The Computer Journal, 51(4):511-522, 2008

Computing with Time: From Neural Networks to Sensor Networks

Short Title: Computing with Time

Boleslaw K. Szymanski and Gilbert G. Chen
Department of Computer Science, Rensselaer Polytechnic Institute
110 Eighth Street, Troy, NY 12180, USA

Corresponding Author: Boleslaw K. Szymanski,
Lally 204, RPI, 110 Eighth Street, Troy, NY 12180, USA
Email: szymab@rpi.edu, tel. +1-518-276-2714, fax: +1-518-276-4033

Abstract

This article advocates a new computing paradigm, called *computing with time*, that is capable of efficiently performing a certain class of computation, namely, searching in parallel for the closest value to the given parameter. It shares some features with the idea of *computing with action potentials* proposed by Hopfield, which originated in the field of artificial neuron networks. The basic idea of computing with time is captured in a novel distributed algorithm based on broadcast communication called the *Lecture Hall Algorithm*, which can compute the minimum among *n* positive numbers, each residing on a separate processor, using only *O(1)* broadcasts. When applied to sensor networks, the Lecture Hall Algorithm leads to an interesting routing protocol having several desirable properties.

Introduction

Although neural network and sensor network are normally viewed as two radically different subjects, they do share one thing in common. The most fundamental way of exchanging information in both kinds of networks is one-to-many communication, i.e., the broadcast. In a biological neural network, a firing neuron sends an action potential to all neurons that are connected to it by synapses, each of which may impose different delay and amplification to the transmitted signal. Similarly, a communication node in a sensor network broadcasts its signal to all nodes within its transmission range. The proposed computing with time paradigm applies to networks in which a broadcast is a communication primitive, such as neural networks in biology or wireless networks in telecommunication.

Another example of such a paradigm is computing with action potentials proposed by Hopfield et al. [1], who observed that analog information can be encoded into firing times of action potentials and that the timings of these action potentials can be used to carry out a vector matching algorithm. The ability to perform broadcast-based communication was not explicitly mentioned as a requirement. Yet, to harness the information carried by the timings of action potentials, synchronization is absolutely necessary, even though it does not have to be very accurate. There must be certain moments at which distributed neurons observe the same events, as if each of them would own a local clock and these clocks were synchronized from time to time by such events. Broadcast naturally provides plenty of such synchronization points.

The basic form of computing with action potentials converts one vector into time intervals between the firings of one set of neurons and the pre-specified global synchronization instance, and another vector into time delays of the synapses connecting those neurons to the set of output neurons. If these two vectors match, the output neurons will all fire at the same time, resulting in their strongest superimposed output.

A different way to make use of timing information, proposed here, is to use the firing times of different neurons to identify one with the smallest firing time. The purpose is to look for an optimum value by associating the firing times with a certain variable in such a way that the smaller this variable is the more desirable the property of the corresponding neuron is. Hence, the neuron firing earliest will naturally be the one whose property variable has the minimum value among the neurons being compared. Hence, the essence here is to introduce competition, instead of superposition in Hopfield's approach, to select a winner that possesses the desired optimality.

This new way of using timing, referred to as *computing with time*, is applicable to any systems that support broadcast as an inherent communication mechanism. We applied computing with time to algorithm design in sensor networks, and the result is an interesting routing algorithm, named Self-Selective Routing. The development of this new routing algorithm illustrates that an idea formalized in neural network may find its application in sensor network because of the similarity in their underlying communication mechanisms, and provides a new understanding of the role of time, as well as broadcast, as a basic means of computation in networks with broadcast as a communication primitive.

Computing with Action Potentials

In a series of papers [1-4], Hopfield and his collaborators suggested a new computing paradigm that uses the exact timings of action potentials to perform useful computation in an efficient way. Unlike traditional computational procedures that are mathematically or algorithmically too complicated to be directly implemented on neurons, the representation and manipulation of information required by the new approach of *computing with action potentials* appear to fit the neuron paradigm rather well.

The key idea of computing with action potentials is captured by an elegant solution to the so-called *analog match* problem which is to match an input vector against a number of known patterns also given in the form of vectors. Let $X^{(0)} = [x_1^{(0)}, x_2^{(0)}, ..., x_n^{(0)}]^T$ be the input vector, and $\{X^{(1)}, X^{(2)}, ..., X^{(k)}\}$ be the given patterns where *for i=1,2,…n*, $X^{(i)} = [x_1^{(i)}, x_2^{(i)}, ..., x_n^{(i)}]^T$. The analog match problem is to find, if one exists, a vector $X^{(i)}$ among $\{X^{(1)}, X^{(2)}, ..., X^{(k)}\}$ such that $X^{(i)}$ is similar to $X^{(0)}$, meaning that there exists a constant $\lambda \approx 1$ such that $x_m^{(i)} \approx \lambda x_m^{(0)}$ for all $1 \le m \le n$.

The conventional solution to the analog match problem is to first normalize all vectors and then to select a vector $X^{(i)}$ in the given patterns such that its inner product with the input vector $X^{(0)}$ is greater than a threshold constant c:

$$1 \geq \frac{X^{(0)} \cdot X^{(i)}}{\left\|X^{(0)}\right\|\left\|X^{(i)}\right\|} > c, \text{where } X^{(0)} \cdot X^{(i)} = \sum_{l=1}^{k} x_l^{(0)} x_l^{(i)}.$$

Normalization is necessary since otherwise pattern vectors with large values would be heavily favored. However, Hopfield et al. argued that normalization is non-trivial for realistic neural networks. Among other problems, this method does not differentiate between individual components according to their importance, making it difficult to adjust the reliance of the result on a particular component.

This weaknesses can be avoided if vectors are represented by the timings of action potential emitted by neurons [1]. The individual components of the input vector are at first encoded in the initial firing time of corresponding action potentials in advance of what Hopfield called a fiducial time T. The smaller the value is the closer the action potential initial firing time is to the fiducial time T, as shown in Figure 1. Then, each action potential is delayed by a period that is determined by the corresponding component of the vector to be matched, $X^{(i)}$. If the two vectors are similar, then all the action potentials will fire at roughly the same time. Thus, a match is found if the maximum of the weighted sum of action potential exceeds a certain threshold. Normalization is no longer needed because the match is defined by synchronization of the firing times defined by the logarithm of the ratios of the corresponding components of the two vectors, thereby eliminating any constant factor considerations. Moreover, the importance of each component can be now reflected in the weights used to sum up action potentials together.

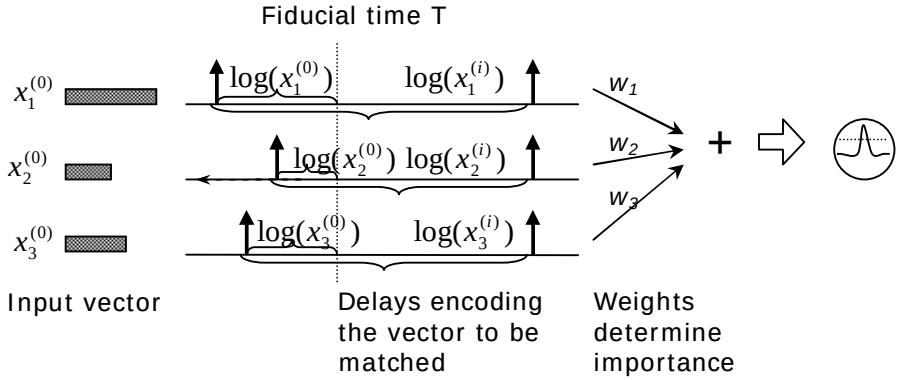


Figure 1. Using timings of action potentials to solve the analog match problem.

