Machine Learning Across the WSN Layers

Anna Förster[1,2] and Amy L. Murphy[3]
[1]*University of Lugano,* [2]*Networking Laboratory SUPSI,* [3]*FBK-IRST*
[1,2]*Switzerland,* [3]*Italy*

Wireless sensor networks (WSNs) have seen rapid research and industrial development in recent years. Both the costs and size of individual nodes have been constantly decreasing, opening new opportunities for a wide range of applications. Nevertheless, designing software to achieve energy-efficient, robust and flexible data dissemination remains an open problem with many competing solutions.

In parallel, researchers have effectively exploited machine learning techniques to achieve efficient solutions in environments with distribution and rapidly fluctuating properties, analogous to WSN domains. Applying machine learning techniques to WSNs inherently has the potential to improve the robustness and flexibility of communications and data processing, while simultaneously optimizing energy expenditure.

This chapter concentrates on applications of machine learning at all layers in the WSN network stack. First, it provides a brief background and summary of three of the most commonly used machine learning techniques: reinforcement learning, neural networks and decision trees. Then, it uses example research from the literature to describe current efforts at each level of the stack, and outlines future opportunities.

1. Wireless Sensor Networks

Extensive research effort has been invested in recent years to optimize communications in wireless sensor networks (WSNs). Researchers and application developers typically use a communication stack model such as that depicted in Figure 1 to structure the communications of WSNs and to better manage its challenges. In particular, the following properties of WSNs should be considered while designing innovative and efficient solutions (Akyildiz et al., 2002; Römer & Mattern, 2004).

- *Wireless ad-hoc nature.* No fixed communication infrastructure exists. The shared wireless medium places restrictions on the communication between nodes and poses new problems such as asymmetric links. However, it offers the broadcast advantage: a transmitted packet, even if sent in unicast to another node, can be overhead and thus received by all neighbors of the transmitter.
- *Mobility and topology changes.* WSNs may support dynamic application scenarios. New nodes may be added to the network, and existing nodes may move either within or out of the network. Nodes may cease to function, and connectivity among surviving nodes changes over time. WSN applications must be robust against such topology dynamics.
- *Energy limitations.* The basic WSN scenario includes a large number of sensor nodes, and a limited number of more powerful base stations. As such, most WSN nodes have

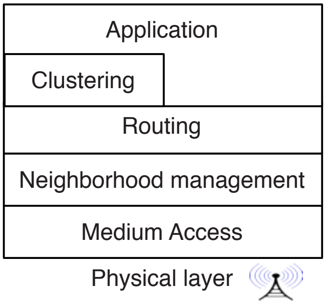


Fig. 1. The WSN communication stack

limited energy supplies and maintenance or battery recharging is often impossible after deployment. Communication tasks consume a large proportion of the energy available on the nodes, and thus to ensure sustained long-term operation, radio communication must be frugally managed.

- *Physical distribution.* Each node in a WSN is an autonomous computational unit that communicates with its neighbors via messages. Data is collected throughout the network and can be gathered at a central station only with high communication costs. Consequently, algorithms that require global information from the entire network become very expensive. Thus, distributed algorithms are highly desirable.

The next section proceeds with a brief introduction to machine learning approaches that have been successfully applied to one or more layers of the communication stack. We then provide concrete examples of how machine learning has been exploited to minimize communication overhead at all layers from neighborhood management up to the application.

2. Machine Learning Techniques

Machine learning (ML) is a sub-field of artificial intelligence that *"is concerned with the question of how to construct computer programs that automatically improve from experience"* (Mitchell, 1997). Precisely this property makes the family of ML algorithms and techniques appealing for efficient communications in WSNs. This section presents some widely applied ML approaches that form the basis for the exemplary applications in the following sections. Alternate ML techniques include, among many others, genetic algorithms (Mitchell, 1997) and swarm intelligence algorithms such as ant colony optimization (Dorigo & Stuetzle, 2004). While these are powerful machine learning techniques for solving various challenging problems, they are less suitable for communications in wireless sensor networks (Kulkarni et al., 2009) because of their high communication overhead.

2.1 Decision Tree Learning

In many classification problems the items to be classified exhibit a number of clearly defined features, represented as attribute-value pairs. For example, if we want to classify all possible fruits, we can use features such as size, shape, color, taste, etc. with corresponding attribute-value pairs such as *color = orange*. We could define the possible classification clusters by their features and attribute-value pairs. Then, for some unclassified object, we check all of its features to match it with one of the clusters. However, a so called classification tree is more efficient, since it offers structure the classification approach and usually classifies a sample based only on a few features. In such a tree the leaves represent classification clusters and the branches represent conjunctions of features. Continuing with our fruit example, a classification tree will ask at the very first branch what is the color of the sample. If there is only one cluster with the color blue (e.g., blueberry), then the branch leads directly to the classification leaf of blueberries without asking for any other features. It is clear from this example that the most important question when constructing such trees is *"which attribute to check at the root of the tree, which next?"*

Decision tree learning is a machine learning technique that uses a set of already classified training samples for constructing the optimal tree. Optimal in this case refers to the number of feature checks before classification. Of course, the classification problem might exhibit noise samples, which also need to be accommodated. For example, strawberries are usually red, but sometimes we observe also green ones. Thus, the decision tree will either wrongly classify the green strawberry as something else or it needs to use all of the other features (size, shape, etc.) and ignore the color. The final decision depends on the samples in the training set and on the importance of different features. As we have seen above, some of the features may become irrelevant, while others become highly important.

There are two main algorithms for constructing decision trees: ID3 and the its successor C4.5 (Mitchell, 1997). Each sample s_i from the training set S consists of a vector of feature values f_i and is already classified as belonging to cluster c_j. C4.5 computes for each feature the information gain when splitting on this feature. In other words, which feature separates the clusters best? In our fruit example from above, checking for the shape in the root of the tree is probably a bad decision, since many if not all fruits are round. However, checking for the size might separate watermelons and melons from all the rest very well. Thus, the information gain of the feature *size* is higher than for any other feature. C4.5 takes the feature with the highest information gain and puts it in the root of the tree. Then it recursively computes the information gain for the resulting subclasses until all or nearly all samples from the training set are classified. Clearly, not all of the samples will be classified successfully, as exemplified in the discussion above. However, this is not possible even with the brute-force method of checking all possible features and their values, because the training set also includes noisy data. A formal description of decision tree learning can be found in (Mitchell, 1997).

