

# TWINE: A Hybrid Emulation Testbed for Wireless Networks and Applications\*

Junlan Zhou      Zhengrong Ji      Rajive Bagrodia  
Computer Science Department  
University of California Los Angeles  
Los Angeles, CA 90095  
{zjl,jizr,rajive}@cs.ucla.edu

## ABSTRACT

In this paper, we present a high fidelity and efficient emulation framework called TWINE, which combines the accuracy and realism of emulated and physical networks and the scalability and repeatability of simulation in an integrated testbed, for evaluation of real protocols and applications. Our measurements show that the TWINE emulation kernel has a memory footprint of less than 100KB, and occupies no more than 3.5% CPU cycles. Thanks to such small overhead and the accurate modelling of physical layer events (at microseconds level), application throughput measured in TWINE is within 5% of the measured throughput from an equivalent physical wireless LAN. A single commodity PC in TWINE can emulate at least four wireless hosts or simulate sixty nodes in real time at microseconds granularity. This paper also illustrates TWINE's novel capabilities via two case studies: a protocol to maintain fairness in mesh networks and an adaptive streaming media application operating in heterogeneous wireless networks. The results from the case studies clearly show the benefit of the TWINE evaluation methodology, by identifying a mismatch between the performance of the protocol or application based on actual user experience versus its performance as measured using traditional network performance metrics such as application throughput.

## 1. INTRODUCTION

Much of the recent research attention in the wireless network community has focused on the use of cross-layer optimization techniques to improve network performance. This has led to the discovery and proposal of a number of algorithms to adaptively control the operation of wireless networks as well as the applications that they support. In addition, there is growing interest in pervasive/mobile computing applications that span diverse types of networks that might include wireless LANs, mobile ad hoc networks, sensor networks and mesh networks. Our goal is to develop an evaluation framework to better assist research efforts in the above fields. Traditionally, network simulators such as *ns-2* [36], GloMoSim [34], OPNET [37] and QualNet [39] or small-scale physical testbeds [22] have been the method of choice for such evaluations.

However, traditional methods are not entirely suitable to accurately evaluate the emerging generation of algorithms that can exploit cross-layer interactions among multiple lay-

ers of the protocol stack. Cross-layer protocol engineering is an emerging research area with a significant potential to support various applications. For example, a cross-layer design for link level optimization can be achieved by adapting the transmission power to the transport level delay and bandwidth requirements of realtime and non-realtime applications [7, 20]. Application layer adaptation techniques based on cross-layer design can be developed for wireless multimedia services which exploit the physical and network layer information [9, 6]. To support evaluation of wireless networks and applications of this nature, the ideal tool must:

- provide *realistic* details particularly at wireless physical layer, since the performance of wireless systems is greatly affected by fluctuating channel conditions resulting from mobility and interference.
- be *repeatable* with controlled parameters so as to support fair comparison among different system optimization techniques.
- be *flexible* to support diverse network conditions, novel radio devices, and link layer techniques.
- be *scalable* to permit evaluation of large-scale target networks in a lab environment.

Simulation tools offer a flexible and scalable approach to create reasonably detailed physical and link models, and to repeat and control target network conditions at user's requirement. Whereas they are extremely useful in evaluating specific solutions, they become unwieldy, inaccurate, or computationally intractable, when used for cross-layer evaluation of applications running on larger, diverse networks. In particular, traditional simulations are unlikely to be the tool of choice if we need to demonstrate the impact of novel cross-layer techniques with existing adaptive applications, such as streaming media application [10, 38], energy aware application [11, 24] and context aware application [14, 15], and the operating system supports for such applications [1, 27]. Similarly, simulations make it harder to capture the realtime interactions between physical applications and network protocols or to provide realistic foreground traffic models for cross-layer design evaluation. In contrast, a physical testbed can definitely capture cross-layer interactions between physical applications down to real physical links; however developing detailed implementations and subsequent scaling of such results using physical testbeds is prohibitively expensive and unsuitable for early design decisions. Physical testbeds are also difficult to control for repeatable experimentations.

\*This draft is NOT the final version that will appear in IEEE INFOCOM 2006.

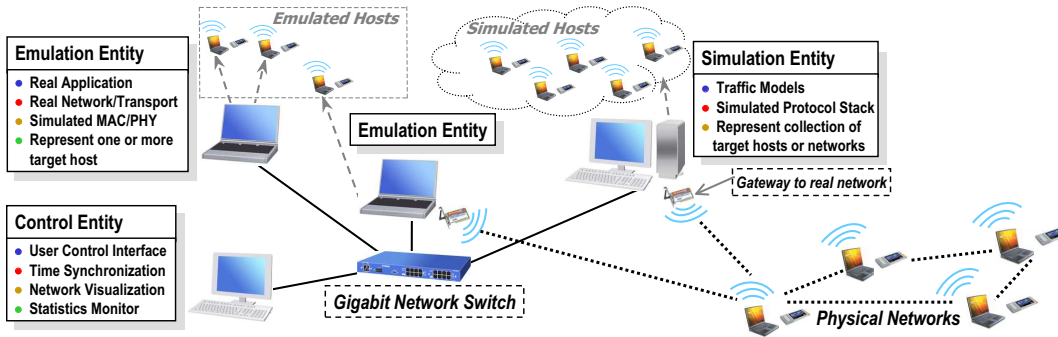


Figure 1: TWINE Architecture

In this paper, we develop a novel framework, that combines simulation, emulation, and physical networks in an integrated testbed, for evaluation of adaptive wireless systems, particularly those exploiting cross layer interactions. The integration of emulation and simulation in the developed framework enables evaluation of real applications and protocols over a wide range of wireless network scenarios while achieving scalability of the testbed. Interfacing with real networks, the developed framework also allows users to directly leverage existing, partially deployed physical testbeds and extend the performance evaluation in a larger-scaled networking context. In the development of such a hybrid testbed, we address the following primary technical challenge : seamless integration of detailed models of radio and MAC protocols into the operating system, such that real applications and protocols are oblivious to the fact that they are executing on simulated rather than physical devices. To the best of our knowledge, TWINE is the first emulation testbed that models the complex behavior of physical radios and channel characteristics at microsecond time granularity in real time.

We have evaluated TWINE regarding its primary performance goals: *realism* and *scalability*. TWINE emulation kernel has very low overhead: with a footprint of less than 100KB (which is smaller than a typical driver module), and incurs 3.5% CPU overhead at peak, when emulating 802.11b radio devices on commodity laptops. Thanks to such small overhead and high fidelity modelling of physical and MAC layers, the average UDP or TCP application throughput measured in TWINE matches physical wireless LAN measurement data closely, with less than 5% error margin. While achieving realism, the TWINE testbed also scales well: a single commodity PC can emulate four wireless hosts or simulate sixty nodes of a target network system with accurate realtime synchronization.

The use of TWINE as a *flexible* and *repeatable* evaluation tool for real protocols and applications has been showcased via two case studies: a protocol to maintain fairness in mesh networks [13], and an adaptive application that adjusts the streamed video quality based on the network conditions [38]. The case studies demonstrated the capabilities of TWINE to facilitate repeatable, flexible and efficient experimentations with protocol implementations and applications in complex network scenarios. The results from the case studies clearly show the benefit of the TWINE evaluation methodology, by identifying a mismatch between the performance of the protocol or application based on actual user experience ver-

sus its performance as measured using traditional network performance metrics such as application throughput. In particular, for the streaming media application, we showed that although variations in channel conditions and mobility patterns both had dramatic impact on the video quality when measured using traditional statistical metrics used by the research community, their impact when measured using a human observer’s perception of the resulting video quality were substantially different.

The remainder of this paper is organized as follows. Section 2 presents architectural overview of TWINE. Section 3 addresses design and implementation challenges. Section 4 and 5 presents system evaluation results, and two case studies of TWINE respectively. Finally, we discuss related work and offer our conclusion.

## 2. TWINE ARCHITECTURE

TWINE adopts a hybrid architecture consisting of multiple instances of emulation, pure simulation, and physical components. This design is motivated by the following two observations on experimentation of wireless networks and applications: First, there exists considerable heterogeneity in the scale, technology, and evaluation requirement for different subnetworks of a large-scale target wireless network; Second, most research groups only have access to a subset of hardware resources that are typically needed to set up their testbed. By exploiting the hybrid architecture of TWINE, a target network can be *decomposed* into interconnected *subnetworks*<sup>1</sup>, with each subnetwork of the target network represented by an instance of emulation, pure simulation, or physical components, depending on both experiment requirements and resource availability. Hence, the TWINE testbed can be tailored to an individual user’s needs and constraints.