Obviously, for the idea described thus far to work, the existence of the fiducial time is indispensable, as it provides a reference point with which the timing information of action potentials can be combined to produce the desired results. In other words, a synchronization point is needed at which all neurons observe the same moment and it can be provided as a natural byproduct of a broadcast.

The idea of computing with action potential has two important implications. First, it provides a solid support for the suggestion that neurons may communicate and compute by temporal coding instead of rate coding. There are several spiking neuron models [5-6] that emphasize asynchronous inter-neuron communication. However, Hopfield's idea is a unique attempt, motivated by a significant body of biological evidence that rate coding is too slow for quick responses in the order of 10ms observed in nature [7-10], to

devise a convincible mechanism by which practical computation can be performed efficiently with respect to the communication cost.

The other implication of Hopfield's idea is far beyond the field of biological neural networks. It shows that time, when appropriately encoded, can be utilized to enable a new generation of distributed algorithms that take advantage of broadcast to minimize the communication cost and do not follow the step-by-step execution paradigm. Such algorithms may be hard to analyze by classical time complexity theory due to their asynchronous nature, but they still fall within the domain of Turing machines in the sense that they can be simulated by a Turing machine.

Lecture Hall Algorithm

Let us start with a simple yet illustrative example to demonstrate the power of computing with time. Assume there are *n* students sitting in a lecture hall and listening to a lecturer. For some reason, the lecturer wants to find the youngest student in the room (assuming that there is only one such student). The well-known method would divide students into pairs. The younger student of a pair wins, and all winners start the second round. The same procedure repeats until only one student is left who must be the youngest. If we assume that a pair of two students can exchange information about their ages without interfering with other students, then the time complexity of this recursive procedure is $O(\log(n))$. The communication complexity is $O(n)$, since $n-1$ comparisons are necessary. It seems that this is the best that one can achieve.

Surprisingly, we devised a much more efficient procedure, referred to as the *Lecture Hall Algorithm*, which is inspired by Hopfield's discovery that continuous variables can be converted into time delays to facilitate computation. This procedure works as follows. First, the lecturer, who can be heard by every student in the room, announces a challenge. Upon hearing it, every student immediately calculates a delay of his or her response that is equal to his or her age. As soon as the delay elapses, the student announces it. However, if a student ever hears another student speaking before his delay elapses, his scheduled action is immediately cancelled.

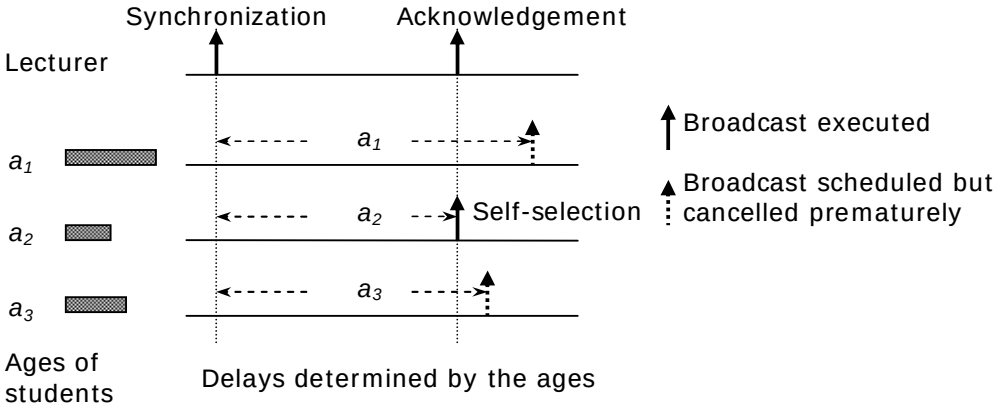


Figure 2. The Lecture Hall Algorithm.

If the transmission delay of announcements and the reaction time of students are equal or negligibly different, the first student that speaks up is guaranteed to be the youngest in the class. We call the action taken by this student *self-selection*, because each student

can determine, without consulting others that the speaker must be the youngest by the simple fact that no other student spoke earlier.

It may happen that the self-selection announcement is not heard by all students. For example, the answering student may be sitting in one corner of the room and thus may not be heard by the students in the other corner. To solve this minor problem, the lecturer can make yet another announcement, called the acknowledgement announcement, once the proper student has been identified. This announcement is to tell all students that the Lecture Hall Algorithm has been completed with a successful outcome, and those who have not heard the self-selection announcement should cancel their scheduled action anyway. The information about the age of the self-selected student can be included in the acknowledgement announcement. Interestingly, acknowledgement broadcast is not necessary for correctness of the algorithm. It simply suppresses activation of multiple self-selected nodes. In some applications, including the one discussed below, sensor network routing, under many scenarios it is more efficient to bypass acknowledgement broadcasts entirely.

Therefore, only two broadcasts, one for a challenge announcement and the other for self-selection announcement, are strictly necessary in the Lecture Hall Algorithm. Optionally, the third, acknowledgement, broadcast may also be used to guarantee unique self-selection. Moreover, in some network applications, such as routing, the self-selection is repeatedly applied across the network. In such applications, the self-selection broadcast of one Lecture Hall Algorithm invocation can be used as the challenge announcement for the subsequent algorithm invocation, cutting the number of broadcasts needed by each call to exactly 1.

What is the Lecture Hall Algorithm time complexity? With the assumption that any transmission or reaction delay can be ignored, the time it takes is determined by the range of the students' ages and the conversion function used to convert the ages into delays. It is completely independent of the number of students involved.

The Lecture Hall Algorithm can easily be modified to perform search. The searched value is broadcast together with precision of the answer permitted. Then, each node receiving the broadcast and holding the value of this type computes its delay based on the difference between the search value and the stored one (assuming that the difference is smaller than the desired precision). In such an application, the time to obtain the answer is proportional just to the desired precision of the answer and is independent of the number of nodes participating or the range of values stored in them.

The Lecture Hall Algorithm can easily be generalized beyond finding the minimum. By taking an inverse of the value (age in this case), it can find the maximum value. By ignoring k^{th} first answers, it finds $k+1^{st}$ smallest value in the set, etc. Another example is the parallel sort whose implementation based on the Lecture Hall Algorithm requires that each processors reveals its value when its timer for self-selection expires (a care has to be taken to add to the timer each time interval during which others broadcast their values). In this case, the time to the solution will be the maximum delay assigned to any value held by the nodes, as oppose to the minimum delay needed in the previous cases. However, in some cases, this maximum delay would be smaller than the time needed to execute $n\log(n)$ operations of the conventional sort algorithms.

