A Sample-Driven Channel Model for Developing and Testing Practical WSN Applications

Pawel Gburzynski
Department of Computer Science, Vistula University
ul. Stoklosy 3, 02-787 Warsaw, Poland
Email: p.gburzynski@vistula.edu.pl

***Abstract*—We present a generic, wireless channel model, parametrized by real-life sample data, intended for simulation (or rather emulation) of Wireless Sensor Networks (WSN) for the purpose of pre-deployment testing and verification. The model is plugged into a virtual execution environment enabling us to execute the full WSN application, including its associated *external* programs (jointly dubbed the Operational Support System, or OSS), which thus can be conveniently developed and authoritatively tested in their target (albeit virtual) setup without the need to run a real (physical) instance of the network. We illustrate the advantage of this approach (focusing on the channel model) with an application in which the closeness of the virtual environment to its real-life counterpart is particularly important, namely, an indoor location tracking system.**

I. Introduction

Modeling wireless channels has been an important part of the art of simulation of wireless telecommunication systems. The most obvious application for such endeavors is in abstract performance studies where different (typically hypothetical) solutions are compared *virtually* to identify their advantages and flaws before (possibly) turning them into practical (real-life) implementations. In such studies, one typically strives to abstract as much as possible from any particular physical setup trying to capture the essence of a wide class of anticipated real-life scenarios, with the intention of making the conclusions as general as possible. Thus, one wants to make the simulation model simple in terms of the number of its parameters, while also making it as realistic as possible, in terms of the applicability of the conclusions to a variety of real-life implementations. This is clearly a tradeoff and, as most of us know, it doesn't trade very well when it comes to modeling wireless channels. This is because wireless channels are capricious and temperamental. While their behavior in any particular environment may fit some wide-enough statistics, it is usually quite unique, in the sense that specific communication episodes from within a specific area follow very specific distribution and correlation patterns, ones that are seldom adequately captured by known-in-advance formulas. Thus, even within the domain of blanket (i.e., simply parametrized) channel models, we see numerous variants intended for certain classes of applications, e.g., indoor areas [1], [2], [3], vehicle-to-vehicle communication [4], [5], or body area networks [6], [7].

The work reported in this paper has been inspired by very practical projects involving real-life, wireless, ad-hoc, sensor networks (WSNs) consisting of hundreds of nodes. Some of those projects were academic (or half-academic) in nature, e.g., see [8], while other were mostly commercial. As a motivating illustration, we shall use ALPHANET which is a comprehensive WSN for independent living (IL) facilities.[1]

The most serious problem faced by an industrial developer of WSN-based solutions (in addition to conceiving and devising them) is testing and debugging the system before its physical deployment. This applies to the programs executed in the wireless nodes, as well as the external programs (the OSS) conversing with the network. Note that in a true, massive, wireless ad-hoc system, consisting of a large number of nodes realizing a complex distributed program, this activity may involve replacing the code (firmware) in all (or most of) the nodes, possibly many times, as problems are detected, diagnosed, and eliminated. Needless to say, the nature of this work (especially when the nodes have been deployed over a sizable area) makes it quite different from debugging a regular (workstation or server) program, from an armchair, with all the physical activities confined to pushing keys on the keyboard and clicking the mouse.

Various tools have been suggested to assist in testing (debugging, replacing) programs in wireless nodes (e.g., [9], [10], [11]). While they do not completely eliminate the burden of dealing with the physical nodes, they facilitate (to various degrees) problem detection, code replacement, or remote inspection of node activities. Unfortunately, their shared cost is the rather serious complication of firmware. From the industrial point of view (often ignored or misrepresented in purely academic studies), being able to attain a given goal with the minimum-footprint devices is usually critical for success (or even for a start). For illustration, the typical amount of RAM in an ALPHANET device is 4KB. Thanks to the ingenuity of our programming platform [12], this is quite enough for the application program, but any attempt to squeeze into the node,

The author is also affiliated with Sendronet, see http://www.sendronet.com.

[1]This is a collaborative effort with Alphatronics, a company in Belgium, see http://www.alphatronics.be.

say, a "small" virtual machine [11], [10] stands no chance at all.

Our solution to the problem is complete virtual execution. This is accomplished by VUEE[2] [13], an event-driven, reactive emulator for WSN applications. VUEE accepts node programs at the source code level, recompiles them and executes in an emulated artificial environment within the comfortable setting of a laptop or workstation. In addition to testing and visualizing the network, VUEE can also interface it to OSS programs. This way those programs can be developed and tested without ever seeing the physical devices.