In the context of wireless sensor networks, classification problems like this arise when classifying links as good or bad based on data such as signal strength or delivery rate, or classifying sensory data as important or not. We show an application of decision trees to link quality estimation in Section 3. Decision tree learning is suited for such classification problems since it is fast to both train and execute. Additionally, implementing a decision tree on a resource-restricted sensor node is simple. On the other hand, training should be performed offline to save node energy, requiring the classification problem to be relatively stable.

2.2 Neural Networks

An artificial neural network (or simply neural network, NN) is a mathematical models of a function $F : X \to Y$. The initial inspiration comes from biological networks of neurons. NNs consist of simple nodes or neurons, interconnected as in Figure 2. Simple functions are usually associated with each node (e.g., addition) and weights are assigned to the connections between the nodes. Data flows from the input (left column of neurons in Figure 2) through the whole network, using the connections between the nodes and arriving at the output neurons (right column of neurons). The most important property of neural networks is their ability to

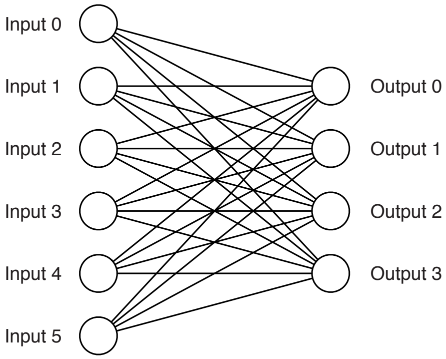


Fig. 2. A generic architecture of an artificial neural network with input and output layers.

learn — to adjust the weights between the input and the output to exactly reflect the learned function.

For learning or *training* a neural network, a set of training data is needed, where possible inputs have already been mapped to the needed output. For example, for classification of hand-written numbers, different pictures (input) are classified as numbers (output). However, in contrast to decision trees, the input cannot be described with features and attribute-value pairs. Instead, it is represented as a point in an N-dimensional space. For example, a hand written picture of the size 32×32 pixels is represented as a point in the $32 \times 32 = 1024$ dimensional space. There will also be 1024 input neurons in the neural network and exactly 10 output neurons, one for each digit. The weights connecting the input neurons with the output ones need to be set such that the correct output neuron "fires" — only the output neuron has a value of 1 and all others a value of 0. This is done by presenting the network with examples, consisting of input and output. With every sample, the weights are corrected such that the correct neuron fires. Thus, in our 1024-dimensional space, the hand-written samples will cluster around some points in this space, representing the different digits from 0 to 9. Incoming input samples can be then classified according to their distance to the clusters and the closest cluster is taken.

The above described neural network is a so called supervised offline learning algorithm. *Supervised* refers to the training set, which has already been classified. *Offline* refers to the necessary training of the network before using it for classification. However, there also exist *unsupervised* and *online* learning neural networks. An example of such a network is used for learning the data model for incoming sensor readings in Section 6. More information about neural networks and how to train them can be found in (Mitchell, 1997).

Neural networks are well suited for complex classification problems where features or attribute-values pairs are not available. However, they have larger memory and processing requirements than, for example, decision tree learning. On the other hand, as we will show in Section 6, these techniques are applicable in WSNs for static classification problems such as data models or link quality estimation. In addition, they can be efficiently implemented even on standard sensor nodes because of their relatively low memory requirements.

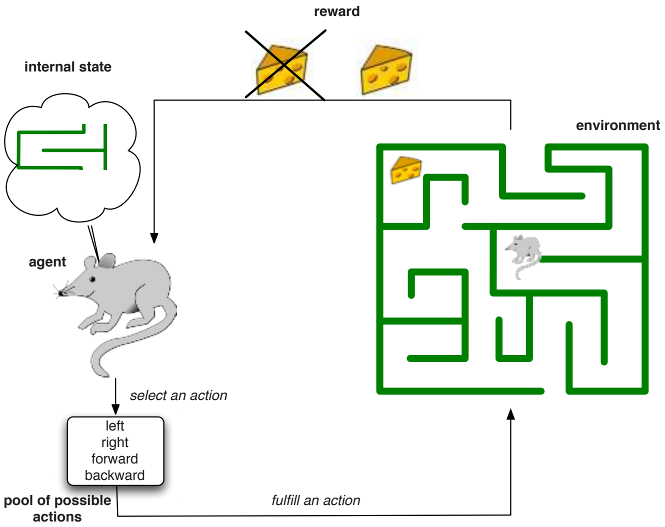


Fig. 3. General reinforcement learning model. The agent selects one action according to its current internal state (current view of the environment and previous knowledge), fulfills this action and observes a reward.

2.3 Reinforcement Learning

Reinforcement learning (RL) (Mitchell, 1997; Sutton & Barto, 1998) is a biologically inspired machine learning technique, where the learning agent acquires its knowledge from direct interaction with its environment. A simple example is a mouse in a maze, trying to find the path to a piece of cheese (see Figure 3). At any moment, it must select a direction to move. The result of each action is either finding cheese or not. This maps to the reinforcement learning technique in which agents (e.g., the mouse) select actions (e.g., direction to move) and receive rewards (e.g., cheese) from the environment for each action. A well-known and widely used RL algorithm is Q-Learning, which model consists of:

Agent states. The learning agent has a finite set of possible states S and s_t represents the agent's state at time step t. In our example from Figure 3, the state of the mouse is its current position in the maze.

Actions. Q-Learning associates a different set of actions A_S to each of the states in S. In our maze environment, the actions are represented by the movement steps of the mouse — forward, backward, left, right.

Immediate rewards. There is an associated immediate reward $r(s_t, a_t)$ with each of the state transitions. In our example, all of the state transitions that do not lead to the goal state have immediate rewards of 0 (no cheese) and the ones leading to the goal state have an immediate reward of 1 (cheese reached). The agent can see only the actions with their associated rewards from its current state. It does not have any global knowledge about the environment, its states and their rewards.

