Failure Detection in Wireless Sensor Networks: A Sequence-Based Dynamic Approach

ABU RAIHAN M. KAMAL and CHRIS J. BLEAKLEY, University College Dublin
SIMON DOBSON, University of St Andrews

Wireless Sensor Network (WSN) technology has recently moved out of controlled laboratory settings to real-world deployments. Many of these deployments experience high rates of failure. Common types of failure include node failure, link failure, and node reboot. Due to the resource constraints of sensor nodes, existing techniques for fault detection in enterprise networks are not applicable. Previously proposed WSN fault detection algorithms either rely on periodic transmission of node status data or inferring node status based on passive information collection. The former approach significantly reduces network lifetime, while the latter achieves poor accuracy in dynamic or large networks. Herein, we propose Sequence-Based Fault Detection (SBFD), a novel framework for network fault detection in WSNs. The framework exploits in-network packet tagging using the Fletcher checksum and server-side network path analysis to efficiently deduce the path of all packets sent to the sink. The sink monitors the extracted packet paths to detect persistent path changes which are indicative of network failures. When a failure is suspected, the sink uses control messages to check the status of the affected nodes. SBFD was implemented in TinyOS on TelosB motes and its performance was assessed in a testbed network and in TOSSIM simulation. The method was found to achieve a fault detection accuracy of 90.7% to 95.0% for networks of 25 to 400 nodes at the cost of 0.164% to 0.239% additional control packets and a 0.5% reduction in node lifetime due to in-network packet tagging. Finally, a comparative study was conducted with existing solutions.

Categories and Subject Descriptors: C.2.2 [**Computer-Communication Networks**]: Network Protocols

General Terms: Performance

Additional Key Words and Phrases: Sensor network, fault detection, wireless sensor networks, sensor data, senor faults, data collection, routing

ACM Reference Format:
Abu Raihan M. Kamal, Chris J. Bleakley, and Simon Dobson. 2014. Failure detection in wireless sensor networkss: A sequence-based dynamic approach. ACM Trans. Sensor Netw. 10, 2, Article 35 (January 2014), 29 pages.
DOI: http://dx.doi.org/10.1145/2530526

1. INTRODUCTION

Wireless Sensor Networks (WSNs) have been used in many applications, including environmental monitoring, industrial plant monitoring, and traffic monitoring. Typically, WSNs contain a large number of resource-constrained sensor nodes which are powered by nonrechargeable batteries. Nodes are often placed in hard-to-reach areas and are kept there for prolonged durations for the purpose of data reporting or event

This work is partially supported by the Higher Education Authority PRTLI4 under grant number R10891 "NEMBES: Networked Embedded Systems."
Authors' addresses: A. R. M. Kamal (corresponding author) and C. J. Bleakley, Complex and Adaptive Systems Laboratory, School of Computer Science and Informatics, University College Dublin, Ireland; email: raihan@ucdconnect.ie; S. Dobson, School of Computer Science, University of St Andrews, UK.

DOI: http://dx.doi.org/10.1145/2530526

monitoring. Experience from real-world WSN deployments, such as Langendoen et al. [2006], Arora et al. [2004], Beckwith et al. [2004], Krishnamurthy et al. [2005], Szewczyk et al. [2004], Buonadonna et al. [2005], and He et al. [2006], indicates that node failure, node reboot, and link failure are common occurrences [Wachs et al. 2007]. For example, during the unwired wine deployment [Beckwith et al. 2004], researchers reported data delivery rates of over 90% in the laboratory but only 77% in the real deployment due to node and link failures. In a surveillance deployment [Arora et al. 2004], extreme environmental conditions caused node failures affecting the entire network. A WSN-based industrial monitoring system deployed in the North Sea was subject to temporally correlated node failures [Krishnamurthy et al. 2005]. The developers of the Great Duck Island experiment reported significant packet loss due to node failure [Szewczyk et al. 2004].

A number of researchers have proposed techniques for detecting failures in WSN deployments. Most existing solutions, such as Sympathy [Ramanathan et al. 2005], Memento [Rost and Balakrishnan 2006], and Residual Energy Scan [Zhao et al. 2002], rely on a proactive approach whereby each node maintains a debugging agent which reports node and link status periodically. The proactive approach suffers from three drawbacks. First, it incurs computational and communication costs at the nodes which significantly shorten network lifetime. Second, it suffers from significant latency since status information is only sent periodically. Third, it relies on the assumption that node and link behavior are completely deterministic and can be controlled locally.

Recently, some researchers have proposed the use of passive information collection for the purpose of failure detection [Liu et al. 2010; Guo et al. 2009; Chen et al. 2008]. These proposals are based on the idea that information useful for failure detection can be extracted from regular data packets sent to the sink node. Unlike the proactive approach, the passive approach does not incur significant network overhead. However, it depends on the sink node using nondeterministic means of inferring the operational status of nodes and links based on the data collected and a network model. As a result, the proposals suffer from poor accuracy and do not scale well with network size.

Herein, we propose a novel framework, named Sequence-Based Failure Detection (SBFD), for detecting WSN failures. SBFD combines lightweight in-network packet tagging and server-side storage-intensive computation. Nodes piggyback *path checksum* tags onto all regular data packets going to the sink node. Each node handling the packet updates the tag with its own node ID by means of the Fletcher checksum algorithm. The resultant path checksum is lightweight in terms of communication overhead and is efficient in terms of node computation. The path checksum is inspected on arrival at the sink. A Network DataBase (NDB) is used to *deterministically deduce* the packet path from the path checksum. On detecting a persistent path change, the sink node injects a series of control messages into the network. Based on the received responses to these control messages, the sink identifies, classifies, and reports the failure.

To the best of the authors' knowledge, SBFD is the first work to propose lightweight in-network path checksum tagging with server-side path deduction and network failure detection. The approach has three major advantages. First, it is lightweight in terms of communication and processing costs and so is suitable for implementation in resource-constrained WSN nodes. Second, because of the distributed nature of SBFD, it is highly accurate in terms of failure detection even for very large networks. Third, it can detect faults with low latency.

We outline the overall contributions of this work as follows.

—A framework (SBFD) for fault detection which operates on top of most greedy multihop routing protocols is proposed. By greedy we mean WSN routing protocols that always select the best path based on the routing metric, such as hop count or/and

connectivity, for example, CTP [Gnawali et al. 2009] or Arbutus [Puccinelli and Haenggi 2010]. The framework does not impose any special requirements or restrictions on the underlying network protocol.
—An algorithm for recording a packet path, based on the Fletcher checksum algorithm, is described. The method generates a path checksum and can be efficiently implemented in resource-constrained sensor networks.
—An algorithm for packet path deduction based on the path checksum is presented.
—Algorithms for accurate detection of node failure, node reboot, and link failure based on analysis of packet path data are presented.
—The accuracy of SBFD is evaluated with a small testbed deployment.
—The performance of SBFD, and its associated algorithms, is assessed in terms of communication and computation costs and their impact on node lifetime.
—The scalability of SBFD is assessed in simulations of large networks of various types of scenarios.
—The performance of SBFD is compared with that of previous proposals.