RF communication in VUEE is simulated, or rather *emulated*, because, e.g., *true* packets are transmitted and delivered. Thus, a channel model is needed to account for signal attenuation, interference, bit error rate (*BER*), and so on. The purpose of this model is somewhat different from the standard case of modeling for performance evaluation. While still interested in performance, we would like to see a reasonably close relationship of the model to the real (intended) deployment environment. A compelling reason for this requirement is that our WSN applications commonly use the received signal strength (*RSS*) indication as a special "sensor" for various proximity based events and predicates. The most drastic variant of this special sensor is the location tracker, i.e., a subsystem implementing RSS-based location estimation of the mobile nodes (aka *Tags*) worn by the IL patients. Having a friendly vehicle for virtual execution of the complete application, one would naturally like to use it for testing/debugging the location tracking component of the OSS.

II. The model

A. The generic part

The built-in part of the RF channel model in VUEE is organized around the event-driven paradigm. It takes care of all the dynamics, while a specific model instance defines selected parameters of those dynamics by specifying a few functions constituting the open ends of the generic model.

The standard stages of a packet transmission include: sending a preamble, starting the packet, and terminating it. All those stages are represented by events. If an event occurs at time t at the sender S, then it will (possibly) occur at the recipient R at time $t + d$, where d is the distance between the two parties transformed into the propagation time.

A signal is always transmitted at some (transmit) power level P_x. By the time it reaches the receiver at any node, the signal becomes *attenuated*, i.e., its level at the receiver becomes $P_r = P_x \times A$, where A is the attenuation value provided by the model instance (one of the instance functions) on the case-by-case basis. As the power is usually expressed in dBm and the attenuation in dB, the above formula takes the shape:[3]

$$P_r|_{dBm} = P_x|_{dBm} - A|_{dB} \tag{1}$$

[2]The acronym stands for Virtual Underlay Execution Engine.

[3]From now on, we will be ignoring the logarithmic qualifiers assuming it never leads to a confusion.

The generic model consults the instance function whenever it needs the attenuation. The function can base its calculations on any conceivable (also dynamic) parameters involving S and R. In a simple case, it can apply some closed formula, e.g., one solely based on the distance, while in general the formula can be driven by arbitrarily complex criteria, e.g., known only to the model instance.

Consider some receiver R. At any moment, R can be reached by any number of signals whose perceived levels add up. The addition function need not be straightforward (it is another instance parameter) and may factor in arbitrary criteria, e.g., accounting for the frequency/code-dependent crosstalk between different channels. Let T denote the total combined signal level perceived by R. Suppose R is receiving (tuned to) some packet at the signal level P_r. The value:

$$\mathit{SIR} = \frac{P_r}{T - P_r + B} \tag{2}$$

where B is the *background noise* level determines the signal-to-interference ratio. The reception opportunities for the packet depend on *SIR* via the *BER* function which transforms a given value of *SIR* into the probability that a single bit will be received in error. In the model instance, the function is specified as a discrete table (a list of *SIR*, *BER* pairs) and its values for the unspecified arguments are interpolated. The model distinguishes between *physical* bits (to which *BER* actually applies) and *logical* bits (the ones forming the packets). This means that physical bits can be grouped into *symbols* with one symbol representing a number of logical bits (which is often less than the corresponding number of physical bits).

One popular reception pattern works like this. The receiver R formally begins to receive a packet when it has recognized (no bit errors) some number of preamble bits followed by the 16-bit *sync word* (a predefined bit pattern). While receiving the packet, R simultaneously waits for two events: the last bit of the packet and the first bit error. The mean waiting time for the latter event is calculated based on *BER* (for the current value of *SIR*). The actual time is drawn from some distribution which is typically Poisson, but can be redefined by an instance function. Whenever the configuration of signals (and thus *SIR*) perceived by the receiver changes (which may occur half way through the received packet), the delay until the first error bit is recalculated and the waiting is restarted from the moment of change.[4] The packet is deemed to have been received correctly, if the bit error event never occurs until the last bit event.

B. The specific part

The RF module used in the real network is CC1100/CC430 manufactured by Texas Instruments [15]. The reception pattern described at the end of Section II-A well corresponds to the operation of the real device. As all RF modules in its class, CC1100 returns the value of *RSS* along with a received packet.

