Journal of Scientific & Industrial Research
Vol. 80, February 2021, pp. 149-158

Realization of Autonomous Sensor Networks with AI based Self-reconfiguration and Optimal Data Transmission Algorithms in Resource Constrained Nodes

Syed Ameer Abbas* and Abirami

Department of Electronics and Communication Engineering, Mepco Schlenk Engineering College, Sivakasi, Tamilnadu, India

Received 15 July 2020; revised 18 October 2020; accepted 01 December 2020

Wireless sensor networks (WSN) prove to be an enabling technology for Industry 4.0 for their ability to perform in autonomous manner even in regions of extreme conditions. Autonomy brings in independent decision making and exerting controls without manual intervention and frequent maintenance. This paper aims to inculcate intelligence to the WSN exploiting the merits of Artificial Intelligence (AI) algorithms in cheap and most preferred ESP8266 and ESP32 based nodes. Autonomy is brought in by means of optimal data transmission, compressive sensing fault detection and network reconfiguration and energy efficiency. Optimal data transmission is achieved using Q-learning based exploration exploitation algorithm. Compressive sensing performed using Autoencoders ensure reduction in transmission overhead. Fault detection is done using Binary SVM classifier and the network re-configures based on physical redundancy. This paper highlights the implementation of such autonomous WSN in real time along with their performance statistics.

Keywords: Autoencoders, Cloud visualization, Compressive sensing, Reinforcement learning, Sensor fault detection

Introduction

Wireless Sensor Networks (WSNs) are found to be the robust technology in ensuring non internet enabled devices like sensors, actuators etc., to gain connectivity. The development of WSNs was inspired by military applications in 1980s in the form of DSNs. Though the idea of WSN was proposed, implementation took far more years due to technical setback during those decades. Advancements in information technology and MEMS have resulted in miniaturization of both sensors and reduction in their price making them economical. The data transmission is made as multi-hop communication to overcome the node's limited range of communication. Multi-hop communication of networks is enabled by Ad-hoc network architecture in which the APs are continuously changing in order to meet the communication needs. However, intelligent firmware is necessary to coordinate and synchronize the network. Such coordination can be achieved with the help of robust and accurate Machine Learning Algorithms. Despite being economical and reliable the major factors hindering the adoption of sensor networks for industrial applications include - requirement of more energy, frequent maintenance, resource constraints and huge cost for advanced nodes.

Hence, an approach to bring in autonomy on resource constraint economical nodes by enabling AI algorithms is discussed here. Further, the nodes are programmed to undergo self-configuration after initial setup. Communication is established following WiFi (IEEE 802.11). Algorithms to achieve optimal data transmission, self-reconfiguration and compressive sensing are discussed.

Related Work

The nodes of this architecture orchestrate forming a single network yet performing their specific functionalities. Accordingly, the memory requirement differs for every node based on their responsibility. Every AI algorithm has unique characteristics and are appropriate for specific applications. Strategies for implementing AI algorithms in a distributed manner at various levels of WSN would enhance the network's autonomy.[1] However, the paper provides a generalized overview and does not discuss the hardware implementation impacts of such algorithms. Apart from these AI techniques, complex algorithms of Computational Intelligence also contribute to bring in intelligence to the sensing network. Most of the CI techniques are nature inspired algorithms and more robust in terms of ensuring QoS, lossless data gathering applications. The WSN's QoS can be ensured by adopting optimal queuing technique.[2] The common CI algorithms including PSO, fuzzy and

*Author for Correspondence
E-mail: ssyed@mepcoeng.ac.in

neural networks suitable for implementation in WSN. But, implementation of these algorithms at hardware level would require highly resourceful nodes and may not be economical.

WSN usually operates on battery source, energy consumption has to be reduced. The reinforcement learning algorithm can be used to achieve energy efficiency using Q-Learning technique.[3] The main issue hindering large scale deployment of WSN is hardware maintenance and frequent inspection. To rule out such shortcomes, self-diagnosis and reconfiguration mechanisms are introduced ensuring autonomy in operation. A supervised learning algorithm is more effective to detect sensor faults using SVM classifier. [4]

Sensor fault detection techniques using time domain features are ideal for real time sensing environments.[5] Other techniques of fault detection like support vector regression brings in computational complexity.[6] Data aggregation procedure is the key ability of a WSN in processing the data. Conventional data aggregation techniques do not inculcate intelligence to the network.[7] Common compressive sensing using transform functions and PCA algorithms are too heavy for a micron device to handle.[8] In this requirement an alternative approach using Autoencoders in big data compression would be an ideal choice.[9]

Proposed Methodology

WSNs are mostly preferred for their, portability and wireless communication support. They are deployed in enormous number covering a wide region to be monitored.

Network Model

The nodes deployed depend on batteries for their operation despite of which they must sustain in sensing environments for longer time. This requirement is met by adopting cluster based hierarchical routing approach. In cluster based hierarchical routing the energy consumption by each node is minimized and a resourceful node is made as cluster head (CH). In this work the proposed WSN is implemented in cluster tree architecture as shown in Fig. 1.

Single hop communication between the sensing nodes and router is not suitable for following reasons — 1) The distant node may die out before data is successfully transmitted to the sink, 2) Energy consumption would increase drastically, 3) Data redundancy will be higher at the sink. Hence, multihop communication is implemented using cluster tree topology which offers following advantages 1) Scalability, 2) Predictable time delay due to hierarchical architecture, 3) Control on nodes become distributed to cluster heads and sink not overloading nodes.