An overview of the TWINE architecture is shown in Figure 1. As depicted in the figure, TWINE has three types of software entities: *emulation*, *simulation* and *control* entities. To mimic a complex target network, the TWINE testbed can have many instances of these entities, where each *subnetwork* is realized by an exclusive subset of emulation or simulation entities. A physical machine in TWINE (such as a PC) can execute one or more emulation entities, or a simulation entity. Of course, a simulation entity can also run in shared memory or distributed platforms to improve

<sup>1</sup>We use “subnetwork” to refer to a self-contained portion of the target network such that nodes in the subnetwork only contend for channel internally, but not with nodes in other subnetworks.

its scalability. Entities for the same *subnetwork* interact with each other to model the target network. Hence, their host machines are interconnected via gigabit network switch. Next we describe the functions of each entity.

An *emulation entity* mimics the behavior of a host in the target wireless network, henceforth referred to as *emulated host*. The protocol stack for an emulated host contains real implementations from the network up to the application layer, while the physical and MAC layers are simulated. A host in the target network is typically emulated when it needs to run real application. Note that multiple emulated hosts can coexist on the same host machine to improve testbed scalability.

A *simulation entity* models the entire protocol stack for a set of target wireless hosts, referred to as simulated hosts. Simulation entities are often used to model large-scale backbone networks, or other competing hosts whose operation does not need to be modelled at the level of fidelity as the emulated hosts. Both simulation and emulation entities can interface with real networks if required by the target network scenario.

A *control entity* manages emulation and simulation entities that belong to the same target subnetwork. It can issue commands to other entities, such as starting or stopping an experiment. It monitors traffics among other entities to produce realtime statistics. It also has a GUI for realtime visualization of network dynamics and online change of parameters in other entities. The functional scope of the control entity is limited to each subnetwork in order for the testbed to scale in large network scenarios.

There are a number of *benefits* to using such a hybrid architecture for evaluation of wireless systems. First, embedding the simulated physical and MAC layers directly into the operating system enables evaluation of real applications and protocols. Second, the integration of simulation entities improves scalability of the testbed for large target network scenarios. Third, interfacing with real networks allows the user to directly leverage existing, partially deployed physical testbed and extend the performance evaluation in a larger-scaled networking context. The end result is an evaluation framework that allows users to construct a wireless network testbed for real applications and protocols and to balance its scalability and fidelity with available hardware resources.

### 3. DESIGN & IMPLEMENTATION

In this section, we address the primary design and implementation issues of the emulation, simulation, and control entities in TWINE.

#### 3.1 Emulation Entity

Inside an emulation entity, the major tasks are performed in an *emulation layer* that is inserted between the network layer and the device driver in the Linux kernel<sup>2</sup>. One of the design requirements for TWINE is that it must be flexible and support novel radio devices and link layer techniques. To address this requirement, we take a modular approach in the design of the emulation layer so that future models can be easily integrated. In our design, the entire emulation layer consists of four simulation components (the core) and

four auxiliary components, the functions of which are briefly introduced below, followed by their interactions.

##### 3.1.1 Emulation Core

As shown in Figure 2, the simulation components (*MAC*, *PHY*, *Mobility* and *Propagation*) form the core of the emulation layer as they simulate the operational logic of physical and link layer, and the characteristics of wireless medium.

The *MAC* component embodies a set of simulation models of wireless MAC protocols. Each protocol model can be implemented as a finite state machine, where each state corresponds to the current action of the protocol, such as backing off. We adopt the event based approach in which MAC state transitions are triggered by events like timeouts. Events from other components are processed by a common event handler which in turn invokes specific routines supplied by the appropriate protocol model. We implemented the 802.11[16] protocol for this study.

The *PHY* component simulates all physical operation modes of a wireless radio as finite state machine. For example, a typical radio could be in transmitting, receiving or sleeping (idle) mode. Similarly to the MAC component, a generic event handler is designed to support multiple emulated network interfaces in one emulated host. An 802.11b PHY model is implemented for this study.

The *Propagation* component implements models that simulate the propagation effects of the wireless channel upon which the radio devices of emulated hosts operate. It receives position updates of emulated wireless hosts of interest from Mobility component, and calculates signal strength of incoming packets for the PHY component. TWINE currently supports Free Space and Two Ray path loss models, plus Ricean fading model[26].

The *Mobility* component currently supports random, group and trace-based mobility models. For trace-based model, the trace is stored in user space and prefetched by the kernel on demand. Mobility component also allows user to control the mobility via interfaces from the auxiliary components.

##### 3.1.2 Emulation Auxiliary

The auxiliary components provide the necessary interface between the emulation core and other protocol layers. The services provided by emulation auxiliary can be grouped into *control*, *communication* and *timing* services. These services are collectively implemented using the four modules described below.

*Local Control Kernel Interface* serves as the *control* interface for the emulation layer. It carries out control commands issued by the local control interface in user space throughout an experiment run. One such example is the initialization of appropriate MAC and PHY modules in the emulation core. It also outputs local reports consisting of experiment results of interest to the user, such as packet loss rate, queueing delay and throughput. It is implemented as a set of commands in the *ioctl* system call.

*Network Packet Service Unit* is the interface to the network layer. It intercepts packets from real applications and network protocols executed in the emulated entity via a netfilter hook, *NF\_IP\_POST\_ROUTING*, and passes them to the emulation core to emulate effects of wireless communication on those packets generated by the applications or

<sup>2</sup>Same design can be ported to other operating systems.

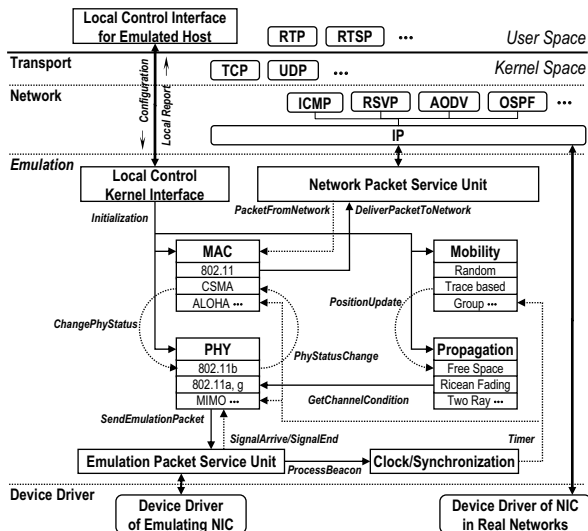


Figure 2: Design of Emulation Entity

protocols. In addition, it delivers incoming packets from remote emulation or simulation entities up to the network layer after they are processed by the emulation core, via respective protocol handlers in the kernel.

*Emulation Packet Service Unit* acts as the interface between emulation core and the underlying device driver in the Linux kernel. It constructs outgoing emulation packets and prefixes them with ethernet frame headers before handing them to device driver. It also performs another important function, timestamping the outgoing emulation packets with local emulation time (see next paragraph). For incoming frames, it decodes and dispatches the emulation packet to other respective emulation components. The above two components together provide essential *communication* service at the boundaries of real and simulated systems inside emulation entity.

*Clock and Synchronization* component handles all *timing* related tasks in the emulation layer. It manages software timer interrupts that drive the finite state machines in the emulation core. It also maintains a local (virtual) emulation clock, the value of which equals the physical time elapsed since an emulation experiment starts, indicating the emulation entity’s notion of time (or emulation time). The emulation clock ticks in real time along with the hardware clock of its host machine. To mitigate effects of clock drifting among different machines, the emulation clock is periodically calibrated with time reference beacons broadcasted by the control entity (see subsection 3.3).

### 3.1.3 Interaction Primitives

To maintain a modular design, we define a small set of primitives to support interoperability among the emulation core and auxiliary components. The primitives are classified into two groups, *external* and *internal*. The external interaction primitives are used to interface the emulation core with auxiliary components. They are enumerated in Table 1, together with a brief description. Note that Service Access Point (SAP) addressing at emulation layer is implemented in external interactions by mapping each emulated wireless interface to a real interface of the physical machine. Hence, one physical machine can support multiple emulated hosts.

Table 1: External Interaction Primitives

Interaction	From	To	Arguments
Initialization	Auxiliary	Core	References to predefined input structures for each component’s kernel modules
PacketFromNetwork	Auxiliary	Core	Reference to a packet from network layer
DeliverPacketToNetwork	Core	Auxiliary	Reference to a packet to be delivered to network layer
SendEmulationPacket	Core	Auxiliary	Reference to an outgoing emulation frame
SignalArrive	Auxiliary	Core	Timestamp and reference to an incoming emulation frame
SignalEnd	Auxiliary	Core	Timestamp and reference to an incoming emulation frame
ProcessBeacon	Auxiliary	Auxiliary	Sequence Number and Timestamp
Timer	Auxiliary	Core	Timestamp and reference to data supplied by Core when the timer is registered

Table 2: Internal Interaction Primitives within Emulation Core

Interaction	From	To	Arguments/Returns
ChangePhyStatus	MAC	PHY	New PHY state and other data (e.g. outgoing packet)
PhyStatusChange	PHY	MAC	New PHY state and other data (e.g. incoming packet)
GetChannelCondition	PHY	Propagation	Return current path loss and fading gain of a channel
PositionUpdate	Mobility	Propagation	Timestamp and new position of a mobile host

The internal interaction primitives (within emulation core) are shown in Table 2. As the emulation core consists only of simulation components, these primitives can simply be viewed as events, with each event triggering an appropriate action by the receiving component. This approach provides us with an easily extensible design: as long as the implementation of all core modules (such as MAC, or PHY) comply with predefined primitives, we can easily reconfigure the emulation entity by replacing one module with another within the simulation component, without affecting the module in other components. For example, we can replace the 802.11 module by the CSMA module in the MAC component, and retain 802.11b as the PHY module.