The remainder of the article is organized as follows. Section 2 summarizes the definitions and notation used throughout the article. Section 3 contains the problem statement. The proposed system is presented in detail in Section 4. Section 5 assesses the performance of the system both in a testbed and in simulation. Related work is discussed in Section 6 and Section 7 concludes.

2. DEFINITIONS AND NOTATION

The following definitions are used in the article.

—*Source Node.* A Source Node (SN) is a node within the network which sends its own sensed data to the sink node.
—*Relay Node.* Relay Nodes (RNs) send received packets from neighboring nodes to other neighboring nodes. RNs are needed if there is no direct radio link between a source node and the sink node due to the limited range of WSN radios. As well as sending its own data, an SN may act as a relay node. Dedicated Relay Nodes (DRNs) do not act as SNs.
—*Sink Node.* The sink node collects the data generated by the SNs. It is sometimes referred to as the Base Station (BS).
—*Network Size.* The network is a collection of N nodes, arranged in such a way that every node, x_i, where i is between 0 and $N-1$, is in radio communication with the sink node, either directly or through a number of intermediate hops. We refer to N as the size of the network.
—*SN Ratio.* For a given network, the SN ratio α is the fraction of the total nodes which are SNs. The ratio ranges between 0 and 1.
—*Network Length.* For a given SN, the shortest path to the sink is the one which uses the least number of intermediate RNs, that is, fewest hops. For a given network, the farthest SN is the one with the longest shortest path to the sink. The network length h is the number of intermediate relay nodes in the shortest path from the farthest SN plus one, that is, the network length includes the SN itself.
—*Mean Connectivity.* A node x_i is deemed to have connectivity d_i if $\sum_{j=0}^{N-1} link(x_i,x_j)/2 = d_i$ where $\texttt{link}(x_i, x_j) = 1$ if there exists bidirectional direct radio communication between nodes x_i and x_j and $\texttt{link}(x_i, x_j) = 0$ otherwise. A network is characterized by its mean connectivity, $d_\mu = \sum_{i=0}^{N-1} d_i/N$, that is, the average number of nodes with which a node has direct connectivity.
—*Dense and Sparse Network.* For a given network length, a network can be classified either as *dense* or *sparse* depending on network size N and mean connectivity d_μ.

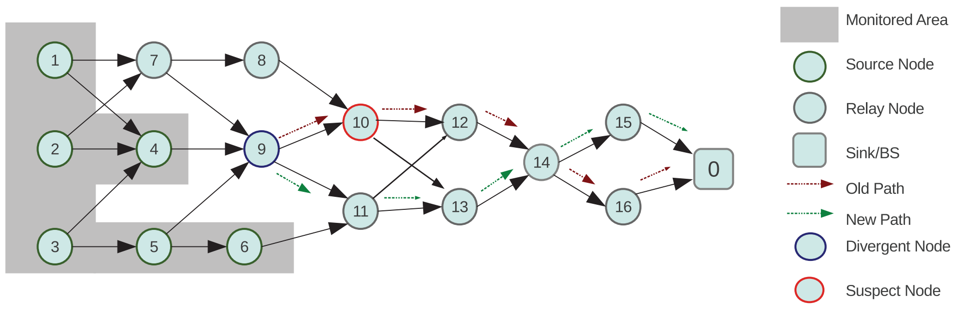


Fig. 1. Example WSN failure.

Higher values of N and d_μ indicate a dense topology and lower values indicate a sparse topology.

—*Network Topology.* The network topology consists of the node IDs, their role (i.e., SN, RN, DRN, or sink), and the connectivity between nodes (the links).

—*Divergent Node.* The Divergent Node (DN) is the node which is the last common node in two paths from a single SN to the sink, that is, the DN is the last node for which $P_{old}(i) = P_{new}(i)$ where $P_{old}(0)$ is the SN ID.

—*Suspect Node.* The Suspect Node (SuN) is the node immediately after the DN in the old path, that is, the SuN is first node of P_{old} for which $P_{old}(i) \neq P_{new}(i)$.

—*Suspect Link.* The Suspect Link (SuL) is the link between the DN and the SuN, that is, the SuL is $P_{old}(i-1) \rightarrow P_{old}(i)$, where $P_{old}(i-1)$ is the DN and $P_{old}(i)$ is the SuN.

—*Idle Node and Link.* Idle nodes or links are those which do not take part in data collection, either as SNs or RNs or DRNs, for prolonged periods of time.

Figure 1 illustrates the preceding definitions. The nodes in the monitored area are SNs, the remainder are RNs, and Node 0 is the sink. The network size N is 16. The mean connectivity d_μ is 1.53. Based on this, the network can be viewed as sparse. The network length h is 7 as the farthest node (i.e., Node 1) is 7 hops away from the sink. The original path for packets from Node 2 to the sink is $\{2,4,9,10,12,14,16,0\}$. After Node 10 fails, the path becomes $\{2,4,9,11,13,14,15,0\}$. For this path change, Node 9 is the Divergent Node (DN) and Node 10 is the Suspect Node (SuN). The link between Nodes 9 and 10 is the Suspect Link (SuL).

3. PROBLEM STATEMENT

Failures in WSN deployments can be broadly classified into the following two groups [Guo et al. 2009]:

Data Failure. A network experiences a *data failure* whenever a source node performs the task of sensing erroneously. As a result, the system fails to accurately report the true underlying physical quantity being monitored. In-network data validation, server-side outlier detection, and model-based methods are the main tools for dealing with such failures.

Network Failure. Network failures can occur due the sudden death of a node, intermittent radio connectivity, packet loss due to routing failure, and so on. More precisely, a network failure can be classified as either of the following.

(1) *Node Failure.* Since nodes in WSNs are often unattended in remote locations, node failures may occur due to component failure, node destruction by an external event, or a sudden depletion of stored energy.

(2) *Link Failure.* Links in WSNs are often failure prone due to temporary blocking by moving external objects, abrupt changes in the radio environment, or temporary interference from other radio sources. Link failure can cause network partition.
(3) *Node Reboot.* Apart from node and link failure, a network may experience repeated node reboot due to energy depletion or software malfunction.

In order to effectively maintain and operate a deployed network it is important to detect network failures. Such failures might cause network partition if they are not detected, localized, and corrected. Therefore, the problem addressed in this work is to detect and identify network failures, specifically node failures, link failures, and node reboots, with high accuracy, low network overhead in terms of processing and communication costs at the nodes, and low latency. The solution should be scalable so that it is effective and efficient in large networks.

In this article we assume the following.

