

MEng Project Report:
OLA Network
Cooperative Transmission in Wireless
Multi-hop Ad Hoc Networks

Dalia Burgos
School of ECE, Cornell University
Ithaca, NY 14850
drb28@cornell.edu

July 22, 2003

Abstract

This research explores a new technique proposed on how to flood a local area network asynchronously. The idea is simple. A node designated as a leader transmits signals. Each close by transceiver adaptively determines the network signature and detects the symbol sent by a designated leader and finally echoes the symbol. Thus, the sum of the signals as it passes through each relay in the network forms the Opportunistic Large Array(OLA). Our task is to properly simulate such a system using MATLAB and thereby confirm the theoretical derivations with the experimental results. In particular, we are concerned with the performance of the system assuming errors were made at each relay. The advantages of such a system are (1) privacy, this jam-resistance property is a byproduct of the spread of the signal; thus even if the signal (OLA) is intercepted by unauthorized users they will have difficulty decoding it without knowledge of the signal; (2) diversity caused by random fading paths followed by the signals; (3) flexibility, no time coordination, no contention is necessary; and (4) scalability, the addition of more nodes to the network or add another network to transmit to the same long distance receiver.

Contents

1	Introduction	1
2	System Model	2
2.1	Network Setup	2
3	Transmission	4
3.1	Constructing OLA signal	5
3.2	Simulation of OLA Transmission	7
4	Threshold Criteria	9
4.1	Simulating the Threshold constraint	11
5	Training	12
5.1	Training Simulation	14
6	Detection	14
6.1	Simulations	16
7	Performance	17
7.1	MSE affects Performance	18
7.2	Error Propagation affects Performance	20
8	Conclusion	23
	Appendix	24

A Program Listing	24
A.1 Network setup	24
A.2 MSE files	24
A.3 Error Propagation files	29
References	38

List of Figures

1	Block Diagram of System Model	3
2	Network Setup with corresponding firing order	4
3	Alternate Block Diagram of DT System Model	7
4	Node signature: $s_{i,m}(t)$	9
5	Effect of Error Propagation on Performance	17
6	Effect of MSE on Performance	20
7	MSE for different SNR	21
8	Theoretical Performance Plots	22

List of Tables

1	Network Configurations	8
---	----------------------------------	---

1 Introduction

In Wireless communications, fast and reliable transmission of data is of great importance. In this report, we analyze a new transmission scheme proposed by Prof. Anna Scaglione and Yao-Win Hong¹. We concentrated on the construction of the OLA signals and the performance of the transmission scheme. OLA is a flooding algorithm that advantageously eliminates routing and multiple access overheads. We organize a multi-hop network consisting of close-by low transmission power transceivers. A signal is transmitted from one node (or a few) designated as the leader using a specified modulation. The network is flooded using multi-hop relays. The key idea is for each node to act like a repeater and echo the symbol sent by the leader. At each relay, the node is an adaptive receiver, capturing the network signature which is the accumulated sum of the echoed signals of all the nodes that have previously transmitted. Thus, the interference of the signals provides a multi-user diversity as long as we can track the changes and discriminate between the sent symbols. In a given communication system, we are allotted a limited bandwidth. Hence, our goal is to make the best use of our resources to increase throughput (data rates).

We believe that the OLA structure generates a natural multi-access system as all the nodes communicate with each other at specified sub-bands of frequency. Essentially, this is a frequency spread spectrum system since it occupies more bandwidth than the actual bandwidth required to transmit the information. Also, simultaneous communication causes signal interference which in turn provides redundancy and increases the signal's power. Hence, we can use a lower transmitter radiated power and a narrower channel bandwidth. ([2] p.657-658) In addition, we want to reduce the detection errors made at each node. Hence, we add a proper SNR constraint on each

¹Part of the ECE Department at Cornell University

transceiver, and thereby bring the probability of error to negligible levels.

2 System Model

The system is broken up into four steps. First, the network is set up to ensure connectivity among all transceivers in order to have a cooperative transmission. This is done by manipulating the distance between nodes, the transmitter's power, and the SNR constraint. Second, the leader transmits M-ary signal waveforms. This is simulated as signals passing through a channel. Each node receives a different waveform, $s_{i,m}(t)$, which we will henceforth call its signature. Third, each receiver has the job of adaptively estimating the signature. Fourth, the node chooses whether to transmit or abstain. If it does transmit, it follows to detect the symbol that was sent, and transmits the corresponding symbol's waveform. (see Figure [1]) All nodes perform the same operation. We conducted all simulations using MATLAB. Throughout the paper, I will refer to the following files *node_setup.m*, *OLA-transmission.m*, *error_propag.m*, and *plot_BER.m*. (See Appendix A)

2.1 Network Setup

Consider an ad-hoc network of N nodes uniformly and randomly distributed within a circular area of radius R. This wireless system is unique in that there is no synchronized transmission times or routing protocols. Thus, certain overheads are eliminated. The density of the network is determined by the transmission power of the transceiver. It is assumed that all nodes are identical and not powerful enough to communicate to a far receiver. Hence, asynchronous multi-hop paths are required. The advantage of having many nodes with low transmitting power as opposed to a high power transmitter

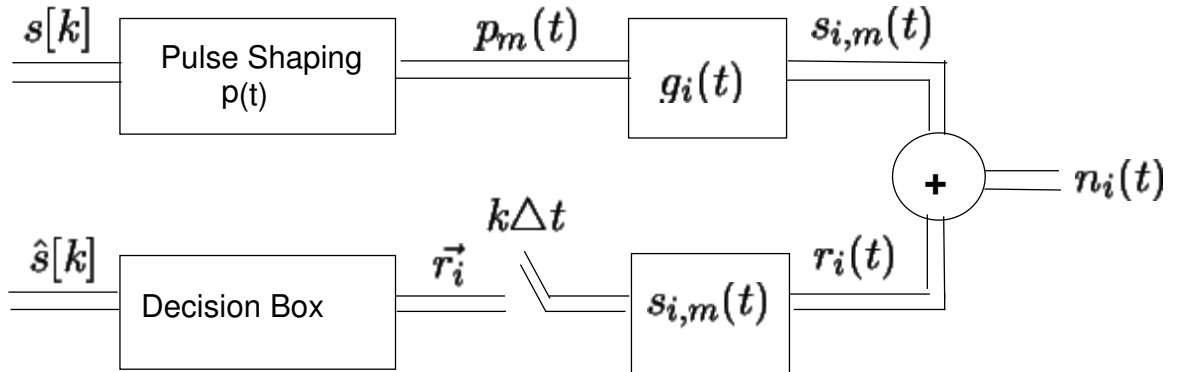


Figure 1: Block Diagram of System Model

is security. The system is designed to continue to operate even if a few nodes are deactivated. Thus, the OLA system is less susceptible to failure and outside attacks. We determined that for a 100mW transceiver, a network of 60 nodes in a 10 m radius is enough to ensure the flooding of the entire network under the assumption that all nodes correctly echoes the leader's signals. The minimum point to point distance is .99 and on the average they are 1.9m apart. The figure below shows the physical network of the system. The numbers correspond to the order in which the nodes' fire. We'll discuss the criteria for determining when a node should transmit in section (4).