Action costs. In addition to rewards, there is also a cost $c(s_t, a_t)$ associated with each action in each state. This is again a scalar value, representing how costly this action is. In our example, it costs one unit of energy (one bite of cheese) for the mouse to make any movement. Costs are often considered negative rewards, thus they are subtracted directly from the immediate reward.

Value function. In contrast to immediate rewards, which are associated to each action in each state and are easily observable, the value function represents the *expected total accumulated reward*. The goal of the agent is to learn a sequence of actions with a maximum value function, that is, the reward on the taken path is maximized.

Q-Values. To represent the currently expected total future reward at any state, a Q-Value is associated to each action and state $Q(s_t, a_t)$. The Q-Value represents the memory of the learning agent in terms of the quality of the action in this particular state. In the beginning Q-Values are usually initialized with zeros, representing the fact that the agent knows nothing. Through trial and experience the agent learns how good some action was. The Q-Values of the actions change through learning and finally represent the absolute value function. After convergence, taking the actions with the greatest Q-Values in each state guarantees taking the optimal decision (path).

Updating a Q-Value. A simple rule exists to update a Q-Value after each step of the agent:

$$Q(s_{t+1}, a_t) = Q(s_t, a_t) + \gamma(R(s_t, a_t) - Q(s_t, a_t)) \tag{1}$$

The new Q-Value of the pair $\{s_{t+1}, a_t\}$ in state s_{t+1} after taking action a_t in state s_t is computed as the sum of the old Q-Value and a correction term. This term consists of the received reward and the old Q-Value. γ is the learning constant. It prevents the Q-Values from changing too fast and thus oscillating. The total received reward is computed as:

$$R(s_t, a_t) = r(s_t, a_t) + c(s_t, a_t) \tag{2}$$

Where $r(s_t, a_t)$ is the immediate reward as defined above and $c(s_t, a_t)$ is the cost of taking the action a_t in state s_t.

Exploration strategy (action selection policy). Learning is performed in episodes, e.g., the mouse takes actions in its environment and updates the associated Q-Values until reaching the cheese. After completion, a new episode begins, repeating until the Q-Values no longer change. The question is how to select the next action. Always taking the actions with maximum Q-Value (greedy policy) will result in finding locally minimal solutions. On the other hand, selecting always random (random policy) will mean ignoring prior experience and spending too much energy to learn the complete environment.

These two extreme strategies are called *exploitation* and *exploration* of routes. The problem of combining and weigthing both so that optimal results are achieved as fast as possible has been extensively studied in machine learning (Sutton & Barto, 1998). The most commonly used strategy is called *ϵ-greedy*: with probability ϵ the agent takes a random action and with probability $(1 - \epsilon)$ it takes the best available action.

RL is well suited for distributed problems such as routing. It has medium requirements for memory and rather low computation needs at the individual nodes. This arises from the need to keep many different possible actions and their values. It needs some time to converge, but it is easy to implement, highly flexible to topology changes and learns the optimal solution (e.g., shortest paths).

3. Neighborhood Management Layer

One major problem of communications in wireless sensor networks is the unreliability of the links. At any time, a previously reliable link may disappear, while others might become more reliable than before. This is influenced by the environmental conditions (weather, moving people, etc.) and cannot be controlled or predicted. Unreliable links are a great challenge for routing protocols, since selecting reliable routes is crucial for saving energy in the network as a whole. Thus, a special layer is needed between the medium access and the routing layers to provide the routing layer with up-to-date information of the reliability of connections to neighbors. The resulting protocols are called link or neighborhood management protocols. The most important properties of a good neighborhood management protocol are (Karl & Willig, 2005):

- **Precision.** The links should be precisely evaluated in their quality and reliability.
- **Agility.** The link manager should react quickly to changes.
- **Stability.** The link manager should not be influenced by short aberrations.
- **Energy efficiency.** The link manager should spend as little communication and processing power for its operation as possible.

Many researchers have put extensive effort in searching for good link estimators. Two main classes exist: passive and active estimators. Passive estimators use readily available information on the nodes for their estimations, such as RSSI of received packets, number of received packets, etc. Active estimators pro-actively send probe packets to discover the link quality to their neighbors. Of course combinations of passive and active estimators also exist that use readily available information as much as possible and send additional probe packets when needed.

Traditional approaches use rules of thumb to estimate the quality of links given some local information on the nodes. Typically they use rules such as "if $RSSI > 80$ then $quality = good$", implement them on a hardware testbed, test it and fine-tune the parameters of the approach. However, this design phase is long and inefficient, based mainly on experience and intuition. Nevertheless, some of these approaches have been extensively evaluated and widely used for real applications, e.g., through integration with existing routing protocols such as MintRoute (Woo et al., 2003) or Arbutus (Puccinelli & Haenggi, 2008).

3.1 MetricMap: Supervised Learning for Link Quality Estimation.

A more sophisticated approach is to try to automatically gather relevant features and properties readily available at the nodes, and to learn to estimate the quality of the links from them. A simple, yet powerful algorithm is MetricMap (Wang et al., 2006), developed at Princeton University in 2006. MetricMap uses decision tree learning to offline learn to estimate link quality based on previously gathered link samples. The decision trees uses locally available data and learns to classify links as good or bad. The acquired rules are integrated with a routing protocol (in this case MintRoute (Woo et al., 2003)) and are used online to predict link quality based only on locally available information such as delivery rate or RSSI levels of incoming packets. The authors of MetricMap designed their algorithm in two main steps: sample collection and offline training. First, they used the MistLab [1] sensor network testbed at MIT to gather link samples together with all available features, shown in Table 1. Each link sample was labeled “good” or “bad”, according to its Link Quality Indication (LQI) value.

[1] http://mistlab.csail.mit.edu

Table 1. Link sample features used in MetricMap.

