Seawind: a Wireless Network Emulator

Markku Kojo, Andrei Gurtov, Jukka Manner,
Pasi Sarolahti, Timo Alanko, and Kimmo Raatikainen
University of Helsinki, Department of Computer Science
P.O.Box 26, FIN-00014 UNIVERSITY OF HELSINKI, Finland
{markku.kojo,andrei.gurtov,jukka.manner,pasi.sarolahti,
timo.alanko,kimmo.raatikainen}@cs.helsinki.fi

Abstract

Behavior of current communication protocols as well as current and future networked applications is of fundamental importance for technical and commercial success of Mobile Internet. The forthcoming wireless Wide-Area Networks, such as GPRS and UMTS, are quite complex and network operators have a large set of parameters to tune the transfer performance of these networks. In this situation it is of great value to be able to execute practical experiments. The Seawind emulation software introduced in this paper enables measurements of protocol implementations in modeled networking environments. The Seawind software provides a rich set of ways to define transfer characteristics including delays and errors. The software has also means to conduct large sets of experiments in an automatic fashion. In addition, tools of analyzing measurement data has been integrated into the Seawind software.

1 Introduction

Nowadays Wireless Wide Area Networks (WWAN) are widely used by mobile users to access data services. New mobile data networks, as for example the General Packet Radio Service (GPRS) [6, 8], and future third generation mobile communication systems [32] are expected to provide a high-speed packet data access suitable for a wide range of Internet services. However, wireless links represent a different communication environment than the wireline Internet. Hence, protocols and applications not particularly designed for wireless links often require enhancements in order to achieve reasonable performance in a wireless environment [4, 17].

Evaluating such enhancements over a real data link or network is often costly; if a system is only in a development stage, the evaluation may be impossible. A frequently asked question is whether next-generation wireless networks could provide multimedia services that meet the end-user expectations. Network emulation is a convenient tool to examine how existing multimedia applications behave. An emulator can also be used in usability studies involving real end-users. The main difference to a field trial is that an existing network is replaced by a model describing characteristics of transfer, delay and error behavior, for example. An emulation also allows controlling the network characteristics and reproduce the environment. On the other hand, the problems of emulation include the accuracy of the model; parameters drawn from real-world phenomena and properties are always estimates.

In this paper we describe the Seawind emulator and present a case study that demonstrates its practical utility. The primary target of Seawind is performance studies of real protocols and applications as seen by the end user in the present and future wireless networks. Although Seawind was developed for modeling wireless networks, GPRS in the first hand, the Seawind emulator is rather generic and it can be used in modeling a wide range of networks.

Emulation is a compromise between two other possible approaches in performance evaluation; between simulation and measurements using a testbed [2]. The main advantage of a network emulator is that the performance of actual implementations of protocols and applications can be examined. This is a clear advantage, for example, over most of the TCP performance studies that rely on the abstract TCP model available in the NS simulator [14]. However, most end users – if not all – connected to the Internet use TCP implementations that are not necessarily close to the one found in the NS simulator (for example, those in Windows or Linux). Therefore, the NS simulator or other simulators having their own TCP implementations do not allow a network operator to tune the networking parameters so that the performance is optimized for real end users and their applications. Furthermore, a real-time emulator provides answers to ”what-if” type of questions. It also allows back-engineering parameters of closed networking implementations.

Emulation studies can be time-consuming because the duration of an experimental session is determined by the speed of the modeled network. In order to enhance the usability of the emulator, the Seawind emulator supports an automatic set up of tests and collection of a sufficient number of test repetitions for statistically valid results. Therefore, the experiments can be run overnight and during weekends without any human intervention. The Seawind package also provides basic tools for statistical analysis and for graphical presentation of results. We use Seawind on the Linux operating system. The Seawind software runs in the user space and can be easily ported to any Unix operating system.

Several extensive performance studies have been made using Seawind. TCP performance has been studied in [13, 25] and GPRS performance in [18]. Seawind was also used in the Monads demonstration at MOBICOM 2000.

The rest of the paper is organized as follows. After a brief summary of related work, we discuss, in Section 2, common characteristics of wireless links. We also derive the requirements for a network emulator taking those features into account. In Section 3 we describe the Seawind architecture. We present the structure of the Seawind simulation Process that is the core of the Seawind emulator. In Section 4 we discuss the features of Seawind that are important in emulation of any wireless network. In Section 5 we present how we validated the Seawind emulator. Finally, a case study is described in Section 6 in order to illustrate the practical value of Seawind.

2 Related Work

Simulation of communication networks is an active research area. A wealth of different simulators are found worldwide; most of them are freely available while some are commercial products. The software tools for network simulations can be divided into two categories: discrete event simulators and real-time simulator or emulators, as we call them. Many simulation tools are discrete event simulators that operate in virtual time. These simulators have their own abstract implementations for modeling different links, protocols, and even applications. Probably the most well-known discrete event simulators are NS [14] and the commercial simulator

OPNET [31]. The Monarch Project at Carnegie Mellow University has created a set of wireless and mobile extensions to NS that provide a more detailed model of the physical and link layer behavior of a wireless network and allow arbitrary movement of nodes within the network [7]. Other network simulation tools include MobSim [27], SWimNet [5] and GloMoSim [33]. A parallel environment for the simulation of mobile wireless network systems, based on the parallel simulation language Maisie [3], has been presented in [26].