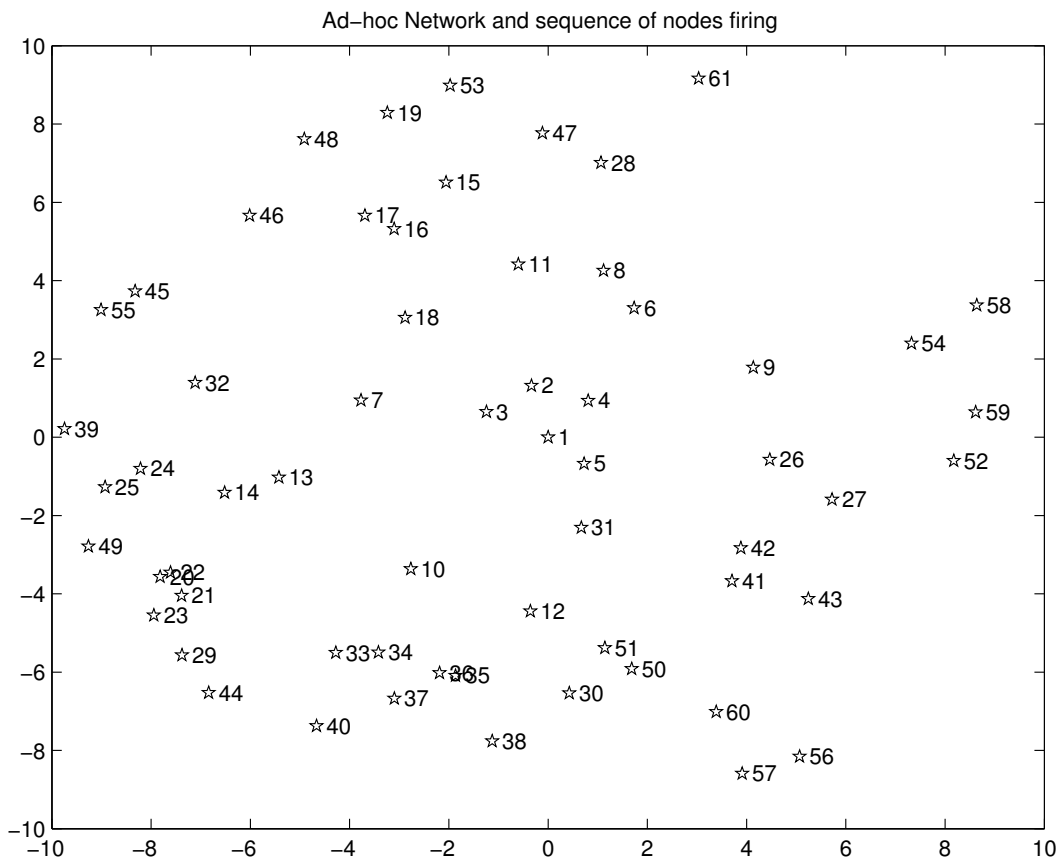


Figure 2: Network Setup with corresponding firing order

3 Transmission

For our modulation scheme, we will use MPSK with alphabet $s_m = \{e^{j\frac{2\pi k}{M}}\}$. To simplify our model, we only considered BPSK modulation and demodulation. We designate the center node as the leader, the only node transmitting data to flood the entire network. Hence, the binary symbols ± 1 are sent by the leader. Its corresponding complex waveform equivalent is $p_m(t)$, with duration $T_p = \frac{1}{W}$ and one-sided bandwidth, W (see equation (1)).

$$p_m(t) = s_m \sqrt{W} \text{sinc}(Wt) \quad (1)$$

3.1 Constructing OLA signal

The leader broadcasts a waveform, $p_m(t)$. Signals experience a node-to-node Rayleigh fading, which means that the fading path to each node is different. Thus, the rest of the nodes receive a delayed and attenuated version of the signal. Let, $A_{i,n}$ be the channel gain and $\tau_{i,n}$ be the delay of the path from node n to node i . The coefficient, equation (2), is the product of a complex fading random variable following a rayleigh distribution times the transmit power, P_t , times the inverse of the distance squared, $d_{i,n}$. The delay is the time it takes for a waveform to reach its destination point, equation (3). A node waits, then detects and finally also broadcasts a waveform, $p_m(t)$. Thus, nodes further down the relay will also receive the signal from this node. A node that is ready to fire has received the pulses from all the previous firing nodes.

T_s is the duration of the OLA signal, meaning that by this time the outermost nodes in the network have received the signals and have transmitted its own echo. Let $\tau_f^i(t)$ be the firing time of the i th node that fired. Thus, the nodes are ordered according to their firing times in ascending order, $\tau_f^1, \dots, \tau_f^N$. Then, T_s is approximately $\tau_f^N - \tau_f^1$. Suppose, in the worst case scenario, node i waits the entire T_p after it first starts receiving a signal period before transmitting. We assume, the node receives nothing prior to \bar{t} and that by waiting T_p it should assure to have received the entire pulse if not more. We define set Λ as the nodes with firing time less than \bar{t} . The m th signal received by this node is then the sum of all pulses transmitted prior to \bar{t} , equation (4). Furthermore, this equation can be broken down into the leader's transmitted pulse and the network impulse response as shown

in equation (5). Hence, each node has its own network impulse response $g_i(t)$. The output of this pulse sent through this channel is called the node's signature, $s_{i,m}(t)$.

$$A_{i,n}(t) = \sqrt{P_t} \frac{\sqrt{0.5}(\text{randn}(1) + j * \text{randn}(1))}{(1 + d_{i,n})^2} \quad (2)$$

$$\tau_{i,n}(t) = \frac{d}{c} \quad (3)$$

The received signal, $r_i(t)$ is the signature with added gaussian noise. When a node detects a symbol, it takes the received signal passes it through a received filter, samples it by Δt and makes a decision based on the alphabet. This matched filter is the node's signature which is obtained through training. If we sample at the Nyquist rate, $\Delta t = T_p$, $p_m(t)$ can be expressed with one sample, $N_p = 1$ and thus $\mathbf{s}_{i,m} = s_m \mathbf{g}_i$. Sampling also causes the frequency spectrum to be scaled by its sampling rate, Δt . Hence, the signature is scaled by $\sqrt{\Delta t}$ to maintain a normalized power spectrum. (see figure [4]). Lastly, the node broadcasts the regenerated symbol waveform.

$$\begin{aligned} s_{i,m}(t) &= \sum_{n \in \Lambda} A_{i,n} p_m(t - \tau_{i,n}) \\ &= p_m(t) \otimes \sum_{n \in \Lambda} A_{i,n} \delta(t - \tau_{i,n}) \\ &= p_m(t) \otimes g_i(t) \end{aligned} \quad (4)$$

$$\text{thus, } g_i(t) = \sum_{n \in \Lambda} A_{i,n} \delta(t - \tau_{i,n}) \quad (5)$$

$$\text{where, } \Lambda = \{n : \tau_f^n > \tau_f^i\} \text{ and } k = 0, \dots, N_s$$

3.2 Simulation of OLA Transmission

Our DT equivalent model of our OLA system is shown below. First, we formulate $\mathbf{s}_{i,m}$, as in equation (7). Then, add discrete gaussian noise to get \mathbf{r}_i . We used the network configurations as tabulated in table 1. Note that when we sample the received signal, we are sampling $g_i(t)$

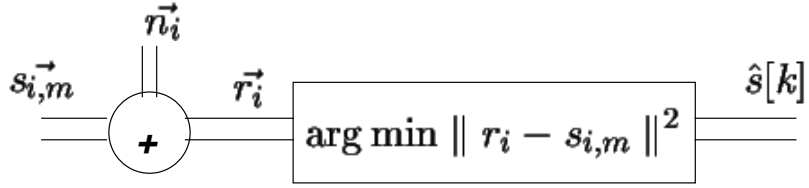


Figure 3: Alternate Block Diagram of DT System Model

$$\mathbf{r}_i = \mathbf{s}_{i,m} + \mathbf{n}_i \text{ where, } \mathbf{n}_i \sim \mathfrak{N}(0, N_o) \quad (6)$$

which is simply the sum of dirac deltas. In Matlab, the discrete impulse response, \mathbf{g}_i , is obtained through interpolation and over-sampling of the continuous network impulse response. (See equation (8)) So we chose to sample at $\Delta t = T_p/10$ to obtain $\frac{T_p}{\Delta t} = 10$ samples in one pulse duration. This is done to improve the time resolution constraint of discrete signals. Figure [4] shows the formulation of the 3rd node's signature. It's the sum of the pulse broadcasted by node1 and node2. In this case, $\mathbf{s}_{i,m}$ is composed of a pulse passed through a channel with 2 dispersive paths, i.e. $g_i(t) = .1\delta(t) + .05\delta(t - 50\Delta t)$.

$$\begin{aligned} \mathbf{p}_m &= p_m(kT_p) \text{ for } k = 0 : N_p - 1 \\ \mathbf{s}_{i,m} &= s_{i,m}(k\Delta t) = \mathbf{p}_m \otimes \mathbf{g}_i = \sum_{j=0}^{N_p-1} \{p_m\}_k \{g_i\}_{k-j} = s_m \mathbf{g}_i \end{aligned} \quad (7)$$