[4]Note that the reception model is highly dynamic and, in principle, accounts for all the relevant real-life attributes affecting the reception. This is unlike virtually all popular channel models [14] which typically determine the packet's fate at a receiver only once.

The *RSS* reading is taken at the end of the sync word and (after trivially applying a constant offset) translates into the actual received signal level in dBm discretized into 0.5dBm steps. The module provides for setting the transmit power P_x in a flexible and dynamic manner: every packet can be transmitted at a different power level.

The most important component of the sampled model is the attenuation function driven by two groups of parameters: 1) a list of samples (provided in one of the data files) including *RSS* readings for specific cases of packet reception in the deployment area under study; 2) a distance-based (blanket) fallback function to be applied when no fitting sample is available. This way the model is able to take advantage of whatever (partial) samples have been collected from the area, while offering some reasonable behavior in the face of lack thereof. A collected sample consists of the coordinates of two points in 3-space, $S = (x_0, y_0, z_0)$, $R = (x_1, y_1, z_1)$ (providing the locations of the sender and the receiver), the transmission power level (P_x) in dBm, and the *RSS* value observed at R. The last two values are transformed into the attenuation in dB:

$$A = RSS - P_x \tag{3}$$

Thus, in addition to the locations of S and R, the sample brings in a single value representing an observed case of attenuation.

The location points are subsequently converted to the simulator's internal grid of 10cm along each dimension with points falling into the same grid cube considered indistinguishable. Multiple collected samples whose both ends become indistinguishable this way are merged into a single (effective) sample, their attenuation values averaged. When this happens, the model also evaluates the sampled standard deviation σ for randomizing the specific instances of attenuation to be generated from the sample. The model can be optionally declared as symmetric which means that S and R can be interchanged. Conceptually, every sample is treated as two samples in that case.

The model implements a function producing (randomized) signal attenuation between any pair of points S and R. Notably, it can also work (in a simplified and substandard fashion) when there are no samples at all. To that end, it employs the fallback function F specified as a list of triplets: (d, A, σ), where d is distance, A is attenuation, and σ is the standard deviation. The function generates randomized attenuation values based solely on distance using interpolation to obtain A and σ for unspecified d and applying Gaussian variates to those values. Normally, when samples are present, the instance attenuation function combines them with F to produce weighted attenuation values, according to the following procedure.

The function is invoked for an (S, R) pair. If the pair exactly matches some sample i (modulo the grid), the mean attenuation value A_i and the sample deviation σ_i are taken directly from the sample. The latter is only defined if the sample is an average of multiple collection samples; otherwise, σ_i is obtained from F by applying the distance between S and R. The function returns $A = A_i + X(\sigma_i)$, where X denotes the Gaussian distribution centered around 0.

If the pair (S, R) cannot be matched exactly to any sample, the function produces a kind of average from the closest samples also incorporating F in proportion to how close/far the closest samples are from the actual case. The objectives of this procedure are: 1) to give priority (higher impact) to close samples; 2) to account for the distance between S and R, if there is no close match by samples; 3) to yield the obvious answer for the exact match naturally, without considering that situation as a special case. To appreciate point 2, note that samples can only refer to packets that have been received, while the function must be able to produce a value for any pair of S and R, which may happen to be located too far apart for a successful transmission. Any attenuation value produced exclusively from the closest samples will always yield a receivable signal level and will make no sense when S and R are not in range. In contrast to many popular channel models, our model incorporates such signals into the overall interference (for which an attenuation value is needed). For 3, we would like to make sure that the averaging scheme is smooth, continuous, and seamless, and it automatically produces the right solution when the situation happens to fit the simplest and most obvious case. This should be an important sanity criterion for any complex scheme based on averages and weights.

Given $S = (x_s, y_s, z_s)$ and $R = (x_r, y_r, z_r)$, the function starts by locating K closest samples, where K is a parameter typically set to 4. The distance from the (S, R) pair to a sample i, $0 \le i < K$ is calculated as:

$$d_i = \sqrt{(x_s^i - x_s)^2 + (y_s^i - y_s)^2 + (z_s^i - z_s)^2} + \sqrt{(x_r^i - x_r)^2 + (y_r^i - y_r)^2 + (z_r^i - z_r)^2} \tag{4}$$

where the superscripted coordinates come from the sample. In plain words, the distance is equal to the sum of the Euclidean distances between the corresponding points. If the channel is symmetric, a second distance, with S and R swapped, is also calculated and the minimum of the two values is taken. Then the distances for all K closest samples are transformed into preliminary weights according to this formula:

$$u_i = e^{-\frac{d_i \times \alpha}{d_{avg}}} \tag{5}$$

where d_{avg} is the simple average of all K distances d_j, $0 \le j < K$ and α is a parameter factor typically set to 0.1. Note that a smaller distance results in a higher weight. As all distances are nonnegative, the function reaches its maximum (of 1) for $d_i = 0$.

