmission of a remote message). 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, the 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, subnetwork control such as clustering, and mobility control such as power control, logical link control, and media access control. 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.

These components can be viewed as fitting in with the OSI Reference Model as shown in Figure 5.

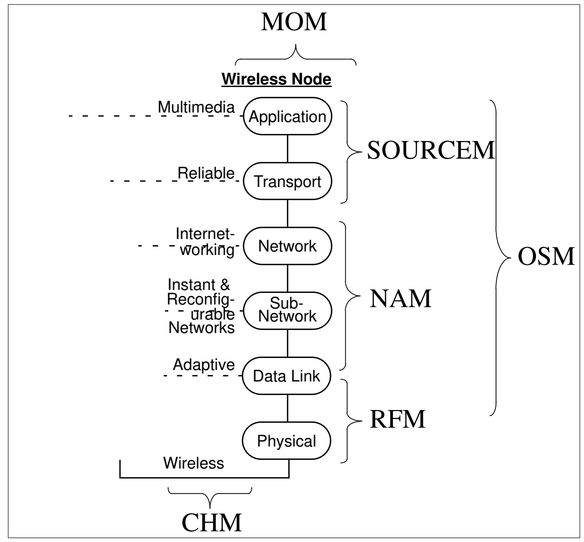


Figure 5: OSI Network View

4.2.1 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 interprocess 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_o, 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}
int id;
ename pktdrvr;
ename ipalgptr;
{
  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;
         newlen=pkt.len-HEADER_SIZE;

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

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

         remtime=endtime-clock();
         hold(ti);
       }
       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. 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.

4.2.2 SOURCEM

The SOURCE Models are composed of 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 analysts 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

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

Table 2: SOURCEM Characteristics

application, the transport protocol that is commonly used, typical packet size, traffic type, and metric to be optimized is listed.

4.2.3 NAM

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

- Network Layer
- Sub-Network Layer
- Data Link Layer

The network layer components include the internetworking functionality shown in Figure 1. 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 [11] 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 */
    /* Info (1) = Not CH;    Info(2) = CH */
    invoke pktdrvr with
           broadcast{id,I_am_ch[id]+1};
  }
}
```

The algorithm works by first clearing out the of table everyone who it can talk to. Then all nodes broadcast a message to inform everyone else who their neighbors are. Starting with the lowest possible ID, with the lowest ID and assign those nodes as clusterheads where each next highest node can only be a clusterhead if none of it's neighbors that it received a broadcast packet from is not a clusterhead. This is done iteratively until all nodes know if they are a clusterhead or not. Each node also knows who its neighbors are. Each node does not have any global knowledge of the topology.

The data link layer models are used to provide mobility and link level control such as power control [6] [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 [13].

4.2.4 MOM

The MOM components are responsible for:

- 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 nodes 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 nodes 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 is able to select a random step size (no greater then the maximum moving range) which it is able to move within. Once the new position is selected and forced to remain within the space (grid) of the simulation, it's new position is updated.

In the following Maisie fragment, we see how the speed of a mobile (MOVING_RANGE) can be modeled with a semi-random direction.

```
  or mtype(move)
  {
    id=msg.move.id;
    position[id].x = position[id].x-
(int)lrand48()%(MOVING_RANGE*2+1)+MOVING_RANGE;
    position[id].y = position[id].y-
(int)lrand48()%(MOVING_RANGE*2+1)+MOVING_RANGE;
    if(position[id].x<0) position[id].x=0;
    if(position[id].y<0) position[id].y=0;
    if(position[id].x>max_x)
      position[id].x=max_x;
    if(position[id].y>max_y)
      position[id].y=max_y;
  }
```

4.2.5 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:

- 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.

In the following Maisie fragment, we see how the channel model is able to determine which nodes a *broadcast* packet should be received at. The actual packet (message) is sent to the appropriate node via the **invoke** statement.

```
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)(posi-
tion[b.id].y-position[i].y), 2.0)) < (double)
COMM_RANGE)
    invoke pktdrvr[i] with pktin{b.id,b.info};
}
```

When a packet is to be transmitted, the time lapse from when the receiver gets the packet is scheduled in the channel model by the hold routine in Maisie.

