ISSN:0975-9646

Detection of Faulty Sensors in Wireless Sensor Networks and Avoiding the Path Failure Nodes

Pinak S. Patel
M.E. Student, GTU
PIET, Limda, Tal. Waghodia,
Vadodara, Gujarat (India)

Asst. Prof **Mohammed Husain Bohara**
Computer Science and Engineering Dept.
PIET, Limda, Tal. Waghodia,
Vadodara, Gujarat (India)

Binita D. Chahwala
M.E. Student, GTU
PIET, Limda, Tal. Waghodia,
Vadodara, Gujarat (India)

***Abstract*— For variety of applications, Wireless Sensor Networks (WSNs) have become a new information collection and a monitoring solution. Faults occurring due to sensor nodes are common due low-cost sensors used in WSNs, deployed in large quantities and prone to failure. The goal of this paper is to detect faulty sensors in WSNs and avoiding the path failure nodes. Fault detection is based on the local pair-wise verification between the sensors monitoring the same physical system. Specifically, a linear relationship is shown between the output of any pair of sensors, when the input of a system comes from a common source. Using this relationship, faulty sensors may be detected by using forecasting model based on the parameter (i.e., temperature) and it also identifies which sensor is normal or abnormal. After the fault nodes are detected, first of all disable all the faulty nodes so that network is not affected by erroneous reading and send the information to the base station. Due to the nature of proposed algorithm, it can be scaled to large sensor networks and also saves energy from reduced wireless communication compared to the centralized approaches.**

Keywords—Fault detection, Forecasting, Wireless Sensor Networks (WSNs).

I. INTRODUCTION

Recent advances in communication technology and embedded systems has resulted in wireless sensors that are smaller in size, consume less power and achieve higher transmission rates. By adopting wireless sensor networks (WSNs), the cost of monitoring systems, such as structural and environmental monitoring systems is greatly reduced due to the eradication of expensive wiring. However, small form factors and low costs also render wireless sensors more susceptible to faults and failures. To ensure wireless sensor networks are reliable over long periods of service, automated detection and identification of sensor faults must be an integral part of their design and operation, especially for applications where the monitoring system is unattended after deployment. With most wireless sensors limited in both energy and processing capacity it is highly desirable to design energy efficient and low complexity fault detection algorithms that can be embedded directly into the wireless sensors for in-network execution.

The WSN is built of "nodes"- from a few to several hundreds or even thousands, where each node is connected to one (or sometimes several) sensors [10]. Each such sensor network node has typically following parts: Radio Transceiver, for transmitting and receiving radio signals. Sensors, for sensing of physical or environmental conditions. Microcontroller, for interfacing with the sensors and for necessary computation. Energy Source, it is usually a battery or an embedded form of energy harvesting.

R. Da and C. Lin [4], presents a new approach for detecting sensor failures which affect only subsets of system measurements. In addition to a main Kalman filter, which processes all the measurements to give the optimal state estimate, a bank of auxiliary Kalman filters is also used, which process subsets of the measurements to provide the state estimates which serve as failure detection references. After the statistical property of the difference between the state estimate of the main Kalman filter and those of the auxiliaries is derived with an application of the orthogonal projection theory, failure detection is undertaken by checking the consistency between the state estimate of the main Kalman filter and those of the auxiliaries by means of the chi-square statistical hypothesis test.

V. Ricquebourg [5], proposed a sensor failure detection method based on the fusion of predicted and observed sensor data. In these approach, Ricquebourg use the Markov Chain to model normal behavior of sensor within the TBM framework. When fusion between predicted and observed data is done, three experts analyze conflict resulting from the fusion process and are able to detect an abnormal behavior of the sensor by looking for high increase of conflict. The testing results show that this method is efficient to detect sensor failure with a TBM approach.