—The nodes send application data to the sink regularly using a multihop routing protocol.
—The timing of data transmissions to the sink is not known in advance. For example, some nodes might send data regularly while others might not send data to the sink for long periods.
—A suitable routing protocol, for example, CTP [Gnawali et al. 2009] and Arbutus [Puccinelli and Haenggi 2010], that can handle multihop data forwarding in the face of unstable wireless connectivity, is active.
—The sink node is a high powered node, in terms of processing, storage, and energy supply.
—Each node in the network has a unique node ID.
—The network topology is known initially.
—Nodes can leave and join the network.
—Nodes are static.
—There is no security threat from an external attacker which can cause node misbehavior.

4. PROPOSED SYSTEM

The following sections describe the proposed system. Section 4.1 presents a system overview. The subsequent sections are dedicated to in-depth descriptions of the main components of the framework. The special case of low data rate networks is dealt with in Section 4.6.

4.1. Framework Overview

The SBFD framework is depicted in Figure 2. The framework consists of four main components:

—In-network Packet Tagging (IPT);
—Network DataBase (NDB);
—Network Path Analysis (NPA);
—Fault Detection and Identification (FDI).

IPT is performed by the nodes in the network. NPA and FDI are performed by, and the NDB is stored at, the sink node.

During normal network operation, IPT causes a path checksum to be added to all packets sent from SNs to the sink. Nodes on the routing path of the packet update the path checksum using their node ID and the current path checksum as inputs to the Fletcher checksum algorithm. When a packet arrivals at the sink, the path checksum is used to determine the packet path by means of lookup in the NDB. The NDB is

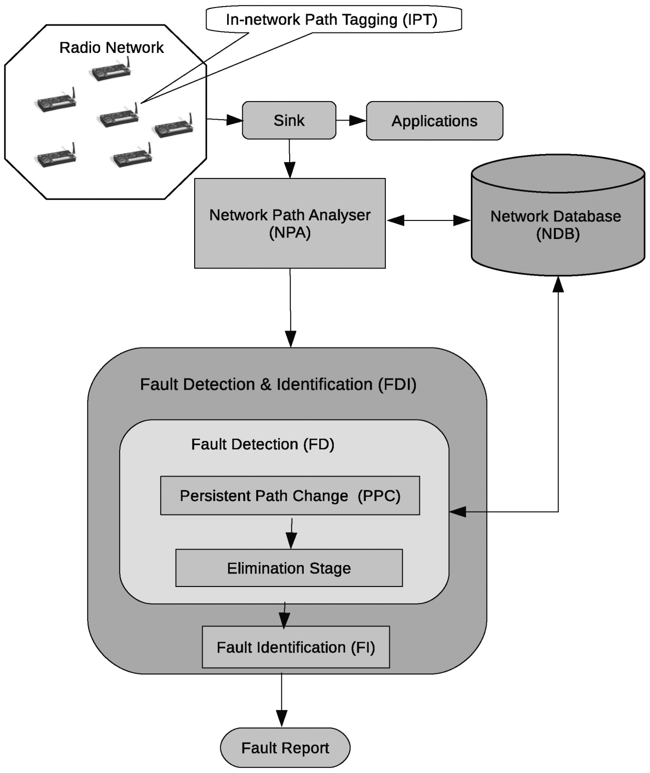

In-network Path Tagging (IPT)
Radio Network
Sink
Applications
Network Path Analyser (NPA)
Network Database (NDB)
Fault Detection & Identification (FDI)
Fault Detection (FD)
Persistent Path Change (PPC)
Elimination Stage
Fault Identification (FI)
Fault Report

Fig. 2. SBFD framework.

prepopulated with the paths and path checksums for the network based on its known topology and NPA. The sink updates the statistics for the packet path in the NDB. The FDI module inspects the NDB to determine if the network path statistics have changed. If they have, the FDI module reviews the NDB path information to determine if a fault is likely. If a fault is suspected, the FDI module sends control messages to the affected nodes. Based on the responses to these control messages, the sink determines if a fault has occurred or not. If a fault is detected the location and type of the fault are reported. The types of faults detected are node failure, link failure, and node reboot.

4.2. In-Network Packet Tagging

IPT appends packet path information to all data packets sent from all SNs to the sink. The goal of IPT is to allow the sink to efficiently monitor activity in the network.

All SNs are required to append their node ID and a path checksum to all packets sent to the sink. The path checksum consists of R K-bit data words (RK bits). The SN sets the initial path checksum to the value of its node ID, that is, $c(0) = x_i$, where $c(i)$ is the value of the checksum at hop i. Later, on receipt of the packet, every RN recovers the current value of the path checksum $c(i)$ and generates a new checksum value $c(i + 1)$, using the Fletcher checksum algorithm, as

$$c(i+1)[0] = |c(i)[0] + x_i|_M, \tag{1}$$

$$c(i+1)[j] = |c(i)[j] + c(i+1)[j-1]|_M \qquad \text{for } j = 1 \text{ to } R-1, \tag{2}$$

where $c(i)[j]$ is word j of checksum c at hop i, x_i is the ID of the RN, $|.|_M$ is the modulus operator, and M is the checksum modulus.

The node replaces the current path checksum in the packet with the new checksum, the packet Cyclic Redundancy Check (CRC) (or forward error correction information) is recalculated, and the packet is forwarded to the sink, if necessary via a neighboring RN.

The Fletcher checksum algorithm was originally introduced for checking that blocks of stored or transmitted data were recovered or received correctly [Fletcher 1982]. The Fletcher algorithm was chosen for this work since its value depends on the sequence of the IDs. Summation checksums only add node IDs together, which gives the same checksum for different paths. For example, the paths {3,0,1,2} and {3,1,0,2} have the same simple summation checksum but different Fletcher checksums. Detecting these changes in the routing path are important in the FDI step.

Herein, we use $R = 2$, $K = 8$ and $M = 255$, giving a 16-bit checksum. This ensures that the checksum is lightweight and is efficient to implement on a resource-constrained processor. The modulo 255 operation can be carried out simply by using a bit mask. A 16-bit checksum provides good disambiguation between paths. If the paths and node IDs are random, a 16-bit checksum gives a 0.001526% probability of two paths having the same checksum. Typically, in WSNs, node IDs are 16 bit, allowing a network of 65,535 nodes with unique IDs. In this case, the checksum input is obtained by XORing the most signifiant byte with the least significant byte of the node ID. The final algorithm is defined formally in Algorithm 1.

ALGORITHM 1: Fletcher Checksum

```
1: Input variables
   :$CHK$, checksum of the traversed nodes,initialized to 0
   :$NODEID$, Node ID of the source node and receiving node
2: Output variables
   : $CHK$, 16-bit cheksum computed over the previous checksum and node ID
3: Local variables
   : $inputdata[4]$, an array for storing data after compacting into 8-bit chunks
   : $S1, S2$, variables to store partial results
4: Initialize variables:
   $S1 = 0$;
   $S2 = 0$;
   //Perform compacting Previous Result(16-bit) into two 8-bit inputs
5: inputdata[1]=CHK &  0xFF
   inputdata[2]=CHK  >>  8
   //Perform compacting Node ID (16-bit) into two 8-bit inputs
6: inputdata[3]=NODEID &  0xFF
   inputdata[4]=NODEID >>  8
7: for $i = 1$ to 4 do
8:   S1=S1+inputdata[i]  %  255
     S2=(S2+S1) %  255
9: end for
10: CHK=(S2  <<  8) | S1
11: Return CHK
```

The algorithm requires a fixed number of basic operations (i.e., 13). Every node computes the partial checksum based on two inputs: the previous checksum and its own node ID. The process is independent of network parameters. Hence, the complexity of the algorithm is O(1). In other words, it runs in constant time.

