FPGA-based Wireless Link Emulator for Wireless Sensor Network

Wei Liu[1], Luc Bienstman[2], Bart Jooris[1], Opher Yaron[1], and Ingrid Moerman[1]

[1] Ghent University - IBBT, Gaston Crommenlaan 8 Bus 201, B-9050 Gent, Belgium,
wei.liu@intec.ugent.be,luc.bienstman@groept.be,
[2] GroepT University College Leuven, Vesaliusstraat 13, B-3000 Leuven

Abstract. Wireless sensor testbeds lack the flexibility for topology control and the accuracy for interference generation. Once the testbed is set up, the topology becomes fixed. Due to the nature of the wireless environment, experimenters often suffer from unpredictable background interference, while at the same time, find it hard to get accurate and repeatable interference sources.
The wireless link emulator addresses these issues by replacing the uncontrollable wireless link by a well-controlled and programmable hardwired medium. A radio interface is then made to behave according to the link configuration, thus offering flexibility for both topology and interference control. This paper describes the implementation of the wireless link emulator based on a number of low-cost Xilinx FPGAs. The system is verified experimentally and compared to existing emulation systems.

Key words: Topology control, interference control, FPGA, wireless link emulation

1 Introduction

Over the years, more and more researchers have realized that simulation results alone are not sufficient to guarantee the proper function of wireless network applications in a real-life environment. Hence many universities and research groups have setup their own testbeds [1] [2]. Such a testbed often consists of a large number of actual sensor nodes which can be programmed remotely. It is a common practice to install the sensor nodes at fixed locations. Therefore once the testbed is setup, the topology of the network is fixed. In addition, many testbeds are deployed within the office environment, the experiments often suffer from unpredictable interference, such as WIFI, Bluetooth or even microwave oven.

Network simulators are generally flexible and predictable, however, they ignore many aspects of the real hardware platforms. A testbed offers real hardware behavior but lacks flexibility and controllability. Is there a way to combine the advantages of both systems? The answer is yes: use emulation instead of simulation and at the lower level, emulate only the wireless ether behavior, not the sensor node itself. This is the solution integrated into WiLab — the wireless sensor network testbed of Gent University [4].

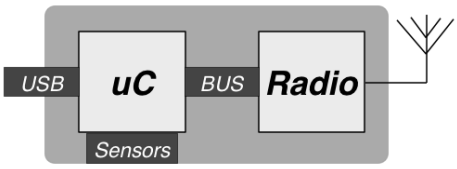

Fig. 1. The original TelosB node

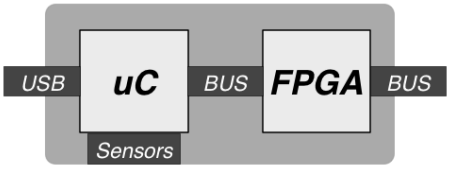

Fig. 2. The modified TelosB node

The WiLab testbed is deployed in an office building of 12x80m and is spread out of three floors. It consists of more than 200 sensor nodes. The sensor node is based upon the TelosB mote [3] (Fig 1). The TelosB mote is an ultra low power wireless module for use in sensor networks. It mainly consists of integrated sensors, a microcontroller and a radio module (Fig 2). The radio module is based on the Chipcon CC2420 radiochip [5]. The CC2420 is a true single-chip 2.4 Ghz IEEE802.15.4 compliant RF transceiver designed for low-power and low-voltage wireless applications. On the TelosB mote, the CC2420 is controlled by TI MSP430 microcontroller.

We extend the WiLab testbed with a group of special nodes. These nodes are also TelosB compatible, however, they communicate via an emulated network instead of the wireless ether (Fig 3, Fig 4). In another word, we introduce a group of nodes with their "private ether" into the testbed. An interface is offered to control the "private ether". For the experimenters of the testbed, those special nodes can be programmed in an identical way as the original nodes.

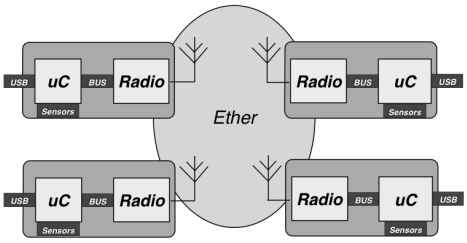

Fig. 3. The wireless network

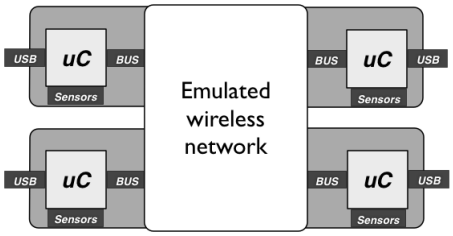

Fig. 4. The emulated wireless network

To implement the wireless link emulator three steps are needed : first, separate the radio from the rest of the hardware on the sensor node. Second, a radio interface is made to maintain the radio functionality. Finally, a hardwire programmable link to other nodes is implemented to replace the ether. In reality this is realized by replacing the radio and its antenna with an interface that connects the MSP430 processor to this private ether. All the ether behavior is emulated with low cost FPGA .

