Computer Communications 73 (2016) 99–107

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Wireless Networking Testbed and Emulator (WiNeTestEr)

J.D. Beshay [a,*], K.S. Subramani [a,*], N. Mahabeleshwar [a], E. Nourbakhsh [a], B. McMillin [a], B. Banerjee [a], R. Prakash [a], Y. Du [b], P. Huang [b], T. Xi [b], Y. You [b], J.D. Camp [b], P. Gui [b], D. Rajan [b], J. Chen [c]

[a] The University of Texas at Dallas, Richardson, TX, USA
[b] Southern Methodist University, Dallas, TX, USA
[c] The University of Arizona, Tucson, AZ, USA

ARTICLE INFO

Article history:
Available online 18 August 2015

Keywords:
Wireless
RF
Channel emulation
Testbed

ABSTRACT

Repeatability, isolation and accuracy are the most desired factors while testing wireless devices. However, they cannot be guaranteed by traditional drive tests. Channel emulators play a major role in filling these gaps in testing. In this paper we present an efficient channel emulator which is better than existing commercial products in terms of cost, remote access, support for complex network topologies and scalability. We present the hardware and software architecture of our channel emulator and describe the experiments we conducted to evaluate its performance against a commercial channel emulator.

1. Introduction

The ability to conduct repeatable experiments is crucial to the development of wireless devices and protocols. Most of the time, researchers make simplifying assumptions about the nature of their test environment and the experiment control procedures. However these assumptions do not always hold good [1] making it harder to isolate device/protocol performance from environmental effects.

The current wireless networking testbeds use a wide range of approaches, varying from fully software-simulated testbeds like ns-3 [2] to real hardware running in Faraday cages. However, the two extremes have their respective limitations. Simulators are easy to implement but they are limited by the models provided in software. Different simulators might yield different results depending on the assumptions and simulation techniques used [3]. On the other hand using hardware in Faraday cages is not always affordable and it is not flexible enough to test complicated scenarios involving mobility or signal reflections. The balance between the two extremes is provided by channel emulators.

The simplest form of channel emulation is achieved by replacing the antennae of two communicating transceivers with shielded RF cables and a programmable attenuator in between. An example of an emulator based on this design is ASSERT [4], developed by our group a few years back. By increasing (or decreasing) the attenuation it simulates the transceivers moving apart (or moving closer). The rate at which attenuation is varied corresponds to the relative speed of the transceivers. This scenario covers the fading effect of wireless channels.

However, wireless transmissions are not only affected by fading but also by multiple reflections of the same signal from obstacles in the surrounding environment (multi-path effects). Devices with multiple antennae (MIMO) exploit these signal reflections to achieve better throughput. Accurate emulation of multi-path effects requires creating multiple copies of the transmitted signal with different time delays (phases). This cannot be achieved by attenuators. Instead it is done by digitizing the signal and manipulating it using digital signal processing (DSP). The resulting digital signal is converted back to analog.

Commercial solutions exist for emulating environments with multi-path effects [5,6]. However they are prohibitively expensive and are limited to simulating environments with 2 pairs of devices or less. Commercial channel emulators are thus impractical for researchers who are usually cost-constrained and interested in experiments that involve the interaction (and interference) between multiple devices with a higher degree of connectivity. As a result, researchers sought to develop their own channel emulators that could achieve multipath effects for multiple devices while maintaining relatively low cost. An example is the work in [7] which uses a single field programmable gate array (FPGA) to simulate a 90 MHz-wide environment for up to 15 devices operating in the 2.4 GHz ISM band. The design in [7] cannot scale due to the FPGA resource constraints.

In this paper, we present our Wireless Networking Testbed and Emulator (WiNeTestEr) which is designed to emulate 100-MHz-wide

* Corresponding authors.
E-mail addresses: joseph.beshay@utdallas.edu (J.D. Beshay), kiruba.subramani@utdallas.edu (K.S. Subramani).

http://dx.doi.org/10.1016/j.comcom.2015.08.007
0140-3664/

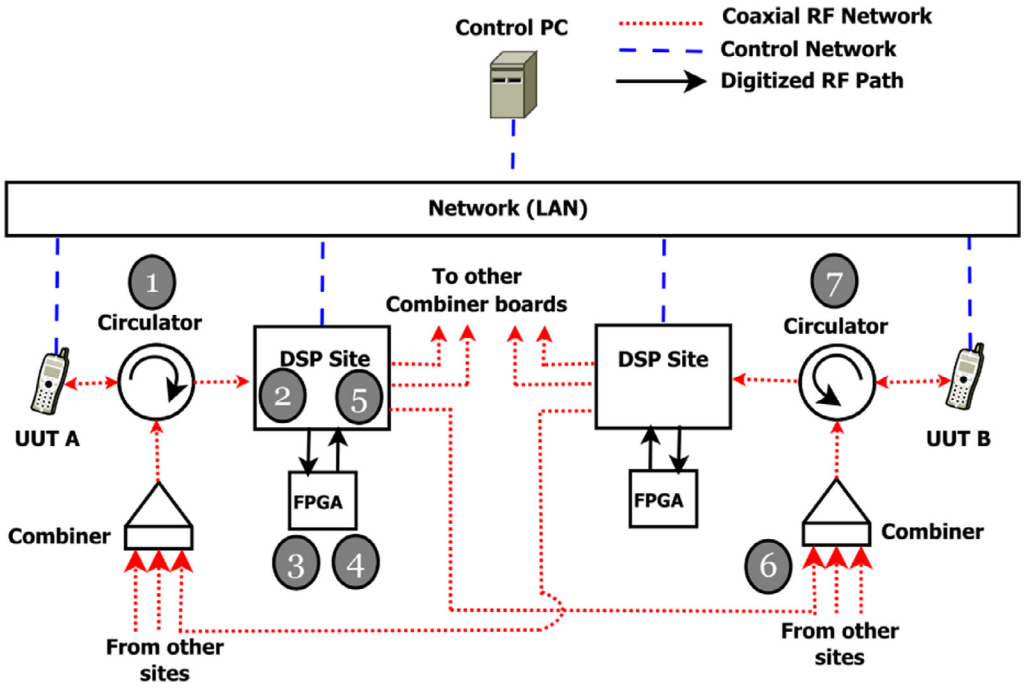

Fig. 1. A simple topology of two UUTs connected by a bidirectional link.