Gayathri Venkataraman [6], presents a large number of sensor nodes in wireless sensor network, it is common for sensor nodes to become faulty and unreliable. Faults mainly result from systems or communication hardware failure and the fault state is continuous in time. In this paper, we consider permanent faults only, means faults occur due to battery depletion, which when not noticed would cause loss in connectivity and coverage. In this paper a cluster based fault management schemes which identify and rectifies the problems that occur due to energy depletion in sensor nodes. The connectivity is still maintained by reorganization of cluster, when a sensor node fails. By using clustering approach, it restricts the number of nodes in each cluster and number of next hop neighbours a node can have and it is used to produce energy efficient clusters.

Wireless sensor networks have been applied in various areas, such as environment and habitat monitoring, condition based equipment maintenance, disaster management, and emergency responses, but due to low cost and large number of nodes, it may prune to failure [3]. Our adopted approach is the mutual testing method at processor

level where each processing elements is capable of testing its neighbours and generates a result based on the success of the test results. The test result may be arbitrary because the tester itself can be faulty. A processor is identified to be good or faulty by knowing the collection of test results. However the topology of wireless sensor networks is not regular, each sensor must maintain a certain number of neighbours that is the degree of the network must be high. Jinran Chen [7], proposed a distributed localized faulty sensor (DLFS) detection algorithm where each sensor identifies its own status to be "good" or "faulty" and it claim is then supported or reverted by its neighbours as they also evaluate node behavior. The algorithm is analyzed using probabilistic approach; it contains the probabilities of faulty sensors being diagnosed as "good" and good sensors not being diagnosed as "good", because they are very low in entire sensor network.

In recent years, sensors are usually simple, low-cost devices, deployed in large quantities, and prone to failure. Chun Lo [2], presents a model-free and reference-free spike fault identification method based on pair-wise verification. When the input of a system comes from a common source, there is a linear relationship between the output of any pair of sensors. This linear relationship between sensor pairs can be obtained through training. We present a method which is able to find faulty sensors suffering from sparse spikes in their outputs by pairwise comparisons even though there is no knowledge of which sensor is normal or abnormal, and no knowledge of the common input.

Chun Lo [1], proposed a novel fault detection method which utilizes system redundancy but without requiring knowledge of a physics-based system model or the existence of reference sensors is proposed. If in a network all the sensors are faulty, it will not affect the algorithm performance. The method is especially well suited for resource constrained WSNs because the detection algorithm is run locally by each wireless sensor node resulting in reduced communication demand compared to existing centralized methods. Under certain conditions, knowledge of system inputs is not necessary for the detection algorithm to work. The method is capable of detecting general faults within arbitrary pairs of sensors (i.e., subsystems). However, the method is further specialized to identify spike faults [9] from other type of faults with the aim of detecting and quantifying (e.g., location, magnitude) spikes so that they could be removed during post processing.

In this paper, we identify the fault nodes in wireless sensor networks and maintain the reliability in the base station. For wireless sensor networks to identify fault in sensors**,** where each sensor nodes are grouped into zones and each sensor nodes are routing to the base station.

The remainder of this paper is organized as follows. In Section II, theory for fault detection is stated using linear relationship between pair of sensors. In Section III, the architecture of proposed work is shown with explanation. In Section IV, specifies the whole system in diagram and results for fault detection. Finally, the conclusion of paper is in Section V with a detail summary and a discussion of future research efforts.

II. THEORETICAL BACKGROUND AND PAIR-WISE LINEAR RELATIONSHIPS: FOR FAULT DETECTION

Construct Linear, Time Invariant (LTI) Models [8], the discrete-time state-space model for a time series is given by the following equations:

$$x(kT+T) = Ax(kT)+Ke(kT) \quad (1)$$

$$y(kT) = Cx(kT)+e(kT) \quad (2)$$

where T is the sampling interval and y(kT) is the output at time instant kT. A, C and K are the matrices and k is a constant. The time-series structure corresponds to the general structure with empty B and D matrices. The state-space format is convenient if your model is a set of LTI differential and algebraic equations. For example, consider the following linearized model of a continuous stirred-tank reactor (CSTR) involving an exothermic (heat-generating) reaction.

$$\frac{dC_A'}{dx} = a_{11}C_A' + a_{12}T' + b_{11}T_c' + b_{12}C_{Ai}' \quad (3)$$

$$\frac{dT'}{dx} = a_{12}C_A' + a_{22}T' + b_{21}T_c' + b_{22}C_{Ai}' \quad (4)$$

where C_A is the concentration of a key reactant, T is the temperature in the reactor, T_c is the coolant temperature, C_{Ai} is the reactant concentration in the reactor feed, and a_{ij} and b_{ij} are constants. Fig. 1 shows the process for CSTR Schematic. The primes (e.g., C_A') denote a deviation from the nominal steady-state condition at which the model has been linearized.