4.3. Network DataBase

The NDB stores statistics on the paths used by packets arriving at the sink and on the activity of the nodes. The network database consists of two tables: the Path Activity Table (PAT) and the Network Activity Table (NAT).

The PAT holds a record for all observed packet paths from all SNs to the sink. Each record contains the following fields:

```
sn_id, path_checksum, timestamp, freq, path_list
```

where `sn_id` is the source node ID, `path_checksum` is the value of the Fletcher checksum when the packet arrives at the sink, `timestamp` is the arrival time of the most recent packet that used the path, `freq` is the number of times that the path was used since network initialization, and `path_list` is the list of the IDs of the nodes on the path (in order from the SN to the sink but excluding the SN and the sink IDs). The unique key of the table is `sn_id,path_checksum`.

The NAT stores a single record for every node in the network. Each record contains the following fields:

```
node_id, timestamp
```

where `node_id` is the node ID and `timestamp` is the time that the node was alive either because of its SN or RN activity. The unique database key is `node_id`.

The information in the NAT can be extracted from the PAT but updating both the PAT and NAT as each packet arrives significantly reduces processing time at the sink.

4.4. Network Path Analysis

The NPA works in association with NDB. Specifically, it is responsible for initializing and updating the NDB.

4.4.1. NDB Initialization. The PAT is populated by generating records for the most probable packet paths in the network using Algorithm 2. For each SN, a set of possible packet paths from the SN to the sink is generated based on the network topology. The paths are generated subject to the constraint that no generated path may be longer than the shortest path from the SN to the sink plus r hops, where r is the search radius.

In general, the number of alternate paths (determined by r) that need to be considered is low since routing algorithms are specifically designed to use optimal paths based on the specific greedy routing protocols used. For each node a predefined number, r, of 1-hop nodes are selected. The search is expanded from each of these selected nodes to a maximum of r of their neighbor nodes and so on. The process continues until the sink is reached. The number of all possible paths from a specific SN to the sink depends on two parameters: r, the number of neighbors to be selected and h, the network length. Thus the algorithm has complexity $O(r^h)$. Although the algorithm incurs high computational overhead, the path computation process is a part of NDB initialization performed at the server side and so has no impact on nodes' lifetime or operational performance. For each path generated, a PAT record is created including the SN, path checksum, and path list. The timestamp and frequency fields are set to zero. The NAT is also initialized based on the topology. The timestamp field is set to zero for all records.

4.4.2. NDB Update. The NPA inspects the path checksums of all packets arriving at the sink. Based on this, it updates the path statistics in the NDB. The goal is to maintain accurate, low latency information on the active packet paths in the network. The entire IPT and path deduction process is illustrated in Figure 3.

When a packet is received at the sink and passes the CRC check, the `sn_id` and `path_checksum` tags are extracted from the packet and used to look up the corresponding record in the PAT. If a corresponding PAT record exists, the `timestamp` field is updated with the current time and the `freq` field is incremented. In addition, the `timestamp` fields are updated in the NAT records of the SN and all of the nodes in the path of the latest packet. If a corresponding PAT record does not exist, the SN, path checksum, and time of arrival are stored in a temporary table on the basis that the information

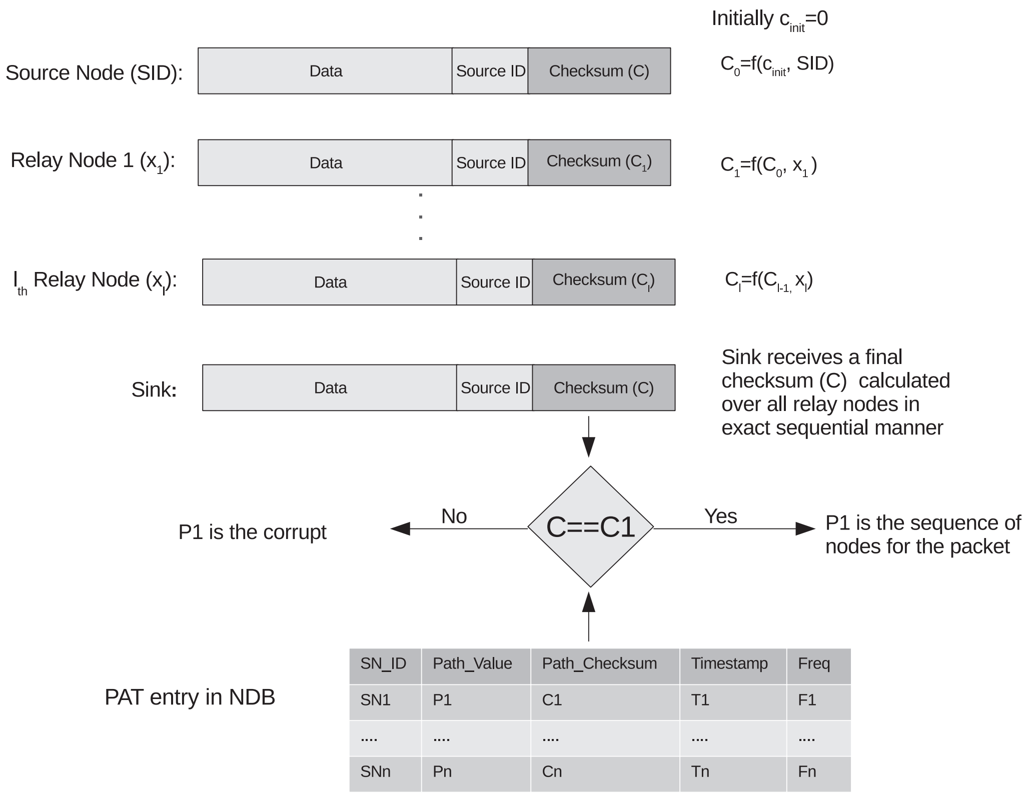
ALGORITHM 2: Path Compute

```
1: Input variables
   :$h$, Network Length
   :$d_\mu$, Mean Connectivity
   :$SN$, Source Node
   :$Sink$, Sink of the network
   :$r$, maximum allowable alternative paths s.t. $r \le d_\mu$
2: Output variables
   : $P_i$, possible paths from SN to Sink
3: Local variables
   : $VN$, to hold the value of SN
   : $NN[]$, an array to hold next nodes at each iteration
4: Initialize variables:
   $VN = SN$
   $r =$ selectValueBetween($1, d_\mu$)
   // The function getNextNodes(VN,r) selects $r$
   //next nodes (1-hop) to form the routing tree rooted at Sink.
   $NN[] =$ getNextNodes(VN,r)
5: repeat
6:
7:   while $NN[] \neq NULL$ do
8:     $x_i =$ getNode(NN[])
       // The function TowardsSink(n1,n2,sink) returns 1 if ¡n1,n2¿ is a valid path
       // based on the routing metric used
9:     if $TowardsSink(VN, x_i, Sink) == 1$ then
10:      $P_i = P_i||VN||x_i$
         $VN = x_i$
         $NN[] =$ getNextNode(VN,r)
11:    else
12:      $x_i =$ getNode(NN[])
13:    end if
14:  end while
15: until $P_i.Length() \le (h - 1)$
```

may be spurious. If the SN and path checksum recur, the NPA module performs an expanded search for paths originating at that SN. If the path is resolved, a new entry is added to the PAT. If not, the NPA module sends a trace path control message to the SN to resolve the path.