In the next stage, the weights are transformed to secondary weights:

$$w_i = u_i \times \prod_{j \ne i} (1 - u_j) \tag{6}$$

Note that after this transformation, the impact of more distant samples is reduced in agreement with postulate 3 above. In particular, for $d_i = 0$ ($u_i = 1$), i.e., a perfect match, w_i

becomes 1, while the remaining weights disappear. Finally, the weights are normalized into:

$$W_i = \frac{w_i}{\sum_{j=0}^{K-1} w_j} \tag{7}$$

so they add to 1, and used to calculate these weighted averages:

$$\begin{array}{rcl} A_{avg} & = & \sum_i W_i \times A_i \\ \sigma_{avg} & = & \sum_i W_i \times \sigma_i \\ d_{avg} & = & \sum_i W_i \times d_i \end{array} \tag{8}$$

where A_i and σ_i are the attenuation and standard deviation values associated with sample i, and d_i is the distance between the sample's endpoints. Then the resulting attenuation is returned as:

$$A = A_{avg} \times \frac{A_F(d)}{A_F(d_{avg})} + X(\sigma_{avg}) \tag{9}$$

where d is the distance between S and R and A_F is the attenuation returned by the distance-based fallback function F for the given distance. Note that for a perfect match to any of the K samples, $A_F(d) = A_F(d_{avg})$ and $A = A_{avg} + X(\sigma_{avg})$. When the match is not perfect, and the distance between S and R is longer than the average of the samples (possibly extending into the nonreceivable range), it will tend to push the attenuation towards the values determined by the distance-based fallback function F for that distance.

C. Feeding the model with data

In our reference application, *RSS* samples are routinely collected, as part of system deployment, in the course of *profiling* needed by the location tracking service. From the viewpoint of this service, the network consists of tracked (mobile) nodes, called *Tags*, and static nodes referred to as *Pegs*. The devices are fairly homogenous within their class. The Tags emit bursts of short packets transmitted at different power levels. Those Pegs that manage to receive any of those packets report their *RSS* readings to the OSS which compares them to samples previously collected from known locations and stored in a database. A database sample typically represents an average of several *takes* and consists of 8 *RSS* readings (for 8 different transmission power levels) obtained from some point within the monitored area. The tracking service only deals with symbolic names of the locations, i.e., we do not care about the numerical values of the coordinates, although the positions of samples are known quite accurately. Usually, a given symbolic location is represented by at least 5 aggregate samples. Typically, they are collected from the location's corners and its center.

As the location samples are readily available, a natural idea is to use them to drive the channel model of the emulator. For that, the emulator's floor-plan visualizer is used to obtain the numerical coordinates of the end-points of the samples, thus turning them into the format acceptable by the channel model. Note that a location sample is already averaged, in fact in two ways. First, the *RSS* entry for each power level represents an average of several takes; second, the readings for the multiple power levels are combined into a single attenuation value, according to formula 3. A trivial add-on to the profiling procedure (not needed by the location estimator) is to calculate the standard deviation of the *RSS* readings contributing to the resulting average attenuation. This way, even though the sample formally amounts to a single reading for a given (R, S) pair, its σ is available and meaningful.