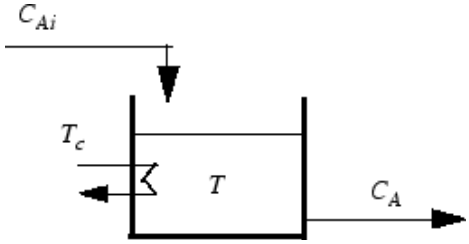

Fig. 1. CSTR Schematic [8]

Measurement of reactant concentrations is often difficult, if not impossible. Let us assume that T is a measured output, is C_A an unmeasured output, T_c is a manipulated variable, and C_{Ai} is an unmeasured disturbance.

The model fits the general state-space format

$$\frac{dy}{dx} = Ax + Bu \quad (5)$$

$$y = Cx + Du \quad (6)$$

where, $x = \begin{bmatrix} C_A' \\ T' \end{bmatrix}$ $u = \begin{bmatrix} T_c' \\ C_{Ai}' \end{bmatrix}$ $y = \begin{bmatrix} T' \\ C_A' \end{bmatrix}$

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

This defines a continous time state-space model. If you don't specify a sampling period, a default sampling value of zero applies. You can also specify discrete-time state-space models. You can specify delays in both continuous-time and discrete-time models.

From the filtered data, spikes are usually detected using an amplitude threshold. The choice of the threshold is a compromise between: i) missing spikes if a high threshold is used (Type II error) and ii) getting false positives due to noise crossing a low threshold (Type I error). An adequate threshold can be set manually, as done in most systems with on-line spike detection. However, an automatic threshold is preferable, especially when processing large number of channels.

The AutoRegressive with eXogenous input (ARX) time-series model is defined [8], as follows:

$$y(t) + a_1 y(t-1) + \dots + a_{n_a} y(t-n_a) = b_1 u(t-1) + \dots + b_{n_b} u(t-n_k-n_b+1) + e(t) \quad (7)$$

where,

- y(t) is the output at time .
- $a_1 \dots a_{n_a}$ and $b_1 \dots b_{n_b}$ are the parameters to be estimated.
- n_a is the number of poles of the system.
- $n_b - 1$ is the number of zeros of the system.
- n_k is the number of input samples that occur before the inputs that affect the current output.
- y(t - 1)…y(t - n_a) are the previous outputs on which the current output depends.
- u(t - n_k)…u($t - n_k - n_b + 1$ are the previous inputs on which the current output depends.
- e(t) is a white-noise disturbance value.

The ARX model in Fig. 2 can also be written in a compact way using the following notation:

$$A(q)y(t) = B(q)\,u(t - n_k) + e(t) \quad (8)$$

Where,

$$A(q) = 1 + a_1 q^{-1} + \dots + a_{n_a} q^{-n_a}$$

$$B(q) = b_1 + b_2 q^{-1} + \dots + b_{n_b} q^{-n_b+1}$$

And q^{-1} is the backward shift operator, defined by,

$$q^{-1}u(t) = u(t-1)$$

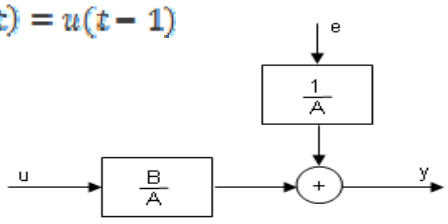

Fig. 2. ARX model structure [8]

Input: The block accepts two inputs, corresponding to the measured input-output data for estimating the model. First input: Input signal, Second input: Output signal. Output: The ARX Estimator block outputs a sequence of multiple models, estimated at regular intervals during the simulation.