### 3.1.4 Improving Emulation Fidelity

TWINE differs from other existing emulation tools and methodologies in the literature in that it strives to maintain the highest fidelity at the physical and MAC layers, as we realize its importance to faithful evaluation of real protocols and applications. To accurately model the dynamics of physical and MAC layers, every emulated host must maintain a view of the arrival and termination of radio signals that is consistent with the global temporal order of the occurrence of these events. Furthermore, the consistent view must be maintained at all time. We refer to the above requirements as *Strict Consistency*.

However, due to physical propagation and processing delays, a packet (corresponding to a signal transmission event) may arrive at its destination later than the radio signal it represents. This causes the destination to receive events that happened in its past. Such events must be processed in correct order to preserve accuracy. This is better illustrated by an example in Figure 3. The expected behavior of a simple wireless LAN scenario is shown in Figure 3(a) where node  $S_2$  and  $D$  should both receive a packet transmitted by node  $S_1$ . However, the corresponding emulation packet from  $S_1$  arrives so late at  $S_2$  that  $S_2$  has completed carrier sensing and transmitted its own packet to  $D$ . The packets from  $S_1$  and  $S_2$  actually collided at  $D$  later (see Figure 3(b)). If this incorrect behavior is unchecked, the testbed’s fidelity would be in jeopardy.

To address the above problem, we implement a scheme similar to optimistic simulation [17, 12]. It allows nodes to locate the last local checkpoint before any consistency

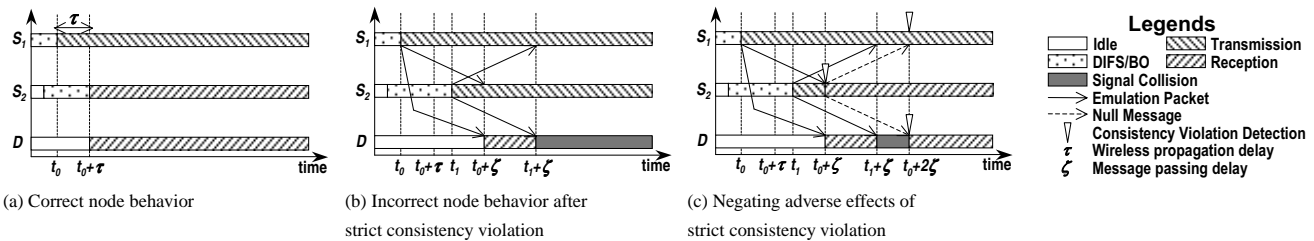


Figure 3: Illustrations to Strict Consistency Violation and How to negate its adverse effects

violation and to restore their correct MAC and physical layer states by negating all adverse effects caused by the consistency violation up to *current* time. To illustrate the above technique, let us again use the example in Figure 3. As shown in Figure 3(c), node  $S_2$  detects a consistency violation upon reception of a packet from  $S_1$ . This means that its most recent transmission to  $D$  should not have occurred because  $S_1$ 's last transmission caused  $S_2$  to sense the channel to be busy and  $S_2$  should have continued to backoff. A null message with a timestamp of  $t_1$  is then broadcast by  $S_2$  to nullify the previous transmission. Upon receiving the null message, both  $S_1$  and  $D$  recompute their physical and MAC layer states starting from  $t_1 + \tau$ .  $S_1$  does not detect any incorrect state transitions, while  $D$  restores its physical state to receiving packet from  $S_1$ . Our current implementation of the above scheme has covered all possible scenarios of consistency violations in 802.11 wireless networks operating in DCF mode. As a result, the physical and MAC layer models in TWINE conform to the 802.11 standard and operate in real time.

### 3.1.5 Key Design Characteristics

We summarize the primary characteristics of our emulation approach that provides the accuracy, extensibility, and low-overheads needed for interoperability with physical and simulated subnetworks: First, with loadable kernel modules in the design of the emulation core, we limit the footprint of a working kernel, thus reducing memory overhead on existing operating system. Second, with all modules conforming to well defined interactions between emulation components, users can dynamically configure modules to model complex physical devices with ease. Finally, we identified essential interactions between the emulation core and auxiliary components, with which detailed or abstract lower layer models such as those in [3, 5] can both be integrated in the emulation entity of TWINE.

To accurately model the complex behavior of radio devices and channel characteristics in real time, some key issues in the implementation of emulation entity must be addressed. First is to provide fine time synchronization (within error margin of few microseconds) among emulation entities. Second is to support high resolution timer event generation for the emulation core, as MAC/PHY finite state machines normally work at microseconds granularity. Finally, the emulation cores of multiple entities must maintain a correct view of the underlying physical channel, despite the physical propagation and kernel processing delays. In our work, fine time synchronization is achieved by a simple time synchronization protocol (see section 3.3). We leverage the KURT project [35] and port its high-resolution timer implementation into our version of the Linux kernel. Finally,

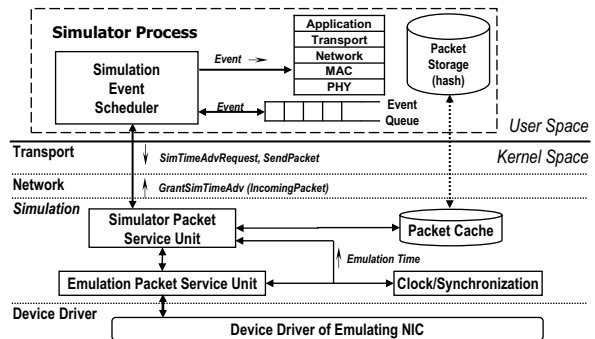


Figure 4: Design of Simulation Entity

to address the last challenge, we have implemented a scheme to maintain consistency of the temporal order among transmitted radio signals previously described in subsection 3.1.4.

## 3.2 Simulation Entity

The simulation entity can directly leverage existing network simulators, such as QualNet (used in this work), to model the full protocol stack of a set of wireless hosts. Inserted between the physical device driver and network layer of the host machine, TWINE *simulation layer* enables information exchange between the simulation entity and other entities in a seamless manner; this is an essential requirement for accurately capturing interactions among simulated and emulated wireless hosts. In a simulation entity, services used by the simulator are very similar to those in emulation entity, except that *control* services (e.g. initialization of simulation models) are typically facilitated by the simulator itself. Therefore, the rest of this section focuses on our contribution, which is providing the necessary *communication* and *timing* services.

### 3.2.1 Communicating with Network Simulator

To emulate packet level communications between hosts in a target network, other remote entities must be able to exchange information with network simulator regarding flows of both data packets and physical layer events. Similar to the design of emulation entity, we insert a simulation layer between the network and device driver layers inside the kernel to facilitate the information exchange (see Figure 4).

The simulation layer links a simulator process with the device driver. The interface to the device driver is realized by the emulation packet service unit. Its function is similar to that of its counterpart in emulation entity, except that it does not perform timestamping of outgoing packets (which has been performed by network simulator). Together with simulator packet service unit, it handles all incoming and outgoing data flows.

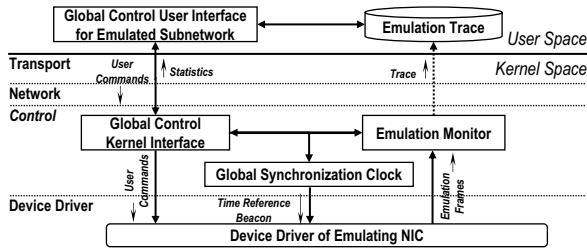


Figure 5: Design of Control Entity

The simulation layer interfaces to a simulator process by its simulator packet service unit. To minimize processing delay, the interface is implemented as file system calls which bypass upper protocol stacks; further, only essential packet header information is passed to the network simulator. The original packet is saved in a small cache in kernel space or moved to storage in user space if it traverses long expected routes in simulation. In most scenarios a real packet will travel through the wireless subnetworks simulated by the network simulator. Thus, the network simulator must be modified to issue a *verdict* for the cached packet. A *verdict* uniquely identifies a packet and announces whether it was lost inside the simulated subnetwork, or is leaving the current simulated subnetwork. Upon reception of a *verdict*, simulator packet service unit will either drop the packet, or update the packet and send it to its remote destination.

### 3.2.2 Synchronizing Network Simulator in Real Time

A network simulator usually manages its time using its own simulation clock. For simulation and emulation entities to interact in a timely manner, execution of the simulator process must be synchronized with that of other remote emulation(or simulation) entities. This is handled by the clock and synchronization component, which maintains an emulation clock that is synchronized with other entities. The current time of the emulation clock limits the advancement of the simulation clock using the concept of lookahead[23], hence pacing the progress of simulation with real time.