Node Architecture

The nodes used are Nodemcu and Wemos D1 R1. Both have common elements except that Wemos D1 R1 is designed follows Arduino layout. The ESP8266 module can operates at 3.3 V and any high voltages may damage the IC. The version of ESP used is ESP–12E ESP8266. It operates in ISM 2.4 GHz band and supports IEEE 802.11b/g/n. The channel in which WiFi transmission and reception occurs can be configured through software while the default channel used before configuration is channel 1.

The three main subsystems of the wireless nodes are power subsystem, processing unit and communication unit. The architecture of the node is depicted in Fig. 2. The power subsystem comprises of a Battery and LDO voltage regulator.

Rechargeable batteries are preferred generally. The processing unit comprises of the microcontroller, memory and other supporting peripheral interfaces. The sensing unit consists of the sensor and an optional ADC connected to the processing unit via interface. Communication subsystem comprises of hardware necessary for wireless data transfer including antenna. Both processing and communication units are dependent on firmware for their operation. The Core module to define access methods of the memory, TCP/IP stack for IEEE 802.11 b/g/n standard and other link layer protocols are dumped in OTP memory.

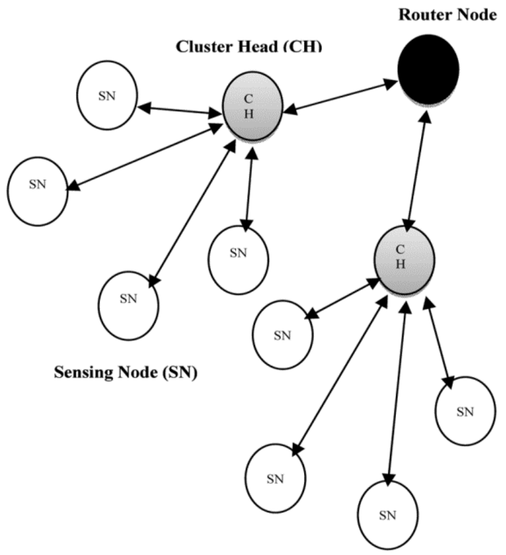


Fig. 1 — Cluster Tree Topology of the WSN with three levels

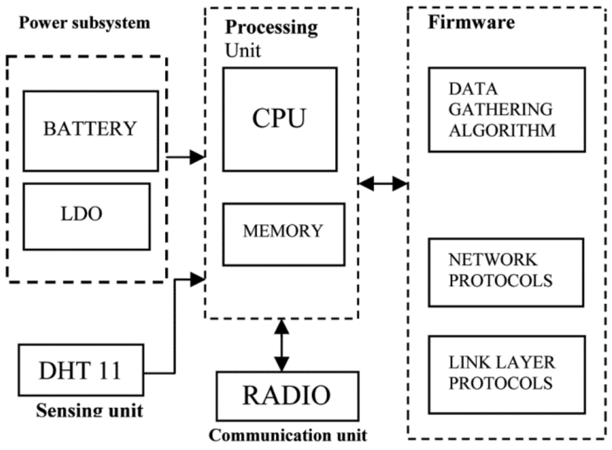


Fig. 2 — Architecture of a wireless node and sub blocks

Optimal Data Transmission

After the deployment of WSN, efficient data transfer is achieved ensuring the flow of non-redundant data alone in the network. The redundancy in data can be estimated by observing the frequency of the redundant data appearance i.e., correlation. The correlation may be temporal or spatial in nature. The temporal correlation estimates the redundancy occurring in data considering a specific node while spatial correlation estimates the redundancy by taking into account the data gathered by a number of nodes scattered in same region of the sensing environment. However, the spatial correlation becomes irrelevant when deployed in industrial environments for the region of sensing and the range of variation of the parameters vary to a greater extent. Hence for the network to suit ubiquitous applications, the optimization is brought in exploiting only the temporal correlation.

Temporal Association

Redundant data denotes duplicate or repeated data without which the physical parameters sensed can be extracted without losing any information. Our aim is to estimate the redundant data by finding the frequency of recurrence of data and make our node learn to sense intelligently in turn reducing energy consumption. Jaccard coefficient is computed to estimate the magnitude of correlation among the available data.[7]

Consider a scenario in which M sensor nodes are scattered across the region to be sensed. A sensor node, Si in the network senses the environment for a duration 't' and takes '2n' samples of data - S1, S2, S3, ..., S2n. These '2n' samples can be grouped into two sets A = S1, S2, S3, ..., Sn and B = Sn+1, Sn+2, Sn+3, ..., S2n. Now, the Jaccard correlation co-efficient can be computed as in Eq. 1.

$$J(A,B) = (A \cap B)/(AUB) \quad \dots (1)$$

where, the value of J varies between (0,1). J value of 0 indicates no redundant data while entire data would be redundant if J value is 1. Based on the extent of correlation among the collected data the data is classified as follows.

Based on the extent of similarity, the Jaccard coefficient value varies between 0 and 1 with 0 indicating no redundancy and 1 indicating complete redundancy.

Q – Learning Based Data Redundancy Reduction

From previous sections it is evident that the number of packets transmitted can be decreased by eliminating temporally correlated data which decreases packet overhead and increases energy optimization. Data mining techniques are widely adopted to implement in WSN for the lightweight algorithms and requirement of minimum data set for processing. One such approach is the reinforcement learning or reward based learning process where the agent learns to act based on the rewards it receives for its actions. Q- Learning stands for Quality learning where the agent learns about the trends of data being sensed and estimates the environmental changes based on this observation. This type of regret learning can occur with minimal dataset which makes this approach suitable for resource constrained WSN nodes. Exploration exploitation is a technique where the learner is allowed to act randomly to certain extent. This short-term randomness is introduced to discover all possible actions that could optimize the solution. Such short term sacrifices are included to achieve high accuracy in the long run.