Consider a set of wireless sensors attached to a time-invariant physical system. Since sensor responses all depend on the common physical system, a linear relationship exists between the system outputs measured by these sensors. Specifically, sensors can pair up and check whether their outputs are consistent with this linear relationship; inconsistencies can then be used to determine whether one or both of the sensors may be faulty. This pair wise comparison can be performed between any pair of sensors and only the result of the comparison needs to be conveyed to the base station or a central processing node in the WSNs. The structure of the proposed algorithm is illustrated in Fig. 3. The algorithm can be separated into a training (also called "model parameter identification") phase and a detection phase. During the training phase, each sensor node learns the relationship between itself and each of its neighbours. For example, in Fig. 3(a), sensor 2 broadcasts its measurement data to neighbouring sensors 1 and 3. After the data is received, sensors 1 and 3 calculate the relationship between their outputs and sensor 2's output [Fig. 3(b)]. This model parameter identification process is performed by each sensor one after another and the results are stored locally, as shown in Fig. 3(c). During the detection phase, the network is partitioned into pairs of neighbouring sensors; this can be done centrally or in a distributed fashion. Each pair of sensors then performs a comparison according to their trained relationship. Fig. 3(d) shows a network partitioned into 3 pairs: {1, 2}, {2, 3} and {4, 5}. For example, consider sensor pair {1, 2}. Sensor 2 first transmits its output to its partner, sensor 1. Sensor 1 then checks whether its measured output agrees with the output predicted by the previously trained relationship [Fig. 3(e)]. Finally, each sensor pair will report its results to the base station [Fig. 3(f)].

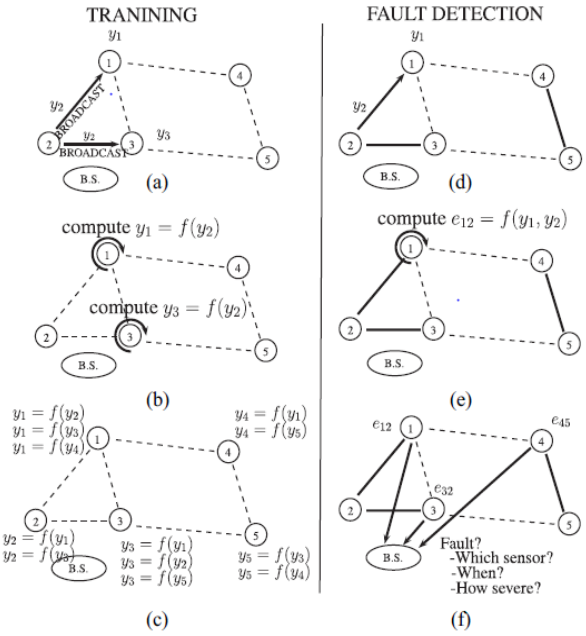

Fig. 3. During training: (a) each sensor broadcasts its output, (b) linear relationship between sensor pairs is calculated, and (c) finally pair-wise linear relationships of the network are constructed. For fault detection: (d) base station divides the sensor network into pairs, (e) each pair performs the fault detection method, and (f) each pair sends their results, *e*, back to the base station (B.S.).

Limitations of Existing System:
Solution by Chun Lo[1] , proposed a novel fault detection method which utilizes system redundancy but without requiring knowledge of a physics-based system model or the existence of reference sensors is proposed. If in a network all the sensors are faulty, it will not affect the algorithm performance. The method is especially well suited for resource constrained WSNs because the detection algorithm is run locally by each wireless sensor node resulting in reduced communication demand compared to existing centralized methods. Under certain conditions, knowledge of system inputs is not necessary for the detection algorithm to work. The method is capable of detecting general faults within arbitrary pairs of sensors (i.e., subsystems).
The solution of Chun Lo only addresses how to identify the faulty sensors. The recovery process is not addressed.

III. PROPOSED WORK

Theory or Steps of Open Forecast [11] , is as follows,

1) Create a DataSet *object.*

2) Add to the data set object a series of DataPoint objects that define a series of observations.