To impose such restrictions on a simulator process, we modify the event scheduler inside a network simulator such that before processing the next event in its event queue, it must invoke a *read* system call to the simulator packet service unit. Through the system call, simulator passes the simulation timestamp of the next event in its queue as a time advancement proposal, and a buffer to receive a potential incoming packet from the kernel. In the system call, simulator packet service unit will evaluate the time advancement proposal. The time advancement is granted if no lookahead violation is found. However, the simulator process will be blocked when its proposal violates lookahead constraints (typically when the requested simulation time is too far ahead of current emulation time). A software timer interrupt is then scheduled to wake up the simulator later when it is safe to proceed. While the simulator is in blocked state, a new packet may be received by the kernel, which needs the simulator’s sooner attention. In this case, the simulator packet service unit cancels the timer, copy the packet headers to the *read* system caller’s (simulator’s) buffer, and resumes execution of the simulator.

## 3.3 Control Entity

As mentioned in previously, the control entity provides centralized control for both emulation entities and simulation entities in each target subnetwork. It does three main tasks: time synchronization, dynamic control and monitoring (such as collecting trace). We will only discuss time synchronization for brevity.

Time synchronization is indispensable as the emulated (or simulated) hosts cannot interoperate without sharing the same notion of time. TWINE needs high timing accuracy as physical layer events occur at tens of microseconds granularity. Fortunately, global synchronization is unnecessary since only entities that interact with each other need to be synchronized. To meet these requirements, we implement a simple yet effective beacon-based time synchronization algorithm. In our solution, the control entity periodically broadcasts time reference beacons to its clients (other entities). The beacon is used by each client to calibrate its end-to-end latency. The sum of the calibrated end-to-end latency and the timestamp of the received beacon is a client’s notion of *time* exactly when the beacon is received. The client then advances its *time* independently with its system clock until receiving the next beacon. As the protocol is implemented in the kernel to reduce OS delay, the simple solution reduces timing error to  $\pm 5\mu s$  at 4 beacons per second based on our measurement, a cheap alternative to GPS.

## 4. SYSTEM EVALUATION

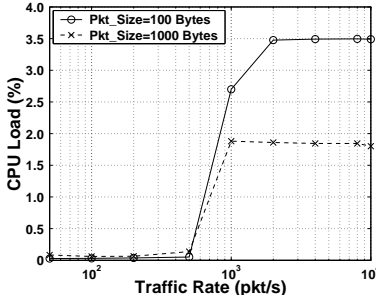
In this section, we evaluate the performance of TWINE regarding its two primary goals: *realism* and *scalability*. To enable realistic evaluation of network protocols and applications, the emulation kernel shall not interfere with execution of operating systems and applications, while maintaining high fidelity in lower-layer modelling to produce trustworthy results. Further, the emulation testbed needs to support large testing scenarios with least amount of hardware to be efficient and cost effective. We used the above three concrete measures as the gauge of TWINE’s success. To summarize our findings: our measurements show that the emulation layer has a memory footprint of up to 100KB, and consumes less than 3.5% of CPU time (under 1% in many cases). Thanks to the small overhead, the average application throughput on emulated wireless nodes, predicted in TWINE, is within 5% difference of the results from the corresponding physical testbed. Using TWINE, a commodity PC can emulate four wireless hosts, or simulate sixty nodes with high fidelity (modelling physical and MAC layer behavior at microseconds granularity).

### 4.1 Emulation Layer Overhead

First, we measure the memory footprint of the emulation layer by loading all of its modules within the Linux kernel. With no mobility, the emulation layer occupies 82.4KB memory at runtime; using mobility profiles specified by periodic positions recorded in a file (trace-based mobility), the emulation layer prefetches the mobility trace from user space in blocks of 16KB at a time, which increases the overall kernel memory consumption to 98.4KB. Even at this level, the total memory consumption is still smaller than that of a typical SCSI driver module. Note that kernel buffers for packets are not counted in the overhead as these are also used by the real device drivers.

**Table 3: Machine Configurations**

Machine Model	CPU	MEM	Ethernet Network Card	Operating System	Usage in TWINS
Dell Latitude D600	Centrino 1.6GHz	512MB	Broadcom BCM5700 1G-BT	Redhat 9.0 (Kernel 2.4.20)	Emulation Entity
Dell Dimension 8400	Pentium-4 3GHz	1GB	Broadcom BCM5700 1G-BT	Redhat 9.0 (Kernel 2.4.20)	Simulation/Emulation


**Figure 6: CPU Overhead of Emulation Layer**

Next, we measure the CPU load of the emulation layer with a simple scenario: A sender and a receiver are 5-meter apart and communicate with 802.11b radios operating in DCF mode at the highest rate (11Mbps). In the emulation, each node is emulated on a laptop (Refer to Table 3 for configuration of machines used in this paper), where the laptops are connected via a physical Gigabit switch. The sender opens a socket and sends fixed-sized UDP packets to the receiver at a constant rate. In order to characterize the overhead over a large operating space, we vary application traffic rate from 50 to 10K packets per second ( $pkt/s$ ), using packet sizes of 100 and 1000 bytes.

Figure 6 shows that the average CPU load of the emulation layer at the sender<sup>3</sup> increases in a nonlinear manner with the offered application traffic load. When the traffic load is less than  $500\text{pkt/s}$ , the CPU load remains less than 0.1% regardless of packet size. This is because the emulated sender-to-receiver link is far from saturation and queuing overhead is negligible. As the volume of application traffic increases, the CPU load saturates at 3.5% for packet of 100 bytes, and 1.8% for 1KB. The super-linear increase in the CPU load before saturation is mainly attributed to the effect of queuing packets at the emulation layer. The CPU load due to the emulation layer saturates when the application traffic itself saturates the corresponding physical 802.11b radio, because additional application packets are declined by the Linux socket as its buffer is full. Further, the saturated CPU load for 100B packets is reached at higher packet rate than that for 1KB packets because more packets are needed to saturate the emulated wireless link. The saturation point is also higher for the 100B packets because the smaller size implies that there are more packets in the “saturation” queue, which in turn means more processing overhead.

The above results demonstrate the dependence of the CPU overhead of the emulation layer on the number of packets transmitted per unit time by the emulated node. Therefore, the CPU load of 3.5% that was measured when the sender-to-receiver wireless link is saturated, represents the worst-case scenario for single channel 802.11b wireless LANs.

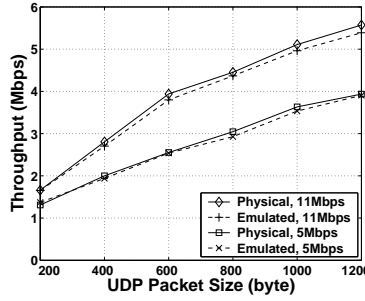
## 4.2 Validation of Application Throughput

In this section, we validate the throughput in TWINE

<sup>3</sup>CPU load at the receiver is lower due to fewer events to process.

**Table 4: TCP Throughput Comparison**

Node type	1Mbps	2Mbps	5Mbps	11Mbps
Physical	90.4kB/s	162.4kB/s	332.3kB/s	483.4kB/s
Emulated	89.9kB/s	158.1kB/s	329.5kB/s	480.2kB/s


**Figure 7: Comparing UDP Throughput of Emulated and Physical Wireless Nodes**

testbed against real measurements for both UDP and TCP traffic using the same topology as in the previous subsection.

First, we compare the maximum achievable UDP throughput under various 802.11b supported data rates. For the real testbed, we place two laptops about 5 meters apart, each installed with a Linksys 802.11b card. Note that the real measurement is conducted in our campus football field at night to alleviate intractable effects on wireless channels like interference. For the equivalent TWINE testbed, we simply run emulation entities on the same laptops in our lab, replacing the 802.11b cards with gigabit ethernet cards. In both cases, our UDP application overruns the real or simulated 802.11b radio in order to achieve maximum UDP throughput. Besides radio’s data rate, we also vary the UDP packet size in the experiment. Figure 7 shows the measured UDP throughput in the emulated and physical wireless nodes (Results are plotted in two graphs for clarity). The throughput difference between the emulated and physical nodes is less than 5%. This is not surprising since the CPU overhead of the emulation layer is less than 3.5%. With such small overhead, the emulation layer has marginal impact on the accuracy of TWINE testbed in this experiment.

Next, we measure the TCP throughput by replacing the UDP application with a real *scp* session<sup>4</sup>. A large file of 80MB is transferred to ensure confidence of the measurement. Table 4 shows the time average of transfer rates reported by *scp* in real and TWINE testbeds. The difference of TCP throughput reported by both testbeds is less than 3% for all data rates. The TCP throughput is lower than the maximum UDP throughput in Figure 7, because receiver also sends acknowledgement(Ack) back to sender, which introduces contention between *scp* source and destination. The close match between the results indicates that contention and its avoidance mechanism are accurately modeled in TWINE.