Another application of the Lecture Hall Algorithm arose in simulating the growth of preferentially connected networks [11] in which newly added nodes connect to existing ones with the probability of connection defined by the out-degree of the target node. Simple modification of the Lecture Hall Algorithm uses single broadcast from the newly added node to start competition for the new connections and the neighbors respond with the random delay that is inversely proportional to their out-degree. The new node then connects to the first k responders, where k is the preset parameter of the network design.

The Lecture Hall Algorithm can easily be recast in terms of PRAM, in which processors use computer cycles to measure a delay. Hence, time is discretized with a time tick equal to a computer cycle. However, algorithms sketched above for finding the minimum, the maximum, or the closest value to a given parameter, as well as sorting translate directly, especially if PRAM with some form of parallel write is used so an accidental overlap of different answers is not a problem. Compared to the traditional binary tree parallelization of these algorithms, they save both memory and time.

Another interesting property of the Lecture Hall Algorithm is that since waiting for the timer to go off is a passive activity, during such wait, a node may participate in many different instantiations of the algorithm, so they are inherently parallelizable. For example, the synchronization announcement of the lecturer may include finding a student with the smallest age, another who was born closest to Katmandu, and yet another with the highest GPA in high school, all over the same time period. The answers by self-selected students would identify which search is completed (with some provisions for interference created by accidental overlap of different answers).

Analysis of the Lecture Hall Algorithm

Usually, there is some small time, denoted here as s, that expires when a node switches from listening to other's reaction to starting its own response. Although this time is usually very short, the existence of such blind period causes the algorithm not to produce a unique outcome every time. Two students whose ages are fairly close to each other may start responding unaware of each other activity, resulting in a collision of their responses. Below, we establish the probability of selecting one and only one student by the Lecture Hall Algorithm.

In the following analysis, let n denote the number of students involved in the self-selection and r the average value of the time delay measured in units of s, which is the minimum time difference between the expiration of timers in two nodes needed to avoid a collision of their responses. All time delays in the following will be expressed in units of s. Any monotonically non-decreasing function can be used to map the value v held by a node into its timer delay t.

If the youngest student speaks up at time t, then another student will not hear it and may react to the lecturer announcement as late as $t+1$. Hence, whether a collision will occur or not depends on the difference between the smallest delay and the second smallest delay. If this difference is larger than a time unit $s=1$, then there will be no collision.

We assume that each time delay is a non-negative, independent and identically distributed random variable defined by the cumulative probability distribution function,

F with the average value *r*. We also assume that each potential responder can hear any other responder (if this assumption does not hold, larger time unit needs to be selected, equal to the sum of the response and acknowledgement times, but the analysis below will remain essentially unchanged).

The probability of collision, P_c, under these assumptions is:

$$P_c = n\int_0^{\infty} dF(x)(n-1)\int_x^{x+1}(1-F(y))^{n-2}dF(y) = 1-n\int_0^{\infty}[1-F(x+1)]^{n-1}F'(x)dx.$$

With uniformly distributed delays over a range *[0,2r]*, it simplifies to

$$(1)\qquad P_c = 1-\frac{n}{2r}\int_0^{2r-1}\left[1-\frac{x+1}{2r}\right]^{n-1}dx = 1+\left[1-\frac{x+1}{2r}\right]^{n}\Bigg|_0^{2r-1} = 1-\left(1-\frac{1}{2r}\right)^{n} \approx \frac{n}{2r}.$$

The approximation holds if $r>>n$ which needs to be selected such anyway to keep the probability of collision low.

To reduce the probability of collision, we can increase *r*, since *s*, the time unit, is a constant determined by physical parameters of the nodes' radios. Such an increase however, impacts the expected delay of self-selection, T_d, which is:

$$T_d = \int_0^{\infty} n[1-F(x)]^{n-1}F'(x)xdx = \int_0^{\infty}[1-F(x)]^n dx.$$

With uniformly distributed time delays, over the range *[0,2r]*, it reduces to:

$$(2)\qquad T_d = \int_0^{2r}[1-\frac{x}{2r})]^n dx = -\frac{2r}{n+1}\left(1-\frac{x}{2r}\right)^{n+1}\Bigg|_0^{2r} = \frac{2r}{n+1}.$$

The maximum delay is naturally *2r*. Hence, there is a trade-off between the probability of selecting exactly one winner and the expected or maximum delay of such self-selection.

When the collision does occur, then multiple winners have self-selected themselves. If this is not acceptable, the self-selection needs to be repeated, until no collision occurs. We start with the important for applications case in which the values hold by the responder nodes are limited to the range $[0,v_{max}]$, and for average delay we also assume that their values are uniformly distributed over that range so $v_{ave}=v_{max}/2$. The function used for mapping values into delays is in this case is simply $t(v)=v*r*2/v_{max}$.

The repeated self-selections are designed slightly differently from the initial one. First, only responders to the previous self-selection participate in the next one. Second, from the timing of the first response received, the challenge broadcaster can compute the smallest value that the responders hold, v_i. Then, the responders recompute their delays as $t_i=(v-v_i)*(2r)^{i+1}/v_{max}$. It is easy to show by induction over the repeated self-selection round number that if the first recorded response started in the i-th selection at time t_i, then the responders values are between $v_i= t_i*v_{max}/(2r)^i$ and $v_i + v_{max}/(2r)^i$. The original broadcaster sends then repeat message that contain just this starting value v_i. We will denote the length of this repeat message broadcast as t_b. Unlike the initial broadcast that must include the details of the challenge, the repeated broadcast may include just unique identifier of a self-selection and its starting value v_i so it could be very short. If there is a collision of responses, another, *i+1* round of self-selection will start. With this

design, we will derive the upper bound for the average and the maximum time to identify the unique responder.

Two cases needs to be considered. The first case involves the continuous values of the random variable used for self-selection. In this case, it is easy to notice that the probability that the smallest and the next smallest value of the random variable at each node differ by a certain value *d* is given by:

$$P_c(d) = 1 - \left(1 - \frac{d}{2v_{ave}}\right)^n \approx \frac{nd}{2v_{ave}} \text{ for } d << 2v_{ave}/n.$$

It is easy to show by using Taylor series that if $r \geq 2.5n \geq 5$ then the probability of collision in i-th round of rebroadcast, denoted $P_c^{(i)}$, satisfies an inequality $P_c^{(i)} < \frac{P_c}{(0.9r)^i}$.

For a collision to occur, we also must have at least two responders participating, so $T_d < 2r/3$. The total average delay of selecting the unique winner, T_{ud}, can be then expressed as:

$$T_{ud} < T_d + \left(\frac{2r}{3} + t_b\right)\sum_{i=0}^{\infty}(i+1)P_c^{(i)} < 1.7r + 1.54t_b,$$

where $r \geq 2.5n \geq 5$. Since $r \approx 2.5n$ is in *O(n)*, so is the average delay of selecting the unique winner in this case.

The maximum delay for more general case of values limited to the range $[0,v_{max}]$ is defined as follows. In the last round of self-selection, denoted as *l*, we must have $d > v_{max}/(2r)^{l+1}$ because all values of the remaining responders from interval $[v_l, v_l + v_{max}/(2r)^l]$ are spread over the interval *[0,2r]*. Hence, the maximum delay, T_{maxd}, is:

$$T_{\max d} = (r + t_b)\lfloor \log_r(v_{ave}) - \log_r(d) \rfloor + r.$$

Hence, the maximum delay is linear in the number of responders and logarithmic in the range of value held by the nodes as well as in the difference between the smallest and next smallest value.