Parameter	value
Modulation	BPSK
c	3×10^8
BW=Bandwidth	83.5 Mbps
N= number of nodes	60
radius	10m
Δt = sampling rate	$\frac{T_p}{10}$
T_s =OLA signal duration	$1500 \times \Delta t$
T_p =pulse duration	11.87 ns
P_t	100W, 200 W

Table 1: Network Configurations

$$\begin{aligned}
\mathbf{g}_i &= \sqrt{W\Delta t} g_i(t) \otimes \text{rect}\left(\frac{t}{T_p}\right) \Big|_{t=k\Delta t} \\
&= \sum_{n \in \Lambda} A_{i,n} \text{sinc}(W(k\Delta t + l_i\Delta t - \tau_{i,n}))
\end{aligned} \tag{8}$$

In this network, propagation delays and firing times are time sensitive because we are operating at high data rates. Since the samples are discrete, then to node i , all nodes within a distance range of $[d_{k-1}, d_k]$ are viewed as having the same distance from it. Thus, there will be multiple nodes with the same propagation delays. The set of nodes appearing to have the same transmission to the i th node are those whose difference in propagation delay is less than Δt , equation (9).

$$R_{i,k}^p = \left\{ n : \frac{\tau_{i,n}}{\Delta t} = k \right\} \tag{9}$$

So, we decrease Δt , to obtain a more accurate time reading and reduce the number of nodes in the unresolvable region. We define each time unit, Δt , to equal $\frac{T_p}{10}$. Thus, we convert propagation delays, $\tau_{i,n}$, to discrete time

units by scaling by $\frac{1}{\Delta t}$. However, the firing times don't need conversions since we already compute τ_f^n from discrete samples (in section (4) will discuss how to obtain τ_f^n). The signal arriving at a node is the sum of all delayed signals from nodes that have fired.

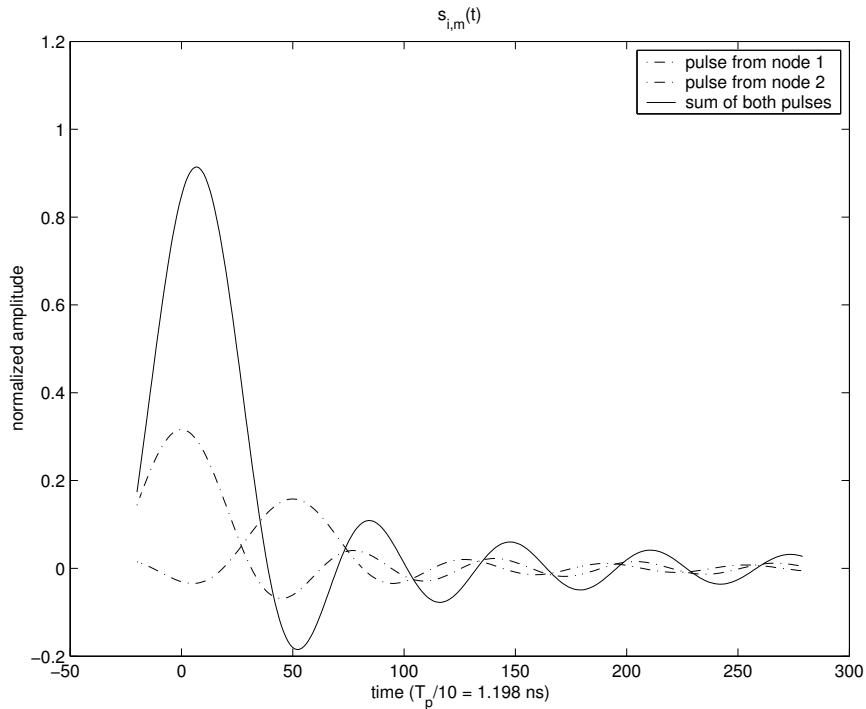


Figure 4: Node signature: $s_{i,m}(t)$

4 Threshold Criteria

When the leader transmits, the closest neighbors will receive the signal first and with minimum attenuation. We assume they will be among the first to transmit. We will exploit the fact that the OLA signal's power accumulates

in time as it receives more and more signals from its neighboring nodes. However, the power varies depending on the density of the network and the power attenuation. So, one of the advantages of using the OLA technique is shortening the overall transmission time. Each transceiver acts like a repeater by transmitting the signal it has detected using its estimated network response obtain from training, $\hat{g}_i(t)$ (to be discussed in section (5)). In a densely populated region, the signature power received by those nodes is strong. Hence, they should be able to transmit right away instead of waiting the entire T_p and receive weak delayed signals from the rest of the transmitting nodes. So how do we quantify this?

Each node operates in two modes: (1) the receive phase, in which the node waits until it presumes to provide "reliable" detection of the signal at which point it 'fires'; and (2) the rest phase, the node silently waits till the end of T_s without any further transmissions. Thus, with each symbol, each node has the option of retransmitting or opting to stay silent if it believes that it's decisions will be unreliable. The node makes this decision based on whether or not it satisfies the connectivity rule for the regenerative scheme defined in [1]:

The i -th regenerative node is connected if, ..., the pairwise symbol error probability of the i th receiver is below a fixed upper bound ϵ .

Thus, the node's firing time is chosen such that the rule is satisfied. Since we use a minimum distance detector, then, the probability of error is solely determined by the Signal to Noise ratio (SNR):

$$P_{BPSK}(E) = P(m \rightarrow \mu) = Q\left(\frac{d}{2\sigma}\right) = Q\left(\sqrt{\frac{2E_b}{N_o}}\right) \quad (10)$$

Since noise is an inherent parameter, it is given a specific value. The only variable left is the signal energy. Therefore, in the receive phase, the receiver

checks if the upper bound, ϵ , is satisfied by continuously polling to get an updated reading of the received signal's energy. The firing time is then the instant the signal energy exceeds a threshold.

4.1 Simulating the Threshold constraint

This network system is simulated in MATLAB by maintaining an N by N_s dimensional array of the signals received at each node, s . Once the nodes transmits, the signature is stored in the array g . Suppose, we want to guarantee the system's performance with an upper bound of ϵ . First, we fix the noise, $N_o = 1$. To obtain such a request, $E_b = \|g_i\|^2$ must equal the threshold ζ .

At the start of data transmission, the leader fires its pulse, $\mathbf{p}_m = s_m$, at firing time $\tau_f^1 = 0$. This pulse is then added to all nodes with it's appropriate delay, $\tau_{i,n}$. That is, for the rest of the nodes, indexed by n , $s_n = s_n + A_{1,n}\sqrt{W\Delta t}\text{sinc}(W(k - \tau_{1,n} - \tau_f^1)/\pi)$. Next, we determine the next nodes to transmit. To find the firing time, an energy search is conducted to determine at which time instant the threshold is exceeded, τ_f^n . The signals' energies are checked to see which ones have achieved the desired energy threshold. If the energy of the n th node satisfies this constraint, (i.e. $E_{s_n} = E_b$), then we assume it is ready to transmit. Note that many nodes can simultaneously reach the energy constraint. Thus, there exists a set of nodes, $R_{i,k}^f$, equation (11), that to the i th node appear to transmit simultaneously. Refer back to the discussion about time resolution in section (3.2).

$$R_{i,k}^f = \{n : \tau_f^n = k\} \quad (11)$$

For this simulation, we made several assumptions. Since we assume that for a long specified period of time (mT_s), $A_{i,n}(t)$ and $\tau_{i,n}(t)$ are constant and

we transmit the correct training symbol. Then, we conclude that the nodes will have a fixed firing time. Thus, the firing times for the nodes and the sequence in which they transmit is obtain with a single transmission by the leader. We then know which node fired first, second, third, and so on.

5 Training

The receiver makes a decision based on the received signal and the transmitted signal. Unfortunately, the transmitted signal is the output from the time varying channel. $\mathbf{s}_{i,m}(t)$ is composed of all the pulses transmitted by nodes that have exceeded the energy threshold. Before those pulses reached node i , they followed different paths, experienced different attenuations, and their time of arrival depended on when the previous nodes had fired. To obtain $s_{i,m}(t)$, we had two options. First, each node can find all three parameters: $A_{i,n}$, $\tau_{i,n}$, and τ_f^n and formulate the signature. But, this is too difficult. Instead, we take advantage that since we are dealing with stationary transmitters then the signals will have only small variations that the receiver can adaptively track. Thus the second option is to estimate the channel (or signature). For our receiver design, we opted to implement the pure signal estimation through training.