## 4.3 Scalability of TWINE entities

In TWINE, each emulation (simulation) entity can emulate (simulate) multiple wireless hosts. However, the computation overhead of the corresponding entity increases with the number of target hosts in an entity. Such increase could result in more events not processed in real time and incur

<sup>4</sup>*scp* is a secure file transfer application.

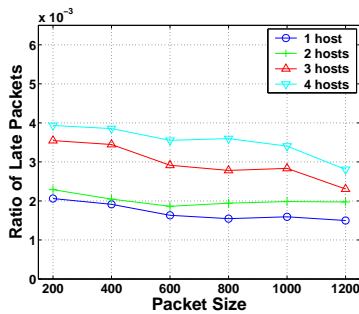


Figure 8: Emulation Scalability: Emulating multiple hosts in orthogonal channels by a PC

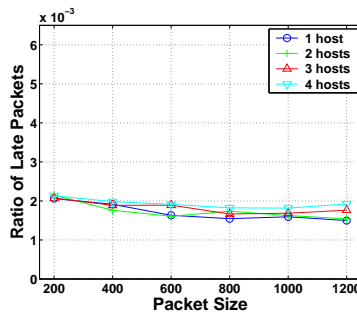


Figure 9: Emulation Scalability: Emulating multiple hosts in a single channel by a PC

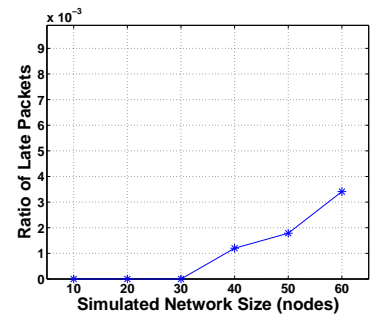


Figure 10: Simulation Scalability: Simulate an ad-hoc network by a PC in real time

inaccuracies on the experiment results. In this section, we conducted experiments to evaluate the inaccuracies incurred on an emulation or simulation entity, using the ratio of late packets as the metric. A packet is considered late when it misses its deadline to be delivered to the real network layer of the intended receiver by  $10\mu s$ .

First, scalability experiments were conducted with emulation entity. To show the trend in scalability, we use a single desktop PC to emulate up to four wireless hosts in the experiments. Each emulated host in the desktop runs an application that sends backlogged UDP packets of fixed size (from 200B to 1200B) to a separate destination emulated on a laptop. Therefore, the target network consists of 1, 2, 3 or 4 pairs of hosts, of which the traffic sources are emulated on the same PC while the destinations are emulated on individual laptops. The experiments are further divided into two scenarios: In scenario one, each source and destination pair of the emulated hosts communicate in an orthogonal channel; whereas in scenario two, all of them operate in a single channel. In both scenarios, the emulated hosts operate in 802.11b DCF mode and transmit at 11Mbps.

Figure 8 shows the ratio of late packets when the desktop PC emulates one to four wireless hosts in scenario one. It is observed that even when four hosts are emulated on the same PC, the ratio of late packets is very small, specifically less than 0.4%. Careful study of the experiment trace reveals that no packet would be late if emulation event handlers were always invoked at their scheduled time. This suggests that imprecision of software timer is the primary cause of late packets. In the current implementation of high resolution timer (leveraged from KURT’s UTIME [35]), timer handling routines are invoked as deferrable functions outside the interrupt handlers to keep kernel response time small. As a result, the delay between a timer interrupt and the invocation of emulation event handler inside the timer handling routine increases with the kernel workload. This explains why the ratio of late packets grows as more hosts are emulated on the desktop. Nevertheless, the small computation overhead in emulation event handler impeded the growth in the ratio of late packets when we scale up the emulation, so that the measured UDP throughput in each emulated host is almost identical to that observed in subsection 4.2.

In contrast, Figure 9 shows that the ratio of late packets remains almost constant when we scale up the emulation in scenario two. This constancy is attributed to an optimization that aggregates and batch-processes emulation events with

closely matched timestamps (within  $3\mu s$ ) in one software timer. It significantly reduces interrupt context switching overheads on the desktop which emulates several hosts in scenario two. This is because physical layer events (e.g. receiving a signal) at different emulated hosts often have very close timestamps and can be aggregated, because the emulated hosts share a same channel. The above results show the emulation entity scales well with increasing number of emulated hosts if they operate in the same channel.

Next, we conducted scalability experiments on a simulation entity. The network simulator used in the experiments is QualNet [39], which has a rich set of high fidelity models for wireless networks. The experiment scenario is similar to that in Figure 17, which consists of two emulated wireless LANs connected via a simulated ad hoc network. Each wireless LAN has an AP and a client which are emulated on two laptops; while the ad hoc network is simulated on a desktop. All nodes in the target network operate in 802.11b DCF mode at 11Mbps. The ad-hoc network and wireless LANs operate in orthogonal channels. Nodes in the ad-hoc network are initially placed in flat terrain in grid formation of 200m distance. The client in one wireless LAN sends backlogged UDP packets of 200B to the one in the other wireless LAN. Besides, 10% of the nodes in the ad-hoc network generate UDP packets (512B) at  $10pkt/s$ . To show the trend of scalability, we vary the size of simulated ad hoc network from 10 to 60.

Figure 10 shows the ratio of late packets measured at the AP of the destination client. Even when running on a commodity desktop, a TWINE simulation entity, synchronized with real time at microseconds granularity, can simulate 60 nodes with a very small ratio ( $< 0.4\%$ ) of late packets. The above results show that even with commodity hardware, inclusion of simulation entity has improved TWINE’s scalability by an order of magnitude. Further optimization on simulation entity can be achieved by execution on parallel and distributed platforms. In depth discussion on simulation scalability can be found in [33], [29] and [18].

## 5. CASE STUDIES

In this section, we present two case studies to showcase TWINE applicability to experimentation with both wireless network protocols and applications. The distinctive TWINE capabilities that are demonstrated by the two case studies are as follows: first, unlike with simulations, physical implementations of protocol or application can be tested

directly with TWINE testbed; second, unlike physical testbeds, repeatable experimentations can be preformed easily while exposing a large set of controllable parameters (such as background traffic, mobility and wireless channel characteristics etc.) to the analyst; and third, the testbeds for complicated scenarios can often be set up with flexibility, ease, and much less cost. Hence, the TWINE approach enables a faster and smoother transition from modeling and analysis to deployment for wireless network protocols and applications.

## 5.1 Inter-TAP Fairness in Mesh Network

In the first case study, we use TWINE to evaluate a distributed QoS mechanism – Inter-TAP Fairness Algorithm (IFA)[13]. IFA intends to eliminate the unfair throughput (or temporal) sharing among users connected to a wired backbone network from different access points (APs) in a mesh (backhaul) network. IFA assigns the temporal share of each inter-AP link, and subsequently the rate of each end-to-end flow, based on traffic load and topology information, so as to maximize the overall throughput of backhaul networks while satisfying the fairness constraints. Each mobile host is then notified by its AP of its throughput share and uses a rate limiter to enforce its share. Full details and simulation (*ns-2*) based evaluation of IFA has been previously published in [13].

Unlike the previous study of IFA, The TWINE study reported in this paper uses an actual implementation together with artificially generated traffic as well as traffic generated from physical applications, to evaluate the performance of IFA. This not only provides a versatile context to evaluate the effectiveness of throughput fairness among APs with different types of application traffic, but it can also resolve implementation and interoperability issues and shorten the development cycle without consuming significant additional resources. Though originally proposed as a MAC layer protocol, we find IFA more suitable for implementation at the network layer of real Linux system. For the purpose of this study, we implemented IFA in Redhat Linux 9 and used that implementation for all results reported in this paper.

### 5.1.1 Evaluation with Mixed Traffic

We begin our first case study with a demonstration of the TWINE’s utility to support evaluation of an actual implementation protocol using a mixture of real and synthetic traffic. We used a topology depicted in Figure 11, which is similar to that originally used in [13]. The topology includes five single-AP wireless LANs, each hosting up to five mobile hosts(MH). The APs form a multi-hop backhaul network. All inter-AP and MH-AP links use 802.11b radios operating at 2Mbps in DCF mode. Each wireless LAN operates in an orthogonal frequency band. As shown in Figure 11, there are three traffic flows, a backlogged UDP flow of 1KB-sized packets from  $MH_7^2$  and two TCP flows from  $MH_1^1$  and  $MH_3^1$ , all destined to  $MH_1^5$ .

The above topology is mapped onto the TWINE testbed by using a combination of emulation and simulation entities (running on machines listed in Table 3). To demonstrate the flexibility of the TWINE framework, we emulate three wireless LANs together with their APs, and simulate the other two. We use emulation for wireless LANs including

both source and destination of TCP traffic. We use a single simulation entity for the rest of this network including all radio interfaces forming wireless LAN 2 and 4, as well as the radios of all APs which form the multi-hop backhaul network. The above mapping allows us to use real applications, in particular *scp* (TCP) file transfers, as the source of TCP traffic, and synthetic CBR sessions for UDP traffic.