The second case that we need to consider involves nodes holding random values that are discrete and uniformly distributed over the range $[1, 2v_{ave}]$. Proceeding like in the previous case, at most $i = \left\lceil \frac{\ln(v_{ave})}{\ln(r)} \right\rceil$ rounds of self-selection will be needed to get all responders having the same value of the variable directing self-selection. The corresponding delay will be smaller then $2.03r + 1.54t_b$. Then, the second stage of the repeated self-selection starts in which each remaining responder generates a random uniformly distributed delay over the range *[0,2r]*. Hence, each subsequent round has the probability of collision defined by Equation (1).

To derive contribution of this second stage to the average delay, we observe that each round ending in collision has at least two responders and since the previous responders are the only eligible participants, the number of responders is at most *n* and is non-increasing over the rounds. Hence, the probability of collision is non-decreasing from round to round and the average delay of each round is bounded by the round with two responders, therefore it is *2r/3*. So, the average delay of randomized rounds, T_{rd}, is

$$T_{rd} < (1-P_c)[T_d + (\frac{2r}{3} + t_b)\sum_{i=1}^{\infty}(i+1)P_c^i] < \frac{0.67r + t_b P_c}{1-P_c} < 3.73r + 4.56t_b,$$

where the final approximation is valid under the assumption that $r \geq 2.5n$. Hence the total delay in this case is also modest $5.43r+6.1t_b$ and since r is in $O(n)$, so is the average delay.

Although working well in practice, the randomized second stage of self-selection cannot provide us with the bound for the maximum delay. To obtain such a bound, we need to use a different design that takes advantage of unique identifiers that all participating nodes possess. Let m denote the length of such identifiers in bits. Then, in m short rounds, the challenge broadcaster sends the request for response based on the $i=1,2,\ldots m$ bit of the responders' identifiers. If this bit is one in the responder identifier, the node immediately responds, if it is zero, it does not. Each round lasts at most $2t_b$ since the response does not carry any data. If the node does not respond but hears others responding, it drops out from future rounds; otherwise it stays. If the node responded, it is eligible for the next round. When the challenge broadcaster detects collision or silence in the round, it continues to the next round, otherwise, the unique winner has been just selected.

With the described above protocol, if the m-th round is reached then at most two nodes can be left active, as the surviving nodes must have the same m-1 bits of their identifiers. Clearly, in such a case these two nodes must differ in the last bit of their identifiers to have distinct from each other. Hence, the complexity of this stage is $2mt_b$, or $O(1)$ in the number of nodes participating (but logarithmic in the total number of nodes in the network, as they all have to possess unique identifiers). It is also $O(1)$ in the range of the values that they hold. The preceding stage is linear in the number of participating nodes and logarithmic in the average value of their holdings.

The final case that needs to be considered involves the applications in which there is no upper limit on the values held by the nodes participating in self-selection. In such a case, we can introduce the range seeking stage that will reduce this case to the ones discussed above in a finite number of steps. This stage is very simple, in each step i, we compute the delay of each node participating in self-selection as $t_{i+1}=\min(2r,\ln(t_i))$ and set t_0 equal to the value held by the node. This stage stops when there are responders (they may not be unique) with the delay smaller than $2r$. If there is more than one responder, all of them will continue to the subsequent stage in which we know the upper limit for the minimum value, v_{min} held by the nodes. Clearly, if this stage continues to step l, then $v_{min}>\exp(\exp(\ldots\exp(2r)\ldots)$ where exponential function is applied l times. Each step increases the range for the minimum value so much that for all practical purposes, a few steps will be sufficient.

In summary, we have demonstrated that the expected delay of self-selection of the unique node is finite and in practical applications small, linearly proportional to the number of nodes participating in the Lecture Hall Algorithm.

There is an important improvement to the Lecture Hall Algorithm in case when the properties for which the nodes self-select are monotonically non-increasing with time (for example as a result of a permanent failure of a node) and self-selection is made repeatedly. In such a case, once the node wins self-selection and does not experience a decrease in property subject to self-selection, it can respond immediately to the

subsequent self-selections, complete removing the delay of each self-selection. However, after the node's property subject to self-selection decreases, such a node should return to the normal rules of computing its delay. This version of the Lecture Hall Algorithm is at the same time efficient (no time delay to self-selection in a normal case) and robust (a selection of the best fit node in case of a change or a failure of the winner).

Self-Selective Routing

In a sensor network, typically there is no direct link between two nodes needing to exchange packets with one another, so packets from the source have to travel through multiple intermediate nodes, making hops, to reach the destination. Hence, a routing protocol has to be deployed to guide the packets from the source to the destination.

The routing problem can be reduced to the problem of finding the destination of the next hop at each routing step, which is very similar to the problem of finding the youngest student in a lecture hall. Naturally, the Lecture Hall Algorithm can be applied. The subsequent question is, what kind of node is the best candidate for relaying the packets?

One simple criterion is based on an observation that the neighbor with the smallest number of hops to the destination is the best candidates for relaying packet at each step as using it will minimize the number of routing steps. Under the assumption of symmetric links, if every packet keeps track of the number of intermediate nodes it has traveled, a node can estimate the distance to the destination by listening to the packets coming from the destination node.

Self-Selective Routing (SSR) [12] is a sensor network routing protocol that attempts to find the next node with the smallest number of hops to the destination using the Lecture Hall Algorithm. In the lecture hall example, given in the previous section, the ages of students are converted into time delays, whereas in Self-Selective Routing, the delays are defined by the number of hops to the destination.

Sensor networks are often deployed densely, so to avoid having too many nodes competing for self-selection, we introduced a version of Self-Selective Routing in which only nodes that are one hop closer to the destination than the sender are allowed to self-select. This version, called Self-Healing Routing (SHR), employs a scheme for route repair. In SHR, all nodes eligible to respond are the same distance to the destination, so randomness is used in computing the response delay to avoid multiple replies. The delay is selected using random variable uniformly distributed over the interval *[0,2r]*. Hence, the average delay of the self-selection is defined by Equation (2) and therefore the average delay at each hop is inversely proportional to the probability of collision of two self-selection responses. Using this relation allows us to keep the collision probability low with reasonable initial delay of selecting a path (the self-selections for subsequent packets sent to the same destination can use zero delay response by the currently self-selected node discussed above that incurs no self-selection delay in a reliable network, yet preserves the protocol ability to reroute in case of failures). Indeed, the typical value of the minimum separation time is *s=0.1* ms, so with modest value of r around *50* ms (500 time units), the probability of collision is below on percent even if there are several nodes capable of forwarding the packet. This is confirmed with simulation results discussed later in this section.