```
hold(TXTIME);
```

The transmit time (*TXTIME*) is actually determined by the RFM.

4.2.6 RFM

The RFM Module is responsible for the data link and physical layer modeling of the radio frequency modem which includes 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 algorithm senses for carrier and defers if someone else is transmitting on the channel and provides collision avoidance by following a request and acknowledgment scheme between source and destination to reserve the channel for transmission. 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 acquisition time which is described later.

Other link level control algorithms such as CRC checking, preamble, bit stuffing, etc. can either be modeled if one is concerned about the details of the bits being transmitted or include this overhead by adding to the packet size and holding the RFM from being able to transmit for the period of time it would take to do such link level control processing.

The NIC bandwidth can be modeled in several ways. The actual interfacing and node architecture can be modeled and limits in bus transfer time, packet process time, etc. can be included. The other option is to measure with experimentation what the characteristics are of the NIC card and add those delays to the overall throughput of the RFM. These delays are included as part of the tail time processing for each packet. The tail time is also used to make sure the clocking in of the data stays active long enough for the whole packet to be received between the radio and CPU.

The raw bandwidth is responsible for determining 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 in bytes, we can use the data rate (in bits/byte) to determine how long it will take for the packet to be transmitted through the wireless channel.

The acquisition time and thus the amount of preamble for a packet should be the maximum amount of time it takes for the Direct Sequence Spread Spectrum (DSSS) modem receiver to lock onto the transmitted signal. Depending upon the acquisition loop used in the radio and various environmental effects this time can very from radio to radio.

For the UCLA radio described, the RFM parameters include 50ms for acquisition of each packet, 10ms for tail processing 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:

$$TXTIME = AcqTime + \frac{PktSize \bullet 8}{DataRate} + TailTime$$

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.

5. Results

Admittedly these are very elementrary models for the very powerful simulation environment described, but allows us to illustrate the interaction of the various models in the simulation environment. 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.

Actual measurements were done using 2 486-based laptops hooked up with the UCLA designed radios (described earlier) running WAMISNOS to provide a point to point wireless link. WAMISNOS is a customized network operating system which includes the complete TCP/IP protocol suite, 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. A file transfer (1.5Megabytes) was done using the FTP application which uses TCP. The TCP protocol provides a reliable error free connection oriented transport service to the application. The application and protocols used are the same ones used throughout the Internet on various systems, however customization of the TCP parameters was done to maximize the possible efficiency. Most TCP/IP implementations do not allow customization to maximize performance over a given link since the TCP/IP protocols are designed for use over a collection of different networks (links with varying qualities, bandwidth, and delays).

5.1 Simulation Models

We have developed several simple modules in this simulation environment to model the functionality and performance of the various components including the 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).

5.1.1 OSM

In order to model the performance of the WAMIS Network Operating System running on the 486 laptop, experimentation was done to find out the average processing time for incoming and outgoing packets. In section 5.3.4 we'll 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.

5.1.2 SOURCEM & NAM

The modeling of the File Transfer application and TCP protocol are done in the SOURCEM module as we saw in section 4.2.2 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, the question which this paper addresses is where the remaining bottlenecks are

5.1.3 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 32,258bits/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 as follows:

Description	Value
Raw Channel Rate	32,258bits/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.

5.2 Comparison of Results

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

	Simulation	Experimentation
Time	823023	942710
Bytes Received	1751560	1751560
Packets In	444	589
Packets Out	443	569

Table 5: Simulation & Experimentation Comparison

find 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 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.

5.3 Performance Bottleneck 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, etc.)	3.4

Table 6: Performance Breakdown

5.3.1 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%. This can be verified by the following calculation:

$$\frac{\frac{FileSize}{ChannelRate} \times \frac{Bits}{Byte}}{TotalTime} = Efficiency$$

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.).

5.3.2 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 formula:

$$\frac{TotalNumPkts\,(Tx + Rx) \times \frac{AcqTime\,(ms)}{Pkt}}{TotalTime\,(ms)} = AcqOverHead$$

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 weren't 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.

5.3.3 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. Tthe 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 algorithm 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%.

5.3.4 OSM Processing

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