The remaining part of the paper is organized as follows: Part 2 discusses the system implementation in detail. Part 3 describes the principles of physical

layer emulation. Part 4 presents the experimental validation of our system. Part 5 compares our emulator with other emulation systems and Part 6 concludes this paper.

2 System implementation

2.1 Requirements for a wireless network emulator

The emulator of WiLab takes a different approach compared to most existing emulation systems. The idea is to remove the wireless ether completely, and replace it by a well-controlled programmable medium. The requirements for such an emulator are :

- The software running on the modified sensor node should behave exactly the same as if it was executed on the original TelosB mote.
- The electrical characteristics of the radio (CC2420) should be maintained, including the SPI commnication to the local processor and the interface signals (CCA, SFD, FIFO,FIFOP, Vref, RST).
- The transmitting and receiving functionality of the radio should be maintained.
- Similar latency as the radio physical layer is mandatory for the emulator.
- Programmable topology.
- Programmable interference.

Among all the requirements, the latency is the most challenging. Electromagnetic waves travel at the speed of light in the air. By nature, the wireless environment is a broadcast medium with extremely low latency. According to the data sheet of CC2420 [5], there is approximately 2 us latency between the transmitter and the receiver due to the bandwidth limitations on both sides. The emulator only needs to behave as well as the radio, hence the actual latency requirement is 2 us. The detailed calculation related to latency is written in Part 2.5.

2.2 A new proposal : the wired emulator to test a wireless network

In order to achieve reliable low level interconnection, wires are used as the physical medium for the emulator. The "private ether" is now a wired network in which all nodes are connected together by wires. The question is, what is the most suitable physical topology for the emulator to meet all the requirements? One option is to use a full meshed topology, where every node can communicate to every other node, Figure 5 . A Master node is required to control the communication parameters of the mesh while the effective data is directly transmitted between the slaves. The definition of the Master node (to manage the ring) and the Slave node (the real transceiver) is used all over in this paper.

It is obvious that such a topology offers the largest bandwidth, but at the same time it also has the largest amount of connections. For a network of N

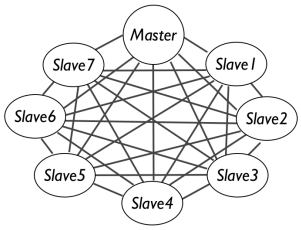

Fig. 5. Full mesh topology

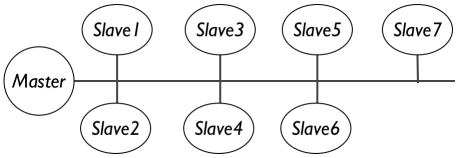

Fig. 6. Bus topology

nodes, a total $N \times (N-1)/2$ connections are required. This could be a considerable number for a complex emulator.

Another option is to connect all the nodes in a common bus topology (Figure 6). All nodes can still communicate to each other, but a complex arbitrator on the bus is required. This could be a serious limitation to the system bandwidth. From a electrical point of view, the more nodes present on the bus, the higher the parasitic capacitance (every node adds some capacitance), and accordingly the lower the switching speed of the bus will be.

The star topology (Figure 7) is a point-to-point network that does not suffer from the accumulation of parasitic capacitance. However, the master node needs to have enough processing power to handle all the incoming and outgoing data to meet the low latency requirement.

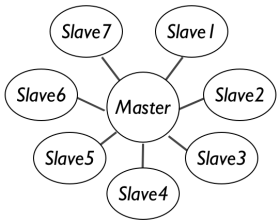

Fig. 7. Star topology

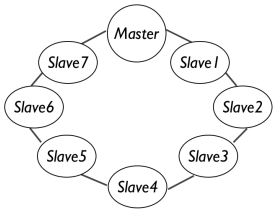

Fig. 8. Ring topology

The ring topology of Figure 8 has some appealing features. It is a point-to-point network, so every link can run at high speed. Although it does not offer direct link between every single node (Slave), the available high speed and an appropriate protocol can compensate for it. The ring topology has an inherent pipelined behavior, which further increases the total network bandwidth. The pipelined architecture allows for simultaneous data transfer between every two neighboring nodes. For example, at the same time instant, while Slave 2 is communicating with Slave 3, Slave 3 can also communicate with Slave 4. This offers a huge advantage over the bus topology. Therefore, we decide to take the ring topology as the physical topology of the emulator.

2.3 The low level protocol

A flow of packets are circulating unidirectionally through the ring. At any moment every single node is receiving a frame from its left neighbor and at the same time is transmitting a frame to its right neighbor. The packet flow is formed by frames (Figure 9). Each frame is allocated to one node (Master, Slave). A node is only allowed to write data into its own frame. When receiving a frame from another node, the content of the frame is retransmitted; when receiving a frame from the node itself, either new data or a dummy empty frame is transmitted. Hence a given frame can circulate only once on the ring. Thanks to the frame structure and inherent pipelining, the ring effectively behaves as multi-access medium without collision, which is exactly what we wanted for the physical connection on the emulator.