When a node wants to send a packet to the destination, it simply broadcasts the packet, without caring which node should be the next to pass the packet. This broadcast is a challenge broadcast that synchronizes the potential responders. The next node, self-selects itself, as described in the Lecture Hall Algorithm, and then rebroadcasts the packet. This rebroadcast is both the self-selection broadcast and the new challenge broadcast for transmitting over the next hop. Every node that initiates a synchronization broadcast must also keep listening on the channel. In one version of the protocol, suitable for networks with high density of the nodes, once the sending node detects that rebroadcast has occurred, it sends out an acknowledgement broadcast, in order to suppress unnecessary retransmissions. In addition, the sender in Self-Healing Routing listens to the response to its broadcast. If such a response does not arrive within *2r* interval (signaling the failure of the previously existing link), the node rebroadcasts the original packet. After the predefined number of unsuccessful rebroadcasts, the sender increases its distance to the destination by 2, starting a process of route repair. The change of distance to destination makes the sender ineligible for self-selection for carrying the current and subsequent packets to the same destination, if the nodes with shorter distance are present in its neighborhood.

Thanks to triple role of each broadcast: self-selection to the previous announcement, synchronization for the next selection and forwarding the packet, the protocol uses just one broadcast for each hop. Our experiments [13] with sensor networks showed that duplicate self-selections were rare for reliable communication and the average number of broadcast per hop in this setting was close to *1.1,* with *10%* of additional packets resulting from multiple self-selections and rebroadcast of packets lost to collisions caused by packets traversing some nearby paths.

Properties of Self-Selective Routing

Self-Selective Routing is a straightforward result of applying the Lecture Hall Algorithm to the routing problem. Interestingly, it possesses several important features that were not considered at all when the protocol was being developed by us.

First, Self-Selective Routing is resilient to node and link failures, because it does not attempt to maintain explicit paths and therefore there is no need to constantly monitor the connectivity of established paths. A failed node or a normal node with a failed link will not participate in the self-selection procedure anymore. Hence, unlike many traditional wireless routing protocols [14-16] that incur a significant amount of control overhead and delay to deal with failures, Self-Selective Routing can quickly establish new alternative paths (Figure 3). The resilience to failures also offers the freedom of turning any node off at any time to conserve energy, even if the node resides on an active path.

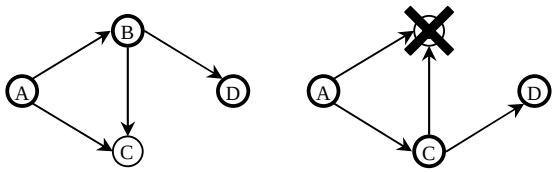


Figure 3. Packets immediately seek an alternative path in case of a node failure.

In addition, Self-Selective Routing can also automatically converge to shortest paths discovered through listening to passing by packets in two-way traffic. Figure 4 illustrates this capability. At the beginning (Figure 4(a)), packets from node A to node D are traveling through nodes B and C. Although there is a shorter path via node E, it is not recognized at this time because of either randomness or the recent arrival of node E in this neighborhood. If the communication continues to flow one way from A to D, then node E would never know its real distance to node D. Once node D transmits a packet to node A, node E will immediately realize that it is within one hop of node D. It will then win the competition against node C because its distance to node A is just one. The next time node A sends a packet to node D, node E will self-select itself for retransmission, which effectively shortens the path between nodes A and D by 1. The same principle applies to cases where the path length is greater than 2.

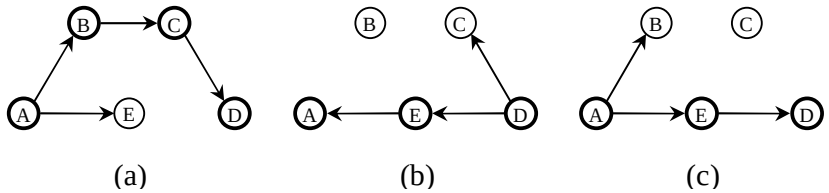
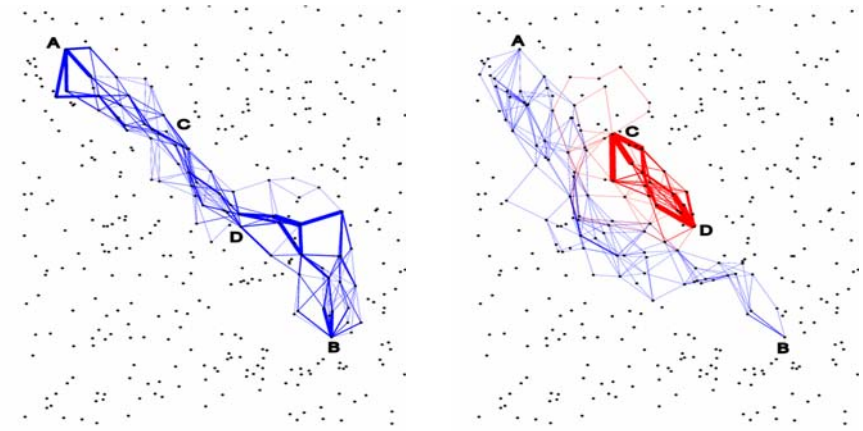


Figure 4. SSR is capable of constantly looking for and switching to shorter paths.

The self-adjustability to shortest paths comes from two factors, the dynamic nature of the protocol and no cost path switching. Because of the randomness introduced in the distance-to-delay conversion function, two consecutive packets traveling from the same source to the same destination may follow completely different paths. This permits nodes on these paths to continuously update their known distances to the source node, based on latest information. Furthermore, the freedom of switching paths without paying any price allows nodes to always select whichever path is shorter. These two factors are not available in many traditional routing protocols [14-16], where paths are relatively fixed so that nodes never get enough chances to evaluate alternative paths. Moreover, switching to shorter paths comes in those protocols at the cost of route maintenance overhead which may offset the benefit.

Finally, Self-Selective Routing can automatically avoid congestion. In high density regions, packets may have to wait for a long time in the transmission queue before being transmitted. When a node with a long transmission computes a short time delay, it will likely not be able to self-select itself as quickly as nodes in less congested areas.



(a) (b)

Figure 5. (a) Normal traffic between nodes A and B. (b) Traffic between nodes A and B avoiding congestion due to the newly introduced traffic between nodes C and D.

Figure 5 illustrates the actual paths taken by different packets in two simulation runs. Figure 5(a) depicts the case with one communicating pair sending packets from node A to node B. Figure 5(b) shows the same network with an additional communicating pair generating traffic from node C to node D. As shown in this figure, with proper selection of conversion function, Self-Selective Routing is capable of forwarding packets around the congested area caused by the intensive traffic between C and D. In spite of the increased path length between A and B, traveling around the congested area may still be faster than traveling through it.

Finally, we will find the bound on the cost of route repair in Self-Healing Routing. As already described, in Self-Healing Routing the sender of a packet listens to the response to its broadcast. If such a response does not arrive within the time λ, signaling the failure of the previously existing link, the node rebroadcasts the original packet. After the predefined number of unsuccessful rebroadcasts (two in the current implementation), the sender increases its distance to the destination by 2. We will call such a step a recalibration of the hope distance. Let's consider a sensor network of n nodes in which there is a failure of nodes or their links after which the shortest path from the source to the destination surviving the failure is of length $l<n$. That means that once all nodes not on any of the surviving paths recalibrate their distance to at most n, and the nodes on the surviving paths recalibrate to their correct value, also at most n, then all traffic will flow through the shortest surviving path. The smallest initial distance that nodes needing recalibration might have is 1, so at most $(n-1)*n/2$, hence $O(n^2)$ recalibration steps are needed.