The elements of Q-Learning are available states, possible actions, reward for actions and an environment to be sensed. The three states considered here are 1) High correlation, 2) Medium correlation and 3) Low correlation as described in previous subsection. The actions to be implemented as the output of algorithm are 1) Transmit entire data, 2) Transmit non- redundant data only and 3) Do-not transmit. The node enters sleep state during action3 i.e., neither sensing nor communication occurs. This mode preserves maximum energy and is quintessential in extending the lifetime of networks built using micron nodes. The primary duty of the agent is to update the Q – matrix which is initialized as all zeros during training phase. It is based on this matrix; the agent decides the action to be implemented.

The exploration and exploitation is implemented using ε–greedy algorithm.[10] This algorithm explores forever finding all possible actions during implementation.[11]

By following – greedy approach, the suitable action is predicted using Eq. 2.

$$\text{Current\_action} = \text{Argmax}\ (Q\ (S, A)) \qquad \ldots (2)$$

The above function chooses the action with maximum Q value, i.e., the action with maximum reward. The Q value after implementation of current_action is updated by the learner using Bellman's equation in Eq. 3.

$$Q(S,A) = Q(S,A) + \alpha\left(R + \gamma \times \left(Q(new:)\right)\right) - Q(S,A) \qquad \ldots (3)$$

where,

α – Learning rate s (0,1)

γ – Discount factor s (0,1)

R – Reward corresponding to the action.

The deep sleep after every action extends the life of the micron devices by reducing the heat exposure.[7] The adaptive sleep cycles make the networks more suitable for dynamic environments.[12,13]

Algorithm: Q Learning based data redundancy reduction

Inputs: Jaccard co-efficient (J), Reward matrix
Outputs: Current action, Sleep duration

```
Train the learner and obtain Q-Matrix
1.  Divide the sensed data into two arrays.
2.  Compute the Jaccard coefficient - J for the two
    arrays.
3.  Estimate current state based on J value.
4.  Initialize randomness factor $\varepsilon$
5.  If ( random(0,1) $\geq 1 - \varepsilon$ )
Current action = $Q(S,A)$
6.  Else Current action = random (states)
7.  Update Q matrix
8.  Diff = $Q_{new} - Q_{old}$
9.  If (Reward $\geq 0$) $\Theta_{new} = \Theta_{old} + W \times Diff$
10. Else $\Theta_{new} = \Theta_{old} - W \times Diff$
11. End If
```

Sensor Fault Detection

In general, WSN deployed in unapproachable regions are prone to failures. The sensor failure may occur due to software, hardware or communication faults. Communication failures occurrences are usually rare. Detecting hardware failure is proposed here using SVM classifier. Five types of faults are simulated for injection. SVM has inner product kernels with ability to classify linearly non-separable cases by mapping to higher dimensions. Further, SVM classifiers allow generalizations to overcome the issue of overfitting.

Dataset Preparation

SVM classifiers are supervised learning algorithms. Hence, labelled dataset has to be prepared to train the fault classifier as in Fig. 3. Fault readings are generated from normal dataset t, at any instant i, using following expressions:

Drift Fault:Readings monotonously either increase or decrease with time deviating from actual reading.
$f(i) = t(i) \pm K$
where K is monotonously increasing constant.

Spike Fault: Spikes of high value shoot up at random time instants in sensed data.
$f(i) = M$
where, M is spike value and i is the position where spike occurs.

Hardover Fault: The values sensed goes beyond bounds of actual values corresponding to the environment being sensed.
$f(i)=\alpha$
where, α is a value out of bounds.

Erratic Fault: Addition of Gaussian noise to actual data sensed from the environment.
$f(i) = \text{awgn}(t(i))$
where, the function awgn adds Gaussian noise to t(i).

Stuck-at Fault: The sensor output remains in same value throughout the duration.
$f(i) = t(n)$
where, n is the last reading at which the value is stuck.

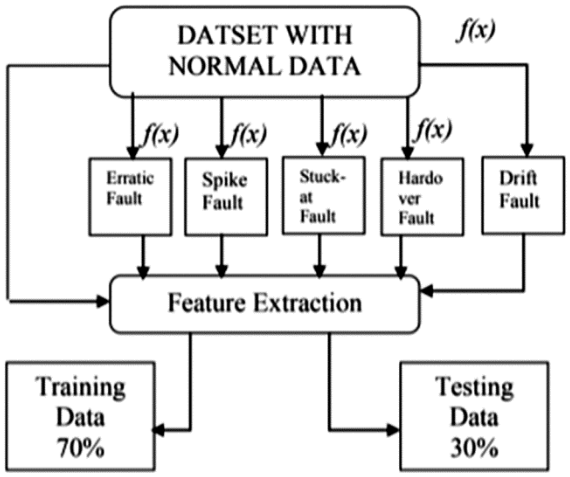


Fig. 3 — Dataset preparation procedure to train fault classifier

In above expressions, f(i) refers to fault signal and t(i) refers to original signal at time instant 'i'.

Feature Extraction

The feature extraction is done to filter the behavioral patterns from the raw data. The time domain features computed are root mean square, peak-to-peak value, crest factor, root amplitude square, kurtosis, skewness, standard deviation, impulse factor, margin factor and shape factor. Feature selection is performed to identify the significant features and eliminate the rest. The significant features selected for training are root mean square, peak-to-peak value, crest factor impulse factor, margin factor and shape factor.