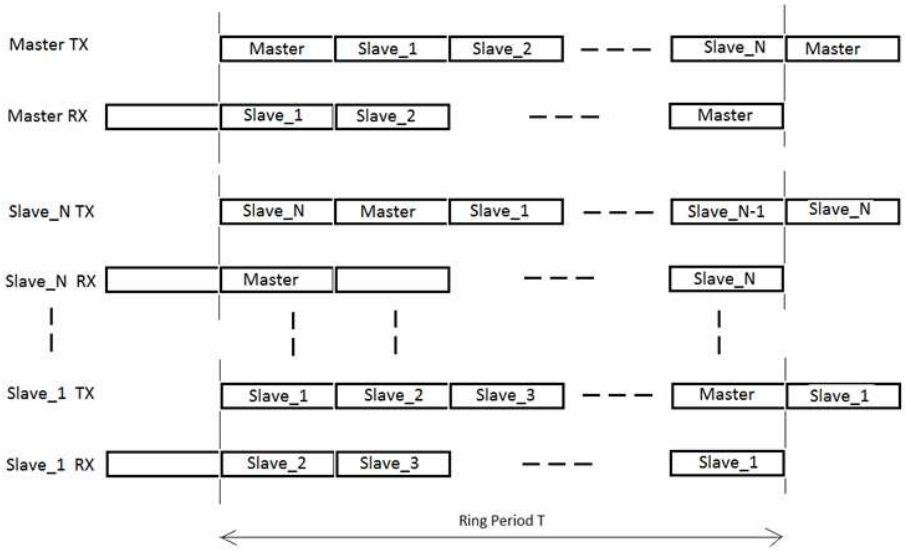

Fig. 9. Frame sequence

A frame is a 32 bit value and can have several formats. Figure 10 shows the normal data frame. A normal data frame is used to transfer data from one slave to the other slaves. The bit D0=1 indicates the normal data frame. The bits D1-D7 contain the source address, which serves as the identifier of the frame on the ring. The D8-D15 bits contain the parameter Channel ID, indicating at which "frequency" the node is transmitting, compatible to the Zigbee channel index. The bits D16-D23 are the transmit power (TX Power) in dBm unit. And the Lower 4 bits of D24-D31 contains the 4-bit actual data, comparable to the 4-bit symbol formed on the real radio, the extra 4 bits are reserved for future extension. The parameters Channel ID and Tx Power cover all the physical property of a symbol. According to the 802.15.4 standard, a symbol stays on the ether for 16

us. To emulate the symbol period, the data frame is only transmitted via the ring every 16 us, however, the actual duration for a frame to circle around the ring is much shorter. This is covered in Part 2.5.

Normal data frame

0	1-7	8-15	16-23	24-31
1	source address	channel ID	Tx Power	Symbol

Fig. 10. Normal data frame

Configuration or node status reporting/logging data frame

0	1-7	8-15	16-23	24-31
0	destination address	parameter ID	parameter value	parameter value

Fig. 11. Configuration or status frame

Besides connections to the ring, the Master node also has connections to a Web Server and a LAN connection to the Wilab Database. Via the Web Server the user can configure the "virtual ether". Parameters for a specific node, such as the noise floor, or path loss, can be programmed via the Master. This is realized by generating a master configuration frame on the ring (Fig 11). This frame circulates through the ring and the addressed slave will copy the data internally. Broadcasting configuration, i.e. addressing several slave nodes with one frame, is possible. Besides configuration, if requested, a slave node can send status reports to the Master. The Master fetches the report and writes it into the WiLab database. The status reports usually contain information related to GPIO activities on the radio interface, or commands received from the local processor. Therefore it is a powerful tool for monitoring the radio activity and software debugging.

Broadcasting is straightforward. A transmitting node will write its frame with a given Channel ID on the ring. Any other node with the same channel ID should read out this frame.

2.4 The physical implementation

A connection between two nodes is made by a standard UTP cable (4 twisted pairs) Fig 12. Low voltage differential signaling (LVDS) is the IO standard on the ring. This ensures good signal integrity at high transmission speed. Among the 4 twisted pairs, one pair is used as clock signal, two pairs are used for data, the last pair is used for synchronization. The Sync signal travels along with the Master frame. It can be considered as the Master frame flag. This Sync signal is essential for the synchronization of the whole ring structure. A node that is receiving a frame while the sync is active is for sure receiving the Master frame.

The two data lines allow to double the transmit speed. Hence to transmit 32 data bits only 16 clock pulses are needed. Three extra clock pulses are needed for internal processing. This results in a total of 19 clock pulses to transmit a 32 bit frame. The timing diagram of the ring is shown in Figure 13. Every frame on the ring corresponds to a time slot of 19 clock pulses.