Feature	Description	Locality
RSSI	received signal strength indication	local
sendBuf	send buffer size	local
fwdBuf	forward buffer size	local
depth	node depth from base station	non-local
CLA	channel load assessment	local
pSend	forward probability	local
pRecv	backward probability	local

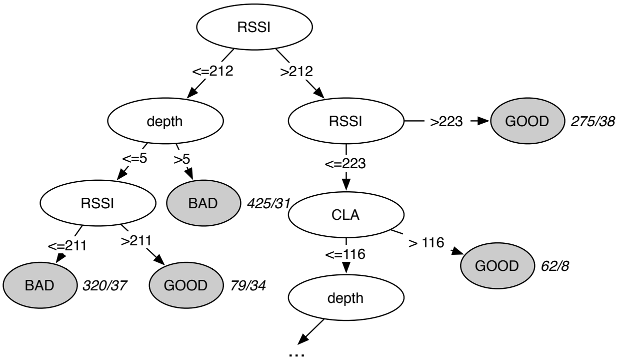


Fig. 4. Part of the decision tree for estimating link quality, computed by MetricMap.

LQI is an indicator of the strength and quality of a received packet, introduced in the 802.15.4 standard and provided by the CC2420 radios of the MicaZ nodes in MistLab. Measurement studies with LQI have shown it is a reliable metric when estimating link quality. However, LQI is available only after sending the packet. It is not available for estimating the future quality of some link before any packets are sent.

The training set, consisting of labeled link samples, was used to compute offline a decision tree, which classifies the links as good or bad, based on the features from Table 1. The output of the decision tree learner is presented in Figure 4 (a), together with classification results from the training phase in the format: (total samples in category / false positive classifications). The authors used the Weka workbench (Witten & Frank, 2005), which contains many different implementations of machine learning techniques, including the C4.5 algorithm for decision tree learning (see Section 2.1).

The acquired rules are used to instrument the original implementation of MintRoute. In a comparative experimental evaluation on a testbed the authors showed that MetricMap outperforms MintRoute significantly in terms of delivery rate and fairness, see Figure 4 (b) and (c). MetricMap also does not incur any additional processing overhead, since the evaluation of the decision tree is straightforward.

3.2 Discussion of MetricMap

The authors of MetricMap have clearly shown that supervised learning approaches are easy to implement and use in a wireless sensor network environment and significantly improve the routing performance of a real system. Similar approaches can be applied to other testbeds and real deployments. The only requirement is that the general communication properties of the network do not change over time. This could be particularly challenging in outdoor environments, where weather, temperature, sunlight, etc., influence the wireless communications. Detailed and long-running experiments under changing climate conditions are necessary to demonstrate the applicability of MetricMap-like routing optimizations. However, the expectation is that the offline learning procedure needs to be re-run in order to adapt to the changing environment, which could be very costly. In case this hypothesis proves to be true, distributed methods for automatic link quality estimation need to be developed. On the other hand, implementing decision tree or rule-based learning on sensor nodes seems to be practical, since these techniques do not have high memory or processing requirements.

4. Routing Layer

The *routing* challenge refers to the general problem of transferring a data packet from one node in the network to another one, where direct communication between the nodes is impossible. The problem is also known as multi-hop routing, referring to the fact that typically multiple intermediate nodes are used to relay the data packet to its destination. A routing protocol identifies the sequence of intermediate nodes to ensure delivery of the packet. A differentiation between unicast and multicast routing protocols exists in which unicast protocols route the data packet from a single source to a single destination, while multicast routing protocols route the data packet to multiple destinations simultaneously.

There is a huge body of research on routing for WSNs and in general for wireless ad hoc networks. The main challenges are managing unreliable communication links, node failures and node mobility, and, most importantly, using energy efficiently. Well-known unicast routing paradigms for WSNs are for example Directed Diffusion (Silva et al., 2003) and MintRoute (Woo et al., 2003), which select shortest paths based on hop counts, latency and link reliability. Geographic routing protocols such as GPSR (Karp & Kung, 2000) use geographic progress to the destination as a cost metric to greedily select the next hop.

Next we present an effort to achieve good routing performance and long network lifetimes with Q-Learning, a reinforcement learning algorithm presented in Section 2.3. It uses a latency-based cost metric to minimize delay to the destination and is one of the fundamental works on applying machine learning to communication problems.

4.1 Q-Routing: Applying Q-Learning to Packet Routing

Q-Routing (Boyan & Littman, 1994) is one of the first applications of Q-Learning, as outlined in Section 2.3 and (Watkins, 1989), to communications in dynamically changing networks. Originally it was developed for wired packet-switched networks, but it is also easily adaptable to the wireless domain.

The learning agents are the nodes in the network, which learn independently from one another the minimum-delay route to the sink. At each node, the available actions are the node's neighbors. A value $Q_{x,t}(d,y)$ is associated with each neighbor, reflecting the delay estimate d at time t of node x to reach the sink through neighbor y. The update rule for the Q-Values is:

$$Q_{x,t+1}(d,y) = Q_{x,t}(d,y) + \gamma\left(q + s + R - Q_{x,t}(d,y)\right) \tag{3}$$

where γ is the learning rate, fixed to 0.5 in the original Q-Routing paper (Boyan & Littman, 1994), q is the time the last packet spent in the queue of the node, s is the transmission time to reach neighbor y and R is the reward received from neighbor y, calculated as:

$$R_y = \min_{z \in (neighbors\ of\ y)} Q_{y,t}(d,z) \tag{4}$$

The authors applied their algorithm to three different fixed topologies with varying numbers of nodes. They measured the network performance of Q-Routing against a shortest-path routing algorithm under multiple network loads. Under high network loads (the paper does not specify the exact load) Q-Routing performs significantly better than shortest-path because it takes into account the waiting time in the queue. Thus, it spreads the traffic more uniformly, achieves lower end-to-end delivery rates and avoids queue overflows. Importantly, the network load can change during its lifetime and Q-Routing quickly and non intrusively re-learns the optimal paths.