SVM Classifier

Support Vector Machine (SVM) is a supervised machine learning algorithm in which each point is represented in n-dimensional space with value of each feature being the value of particular coordinate appropriate for real time sensing techniques.[14] In training phase, an appropriate hyperplane to differentiate the two classes very well is formed. The classifier then finds the minimum distance of the frontier from closest support vector. In generalized form, the equation of decision vector is given in Eq. 4.

$$W^T x + b = 0 \qquad \dots (4)$$

where,

W – weight vector determined during training phase

b – bias and

x – the input vector to be classified.

In case of binary classification, the input vector belongs to positive class if $W^T x + b > 0$ and negative class if $W^T x + b < 0$. To solve problems that ae not linearly separable, the points are transformed to higher dimension to convert them into linearly separable problem. The SVM finds the optimal hyperplane satisfying Eq. 5.

$$Min\big(\emptyset(W)\big) = 0.5(W^T W) \qquad \dots (5)$$

The decision boundary is formed using Eq. 6.

$$\sum_{i=1}^{N} \alpha_i K(x, x_i) y_i + b \qquad \dots (6)$$

where,

K – Kernel function

α_i – Label

b – Bias

The kernel function used in this work is quadratic function of degree 2, a non-linear transform function to minimize classification errors as in Eq. 7.

$$K(x, x_i) = (1 + xx^T)^2 \qquad \dots (7)$$

SVM are suitable classifiers for their accuracy in high dimensional spaces, clear margin of separation and efficiency obtained from simple support vectors. Also, the classifier can be implemented with trained confusion matrix even in resource constrained nodes and effective for time series calculation.

Compressive Sensing

The Compressive sensing theory states that it is possible to reconstruct the original data from fewer samples than actual number of samples being transmitted thereby increasing energy efficiency and optimal resource utilization. Compressive sensing is usually achieved using either of two approaches – 1. Sending only the essential components containing maximum information relating to the data, 2. Reducing the count of data by using transform function. The widely implemented method for compressive sensing is PCA that follows former approach but, suffers from a serious disadvantage of poor performance on non-linear data and requirement for resourceful nodes. Whereas, the Autoencodesrs following latter approach overcomes these setbacks.

Autoencoder

Autoencoders are special types of neural networks following unsupervised learning technique. Unlike other Neural Networks Autoencoders map the input domain back to input domain. A typical Autoencoder comprises three layers, encoder, code and decoder (Fig. 4). The encoder block performs transform operation on input data and generates code which when operated using decoder block reconstructs original data. In most of the cases, the decoder is inverse transform of encoder transform function. In this work, the encoder block is implemented at sink node and code generated is transmitted to the cloud where reconstruction using decoder block occurs. Autoencoders are unsupervised and entirely data specific. Models trained on particular type of data can used for compression of similar data only. However, they are not completely error free and hence an error minimization function added inherently.

A simple Autoencoder containing only one hidden layer would suit our requirement on a resource constrained node. The transform function used is

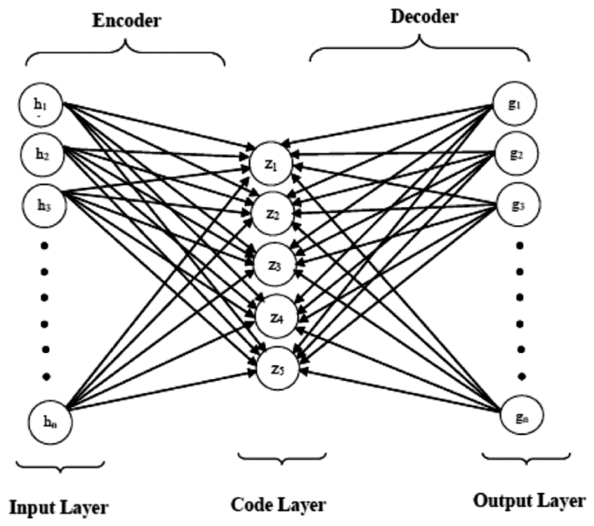


Fig. 4 — Architecture of fully connected Autoencoder network

logistic sigmoid function mentioned in Eq. 8. The loss function used is mean squared error with sparse regularization given by Eq. 9. The encoder (recognizer) block is concerned with recognizing the pattern hidden in input sequence and the decoder (reconstructor) block reconstructs the original sequence based on learned parameters.

The input and output blocks contain same number of neurons while the hidden layer neurons decrease to code layer on encoder block and increases to output layer at decoder block. The Autoencoder implemented is fully connected.

$$f(x) = \frac{1}{1+e^{-z}} \quad \ldots (8)$$

Where,

x – code value

z – input values

$$E = \frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K}(x_{kn} - \hat{x}_{kn})^2 + \lambda\Omega_w + \beta\Omega_s \quad \ldots(9)$$

Sparse regularization is done to lay constraint on output values so that the outputs do not deviate too much from the input values. Sparsity is the inherent characteristic of Autoencoders and is adjusted to required extent using sparsity function and L2 regularization function to fine-tune sparsity regulator ate represented as Ω_s and Ω_w respectively in Eq. 9.

Results and Discussion

The experiment is carried out by distributing 8 sensing nodes across laboratory environment along with two cluster heads and one sink node. AI learners are trained using MATLAB Machine Learning toolbox, algorithms are dumped into nodes using Arduino IDE and ThingSpeak cloud is used for final visualization.