As mentioned earlier, all the logic needed to implement the ring structure and to emulate the radio module is implemented on the FPGA. Every node

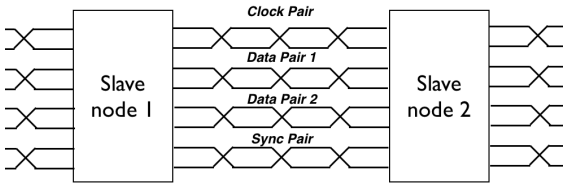

Fig. 12. The 4 pair UTP cable connection between nodes

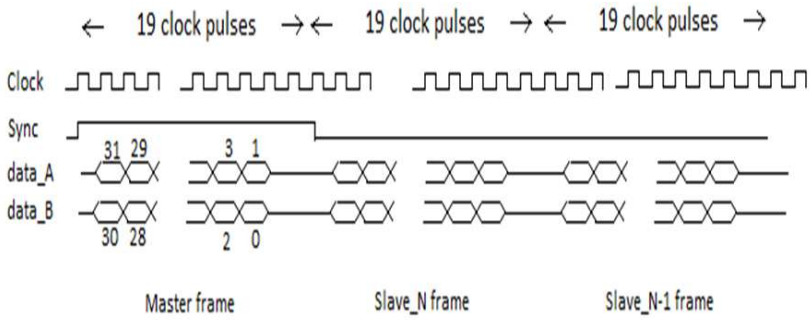

Fig. 13. Detailed timing diagram of the data on the ring

(master, slave) has one FPGA board. More specifically, the slave node is built on the Xilinx Spartan-3A-400 FPGA [7], while the master is built on the Xilinx Spartan3E-500 FPGA [6]. The FPGA chip has a large amount of logic gates available to build all types of dedicated logic, on top of that a powerful 32 bit "soft" microprocessor (microBlaze [15]) is also available as an IP core(intellectual property). To implement the low level ring protocol, a dedicated ring transmitter is built with VHDL(hardware description language).

The radio interface is a combination of software and customized hardware on the FPGA, with the software part running on the "soft" processor — microBlaze, mainly responsible for connecting the ring transceiver and the radio interface. In addition, the software also performs processing needed for physical layer emulation, to be explained in Part 3. The core of the hardware part of the radio interface is a dedicated finite state machine, which takes care of the SPI communication towards the MSP processor, generating necessary interrupt towards software and partially controls the GPIO signals on the radio interface. The block diagram of a slave is shown in Figure 14.

Special care is taken to maintain the quality of the clock signal. The Xilinx Spartan-3A FPGA has on-board high-speed LVDS transceivers to drive the ring. The clock recovery is executed by the PLL inside the FPGA. The Master node is generating the clock while each slave node is reconstructing the clock on its output with minimal phase delay with respect to the input. This recovered clock is used in the ring transceiver logic. Thanks to this structure, the clock quality

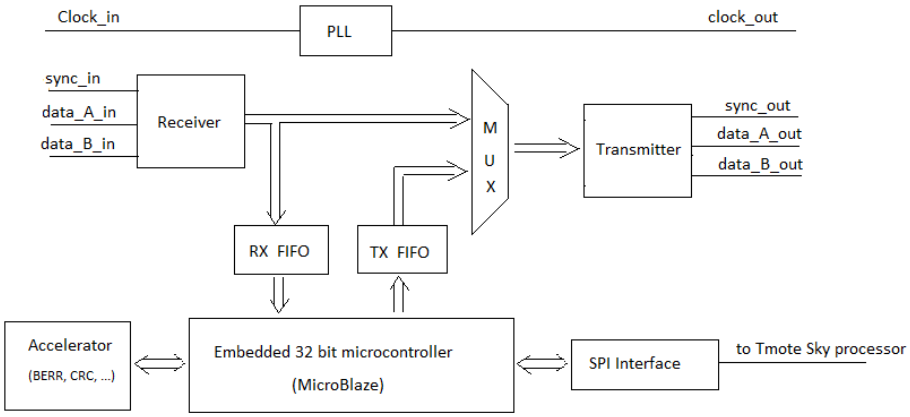

Fig. 14. Block diagram of the slave node

is maintained through the entire ring. This enables the ring clock to run at very high speed.

2.5 Timing considerations

The clock speed of the ring is 100 MHz. A complete 32-bit frame requires 19 clock pulses. We currently only implement a ring with 6 slave nodes and one master. When a frame passes a node, it is first received completely and then transmitted. Hence the duration for a frame to reach all the other nodes on the ring equals:

$$T_1 = 6 \times 19 \times 10nSec = 1.14us \tag{1}$$

The duration for one frame to completely circulate through the ring is :

$$T_2 = (6+1) \times 19 \times 10nSec = 1.33us \tag{2}$$

Be aware that not one but seven frames do travel around the ring during the 1.33 uSec. Hence a node can get access to the ring every 1.33 us.

The latency on the ring is defined as the time between transmitting a frame by a given node and receiving that frame by another node. The best case is when a node is transmitting to its left neighbor, the worst case is when a node is transmitting to its right neighbor. The equation (1) shows that our worst case latency is 1.14 us, smaller than the 2 us latency of CC2420 chip. Hence this design meets the initial requirement.