4.2 Discussion of Q-Routing

While the original paper contains no explanation for the selected learning rate, nor details about initialization and action selection policy, and the reward delivery implementation is not given, the experience of other researchers offer answers to these questions. They show that a simple ϵ-greedy action policy is energy-efficient and easy to implement. Initialization of Q-Values can be random, zero or with some a priori available routing information on the nodes, such as estimation of the delay to the sinks. The main goal of the learning rate is to avoid initial oscillations of the Q-Values. We have shown in our analysis of the multicast routing protocol FROMS (Förster & Murphy, 2007) that it can be fixed to 1 if the Q-Values are initialized with good estimates of the real costs. In such a case, a learning rate of 1 speeds up the learning process significantly without the risk of oscillating values. We have also shown an efficiently mechanism to implement the reward mechanism in WSNs, specifically by piggybacking rewards on usual data packets. Due to the inherent broadcast nature of the wireless communication,all the neighboring nodes hear the data packets together with the rewards. Additionally, not only will the preceding node update its Q-Values, but all overhearing nodes can as well, further speeding up the learning process.

The authors of Q-Routing have clearly shown how to efficiently apply reinforcement learning techniques to challenging communication problems and to significantly improve network performance. Although the work is rather preliminary as the experiments are limited to only a few topologies and evaluation metrics, Q-Routing has inspired a number of other routing protocols, especially in WSNs.

5. Clustering and Aggregation Layer

Clustering and data aggregation are powerful techniques that inherently reduce energy expenditure in wireless sensor networks while at the same time maintaining sufficient quality of the delivered data. Clustering is defined as the process of dividing the sensor network into groups. Often a single cluster head is then identified within each group and made responsible for collecting and processing data from all group members, then sending it to one or more base stations.

While this approach is seemingly simple and straightforward, efficiently achieving it involves solving four challenging problems. First, the clusters themselves must be identified. Second, cluster heads must be chosen. Third, routes from all nodes to their cluster head must be discovered. And finally, the cluster heads must efficiently route data to the sink(s).

Traditional clustering schemes can be coarsely divided into two main classes: random- and agreement-based approaches. The first class are mostly variations or modifications of

LEACH (Rabiner-Heinzelman et al., 2000), in which nodes choose to be cluster heads with an a-priori probability. Subsequently, cluster heads flood a cluster head role assignment message to their neighbors, which in turn identify the nearest cluster head as their own. In contrast, agreement-based protocols first gather information about their k-hop neighborhood and then decide on the cluster heads (Bandyopadhyay & Coyle, 2003; Demirbas et al., 2004; Younis & Fahmy, 2004). Again, the cluster heads announce themselves to the network. The main difference between these two classes are the properties of the resulting clusters: their shape, size, number of nodes per cluster, and spreading of remaining energy among the nodes in a cluster. Random-based protocols produce non-uniformly sized clusters with varying remaining energies on the nodes. However, they do not require a lot of communication overhead for selecting the cluster heads. On the other hand, agreement-based protocols produce well-balanced clusters, but require extensive communication overhead for gathering the neighborhood information and for agreeing on the cluster head role.

5.1 CLIQUE: Role-Free Clustering Protocol with Q-Learning

One of the challenges facing state of the art clustering is handling node and cluster head failures without losing a substantial part of the data during the recovery process. Here we present a protocol that explicitly addresses recovery after such failures, while at same time avoiding completely the cluster head agreement process. CLIQUE (Förster & Murphy, 2009) is our own role-free clustering protocol based on Q-Learning (Section 2.3). First, it assumes that cluster membership is known a priori, for example based on a geographic grid or room location information on the sensor nodes. It further assumes that the possibly multiple sinks in the network announce themselves through network-wide data requests. During the propagation of these requests all network nodes are able to gather 1-hop neighborhood information including the remaining energy, hops to individual sinks and cluster membership. When data to transmit becomes available, nodes start routing it directly to the sinks. At each intermediate node they take localized decisions whether to route it further to some neighbor or to act as a cluster head and aggregate data from multiple sources.

The learning agents are the nodes in the network. The available actions are $a_{n_i} = (n_i, D)$ with $n_i \in \{N, self\}$, in other words either routing to some neighbor in the same cluster or serving as cluster head and aggregating data arriving from other nodes. After aggregation, CLIQUE hands over the control of the data packet to the routing protocol, which sends it directly and without further aggregation to the sinks. In contrast to the original Q-Learning, we initialize the Q-Values not randomly or with zeros, but with a initial estimation of the real costs of the corresponding routes, based on the hop counts to all sinks and the remaining batteries on the next hops.

The update rule for the Q-Values is:

$$Q_{new}(a_{n_i}) = Q_{old}(a_{n_i}) + \alpha(R(a_{n_i}) - Q_{old}(a_{n_i})) \tag{5}$$

where $R(a_{n_i})$ is the reward value and α is the learning rate of the algorithm. We use $\alpha = 1$ to speed up learning and because we initialize the Q-values with non-random values. Therefore, with $\alpha = 1$, the formula becomes $Q_{new}(a_{n_i}) = R(a_{n_i})$, directly updating the Q-value with the reward. The reward is calculated as:

$$R(n_{self}) = c_{n_i} + \min_{n_i \in N} Q(a_{n_i}) \tag{6}$$

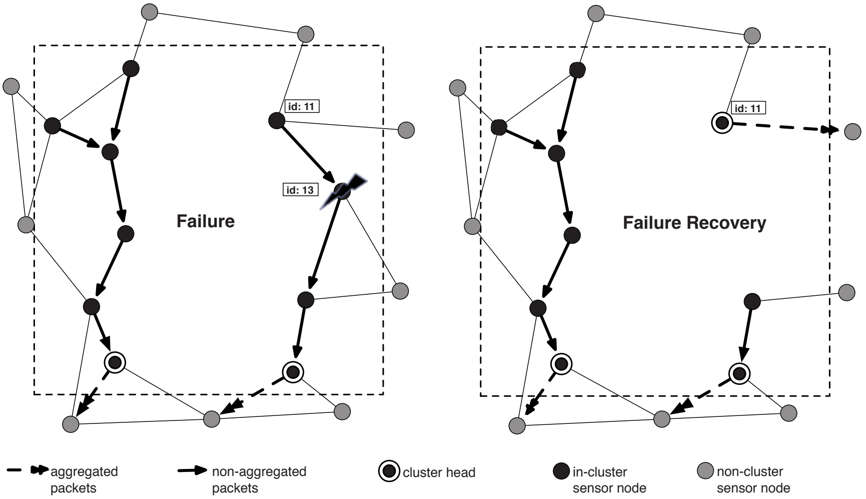


Fig. 5. Learned cluster head in a disconnected scenario (a), recovery after node failure (c) and some experimental results with CLIQUE for delivery rate and network lifetime.

where c_{n_i} is the cost of reaching node n_i and is always 1 (hop) in our model. This propagation of Q-values upstream is piggybacked on usual DATA packets and allows all nodes to eventually learn the actual costs. We use traditional ϵ-greedy action selection policy with low ϵ for exploring the routes and learning the optimal cluster head.