environments with multipath in the 2.4 GHz ISM band. The main features of WiNeTestEr are:

- Scalability: the system uses a distributed channel emulation architecture running on multiple FPGAs so it can potentially scale to hundreds of nodes,
- remote access: a device control protocol allows the system to run experiments without onsite-operator intervention,
- concurrent experiments: the modular design allows for multiple independent experiments to be run by different users at the same time,
- multi technology support: experiments can be performed on different technology devices operating in their native frequencies (Bluetooth, WiFi, Zigbee, etc.). The design is flexible to allow adding more frequency bands in the future with minimal changes,
- full duplex channels: the channel between two devices is full duplex with support for non-reciprocal channel conditions i.e. the signal can experience a certain environment in one direction and a different one in the other.

WiNeTestEr is our followup to ASSERT [4]. ASSERT performs channel emulation in the 900 MHz ISM band using attenuation. Attenuators are used to control the transmitted signal strength to emulate the required channel conditions (deep fade, slow fade) but it cannot emulate multi-path effects. WiNeTestEr was developed to bridge this gap.

2. System overview

WiNeTestEr is a DSP-based channel emulator. The system is built around the following core function: RF signals generated by a transmitter are digitized, processed in the digital domain, converted to analog and delivered to the receiver. The processing phase can apply attenuation and amplification to emulate fading, or create multiple copies of the signal with different signal levels and phases to emulate multipath effects.

WiNeTestEr can be logically broken into three parts: units under test (UUTs), DSP sites and the Control PC. UUTs are the devices with RF transceivers. Signals traveling between pairs of devices are manipulated to provide the desired channel conditions for the experiment being conducted. The device antennae are removed and replaced with coaxial cables connected to the DSP sites.

DSP sites, we call them sites for short, are the heart of channel emulation. A site takes the signal transmitted by a UUT, manipulates it in the digital domain using an FPGA to apply the specified channel conditions and delivers the resulting analog signal to the receiving UUT via a coaxial RF cable. This constitutes a uni-directional link between two UUTs. Bidirectional communication is achieved by using multiple sites. The interconnection of UUTs in the emulated environment is realized by coaxial cables between sites. This implies that a transciever needs to have at least two coaxial cables connected to its antenna port; one for outgoing and the other for incoming signals. The higher the degree of connectivity the more cables that need to be connected to the UUT. This is achieved by using circulators and combiner boards. A circulator has three terminals; signal from terminal #1 goes to #2, #2 to #3 and #3 to #1. A combiner board takes a number of input signals over different coaxial cables and combines them into one output signal over a single coaxial cable. The desired connectivity is achieved by connecting the UUT antenna port to terminal #1 of the circulator. The transmitted signal comes out on #2 which goes to the site. The combiner board takes all received signals and passes them to #3 which comes out on #1 and goes into the UUT. This setup is illustrated in Fig. 1.

The Control PC is what glues the whole system together. WiNeTestEr software is highly distributed with different subcomponents running on UUTs, sites and a remote GUI running on user machines. These subcomponents are all connected to the Control PC over an IP network. The Control PC is responsible for:

- Keeping track of the status of all UUTs and sites.
- Keeping track of the environment topology (cable interconnections between sites and UUTs).
- Accepting requests to run experiments from users.
- Coordinating sites and UUTs to run the desired experiments.
- Collecting results and delivering them to the user.

Another approach to understanding the design of WiNeTestEr is to consider the hardware and software aspects separately. Fig. 1 shows how the hardware is physically connected to emulate a bidirectional

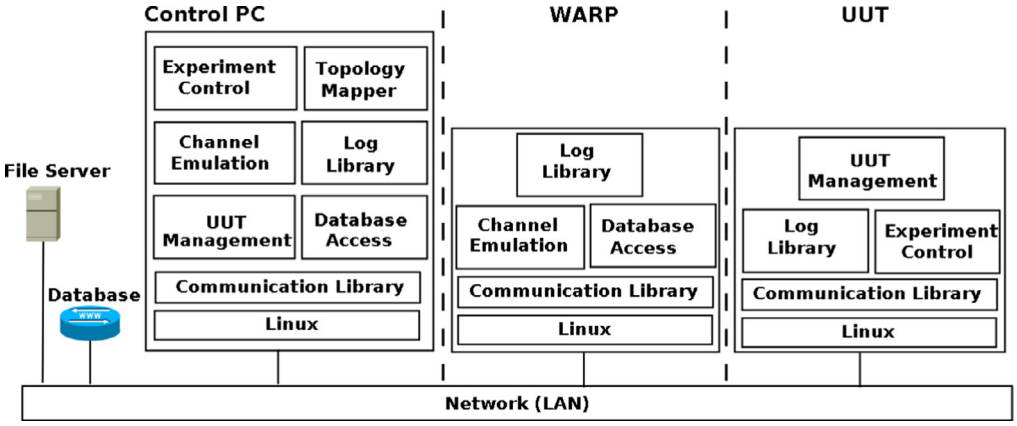

Fig. 2. Software architecture.

link between two UUTs. The numbers on the figure correspond to the steps involved in emulation of a single direction of the channel:

1. UUT *A*'s output is connected to UUT *A*'s site through a wideband duplexer (circulator).
2. Site digitizes the signal and passes it to the FPGA.
3. FPGA makes copies of the signal for each of the three outputs.
4. FPGA processes the digital signal based on the channel conditions set by the control PC for each output.
5. Site converts the signals back to analog and outputs each of them on its port.
6. The equivalent site output is connected to UUT *B*'s combiner board through a coaxial cable.
7. Combiner board passes the combined signal of all of its inputs to UUT *B* through the circulator.