The steps for finding the estimation signal is described as follows. Estimated $s_{i,m}(t)$ is obtained by the leader transmitting L symbols which is known as the training sequence. Since we are dealing only with BPSK, the leader sends all ones. Thereby, for each symbol, the receiver will acquire the waveform, $s_{i,1}(t)$, corresponding to $s_1 = 1$, which equals to the signature, $g_i(t)$. By symmetry, $s_{i,0}(t)$ is then the negation of the signature. (Note that for M-PSK signals, the receiver will need to keep track of only $\frac{M}{2}$ signatures because $\pm e^{\frac{j2\pi k}{M}}$ for $k = 1, \dots, \frac{M}{2}$).

The firing times for the nodes and the order in which they fire are obtained with the first training symbol transmission including the gaussian additive noise and are assumed to remain constant for the rest of the transmission period. Transmission is the same as described in section (3) with the exception that all nodes transmit the training symbol. In summary, the network is flooded by the leader. All nodes check to see if they have reached the desired threshold. If so, they transmit the training pulse to all the nodes. The threshold check and subsequent transmission is repeated until all nodes have fired. During the rest of the training phase, the nodes keep track of the cumulative sum of the received signal, $r_{i,m}(t)$. At the end of the period, we divide this sum by L . Thus, the estimated signature is defined as the average of all the received training signals, equation (13).

Ideal $g_i(t)$ is obtained by flooding the network with a single transmission of a BPSK value of 1 and omitting the noise in the channel. Estimated $g_i(t)$ is obtained by transmitting L training sequence.

$$N_p=1, \text{ Thus, } \mathbf{p}_m = s_m$$

$$\mathbf{s}_{i,m} = s_m \mathbf{g}_i \quad (12)$$

$$\hat{\mathbf{s}}_{i,m} = \frac{1}{L} \sum_{i=1}^L \mathbf{s}_{i,m} \quad (13)$$

$$\text{Where, } \mathbf{g}_i = \sqrt{(W\Delta t)} \sum_{n \in \Lambda} A_{i,n} \text{sinc}(W(k\Delta t + l_i\Delta t - \tau_{i,n})) \Big|_{t=k\Delta t} \quad (14)$$

if $N_p = 1$, and $s_m = 1$

then, $\mathbf{g}_i = \mathbf{s}_{i,m}$, and $\hat{\mathbf{g}}_i = \hat{\mathbf{s}}_{i,m}$

and, $MSE = E \left\{ \|\hat{\mathbf{g}}_i - \mathbf{g}_i\|^2 \right\}$

5.1 Training Simulation

This is the process of acquiring the signal space. In MATLAB, all node signatures are stored in an N by N_s array labelled \mathbf{g} . (Note: for M-PSK the array would be $N \frac{M}{2} \times N_s$) This signal space will be used for detection. Due to this estimation approximation, the optimum ML detector behaves differently from the ML detector in AWGN. Thus, for performance evaluation, the signal estimation error is modelled as an addition to the noise variance. And if we make this error small enough, the detector's performance will be the same as that of the minimum distance detector for AWGN. The mean square estimation error is theoretically computed in [1] as equation (16). As a check, the mean square error (MSE) of the signature can be computed experimentally as the average of square error of the difference between the estimated signal and the ideal signal, equation (15). For a fixed training length, both MSE should be equivalent. Note, that as the length of the training sequence approaches infinite the average will resemble the actual signal, i.e. MSE approaches zero.

$$MSE = \frac{1}{MC} \sum_{i=1}^{MC} \|\hat{\mathbf{s}}_{i,m} - \mathbf{s}_{i,m}\|^2 = \frac{1}{MC} \sum_{i=1}^{MC} \|\hat{\mathbf{s}}_{i,m} - s_m \mathbf{g}_i\|^2 \quad (15)$$

$$MSE = E \left\{ \|\hat{\mathbf{s}}_{i,m} - \mathbf{s}_{i,m}\|^2 \right\} = E \left\{ \|\hat{\mathbf{s}}_{i,m} - s_m \mathbf{g}_i\|^2 \right\} = \frac{E \left\{ \|n_i\|^2 \right\}}{L}$$

where, $E \left\{ \|n_i\|^2 \right\} = \tau_f^i \frac{N_0}{2}$ (16)

6 Detection

We have stated that each node uniquely receives a different signature due to diversity. Thus, the node must decide whether to contribute to the OLA

signal or remain silent. Its criteria for transmitting depends on its own evaluation of successfully detecting, refer to the connectivity rule discussed in section (4). In other words, if it transmits, then the node has reached a threshold energy, E_b . In the receive phase, the node acts as a receiver. The vector \mathbf{r}_i is received. It's modelled as the node's signature, $\mathbf{s}_{i,m}$, as defined in equation (7), plus some additive white gaussian noise, \mathbf{n}_i . Assuming, the node reaches the required E_b , then it detects using a maximum likelihood detector. For this model, the ML detector happens to be a minimum distance detector (see equation (17)).

$$\begin{aligned}
p(r_i|s_{i,m}) &= \frac{1}{(2\pi N_o)^{\frac{N}{2}}} e^{-\frac{1}{N_o} \|\mathbf{r}_i - \mathbf{s}_{i,m}\|^2} \\
\mathbf{n}_i &\sim \mathfrak{N}(0, N_o) \\
\hat{s}_{i,m} &= \arg \max_m p(r_i|s_{i,m}) \\
&= \arg \max_m \ln p(r_i|s_{i,m}) \\
&= \arg \max_m \frac{-1}{N_o} \|\mathbf{r}_i - \mathbf{s}_{i,m}\|^2 + const \\
&= \arg \min_m \|\mathbf{r}_i - \mathbf{s}_{i,m}\|^2
\end{aligned} \tag{17}$$

The file *OLATransmission.m*, simulates the formation of the OLA signal for k data symbols. For each symbol, the transmission process is the same. First, the leader's pulse floods the entire network. Thereafter, each node fires when it achieves a specified energy requirement. We have made the assumption that each node will have the same performance throughout the entire data transmission. Hence, we use the same firing time predetermined during the training. The node re-initializes every T_s . At the firing instant, the node detects by choosing the signature yielding the minimum distance between the received vector and all the signatures. That is, when it's the i th node to fire, it has received \mathbf{r}_i . Then it checks the distance between it's

vector and $s_m \mathbf{g}_i$. Recall, we obtained \mathbf{g}_i through training. Furthermore, the node regenerative transmit its detected symbol. Lastly, the node remains in the rest phase for the rest of Transmission period.

Throughout, the data transmission we keep track of the performance of each individual node. That is, we obtain the average detection error. We expect that in an ideal channel all nodes will retransmit the symbol originally sent by the leader. In which case, the performance of all the nodes will follow probability of error requirement, ϵ . Unfortunately, when a node incorrectly detects, it transmits a waveform that will change the received signature, $\mathbf{s}_{i,m}$, at the other nodes. This effect is called error propagation and it degrades the node's error performance.

6.1 Simulations

The entire OLA system is simulated using the file *OLAsimulate.m*. For different snr's requirement, the 60 node network follows 3 steps. First, *OLA-training.m* is used to obtain the estimate of the signal's signature for all nodes. The signature is trained for the specified threshold constraint. Second, *OLAtransmission.m* is used to flood the network with the k BPSK data symbols. Each node performs its detection at the firing time, τ_f , obtained during training. Any error that a node makes is logged for future performance evaluation.