Discrete event simulators are great tools for overall network performance simulations and other more theoretical testing. These cannot, however, be used with actual protocol implementations and applications, unless the implementations are ported to the simulation package. During a product implementation and test, intermediate versions of the software emerge from time to time, and it would not be feasible to port each version to the simulation environment for testing. In addition, simulations cannot give a real-time view of how a user would experience some service using a new application, protocol, or network. Therefore, real-time tests and actual protocol implementation studies require a real-time simulator.

Real-time simulators or emulators allow researchers to create network topologies and conditions, which are difficult to achieve in a reproducible manner on production networks, or to perform real-time tests with various prototype protocols and products, for example. Such an emulator environment is well controlled and reproducible.

The common nominator within the emulators available in this category is that they provide different delay, packet drop, and queue-handling functionality in order to simulate some communication medium or network. NIST Net [11] is implemented as a kernel module extension to the Linux operating system and an X Window System-based user interface application. NIST Net provides parameters such as delay, packet drop probability, fraction of duplicated packets, and link bandwidth. Dummynet [24] is a similar tool, implemented as a FreeBSD Unix kernel patch in the IP stack. Dummynet works by intercepting packets on their way through the protocol stack; it uses parameters similar to the ones in NIST Net to affect the flow of packets. A third similar kind of emulator is the Ohio Network Emulator, ONE [1]. ONE uses three parameters to simulate a communications network, namely a transmission delay, a propagation delay, and packet queues.

The functionality offered in these emulators enable the simulation of a variety of different links, networks, and protocols. However, parameters such as delay, bandwidth, and queue sizes are not enough for all simulation purposes. A key functionality that is missing in the above emulators is the lack of timed events and changes of the simulated network environments. Especially in wireless networks, the network characteristics can change drastically due to the movement of mobile terminals and even the present weather conditions. To expand the area of studies that can be performed with an emulator, changes in the simulated environment and other timed events, such as handovers, should be provided by an emulator. In addition, the portability of the emulator to other platforms is more complicated in the emulators mentioned above since those have been tightly coupled with specific operating system kernels.

3 Wireless Network Characteristics

In this section we present a summary of properties of wireless links that present challenges for efficient data communication. Wireless links typically have relatively low bandwidths, high latency and high error rates. We discuss how these properties relate to the requirements for the network emulator.

Slow, asymmetric and changing line rate. The line rate of a wireless WAN link does not often exceed some tens of kilobits per second. Such a link speed is typical also for dial-up modem users. For some wireless links, the line rate can vary over time, due to a change in the amount of radio resources assigned to the user or change of the channel encoding scheme. The line rates may be asymmetric, for example when using certain types of satellite links or GPRS. Thus, the emulator should provide the desired line rate by delaying data packets and provide means to emulate changes in the line rate, in both directions independently. For the majority of W-WANs a rate up to 100 kbps is sufficient. However, modeling future broadband wireless networks will require line rates at least up to 2 Mbps.

High latency and variable delays. The propagation delay of wireless links is typically high. The delay comes from the special transmission schemas on the wireless link and from the processing delays of the link hardware. For example, the Global System for Mobile Communications (GSM) uses interleaving of data on the radio link to reduce the effect of error bursts, and this introduces a latency of 90 ms independent of packet size [20]. Additional latency in using a GSM data service is caused by the connection to the Internet Service Provider (ISP) and the processing time within the GSM system. The total one-way latency in GSM sums up to 200-300 ms[1]. The emulator should correctly model this delay by adding a propagation delay to each packet. Variable delays may appear on a wireless link due to a number of reasons, for example Link-level ARQ recovery, radio resource (re-)allocation and handovers to mention a few. There should be a possibility to add such random delays to a packet flow.

Error losses. Some wireless links impose a significant amount of data corruption due to transmission errors. The error rate depends on the current radio conditions and the strength of the channel coding schema. For example, in the transparent GSM data service the residual bit error rate (BER) of the link is allowed to be as high as 10^{-3} after the Forward Error Correction (FEC) [21]. Radio conditions can vary greatly. In the ideal conditions all protocol data units (PDU) are delivered correctly, and in the worst case nothing can be correctly sent over the link. For accuracy of emulations and ease of use the emulator should be able to drop packets on a per-packet basis or using a bit error probability. The transmission error on the link can be seen by the upper layers as a delay in data delivery (reliable link level), loss of a PDU (error detection in link layer) or as a corrupted data packet (transparent link layer). The emulator should provide all three cases.

Congestion losses at the bottleneck queue, over-buffering. The wireless link is often the bottleneck in the path of a data flow, because fixed networks are fast and reliable compared to the capabilities of the wireless link. Routers play a significant role because congestion data losses are most likely to happen at the bottleneck queue. A limited number of buffers can be allocated in a last-hop router per user. The emulator should contain a queue at the emulated bottleneck link and provide means to limit the queue size in terms of bytes and number of packets. Optionally, a timer would be used to limit the time a packet can be buffered. New queue management algorithms and drop policies should be easily attached.

[1]Note, that we do not include the transmission delay into the link latency. Thus the *round-trip time* is defined as the sum of transmission and propagation delays in both directions.

Handovers. In a cellular radio network the mobility is accomplished by changing the access point that serves the user, according to the user's current location. The handover process may cause data losses and a drastic change in the service provided, when the user moves from a less busy to a more occupied service area. Modeling a handover would cause a change in a number of simulation parameters at once. For example, in a packet radio network, a new service area may have more users using the same shared medium than in the previous location; the user will notice this as less available bandwidth after the handover. In addition, the handover process itself may cause a delay or loss in data delivery. The emulator should be able to allow changing a set of parameters at the same point of time to emulate changes in network service.