WiNeTestEr software runs on different flavors of Linux deployed on UUTs, sites and the Control PC. The devices are connected over a local area network and they share access to database and file servers. Related subsets of functions are packaged in distinct executable files that run on the relevant devices, we use the word *slices* to refer to these subsets. The software architecture of the system is shown in Fig. 2.

In the following sections we take a closer look at the components of WiNeTestEr , how they are designed and how they work.

3. DSP sites

We start our discussion with the site since that is where channel emulation happens. WiNeTestEr emulates a 100 MHz-wide environment which requires very high processing power to manipulate in realtime. We chose a development board by Khattab et al. [8] as the basis for the site. The WARP development board version 2.2 has a Xilinx Virtex 4 FPGA, a PowerPC processor, a RAM slot, a CompactFlash card slot, Ethernet and serial ports. We designed an RF board to handle the analog-to-digital and digital-to-analog conversion of the signal. Two RF boards are mounted on one WARP board using four daughtercard slots with 124 pins each. The large number of pins was crucial to our choice of the board so the digitized 100 MHz-wide signal can flow to and from the FPGA in realtime. A schematic of the site is shown in Fig. 3.

We run Linux on the PowerPC processor and use it to control the FPGA and coordinate the execution of experiments with the rest of the system.

3.1. RF board

The RF board consists of two signal chains for down-conversion and up-conversion. Each component on the board can be broadly classified as being part of one of the two chains. The down-conversion

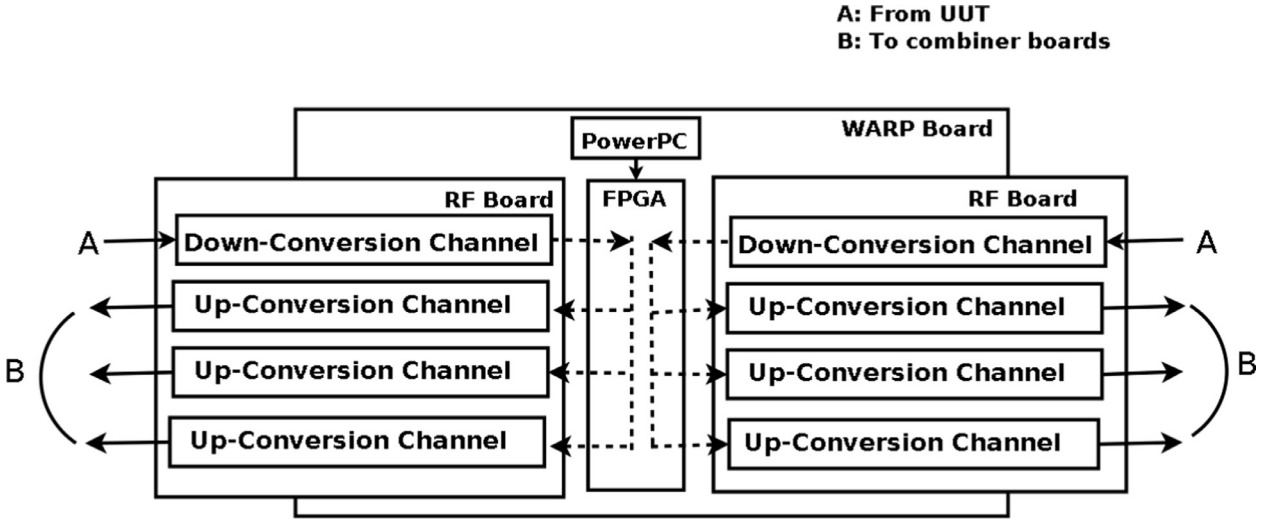

Fig. 3. DSP site schematic.

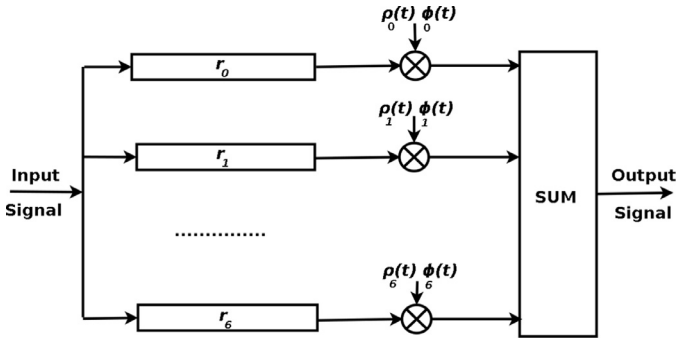

Fig. 4. Multi-tap fading generator.

chain consists of digital step attenuator (DSA), quadrature demodulator, variable gain amplifier (VGA) and Analog to Digital Converter (ADC). The up-conversion chain consists of Digital to Analog Converter (DAC), VGA and quadrature modulator. Clock circuit present on each board helps in synchronizing data conversion operations. The clock circuit also provides reference input to Phase Locked Loop (PLL) which is used in generating Local Oscillator (LO) signal for down-conversion and up-conversion operations. As shown in Fig. 3, the FPGA creates three identical copies of the original signal transmitted by the UUT. These signals can be independently modified as instructed by the PowerPC processor to emulate the desired environment. Each signal corresponds to an output port that will eventually be connected to a UUT via its combiner board.

The architecture of the RF board is shown in Fig. 5. RF interface to the board is provided by means of SMA jacks and coaxial cables. The use of coaxial cables minimizes interference among signals thereby increasing the robustness and repeatability of experiments. The RF board supports input signal power in the range of −25 dBm to +26 dBm and an output power of −20 dBm to −110 dBm. To achieve this target specification, a maximum attenuation of 136 dB is required from the design. This value is distributed among multiple components on the board such as DSA, VGA and FPGA.