3 Physical layer emulation

In wireless systems, bit errors occur during the decoding of received symbols. When the received signal is much stronger than the local noise floor, the received

symbol is almost always correctly decoded, hence hardly any bit error can appear. On the other hand, if the received signal is not strong enough to decode, frequent bit errors will appear. In between the two extremes, there is a "gray zone" where the bit error rate varies . We now focus on this zone. It is known that for each modulation technique, there is a given relationship between the bit error rate(BER) and the signal to noise ratio(SNR). The 802.15.4 standard features the OQPSK and DSSS modulation. The theoretical BER curve for 802.15.4 is shown in Figure 15, [13]. Once the SNR is known, we can generate the bit error accordingly.

To enable the calculation of SNR, several parameters need to be considered:

- The transmit power
- The path loss between transmitter and receiver
- The local noise floor and interference level at the receiver

The transmit power accompanies with each symbol as explained in Part 2.4. Each slave has a path-loss table which contains the path loss to all the other nodes. The local noise floor is also a parameter configured by the Master. All the parameters are stored in dB scale. For each incoming symbol, the SNR can be calculated as

$$SNR = TxPower - PathLoss - NoiseFloor \tag{3}$$

If there are multiple senders active at the same time on the same channel, receivers can only recognize the symbol from one sender, the symbols of the other senders are treated as interference. To emulate the interference from other nodes, the strongest interference is used instead of the noise floor in the calculation of SNR. In this case the SNR is actually the same as SINR (signal to interference and noise ratio), for simplicity, we use the term SNR throughout this paper.

3.1 Quantized SNR and its link to bit error rate

Once the SNR value is obtained, theoretically we could calculate the corresponding BER, however, practically this would give too much processing load on the embedded FPGA system. Hence we quantize the BER vs SNR curve and store the most interesting part into a local look-up table. As explained above, the "gray zone" of SNR is closely related to the bit error. So the first step is to quantize this "gray zone". This is illustrated in Figure 15. The SNR value can be expressed by the formula below:

$$SNR = SNR_{offset} + SNR_{step} \times n \tag{4}$$

SNR_{offset} and SNR_{step} are two important parameters. SNR_{offset} represents the lowest SNR value at which data can still be received, albeit with errors. Below this value packets are completely corrupted. SNR_{step} is the quantization step. Thus SNR becomes a function of n. The maximum value of n represents a threshold set by the user. When the SNR value is above the selected threshold, the data is processed without introducing bit error. The value n is also used as the index to look for the proper BER in the look-up table. In the remainder of the paper, the value n is referred as SNR_{index}.

3.2 Bit error generation

The bit error is generated in software. BER value by nature is a fraction number, however, calculation based on floating point and fraction number is slow and expensive. Therefore we express BER as $1/X$, the$X$ is the nearest integer of the BER value's reciprocal. Only X is stored in the look up table. For instance, when BER is 0.1%, the value 1000 will be stored in the look-up table. The software counts the total number of received bits, and will toggle one bit every X bits. The toggle location is generated randomly. When X bits are received, the bit count is cleared to zero, and a new cycle starts with a new random toggle location generated. Such a cycle is called a bit-error cycle.

This solution is simple to implement, but has one major drawback, when transmitting a fixed number of packets with a fixed packet size, the packet error rate(PER) becomes a constant. To avoid this situation, another parameter is introduced — run-length of the bit error. This parameter defines the maximum number of bit errors that can appear in a roll. Thus in the beginning of one bit-error cycle, the random toggle location and the run-length of bit error are generated. When the bit count reaches the toggle location, it will continue to toggle the received bits until the number of toggled bits reaches the run-length. When more than one bits are toggled in a bit-error cycle, there will be none toggled in the following cycles. Hence eventually bit error rate stays the same. The run length parameter effectively characterizes the burst behavior of the bit error.

The random generator used here is based on a 32-bit hardware CRC shift register. So it is actually a pseudo random generator. We admit this can cause certain level of distortion. However, during experiments (see Part 4), we are able to obtain emulation results that are compatible with real measurements, the distortion introduced here is considered to be insignificant.

3.3 Topology Control and Interference Generation

Until now the link between bit error rate and SNR is established. In summary, the topology control is directly achieved by specifying path loss between each node. The path loss will affect several parameters, namely, RSSI and SNR, and eventually affect the bit error rate of the received packet. By configuring the path-loss table in each node inside the "virtual ether", an arbitrary logical topology can be formed, with no impact from the physical ring topology. When the path-loss table is configured in real time, the logical topology becomes dynamic. This allows us to emulate a network with mobile nodes.

As for interference generation, there are two options. One option is to directly generate interference configuration from the Master. This is realized by configuring the local noise floor parameter in all the Slave nodes. The noise configuration can be based on a simple pattern, as illustrated by experiment in Part 4.2. Another possibility is to record the interference in a certain environment and replay it by the Master afterwards. The quality of this approach depends on the time resolution of the recorded interference.