Link blackouts. Wireless links are prone to temporal interruptions of service. A typical example is loss of radio coverage; this might happen due to driving in a tunnel or moving away from the serving access point. Blackouts cause a situation when, for a period of time, no user data is successfully transmitted through the link. If QoS support is implemented in the wireless network, higher priority packet traffic can as well cause blocking of lower priority traffic. The emulator should provide means to specify the timing of blackout periods and the handling of PDUs during that time, for example, whether to drop or store the PDUs during the blackout.

Finally, a number of features would enhance the usability of the emulator. It is desirable that the emulator could be running in user space on an unmodified operating system. Binding the emulator in the operating system kernel is not desirable because it complicates the portability of the emulator between different operating systems and even between different operating system versions. The emulator must provide accurate execution of all timed events and notify if some scheduled event slipped past a certain threshold value and would thus affect the result of the test run. The emulator should have an easy to use user interface to enable its use by a wide range of specialists unfamiliar with its implementation details.

4 Seawind Architecture

The fundamental architecture behind the Seawind emulator is shown in Figure 1. Seawind intercepts the traffic flow between the client and the server transparently to the endpoint hosts. The desired link characteristics are emulated by delaying, dropping and modifying packets in the flow. The socket API and the protocol implementations in the client and the server need not be changed. The client and server can be directly connected to Seawind (e.g. by a serial cable), or they can be located anywhere in the network. For example, the latter option is useful, if the researcher wants to experiment with a data transfer over an emulated wireless link to a server located in the global Internet.

We have been mostly interested in studying the TCP/IP protocols and the behavior of TCP-based applications. However, Seawind can be used with any data flow, for example with traffic produced by the WAP protocol [19]. This generic approach allows comparing the performance of different protocols (e.g. WAP and TCP) or different implementations of the same protocol (e.g. Windows TCP and Linux TCP) under the same emulated network characteristics.

4.1 Seawind components

Figure 2 illustrates the Seawind components, which are used in setting up the test runs and running successive performance tests with various parameters automatically. The core of the

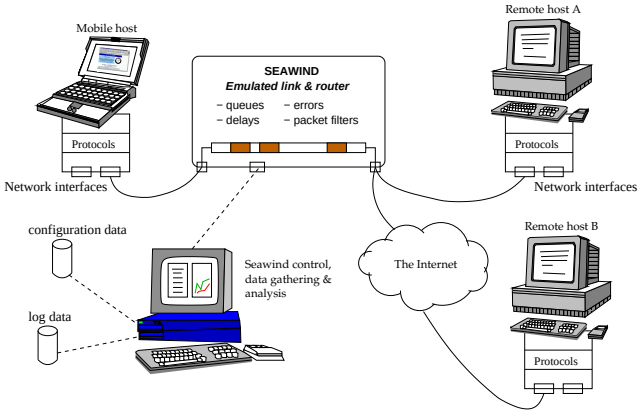

Figure 1: Structure of Seawind.

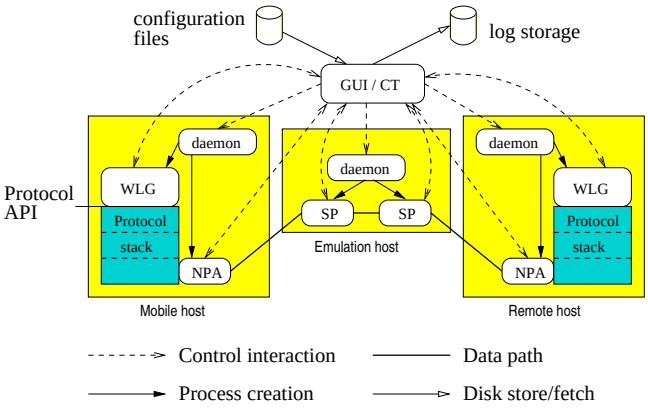

Figure 2: Architecture of the Seawind emulator.

emulation is the *Simulation process (SP)* that cause delays and packet losses to emulate the target link or the target network with various queues and buffers. Before describing the SP functionality in detail, we briefly introduce the other Seawind components shown in Figure 2.

The user sets up the tests using a *Control Tool (CT)* with a graphical user interface. With the control tool the user creates a number of entities called *replication sets*. A replication set defines the workload used in the performance tests and the parameters of the emulated network. For each replication set the user also gives the number of test run repetitions (replications) to be made with the given parameters. After the user has defined a sequence of replication sets to be tested, he may save the parameter settings for later use and start the test run with the given sequence of replication sets.

A replication set configuration consists of a *network configuration* and a *workload configuration* which are set up independently. Any combination of workload and network configurations can be selected to be repeatedly tested in a replication set. Workload configuration defines the tools that are used for generating the workload for the test and the parameters for the tool. Any external tool or script can be used as a workload generator. For example, the `ttcp` tool [30] can be used with Seawind to generate simple bulk data. Seawind also works in manual mode, in which the user may launch arbitrary, possibly interactive applications for generating the workload (e.g. a web browser and a http server), which communicate through the network emulated by Seawind. Network configuration consists of parameters defining the characteristics of the

emulated network and the *Network Protocol Adapter (NPA)* configuration, which we describe below.