Notably, the samples allow us to fill in most of the remaining parameters of the channel model. One of those parameters is the configuration of *effective* transmission power levels, i.e., the dBm value of P_x (for each of formal settings from 0 through 7) to be used for calculating P_r in formula 1. The problem is that only the maximum power level setting (level 7) can be reliably obtained from the device's datasheet; the remaining ones have been arrived at by an elaborate trial and error procedure to produce the right kind of diversity for the location estimator.[5] A natural way to calibrate the setting for a power level i, $i < 7$ is to average the *RSS* drop for that level, compared to the *RSS* value for power level 7, i.e., $P_x(i) = P_x(7) - \text{Avg}(\textit{RSS}(7) - \textit{RSS}(i))$, where the average is taken over all samples. Similarly, the rough dependence of attenuation on distance, for the fallback function F, can be inferred by fitting the attenuation observed in the samples, which always comes with an associated (S, R) pair, and thus distance, to some function. As F is specified as an interpolation table, and thus need not follow any particular formula, we derive the table from the samples via a simple algorithm. We start by setting d and w to 0 and 2, respectively. The first value is the minimum distance to be considered in the current step, while the second one gives the width of the distance margin in meters. For a given pair d and w we select all samples for which the distance from S to R falls between d and $d+w$; then we calculate 1) the average of those distances, 2) the average attenuation, 3) the standard deviation (σ) of that attenuation, and output the three values as the next entry in the description of F. Then we set $d = d + w/2$, $w = w + 1$ and proceed with the next iteration for as long as we have not covered all the samples. Note that the new interval partly overlaps the previous one (to allow for an imbalance in the distribution of distances in the samples). Its width also increases with every iteration, because longer distances tend to bring in more uncertainty and they also offer fewer opportunities for a successful reception. Note that the role of F is secondary, and its accuracy is not critical, especially with a good coverage of the area with samples. The table for F produced in the above way is augmented by one more (last) entry corresponding to the maximum separation distance between a pair of nodes in the network (the network diameter). The attenuation value and σ of that entry are extrapolated from the previous two entries. Note that when the attenuation is expressed in dB, the linear extrapolation becomes in fact exponential.

The calibration procedure for the model has been programmed into a script which, given the database of samples of the location estimator + the coordinates of Pegs + the

[5]The details are beyond the scope of this paper.

coordinates of sample points, produces the complete input to the model. Two elements of the model that must still be defined are the *BER* function (or rather table) describing the bit error rate as the function of *SIR*, and the background noise level (B in formula 2). The latter can be measured by the RF module (by taking *RSS* readings without a packet reception) and is normally fixed, unless the environment is particularly noisy. The former can be obtained experimentally and, notably, does not depend on the deployment; thus, it can be viewed as a fixed parameter of the model, not subjected to deployment-specific calibration.

III. Conclusions

The channel model discussed above has been used in the development and testing of a real-life WSN in a purely virtual environment. One of the services offered by the network is location tracking. In fact, the need to have a reliable and authoritative tool for testing the location tracking component of the entire system, including the WSN, as well as the OSS server, inspired us to develop a channel model that would produce results consistent with those observed in the production setup. Note that we did not care as much about the deliverance of realistic events testing the "intelligence" of our server where it comes to correctly guessing a Tag's location, as about exposing it to traffic patterns synonymous with those occurring in the physical system. In particular, the database of samples used by the server (and also driving the channel model) has been obtained from the real deployment. Our experience with indoor location tracking strongly suggests that RF channel models are not going to be extremely helpful for the operation of tuning location estimation engines, unless they begin to adequately capture the diversity of practical cases of signal attenuation, e.g., involving variations in the proximity of the RF device to the human body, its orientation, as well as various ad-hoc disturbances (and sources of multi-path propagation) resulting from people passing nearby, equipment being moved, the furniture being rearranged, and so on. On the other hand, the model produces a good illusion of conversing with the real network and has proved very handy for the "off-line" development work, like testing new features in the OSS, and (much more importantly) testing new firmware before uploading it into real nodes.

The limited size of this paper does not allow us to include here a meaningful report from our extensive experience. For a quick impression, below are two excerpts from the server's log (the values between $<$ and $>$ are *RSS* readings for the 8 transmit power levels):

```
12:17:43 peg=39624 tag=39939 <0 0 0 0 34 49 67 73>
12:17:43 peg=39369 tag=39939 <0 0 0 0 40 60 72 79>
12:17:43 peg=39351 tag=39939 <0 0 0 0 48 62 78 90>
12:17:44 peg=39601 tag=39939 <0 0 31 40 58 67 80 90>
12:17:44 LE for 39939 -> 10 91 7 9

  ...

21:34:19 peg=39369 tag=39939 <0 0 0 0 38 62 70 80>
21:34:19 peg=39351 tag=39939 <0 0 0 0 46 64 77 87>
21:34:19 peg=39624 tag=39939 <0 0 0 0 39 51 68 72>
21:34:19 peg=39601 tag=39939 <0 0 35 42 60 67 79 92>
21:34:20 LE for 39939 -> 10 92 7 8
```

One of them comes from a real-life experiment, while the other has been obtained from the model (under identical conditions). Needless to say, it is never possible to tell which one is which.