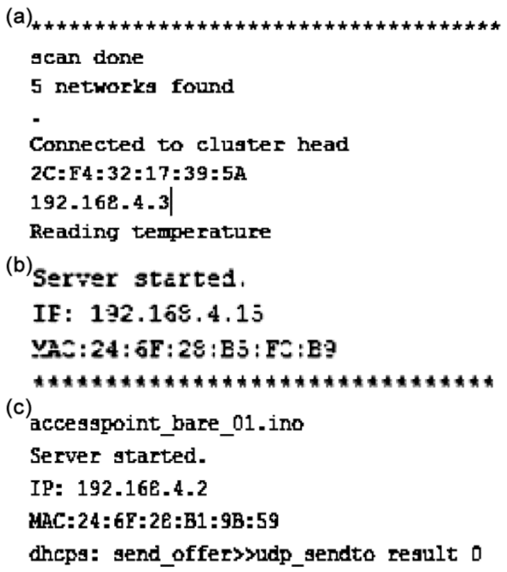

```
(a)************************************
   scan done
   5 networks found
   .
   Connected to cluster head
   2C:F4:32:17:39:5A
   192.168.4.3
   Reading temperature
(b)Server started.
   IP: 192.168.4.15
   MAC:24:6F:28:B5:FC:B9
   *********************************
(c)accesspoint_bare_01.ino
   Server started.
   IP: 192.168.4.2
   MAC:24:6F:28:B1:9B:59
   dhcps: send_offer>>udp_sendto result 0
```

Fig. 5 — Initialization at (a) Sensing node, (b) Cluster head and (c) Sink node

Network Configuration

The communication among the nodes is established using IEEE 802.11 standard and TCP/IP protocol which is connection based.[15] Initialization at Sensing node, Cluster head and Sink node are shown in Fig. 5.

The sensing nodes always function as Station Points (STA) i.e., connect to Access Points (AP) and relay on them to transmit and receive data. The cluster heads function as both Station and Access points during data transmission and switch between the modes based on requirement. Cluster Heads are AP for sensing nodes and STA for sink node whereas; the sink node is an AP for cluster head and STA for local router while transmitting data to the cloud. The sink and cluster heads are assigned fixed IP and the cluster heads allocate IP for the sensing nodes using DHCP protocol.

Optimal Data Transmission

MATLAB simulation result for optimal data transmission algorithm is shown in Fig. 6. The term optimal data transmission in this scenario implies ensuring non redundant data transmission in the network. The redundancy in data is eliminated at the initial stage itself i.e., at sensing nodes. Apart from eliminating redundant data, the algorithm also

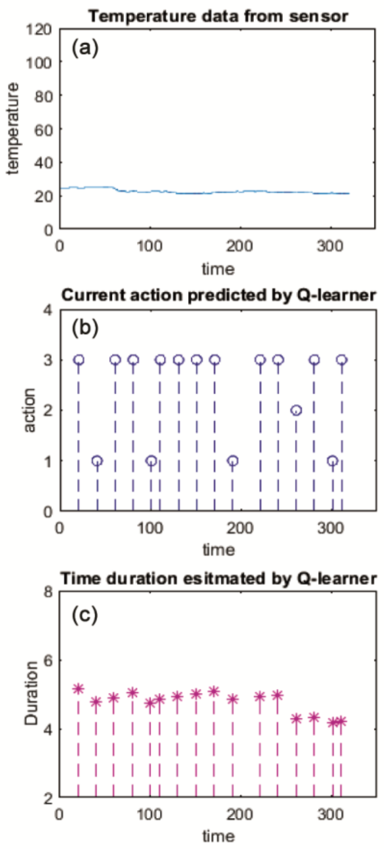


Fig. 6 — MATLAB simulation result for optimal data transmission algorithm

chooses the next state for the node to operate and the duration for node sleep. The reward matrix is initialized with weights corresponding to the state and action based on desired operation. Q-matrix initially declared as null is updated during the number of iterations in training phase. In testing phase, real time data are considered and actions are implemented based on the Bellman's equation.

The values of alpha, gamma and epsilon are set as 0.3, 0. 9 and 0.6, respectively. The data transmissions on various actions and the sleep duration estimated using Eq. 10.

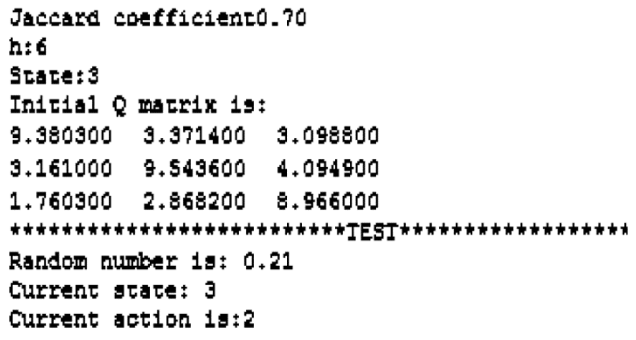
$$\boldsymbol{\theta_{new} = \theta_{old} \pm W \times Diff} \qquad \dots (10)$$

where, Diff – Change in Q value

W – Constant

Θ – Sleep Duration

For action = 1, the entire sensed data is transmitted to the cluster head and for action = 2, only the non-redundant data is transmitted while no transmission



```
Jaccard coefficient0.70
h:6
State:3
Initial Q matrix is:
9.380300   3.371400   3.098800
3.161000   9.543600   4.094900
1.760300   2.868200   8.966000
*************************TEST*****************
Random number is: 0.21
Current state: 3
Current action is:2
```

Fig. 7 — Implementation of optimal data transmission algorithm at sensing node

occurs for action = 3. Initially 20 data gathered are grouped into two arrays of 10 elements each. Then redundant data are eliminated by computing their union. Q-Learner then predicts an appropriate state and action. The sleep duration estimated is also displayed in the serial monitor.