To show that this limit is tight, let's consider a network consisting of two separate lines, each of $n/2-2$ nodes connected to the source and the destination. Let's assume that one line is cut off from the destination at the last hop. It is easy to show by induction on the size of the network, n, that $n(n-2)/8$ recalibrations and $n(n-2)$ hops are required to recalibrate this network after such failure. Once the first packet recalibrates the network, all the subsequent ones would be able to follow the shortest surviving path. To find a bound on the number of hops that such recalibrating packet does, we notice that a recalibration must happen at least every n hops. Indeed, each hop without the

recalibration decreases the distance of the packet to the destination by *1*, so after *n* hops the packet would arrive there. From this observation, it immediately follows that the number of hops made by the packet recalibrating the network after a failure is less than the cube of the number of nodes in the network.

There are other protocols that route on the premise of avoiding neighbor state maintenance and letting receivers contend to forward packets. A typical protocol in this class is GRAB [17] that avoids the use of geographical location information but does not support explicit route repair. GRAB also uses a more aggressive and complex fault-tolerance technique by actively enforcing the flow of redundant packets to follow multiple paths to a destination. Other opportunistic protocols rely on geographic location information to support routing decisions. For instance, BLR [18] uses location coordinates to allow only receivers in an "eligibility region," defined as a region in which all nodes are closer to the destination than the sender and can overhear each others' transmission, to contend to forward packets. A prioritized back-off delay scheme, similar to one used in SHR, ensures that the closest node forwards the packet and suppresses redundant transmissions. However, upon learning the closest receiver, the sender will then forward following packets only to that receiver for a set number of transmissions. This latter technique may only be effective with ideal links. GeRaF [19] employs a similar eligibility region with a prioritized back-off delay technique. However, GeRaF also uses a dual-radio approach with busy-tone signaling to make sure channels are clear before sending data to reduce the probability of collisions. GeRaF uses a request-to-send/clear-to-send (RTS/CTS) packet forwarding technique which imposes more packet forwarding overheard than SHR's approach. IGF [20] is similar to the above protocols, but uses an eligibility region defined as a 60° fan-shaped sector extending from a sender directly towards the destination. If the sender does not hear any responses, it will shift the eligibility region and try to find other receivers. Other similar location-based protocols include PSGR [21] and SIF [22].

Finally, the Cognitive Packet Network (CPN) routing [23] makes another connection between neural and sensor networks. CPN uses Random Neural Networks with Reinforcement Learning (RNNRL) to make routing decisions in a distributed fashion at each node to ensure the best effort QoS based on different user defined QoS criteria, including power saving, important for sensor networks. Unlike SHR, the transmission is carried by Dumb Packets that followed routes selected during the cognitive stage. CPN is well suited for wireless networks with large volume of data transmitted and users with different QoS requirements. In contrast, SHR targets ad hoc, unreliable sensor networks often with a simple pattern of communication from sensing node to a single base station.

Performance of the New Protocol

Traditional routing protocols often use routing tables that dictate to each node the exact neighbor to which a packet should be sent in order to reach a specific destination. Prominent examples of such an approach include AODV [24] and Directed Diffusion [25]. This fundamentally unicast routing approach intrinsically requires each node to actively maintain knowledge of who its neighbors are and what their states are (e.g., active, sleeping, destroyed). It should be also noted that in typical sensor network setting, each node potentially communicates with a base station, so the sensor network with *n* nodes has *n* sources and a single destination. SHR requires in such a case just a single distance to the base station to be stored at each node. This is certainly no more

routing information than traditional path based routing protocols keep as they have to store entire path to the destination either at each source node or along the path to the destination. It should be noted that in reliable network, SHR-PP described above will use the single route to the destination without any delay, but at the same time, it will be ready to change this route in case of any failures. This version of SHR, as shown below, dramatically improves the delay and the delivery ratio of the SHR protocol.

To test the performance of both versions of Self-Healing Routing, we ran a set of simulations of a large scale network to compare SHR performance with the performance of AODV [24] used as an example of a traditional routing protocol. SHR used *r* equal to 50 ms. We ran simulations over a square of size of 8 units populated with randomly placed 500 sensor nodes, each of which had a nominal transmission range of 1 unit. We used the free space propagation model [26] to simulate wireless medium over which packets with a mean size of 1000 bytes were sent at a mean interval of 40 s. Each simulation was executed ten times with different random number seeds. Since all sensor network protocols use flooding to obtain initial routing information (paths to destinations in case of traditional protocols, distances to destination in case of SHR and other grade-based algorithms), we excluded this stage from comparison as it is essentially the same for all ad-hoc sensor network routing protocols.

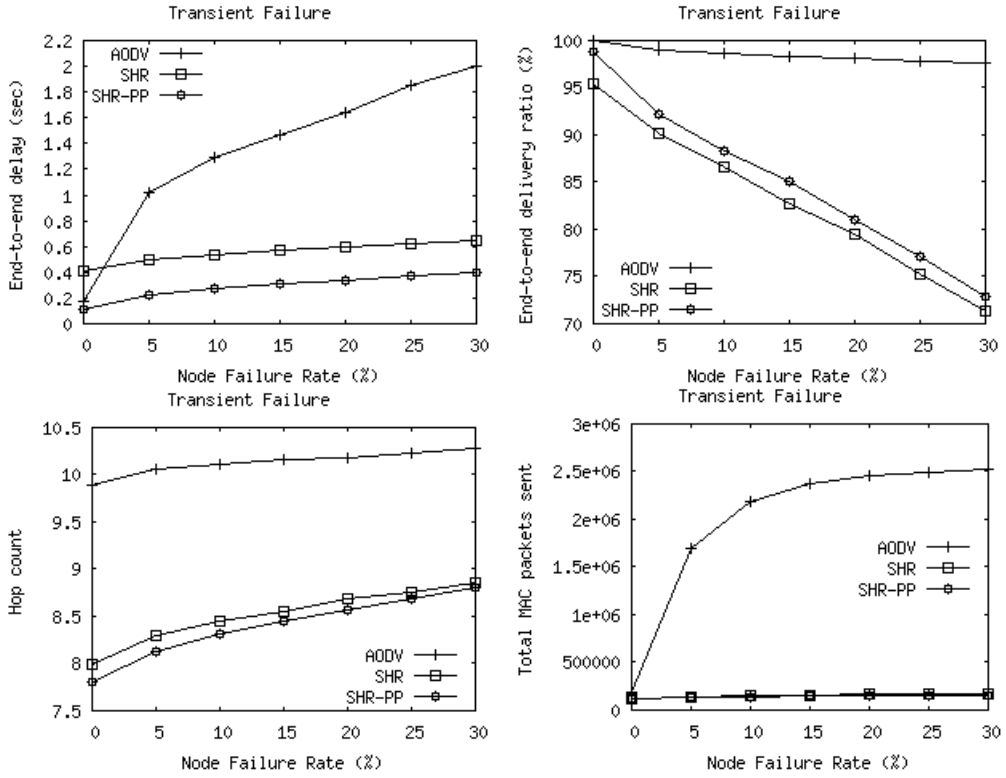


Figure 6. SHR, SHR-PP and AODV performance based on the rate of transient failures.