3) Using the static getBestForecast method of Forecaster, obtain a reference to the most appropriate forecast model for your data set.

4) Use the forecast method of this ForecastingModel to forecast additional values.

Create a DataSet object

A DataSet is simply a collection of DataPoint objects. In many respects, you can think of it as just another Java2 Collection. In fact, the DataSet class does implement the java.util.Collection interface. You can create a new DataSet object directly, as follows:

- DataSet observations = new DataSet();

Add DataPoints to the DataSet

Once you have a data set, you can begin to add DataPoint objects to it, using the DataSet add method. There are primarily two ways of defining DataPoint objects. If you already have your observations/data points defined in some Java class, you could extend or modify that class to implement the DataPoint interface. Alternatively, OpenForecast provides an implementation of the DataPoint interface called, Observation. The Observation class is most convenient if you don't currently have an implementation of your observation data.
Using the Observation class
The Observation class provides a complete implementation of the DataPoint interface. Consider the quarterly sales of a company product to be $500 (thousand) for period 1, $600 (thousand) for period 2, and $700 (thousand) for period 3 for creating 3Observation objects representing these observations.
Note that the Observation constructor takes a single value. This value is the dependent value - the value we observe initially, but later want to forecast. After creating a new Observationobject, we then invoke the setIndependentValue method for each independent value associated with the observation. In this case, we have just one independent variable, and that is, "quarter". Therefore, for each observation we must set the value of the independent variable, quarter, to the appropriate value.
Note that the independent variable name, quarter, used in each of the observations must be exactly the same among the different observations if they really refer to the same independent variable. Next, we must add these to our DataSet. If, in addition, we expect that the average daytime high temperature for the quarter has an influence on sales, then we can add the value of this independent variable to each observation. For example, if we knew the average daytime high temperatures for quarters 1, 2 and 3 were 45°F, 63°F and 97°F respectively, then we could define the three observations.

Obtain a ForecastingModel

Now that we have a set of DataPoint objects defining our observations, we need to obtain a ForecastingModel. Two primary approaches are available here. The first approach requires little knowledge of the different forecasting models available and, in general, is the preferred approach. The second approach to obtaining a ForecastingModel is to decide which model to use, and instantiate it directly. This provides for selection of a specific model, however, the trade-off is that you may not necessarily get the model that best fits your data.

Generate forecasts

Defining the forecast data set,
To obtain a forecast for other data points, you first need to decide - or otherwise determine - what data points you want to produce a forecast for. Using the quarterly sales examples from the section called Using the Observation class, say that we want to produce quarterly sales forecasts for quarters 4 and 5 (the first quarter in the following year). Then we'd need to define two data points, as before, but referring to quarters 4 and 5. Note that we initialize the dependent value for each DataPoint to 0.0. It really doesn't matter what value is used here because the dependent value - the value that we intend to forecast - will be updated when we call forecast. Once we have created the required DataPoint objects, we gather them together in a data set, fcDataSet.
Obtaining Forecast values,
Once we have defined a data set containing the data points for which we require forecasts of the dependent values, all that is necessary is to pass this to model's forecast method. For convenience, the forecast method returns the DataSet passed in, that will have been updated with the new forecast values. However, since it is the same as the one passed in, it is not uncommon to ignore the return value.
The solution of Chun Lo only addresses how to identify the faulty sensors, but recovery process is not addressed. The transmission must be stopped after identifying faulty sensor. This requires the role of base station. So we are trying to propose a mechanism for communication of fault information in the network & disabling of the faulty sensors so that the network is not affected by erroneous reading from the faulty devices.

The Fig. 4 explain the system flow. First we configure a network and create a new network specifying the number of nodes and range for which each node can communicate with its neighbours within a given range. These nodes are divided into zones. If some event has been triggered in one or more zones, then select that zones using zoneid and it will show all the nodes in that particular zone, than indicate that event (i.e. checking of faulty nodes). It also includes updating queue and forecast the next value. Queue may contain at least five or more observations for forecasting the next value. Check triggered value against forecasted value, if it is out of threshold value than that node will be assumed as faulty or if the value of node is within threshold than we can say that it is working properly. So if the node is identified as faulty, than the base station will not accept the data from faulty node, so the network is not affected by erroneous reading from faulty devices.