5.2 Discussion of CLIQUE

The most important property of CLIQUE is its role-free nature. In contrast to most cluster head selection algorithms, it does not try to find the optimal cluster head (in terms of cost), but incrementally *learns* the best without knowing either where or who the real cluster heads are. As a result, at the beginning of the protocol, multiple nodes in the cluster may act as cluster heads. While this temporarily increases the overhead, it is a short-term tradeoff in comparison to the overhead required to agree on a single cluster head. Later in the protocol operation, after the real costs have been learned, multiple cluster heads occur only in disconnected clusters, where a single cluster head cannot serve all cluster members.

A particularly interesting cluster head learning scenario is presented in Figure 5 (left), where the cluster is disconnected. Such a scenario is challenging for traditional clustering approaches as they need a complicated recovery mechanism, typically with large control overhead. On the contrary, CLIQUE automatically identifies two cluster heads, as shown in the figure. Figure 5 (right) shows a recovery scenario in which node 13 fails. Node 11 is no longer able to send its data to the cluster head and needs to find a new solution. Instead of searching for a new route to the cluster head it simply becomes a cluster head itself. Because of its learning properties and network status awareness, this requires no control overhead.

We believe that CLIQUE represents the beginning of a new family of role-free clustering protocols, with low communication overhead and very robust against node failures. Various cost metrics can be easily incorporated. Nevertheless, one drawback is the use of the geographic grid for cluster membership, which requires location information on the nodes. Further research in this area is desirable to improve the protocol.

6. Data Integrity

One of the major problems of in-network processing and aggregation in WSNs is the recognition and filtering of faulty data readings before they are sent to the base stations. This is often referred to as the data integrity problem. A typical example is a large climate monitoring sensor network, delivering information about temperature, humidity or light conditions. Multiple sensors are usually deployed to monitor the same area for redundancy. While in the previous sections we have broadly discussed how to manage communication failures, data integrity refers to the problem of sensing failures. For example, some light sensing nodes could be covered by debris and deliver faulty readings. It is desirable to recognize these readings as fast as possible in a distributed way before they are sent to the base station to minimize communication.

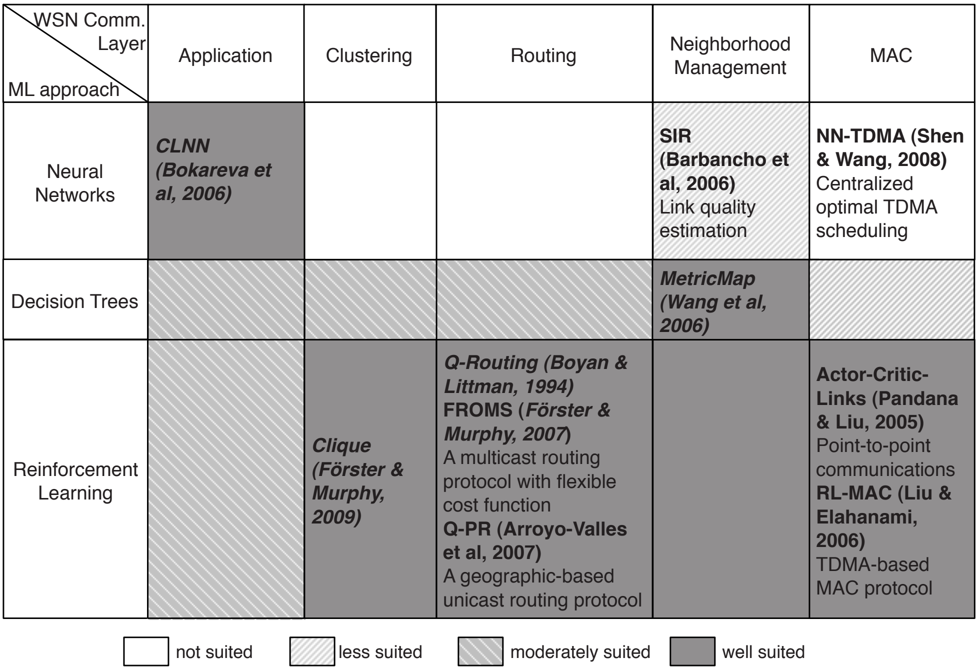
6.1 CLNN-Integrity: Using Neural Networks to Recognize Faulty Sensor Data

Neural networks are very often used to learn to classify data readings. Here we present a semi-distributed approach to learn the data characteristics of incoming sensory data and to classify it as valid or faulty. The learning neural network is implemented on cluster heads, where they use the data coming from their cluster members. The application uses competitive learning neural networks (CLNN), therefore we refer to it here as CLNN-Integrity (Bokareva et al., 2006). Their NN consists of eight input and eight output neurons, which are connected with weights, represented as the weight matrix W. Each row of it w_i represents a connection between all input neurons $x_0, ..., x_7$ and the one output neuron y_i. Every time an input is presented to the network, the Euclidean distances between the input and each of the outputs is calculated and the *winning* output neuron is the one with the smallest distance. The corresponding weights row w_i of the winning neuron is updated according to the following rule:

$$w_i(t+1) = w_i(t) + \lambda \times (x(t) - w_i(t)) \tag{7}$$

where λ is a constant learning rate and $w_i(t+1)$ is the updated weight vector of the winning neuron. Thus, when the network is next presented with a similar input, the probability that the same output neuron will win is higher. After the network has been trained with many input samples, it learns to differentiate between valid and false data. Of course, one of the main requirements is that during training most samples are valid. A further requirement is the intelligent initialization of the weights of the neural network. It is important that in the beginning the output neurons are spread throughout the whole possible output space. For example, the authors use light measurements, which are between 0 and 1200 units. Thus, the output neurons need to classify data into 8 different classes spread from 0 to 1200 units.