The measured throughput is plotted in Figure 12. We observe that throughput fairness for mixed traffic is very poor without IFA, as the *scp* session at  $MH_1^1$  is completely stalled. The improvement by IFA to the fairness among three traffic flows is obvious from the figure. However, *scp* session at  $MH_1^1$  suffers 35% lower throughput compared to others. This is because its Round-Trip Time(RTT) is still larger than that of the *scp* session from  $MH_3^1$  to  $MH_1^5$  despite IFA. Another observation made from Figure 12 is that IFA improves the fairness among mixed TCP/UDP flows at a cost of significant overall reduction in throughput. With synthetic UDP as background traffic in TWINE, it appears that IFA is less effective in maintaining fairness for traffic from a real application, *scp*. This result motivated us to use a scenario where all traffic was generated from actual applications, and to evaluate the performance of the scheme based on both measured statistics as well as a human user’s response to the effectiveness of the fairness strategy.

### 5.1.2 Evaluation with Real Traffic

In this experiment, we show that TWINE can support evaluation of IFA with the highest fidelity by relying entirely on traffic generated from real applications. For this purpose, we further extend the testbed setup shown in Figure 11 for the previous experiment. The changes to the previous scenario and setup are as follows. Each of  $AP_1$ ,  $AP_2$ , and  $AP_3$  is now associated with a single mobile host, referred to as  $MH_1^1$ ,  $MH_1^2$  and  $MH_1^3$  respectively, each of which is represented by an emulation entity. The remaining setup for  $AP_4$  and  $AP_5$  is identical to what it was in the previous experiment. Three undergraduate students (users) were assigned to run real applications on top of the emulated mobile hosts to mimic the traffic from the previous experiment. In particular, users *A* and *B* each start a *scp* session on  $MH_1^1$  and  $MH_1^3$  respectively, to copy a large file to  $MH_1^5$ . At the same time, user *C* runs a VLC media player[40] server at  $MH_1^2$ . The same user also opens a VLC media player client at  $MH_1^5$  to display a video clip streamed from the server.

The primary purpose of the experiment was to record the impact of the “unfairness” in the bandwidth sharing, both with and without IFA, as perceived by human users working with real applications. So, together with the statistics collection, we also observed the response of each user. In the scenario before IFA was applied, the first *scp* user (user *A*) became frustrated as his *scp* was virtually stalled all the time, and he decided to quit after the fifth attempt. However, users *B* and *C* seemed happy with the performance of their applications and continued to use the system. When IFA was applied to this scenario, the quality of the video stream played at the VLC client became sufficiently poor (its throughput was cut by half), that the user *C* too “quit”

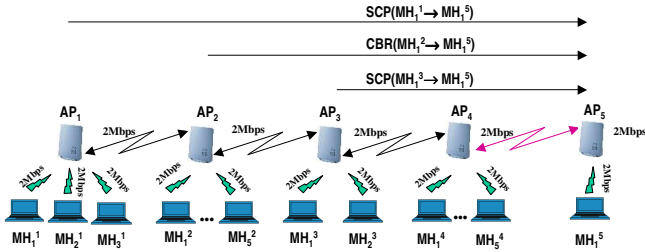


Figure 11: Mesh network scenario with mixed traffic

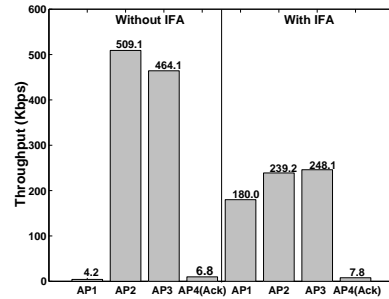


Figure 12: IFA performance with mixed traffic

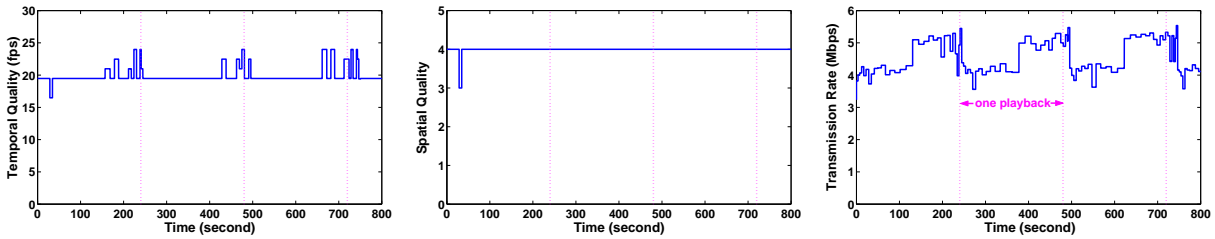


Figure 13: QStream performance in baseline scenario

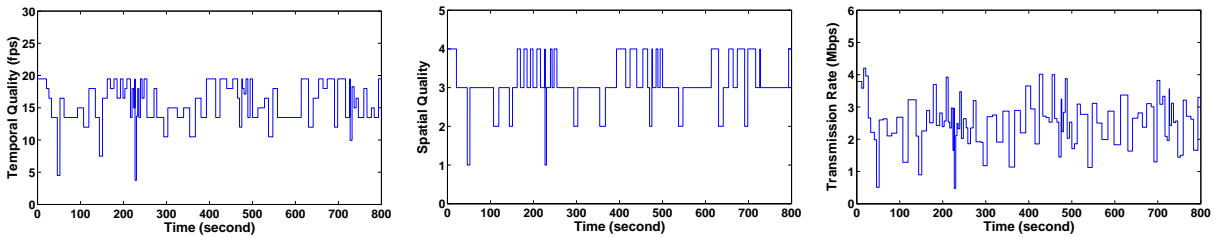


Figure 14: QStream performance in multipath fading environment

using the application. User *A* was happy with the IFA as his *scp* session achieved an average throughput of 22KB/s. User *B*'s *scp* session was not stalled, but its throughput was cut by almost half. The results from this simple study using the mixed quantitative and perceptual valuation supported by TWINE clearly showed that, in addition to throughput, fairness algorithms must also consider other issues like application QoS requirements to be effective. Evaluations with realistic hybrid emulation testbeds like TWINE provide a cost-effective approach for such analysis.

## 5.2 Adaptive Streaming Media Application

In the second case study, we use TWINE to evaluate the performance of an adaptive streaming media application, QStream [38] in wireless networks. QStream supports two independent video quality dimensions: spatial and temporal. Spatial quality is a measure of spatial resolution in a video frame; current codecs like SPEG (Scalable MPEG) [21] support four spatial-quality levels. Temporal quality refers to frame rate or number of frames displayed per second (fps). To support rate adaptation, streaming data packets are mapped to different priorities based on user defined adaptation policies that specify the utility for each quality level of streaming media. The packets in the same time window (or adaptation window) are then transmitted in descending order of their priorities (referred as Priority Progress Streaming (PPS) [21]), while outstanding packets are dropped upon expiration of the current adaptation window. In this way, QStream relies on transport protocol (TCP) for rate control, and maintains quality and timeliness

of streamed video by data packet reordering and dropping.

Using a small set of host machines and a scalable simulation environment, we use TWINE to analyze the performance of QStream in various wireless network scenarios with different settings of propagation models, traffic patterns, node populations, and mobility. Once again, we use both synthetic traffic and real applications in the study, and use statistical and subjective measures, which are useful for a comprehensive evaluation of multimedia applications. In fact, the results from both measures in the fourth experiment (see subsection 5.2.4) revealed a clear mismatch between the responsiveness of the quality adaptation mechanisms to dynamically changing network throughput and the improvements on a user's viewing experience. To the best of our knowledge, TWINE is the first demonstration of an emulation tool capable of experimenting with real applications like QStream under various wireless network conditions that include multi-path fading, congestion and mobility.

Two network topologies were used for this study: a wireless LAN topology of 6 mobile hosts in an area of  $200m \times 200m$  for the first three experiments; and a heterogeneous network topology (see Figure 17) for the last experiment. The former topology is sufficiently small that it can be realized using only emulation entities, while the latter uses a combination of emulation and simulation entities in order to accommodate the significantly larger configuration (see subsection 5.2.4). The common setup in all the experiments is as follows. All nodes in both topologies use 802.11b radio at a fixed rate of 11Mbps in DCF mode. Signal

attenuation follows a Free-Space pathloss model, unless specified otherwise. There is one QStream server and one client. When requested, the server will stream a 4-minutes movie clip in a continuous loop to the client. The video stream has a variable bit rate from 5 to 10Mbps without adaptation, and minimum rate of 380Kbps with lowest-quality adaptation. The same linear utility function has been specified as the adaptation policy for both temporal and spatial quality of the streamed video at the QStream server.