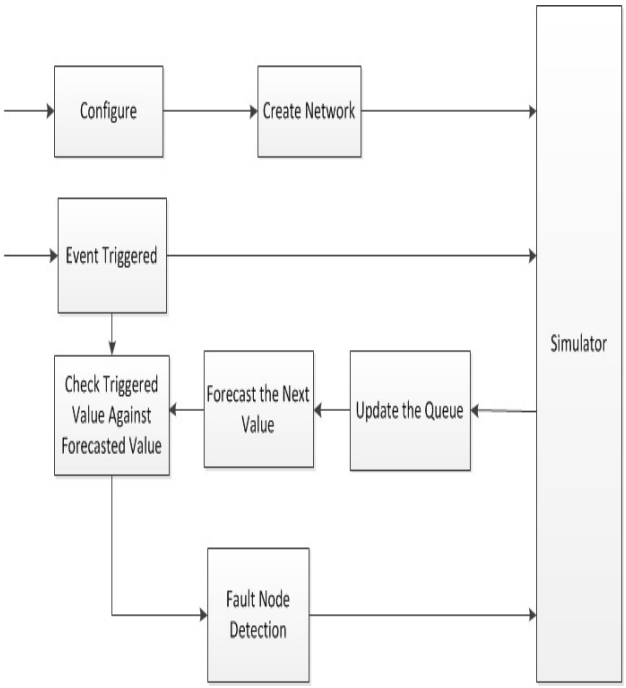

Fig. 4. System Architecture

A novel fault detection method is capable of detecting general faults within arbitrary pairs of sensors (i.e. subsystems). For evolution of algorithm, temperature is the best parameter still we have chosen forecast as a parameter to evaluate algorithm. Forecast value is the average value of the readings of temperature taken by each node.

IV. SEQUENCE DIAGRAM OF SYSTEM OPERATION & RESULTS

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Fig. 5 and Fig. 6 shows the total flow for network creation and successful operation.

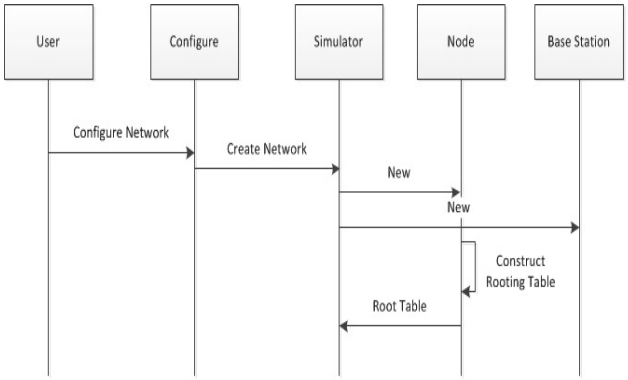

Fig. 5. Network Creation Sequence diagram

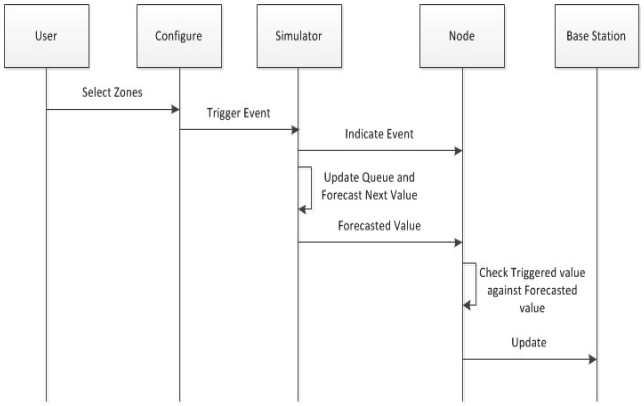

Fig. 6. Succesful Operation Sequence diagram

Expected outcome of proposed work:
To identify faulty nodes in the network and what information is provided by faulty nodes will not be considered by the base station. Means if any event is triggered on faulty node will not be accepted by the base station.