DSA performs the first stage of attenuation on the input signal. Owing to high signal strength of the input, there is a very high chance of it saturating the quadrature demodulator. Placing a DSA at the start of the chain helps restrain signal strength to reasonable values. Modern wireless standards use complex modulation schemes such as QPSK or QAM to achieve higher speed and lower error rate. To support these standards, broadband quadrature modulator and demodulator with good performance specifications have been used in the up-conversion and down-conversion circuitry. For best-case performance of quadrature modulator, signal properties of In-phase (I) and Quadrature (Q) components need to be perfectly matched. In other words, a mismatch between these two components in terms of DC offset, gain and phase results in LO leakage and sideband issues thereby degrading the quality of output signal. Baseband VGAs are used to offset amplitude mismatches of I/Q signals. To obtain very low value of sideband signal, both gain and phase need to be carefully adjusted in the FPGA. Also, in down-conversion chain, VGA helps to amplify or attenuate the baseband signal to meet the dynamic range requirements of ADC. In WiNeTestEr since the baseband signal has a bandwidth of 100 MHz, a sampling clock of 200 MHz is needed for the data converters to avoid aliasing. The clock circuit consists of a voltage controlled crystal oscillator (VCXO) as the clock source with clock distribution/divider IC's providing identical clocks to data converter chips (ADC and DAC).

Resolution of data converter ICs has significant impact on the performance of the system. During down-conversion, the demodulator outputs I/Q signals, requiring separate ADC to digitize each signal. Two 12-bit ADCs were used in the design to satisfy the signal to noise ratio (SNR) requirement of the receiver. Up-conversion, on the other hand, handles a wide range of output signal strength. Also, each RF board houses 3 up-conversion chains, with each chain having its own I/Q signals. Hence a dual 16-bit DAC was used on each chain to convert the signal back to analog domain.

Fig. 6 shows the RF front end that was designed for WiNeTestEr. A single ended signal from the circulator is connected to the first SMA jack marking the input to the board. The other three SMA jacks correspond to the output from each up-conversion chain. The RF board supports three different power sources: external adapter, WARP FPGA board and screw terminal. Selected source is regulated using low-dropout regulators (LDO) to generate 5 V, 3.3 V and 1.8 V analog and digital supplies. Utmost care has been taken to reduce supply noise with the help of global and local decoupling capacitors and

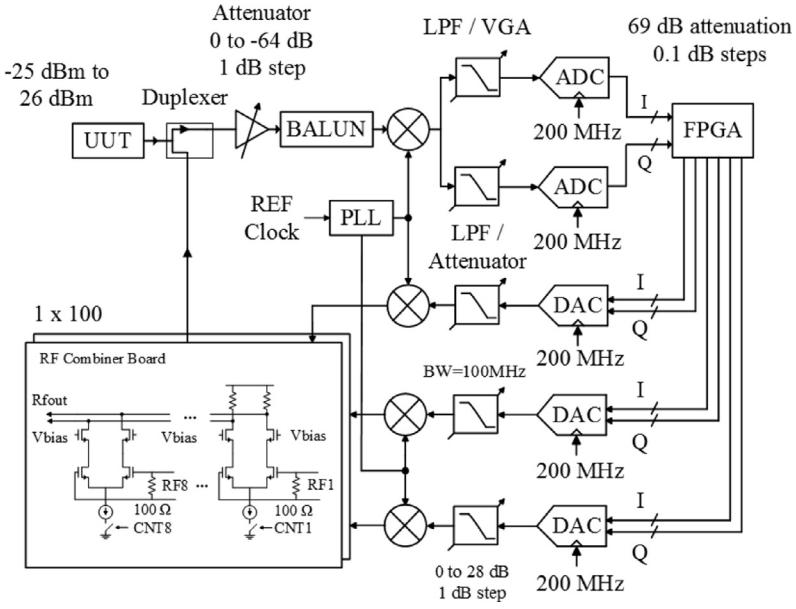

Fig. 5. RF board architecture.

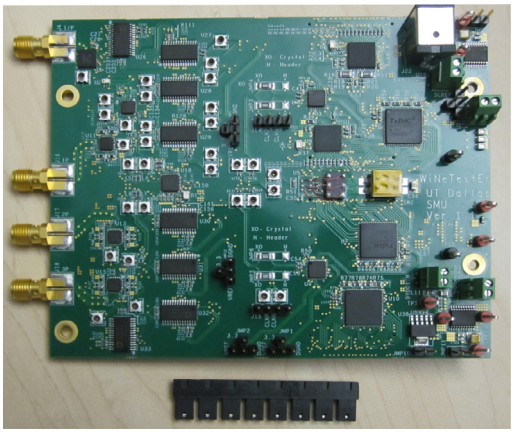

Fig. 6. RF board for WiNeTestEr.

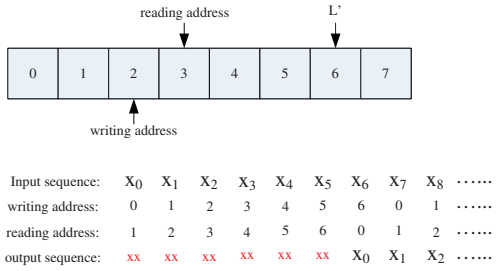

Fig. 7. Address operation for generating arbitrary number of clock-cycle delay.

ferrite beads. Careful floor planning, layer management and termination techniques have been followed to obtain the best performance from the board.

3.2. FPGA

The digitized baseband signal is forwarded from the RF board to the FPGA where channel emulation is implemented. An emulated channel is characterized by the number of copies of the transmitted signal that reach the receiver, their phases and their attenuation levels. This way the channel can represent a multipath environment where the signal reaches the receiver via multiple reflections on surrounding obstacles. A copy of the signal along with its phase and fading parameters is called a tap.

A block diagram for our channel emulation method is shown in Fig. 4. Each tap has a delay unit(r_x) controlled by a user-defined value, followed by scaling based on fading parameters ($\rho_x(t)\phi_x(t)$) where $x \in (0,6)$. The resulting signals from all taps are added together to emulate a multi-tap fading channel.