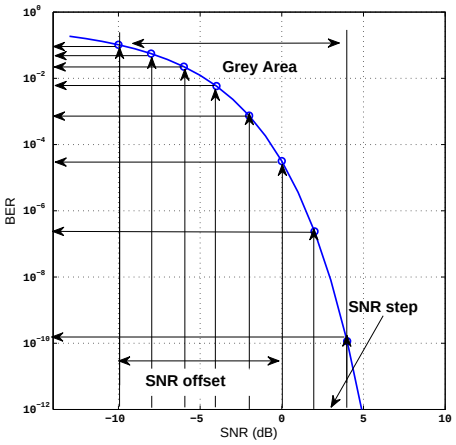

Fig. 15. Quantized BER curve

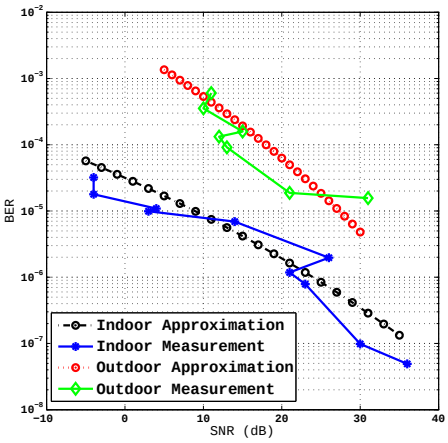

Fig. 16. Approximated BER curve

The second option is done by estimating the BER curve. Sometimes interference does not have a simple pattern. Recording interference with high resolution consumes a large amount of memory. Hence direct interference generation is not always a good option. If we can obtain the BER vs SNR curve under a certain environment, an appropriate amount of bit errors can be generated. The amount of generated bit errors should be equivalent to what is caused by the interference in the given environment. Therefore the desired amount of interference is obtained.

The measurement of BER is usually not that straightforward. It can be done in many ways. Here for simplicity, we assume every bit inside a packet is independent, for a packet of N bits, the BER and PER have the following relationship:

$$1 - PER = (1 - BER)^N \tag{5}$$

Therefore BER can be derived from PER when the packet size is known. The measurement of PER is usually simple. When combine the PER measurement with a simple energy recording, the BER vs SNR curve can be derived. This is further explained with experiment, Part 4.1.

4 Experiments

4.1 Emulation of indoor and outdoor environment by BER estimation

In this experiment we aim to emulate different environments by the proper estimation of BER curve. We used the experiment result in [13] as input, where a set of packet error rate(PER) measurements are performed in function of the distance between the transmitter and the receiver. The measurement is performed

both indoor and outdoor. In addition to PER, RSSI (received signal strength indicator) is also recorded. The indoor experiment is performed multiple times. Each time a different packet size is used. We only selected the measurement with packet size of 127 bytes for the indoor emulation. The PER in the outdoor environment is measured only with a packet size of 20 bytes. For details of the experiment, readers are referred to [13]. We derived the path loss from the measured RSSI, the result is shown in Figure 17. These derived path loss is used as the direct input for the link configuration.

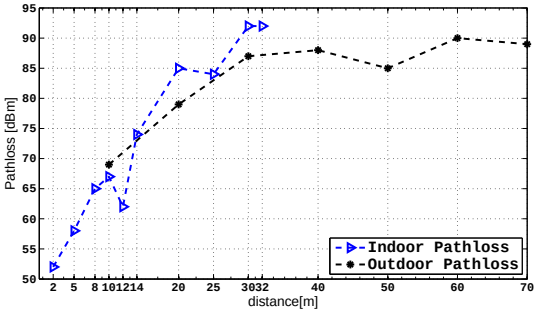

Fig. 17. Path loss vs distance

The next step is to estimate the BER curve. We first calculate the measured BER based on the PER measurement (Fig 18, Fig 19) and Equation (5). We measured the indoor noise floor in our office with Airmagnet [9], which is around -88 dBm. For the outdoor environment, -100 dBm is the selected average noise level based on calculations [17]. Given the transmit power (0 dBm), path loss, and local noise floor, SNR can be calculated according to Equation (3). These lead us to the measured BER curve, shown in Figure 16. Based on the theoretical relationship of SNR and BER, the approximation of measured BER curves are derived (Figure 16). These estimated BER curves are stored in the look-up table for our emulation. The results of the emulated PER for both indoor and outdoor environments are plotted in Figure 18 and Figure 19 respectively.

In general, the emulated PER approaches the measured PER very well for both indoor and outdoor scenarios. There are some deviations at certain locations, these are most likely caused by inaccurate RSSI or simply fluctuations of measurements.

Hence, we prove that by estimating the BER curve properly, we are able to emulate different environments. The PER increase with the distance, which also prove that our methodology for topology control works as expected.