In addition to the CT, Seawind uses a number of other processes to set up the tests. CT controls the creation and the cleanup of the processes in the beginning and in the end of each replication set. The processes may be distributed into multiple hosts to avoid having multiple resource-consuming processes running on the same host. To control the processes there is a *Seawind-daemon* process running on each host in the background. A Seawind-daemon creates the processes needed in a test run according to the requests from CT and terminates them after the test run.

The task of the NPA is to capture network packets from the endpoint host and forward the packets to the SP. For example, a NPA that captures IP traffic creates a dedicated network interface from which it captures packets, and adds a route to the created network interface in the IP routing table. There are various ways for doing this. One alternative is to use *Point-to-Point Protocol (PPP)* [28] that uses a pseudo-terminal device connected to NPA. Secondly, some operating systems support virtual network interfaces that deliver the received packets to and from the user-space applications. Finally, it is possible to capture packets directly from the Ethernet. We have implementations for all above-mentioned variants.

After the NPAs have been started and the simulation pipeline is properly initialized, Seawind starts the *workload generators (WLGs)* at both ends of the simulation pipeline. Workload generator can be any conventional tool that generates network traffic. No modifications need to be made, because the NPAs capture the network traffic transparently to the WLGs. For example, when IP traffic is used, the workload generator can be any tool that generates IP packets using the standard application interface (e.g. Berkeley sockets). The packets that are transmitted to the specified IP address are routed to the network interface connected to NPA and furthermore to SP. At the receiving end the NPA delivers the packet to the IP protocol using the created network interface.

The architecture presented above allows replacing any of the WLG, NPA or even SP components by alternative implementations. It is also possible to read the emulation data from a serial port, which makes it possible to connect an arbitrary machine to the emulation host using a null-modem cable. For example, we have used this facility to connect Windows hosts to Seawind. Furthermore, the receiving end NPA does not have to be attached to the endpoint host, but it can optionally be used to forward packets from SP to the network between the SP and the endpoint host. Thus, any Internet host can be used as an endpoint in the performance tests, making it possible to create a realistic model of communicating to a host in the Internet over the emulated link.

4.2 Pipelining

A mobile network typically includes several logical entities that affect the overall performance and throughput seen by the end user. For example, from the GPRS architecture [8] we can identify three possible emulated components: the base station subsystem (BSS) including the wireless link, the Serving GPRS Support Node which acts as a last-hop router in the GPRS network and the Gateway GPRS Support Node, which is the gateway router in the mobile terminal's home GPRS network.

As a single SP is often used to model a single network element with a link, to model a network path with multiple network elements, Seawind allows connecting several SPs together to form a simulation pipeline. Data packets are forwarded between SPs, and the last SP in the

pipeline sends the packet out to the destination.

Some links use flow control at the link level. This means that the rate at which the packets are transmitted from one network element to another is controlled by the receiving network element. Seawind supports flow control between SPs and between NPAs by using a sliding window based algorithm.

4.3 Channel model

While computer networks become increasingly complex, the principle of having a set of routers (switches, etc.) interconnected by communication links remains the same. In Seawind, a single Simulation Process models an outgoing link, optionally attached to a network node with buffering and a specified queue management policy. Several SPs can be pipelined to represent a path through the network between the client and the server.

The emulated link is modeled as a direction-specific *channel*, which is maintained separately for the uplink (towards the fixed end) and for the downlink (towards the mobile end) traffic. The downlink and uplink channels are largely independent, with an exception of some special events (for example a blackout). The model of the channel is shown in Figure 3. Packets arriving to the emulator are placed into the input queue. The maximum queue length can be limited in terms of bytes or packets. Different packet-drop policies can be applied on the queue (e.g. the traditional tail-drop or RED active queue management [9]). For example, the input queue can be used to model a queue in a router, and thus inspect the effect of congestion and congestion-based losses at a network node.

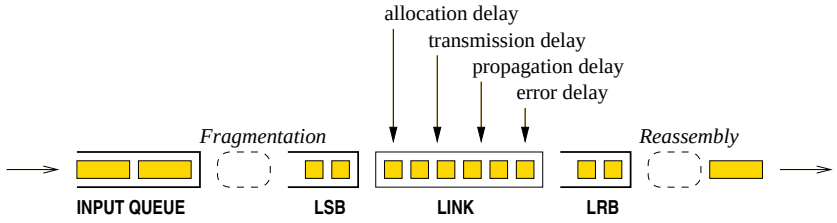

Figure 3: The channel model.

Some link protocols (e.g. Radio Link Control (RLC) [23] in GSM) fragment PDUs of the upper protocol layer as a part of internal operation. The fragmentation unit before the link and the reassembly unit after the link allow to logically fragment the data packet into smaller pieces for the purpose of the different events performed during the emulations. The actual size of the transmitted data transmitted by lower-level protocols can increase due to protocol overhead (e.g. added header) or decrease due to compression. This is also taken into account in the calculations.

The channel model also includes *Link Send Buffer (LSB)* and *Link Receive Buffer (LRB)* to model the send and receive buffers that are present with real links. The link send buffer is used to store the frames to be transmitted to the link, and the link receive buffer is used to store frames at the receiving end until all pieces of a fragmented unit have arrived, allowing reassembly. The link receive buffer is also needed to store out-of-order frames, when a link layer Automatic Repeat Request (ARQ) mechanism is used for retransmitting corrupted or missing frames. The size of these buffers should be large enough allowing the ARQ sliding window protocol to keep the link fully utilized. These buffers may significantly increase the buffering capacity of the link.

