Paweł Gburzyński[1]
Akademia Finansów i Biznesu Vistula – Warszawa

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

Summary

We present a generic, wireless channel model, parametrized by real-life sample data, intended for simulation 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 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.

Key words: wireless communication, simulation, channel models.

JEL codes: L86

Introduction

Modeling wireless channels has been an important part of the art of simulation of wireless telecommunication systems. The most obvious application for simulation in this area is in abstract performance studies where different (typically hypothetical) solutions are compared *virtually* to identify their advantages and flaws before (possibly) turning them into 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 it doesn't fare 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

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

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 others were mostly commercial. As a motivating illustration, we shall use ALPHANET [9] which is a comprehensive WSN for independent living (IL) facilities[2].

The most serious problem faced by a 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 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 on the mouse.

Various tools have been suggested to assist in testing (debugging, replacing) programs in wireless nodes (e.g., [10, 11, 12]). 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 the success (or even for the start) of a marketable project. For illustration, the typical amount of RAM in an ALPHANET device is 4KB. While this may be quite enough for the carefully designed and implemented application program [13], any attempt to squeeze into the node, say, a "small" virtual machine [11, 12] stands no chance at all (to put it mildly).

Our solution to the problem is authoritative virtual execution of a faithful model of the complete real-life system (the so-called *praxis* [13]). This is accomplished by VUEE[3] [14], 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 visualiz-

[2] This is a collaborative effort with Alphatronics, a company in Belgium, see: http://www.alphatronics.be.
[3] The acronym stands for Virtual Underlay Execution Engine.

ing 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 in the process.

RF communication in VUEE is simulated, or rather *emulated*; for example, *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 that model is somewhat different from the standard case of modeling for abstract performance evaluation. While still interested in performance, we are also (in fact mostly) interested in exhaustive and authoritative testing, so 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 [9].

The physical RF module used in the reference network, i.e., ALPHANET, is CC1100/CC430 manufactured by Texas Instruments [15]. As all devices in its class, CC1100 returns the value of *RSS* along with a received packet. The module provides for setting the transmit power in a flexible and dynamic manner with the maximum of 10 dBm.

The model

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 sender S, then its counterpart will (possibly) occur at a recipient R at time $t+d$, where d is the (current[4]) 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 $P_r = P_x/A$, where A is the attenuation value provided by the model instance (one of the in-

[4] The model incorporates node mobility.

stance functions) on the case-by-case basis. As the power is usually expressed in dBm and the attenuation in dB, the above formula can be written as:

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

The generic model invokes the instance function whenever it needs an attenuation value. The function can base its calculations on any conceivable (possibly dynamic) parameters involving *S* and *R*. In a simple case, it can apply some closed formula, e.g., one based exclusively 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. Again, 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. Suppose *R* is receiving some packet *p* at the signal level P_r. The value:

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

where *T* denotes the combined strength of all the interfering signals (i.e., all signals other than the one corresponding to *p*) and *B* is the *background noise* level, determines the signal-to-interference ratio for *p* at *R*. 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, i.e., a list of (sample) (*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)[5].

A typical reception model 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 a *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 a packet reception), the delay until the first er-

[5] One feature of the model is the possibility to abort a packet reception on an illegal symbol resulting from one or more incorrectly received (physical) bit(s).

ror bit is recalculated and the waiting is restarted from the moment of change[6]. The packet is deemed to have been received correctly, if the *bit error* event never occurs before the *last bit* event.

The specific part

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 sensible, default behavior if no close samples are available.

Formally, the fallback function, denoted $F(l)$, takes a distance as the argument and returns a pair (A,σ), where the first item is the mean value of attenuation for distance l, and the second one is the standard deviation used to randomize the actual instances of attenuation generated for the given distance. F is specified as a discrete list of triplets: (l, A, σ); the model applies interpolation to produce the values of F for arguments not directly covered by the list. By $F_A(l)$ and $F_\sigma(l)$ we shall denote the natural projections of $F(l)$.