4.2 Emulation of microwave oven interference by direct configuration

In this experiment, we generate the interference configuration directly from the Master. We aim to compare our emulation result with JamLab [12]. JamLab

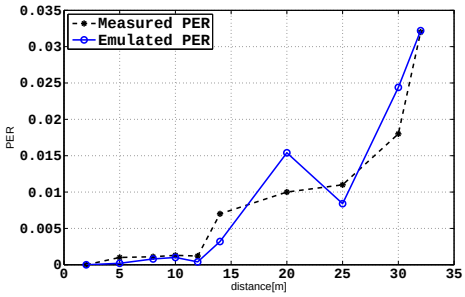

Fig. 18. PER indoor

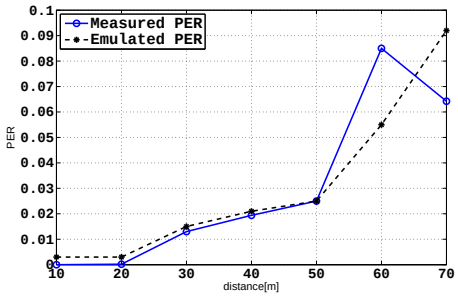

Fig. 19. PER outdoor

focuses on interference emulation based on existing testbed facilities, such as TelosB nodes. According to [12], the interference of microwave oven has a simple on-off pattern with 20 ms period time and 50 % duty cycle. We decide to use this simple pattern to emulate the interference of microwave oven. The experiment scenario is identical as in JamLab. One node sends 400 packets to another node at 1 pkt/sec. The transmitter and receiver are placed about 3 meters apart with no obstacles in between. According to the widely-used path loss model[16], path loss at distance d from the transmitter equals:

$$PL(d) = PL(d_0) + 10\lambda \log \frac{d}{d_0} \tag{6}$$

The path loss at 2 meter is known to be 46 dB [12]. The path loss coefficient λ for indoor environment is typically 2.5. When substitute these values into equation (6), we get 50 dBm as the path loss at 3 meter. Thus the topology can be configured.

The only difference between our emulator and JamLab is how the interference is generated. JamLab used another TelosB node to generate the interference. In our system, Master transmits a configuration frame every 10 ms via the broadcast configuration channel, thus the interference is turned “on” or “off” every 10 ms.

We only emulated for NULL MAC with packet size of 100 bytes. The emulated packet receiving rate (PRR) from our emulator is 44.1%, from JamLab is 43.6%. We can see, that both emulation results are compatible to each other. However, in JamLab, the location of the interferer has to be carefully chosen, the transmit power also needs to be adjusted in order to obtain the right level of interference. If interference is required in a large area, the coverage becomes an issue. When multiple interference sources are present, JamLab needs a careful planning to avoid cross talk between different interference areas. Compared to JamLab, we only need to configure the Master. Every node gets the exact amount of interference as configured, which is much more accurate and flexible.

4.3 Testing at MAC layer

There is always a concern that the physical topology of the ring will influence the network behavior on the higher level. In this experiment we further prove the reliability of the emulator by performing a throughput test at MAC layer.

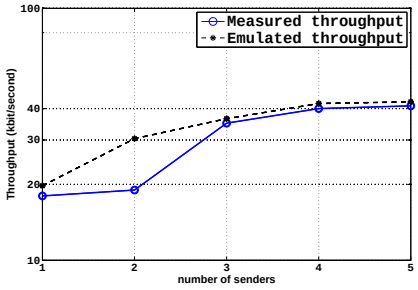

Fig. 20. Throughput vs Senders

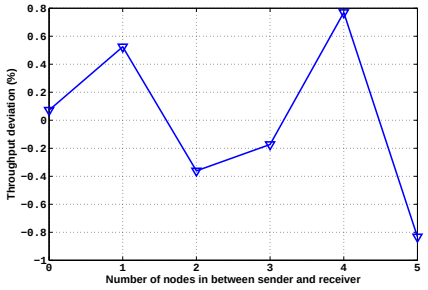

Fig. 21. Deviation vs physical topology

We emulate the throughput experiment presented in [10]. The original experiment is performed on a set of TelosB motes. The receiver is situated in the center while a set of 1 to 10 senders are placed 1 meter away from the receiver. To emulate the same topology, we configured the receiver to have identical path loss to all the senders. The path loss value is calculated based on Equation (6). The senders sent packets to the receiver as fast as permitted by the MAC layer. The receiver counted the number of packets received successfully over the duration of 60 seconds. We only performed the test for XMAC[14] with 1 to 5 senders. The software is downloaded from [11], identical as used in [10]. The result is shown in Figure 20. In general, the throughput increases with the number of senders. The emulated throughput curve fits well with the measurements.

We also take an extra step to exclude the influence of the physical topology. In case of only one sender and one receiver, we change the sender and receiver's physical location, from being neighbors on the ring till 5 hops apart. The deviation of the throughput in percentage is plotted in Figure 21. The deviations do not show a trend related to the physical topology, and the value itself is also very small, less than 1% of the total throughput. Hence we conclude the impact of the physical ring topology on the performance of higher level protocol is neglectable.

5 Related work

Many efforts have been made to overcome the limitations associated with wireless testbeds. For example, the solution presented in [12]. This solution focuses on generating interference with existing off-the-shelf hardware in a testbed. The transmit power and the location of the interferer have to be carefully set up in

order to obtain desired interference coverage and low cross-talk between interference sources. Our solution, on the other hand, does not require such kind of physical deployment. Both topology and interference are controlled by software parameters, and can be realized in real-time.