Before data can be delivered over the link, the radio resources often have to be allocated first. The delay can be rather high due to possible contention or even queueing for resources. In

the current model the *allocation delay* is triggered when a data unit arrives to an empty queue. Once the resources are allocated, data units are taken from the head of the queue one-by-one for "transmission" over the link. The length of the *transmission delay* is computed according to the line rate and the packet size. When the transmission delay for the data unit is completed, a *propagation delay* is issued for the unit.

Transmission errors on a link are modeled by specifying a probability that is evaluated on per-packet basis or on per-bit basis. If a data unit is corrupted, the following actions depend on the desired error mode. In the *corrupt* mode the data unit is forwarded in the channel with the corrupted bits. In the *drop* mode the corrupted data unit is dropped by Seawind (i.e. the link layer receiver is assumed to detect the transmission error). In the *delay* model the data unit is delayed for a user-specified amount of time. This can be used to emulate a link ARQ protocol retransmitting the corrupted data unit to recover from transmission errors. In such a case the upper protocol layers experience only an excessive delay for the affected packet.

5 Seawind Features

5.1 Emulation

Protocol filters. A protocol filter is a protocol-specific module that is implemented separately for each protocol (e.g. TCP, WAP) used with Seawind. New protocol filters can be easily added using the interface provided. A protocol filter has two functions: *packet recognizer* recognizes the packet boundaries from the incoming data and populates Seawind structures used in different emulation calculations. Usually the packet recognizer is based on the link layer protocol used for transmitting the Seawind packets (e.g. PPP used over an asynchronous link). Another function of the protocol filter is the *packet printer*, which scans the required information from the protocol headers and stores it to a log file. This information later allows combining the packet trace with the SP event log to determine various events, like the reason for a packet loss and to measure round-trip times. For example, when using TCP/IP protocols the packet outputter uses an output format which is compatible with the `tcpdump` [15] tool to allow interoperability with existing tools used for analysis.

State changes. The user can define multiple sets of parameters (states) that are changed during a test run. For example, the available bandwidth, error properties and delay properties can be changed simultaneously according to the given time interval distribution. This feature can be used to model changes in the mobile communication environment, e.g. due to handoffs. The state is changed synchronously at all SPs used in the emulation. Seawind also provides an interface to trigger state changes from an external program, thus providing a flexible way for creating a wide range of mobility scenarios.

Random distributions. A wide variety of random distributions are included in Seawind to model different kinds of network properties. The list includes the basic distributions (e.g. uniform, exponential, Cauchy) and a two-state Markov distribution. Additionally, any arbitrary distribution can be stored in a file to be used by different parameters during the emulation. Seawind uses its own random number generator to avoid being affected by the biased random number generators that some operating systems may have.

Parallel workload generators. Seawind allows using an arbitrary number of WLGs in automatically run tests and in manually executed tests. Starting the workload generating tools manually is straightforward, as user may launch any number of applications using the command shell, and Seawind transparently captures the data generated by the applications from the network device interface. In automatic operation the user may enter a number of WLG definitions with a starting time relative to the beginning of the test run. This makes it possible to inspect competing traffic flows over the network emulated by Seawind.

Multiple queues. The user data packets may belong to different priority classes. Multiple queues are present in SP to hold user packets of different priority and background load packets (Figure 4). A number of algorithms are given for each queue. The classifier algorithm assigns a specific queue to an arrived packet. The queue management algorithm (e.g. RED [9]) actively marks packets based on the averaged queue length. The drop policy algorithm (e.g. head drop) discards packets that exceed state queue size or length limits. Finally, packets can be marked using *Explicit Congestion Notification (ECN)* [10]. All algorithms have well-defined interfaces so new implementations can be easily added. We currently have a basic implementation of the single-priority traffic, but in the near future we will implement more classifier algorithms for handling multiple queues.

Background load. In addition to the main workload captured by the NPA components, Seawind provides a framework for generating virtual *background load (BGL)*, which affects the internal queueing and delay calculations of SP components. With this feature, the user may create a flexible model of the effects caused by other users in the emulated access network. Figure 4 illustrates the background load framework and how it can be used with multiple Seawind input queues. A *BGL generator* can be attached to any SP in the simulation pipeline to generate packets according to the defined model. Because the BGL packets exist only virtually, the information about BGL packets is forwarded to the next SP using a dedicated BGL channel. The BGL can be consumed by any SP in the pipeline, but it is not forwarded to the NPAs.

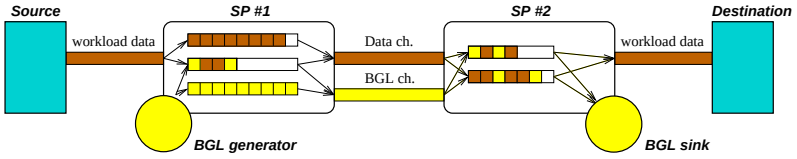

Figure 4: Background load generators with two pipelined SPs with multiple queues.

5.2 Output analysis

The CT collects the log output from various Seawind components and stores them on the disc for further analysis. Two kinds of logs are generated by the Seawind components. *Filter log* is generated by the NPAs and SPs. It contains information about the network packets that have traversed through Seawind. For example, when monitoring IP packets, Seawind uses `tcpdump` for this purpose. The filter log is created by the protocol filter described above and is protocol dependent. *Seawind log* contains Seawind-specific information of the test run. SP stores the information about the events such as delays or losses on each data unit. For each event a timestamp is stored to make it possible to verify that the events have been performed on time. Seawind

log is also collected from NPAs and WLGs. The contents of those Seawind logs depend on the mechanism used in NPA or on the tool used as WLG.