We performed simulations with variable rate of transient node failures in which each node was assigned a mean active time and a mean sleep time. The sum of these two times was fixed at 200 seconds. The time spent in each mode was distributed exponentially about the mean value. There are several possible causes for transient node failures such as error-prone links, power management induced duty cycles, or excessive packet collisions. Of these, the duty cycle induced failures are the least

disruptive since they may be coordinated with the networking protocol. The presented simulation results are based on a random transient failure model, so they exaggerate the effect of duty cycles on the protocols.

As seen in Figure 6, AODV has the worst transmission delay that increases significantly with the transient failure rate. SHR-PP has by far the smallest delay of the three protocols compared, by factor 10 shorter then AODV's for the most failure prone case. SHR has lower delays than AODV for all cases in which transient failures are present. Both SHR and SHR-PP slightly increase the incurred transmission delay when the transient failure rate is growing and the ratio of transmission delay between the two shrinks from about 4 in case of reliable network to 3 for the transient failure rate reaching 30%. In terms of delivery ratio, AODV is the best, dropping from 100% in a reliable case to 95% for 30% transient failure rate. SHR-PP delivery ratio drops from 100% to 76% over the same region while SHR's is slightly lower, dropping from 92% to 75%. However, AODV requires much greater number of MAC packet transmissions than either SHR or SHR-PP does. This is because to find a new path, the AODV's route repair algorithm initiates a new route request phase, causing a broadcast flood of packets from the point at which the route is severed. AODV uses over 30 times more packets than SHR-PP does. Hence, by implementing a simple replication scheme, in which each packet in SHR-PP is sent 3 times, we could bring the SHR-PP delivery rate above that of AODV while still keeping the number of MAC packets 10 times lower than in AODV. Our preliminary testing of such a scheme demonstrated that indeed packet drops are independent of each other and replication brought expected improvements in the delivery rate.

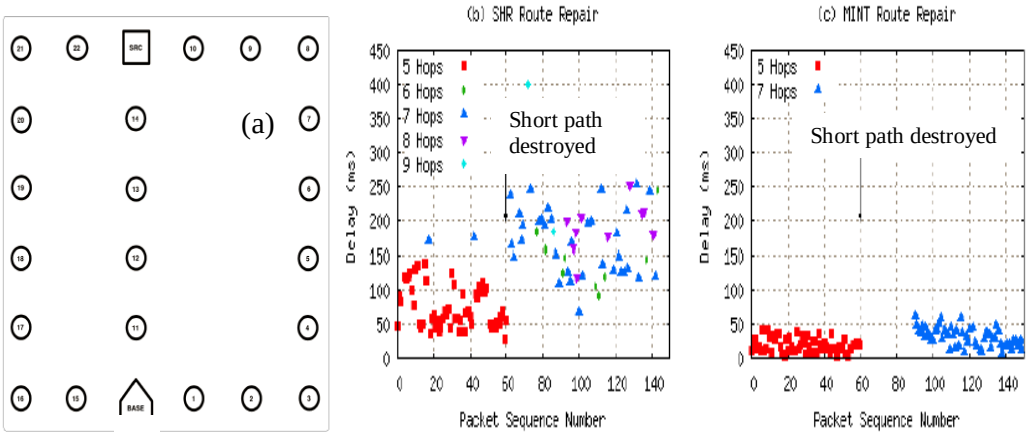


Figure 7. (a) The route repair topology. (b) SHR route repair performance. (c) MintRoute repair packet delivery.

We have also implemented the SHR protocol on MicaZ motes [27] using TinyOS version 1.1.7 to compare performance of this implementation with the basic MintRoute protocol available for this hardware [28]. In the implementation, we used B-MAC with acknowledgments disabled to provide link layer functionality. DATA packets of 29 bytes were sent for 12.5 min at a rate of 5sec/packet in the indoor environment. The radio power was set to -21dBm and a distance of 1m provided a reliable delivery rate. However, with moderate probability some long distance transient links also formed. SHR used λ of 22ms.

The tests were run on a route repair topology shown in Figure 7(a). It contains three disjoint paths of unequal length: a short, medium and long one to test the repair capabilities of the protocols. During testing we blocked mote 12 and 13 in the network by placing a metal container over the motes after the first 5 minutes of the test.

	SHR	***MINTROUTE***
Packets sent	3,368	12,882
Packets received	8,563	41,050
Packet ratio (receive/send)	2.54	3.19
Delivery rate	78%	79.33%
End to end delay	128ms	25 ms
Average hop count	6.10	5.99
Route setup time	5 sec	190 sec

Table 1. Results of experimental runs on MicaZ motes.

As shown in Table 1, both protocols had similar delivery rate and average hop count. The MintRoute has much smaller end-to-end delay, as SHR uses self-selection at each hop which slows down its packet transmission. However, SHR was faster by more than the order of magnitude in establishing the routes to the destination (5 sec. versus 190 sec. for MintRoute). SHR was also much more energy efficient, as it sent about 4 time less packets and received about 5 times less packets than MintRoute. These two radio operations of sending and receiving packets are usually the most expensive in terms of energy in wireless sensor networks. Finally, SHR can repair a broken route and find the next shortest and reliable path quickly, in a few milliseconds. Hence the removal of motes is not detrimental to SHR's performance. MintRoute also recovered from the broken shortest path but required 150 seconds to do so, see Figure 7(b). The destruction of motes can be devastating to MintRoute, making it inadequate for a situation where motes can be compromised.

Concluding Remarks

Intuitively, the more often failures occur in a sensor network, the more control packets are needed in response to topology changes in traditional routing protocols [14-16] that keep explicit routes. This, in turn, increases the number of packet transmissions and packet delivery delays. However, this is not the case for Self-Healing Routing which has been shown to be able to maintain constant end-to-end delays and constant energy consumption as the node failure rate increases [29-30].

Self-Selective Routing works well because the routing problem is essentially the problem of finding the most suitable relay node, and the Lecture Hall Algorithm happens to be an extremely efficient algorithm for identifying an optimum value held by a set of nodes/processors. Following the same principle, any sensor network algorithm that depends on finding the required value or optimum held by the nodes within communication range of the node running the algorithm can greatly benefit from the Lecture Hall Algorithm. As a simple example, in many clustering algorithms, a node must be selected among a group of neighbor nodes to take the role of a local leader. A suitable node could be the one with the most residual energy or the one with the best connectivity. Applying the Lecture Hall Algorithm to such a problem is rather straightforward.

The implications of the concept of computing with time could be more far-reaching. The important consequence is that it effectively connects two distinct research areas. The idea of converting quantities into time delays that dramatically improved the computational efficiency for artificial neural networks has proven to be of great benefit to sensor networks. Moreover, the self-selection mechanism in the Lecture Hall Algorithm is not unlike the winner-takes-all competition in Competitive Learning [31-33], a well-established field of artificial neural networks. Hence, we plan to explore the computing with time paradigm for artificial neural networks as well as any other networks that use broadcast as the basic communication primitive.