The dataset used for training is Intel Labdata available at http://db.csail.mit.edu/labdata/labdata.html. It contains nearly 1100000 readings collected using 54 sensor nodes deployed in the Intel Berkeley Research lab between February 28th and April 5th, 2004 validated by taking multiple readings for particular region from different nodes. Of these nearly 1000 data taken from a single node monitoring a specific region is used to train the Q-Learner implemented in the sensing nodes. Training is done using MATLAB and the final trained Q-matrix is used to implement the algorithm at node level. Then, 300 data of same node are used to test the algorithm output. The behavior of the proposed algorithm in realtime is displayed in Fig. 7.

Sensor Fault Detection

Sensor fault detection is done using SVM classifier. The first step in the process is preparation of dataset. The source dataset used is Intel Labdata same as used for previous algorithm. The original dataset contains nearly 10 lakh data of which 42000 data bound to the range 19–25ºC is considered. These data are grouped into 42 samples with 1000 data per sample. Faults are generated using 5 such samples. Five types of faults are generated as discussed in previous section. The pattern of fault occurrence is varied as in 25%, 50 %, 75% and 100% of source data. One such pattern generated is displayed in Fig. 8. The dataset containing 42 normal and 25 faulty samples is used for training.

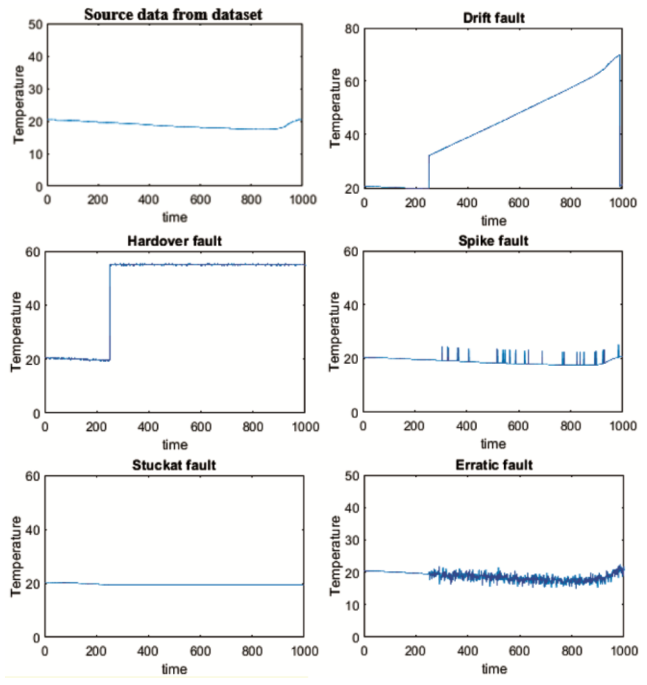
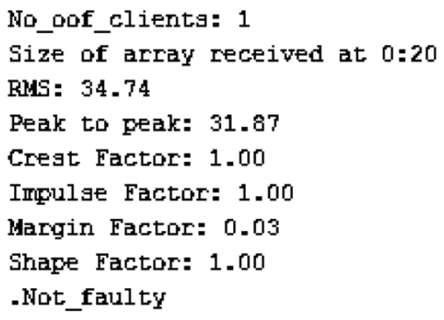


Fig. 8 — Faults generation using MATLAB



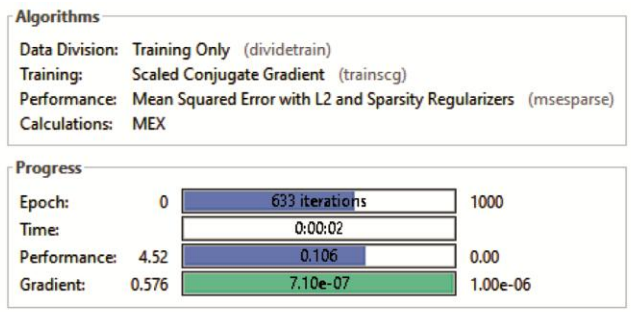
```
No_oof_clients: 1
Size of array received at 0:20
RMS: 34.74
Peak to peak: 31.87
Crest Factor: 1.00
Impulse Factor: 1.00
Margin Factor: 0.03
Shape Factor: 1.00
.Not_faulty
```

Fig. 9 — Implementation of sensor fault detection algorithm at cluster head

Multiclass SVM is used for classifying the sensor state adopting one-vs-all approach. Accuracy is the main parameter determining the performance of the classifier. Mathematically, accuracy is the ratio of correct predictions to total predictions given by Eq. 11

$$A = \frac{TP+TN}{TP+FP+TN+FN} \quad \dots (11)$$

Further, the learner after implementation in node predicts the state of the sensor. The results predicted by the learner on processing real time sensor data relayed is displayed in Fig. 9. This algorithm is implemented at level 2 in Cluster Heads. The data is checked for fault only when entire data is being transmitted. Features are extracted for the received data and then using decision function of the learner prediction is done based on the sign of the output.