We ran two simulation to test the performance of the OLA network with and without error propagation. This is done by enabling and disabling the 'error_prop' flag before running *OLAsimulate.m*. Note that to ensure a probability of error the transmitter power must be high enough to ensure flooding of the entire network. Hence, for higher snrs of 5 to 7, we doubled the transmitter power to 200 mW. Figure [5] shows the results obtained

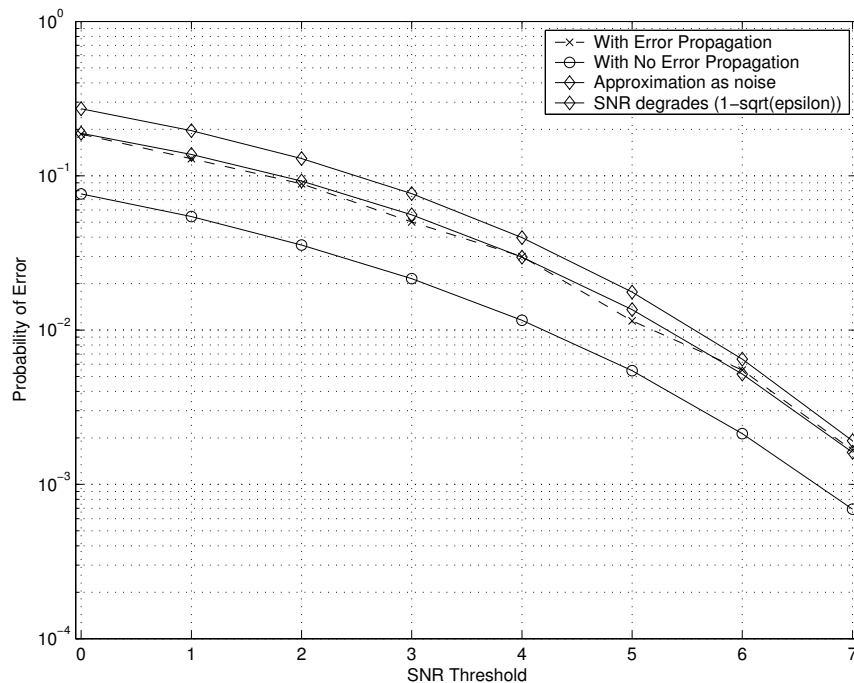


Figure 5: Effect of Error Propagation on Performance

through simulations. Next we will show that the theoretical formulas for the OLA system match these results.

7 Performance

Ideally, we expect that the average error performance for all the nodes will be $Q\left(\sqrt{\frac{2E_b}{N_o}}\right)$. That is, if all nodes transmit the correct symbol waveform, then the node's probability of error is ϵ . This BER formula matches the result obtained through simulation. The simulation consisted of flooding network and allowing no error propagation. However, this is not always the case. There are other sources of errors derived from the estimation of the

signature space, and error propagation.

7.1 MSE affects Performance

The actual signature differs from the estimate acquired from training. This difference is measured by its MSE. Apparently, this error measure is related to the length of the training sequence. Ideally, we want to use a large enough training sequence so that the MSE approaches zero. So how large is large enough? It depends. There's always the tradeoff of the training overhead versus performance. Thus we want the training sequence to be large enough to guarantee the same performance assuming the estimate was perfect. Figure [7] shows the performance of the OLA system if we consider MSE to be negligible.

How does MSE affect the performance of the receiver? If we model the received vector as the received signature plus some noise, (equation (18)) we show that the error induced by training is added to the noise variance. Thus, we obtain a new probability of error, (equation (19)).

$$\mathbf{g}_i = \hat{\mathbf{g}}_i + \mathbf{e} \quad (18)$$

$$P(E) = Q\left(\sqrt{\frac{E_b}{MSE + \frac{N_o}{2}}}\right) \quad (19)$$

The following is the derivation of the optimum detector using the estimate of the signature as obtained from [1]:

$$p(r_i | s_{i,m}) = \frac{1}{(2\pi\sigma)^{\frac{N}{2}}} e^{-\frac{1}{\sigma} \|\mathbf{r}_i - \hat{\mathbf{s}}_{i,m}\|^2}$$

$$\mathbf{p}_m \otimes \mathbf{e} + \mathbf{n}_i \sim \mathfrak{N}(0, \sigma = N_o + |p_m|^2 MSE)$$

$$\begin{aligned}
\hat{s}_m &= \arg \max_m p(r_i | s_{i,m}) \\
&= \arg \max_m \ln p(r_i | s_{i,m}) \\
&= \arg \max_m -\|\mathbf{r}_i - \hat{\mathbf{s}}_{i,m}\|^2 - N\sigma \ln(2\pi\sigma) \\
&= \arg \min_m \|\mathbf{r}_i - \hat{\mathbf{s}}_{i,m}\|^2
\end{aligned} \tag{20}$$

We know that a larger training length lowers the MSE. Thus, we search for a training length until we obtain the same (or close enough) performance (i.e. for a desired snr constraint) as in the figure. For simulation purposes, we want to obtain the training length without simulating the actual training phase because it will require a long time. Instead, we obtain the desired training length by using the theoretical equation. We checked that both methods give the same MSE for a specified training length.

The first set up requires computing the theoretical MSE. First, ideal training as described in section (5) is used to obtain the firing times. Then, the training length needed to obtain an ideal estimate of the signature for a given snr value is found by using file *findtrainLen.m*. The second set up simulates the actual training to obtain the experimental MSE. This set up uses files *MSEtraining.m*, and *MSEsimulate.m*

Using the same training length of 10000 for both set ups, the MSE should be the same. We add the MSEs to the noise variance to produce identical performance plots as seen in the figure below. This figure is plotted using *MSEplot.m* Note that for higher snr values, the estimate must be very accurate. Thus, the training length needs to be increase in order to decrease the MSE and thus smooth out the added noise. Actual training with a large training sequence is time consuming. Since, they are equivalent, we can make the following simplification. We use the first setup from here on.

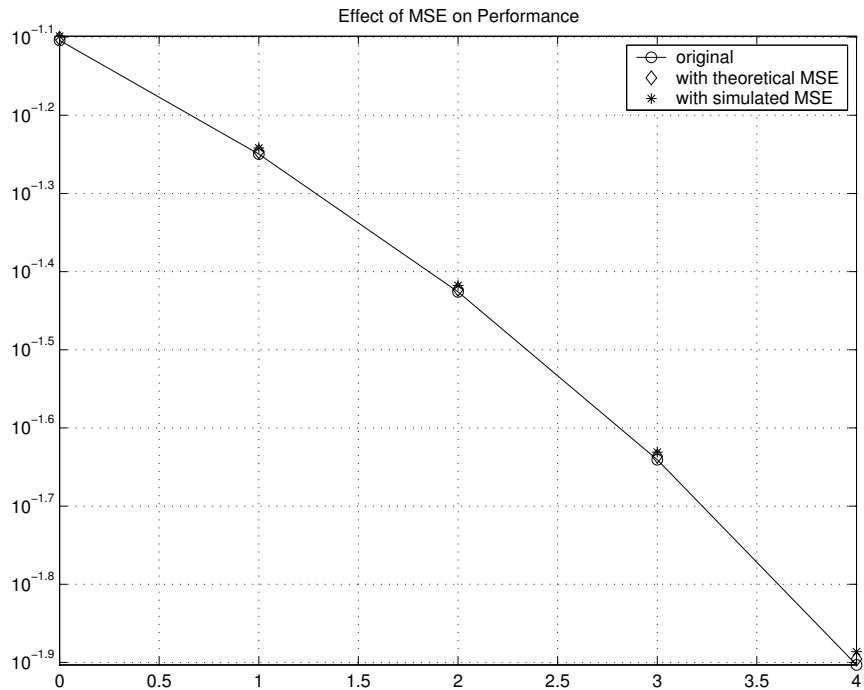


Figure 6: Effect of MSE on Performance

7.2 Error Propagation affects Performance

Unfortunately, since each node has the same probability of making a detection error, its corrupted signal gets added to the rest of the network's signatures. This error propagation lowers the overall performance of the nodes. However, due to signal attenuation and delay, only the closest neighbors are affected by this error. The nodes that are too far away receive only a small energy contribution, too insignificant to affect its detection capabilities.