### 5.2.1 Single-Hop(Baseline) Performance

In the first experiment, we establish the baseline performance of QStream by disallowing competing traffic from other mobile hosts. Thus, the only traffic that exists in the target wireless LAN is the video stream from the server directly to the client. The results obtained here will be used as a reference for evaluation of subsequent configurations. Figure 13 shows the temporal and spatial quality of the video received at the client, and the transmission rate of the video streamed by the server. In this best case scenario, the video displayed at the client reaches its maximal spatial quality level-4. The transmission rate and frame rate of the video also increases in the second half of each playback. Such growth is related to window scaling in PPS. Window scaling is used in PPS to tradeoff the responsiveness of the streamed video with its adaptation granularity. With window scaling, the adaptation window starts out being set to a minimal value to ensure a small startup delay of video, and grows as it detects unused bandwidth to support more fine-grained quality adaptation. As a result, the transmission rate of streamed video data grows and the temporal quality of streamed video improves as the video continues to play.

### 5.2.2 Impact of Fading Channels

In the second experiment, we evaluate the quality of streamed video over a Rayleigh fading channel. The setup of this experiment is same as the previous one except that the Rayleigh fading channel model is enabled. Figure 14 shows the impact of this change where the transmission rate of video data varies wildly during the streaming, and both temporal and video quality fluctuates frequently. In accordance with the quantitative measures, human observers noted constant and dramatic variations in the picture quality and smoothness of motion. In the worst case, the video was frozen for about 3 seconds. Such variation can be attributed to the suboptimal performance of TCP[2] over lossy wireless links. The TCP sender assumes that the main cause of loss is congestion and responds by halving the TCP window. As a result, the amount of video data transmitted per adaptation window varies wildly as shown in Figure 14, though the available network bandwidth remains constant. This leads to the undesirable frequent changes of video quality in contrast to the baseline performance.

### 5.2.3 Adaptation to Congestion

In the third experiment, we evaluate the performance of QStream under congestion. Using the same wireless LAN topology, each of the four previously silent hosts now have a backlogged UDP traffic session. The UDP sessions are started at emulation time 120s, 240s, 360s and 480s respectively so as to gradually increase the degree of

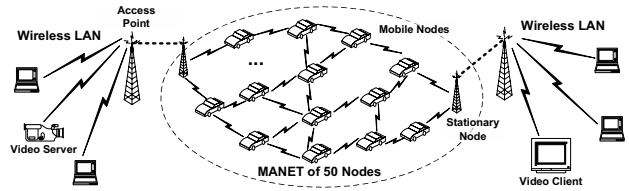


Figure 17: Heterogeneous wireless network scenario

congestion; once started, they remain active until completion of the experiment. Figure 15 shows the temporal and spatial quality of the streamed video, and the transmission rate of streaming data. The transmission rate of streaming data gradually decreases due to the increasing number of UDP sessions. The drop in transmission rate of streamed video peaks (by about 1/2) when the first UDP session starts, and becomes less and less when later UDP sessions become alive. In fact, QStream(TCP) still gets a proportional share of the total capacity of the wireless LAN. This is not a surprise as backoff procedure in the 802.11 MAC protocol help maintain fairness among active nodes in a fully connected network. Overall, spatial quality and transmission rate of the streamed video gradually degrade as the number of UDP sessions increase, but the temporal quality oscillates dramatically, especially between 400s and 500s time marks. This oscillation is most likely caused by nonuniformity in frame sizes(related to video attributes such as motion). From the perspective of a human observer, a definite degradation in video quality is observed when the first two UDP sessions are initiated. Thereafter, the degradation was not perceivable to the human observer. This shows that quality degradations perceived by human users are not always correlated to the objective measures like data transmission rate and further demonstrate the importance of perceptual evaluation for adaptive media applications, which can be easily supported by TWINE.

### 5.2.4 Impact of Mobility in Ad Hoc Networks

In the last experiment, we use TWINE to evaluate the performance of QStream in a heterogeneous wireless network with mobility, as depicted in Figure 17. This target network includes two wireless LANs and one mobile ad-hoc network (MANET), operated in three non-overlapping channels. There is no node mobility in the two wireless LANs, where the QStream server and client are respectively located. The MANET includes 50 nodes in a  $1500m \times 1500m$  area. Two stationary nodes in opposite corners are connected to the access points of the wireless LANs via wireless point-to-point links. The others start at random positions and move in accordance with the random-way-point mobility model at a speed of  $12m/s$ . The MANET uses AODV as its routing protocol. The video stream is the only traffic flow in the target network. To set up the target heterogeneous wireless network in TWINE, we use eight emulation entities and one simulation entity. Each emulation entity emulates a wireless node or AP in the two wireless LANs. The simulation entity models the MANET.

Figure 16 plots the temporal and spatial quality of the streamed video, and the transmission rate of streaming data. Due to self-coupling effect in the multi-hop wireless network, the end-to-end throughput of streamed video is significantly lower than that in the baseline experiment.

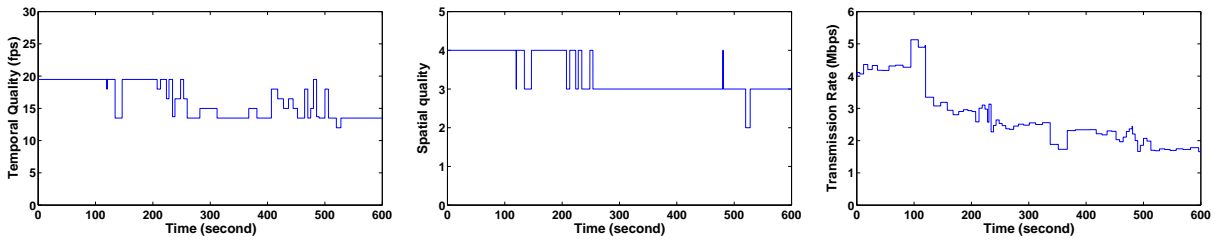


Figure 15: QStream performance under congestion

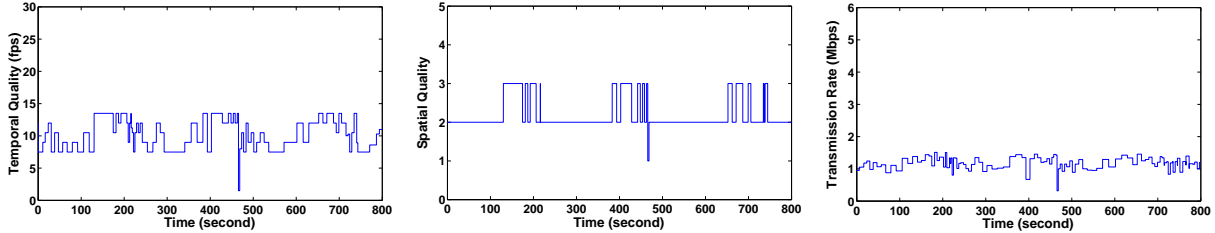


Figure 16: QStream performance in heterogeneous networks

Some packet losses are expected for two reasons: high mobility causes frequent route failures; AODV prefers route with smaller hop count (longer distance per hop), while QStream transmits at maximum IP packet size. These packet losses have caused fluctuations in all results in Figure 16 due to TCP's congestion avoidance mechanism. The fluctuations are milder compared to the second experiment because the packet loss is still less frequent than that in a fading channel. Despite these fluctuations, a human observer did not perceive any variations in the played video's smoothness or image quality here, once again indicating the disconnect between the quantitative measure and the subjective evaluation of the streamed video. It suggests that although agile response of a streaming media application to dynamically changing networking condition gives better bandwidth utilization, the quality improvement by such responsiveness, in a human observer's opinion is scenario dependent.

As a final experiment, the mobility pattern was varied to create intermittent connectivity in the MANET. We found that if the route is broken for the duration of a few seconds, QStream will not be able to recover even though TCP connection has not yet timed out. In contrast, an *scp* session continued to work perfectly fine in the same situation.

## 6. RELATED WORKS

Wireless network analysis using emulation, simulation and physical testbeds has a rich history. For brevity, we restrict attention to existing emulation tools. Many existing emulation tools such as those in [31, 4, 25] require a one-to-one mapping between a wireless node and the host machine that emulates it, therefore limiting the scalability of such tools. MobiEmu [31] uses a centralized approach, which defines a master node and requires that all packets generated by emulated nodes pass through the master node. MobiEmu does not emulate wireless MAC and PHY layers and is intended for testing rather than performance evaluation of protocols or applications (quote from [31]). NISTNet [4] and the emulation tool presented in [25] generate delay, loss, and other characteristics of packets based on either simple statistical models, or traces of measurements from real wireless networks. This approach fails to capture the dynamic dependencies among physical channel characteristics (such

as interference), protocol (MAC, network and transport) behaviors, and application traffic patterns. Thus, they have limited capabilities in terms of modeling diverse network scenarios. A more recent emulation tool EMPOWER [32] has improved emulation scalability compared to the previously mentioned tools, because it maps multiple wireless nodes (each emulated on one ethernet device) onto a single host machine. However, EMPOWER does not model the physical layer of emulated wireless networks (quote from [32]).