4.4.3. Network Dynamics. The NDB must be updated when nodes join or leave the network. When joining the network, new nodes must notify the sink of their ID and 1-hop connectivity. The NPA module must add the node to the NAT and generate records for the new network paths in the PAT. Path generation proceeds in a similar way to that described in Algorithm 2 except that the SNs are constrained to only include the new node and those SNs further from the sink. When a node leaves the network, the NAT is updated and all records involving the node ID are deleted from the PAT. If a node fails permanently, the NDB can be updated in a similar fashion.

4.4.4. Implementation Issues. There is obviously a trade-off between the size of the NAT and computational complexity at the sink in terms of on-the-fly path searches. A high value of r leads to a large NAT with many unused paths (`freq`= 0) but low operational computational complexity. Conversely, a low value for r leads to low storage requirements but higher operational computational complexity. Since WSNs are battery powered and targeted at long network lifetime, the number of packets arriving



SN_ID	Path_Value	Path_Checksum	Timestamp	Freq
SN1	P1	C1	T1	F1
....	....	....	....	....
SNn	Pn	Cn	Tn	Fn

Fig. 3. Path deduction process.

per second is typically low. Therefore, provided that a powerful sink node is used, on-the-fly path searches are not particularly onerous in terms of computational complexity. NAT size can be reduced during operation by removing paths which have never been observed, that is, `freq`= 0, or have not been observed for a long time, that is, `timestamp`$\ll$`current_time`.

Clearly, the size of the NDB must be reasonable for practical implementation. The average size of each field in the PAT and the NAT is $\{2, 2, 4, 4, h\}$ and $\{2, 4\}$ bytes, respectively. The average `path` field is a sequence of 2-byte node IDs where the average number of nodes in the path is half the network length ($h/2$). Clearly, the total size of the NAT is $6N$ bytes. The total size of the PAT can be estimated as follows. If α is the fraction of SNs in the network then αN is the total number of SNs. For each SN, and only considering $r = 0$, there are $(d_\mu/2)^{h/2}$ paths since, on average, there are $d_\mu/2$ links closer to the sink at each hop and a total of $h/2$ hops to the sink. For a small r, the total number of paths from a SN to the sink stored in the database is, on average, approximately $(r + 1)(d_\mu/2)^{h/2}$. The total size S of the NDB in bytes is then

$$S = \alpha N(r + 1)(d_\mu/2)^{h/2}(h + 12) + 6N. \tag{3}$$

For a circular network of N = 2,000 nodes with the sink in the center, we have $\alpha = 0.5, h = 15, d_\mu = 10, r = 5$ and total storage requirements of S = 5.7GB. Today, modern database systems are capable of efficient data handling in the range of TBs without significant performance penalty [Oracle Database 2012; MySQL Reference Guide 2012]. For larger networks, clustering techniques can be used to reduce the number of paths and so decrease the data storage requirements. Moreover, path computation and storage can be done offline before the actual network becomes operational. Since IPA operates on copies of the received packets, IPA has no functional impact on the application software and can run concurrently with it at the sink. IPA can be scheduled so as to only use processor cycles which are not used for the application. In this way, IPA has no impact on the latency of application processing at the sink.

4.5. Fault Detection and Identification

The sink performs FDI in two steps. The first step, Fault Detection (FD), identifies possible faults in the network by monitoring the NDB. The output FD is a list of possible faults, called the Suspect List (SL). The second step, Fault Identification (FI), uses this SL to localize and classify faults. It tests its hypothesis by sending control packets to nodes within the network. FI seeks to identify three types of failure, namely node failure, node reboot, and link failure.

4.5.1. Fault Detection. The overall FD algorithm is presented in Algorithm 3. FD consists of two steps: Persistent Path Change (PPC) detection and elimination.

On arrival of each packet at the sink and successful CRC checking, the sink searches the PAT for all records with matching SN IDs to obtain the path of the most recently received packet from that SN. If the path of the received packet (P_{new}) matches that of the most recently received previous packet from that SN (P_{old}) then there has been no path change, there is no evidence of a new fault, and FD moves on the next received packet. Otherwise, the sink temporarily stores the old path checksum C_{old} and the new path checksum C_{new} and places a watch on the SN ID for the next T_{th} time units where $T_{th} = mf$, where m is a multiplicative sensitivity factor (>1) and f is the sensing frequency of the SN (line number 5). After T_{th} time units, the sink checks whether any new packets have been received from the SN via the old path. If one or more packets have been received using the old path then the change was not persistent, there is no evidence of a new fault, and FD moves on to the next packet (line number 7, 8). Otherwise, a *persistent path change* has been detected (line number 13, 14).

Once a persistent path change has been detected, FD identifies the divergent node, the suspect node, and the suspect link (line number 17). As illustrated in Figure 1, the divergent node is the node which is the last common node in two paths from a single SN to the sink. The suspect node is the node immediately after the DN in the old path. The Suspect Link (SuL) is the link between the DN and the SuN.

Only one SuN and one SuL are considered since it is assumed that only one node or link failure can occur on a single path at a time. If multiple failures do occur simultaneously on a single path, only the failure closest to the SN will be detected. However, it is likely that the other failures will be detected later due to changes in other paths from different SNs.

The FD module retrieves the NAT record for the SuN and compares the `timestamp` to the current time. If the SuN has been active in the previous T_{th} time units then it must be used as part of a different path. As a consequence, the node must still be active and is eliminated as an SuN (i.e., `SuN=NULL`, line number 20, 21). If the SuN is eliminated, the `timestamp` of the first node in the SuL pair (i.e., the DN) is checked in the NAT. If both sides of the SuL have the same `timestamp` in the NAT then it is very likely that a single packet passed through both nodes and used the SuL. In this case, the SuL is also eliminated (i.e., `SuL=NULL`, line number 23). If processing time allows, the PAT entries of the NDB can be searched for the occurrence of the SuL in other paths within the last T_{th}. This final step is time consuming and is not used in the implementation described shortly.