For each tap, there is a separate signal delay and a fading parameter. We use dual-port RAMs (random access memory) to realize the user-specified time delay. Assume the memory depth allocated to each tap is L, which depends on τ_{max}, the maximum delay that the emulator aims to support. Let f_s denote the sampling clock frequency of the signals. Then, we can express L as follows:

$$L = \lceil \tau_{max} f_s \rceil \quad (1)$$

The users are allowed to configure the tap delay for each tap with a range of $0 \le \tau \le \tau_{max}$. The tap delay, τ, is then translated to the number of clock cycles to be delayed, denoted by L', with a value of $\lceil \tau f_s \rceil$. By manipulating the reading and writing address of the dual-port RAM, we can realize the delay function, as shown in Fig. 7. The original data is written to one port of the RAM and the delayed data is read out from the other port of the RAM. The reading address always leads the writing address by one. Whenever either the reading address or the writing address reaches L', they restart from the address of zero.

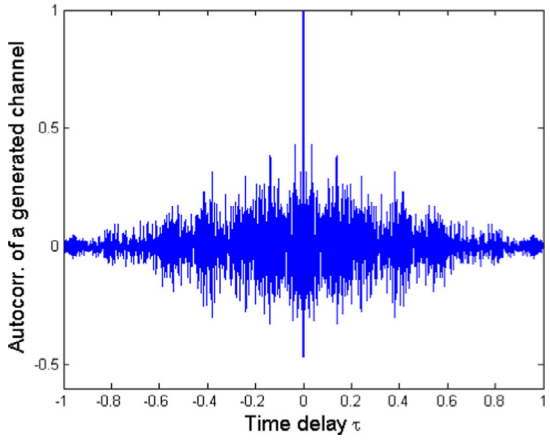

Fig. 8. Autocorrelation of a fading channel generated on WiNeTestEr DSP site.

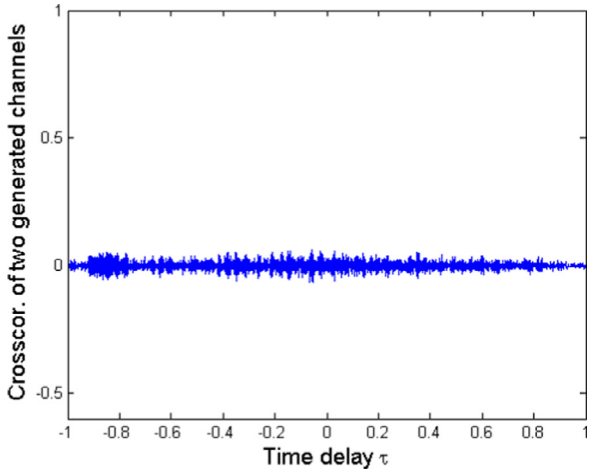

Fig. 9. Crosscorrelation of two fading channels generated on WiNeTestEr DSP site.

In our implementation, we support a maximum of 7 taps, with each tap has a maximum delay of 1024 clock cycles (1.024 μ s with a sampling clock of 100 MHz).

For the channel emulation in a wireless network, if there concurrently exists more than one communication link, multiple fading channels are needed. Therefore, WiNeTestEr board needs to be equipped with the ability of generating multiple fading channels simultaneously. In real communication environments, the fading channels are different from each other which means they are statistically independent.

To generate the independent fading channels simultaneously, we consider the method proposed in [9]. According to the method in [9], fading channels is generated by summing the phase-frequency-dithered sinusoids (SODS). Different from the conventional sum-of-sinusoid method, the SODS method add random disturbances on the frequency and the initial phase of each sinusoid. The values of the disturbances are much smaller than the frequency and the initial phase respectively. Using the SODS method, we can generate the independent fading channels on our board.

Figs. 8 and 9 illustrate the autocorrelation and the crosscorrelation function of the fading channels which are generated via the fixed-point computation on MATLAB. The maximum value of the autocorrealtion is normalized to unit one, and the crosscorrelation is also normalized via dividing it by the maximum value of the autocorrelation. From Figs. 8 and 9, compared with the

autocorrelation, the value of the crosscorrelation function is in a small range. Thus, the independence of the generated fading channels is achieved.

In WiNeTestEr , summation of sinusoids (SOS) method [10,11] has been used to generate fading channel. This method of channel generation for emulation has been widely investigated in the past [9,12,13]. However, existing fading channel emulators demand large memory resources to generate channels, which degrades the scalability. In WiNeTestEr a single FPGA is required to process two different input signals and produce up to 6 outputs (two RF boards) with different channel conditions. We proposed a novel structure to implement SOS based channel generation specifically for this purpose [14]. With this structure, the generation of one Rayleigh channel consumes only 1 unit memory resource ($\sim$ RAMB16) of the 376 available in the FPGA.

Besides reducing memory requirements, this work has also optimized word length selection and channel data update rate. Intuitively, larger bit width generates higher channel accuracy at the cost of hardware resources. Similar tradeoff exists between channel data generation rate and accuracy over time domain. We chose to optimize the two terms (bit width and update rate) aiming at minimizing hardware resources while maintaining a certain channel accuracy level. Our technique is discussed in more detail in [14]. With the reduction of memory consumption and optimization on the two terms, scalability is drastically improved in WiNeTestEr.

3.3. Linux

The resources saved by the new channel emulation method made it possible to fit the necessary components on the FPGA to be able to run Linux. WARP board has a 2GB RAM chip that can be accessed by the PowerPC processor. Unfortunately the processor does not have access to persistent storage, the board has a CompactFlash slot which is only connected to the System ACE chip [15].

Booting Linux on the board involves two steps; writing the FPGA bitstream (system.bit file) to the FPGA and loading the Linux Kernel Image to the PowerPC processor. The bitstream and the kernel image are combined into the system.ace file and placed it on a FAT12 formatted CompactFlash card. The SystemACE chip reads the file and writes its components to the FPGA and the PowerPC processor. We configured the kernel to use an NFS share as its root filesystem to overcome storage limitations and centrally manage the root filesystem for all sites. The NFS rootfs mounting details are provided to the board through the DHCP server.

We use Linux 3.0 and our root filesystem is based on ELDK 4.2 [16]. A number of libraries required for our software slices had to be cross-compiled, including The Boost C++ Libraries and MySQL Connector for C++. This was achieved using the ELDK 4.2 ppc4xx toolchain.