Note that some features of complex, non-empirical models, like those striving to capture the impact of walls [1], are implicitly captured in our model via the samples, especially if their density allows the endpoints to be statistically separated by the same walls that their closest samples. Of course, the denser the coverage of the deployment area by samples, the better the model is going to reflect environmental intricacies. The next step in the model's evolution will be to account for some of the dynamic ones.

References

[1] T. Chrysikos, G. Georgopoulos, and S. Kotsopoulos, "Wireless channel characterization for a home indoor propagation topology at 2.4 ghz," in *Wireless Telecommunications Symposium (WTS), 2011*. IEEE, 2011, pp. 1–10.

[2] M. Fryziel, C. Loyez, L. Clavier, N. Rolland, and P. A. Rolland, "Path-loss model of the 60-ghz indoor radio channel," *Microwave and optical technology letters*, vol. 34, no. 3, pp. 158–162, 2002.

[3] K. Haneda, J. Jarvelainen, A. Karttunen, M. Kyro, and J. Putkonen, "A statistical spatio-temporal radio channel model for large indoor environments at 60 and 70 ghz," *Antennas and Propagation, IEEE Transactions on*, vol. 63, no. 6, pp. 2694–2704, 2015.

[4] M. Boban, "Realistic and efficient channel modeling for vehicular networks," *arXiv preprint arXiv:1405.1008*, 2014.

[5] T. Abbas, K. Sjöberg, J. Karedal, and F. Tufvesson, "A measurement based shadow fading model for vehicle-to-vehicle network simulations," *International Journal of Antennas and Propagation*, vol. 2015, 2015.

[6] J. Ryckaert, P. De Doncker, R. Meys, A. de Le Hoye, and S. Donnay, "Channel model for wireless communication around human body," *Electronics letters*, vol. 40, no. 9, p. 1, 2004.

[7] A. Fort, C. Desset, P. De Doncker, P. Wambacq, and L. Van Biesen, "An ultra-wideband body area propagation channel model-from statistics to implementation," *Microwave Theory and Techniques, IEEE Transactions on*, vol. 54, no. 4, pp. 1820–1826, June 2006.

[8] N. M. Boers, D. Chodos, P. Gburzynski, L. Guirguis, J. Huang, R. Lederer, L. Liu, I. Nikolaidis, C. Sadowski, and E. Stroulia, "The smart condo project: services for independent living," *E-Health, assistive technologies and applications for assisted living: challenges and solutions. IGI Global*, 2010.

[9] K. Whitehouse, G. Tolle, J. Taneja, C. Sharp, S. Kim, J. Jeong, J. Hui, P. Dutta, and D. Culler, "Marionette: using RPC for interactive development and debugging of wireless embedded networks," in *Proceedings of the 5th international conference on Information processing in sensor networks*. ACM, 2006, pp. 416–423.

[10] P. Levis and D. Culler, "Maté: A tiny virtual machine for sensor networks," *ACM Sigplan Notices*, vol. 37, no. 10, pp. 85–95, 2002.

[11] C.-L. Fok, G.-C. Roman, and C. Lu, "Rapid development and flexible deployment of adaptive wireless sensor network applications," in *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on*. IEEE, 2005, pp. 653–662.

[12] P. Gburzynski and W. Olesinski, "On a practical approach to low-cost ad hoc wireless networking," *Journal of Telecommunications and Information Technology*, vol. 2008, no. 1, pp. 29–42, Jan. 2008.

[13] N. Boers, P. Gburzyski, I. Nikolaidis, and W. Olesiski, "Developing wireless sensor network applications in a virtual environment," *Telecommunication Systems*, vol. 45, no. 2-3, pp. 165–176, 2010. [Online]. Available: http://dx.doi.org/10.1007/s11235-009-9246-x

[14] D. Mahrenholz and S. Ivanov, "Real-time network emulation with ns-2," in *Proceedings of Eighth IEEE International Symposium on Distributed Simulation and Real-Time Applications*, 2004, pp. 29–36.

[15] Texas Instruments, "CC1100 Single Chip Low Cost Low Power RF Transceiver," 2014, document SWRS038D. [Online]. Available: http://focus.ti.com/lit/ds/symlink/cc1100.pdf