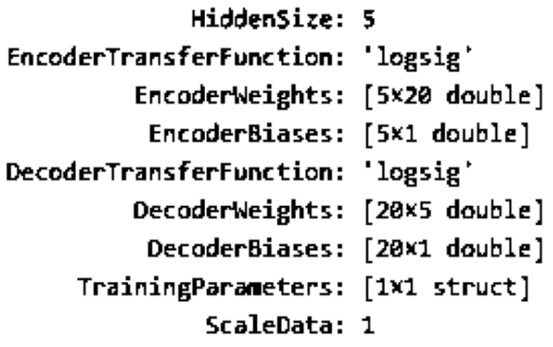
Algorithms

Data Division:	Training Only (dividetrain)
Training:	Scaled Conjugate Gradient (trainscg)
Performance:	Mean Squared Error with L2 and Sparsity Regularizers (msesparse)
Calculations:	MEX

Progress

Epoch:	0	633 iterations	1000
Time:		0:00:02	
Performance:	4.52	0.106	0.00
Gradient:	0.576	7.10e-07	1.00e-06

Fig. 10 —Autoencoder training parameters



```
            HiddenSize: 5
EncoderTransferFunction: 'logsig'
        EncoderWeights: [5×20 double]
         EncoderBiases: [5×1 double]
DecoderTransferFunction: 'logsig'
        DecoderWeights: [20×5 double]
         DecoderBiases: [20×1 double]
    TrainingParameters: [1×1 struct]
             ScaleData: 1
```

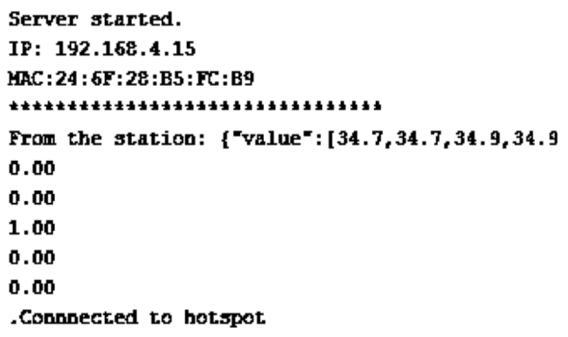
Fig. 11 — Autoencoders properties

When the data is classified as faulty, message is sent to corresponding node and the node switches to the alternate sensor connected the node i.e., network reconfiguration achieved at the cost of physical redundancy.

Compressive Sensing

Once the Sink receive entire data sensed, it compresses the data to 5 values and sends to the cloud where decoder part of Autoencoder is present and the data is retrieved back however with some error. The Autoencoder is trained in MATLAB using Neural Network toolbox. The training data is generated using MATLAB random function by generating data within the bounds within which the physical parameter sensed would vary in the environment being sensed in the order as that of data received from the cluster head. The Autoencoder training parameter using MATLAB is displayed in Fig. 10 and its properties in Fig. 11.

The model trained using MATLAB is implemented at sink node which is responsible for relaying the received data directly to the cloud. The encoder weights and bias values are copied from MATLAB and using the encoder function i.e., Logistic sigmoid function, the data are encoded. The results of such compression are displayed in Fig. 12.



```
Server started.
IP: 192.168.4.15
MAC:24:6F:28:B5:FC:B9
********************************
From the station: {"value":[34.7,34.7,34.9,34.9
0.00
0.00
1.00
0.00
0.00
.Connected to hotspot
```

Fig. 12 — Implementation of compressive sensing at sink node

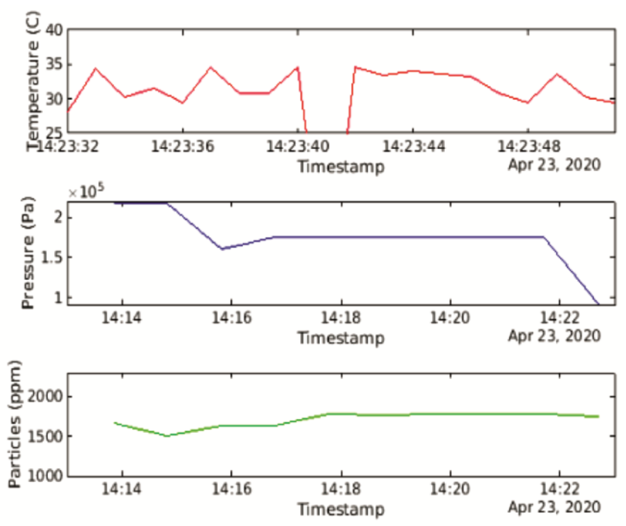


Fig. 13 — Visualization of sensed data at cloud

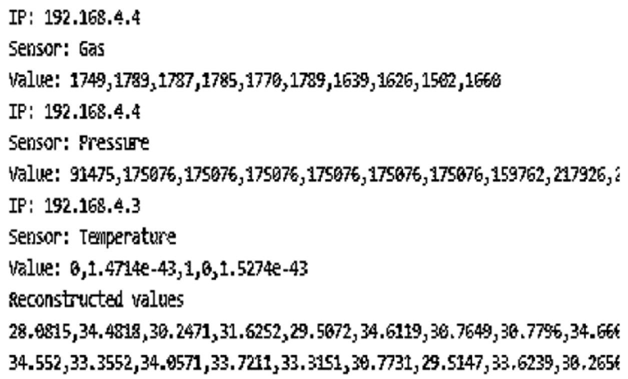
Visualization at Cloud

The compressed data is sent to the cloud i.e., ThingSpeak platform where the decoder block of Autoencoders are present. The Autoencoder is trained using same data as in offline MATLAB and data is recovered using inbuilt decode function.