4. Control PC

The Control PC is the where WiNeTestEŕs control network meets the rest of the world. On one side it accepts connections from users to submit experiments and collect results. On the other side it connects to the different software slices running on the sites and UUTs to coordinate the execution of experiments. An experiment submitted by the user includes the requested number of UUTs and the set of links between them. For every link the user defines the channel conditions to be emulated by WiNeTestEr. The user may also specify different binaries to be executed on the UUTs. In the following sections, we discuss the different software slices that work together on the Control PC to fulfill the user experiment.

4.1. Diagnostics

The diagnostics slice maintains information about running experiments as well as the hardware resources available in the system (sites, UUTs and cable connections). It communicates with corresponding slices running on sites and UUTs to monitor their status and update the database accordingly.

Detecting cable connections is more complicated since WiNeTestEr sites do not have built-in rssi (signal strength) meters or signal generators. The diagnostic slice has to coordinate between sites and UUTs to detect which UUT is connected to which site's input and also the virtual topology between sites. A directed link between two UUTs in WiNeTestEr is captured by the tuple <UUT A, site, output number, UUT B>. An outline of the algorithm is shown in listing 1. The functions performed by the UUT are part of the UUT Management protocol described later.

Algorithm 1: Topology (physical cable connection) detection algorithm.

```
Data: U =set of available UUTs, S =set of available sites
Result: UUTMap<UUT, site>, List<DirectedLinks>
UUTMap ⟵ ∅;
DirectedLinks ⟵ ∅;
put all UUTs in listen mode;
set high attentuation on all outputs of all sites;
for u ∈ U do
    uutMapped ⟵ false;
    put u in broadcast id mode;
    for s ∈ S do
        for each output o ∈ s do
            set 0 attenuation on output o;
            for v ∈ U' = U − u do
                retrieve the set of heard UUT id's H;
                if H ≠ ∅ then
                    /* H is a singleton. H = {u}        */
                    if (u : s) ∉ UUTMap then
                        add (u : s) to UUTMap;
                        uutMapped ⟵ true;
                    end
                    add <u, s, o, v> to DirectedLinks;
                    break;
                end
            end
            set high attenuation on output o;
        end
        if uutMapped then
            break;
        end
    end
    put u in listen mode;
end
```

4.2. Topology mapper

The topology mapper slice is responsible for selecting the set of UUTs to run the experiment. UUTs selected along with their associated sites must have the physical connections to embed the topology specified by the user. This problem is an instance of Directed Subgraph Isomorphism. Given the physical system graph $G = (V, \overrightarrow{E})$ and an experiment topology $H = (V', \overrightarrow{E'})$, the goal is to find a subgraph $I = (V'', E'')$ of G for which the following function is defined $f : V'' \longrightarrow V'$ such that $(v_1, v_2) \in E'' \Leftrightarrow (f(v_1), f(v_2)) \in E'$. This problem is NP-Complete even with undirected edges [17]. In WiNeTestEr we use an approximation algorithm with worst-case complexity $O(n^3)$ for a random topology and $O(n^2)$ or $O(n)$ for common topologies like linear graphs and star graphs. The algorithm is a result of the work in [18].

The success of the topology mapper is dependent on the availability of links in the physical topology. We collected the usage reports from our previous testbed ASSERT to identify the most common experiment topologies. A thorough study of the possible physical topologies was conducted in [18]. A number of topologies of different sizes are proposed to enable embedding various experiment topologies and also to maximize the number of concurrent experiments.

4.3. Channel emulation

Once the topology mapper has decided on the physical sites and UUTs for an experiment, the channel emulation slice on the Control PC passes the user specified channel conditions for each link to the channel emulation slice on corresponding site. The slice provides a set of preconfigured environments for different mobility patterns as defined in [19]. Advanced users are given the option to create their own environments from scratch by defining the parameters for individual taps.

The links in a WiNeTestEr experiment are not reciprocal due to the design of the system. Each direction of a link is handled by a different site and thus can potentially have different channel conditions. The freedom is given to the user to configure the environment on each direction separately. However, when a reciprocal channel is desired, the channel emulation slice sends the parameters to the site along with a future timestamp to apply them. The sites are locally synchronized with the Control PC to sub-millisecond accuracy.

4.4. UUT management

UUT management is essential for running fully automated experiments. We developed a protocol that allows the Control PC to perform the following actions on the UUT:

- Discover available UUTs.
- Transfer executable images. These are the different binaries UUTs will run to perform the experiment.
- Begin experiment execution.
- Query experiment status.
- Terminate experiment.
- Collect logs.

Implementing this protocol on the UUT is optional. We have implemented it on ARM boards running Linux like Raspberry Pi and BeagleBone Black which we currently use as UUTs in WiNeTestEr. Users wishing to use other UUTs can either implement the protocol so the Control PC can talk to them directly or just use WiNeTestEr to provide the environment and manually start and stop their devices through an out-of-band method. Resource constrained UUTs such as sensor nodes can provide a UUT management proxy that receives commands from the UUT management slice on the Control PC and forward them to the actual nodes by other means.

5. Using the system

A GUI application has been developed for the end user to run experiments. This is a standalone application which authenticates the user and allows them to create, start and abort experiments. It is also used to load any UUT images and collect the experiment logs. The user can create the experiment topology using the GUI, selecting different UUTs and specifying all the communication constraints among the UUTs. This application communicates only with the Control PC. A screenshot of the GUI is shown in Fig. 10. As shown in the figure, the user can create different types of UUTs, communication links between them and specify experiment parameters. This UUT topology can be saved to disk and used whenever the experiment needs to be repeated. The GUI can also be used to view properties of the testbed like physical topology of the UUTs, available/allocated UUTs for experiments etc.