The neural network of CLNN-Integrity is deployed at dedicated cluster heads in the network. They gather data from all cluster members, use it for training the network first and then to classify data readings and to filter faulty ones. The authors have implemented the approach on a real hardware testbed consisting of 30 MicaZ motes and have tested the neural network with light measurements. The authors have simulated faulty data readings by placing paper cups on top of the light sensors of some of the nodes.



WSN Comm. Layer / ML approach	Application	Clustering	Routing	Neighborhood Management	MAC
Neural Networks	■ ***CLNN (Bokareva et al, 2006)***	□	□	▨ **SIR (Barbancho et al, 2006)** Link quality estimation	□ **NN-TDMA (Shen & Wang, 2008)** Centralized optimal TDMA scheduling
Decision Trees	▧	▧	▧	■ ***MetricMap (Wang et al, 2006)***	▨
Reinforcement Learning	▧	■ ***Clique (Förster & Murphy, 2009)***	■ ***Q-Routing (Boyan & Littman, 1994)*** **FROMS (*Förster & Murphy, 2007*)** A multicast routing protocol with flexible cost function **Q-PR (Arroyo-Valles et al, 2007)** A geographic-based unicast routing protocol	■	■ **Actor-Critic-Links (Pandana & Liu, 2005)** Point-to-point communications **RL-MAC (Liu & Elahanami, 2006)** TDMA-based MAC protocol

□ not suited ▨ less suited ▧ moderately suited ■ well suited

Fig. 6. Summary of machine learning applications to various layers of the WSN communication stack. The protocols used in this chapter as examples are emphasized.

6.2 Discussion of CLNN-Integrity

The authors of CLNN-Integrity have shown that implementing neural networks for WSNs is possible, even with online learning and on typical sensor nodes (the cluster heads, on which the CLNN was implemented, are normal sensor nodes, not special, dedicated hardware). Neural networks are very well suited for solving complex classification problems, such as recognizing faulty data readings or detecting various events based on sensor readings.

7. Conclusions and Further Reading

As demonstrated with several examples in this chapter, machine learning is a powerful tool for optimizing the performance of wireless sensor networks at all layers of the communication stack. Additional protocols and algorithms are summarized in Figure 6, where we also address the general applicability of various ML approaches to networking concerns (Kulkarni et al., 2009).

Neural networks have been successfully applied to data model learning, as in the CLNN-Integrity example described in Section 6. They are also relatively well suited for link quality estimation, since for many networks and environments the training of the neural network can be performed offline. However, neural networks are not suited for problems in distributed and fast changing environments such as at the medium access control layer. For example, (Shen & Wang, 2008) uses a NN to centrally compute the optimal TDMA schedule for a WSN. The optimality of the schedule, however, depends on the current network traffic and is thus a distributed problem, making a distributed technique such as reinforcement learning a better option. Further applications of neural networks in WSNs and their high-level descriptions can be found in (Di & Joo, 2007; Kulkarni et al., 2009).

Section 3 showed MetricMap, an application of decision tree learning to neighborhood management. This approach is well suited for nearly all layers of the communication stack due to its low memory and processing requirements and easy applicability. However, the decision tree is usually formed offline and only the rules are applied online. On the other side, this is not an issue with many classification problems, where learning samples can be easily gathered and future samples for classification are not expected to change their features. These and other benefits strongly support the investment of additional research in this direction.

Based on our survey, reinforcement learning seems to be the most widely used technique, due to its distributed nature and flexible behavior in quickly changing environments. As discussed in Section 4, Q-Routing has inspired multiple WSN routing protocols. Q-Probabilistic Routing (Arroyo-Valles et al., 2007) uses geographic progress and ETX as a cost metric for optimizing unicast routing. FROMS (Förster & Murphy, 2007) is our own multicast routing protocol, able to accommodate various cost functions, including number of hops, remaining energy at nodes, latency, etc. Additional routing protocols based on reinforcement learning, together with their properties are discussed in (Di & Joo, 2007; Kulkarni et al., 2009; Predd et al., 2006). Examples of applying reinforcement learning to medium access are available in (Liu & Elahanany, 2006; Pandana & Liu, 2005).

Another candidate for improving routing performance in WSNs is swarm intelligence. This technique, especially Ant Colony Optimization (Dorigo & Stuetzle, 2004), has been successfully applied to routing in Mobile Ad Hoc Networks (MANETs), as in AntHocNet (Di Caro et al., 2005). However, all attempts to apply it to the highly energy-restricted domain of WSNs (Kulkarni et al., 2009) have been rather unsatisfying, achieving good routes with low delay, but introducing a large amount of communication overhead for the traveling ants. One possibility to counter this communication overhead is to attach the ants to standard data packets. This will lengthen the paths taken by data packets and will increase the overall delivery delay, but at the same time will decrease total communication overhead. Further research is required to test this hypothesis.

In contrast to the widely held belief that machine learning techniques are too heavy for the resource constraints of WSN nodes, this chapter clearly demonstrates the opposite, namely that the domains of machine learning and WSNs can be effectively combined to achieve low cost solutions throughout the communication stack on wireless sensing nodes. This has been successfully shown through multiple examples, evaluated in both simulation to show scalability and in real testbeds, to concretely demonstrate feasibility.

8. References

Akyildiz, I., Su, W., Sankarasubramaniam, Y. & Cayirci, E. (2002). A survey on sensor networks, *IEEE Communications Magazine* **40**(8): 102–114.

Arroyo-Valles, R., Alaiz-Rodrigues, R., Guerrero-Curieses, A. & Cid-Suiero, J. (2007). Q-probabilistic routing in wireless sensor networks, *Proceedings of the 3rd International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, Melbourne, Australia, pp. 1–6.

Bandyopadhyay, S. & Coyle, E. (2003). An energy efficient hierarchical clustering algorithm for wireless sensor networks, *Proceedings of the Annual Joint Conference of the IEEE*

Computer and Communications Societies (INFOCOM), Vol. 3, San Francisco, CA, USA, pp. 1713 – 1723.