As the samples are optional, while F is mandatory, the model will function without a single sample; so the role of the samples can be described as selectively replacing the fallback function F for those cases of attenuation that happen to be covered by the samples. This replacements is seamless and automatic. Adding more samples to the model will tend to improve its quality understood as the accuracy of approximation of the real environment.

A collected sample consists of two points in 3-space, S and R, representing the sender and the receiver, the transmission power P_x (in dBm) applied by the sender, and the *RSS* reading taken by the receiver. We can generally assume that the *RSS* value can be converted to some normalized power indication expressed in standard units. For example, in CC1100/CC430, the *RSS* (taken at the end of the sync word) translates into the actual received signal level in dBm discretized into 0.5 dBm steps[7].

When the set of samples is input to the model, P_x and *RSS* are preprocessed into a single measure of attenuation:

$$A|_{dB} = P_x|_{dBm} - RSS|_{dBm} \tag{3}$$

[6] 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 [16] which typically determine the packet's fate at a receiver only once.

[7] The trivial translation involves adjusting the value by a fixed offset.

The coordinates of the two endpoints *S* and *R* are discretized to the simulator's internal (settable) grid, typically 10 cm along each dimension, with points falling into the same grid cube considered indistinguishable. Multiple samples whose both ends become indistinguishable this way are merged into a single sample, their attenuation values averaged[8]. The model can be optionally declared as symmetric which means that *S* and *R* can be interchanged. In that case, the merging/averaging is also applied when the discretized endpoints of two input samples fall into the same grid cubes when *S* of one sample is compared to *R* of the other sample, and vice versa.

Suppose that *N* is the total number of stored (preprocessed) samples. A single sample S_i, $0 \le i < N$ is a 4-tuple:

$$S_i = \{S_i, R_i, A_i, \sigma_i\} \tag{4}$$

where $S_i = (x_s^i, y_s^i, z_s^i), R_i = (x_r^i, y_r^i, z_r^i)$ and σ_i is the standard deviation of A_i to be used for randomizing the specific instances of attenuation generated from the sample. If the stored sample is an average of multiple input samples, σ_i is the sampled standard deviation of the multiple input attenuation values contributing to A_i. Otherwise, σ_i is obtained as $F_\sigma(l_i)$, where $l_i = \sqrt{(x_r^i - x_s^i)^2 + (y_r^i - y_s^i)^2 + (z_r^i - z_s^i)^2}$ is the Euclidean distance between the sample's endpoints. The model can specify a threshold for the number of averaged input samples below which σ_i is determined from the fallback function *F* rather than as the sampled standard deviation of the input values.

When samples are present, the instance attenuation function combines them with *F* to produce weighted, randomized attenuation values, according to the procedure described below. The function is invoked for an (*S*,*R*) pair where $S = (x_s, y_s, z_s)$ and $R = (x_r, y_r, z_r)$. 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. Otherwise, the function produces an average from the closest samples also incorporating *F* in proportion to how close/far the closest samples are from the actual case. Formally, the algorithm proceeds as follows:

1. Initialize. Set $A_c = 0$, $\sigma_c = 0$, $l_c = 0$. These variables will be used to accumulate weighted averages over all samples. Set $W_c = 0$. This variable will store the sum of weights for normalization. Set $l = \sqrt{(x_r - x_s)^2 + (y_r - y_s)^2 + (z_r - z_s)^2}$ (the Euclidean distance between *S* and *R*). Then execute steps 2 through 4 for every sample *i*, $0 \le i < N$. When done, proceed at 5.
2. Calculate the sample's distance from the pair (*S*,*R*):

$$d_i = \sqrt{(x_s^i - x_s)^2 + (y_s^i - y_s)^2 + (z_s^i - z_s)^2} + \sqrt{(x_r^i - x_r)^2 + (y_r^i - y_r)^2 + (z_r^i - z_r)^2} \tag{5}$$

[8] When collecting samples, it is usually quite natural to take multiple readings for a given (S,R) pair.