During a test run there are usually a large amount of log information generated. Therefore it is essential to have tools and scripts for analyzing the logs. For a filter log containing tcpdump-compatible information about IP packets this is easy, as there are a wide variety of publicly available tools such as `tcptrace` [22] available.

Currently our scripts collect information about various time measurements, throughput, number of retransmissions, number of packet losses and fairness (using Jain's fairness index [16]). Optionally, the information about different TCP variables in the Linux kernel, such as congestion window size or RTT/RTO estimates, can also be stored to be coupled with the other statistics. We also plan to enhance the graph plotting scripts to show various Seawind or Linux TCP events such as delays, packet losses and retransmission timeouts.

6 Implementation Issues

Developing a real-time emulator for an operating system and network environment that do not guarantee real-time response is not straightforward [12]. An off-the-shelf personal computer and Unix OS are not designed for real-time use, have coarse timer resolution, and are prone to delays caused by the I/O (a disk or network access). Especially in a multi-process environment, keeping a real-time schedule is hard, because processes have to compete for the system resources.

We have not set any absolute real-time requirement for the response times of Seawind events, as it would be impossible to guarantee the required response time in the general case. However, in this section we introduce how we try to ensure that the Seawind response times are accurate enough for performance tests and how we monitor the accuracy of emulation.

Simulation process. We have enhanced the timing accuracy of the events by waking up SP a configurable amount of milliseconds prior the event is due and wait in a busy loop until the actual event time is reached. Before running the performance tests, the user can take a few preliminary runs to adjust the timing estimator for his environment, if the default value is not good for some reason. By distributing the Seawind processes we wanted to make it possible to run the *simulation process* in a lightly loaded host in order to avoid competition of the system resources.

After each event Seawind takes a timestamp from the system clock and stores it to the log. If a threshold value given by the user is exceeded, a warning message is printed so that the user can discard the results for the particular test. However, if the timing estimator is correctly set and there are no other resource-consuming processes running, this occurs very rarely. In our experience the accuracy of Seawind remains within 1 ms with rare exceptions.

During the test runs Seawind avoids unnecessary I/O access, which could cause harmful context switches. It only reads and writes the workload and background load to and from its neighboring processes. The configuration file is read before the test starts and the log is only stored in the memory buffers during the test. After the test is over, the memory buffer is flushed on the disk.

Communication. The inter-process communication between Seawind components need to be performed in a timely manner to ensure correct emulation results. Seawind uses TCP connections between the neighboring components. This is an obvious selection, because the compo-

nents can be distributed into multiple hosts located anywhere in the network, and the connection is required to be reliable. However, certain TCP features, namely the slow start and Nagle's algorithm [29], may cause unwanted delays in the delivery of workload data.

The packet size for the workload data should be selected to be small enough to fit in a single TCP segment in order to avoid the effect of slow start on the transmission rate. Usually this is the case, as the packet size is selected to be small on the emulated slow link, and on the other hand, Seawind is often used over Ethernet using 1500-byte frames. We have disabled Nagle's algorithm from the TCP connections used in the internal Seawind communication in order to allow TCP sender to transmit the TCP segments without delaying them. Finally, it is assumed that the link used to transmit the packets between NPAs and SPs is substantially faster than the emulated link. For example, 100 Mbps Ethernet is sufficient when emulating line rates up to 2 Mbps.

7 Case Study

We now show an example of how to study the behavior of a real TCP implementation by using Seawind. Figure 5 illustrates the target environment we are modeling and how it is emulated using Seawind. In this test case we assume a wireless last-hop link with a bandwidth of 9600 bps and a last-hop router with a buffer size for 7 network packets. The last-hop router is located on the same 10 Mbps LAN with the remote end host. Additionally, there are link buffers for four packets at both ends of the wireless link. Thus, the sending link buffer extends the total buffering capacity to 11 packets. These network properties are close to what GSM data has, for example. We do 20 replications of this test case.

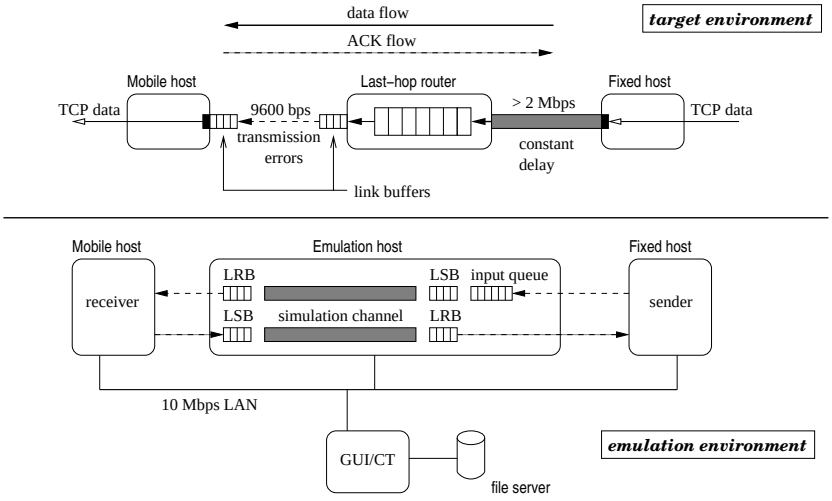

Figure 5: The emulated environment and its setup in Seawind.