Then, the data is plotted using MATLAB visualizer app in ThingSpeak cloud. Sensors considered for this implementation are DHT11 (Temperature), BMP280 (Pressure) and MQ-2 (Methane gas). The data plotted at the cloud is displayed in Fig. 13.

The decoded JSON at the cloud and the temperature received as code is shown in Fig. 14. Data is reconstructed at cloud using Autoencoder's decoder block and displayed.

In summary, the proposed work brings in autonomy to the sensor networks in optimal data transmission, self-reconfiguration and compressive sensing with the aid of multiple AI algorithms implemented at various levels of hierarchical architecture even in resource constrained micron nodes.



```
IP: 192.168.4.4
Sensor: Gas
Value: 1749,1789,1787,1785,1770,1789,1639,1626,1502,1660
IP: 192.168.4.4
Sensor: Pressure
Value: 91475,175076,175076,175076,175076,175076,175076,159762,217926,
IP: 192.168.4.3
Sensor: Temperature
Value: 0,1.4714e-43,1,0,1.5274e-43
Reconstructed values
28.0815,34.4818,30.2471,31.6252,29.5072,34.6119,30.7649,30.7796,34.66
34.552,33.3552,34.0571,33.7211,33.3151,30.7731,29.5147,33.6239,30.265
```

Fig. 14 — Data reconstruction from code at cloud

Conclusions

In this work, a robust autonomous WSN is implemented using inexpensive ESP8266 and ESP32 based nodes with light weight algorithms is implemented in cluster tree architecture having three hierarchical levels to improve scalability and energy efficiency. Optimal data transmission achieved by the Q-Learning algorithm implemented in level 1, fault detection by SVM classifier at level 2 and compressive sensing at level 3 ensures transmission overhead reduction apart from bringing in innate intelligence to the WSN and autonomy in functioning. The WSN implemented in this work is homogeneous in nature. The incorporation of AI algorithms to control the operation of WSN ensures self-reliant operation of the network to a greater extent. These networks are suitable for implementation at industries, forest monitoring, pollution monitoring etc., The WSNs can be enhanced by introducing OTA communication which would empower in increasing the capacity of hardware without increasing the physical hardware complexity.

Future Work

OTA is an emerging technique which would empower in increasing the capacity of hardware without increasing the physical hardware complexity. The micron nodes like ESP8266 and ESP32 also support OTA communication which can be exploited to enable firmware update provision thus supporting periodic evolution of the network.

References

1 Alsheikh M A, Lin S, Niyato D & Tan H-P, Machine learning in wireless sensor networks: algorithms, strategies, and Applications, *IEEE Commun Surv Tutor*, **(2)** (2015) 1996–2018.

2 Byun H & Yu J, Adaptive duty cycle control with queue management in wireless sensor networks, *IEEE Trans Mobile Comput*, **12(6)** (2013) 1214–1225.

3 Das S N, Misra S, Wolfinger B E & Obaidat M S, Temporal-correlation-aware dynamic self-management of wireless sensor networks, *IEEE Trans Industr Inform*, **12(6)** (2016) 2127–2138.

4 Zidi S, Moulahi T & Alaya B, Fault detection in wireless sensor networks through svm classifier, *IEEE Sens J*, **18(1)** (2018) 340–347.

5 Jan S U, Lee Y-D, Shin J & Koo I, Sensor fault classification based on support vector machine and statistical time-domain features, *IEEE Access*, **(5)** (2017) 8682–8690.

6 Cheng Y, Liu Q, Wang J, Wan S & Umer T, Distributed fault detection for wireless sensor networks based on support vector regression, *Wirel Commun Mob Comput*, (2018) 1–8.

7 Harb H, Makhoul A, Tawbi S & Couturier R, Comparison of different data aggregation techniques in distributed sensor networks, *IEEE Access – Special Section on heterogeneous crowdsourced data analytics*, **(5)** (2017) 4250–4263.

8 Kong L, Zhang D, He Z, Xiang Q, Wan J & Tao M, Embracing big data with compressive sensing: a green approach in industrial wireless networks, *IEEE Commun Mag*, **54(10)** (2016) 53–59.

9 Sirshar M, Saleem S, Ilyas M U, Khan M M, Alkatheiri M S & Alowibdi J S, Big data dimensionality reduction for wireless sensor networks using stacked autoencoders, in *Research & Innovation Forum 2019* edited by A Visvizi, M Lytras, RIIFORUM 2019, Springer Proceedings in Complexity, Springer, Cham, 391–400, https://doi.org/10.1007/978-3-030-30809-4_35

10 Sutton R S & Barto A G, in *Reinforcement Learning: An Introduction* (MIT Press, USA) 143 – 166.

11 Silver D, UCL course on reinforcement learning, UCL London Global University, Available: http://www.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html.

12 Abdel Salam H S & Olariu S, Toward adaptive sleep schedules for balancing energy consumption in wireless sensor networks, *IEEE Trans Comput*, **61(10)** (2012) 1443–1458.

13 Savaglio C,Pace P, Aloi G, Liotta A & Forting G, Lightweight reinforcement learning for energy efficient communications in wireless sensor networks, *IEEE Access*, **(7)** (2019) 29355–29364.

14 Ahmed M B & Ambhaikar A, Wireless sensor networks: techniques for detecting faults using artificial intelligence, *IJRASET*, **7(4)** (2019) 1343–1349.

15 Al Aghbari Z, Khedr A M, Osamy W, Arif I & Agrawal D P, Routing in wireless sensor networks using optimization techniques: a survey, *Wirel Pers Commun*, **3** (2020) 2407–2434.