Essentially, error propagation affects a minimum distance detector in the worst way, by reducing the distance between the signatures. With closer decision regions, the detector is more prone to err. In [1], the effect of error propagation is also modelled as an additive noise contribution. The error

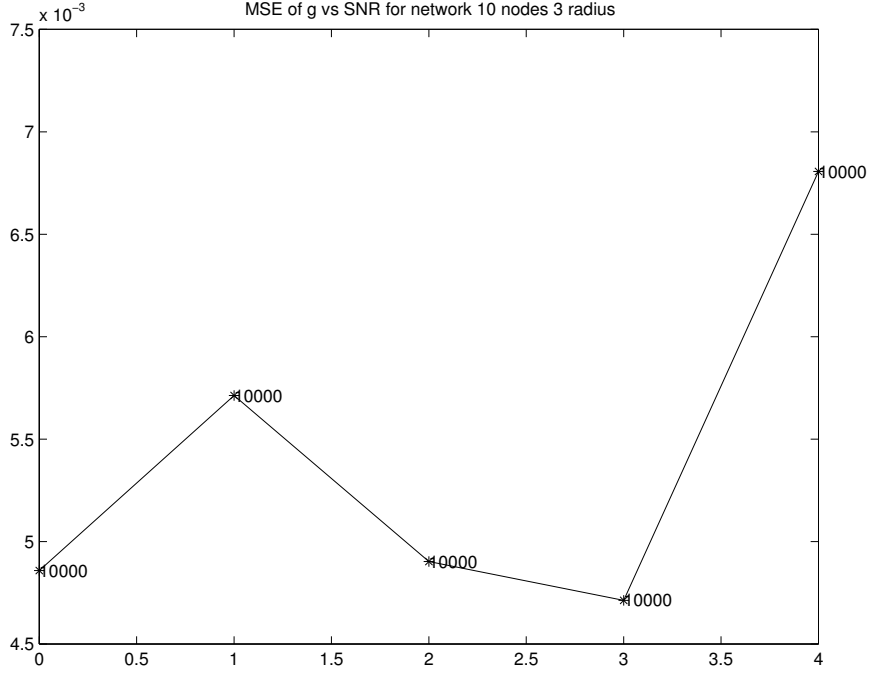


Figure 7: MSE for different SNR

vector, $\mathbf{e}_{i,\mu}$, is introduced in order to split the received vector into the node's signature and some error. (see equation (21)). Error propagation causes our original system model, equation (6)), to change. Now, the gaussian noise variance is no longer just N_o but has changed to $\sigma = \|\mathbf{e}_i\|^2 + N_o$. In general, the percentage of nodes that make a mistake in detection is about ϵ . Thus, the error variance is in the order of E_b times the percentage of nodes that make a mistake, $\|\mathbf{e}_i\|^2 = O(\|g_i\|^2\epsilon)$.

$$r_i = \mathbf{p}_m \otimes \mathbf{g}_i + \sum_{\mu \neq m} (\mathbf{p}_\mu - \mathbf{p}_m) \otimes \mathbf{e}_{i,\mu} + n_i \quad (21)$$

$$P(E) = 2Q \left(\sqrt{\frac{E_b}{\|\mathbf{e}_i\|^2 + \frac{N_o}{2}}} \right) \quad (22)$$

$$P_{UB}(E) = 2Q \left(\sqrt{\frac{E_b}{\frac{N_0}{2}} (1 - \sqrt{\epsilon})^2} \right) \quad (23)$$

$$(24)$$

In [1], the paper concludes that in the worst case scenario the error propagation leads to an SNR degradation of $(1 - \sqrt{\epsilon})^2$. Thus, an upper bound to the probability of error is obtain, equation (23). We proceeded to obtain a closer approximation. As before, we add the error variance to the noise variance. Also, BER is scaled by a factor of 2. Thus, we conclude that the best approximation to the effect of error propagation is equation (22). Figure [8] plots both the theoretical BER formulas and the simulated results with error propagation.

Figure 8: Theoretical Performance Plots

8 Conclusion

This paper outlines the concepts behind the OLA transmission protocol. Our Matlab simulation approach was discussed in detailed. The goal of this project was to provide experimental support to the theoretical concepts proposed in [1]. There is still further work to be done on this project. One aspect to explore is the performance of a non-regenerative transmission scheme. Also, we used BPSK modulation. It'd be interesting to explore the performance results using other modulation schemes.

A Program Listing

A.1 Network setup

Node setup

```
%-- node_setup.m
%-- Network Configuration
N=60;
radius=10;

U = rand(1,N);
theta = 2*pi*rand(1,N);
% randomly place the nodes in the circle
nodes = radius.*sqrt(U).*exp(j*theta);
x = [real(nodes)].';
y = [imag(nodes)].';

%   figure(1);
%   plot(nodes,'p'); hold on;
%   for i=1:N, text(real(nodes(i)), imag(nodes(i)), int2str(i)); end

%---Relative distance matrix
d=sqrt((x*ones(1,N)-ones(N,1)*x').^2+(y*ones(1,N)-ones(N,1)*y').^2);
[app, ind]=sort(d(:,1));
x=x(ind); y=y(ind); %--> points are sorted with ascending distance from the leader
distmatrix=sqrt((x*ones(1,N)-ones(N,1)*x').^2+(y*ones(1,N)-ones(N,1)*y').^2);

%-- To calculate the path loss and Rayleigh Coefficients {A}ij
a=2; %--> because we are using amplitude, not power or energy
alpha=sqrt(0.5)*(randn(N)+j*randn(N))./(1+distmatrix).^a;
alpha=(alpha-tril(alpha))+(alpha-tril(alpha)).'; % ----> reciprocity

save OLANode60.mat;
```

A.2 MSE files

Find Training Length

```

% findtrainLen.m
% Goal: For a specific network, find the appropriate training Length in order
% to achieve the same performance as when MSE is zero.

%close all; clear all; clc;
% create network
N=10; radius=3;
%node_setup;
load OLANode10;

%find out the appropriate training lengths for different snrs
trainingLen=1e4;
ThsdB=[3];
Ths =10.^(ThsdB./10);
for Thloop = 1:length(Ths)
    Th = Ths(Thloop);
    OLAtraining;          % ideal training
    for trainloop=1:length(trainingLen)
        trainLen=trainingLen(trainloop);
        BER_orig = .5*erfc(sqrt(2*Th)/sqrt(2));
        figure(1); semilogy(ThsdB(Thloop), BER_orig,'ro'); hold on;
        MSEg = (mean(tau_f_ideal)*.5)./(trainLen);
        newsnr=(1./(MSEg+1/2)).*Th;
        BER_mse_ideal = .5*erfc(sqrt(newsnr)/sqrt(2));
        semilogy(ThsdB(Thloop), BER_mse_ideal,'*'); text(ThsdB(Thloop), ...
            BER_mse_ideal, int2str(trainLen));

        grid on;
    end
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%results : trainLen = [1e4 1e4 1e4 1e4 1e4] for corresponding ThdbB=[0:4]

```

MSE training

```

%-- MSEtraining.m
%-- Goal: Perform the Estimate Pure Signal Through Training Approach.

close all;
%----- Training Phase -----
No=1;

```

```

%trainLen = 100;    %-- an external input
MSE_g = zeros(N,1);
tm = ones(1,length(trainLen));
g =zeros(N,Ts);          %-- the received vector space
%-- for the first training symbol sent, get the firing times (i.e. tau_f)
k=1;
s=zeros(N,Ts);  f=s;
s(1,:)=sqrt(pulsepower*W).*sinc(W*t/pi);
g(1,:)=g(1,:)+s(1,:);
tau_p=distmatrix/c/dt;    %---> propagation times
tau_f=[0,inf*ones(1,N-1)]'; %---> firing times (initially infinite)
for n=2:N
    s(n,:)=s(n,:) + alpha(n,1)*sqrt(pulsepower*W)*    ...
        sinc(W*(t-tau_p(n,1)-tau_f(1))/pi); %-> add to all leader signal
end

%-- calculate firing time
fired=[1];
prev_sum=0;
while prev_sum <sum(fired) %--> if the number of nodes that fired changed
    Es=cumsum(abs(s).^2,2); %---> energy over time X node
    Es(fired,:)=zeros(length(fired),Ts);
    [time, node]=find(Es.'>Th); %--> find times when Es exceeds
    %-->the threshold for all nodes
    prev_sum=sum(fired);
    if ~isempty(node)
        ind1=[1; find((node(1:length(node)-1)-node(2:length(node))))<0)+1];
        time=time(ind1); node=node(ind1);
        [val p]=min(time);
        firing=node(p);
        tau_f(firing)= time(p); %--> a new node firing
        fired=[fired, firing];
        %-- update
        g(firing,1:tau_f(firing)) = g(firing,1:tau_f(firing)) + ...
            s(firing, 1:tau_f(firing)) + ...
            sqrt(No/2).*(randn(1,tau_f(firing))+j*randn(1,tau_f(firing)));
    end
    %-- compose the signal with the new pulse added to all other nodes
    for k=1:N