In our scenario the wireless link is prone to transmission errors. The transmission errors are assumed to be detected and the corrupted packets are dropped. In our model the packet-drop probability is 1 % for the first 40 seconds of the test run. After 40 seconds the link quality decreases (e.g. the mobile user moves to a location with a weaker radio link quality) and the packet drop probability decreases to 10 %.

In Figure 5 we can see how the emulation is configured to use three hosts. One of the hosts acts as the mobile-end receiver, one of the hosts is the fixed end sender and one host is dedicated to the real-time emulation. Table 1 summarizes the Seawind parameters that were used by the Seawind SP to model the link at the emulation host. The router buffer is modeled with an input queue that drops the packets that do not fit in the queue using the *tail-drop* algorithm. The scenario is modeled with two distinct states in the Seawind state machine, one state for the first 40 seconds and another state for the rest of the test. We have left out from the table the Seawind parameters regarding the features that were not used in this test case. The workload we are using in this test is a bulk transfer of 100 KB using a single TCP connection over the IP protocol.

Table 1: Seawind parameters used in the case study.

Parameter Name	Value
input queue length	7 pkts
queue overflow handling	drop
queue drop policy	tail-drop
link send buffer size	4 pkts
link receive buffer size	4 pkts
transmission rate	9600 bps
propagation delay	200 ms
error handling	drop
packet error probability	state 1: 0.01, state 2: 0.10

Table 2: Summary of measurements.

Metric	Value
Elapsed time, 10th percentile	153.27 s.
Elapsed time, median	170.51 s.
Elapsed time, 90th percentile	196.40 s.
Throughput, median	601 Bps
Rexmitted pkts, median	65
Dropped pkts, median	47

After the 20 replications of the test have been run, Seawind has generated the logs of the test runs. First, we can have a look at the summary of the measurement results, which are shown in Table 2. The shown values are measured from the sending end TCP log. The table shows the median of the selected metrics. Additionally, 10- and 90- percentiles are shown for the elapsed time to illustrate the level of variability in the results. It is also possible to have a separate look at the statistics of each of the 20 replications. As every packet is logged with timestamps, protocol information and related Seawind events, measuring different kinds of metrics and performing different kinds of analysis is only a matter of having suitable scripts for the purpose.

After inspecting the general statistics for the replication set, the user can have a detailed look at what happens at the packet level. One way to do this is to generate a time-sequence diagram of the TCP segments, which is shown in Figure 6. When comparing the time-sequence diagram to the the Seawind event log, we can have an understanding of what happened during the test run.

There are only two corruption losses before the error rate changes after 40 seconds. These two packet losses occur in the beginning of the test and they cause the TCP sender to adjust its *slow start threshold* and enter congestion avoidance, in other words, reduce its sending rate. Thus, the last-hop router buffer load increases moderately, and there are no congestion-related losses until 35 seconds have passed. After the error rate has changed, there are 44 packet losses due to emulated transmission errors. Because of the higher loss rate, the TCP sender keeps transmitting at a low rate and the router buffer queue does not overflow for the rest of the test run.

We used Linux kernel version 2.4.0 at the endpoint hosts. Therefore the phenomena shown in the trace would really occur, if the Linux machine in question is used in the environment similar to what was modeled here.

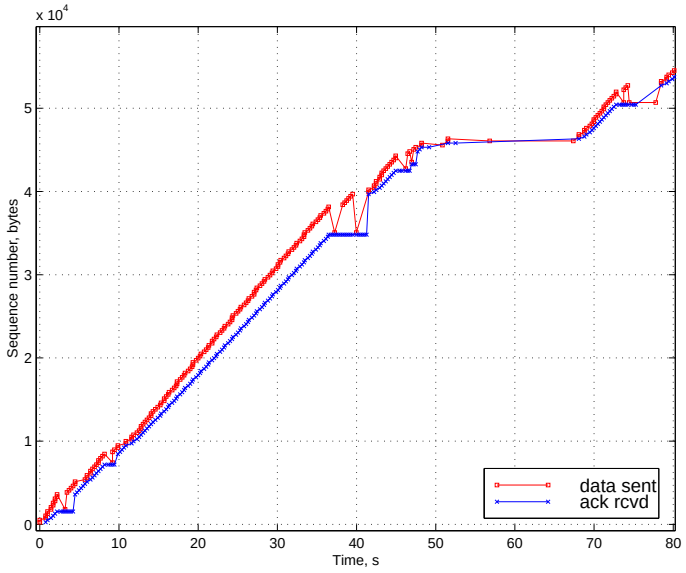

Figure 6: A time-sequence graph of the TCP segments in a test run.

8 Concluding Remarks

This paper presented a wireless network emulator called Seawind. The emulation approach allows performance evaluation of existing implementations of protocols and applications over a wide range of network characteristics. User scenarios that are difficult to reproduce in existing wireless networks or impossible when the network is only in the design phase can be easily presented in the emulator. Distinguishable features of the Seawind network emulator are its wireless-oriented design, portability, easy extendibility and an extensive environment of scripts and tools for the automatic set up of tests and analysis of results. The practical utility of Seawind is demonstrated by a case study and a number of studies beyond this paper. We have experimented with different operating systems and discovered a number of implementation specific features, of which some did not conform to the RFC specifications. We believe that slow links are an environment which have not been considered carefully enough when designing and testing the different implementations of TCP and other protocols. Therefore, we believe that Seawind is a valuable tool for testing the protocol implementations in different networking environments in a controllable fashion.