Acknowledgement

The presented research was sponsored by US Army Research laboratory and the UK Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the U.S. Government, the UK Ministry of Defence, or the U.K. Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon. Authors wish also to acknowledge help of Mark Lisee and Christopher Morrell in running the simulations with the protocols discussed in this paper.

References

1. Hopfield, J. J., Brody, C. D. and Roweis, S. (1998) Computing with action potentials. *Proceedings of Advances in Neural Information Processing Systems 10*, Denver, CO, 1-6 December, pp. 166-172. MIT Press, Cambridge, MA.
2. Hopfield, J. J. (1995) Pattern recognition computation using action potential timing for stimulus representation. *Nature*, **376**, 33-36.
3. Hopfield, J. J. (1996) Transforming neural computations and representing time. *Proc. National Academy of Sciences of the United States of America*, **93**, 15440-15444.
4. Unnikrishnan, K. P., Hopfield, J. J. and Tank, D. W. (1992) Speaker-independent digit recognition using a neural network with time-delayed connections. *Neural Computation*, **4**, 108-119.
5. Gerstner, W. and Kistler, W. M. (2002) *Spiking Neuron Models*, Cambridge University Press, Oxford.
6. Maass, W. and Bishop, C. M. (1998) *Pulsed Neural Networks*. MIT Press, Cambridge, MA.
7. Bialek, W., Rieke, F., van Steveninck, R. R. d. R. and Warland, D. (1991) Reading a neural code. *Science*, **252**, 1854-1856.
8. Thorpe, S., Fize, D. and Marlot, C. (1996) Speed of processing in the human visual system. *Nature*, **381**, 520-522.
9. VanRullen, R. and Thorpe, S. J. (2001) Is it a bird? Is it a plane? Ultra-rapid visual categorisation of natural and artifactual objects. *Perception*, **30**, 655-668.

10. VanRullen, R. and Thorpe, S. J. (2001) The time course of visual processing: From early perception to decision-making. *J. Cognitive Neuroscience*, **13**, 454-461.
11. Mowshowitz, A., and Bent, G. (2007). Formal Properties of Distributed Database Networks. *Proceedings of the First Annual Conference of Information Technology Alliance,* Adelphi, MD, September 25-27.
12. Chen, G., Branch, J. W. and Szymanski, B. K. (2006) A Self-selection Technique for Flooding and Routing in Wireless Ad-hoc Networks. *J. Network and Systems Management*, **14**, 359-380.
13. Wasilewski, K., Branch, J., Lisee, M. and Szymanski, B. K. (2007). Self-healing routing: a study in efficiency and resiliency of data delivery in wireless sensor networks. *Proceedings of Unattended Ground, Sea, and Air Sensor Technologies and Applications IX*, Orlando, FL, 9-13 April, pp. 656218-1 - 656218-12, vol. 6562, SPIE Press, Bellingham WA.
14. Johnson, D. B., Maltz, D. A. and Broch, J. (2001) DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks. In Perkins, C. E. (ed.), *Ad Hoc Networking*, Addison-Wesley, Reading, MA.
15. Perkins, C. E. and Bhagwat, P. (1994) Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. *Computer Communication Review*, **24**, 234.
16. Perkins, C. E. and Royer, E. M. (1999) Ad-hoc on-demand distance vector routing. *Proceedings of WMCSA99,* New Orleans, LA, *25-26* February, pp. 90-100, IEEE CS Press, Los Alamitos, CA.
17. Ye, F. Zhong, G., Lu, S. and Zhang, L. (2005) Gradient broadcast: a robust data delivery protocol for large scale sensor networks, *ACM Wireless Networks*, **11,** 285-298.
18. Heissenbttel, M., Braun, T., Bernoulli, T. and Waelchli, M. (2004) BLR: beaconless routing algorithm for mobile ad hoc networks, *Elsevier's Computer Communications Journal*, **27**, 1076-1086.
19. Zori M. and Rao, R. R. (2003) Geographic Random Forwarding (GeRaF) for ad hoc and sensor networks: multihop performance, *IEEE Transactions on Mobile Computing*, **2**, 337-348.
20. Blum, B. M., He, T., Son, S. and Stankovic, J. A. (2003) *IGF: a robust state-free communication protocol for sensor networks*, Technical Report CS-2003-11, University of Virginia, VA.
21. Xu, Y. Lee, W.-C., Xu, J. and Mitchell, G. (2005) PSGR: priority-based stateless geo-routing in wireless sensor networks, *Proceedings of IEEE Conf. Mobile Ad-hoc and Sensor Systems*, Washington, DC, 7-10 November, pp. 8, IEEE Press, Los Alamitos, CA.
22. Chen, D., Deng, J. and Varshney, P. K. (2005) A state-free data delivery protocol for multihop wireless sensor networks, Proceedings of IEEE Wireless Communications and Networking Conference, New Orleans, LA, 13-17 March, pp. 1818-1823, IEEE Press, Los Alamitos, CA.
23. Gelenbe, E. and Lent, R. (2004) Power aware ad hoc cognitive packet networks, *Ad Hoc Networks*, **2**, 205-216.
24. Perkins, C. E. and Belding-Royer, E. M. (1999) Ad hoc On-Demand Distance Vector Routing. Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, New Orleans, LA, 25-26 February, pp. 90-100, IEEE Press, Los Alamitos, CA.

25. Intanagonwiwat, C., Govindan, R., and Estrin, D. (2000) Directed diffusion: a scalable and robust communication paradigm for sensor networks, Proceedings of ACM MobiCom, Boston, MA, 6-11 August, pp. 56-67, ACM Press, New York, NY.
26. Rappaport, T. S. (2002) *Wireless Communications: Principles and Practice*, Prentice Hall, Upper Saddle River, NJ.
27. Crossbow Technology, Inc., http://www.xbow.com.
28. Woo, T. Tong, and Culler, D. (2003) Taming the underlying challenges of reliable multihop routing in sensor networks,. Proceedings of the ACM SenSys, Los Angeles, CA, 5-7 November, pp. 56-67, ACM Press, New York, NY.
29. Chen, G., Branch, J. W. and Szymanski, B. K. (2005) Local Leader Election, Signal Strength Aware Flooding, and Routeless Routing. *Proceedings of WMAN05, Parallel and Distributed Processing Symposium*, Denver, CO, 4-8 April, p. 244, IEEE CS Press, Los Alamitos, CA.
30. Chen, G., Branch, J. W. and Szymanski, B. K. (2006) A Self-selection Technique for Flooding and Routing in Wireless Ad-hoc Networks. *J. Network and Systems Management*, **14**, 359-380.
31. Grossberg, S., (1976) Adaptive Pattern-Classification and Universal Recoding.1. Parallel Development and Coding of Neural Feature Detectors. *Biological Cybernetics*, **23**, 121-134.
32. Malsberg, v.d. (1973) Self-Organizing of Orientation Sensitive Cells in the Striate Cortex. *Kybernetick*, **14,** 85-100.
33. Rumelhart, D. E. and Zipser, D. (1985) Feature Discovery by Competitive Learning. *Cognitive Science*, **9**, 75-112..