```

```

                s(k,:)=s(k,:)+alpha(k,firing)*sqrt(pulsepower*W)* ...
                    sinc(W*(t-tau_p(firing,k)-tau_f(firing))/pi);
            end
        end %if
    end %while
    %-- for the rest of the training symbols sent, tau_f is the same
    h = waitbar(0,'Training period');
    for k = 2:trainLen
        waitbar(k/trainLen,h)
        s=zeros(N,Ts);
        s(1,:)=sqrt(pulsepower*W).*sinc(W*t/pi);
        g(1,:)=g(1,:)+s(1,:);
        for n=2:N
            s(n,:)= s(n,:) + alpha(n,1)*sqrt(pulsepower*W)* ...
                sinc(W*(t-tau_p(n,1)-tau_f(1))/pi); %--> add to all leader signal
        end

        %-- calculate firing time
        for p=2:length(fired)
            firing= fired(p);
            % update
            g(firing,1:tau_f(firing)) = g(firing,1:tau_f(firing)) + ...
                s(firing, 1:tau_f(firing)) + ...
                sqrt(No/2).*(randn(1,tau_f(firing))+j*randn(1,tau_f(firing)));
            %-- compose the signal with the new pulse added to all other nodes
            for k=1:N
                s(k,:)=s(k,:)+alpha(k,firing)*sqrt(pulsepower*W)* ...
                    sinc(W*(t-tau_p(firing,k)-tau_f(firing))/pi);
            end
        end
    end %training for loop
    close(h);
    g=g./trainLen;
    MSE_g= MSE_g + sum((abs(g-g_ideal).^2),2); %--> calculate MSE

```

MSE Simulation

```

%-- MSEsimulate.m
%-- Goal: Find the MSE for a specific training length and different SNR.

```

```

close all; clear all; clc;
%-- Network setup
N=10; radius=3;
% node_setup;
load OLANode10;

%-- SNR specifications
ThsdB = [0:3];
Ths=10.^(ThsdB/10);

%-- Use findtrainLen.m to get appropriate training lengths
%-- that will match the original performance
trainingLen = [1e4 1e4 1e4 1e4];
tauf_ideal = zeros(N,length(Ths));
MSEg = zeros(N,length(Ths));
MSEtau_f = zeros(N, length(Ths));
for Thloop = 1:length(Ths)
    Th = Ths(Thloop);
    OLAttraining;
    tauf_ideal(:,Thloop) = tau_f_ideal;
    trainLen=trainLen(Thloop);
    MSEtraining;
    MSEg(:,Thloop) = MSE_g;
end

save MSEtest0_3.mat MSEg trainingLen N radius ThsdB tauf_ideal;

```

Plot MSE

```

%-- plot_MSE.m
%-- Goal: Produce plots for the MSE simulation

clear all; close all; clc;
% % Combine all test files into one
% load 'MSEtest0_3.mat';
% trainLen=trainLen; snrdB=ThsdB;
% MSEg_avg=sum(MSEg(2:end,:))./(N-1);
% MSEtau_avg=mean(tauf_ideal(1:end,:));
%
% load 'MSEtest4.mat';

```

```

% trainLen=[trainLen trainingLen]; snrdB=[snrdB ThsdB];
% MSEg_avg=[MSEg_avg (sum(MSEg(2:end,:))./(N-1))];
% MSETau_avg=[MSETau_avg mean(tauf_ideal(1:end,:))];
%
% save mse.mat trainLen MSEg_avg MSETau_avg snrdB N radius;

% plot simulations
load 'mse.mat';
figure; plot(snrdB, MSEg_avg,'*-');
for i=1:length(snrdB)
    text(snrdB(i), MSEg_avg(i), int2str(trainLen(i)));
end
title(sprintf('MSE of g vs SNR for network %d nodes %d radius',N,radius));
print -deps MSE.eps

snr = (10.^(snrdB./10));
BER_orig = .5*erfc(sqrt(2*snr)/sqrt(2));
figure; semilogy(snrdB, BER_orig,'r-o'); hold on;

MSE = (MSETau_avg*.5)./trainLen;
newsnr=(1./(MSE+1/2)).*snr;
BER_mse_ideal = .5*erfc(sqrt(newsnr)/sqrt(2));
semilogy(snrdB, BER_mse_ideal,'d');

MSE = MSEg_avg;
newsnr=(1./(MSE+1/2)).*snr;
BER_mse_est=.5*erfc(sqrt(newsnr)./sqrt(2));
semilogy(snrdB, BER_mse_est,'g*');
legend('original','with theoretical MSE','with simulated MSE')
title(sprintf('Effect of MSE on Performance'));
grid on;
hold off;

print -deps MSEBER.eps

```

A.3 Error Propagation files

Training

```

%-- OLAttraining.m

```

```

% Goal: to estimate firing times for each nodes by flooding the network,
% it is assumed in this simulation that each node needs to exceed
% a certain SNR threshold in order to be declared as reachable and
% to be able to retransmit.

% close all; clear all; clc;
showplots=0;

%---Relative delay
c=3e8;
BW=83.5*10^6;
dt=1/BW/10;    % oversampling-- dt units
tau_p=distmatrix/c/dt;

%=====
%---generate signals
W = BW*dt;
pulsepower=100;          %mW energy of one sinc pulse
Ts=1500;  t=(-50:Ts-51);    %kTc
% pulse=sqrt(pulsepower*W)*sinc(W*t);
% figure;
% plot(t,pulse);
% xlabel('time (T_p/10 usec)');
% ylabel('normalized amplitude');

s=zeros(N,Ts);  f=s; g=zeros(N,Ts);
s(1,:)=sqrt(pulsepower*W).*sinc(W*t/pi);
g(1,:)=s(1,:);
tau_p=distmatrix/c/dt;    %---> propagation times
tau_f=[0,inf*ones(1,N-1)]'; %---> firing times (initially infinite)
%----- ideal training-----
% Th = 1; % external input
% leader fires
for n=2:N
    s(n,:)= s(n,:)+alpha(n,1)*sqrt(pulsepower*W)*
        sinc(W*(t-tau_p(n,1)-tau_f(1))/pi); %-> add to all leader signal
    if (showplots), figure(n), plot(abs(s(n,:))), pause, end
end

%-- calculate firing time

```

```

fired=[1];
prev_sum=0;
while prev_sum <sum(fired) %--> if the number of nodes that fired changed
    Es=cumsum(abs(s).^2,2); %---> energy over time X node
    Es(fired,:)=zeros(length(fired),Ts);
    [time, node]=find(Es.'>Th); %--> find times when Es it
                                %--> exceeds the threshold for all nodes
    prev_sum=sum(fired);
    if ~isempty(node)
        ind1=[1; find((node(1:length(node)-1)-node(2:length(node)))<0)+1];
        time=time(ind1); node=node(ind1);
        [val p]=min(time);
        firing=node(p);
        tau_f(firing)=time(p); %--> a new node firing
        %
        if(flag1)
            %
            f(firing,tau_f(firing))=abs(s(firing,tau_f(firing)));
            %
            figure(firing), plot(abs(s(firing,:)),'r'), hold on,
            %
            plot(f(firing,:),'g') , hold off
            %
            title(['node',num2str(firing)]), pause
            %
        end
        fired=[fired, firing];
        g(firing,1:tau_f(firing)) = s(firing, 1:tau_f(firing));
        %-- compose the signal with the new pulse added to all other nodes
        for k=1:N
            s(k,:) = s(k,.)+alpha(k,firing)*sqrt(pulsepower*W)* ...
                sinc(W*(t-tau_p(firing,k)-tau_f(firing))/pi);
        end
    end
end
g_ideal = g;
tau_f_ideal= tau_f;
Eg=real(diag(g_ideal*g_ideal'));

%save training.mat

```

Transmission and Detection

```

% OLATransmission.m

% close all; clear all;