A common characteristic in the above tools is that applications or protocols execute directly on machines with emulated radio device(s). In contrast, the emulation (rather hybrid simulation in a strict sense) tools presented in [28, 30, 33] integrates physical network nodes with simulated networks. In these tools, protocol implementations and application run on separate machines from those simulating all radio devices. Due to the propagation delay between the machines, the above tools cannot support seamless execution of a protocol implementation or application on simulated radio devices, as is possible in TWINE. The emulation testbeds presented in [8] [19] use hardware (radio signal attenuators or a FPGA based DSP engine) to emulate wireless channels, and significantly reduce the space requirements for constructing multi-hop wireless network scenarios. However, they require the use of physical radio devices and thus have limited flexibility in supporting evaluation of protocol and application with various emerging or futuristic radio technologies.

## 7. CONCLUSIONS

In this paper, we presented a high fidelity and efficient emulation framework called TWINE, that integrates emulation, simulation and physical components in a common framework, to support accurate and scalable evaluation of wireless networking systems. Our hybrid approach is particularly applicable to studying adaptive networking protocols and applications, that exploit cross-layer interactions to optimize system performance. Compared to existing evaluation tools and methodologies, the hybrid approach of TWINE has the following advantages: First, embedding the simulated physical and MAC layers directly into the

operating system enables evaluation of real applications and protocols. Second, the integration of simulation entities improves scalability of the testbed for large target network scenarios. Third, interfacing with real networks allows the user to directly leverage existing, partially deployed physical testbed and extend the performance evaluation in a larger-scaled networking context. Our experiment results show that TWINE has minimal overhead in the kernel, and the application throughput measured in TWINE closely matches the actual measurement in the equivalent physical testbed.

To demonstrate the utility of TWINE, two case studies were presented: a protocol to maintain fairness in mesh networks and an adaptive streaming media application. The case studies demonstrated the capabilities of TWINE to facilitate repeatable, flexible and efficient experimentations with protocol implementations and applications in complex network scenarios. The results from the case studies clearly demonstrate the benefit of the TWINE evaluation methodology, by identifying a mismatch between the performance of the protocol or application based on actual user experience versus its performance as measured using traditional network performance metrics such as application throughput. In particular, for the streaming media application, we showed that although variations in channel conditions and mobility patterns both had dramatic impact on the video quality when measured using traditional statistical metrics used by the research community, their impact when measured using a human observer's perception of the resulting video quality were substantially different.

## 8. ACKNOWLEDGEMENT

This work is funded by the National Science Foundation under NRT grant ANI-0335302 *WHYNET: Scalable Testbed for Next Generation Mobile Wireless Networking Technologies*. We would like to thank Professor Charles Krasic from University of British Columbia for his technical support to QStream [38] which is used in our case study.

## 9. REFERENCES

- [1] D. G. Andersen, D. Bansal, D. Curtis, S. Seshan, and H. Balakrishnan. System support for bandwidth management and content adaptation in internet applications. In *OSDI 2000*, San Diego, CA, Oct 2000.
- [2] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz. A comparison of mechanisms for improving TCP performance over wireless links. *IEEE/ACM Transactions on Networking*, 5(6):756–769, 1997.
- [3] G. Bianchi. Performance analysis of the IEEE 802.11 distributed coordination function. *IEEE JSAC*, 18(3):535–547, Mar 2000.
- [4] M. Carson and D. Santay. NIST Net: a linux-based network emulation tool. *ACM SIGCOMM Computer Communication Review*, 33(3):111–126, Jul 2003.
- [5] M. M. Carvalho and J. J. Garcia-Luna-Aceves. A scalable model for channel access protocols in multihop ad hoc networks. In *ACM MOBICOM'04*, Philadelphia, PA, Sept. 2004.
- [6] K. Chen, S. H. Shah, and K. Nahrstedt. Cross-layer design for data accessibility in mobile ad hoc networks. *Wireless Personal Communications*, 21(1):49–76, 2002.
- [7] M. Chiang. To layer or not to layer: Balancing transport and physical layers in wireless multihop networks. In *IEEE INFOCOM'04*, Hong Kong, China, Mar. 2004.
- [8] P. De, A. Raniwala, S. Sharma, and T. cker Chiueh. Mint: A miniaturized network testbed for mobile wireless research. In *IEEE INFOCOM'2005*, Miami, FL, March 2005.
- [9] P. de Cuetos and K. W. Ross. Unified framework for optimal video streaming. In *IEEE INFOCOM'04*, Hong Kong, China, Mar. 2004.
- [10] N. Feamster, D. Bansal, and H. Balakrishnan. On the interactions between layered quality adaptation and congestion control for streaming video. In *11th International Packet Video Workshop*, Kyongju, Korea, Apr 2001.
- [11] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *ACM SOSP'99*, Charleston, SC, Dec 1999.
- [12] R. Fujimoto. *Parallel and Distributed Simulation Systems*. John Wiley & Sons, 1999.
- [13] V. Gambiroza, B. Sadeghi, and E. W. Knightly. End-to-end performance and fairness in multihop wireless backhaul networks. In *ACM MOBICOM'04*, Philadelphia, PA, Sep 2004.
- [14] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste. Project aura: Towards distraction-free pervasive computing. *IEEE Pervasive Computing*, 1(2), 2002.
- [15] C. K. Hess, M. Roman, and R. H. Campbell. Building applications for ubiquitous computing environments. In *IEEE PerCom'02*, Zurich, Switzerland, Aug 2002.
- [16] IEEE Computer Society LAN/MAN Standards Committee. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *ANSI/IEEE Std 802.11, 1999 Edition*, 1999.
- [17] D. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7:404–425, 1985.
- [18] Z. Ji, J. Zhou, M. Takai, and R. Bagrodia. Scalable simulation of large-scale wireless networks with bounded inaccuracies. In *ACM MSWIM'2004*, Venice, Italy, Oct 2004.
- [19] G. Judd and P. Steenkiste. Repeatable and realistic wireless experimentation through physical emulation. In *HotNets'2003*, Cambridge, MA, Nov 2003.
- [20] U. Kozat, I. Koutsopoulos, and L. Tassiulas. A framework for cross-layer design of energy-efficient communication with QoS provisioning in multi-hop wireless networks. In *IEEE INFOCOM'04*, Hong Kong, China, Mar. 2004.
- [21] C. Krasic, J. Walpole, and W. chi Feng. Quality-adaptive media streaming by priority drop. In *NOSSDAV'03*, Monterey, CA, Jun 2003.
- [22] D. A. Maltz, J. Broch, and D. B. Johnson. Quantitative lessons from a full-scale multi-hop wireless ad hoc

- network testbed. In *IEEE WCNC'00*, Chicago, IL, Sep 2000.
- [23] R. A. Meyer and R. Bagrodia. Improving lookahead in parallel wireless network simulation. In *MASCOTS'98*, Montreal, Canada, Jul 1998.
  - [24] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker. Agile application-aware adaptation for mobility. In *ACM SOSIP'97*, Saint-Malo, France, Oct 1997.
  - [25] B. D. Noble, M. Satyanarayanan, G. T. Nguyen, and R. H. Katz. Trace-based mobile network emulation. In *ACM SIGCOMM97*, Cannes, France, Sep 1997.
  - [26] T. S. Rappaport. *Wireless Communications: Principles and Practice*. Prentice Hall, 2nd edition, 2001.
  - [27] U. Saif and J. M. Paluska. Service-oriented network sockets. In *ACM MOBISYS'03*, San Francisco, CA, May 2003.
  - [28] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. In *USENIX OSDI'02*, Boston, MA, Dec 2002.
  - [29] K. Walsh and E. Sirer. Staged simulation: A general technique for improving simulation scale and performance. *ACM Transactions on Modeling and Computer Simulation*, 14(2), Apr 2004.
  - [30] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *USENIX OSDI'02*, Boston, MA, Dec 2002.
  - [31] Y. Zhang and W. Li. An integrated environment for testing mobile ad-hoc networks. In *ACM MOBIHOC'02*, Lausanne, Switzerland, Jun 2002.
  - [32] P. Zheng and L. M. Ni. Empower: A network emulator for wireline and wireless networks. In *IEEE INFOCOM'03*, San Francisco, CA, Apr 2003.
  - [33] J. Zhou, Z. Ji, M. Takai, and R. Bagrodia. MAYA: Integrating hybrid network modeling to the physical world. *ACM Transactions on Modeling and Computer Simulation*, 14(2), Apr 2004.
  - [34] GloMoSim. <http://pcl.cs.ucla.edu/projects/glomosim/>.
  - [35] Kansas University Real-Time Linux. <http://www.ittc.ku.edu/kurt/>.
  - [36] ns-2. <http://www.isi.edu/nsnam/ns/>.
  - [37] OPNET. <http://www.opnet.com/>.
  - [38] QStream Project. <http://www.qstream.org>.
  - [39] QualNet User Manual v3.7. Scalable Network Technologies, Culver City, CA. <http://www.qualnet.com>.
  - [40] VLC Media Player. <http://www.videolan.org/vlc/>.