Barbancho, J., León, C., Molina, J. & Barbancho, A. (2006). Giving neurons to sensors: QoS management in wireless sensors networks., *in* C. Leon (ed.), *Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, Prague, Czech Republic, pp. 594–597.

Bokareva, T., Bulusu, N. & Jha, S. (2006). Learning sensor data characteristics in unknown environments., *Procedings of the 1st International Workshop on Advances in Sensor Networks (IWASN)*, San Jose, California, USA, p. 8pp.

Boyan, J. A. & Littman, M. L. (1994). Packet routing in dynamically changing networks: A reinforcement learning approach, *Advances in Neural Information Processing Systems* **6**: 671–678.

Demirbas, M., Arora, A., Mittal, V. & Kulathumani, V. (2004). Design and analysis of a fast local clustering service for wireless sensor networks, *Proceedings of the 1st International Conference on Broadband Wireless Networking (BroadNets)*, San Jose, CA, USA, pp. 700–709.

Di Caro, G., Ducatelle, F. & Gambardella, L. (2005). AntHocNet: an adaptive nature-inspired algorithm for routing in mobile ad hoc networks, *European Transactions on Telecommunications* **16**: 443–455.

Di, M. & Joo, E. (2007). A survey of machine learning in wireless sensor networks, *Proceedings of the 6th International Conference on Information, Communications and Signal Processing (ICICS)*, Singapore, pp. 1–5.

Dorigo, M. & Stuetzle, T. (2004). *Ant Colony Optimization*, MIT Press.

Förster, A. & Murphy, A. L. (2007). FROMS: Feedback routing for optimizing multiple sinks in WSN with reinforcement learning, *Proceedings 3rd International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, Melbourne, Australia, pp. 371–376.

Förster, A. & Murphy, A. L. (2009). CLIQUE: Role-Free Clustering with Q-Learning for Wireless Sensor Networks, *Proceedings of the 29th International Conference on Distributed Computing Systems (ICDCS)*, Montreal, Canada.

Karl, H. & Willig, A. (2005). *Protocols and Architectures for Wireless Sensor Networks*, John Wiley & Sons.

Karp, B. & Kung, H. T. (2000). GPSR: greedy perimeter stateless routing for wireless networks, *Proceedings of the 6th annual international conference on Mobile computing and networking (MobiCom)*, Boston, MA, USA, pp. 243–254.

Kulkarni, S., Förster, A. & Venayagamoorthy, G. (2009). A survey on applications of computational intelligence for wireless sensor networks, *under review* .

Liu, Z. & Elahanany, I. (2006). RL-MAC: A reinforcement learning based MAC protocol for wireless sensor networks, *International Journal on Sensor Networks* **1**(3/4): 117–124.

Mitchell, T. (1997). *Machine Learning*, McGraw-Hill.

Pandana, C. & Liu, K. J. R. (2005). Near-optimal reinforcement learning framework for energy-aware sensor communications, *IEEE Journal on Selected Areas in Communications* **23**(4): 788–797.

Predd, J., Kulkarni, S. & Poor, H. (2006). Distributed learning in wireless sensor networks, *IEEE Signal Processing Magazine* **23**(4): 56–69.

Puccinelli, D. & Haenggi, M. (2008). Arbutus: Network-layer load balancing for wireless sensor networks, *Proceedings of the IEEE International Conference on WWireless Communications and Networking Conference (WCNC)*, pp. 2063–2068.

Rabiner-Heinzelman, W., Chandrakasan, A. & Balakrishnan, H. (2000). Energy-efficient communication protocol for wireless microsensor networks, *Proceedings of the 33rd Hawaii International Conference on System Sciences (HICSS)*, Hawaii, USA, p. 10pp.

Römer, K. & Mattern, F. (2004). The design space of wireless sensor networks, *IEEE Transactions on wireless communications* **11**(6): 54–61.

Shen, Y. J. & Wang, M. S. (2008). Broadcast scheduling in wireless sensor networks using fuzzy hopfield neural network, *Expert Systems with Applications* **34**(2): 900–907.

Silva, F., Heidemann, J., Govindan, R. & Estrin, D. (2003). *Frontiers in Distributed Sensor Networks*, CRC Press, Inc., chapter Directed Diffusion, p. 25pp.

Sutton, R. S. & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*, The MIT Press.

Wang, Y., Martonosi, M. & Peh, L.-S. (2006). A supervised learning approach for routing optimizations in wireless sensor networks, *Proceedings of the 2nd International Workshop on Multi-hop ad hoc networks: from theory to reality (REALMAN)*, Florence, Italy, pp. 79–86.

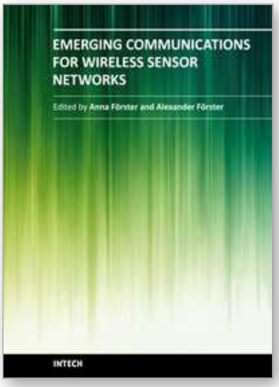
Watkins, C. (1989). *Learning from Delayed Rewards*, PhD thesis, Cambridge University, Cambridge, England.

Witten, I. & Frank, E. (2005). *Data Mining: Practical machine learning tools and techniques*, 2nd. edn, Morgan Kaufmann.

Woo, A., Tong, T. & Culler, D. (2003). Taming the underlying challenges of reliable multihop routing in sensor networks, *Proceedings of the 1st international conference on Embedded networked sensor systems (SenSys)*, Los Angeles, CA, USA, pp. 14–27.

Wu, Q., Rao, N., Barhen, J., Iyengar, S., Vaishnavi, V., Qi, H. & Chakrabarty, K. (2004). On computing mobile agent routes for data fusion in distributed sensor networks, *IEEE Transactions of Knowledge Data Engineering* **16**(6): 740–753.

Younis, O. & Fahmy, S. (2004). HEED: a hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks, *IEEE Transactions on Mobile Computing* **3**(4): 366–379.


EMERGING COMMUNICATIONS FOR WIRELESS SENSOR NETWORKS
Edited by Anna Förster and Alexander Förster
INTECH

Emerging Communications for Wireless Sensor Networks
Edited by

ISBN 978-953-307-082-7
Hard cover, 270 pages
Publisher InTech
Published online 07, February, 2011
Published in print edition February, 2011

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821