References

[1] M. Allman, A. Caldwell, and S. Ostermann. ONE: The Ohio Network Emulator. Technical Report TR-19972, School of Electrical Engineering and Computer Science, Ohio University, August 1997.

[2] M. Allman and A. Falk. On the effective evaluation of TCP. *ACM Computer Communication Review*, 5(29), October 1999.

[3] R. Bagrodia and W-L. Liao. Maisie: A language for design of efficient discrete-event simulations. *IEEE Transactions on Software Engineering*, April 1994.

[4] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz. A comparison of mechanisms for improving TCP performance over wireless links. In *Proceedings of ACM SIGCOMM '96*, Stanford, CA, August 1996.

[5] A. Boukerche, S.K. Das, A. Fabbri, and O. Yildiz. Exploiting model independence for parallel PCS network simulation. In *13th workshop of Parallel and Distributed Simulation 1999*, pages 166–173, May 1999.

[6] G. Brasche and B. Walke. Concepts, services and protocols of the new GSM phase 2+ general packet radio service. *IEEE Communications Magazine*, pages 94–104, August 1997.

[7] J. Brosh, D. A. Maltz, D. B. Johnson, Y. Hu, and J. Jetcheve. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM)*, pages 85–97, Dallas, Texas, October 1998.

[8] J. Cai and D. J. Goodman. General packet radio service in GSM. *IEEE Communications Magazine*, pages 122–131, October 1997.

[9] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[10] S. Floyd and K. K. Ramakrishnan. A proposal to add explicit congestion notification (ECN) to IP. IETF RFC 2481, January 1999.

[11] NIST Internetworking Technology Group. NIST net network emulation package. *http://www.antd.nist.gov/itg/nistnet/*, June 2000.

[12] A. Gurtov. Technical Issues of Real-Time Network Emulation in Linux. In *Proceedings of FDPW*, June 1999. Available at: http://www.cs.Helsinki.FI/u/gurtov/papers/.

[13] A. Gurtov. TCP performance in presence of congestion and corruption losses. Master's thesis, Department of Computer Science, University of Helsinki, December 2000. Available at:
http://www.cs.Helsinki.FI/group/iwtcp/papers/.

[14] ISI at University of South California. Network simulator 2. Available at: http://www.isi.edu/nsnam/ns/.

[15] V. Jacobson, C. Leres, and S. McCanne. `tcpdump`. Available at http://ee.lbl.gov/, June 1997.

[16] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling*. John Wiley & Sons, 1991.

[17] M. Kojo, K. Raatikainen, M. Liljeberg, J. Kiiskinen, and T. Alanko. An efficient transport service for slow wireless links. *IEEE Journal on Selected Areas In Communications*, 15(7):1337–1348, September 1997.

[18] J. Korhonen, O. Aalto, A. Gurtov, and H. Laamanen. Measured performance of GSM HSCSD and GPRS. In *Proceedings of the IEEE International Conference on Communications*, 2001.

[19] N. Leavitt. Will WAP deliver the wireless internet. *IEEE Computer*, 33(5):16–20, May 2000.

[20] R. Ludwig. *Eliminating Inefficient Cross-Layer Interactions in Wireless Networking*. PhD thesis, Aachen University of Technology, April 2000.

[21] M. Mouly and M. Pautet. *The GSM System for Mobile Communications*. Europe Media Duplication S.A., 1992.

[22] S. Ostermann. `tcptrace`. Available at: http://jarok.cs.ohiou.edu/software/tcptrace/tcptrace.html.

[23] M. Rahnema. Overview of the GSM system and protocol architecture. *IEEE Communications Magazine*, 31(4):92–100, April 1993.

[24] L. Rizzo. Dummynet: A simple approach to the evaluation of network protocols. 27(1):31–41, January 1997.

[25] P. Sarolahti. Performance analysis of TCP improvements for congested reliable wireless links. Master's thesis, Department of Computer Science, University of Helsinki, February 2001. Available at: http://www.cs.Helsinki.FI/group/iwtcp/papers/.

[26] J. Short, R. Bagrodia, and L. Kleinrock. Mobile wireless network system simulation. In *Proceedings of the First Annual International Conference on Mobile Computing and Networking (MOBICOM)*, pages 195–209, Berkeley, CA USA, November 1995.

[27] Simulation Laboratory (SimLab). MobSim++. Web page: http://www.it.kth.se/labs/sim/demoVisTool/mobsimdemo.html, October 1995.

[28] W. Simpson. The point-to-point protocol (PPP). IETF RFC 1661, July 1994.

[29] W. Stevens. *TCP/IP Illustrated, Volume 1; The Protocols*. Addison Wesley, 1995.

[30] R. H. Stine. FYI on a network management tool catalog: Tools for monitoring and debugging TCP/IP internets and interconnected devices. IETF RFC 1147, April 1990.

[31] OPNET Technologies. OPNET Modeler. Commercial, Information at: http://www.mil3.com/products/modeler/home.html, 2001.

[32] B. H. Walke. *Mobile Radio Networks: Networking and Protocols*. John Wiley, first edition, 1999.

[33] X. Zeng, R. Bagrodia, and M. Gerla. GloMoSim: a library for parallel simulation of large-scale wireless networks. In *Proceedings of the 12th Workshop on Parallel and Distributed Simulations (PADS '98)*, May 1998. Available at: http://pcl.cs.ucla.edu/projects/glomosim/documents.html.