Different CPU Processing is done when a packet is either transmitted or received at a node.

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.

5.3.5 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 20.4seconds (2.2%). Current research is being done in the Internet Engineering Task Force to come up with a new set of TCP and IP protocols which would remove some of the unnecessary information out of the headers. One such protocol, SIPP (Simple Internet Protocol Plus), would reduce the number of unused bytes in the header but double the number of usable IP addresses (64 bits rather then 32.) The unused bytes were introduced since when the standard was developed, there wasn't much experience to realize which fields were important and which weren't.

5.3.6 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.

5.4 Extending Analysis through Simulation

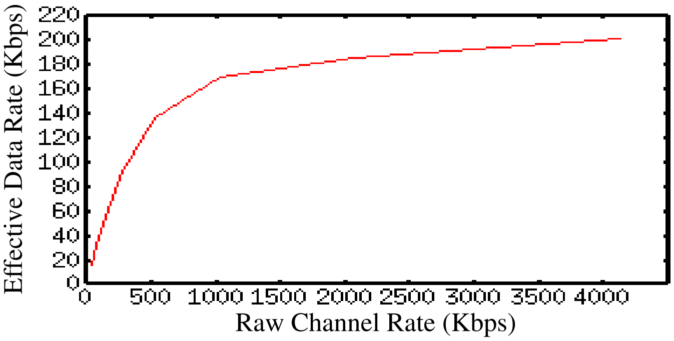
We found that through customization of TCP parameters, we were able to achieve a link efficiency of 46% (14.8Kbps) using UCLA's Radio with WAMISNOS. As shown in Table 7, the three

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

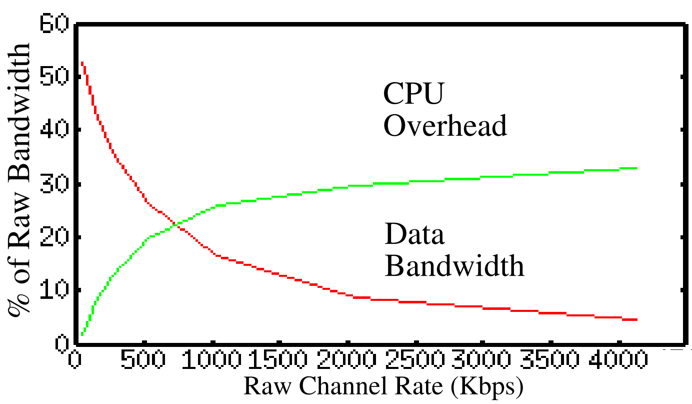
largest bottlenecks in this system are 1) the raw channel rate, 2) acquisition delays, and 3) time-outs in TCP caused by bit errors and packet losses in the link. We also see that the simulation environment breakdown is very similar to that found through experimentation.

As technology advancement will allow higher channel rates, other system components will impact the effective data rate. By using this simulation environment to test parameter space which is not possible with current technology, we can see in Graph 1, the trade-off between the effective data rate improvement as the raw channel rate increases.



Graph 1: Effective Data Rate vs. Raw Channel Rate

Various system bottlenecks are preventing the linear growth of the effective data rate as a function of the raw cha1nnel rate. In Graph 2, we see how the CPU Overhead becomes a dominant

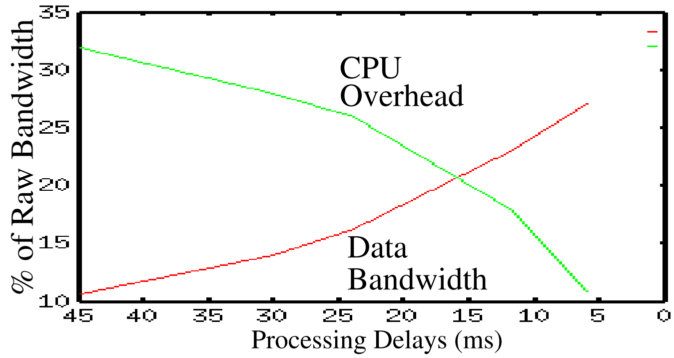


Graph 2: Bottleneck vs. Channel Rate

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.

As technology will also advance the CPU performance, we can see in Graph 3 the trade-off 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 1Mbps. When the CPU processing delays fall below 17ms per packet, the channel rate starts becoming the larger bottleneck.

6. Conclusion



Graph 3: Bottleneck vs. Processing Delay

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 such as system: network operating system, traffic models, protocol models of the network, data, and physical link levels, radio models, and mobility patterns. 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. A companion paper submitted to this conference [Gerla, Tsai, Wu] used the protocol and mobility models to evaluate the performance of a clusterhead election algorithm as a function of channel characteristics. We plan to integrate the two preceding models to study the performance of the protocols as a function of the radio and NOS characteristics.

The simulation environment was designed using an existing simulation language call Maisie [2]. Maisie has been implemented on both sequential and parallel architectures including distributed memory machines like the IBM SP2 and shared-memory multiprocessor machines like the Sun Sparc 1000. The experiments reported in this study used only the sequential Maisie implementations. Parallel Maisie implementations have yielded significant performance improvements for a number of applications that include circuit-simulation [3], queueing networks [1], and data parallel programs. We intend to explore the viability of the parallel implementation in improving the performance of simulation models for wireless networks like those described in this paper.

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. And everyone in the WAMIS project for their contribution in development of the implemented wireless multimedia system.

8. References

[1] R. Bagrodia, K. M. Chandy, and W-L. Liao, “An Experimental Study on the Performance of the Space-Time Algorithm”, *Workshop on Parallel and Distributed Simulation*, Jan. 1992, pp. 158-168.

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

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

[4] W. Boring, J. Short, "UCLA WAMISNOS Network Protocol Programmers Guide", UCLA Comp. Sci. Dept. Technical Report, 950010, April 1995.

[5] S. Chen, N. Bambos, G. Pottie. "Admission control schemes for wireless communication networks with adjustable transmitter powers" *INFOCOM 94*, Toronto, Canada. IEEE 1994.

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

[7] S. Chen, N. Bambos, and G. Pottie, "On Power Control with Active Link Quality Protection in Wireless Communications Networks," *IEEE Proceedings 28th Annual Conference on Information Sciences and Systems.*

[8] C. Chien 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] B. Chung, C. Chien, H. Samueli, and R. Jain, "Performance Analysis of an All-Digital BPSK Direct-Sequence Spread-Spectrum IF Receiver Architecture", *IEEE Journal on Selected Areas in Communications*, Vol. 11, No. 7, September 1993.

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

[11] IP Mobility Working Group, "Routing Support for IP Mobile Hosts", Internet Engineering Task Force, Internet Draft, 1994.

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

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

[14] B. Schoner, J. Villasenor, S. Molloy, and R. Jain, "Techniques for FPGA Implementation of Video Compression Systems," *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays 1995,* Monterey, California, 1995.

[15] J. Villasenor, B. Belzer, and J. Liao, "Wavelet Filter Evaluation for Efficient Image Compression," accepted for publication in *IEEE Transactions on Image Processing*, August, 1995.