If the channel has been declared as symmetric, then also calculate:

$$\hat{d}_i = \sqrt{\left(x_s^i - x_r\right)^2 + \left(y_s^i - y_r\right)^2 + \left(z_s^i - z_r\right)^2} + \sqrt{\left(x_r^i - x_s\right)^2 + \left(y_r^i - y_s\right)^2 + \left(z_r^i - z_s\right)^2} \tag{6}$$

and set $d_i = min\{\hat{d}_i, d_i\}$. Note that, owing to the grid, the distances are discretized so, for example, d_i cannot be arbitrarily close to zero, while being different from zero.

3. If $d_i = 0$, then we have found an exact match. Look no further. Set $A_c = A_i$, $\sigma_c = \sigma_i$ and proceed to 6.
4. Set:

$$(A_c, \sigma_c, l_c, W_c) = (A_c, \sigma_c, l_c, W_c) + \frac{(A_i, \sigma_i, l_i, 1)}{d_i} \tag{7}$$

The three attributes (A_i, σ_i, l_i) of the sample are added to (A_c, σ_c, l_c) with the factor of $1/d_i$ (note that $d_i > 0$). W_c stores the partial sum of the factors.

5. The loop has run out. Normalize the weighted averages:

$$(A_c, \sigma_c, l_c) = \frac{(A_c, \sigma_c, l_c)}{W_c} \tag{8}$$

and adjust the attenuation for distance:

$$A_c = A_c + F_A(l) - F_A(l_c) \tag{9}$$

6. Return:

$$A|_{dB} = A_c + X(\sigma_c) \tag{10}$$

The way the attributes of samples are weighed and combined with F targets the following objectives:

1. To make the impact of a given sample on the result proportional to its proximity to the (S,R) pair. This is why the attenuation value brought in by a given sample is factored with $1/d_i$, i.e., in reverse proportion to the sample's distance from the pair.
2. To make it possible to generate a sensible attenuation value when there is no close match by samples, in particular, when the distance l between S and R is non-receivable. In such a case, $F_A(l)$ should be factored into the result, in proportion to how poor the overall match actually is.

Note that in step 5, i.e., when there is no exact match, the mean attenuation (expressed in linear units) is in fact multiplied by the ratio $F_A(l)/F_A(l_c)$– see assignment (9). For example, if $l > l_c$, meaning that the distance between S and R is larger than the fitted average of the samples, the fallback attenuation for l is (likely) going to be higher than for l_c, so the factor will tend to increase the resulting attenuation. Conversely, if $l < l_c$, which means that S and R are closer than the best fitting samples, the attenuation will tend to be decreased respectively.

The above algorithm may seem overly complex for an application in a real-time model of a wireless channel. Note that every case of attenuation seems to require scanning through all the reference samples, or, in the lucky case of an exact match, through half of them on the average. For every transmission at some node S, potentially all nodes that can perceive any trace of the signal emitted by S must have the attenuation properly determined to assess the reception opportunity for the packet sent by S, or the interference level caused by it. To speed up the calculations, the function stashes the pair of values (A_c,σ_c) used in assignment (10) of the algorithm in a hash table indexed by the (discretized to the grid) coordinates of the (S,R) pair. The next time an attenuation value is needed for the same sets of coordinates, its mean and standard deviation will be taken from the hash table.

Feeding the model with data

In our reference application, *RSS* samples are routinely collected, as part of the system deployment, in the course of *profiling* needed by the location tracking service [9]. 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 eight *RSS* readings (for eight 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 five 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 the discrete 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.[9] 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., $P_x(i) = P_x(7) - Avg(RSS(7) - RSS(i))$, where the average is taken over all samples. Similarly, the rough dependence of attenuation on distance, for the fallback function *F*, can be determined by fitting the attenuation observed in the samples, which always comes with the 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 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,

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

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.

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 (and other activity) 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 "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.