```

```

% load OLANode60;      % first, network needs to be setup
% load training;      % second, training needs to be done

%%-----Send DATA
No=1;
M=2; alphabet = [-1; 1];
% dataLen=500;      % external input
% erro_prop=1;      % external input
data = alphabet(ceil(M*rand(1,dataLen)));
ecount = zeros(N,1);          % keep a count of error
h = waitbar(0,'Sending Data');
for m = 1:dataLen
    waitbar(m/dataLen,h);
    symbol = zeros(1,N);      % symbol detected by transmitting node
    symbol(1) = data(m);

    s=zeros(N,Ts);  r=zeros(N,Ts);
    s(1,:)=data(m)*sqrt(pulsepower*W).*sinc(W*t/pi);
    for n=2:N
        s(n,:)= s(n,:) + data(m)*alpha(n,1)*sqrt(pulsepower*W)* ...
            sinc(W*(t-tau_p(n,1)-tau_f(1))/pi); %-> add to all leader signal
    end
    for p=2:length(fired)
        r(fired(p),1:tau_f(fired(p))) = s(fired(p),1:tau_f(fired(p))) + ...
            sqrt(No/2).*(randn(1,length(1:tau_f(fired(p))))+ ...
                j*randn(1,length(1:tau_f(fired(p)))));

        %make detection
        for n = 1:M
            distance_sq(n) = norm(r(fired(p),1:tau_f(fired(p))) - ...
                alphabet(n).*g_ideal(fired(p),1:tau_f(fired(p))))^2;
        end
        [val ind] = min(distance_sq);
        symbol(fired(p)) = alphabet(ind); % make detection for node fired(p)
        ecount(fired(p)) = ecount(fired(p)) + (symbol(fired(p))~= data(m));
    %end detection
    %-- compose the signal with the new pulse added to all other nodes
    for k=1:N
        if (error_prop)
            s(k,:) = s(k,.)+ symbol(fired(p)).*alpha(k,fired(p))* ...
                sqrt(pulsepower*W)*sinc(W*(t-tau_p(fired(p),k)-tau_f(fired(p)))/pi);
        end
    end
end

```

```

        else
            s(k,:)=s(k,:)+ data(m).*alpha(k,fired(p))*sqrt(pulsepower*W)* ...
                sinc(W*(t-tau_p(fired(p),k)-tau_f(fired(p)))/pi);
        end
    end
end
end
close(h);

sigma = sqrt(No/2);
c_nodes = find(Eg > 0);
num_conn = length(c_nodes);
pe = ecount(c_nodes)./dataLen;
pe_avg = mean(pe);
std_ber = std(pe,1);
% figure;
% semilogy(c_nodes,pe,'o');
PE = zeros(1,length(c_nodes));
for i = 1:length(c_nodes)
    dhat = 2*norm(g_ideal(c_nodes(i),1:tau_f(c_nodes(i)))));
    PE(i) = 0.5.*(erfc(dhat/(2.*sigma.*sqrt(2))));
end
PE_ideal = mean(PE);
% hold on;
% semilogy(c_nodes, PE,'x');

```

OLA Simulation

```

%-- OLAsimulate.m
close all;
clear all;

Ths_db = [0:7];          % SNR
Ths = 10.^(Ths_db./10);
BERavg = zeros(length(Ths),1);
BER_ideal = zeros(length(Ths),1);
BER_std = zeros(length(Ths),1);
conn = zeros(length(Ths),1);
h3 = waitbar(0,'Simulation running');

```

```

load OLANode60a;
for runs = 1:length(Ths)
    %-- inputs to OLAtaining
    Th = Ths(runs);
    pulsepower=100;    % for snr values 0-4dB
    % pulsepower=200; % for snr values 5-7dB
    OLAtaining;
    %-- inputs to OLATransmission
    dataLen=500;      % for snr values 0-2dB
    % dataLen=1000;   % for snr values 3-5dB
    % dataLen=2000;   % for snr values 6B
    % dataLen=3000;   % for snr values 7dB

    error_prop = 1;
%   error_prop = 0;    % for snr values 0-5dB, datalen = 300; pulsepower=100;

    OlaTransmission;
    BERavg(runs) = pe_avg;
    BER_ideal(runs) = PE_ideal;
    BER_std(runs) = std_ber;
    conn(runs) = num_conn;
    waitbar(runs/length(Ths),h3);
end
close(h3);

save errorprop02p.mat Ths_db BERavg BER_ideal BER_std conn
% save errorprop34p.mat Ths_db BERavg BER_ideal BER_std conn
% save errorprop5pa.mat Ths_db BERavg BER_ideal BER_std conn
% save errorprop6pa.mat Ths_db BERavg BER_ideal BER_std conn
% save errorprop7pa.mat Ths_db BERavg BER_ideal BER_std conn
% save noerrorprop.mat Ths_db BERavg BER_ideal BER_std conn

```

OLA Performance plots

```

%-- plot_ber60.m
%-- generates BER07.eps
close all;
clear all;
%%--- Combine all the simulations for different snr into one file: ber.mat

```

```

% load errorprop02p.mat %0:2
% x = [0:1:7];
% Ths = 10.^(x./10);
% y = BERavg;
% num_conn = conn;
% std = BER_std;
% y_ideal = BER_ideal;
%
% load errorprop34p.mat %3:4
% y = [y; BERavg];
% num_conn = [num_conn; conn];
% std = [std; BER_std];
% y_ideal = [y_ideal; BER_ideal];
%
% load errorprop5pa.mat %5
% y(6) = BERavg;
% num_conn(6) = conn;
% std(6) = BER_std;
% y_ideal(6) = BER_ideal;
%
% load noerrorprop.mat
% y_noerror = BERavg;
% std_noerror = BER_std;
%
% load errorprop6pa.mat %6
% Ths = 10.^(x./10);
% y = [y; BERavg];
% num_conn = [num_conn; conn];
% std = [std; BER_std];
% y_ideal = [y_ideal; BER_ideal];
%
% load errorprop7pa.mat %7
% y = [y; BERavg];
% num_conn = [num_conn; conn];
% std = [std; BER_std];
% y_ideal = [y_ideal; BER_ideal];
%
% % to calculate std deviations
% y_high = y + std;
% y_low = y - std;

```

```

%
% y_noerror_high = y_noerror + std_noerror;
% y_noerror_low = y_noerror- std_noerror;
%
% points=[1:8];
% x=x(points);
%
% save ber.mat x y num_conn std y_ideal;

% Plot simulation results w/ error propagation
load 'ber.mat';
figure(1); semilogy(x, y, 'x--k');

% %=====
% % Plot theoretical computations
% % simulations results for no error propagation
% hold on;
% semilogy(x, y_ideal, 'o-r');
% semilogy(x(1:5), y_noerror(1:5), '*-b');
% for i = 1:length(x)
%     semilogy(x(i), linspace(y_high(i), y_low(i), 100), 'r-');
% end
% hold off;
% %=====
% Plot theoretical Probability of error no error propagation
Ths = 10.^([0:7]./10);
BER_formula = .5*erfc(sqrt(2.*Ths)./sqrt(2));
hold on; semilogy(x, BER_formula, 'o-k');

%-- approximations
% Plot approximation of P(E) taking error propagation into effect
error_variance = BER_formula.*Ths;
new_SNR = Ths./(error_variance + 1/2);
BER_new = erfc(sqrt(new_SNR)./sqrt(2));
figure(1); hold on;
semilogy(x, BER_new, 'd-r');

% Plot approximation of P(E) for worst case scenario
factor = (1-sqrt(BER_formula)).^2;

```

```

new_SNR = factor.*Ths./(1/2);
BER_new = erfc(sqrt(new_SNR)./sqrt(2));
figure(1); hold on;
semilogy(x, BER_new, 'd-g');

grid on;
legend('With Error Propagation', 'With No Error Propagation', ...
      'Approximation as noise', 'SNR degrades (1-sqrt(epsilon))');
xlabel('SNR Threshold'); ylabel('Probability of Error');
axis([-0.05 7.05 10^-4 10^0]);
%print -deps BER07new.eps

% figure(2)
% plot(x, num_conn(points), '*-');
% grid on;
% xlabel('SNR Threshold'); ylabel('Number of Connected Nodes');

```

References

- [1] A. Scaglione, Y.W. Hong. "Opportunistic Large Arrays Formed with Wireless Ad Hoc Networks", School of Electrical and Computer Engineering, Cornell University, 2002.
- [2] Bernard Sklar. Digital Communications, 2nd ed. Prentice Hall,2001.