ALGORITHM 3: Fault Detection

```
1:  Input variables
    :$SN$, Source Node
    :$P_{old}$, old path
    :$P_{new}$, new path
2:  Output variables
    : $SuN$, Suspect Node
    : $SuL$, Suspect Link
3:  Local variables
    : $TPacket$, to hold the value of total packets received from a specified SN
    :$Counter$, to count the number of packets
    :$PPC$, Persistent Path Change variable
4:  Initialize variables:
    $SuN = NULL$, $SuL = NULL$
    $TPacket = 0$, $Coutner = 0$, $PPC = False$
5:  **if** $P_{old} \neq P_{new}$ for the next $T_{th}$ time units **then**
6:    $TPacket ++$
7:    **if** $Packet_i \subseteq P_{old}$ **then**
8:      $Exit()$
9:    **else**
10:     $Counter ++$
11:   **end if**
12: **end if**
13: **if** $Counter == TPacket$ **then**
14:   set PPC=True
15: **end if**
    // Persistent Path Change occurs so, go to the next steps:
16: **if** $PPC = True$ **then**
17:   $SuN$=getSuspectNode($P_{new}$, $P_{old}$)
      $DN$=getDivergentNode($P_{new}$, $P_{old}$)
      $SuL$=[DN,SuN], link from DN to SuN
18: **end if**
    // Now go for elimination steps:
19: Consult $NAT$ entries of $NDB$
20: **if** $SuN$.status==active **then**
21:   set $SuN$=NULL
22:   **if** $SuN$.Timestamp==$DN$.Timestamp **then**
23:     set $SuL$=NULL
24:   **end if**
25:   Consult $PAT$ entries of $NDB$
26:   **if** $SuL \subseteq PAT$ **then**
27:     set $SuL$=NULL
28:   **end if**
29: **end if**
```

4.5.2. Fault Identification. The overall FI algorithm is presented in Algorithm 4. FI uses the values of DN and SuN obtained using Algorithm 3 as inputs. The algorithm consists of four steps.

The first step is to test for link failure. A `response_request` control packet is sent from the sink to the divergent node. The `response_request` includes two parameters: `req_avoid_node` and `resp_via_node`. Both parameters are set equal to the SuN ID. When the packet is routed to the target node, the RNs avoid sending it via the `req_avoid_node`. When the packet arrives at the DN, the DN responds by sending a `response` packet to the sink via the `resp_via_node` (line number 4). Beyond the

ALGORITHM 4: Fault Identification

```
1: Input variables
   :$SuN$, $DN$, these values are obtained from the Fault Detection algorithm
   :$SuL$=[DN,SuN]
   :$Q_{max}$, upper limit of neighbors to be selected
2: Output variables
   : status of $SuN$ and $SuL$
3: // Check for Link Failure first:
   response_request.Destination=$DN$
   response_request.avoid_node=$SuN$
   response_request.sendMessage()
4: response.Destination=$Sink$
   response.via_node=$SuN$
   response.sendMessage()
5: if $Sink.receivedResponse$==True then
6:   set $SuN.status$=Active
     set $SuL.status$=Active
     exit()
7: else
8:   Q=($d_i$ < $Q_{max}$)?$d_i$ : $Q_{max}$
     // getNeighbors(x,y) returns a maximum of y 1-hop neighbors of node x
     response_request.Destination=getNeighbors($SuN$,Q)
     response_request.avoid_node=$SuN$
     response_request.sendMessage()
9:   response.Destination=$Sink$
     response.via_node=$SuN$
     response.sendMessage()
10:  if $Sink.receivedResponse$==True then
11:    set $SuN.status$=Active
       set $SuL.status$=Failed
       exit()
12:  else
13:    //Check for Node Failure and Reboot:
       Wait upto $T_{reboot}$ time units
       response_request.Destination=$SuN$
       response_request.sendMessage()
14:    Wait upto $T_{resp}$ time units
15:    if $Sink.receivedResponse$==True then
16:      set $SuN.status$=Reboot
17:    else
18:      set $SuN.status$=Failed
19:    end if
20:  end if
21: end if
```

`req_avoid_node` and `resp_via_node`, the path to and from the target node is unspecified and is decided by the routing algorithm. After sending the `response_request`, the sink waits T_{resp} time units for the `response` packet to arrive. If the `response` packet is received within that time, the SuL and SuN are proven active and FI terminates (line number 6). If the packet is not received within that time, FI moves on to the second step.

The second step tests for node inactivity. A `response_request` packet is sent to all 1-hop neighbors of the SuN, excluding the neighbor (DN) already tested in the first step. Since, in some highly dense networks (i.e., for higher value of d_μ), the number of 1-hop neighbors may increase the message overhead, the algorithm restricts the total number of 1-hop nodes to a predefined maximum value Q_{max}. This is implemented in line 8. Again, the `req_avoid_node` and `resp_via_node` parameters are set equal to the SuN. The sink waits T_{resp} time units for the `response` packets to arrive. If one or more `response` messages are received within that time, the SuN is proven active and the SuL is proven to have failed. Link failure is reported and FI terminates (line number 11). If no packet is received within that time, the SuN is shown inactive and FI moves on to the third step.

The third step checks for node failure and node reboot. A final `response_request` packet is sent to the SuN after T_{reboot} time units. If no `response` is received at the sink within T_{resp} time units, the node is deemed to have suffered node failure (line 18). If a response is received within that time, the node is considered to have undergone a node reboot (line 16). In both cases, the FI result is reported.

The Fault Identification (FI) algorithm issues a maximum of $(Q + 2)$ control messages (i.e., in the case of node failures or reboots) from the Suspect Node (SuN) and its 1-hop nodes towards the sink in each round of fault detection. The value of Q is conditionally set as has already been explained. Therefore, the algorithm incurs complexity $O(1)$ which is independent of network parameters.

4.6. Low Data-Rate Network

SBFD detects faults in networks with periodic data transmission to the sink. In most applications [Beckwith et al. 2004; Szewczyk et al. 2004; Corke et al. 2010] data collection periods are short. However, in some applications the data reporting rate is very low. To cope with such scenarios, we employ the concept of a *heartbeat* message [Rost and Balakrishnan 2006]. At regular intervals, every SN computes the time difference T_{diff} between the current time T_{now} and the time it sent its last packet T_{last}. If the time difference T_{diff} goes beyond a predefined threshold T_{th} the node realizes that it did not send any data for a prolonged period. As a result, it sends a *heartbeat* message to the sink. The checksum field of the message is treated exactly in the same manner as a regular data packet. Algorithm 5 summarizes the process. The path deduction and fault identification processes are the same as described before.

ALGORITHM 5: Heartbeat Message

```
1: System variable:
   $T_{th}$, The predefined value of time threshold to verify "No traffic for long time"
2: $T_{diff} = T_{now} - T_{last}$
3: if $T_{diff} > T_{th}$ then
4:     heartbeatmsg.Destination=Sink
       heartbeatmsg.sendMessage()
       $T_{last} = T_{now}$
5: end if
```

5. EVALUATION

The efficiency and accuracy of the SBFD system was evaluated via a testbed implementation and in comprehensive simulations. A performance comparison with existing solutions was also conducted.