Bibliography

Chrysikos T., Georgopoulos G., Kotsopoulos S. (2011), *Wireless channel characterization for a home indoor propagation topology at 2.4 ghz*, (in:) *Wireless Telecommunications Symposium (WTS)*, IEEE.

Fryziel M., Loyez C., Clavier L., Rolland N., Rolland P.A. (2002), *Path-loss model of the 60-ghz indoor radio channel*, "Microwave and Optical Technology Letters", Vol. 34, No. 3.

Haneda K., Jarvelainen J., Karttunen A., Kyro M., Putkonen J. (2015), *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.

Boban M. (2014), *Realistic and efficient channel modeling for vehicular networks*, arXiv preprint arXiv:1405.1008.

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

Ryckaert J., De Doncker P., Meys R., de Le Hoye A., Donnay S. (2004), *Channel model for wireless communication around human body*, "Electronics Letters", Vol. 40, No. 9.

Fort A., Desset C., De Doncker P., Wambacq P., Van Biesen L. (2006), *An ultra-wideband body area propagation channel model-from statistics to implementation*, "Microwave Theory and Techniques, IEEE Transactions on", Vol. 54.

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

Gburzyński P., Olesiński W., Van Vooren J., *A WSN-based, RSS-driven, real-time location tracking system for independent living facilities*, (in:) *13-th International Joint Conference on e-Business and Telecommunications*, Lisbon, Portugal.

Whitehouse K., Tolle G., Taneja J., Sharp C., Kim S., Jeong J., Hui J., Dutta P., Culler D. (2006), *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.

Levis P., Culler D. (2002), *Maté: A tiny virtual machine for sensor networks*, "ACM Sigplan Notices", Vol. 37, No. 10.

Fok C.-L., Roman G.-C., Lu C. (2005), *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.

Gburzyński P., Olesiński W. (2008), *On a practical approach to low-cost ad hoc wireless networking*, "Journal of Telecommunications and Information Technology", Jan.

Boers N., Gburzyński P., Nikolaidis I., Olesiński W. (2010), *Developing wireless sensor network applications in a virtual environment*, "Telecommunication Systems", Vol. 45, No. 2-3.

Texas Instruments (2014), "CC1100 Single Chip Low Cost Low Power RF Transceiver", Document SWRS038D.

Mahrenholz D., Ivanov S. (2004), *Real-time network emulation with ns-2*, (in:) *Proceedings of Eighth IEEE International Symposium on Distributed Simulation and Real-Time Applications*.

Model kanału radiowego oparty na próbkach sygnału dla testowania aplikacji bezprzewodowych sieci sensorów

Streszczenie

W artykule zaprezentowano generyczny model kanału radiowego parametryzowany próbkami sygnału pobranymi z otoczenia. Model jest przeznaczony do wykorzystania w symulacji bezprzewodowych sieci sensorowych, dla weryfikacji oraz testowania poprzedzającego fizyczną instalację. Charakter modelu pozwala na skonfigurowanie go w system wirtualnego wykonania, co umożliwia wirtualne wykonanie pełnej aplikacji sieciowej, włącznie z jej zewnętrznymi programami interfejsu z użytkownikiem. Dzięki temu aplikacja sieciowa oraz rzeczone programy zewnętrzne mogą być wygodnie projektowane, tworzone i następnie weryfikowane w środowisku wirtualnym, bez uciążliwej konieczności instalowania fragmentów rzeczywistej sieci. Wykazano zalety prezentowanego podejścia na przykładzie aplikacji, w której zgodność modelu kanału radiowego z rzeczywistością jest szczególnie istotna, a mianowicie w systemie lokalizacji wewnątrz budynków.

Słowa kluczowe: komunikacja bezprzewodowa, symulacja, modele kanałów radiowych.

Kody JEL: L86

Artykuł zaakceptowany do druku w lutym 2018 roku

Afiliacja:
dr hab. Paweł Gburzyński
Akademia Finansów i Biznesu Vistula
Wydział Inżynierski
ul. Stokłosy 3
02-787 Warszawa
e-mail: p.gburzynski@vistula.edu.pl