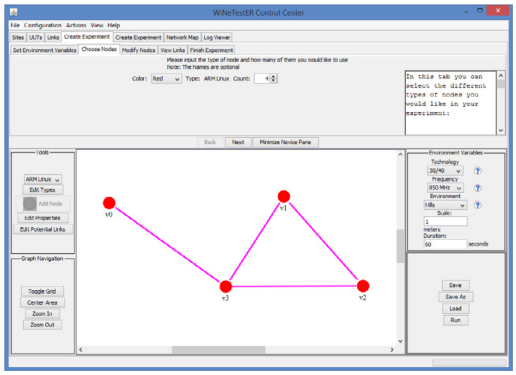

Fig. 10. A screenshot of the GUI showing a topology of four UUTs.

Table 1
Environment 1 channel parameters.

Doppler	1 Hz		
	Tap_Delay (ns)	Tap_Gain (db)	K-Factor (db)
Tap 1	0	0	−99

6. Experimental validation

The performance of the novel implementation of fading channels on FPGA is thoroughly discussed in [14]. In this paper, we focus instead on comparing the performance of WiNeTestEr as a complete system. We compare WiNeTestEr's channel emulation performance with a commercial channel emulator, Azimuth ACE MX MIMO [5]. It is one of the state-of-the-art channel emulators used by industry and academia to test complex wireless protocols. We chose not to perform any over-the-air experiments due to the difficulty in controlling the multi-path parameters. Even a seemingly simple environment like an open-air football field will have several taps (paths taken by different signal reflections).

We used two Ubiquiti SR-71 Cardbus WiFi adapters (Atheros AR9160 chipset) as UUTs. The cards were connected to two laptops running Linux 3.2 which includes the ath9k driver. Both cards were set to join an adhoc network operating on channel 14 (center frequency 2484 MHz) as per the IEEE 802.11 PHY/MAC standard for the 2.4 GHz ISM band. The basic rate of the adhoc network was fixed to 36 Mbps to avoid the results being affected by the driver's auto-rate algorithm. We could have picked another value for the fixed rate. However, we found 36 Mbps to provide a good balance between sensitivity to different channel conditions and the ability to gracefully degrade in performance as the channel worsens.

An experiment starts by the cards joining the adhoc network. Once the cards associate, we use Iperf [20] to send UDP packets carrying a 1470 byte payload from one machine to the other for 4 min while recording the average throughput achieved every second. At the end of the experiment, Iperf reports the number of packets lost during the session.

We defined four environments (channel conditions) each with a different number of taps. Table 1 shows the emulation parameters for Environment 1. It represent the basic case of having a single copy of the signal propagating through the channel. Environments 2, 3 and 4 use the top 2, 3 and 4 taps (respectively) of the ITU Vehicular A Channel Model [19] shown in Table 2. The model uses a Doppler value of 184 Hz which results in the signal fading in a way roughly equivalent to that experienced by a vehicle traveling at 80 km/hr.

For each environment, we ran the experiment through both WiNeTestEr and Azimuth and recorded the throughput results. Figs. 11 and 12 show the results obtained from WiNeTestEr and Azimuth, respectively.

Table 2
ITU Vehicular A channel model.

Doppler	184 Hz		
	Tap_Delay (ns)	Tap_Gain (db)	K-Factor (db)
Tap 1	0	0	−99
Tap 2	310	−1.0	−99
Tap 3	710	−9.0	−99
Tap 4	1090	−10.0	−99
Tap 5	1730	−15.0	−99
Tap 6	2510	−20.0	−99

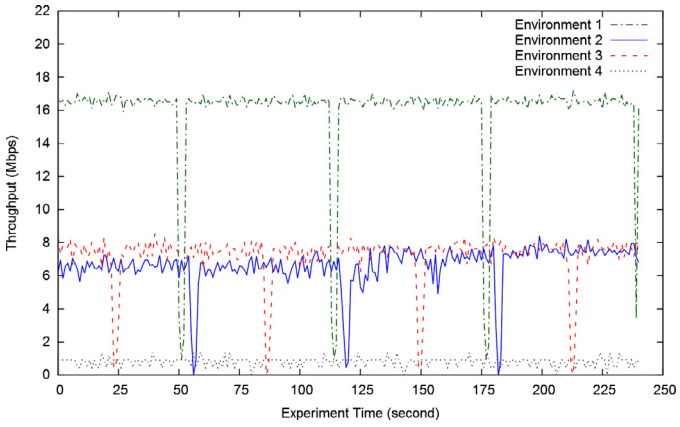

Fig. 11. Throughput vs. time plot for the experiments through WiNeTestEr.

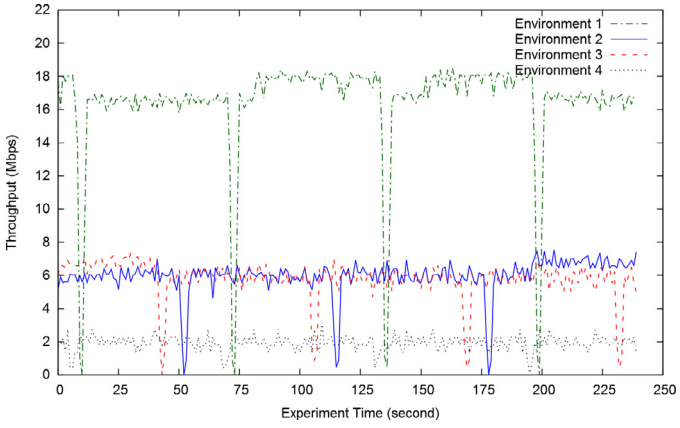

Fig. 12. Throughput vs. time plot for the experiments through Azimuth.

6.1. Results analysis

The results show a dip in the throughput roughly every minute. This happens with both WiNeTestEr and Azimuth. To isolate the issue, we ran the experiment over-the-air in an indoor interference-free environment. The cards experienced the same dips in throughput. While it would be interesting to dig deeper into this observation and find the root cause, it is out of the scope of this paper. We attribute these dips in throughput to the hardware or the driver of the adapters.

Both WiNeTestEr and Azimuth results share a trend of decreasing throughput with the increase of the number of taps. This matches the expectations we have for different environments especially environments 2,3 and 4 which emulate high mobility.