5.1. Implementation

SBFD was implemented in nesC, the language for TinyOS, an open-source operating system for WSNs [TinyOS 2010]. TinyOS version 2.1.1 and nesC compiler version 1.3.1 were used. The Collection Tree Protocol (CTP) was used in the routing layer. The Crossbow TelosB mote TPR2400 [Crossbow 2012] was used in the testbed for both

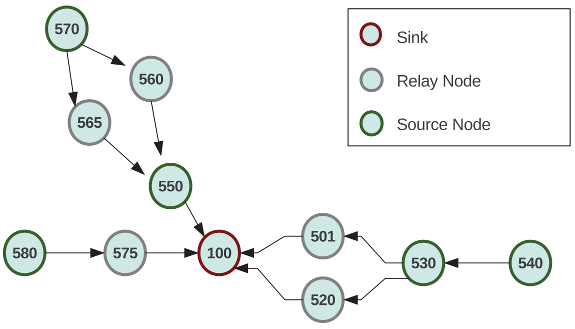


Fig. 4. Testbed network topology.

SNs and RNs. The TelosB mote has an 8 MHz microcontroller, RAM size of 10KB, flash memory of 48KB, and integrated temperature, humidity, and light sensors. The radio was a 2.4 GHz IEEE 802.15.4 Chipcon wireless transceiver with a data rate of 250kbps. The sink node was connected via USB to a PC server running Oracle Express Edition 10g, version 10.2.0.1.0, supporting the NDB. Linux 3.0.0-17-generic was used for the OS of the server. The Java-based `MessageReader` was used to capture packets from the radio network for processing by the NPA module. On reception of each packet, `MessageReader` posted it to the NDB. PPC detection was automated with the aid of a row-level trigger in the appropriate entries of the NDB.

5.2. Testbed Results

This section reports on the performance of SBFD in a real network testbed.

5.2.1. Accuracy. The accuracy of SBFD was evaluated using a small testbed network into which failures were manually inserted. Accuracy is defined as the ratio of the total number of faults detected correctly to the total number of faults inserted into the network. A network of 10 nodes was set up as shown in Figure 4. The network contained both SNs and RNs. The sink (Node ID 100) was located in the middle of the network. The SNs sensed temperature every 5 seconds and sent the data to the sink. The testbed ran for almost 50 minutes. On average each SN sent 569 data packets to the sink. For PPC detection T_{th} was set to 15 seconds.

All three failures types were manually inserted into the network. A node failure was inserted by removing the battery from a node. Pressing the reset button on a node was used to induce node reboot. To insert link failure, an indirect method was used. The node on the end of the link was physically moved so that it could not communicate with its previously neighboring node. A total of 10 failures were inserted into the network during operation.

Table I lists each fault event, the time at which it was inserted, and the consequences in terms of path change observed at the sink using SFBD. Figure 5 shows example variations in the path checksums receiver for four SNs with time. The corresponding paths and checksums are given in Table II. These observations indicate that the sink was successful in detecting Persistence Path Changes (PPC) for most of the network failures.

The accuracy of SFBD during the experiment is summarized in Table III. The overall accuracy achieved was 90%. Only 1 fault, a node reboot (at time 34), was not detected. This was due to the fact that RN 560 received packets from its neighboring SN 570 every 5 seconds. RN 560 rebooted successfully before a new packet arrived from the SN. As a result, no path change was not observed at the sink. The results indicate good accuracy for the method on a small network. Simulations were conducted to assess its performance on a large network, as detailed in Section 5.3.

Table I. Experimental Log (started 19:02, ended 19:51)

Time Elapsed (min)	Fault inserted	Path change detected
03	Node 520 failed	Path change occurred for both Node 530 & 540
10	Node 520 repaired & Node 501 rebooted	Path change occurred for both Node 530 & 540
17	Node 560 failed	Path change occurred for Node 570
25	Node 560 repaired & Node 565 moves out of radio range of Node 570 i.e. Link failed	Path change occurred for Node 570
28	Node 520 failed	Path change occurred for both Node 530 & 540
34	Node 560 rebooted	No path change observed
35	Node 565 was relocated & Node 560 failed	Path change occurred for Node 570
40	Node 501 moves out of radio range of Node 530 and was placed near Node 575 i.e. Link failed	Path change occurred for both Node 530 & 540
43	Node 550 failed	Path change occurred for Node 570
46	Node 575 failed	Path change occurred for Node 580
47	Node 530 & 540 exchanged location	Path change occurred for both Node 530 & 540 since Fletcher checksum is order-sensitive

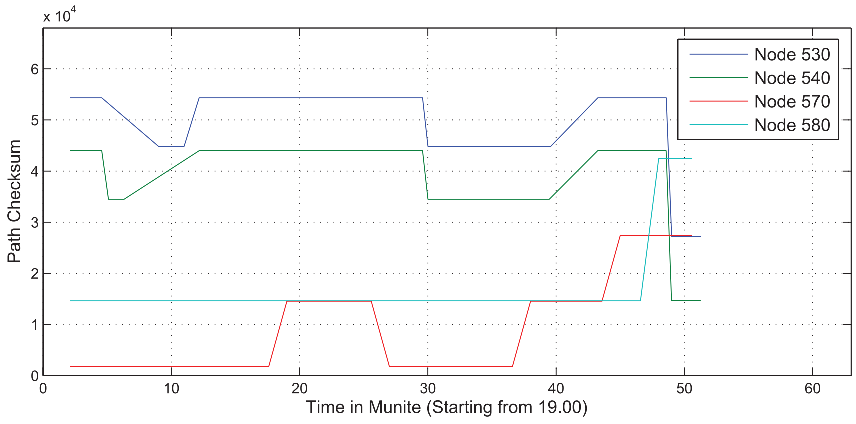


Fig. 5. Observed path checksum variation with time for 4 SNs.

Table II. Path Checksums

Source Node	Path	Path Checksum
530	530,520	54340
530	530,501	44849
530	530,540,520	27231
540	540,530,520	43971
540	540,520	14690
540	540,530,501	34480
570	570,565,550	14546
570	570,565,575	27371
570	570,560,550	1731
580	580,575	14610
580	580,501	42439

Table III. Testbed Fault Detection Accuracy

Fault Type	Number Inserted	Number Detected	Accuracy (%)
Node Failure	6	6	100
Link Failure	2	2	100
Node Reboot	2	1	50
Total	10	9	90

Table IV. Node Reboot Detection Accuracy Variation with Sensing Period

Sensing period	Number Inserted	Number Detected	Accuracy (%)
10 s	50	37	74
5 s	50	41	82
2.5 s	50	46	92
500 ms	50	48	96
250 ms	50	50	100

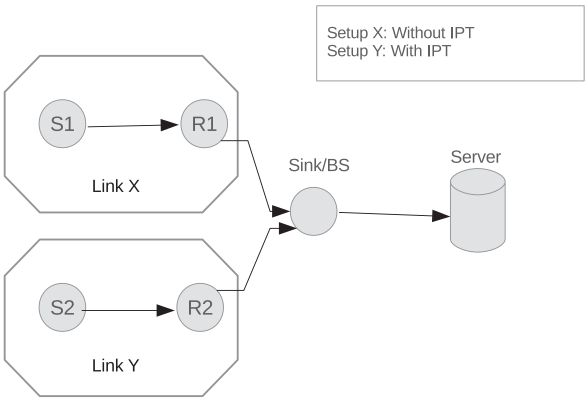


Fig. 6. Network setup for node lifetime evaluation.