The work presented in [18] is comparable to our work in the sense that they also use FPGA to replace the wireless media. However, there are three major differences: first at hardware level, they actually intercept the analog RF signal at the antenna port, while in our system, the data is never modulated into RF signal; secondly, in their work the FPGA is used to perform digital signal processing(DSP) based on a certain channel model, while in our system it is used to implement the radio interface and ring transceiver; Finally in essence, our system relies on the relationship between SNR and BER to achieve topology and interference control, while they rely on the physical layer channel model and DSP. From a user perspective, the topology and interference control is realized by tuning DSP parameters of the selected channel model, eg large-scale attenuation or fine grain fading. While for our system it is achieved by specifying parameters such as pathloss and noise level directly. The work presented in [18] is more suitable to emulate certain physical layer phenomena, while our system focuses more on general network performance. Also we believe our system is more user-friendly for researchers without DSP and physical layer background.

6 Conclusion and future work

We implemented a wireless link emulator based on low-cost Xilinx FPGA. This emulator differs from previous work by its unique hardware aspects, including the customized radio interface and the ring transceiver. The high speed hardware design is the key enabler for the concept of physical layer emulation.

We introduced the methodology used by our system to emulate various environments, and demonstrated experimental results that are compatible with real-life measurements.

Nevertheless, many interesting directions remain for further research. For example, the real-time topology control can be used to emulate a network with mobile nodes, and the real-time interference control creates the possibility to replay interference from recordings.

Our solution is a low cost yet very powerful and flexible test facility, that can be extended to other radio chips and wireless technologies. The scale of the emulated network can be further increased by using the newest FPGA family (Spartan6) yielding higher speeds clock on the ring. In the future, we aim to enhance the ring clock speed by a factor of eight, resulting in 48 nodes in the emulated network.

Acknowledgment

The research leading to these results has received funding from the European Union's Seventh Framework Programme FP7 under grant agreements number 258301 (CREW project) and number 287581 (OpenLab project).

The authors would also like to thank Piet Cordmans, Yang Yang, Stefan Schipper, Libo Li and Peter Ruckebusch for their contribution to this work.

References

1. Werner-Allen G., Swieskowski P. and Welsh M.:Motelab: A wireless sensor network testbed. 4th Annual Conference on Information Processing in Sensor Networks (IPSN) 2005
2. Handziski V. et al.:TWIST: a scalable and reconfigurable testbed for wireless indoor experiments with sensor networks. In Proceedings of the 2nd international Workshop on Multi- Hop Ad Hoc Networks: From theory To Reality (REALMAN 06), Florence, Italy, pp. 63-70, May 26 - 26, 2006
3. Tmote sky: ultra low power IEEE 802.15.4 compliant wireless sensor module.2006
4. Lieven Tytgat, Bart Jooris : WiLab: a real-life Wireless Sensor Testbed with Environment Emulation 2009
5. Chipcon. CC2420 datasheet 2007 March Available from http://focus.ti.com/lit/ds/symlink/cc2420.pdf
6. Xilinx, UG331 Spartan-3 Generation FPGA User Guide, www.xilinx.com
7. Avnet, Spartan-3A Evaluation Kit, User Guide, www.avnet.com
8. Digilent, Spartan-3E Starter Kit, www.digilent.com
9. Fluck Corporation *AnalyzerAir User Manual*, Rev.2, 2006
10. Kevin K. et al.: A Component-Based Architecture for Power-Efficient Media Access Control in Wireless Sensor Networks. In : SenSys 2007
11. [Online]. Available: http://tinyos.cvs.sourceforge.net/tinyos/tinyos-2.x-contrib/wustl/upma/
12. Carlo Alberto B. et al.:JamLab: Augmenting Sensornet Testbeds with Realistic and Controlled Interference Generation. In: IPSN11, April 1214, 2011, Chicago, Illinois.
13. Petrova, M., Riihijarvi, J., Mahonen, P., Labella, S.: Performance study of IEEE 802.15.4 using measurements and simulations.In: Wireless Communications and Networking Conference, 2006. WCNC 2006. IEEE
14. M. Buettner, G. V. Yee, E. Anderson, and R. Han : X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks. In SenSys, 2006
15. MicroBlaze Processor Reference Guide: Embedded Development Kit EDK 10.1i, http://www.xilinx.com
16. M. Zuniga and B. Krishnamachari.: Analyzing the Transitional Region in Low-Power Wireless Links. In SECON04
17. S. Haykin, *Communication Systems*, 4th ed. New York: Wiley, 2001, p. 61
18. Kevin Borries, Xiaohui Wang, Glenn Judd, Eric Anderson, and Peter Steenkiste : Experience with a Wireless Network Testbed based on Signal Propagation Emulation, , IEEE European Wireless 2010 Lucca, Italy, April 12-15, 2010