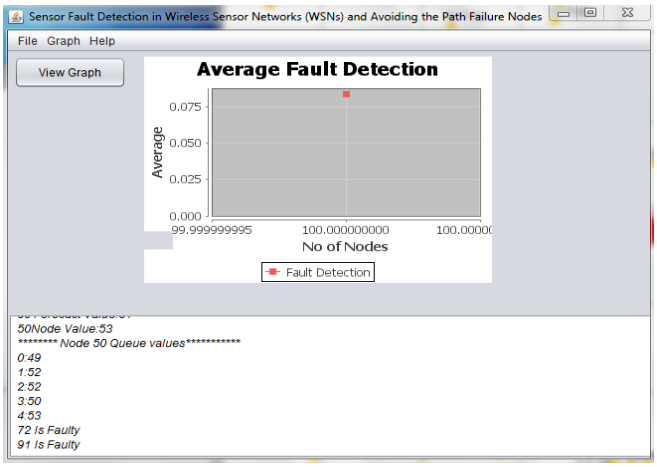

Fig. 7. Average Fault Detection Graph

In these graph, its shows the average fault detection from the number of nodes specified by user. Here user created a sensor network for hundred nodes and on checking of fault detection, it finds two nodes as faulty and the graph is as shown in fig. 7.

V. CONCLUSION

In this survey we consider an ARX-based spike fault detection method which does not require the system-input information or *a* priori establishment of reference sensors is proposed for LTI physical systems. The method is based on pair-wise relationships of sensors, and these relationships are learned online when the system is functioning normally. Moreover, the proposed method is able to identify all of the faulty sensors and Communication of fault information in network. Through Simulation we are blocking the faulty nodes and the fault node information is send to the base station. This method gives good performance; it only loses part of its effectiveness under situations where a pair of sensors is highly correlated. Further research will be done to develop a method to separate types of fault occurs when the nodes become faulty and also try to recover that fault.

REFERENCES

[1] Chun Lo, Student Member, IEEE, Jerome P. Lynch, Member, IEEE, and Mingyan Liu, Senior Member, "Distributed Reference-Free Fault Detection Method for Autonomous Wireless Sensor Networks", IEEE SENSORS JOURNAL, VOL. 13, NO. 5, MAY 2013.

[2] Ph.D. Candidate Chun Lo, Assoc. Prof. Jerome P. Lynch, Ph.D. Assoc. Prof. Mingyan Liu, Ph.D., "Reference-free Detection of Spike Faults in Wireless Sensor Networks", 978-1-4244-9293-0/11/$26.00 ©2011 IEEE.

[3] E.chow and A.Willsky, "Analytical redundancy and the design of robust failure detection systems", IEEE Trans. Automat. Control, vol. 29,no. 7,pp. 603-614, Jul. 1984.

[4] R. Da and C. Lin, "Sensor failure detection with a bank of kalman filters", in Proc., Amer. Control Conf., vol. 2. 1995, pp. 1122–1126.

[5] V. Ricquebourg, D. Menga, M. Delafosse, B. Marhic, L. Delahoche, and A. Jolly-Desodt, "Sensor failure detection within the tbm framework: A markov chain approach", in Proc. Inform. Process. Manag. Uncertain., vol. 8. 1991, p. 323.

[6] Gayathri Venkataraman, Sabu Emmanuel, Srikanthan Thambipillai, "A Cluster-Based Approach to Fault Detection and Recovery in Wireless Sensor Networks", IEEE ISWCS 2007, 1-4244-0979-9/07/$25.00 © 2007 IEEE.

[7] Jinran Chen, Shubha Kher, and Arun Somani, "Distributed Fault Detection of Wireless Sensor Networks", Dependable Computing and Networking Lab Iowa State University Ames, Iowa 50010.

[8] Linear-time-invariant-lti-models, time-series-model, arx, "www.mathworks.in"

[9] Spike_sorting, "www.scholarpedia.org"

[10] Wireless Sensor Network, "http://en.wikipedia.org"

[11] Open Forecast, "http://www.stevengould.org"