The results however are not identical. Despite sharing the same trend, WiNeTestEr and Azimuth achieve different throughput for the same channel parameters. We attribute this to the different level of control we have over the low level multi-path parameters in Azimuth compared to WiNeTestErEach tap in the multi-path parameters consists of a set of components. Unlike WiNeTestEr Azimuth does not provide an interface to specify the value of each of these components. WiNeTestEr allows for repeating experiments under identical multi-path parameters even in terms of individual tap components. It is worth noting that specifying these components is optional and WiNeTestEr provides a set of default components for unfamiliar users.

7. Conclusion and future work

WiNeTestEr is a scalable and cost-effective channel emulator that allows researchers to perform accurate, repeatable and complex experiments (topology, multipath effects). The distributed design of the system is evident in both the hardware and the software architecture. The hardware consists of UUTs, DSP sites and combiner boards, interconnected to form the base topology. The software consists of a central Control PC that communicates with embedded Linux running on sites to control the FPGA that manipulates the signal using DSP, and UUT to automate the execution of experiments. The paper discussed the different design and implementation aspects of each of the components.

Experimental results show that performance of WiNeTestEr is comparable to commercially available solutions. Being remotely accessible makes it an efficient alternative for researchers in academia and the industry alike.

WiNeTestEr is currently limited to operating in the 2400–2500 MHz band due to the RF board design. We intend to address this in the future revision of the RF board to be able to emulate environments in the range of 700 MHz to 6 GHz. Old RF boards will be swapped with the new ones without requiring any modification to the rest of the system. This will allow testing a wider range of devices and technologies (GSM, CDMA, LTE, 5 GHz Wifi, etc.).

Acknowledgment

We thank Krypton for their input in the design of RF and Combiner boards. We also thank Amba Kalur for her support during the initial phases of the RF board design. This work is supported in part by the National Science Foundation MRI program under grant numbers 1040422 and 1040429.

References

[1] Ryan Burchfield, Ehsan Nourbakhsh, Jeff Dix, Kunal Sahu, S. Venkatesan, Ravi Prakash, RF in the jungle: effect of environment assumptions on wireless experiment repeatability, in: Proceedings of IEEE International Conference on Communications, IEEE, 2009, pp. 1–6.
[2] NS-3, Network Simulator 3, http://www.nsnam.org/.
[3] David Cavin, Yoav Sasson, André Schiper, On the accuracy of MANET simulators, in: Proceedings of the Second ACM International Workshop on Principles of Mobile Computing, ACM, 2002, pp. 38–43.
[4] Ehsan Nourbakhsh, Jeff Dix, Paul Johnson, Ryan Burchfield, S. Venkatesan, Neeraj Mittal, Ravi Prakash, ASSERT: a wireless networking testbed, in: Testbeds and Research Infrastructures. Development of Networks and Communities, Springer, 2011, pp. 209–218.
[5] Azimuth, ACE MX MIMO Channel Emulator for Broadband Wireless, 2014, (http://www.azimuthsystems.com/products/ace-channel-emulators/ace-mx/).
[6] octoScope, octoBox MPE (Multi Path Emulator) Wireless Testbed, 2014, http://www.octoscope.com/English/Products/octoBox_MPE/octoBox_MPE.html.
[7] Kevin C. Borries, Glenn Judd, Daniel D. Stancil, Peter Steenkiste, FPGA-based channel simulator for a wireless network emulator, in: Proceedings of the 69th Vehicular Technology Conference, VTC, IEEE, 2009, pp. 1–5.
[8] Ahmed Khattab, Joseph Camp, Chris Hunter, Patrick Murphy, Ashutosh Sabharwal, Edward W. Knightly, WARP: a flexible platform for clean-slate wireless medium access protocol design, SIGMOBILE Mob. Comput. Commun. Rev. 12 (1) (2008) 56–58, doi:10.1145/1374512.1374532.
[9] Chengshan Xiao, et al., Novel sum-of-sinusoids simulation models for Rayleigh and Rician fading channels, IEEE Trans. Wirel. Commun. 5 (12) (2006) 3667–3679, doi:10.1109/TWC.2006.256990.
[10] R.H. Clarke, A statistical theory of mobile-radio perception, Bell Syst. Tech. J. 47 (1968) 957–1000.
[11] W.C. Jake, Microwave Mobile Communication, Wiley-IEEE Press, Piscataway, NJ, 1974.

[12] A. Alimohammad, B.F. Cockburn, Modeling and hardware implementation aspects of fading channel simulators, IEEE Tran. Veh. Technol. 57 (4) (2008) 2055–2069, doi:10.1109/TVT.2007.914060.
[13] Jianguo Xing, et al., FPGA-accelerated real-time volume rendering for 3d medical image, in: Proceedings of the 3rd International Conference on Biomedical Engineering and Informatics (BMEI), 1, 2010, pp. 273–276, doi:10.1109/BMEI.2010.5639475.
[14] Pengda Huang, M.J. Tonnemacher, Yongjiu Du, D. Rajan, J. Camp, Towards scalable network emulation: channel accuracy versus implementation resources, in: Proceedings of IEEE INFOCOM, 2013, pp. 1959–1967, doi:10.1109/INFCOM.2013.6566996.
[15] Xilinx, XPS System ACE Interface Controller, 2009, http://goo.gl/uYa0mi.
[16] DENXSoftware Engineering, Embedded Linux Development Kit, http://www.denx.de/wiki/DULG/ELDK.
[17] Stephen A. Cook, The complexity of theorem-proving procedures, in: Proceedings of the Third Annual ACM Symposium on Theory of Computing, ACM, 1971, pp. 151–158.
[18] David Guill, Improving wireless testbed utilization through preferential node mapping and grid topology selection, The University of Texas, Dallas, 2013.
[19] WP2.1 SUIT Project Deliverable, Fixed and Mobile Channel Models Identifications, 2006.
[20] Iperf, http://iperf.fr/ (accessed April 2015).