To assess the relationship between the accuracy of node reboot detection and sensing frequency a further testbed experiment was performed. Four nodes were used: a SN and sink with two RNs placed between them. The SN could communicate with the sink via either RN. The sensing period of the SN was set to a specific value and node reboots were manually applied to the RNs. Reboots were inserted individually and the node affected was selected randomly. The experiment was repeated 5 times with each round taking 15 minutes and the results averaged. The experiment was repeated for sensing periods from 10 s to 250 ms. Table IV presents the experimental results. We manually measured the reboot time as 5 seconds. Clearly, accuracy of detection improves as the sending period decreases. Specifically, the accuracy is just greater than 80% when the sensing period is equal to the reboot time.

5.2.2. Impact on Node Performance. This section describes experiments conducted to determine SBFD's impact on node lifetime and its memory requirements.

Impact on Relay Node (RN). A network was set up as shown in Figure 6. The SNs and RNs were placed 30m apart. In Link X, the RN (R1) does not perform IPT. In Link Y, the RN (R2) does perform IPT. Environmental temperature and node internal voltage-level data were collected by the SNs (S1 and S2) and sent to the BS via their respective RNs (R1 and R2) every 2 seconds. The packet payload lengths were 10 and 12 bytes in Link X and Y, respectively. The nodes were loaded with fresh batteries and

Table V. Relay Node (RN) Lifetime

Node	Lifetime (in packets)	Lifetime (in seconds)	Lifetime w.r.t. R1 (%)
R1 (without IPT)	210,871	421,742	100.0
R2 (with IPT)	209,903	419,806	99.5

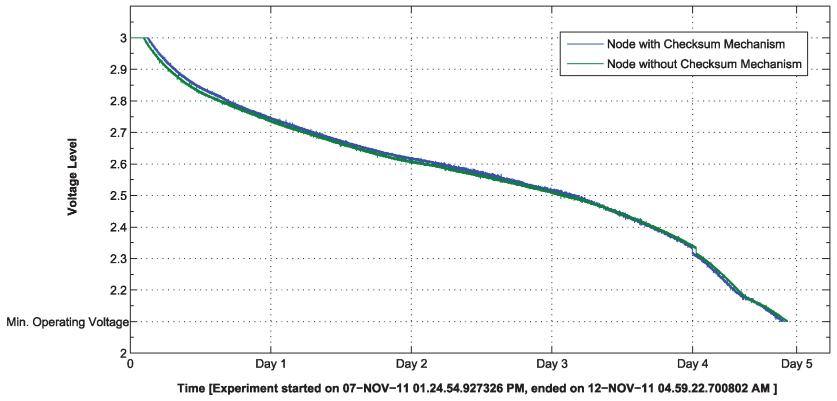


Fig. 7. RN internal voltage-level variation with time (min. operating voltage = 2.1V).

the network was run until the interval voltage level of a RN went below its minimum operating voltage (2.1V) [Crossbow 2012]. The collected packets were then analyzed to evaluate the impact of IPT on node lifetime. Since all nodes operated under identical conditions, any variations in the number of packets processed or voltage levels were a direct result of the processing and transmission overhead of IPT.

The lifetimes of the links are summarized in Table V. Link X and Y processed 210,871 and 209,903 packets, respectively, before their associated RNs ran out of energy. Overall, Link Y forwarded only 0.46% less packets than Link X. The internal voltage levels of both RNs are plotted against time in Figure 7. The RN voltage levels show almost identical trends. These results show that IPT has very low overhead in terms of lifetime reduction.

Impact on Dedicated Relay Node (DRN). A second network was deployed using the same topology as shown in Figure 6. In this case, the SNs' (i.e., S1 and S2) power supply was from the mains. The intermediate nodes (i.e., R1 and R2) were simply Dedicated Relay Nodes (DRN), that is they did not perform sensing. Since the SNs could not directly transmit packets to the sinks, a persistent discontinuation of packet reception from a specific SN at the sink indicated the intermediate DRN's death due to energy depletion. We conducted 3 rounds of experiments. In each round the DRNs were loaded with fresh batteries and the lifetime of the network was assessed in terms of the number of packets received at the sink. The results are shown in Table VI. The number of data packets processed by each network was higher than in the RN case (i.e., Table V). The results show that the average node lifetime penalty due to IPT was 0.74% in terms of the number of packets processed by the DRNs. It is important to note that the impact of IPT is similar in both the RN and DRN cases. This shows that IPT incurs very low overhead and does not significantly reduce network lifetime.

Impact on Node Memory. To assess the impact of IPT on node memory, two separate versions of a program for data collection were written and installed in two nodes. The only difference between the versions was that one supported IPT and the other did not. Both programs used CTP as the multihop routing protocol. As shown in Table VII,

Table VI. Dedicated Relay Node (DRN) Lifetime

Node	Lifetime (in packets)	Lifetime (in seconds)	Lifetime w.r.t. R1 (%)
R1 (without IPT)	269,258	538,515	100.0
R2 (with IPT)	267,255	534,509	99.26

Table VII. SBFD Memory Size

	ROM size (in bytes)	Increase (in bytes)	Increase (%)	RAM size (in bytes)	Increase (in bytes)	Increase (in %)
Node without IPT	30,254	–	–	1,910	–	–
Node with IPT	30,558	304	1.01	1,912	2	0.11

Table VIII. Simulation Parameter Settings

Parameter	Value
PHY Layer	2.4GHz IEEE 802.15.4
MAC Layer	Default CSMA
Clear Channel Assessment	−70dbM
Routing Layer	CTP
External Noise	Added (Mayer Library)
Network Size	Variable (25 to 400)
SN ratio α	.5
No. of BS/Sink	1 (ID=1)
Sensing Period f	250 ms
T_{th}	750 ms
m	3
r	3
Q_{max}	5
Sim. Duration	100 s
No. of Iterations	3

including IPT was found to increase RAM size by 2 bytes (0.1%) and ROM size by 304 bytes (1.01%). It is clear that SBFD has negligible impact on node memory size.

5.3. Simulation Results

This section considers the performance of SBFD in large networks.

For the purposes of experimentation, the network testbed was replaced by TOSSIM, a discrete event network simulator which compiles programs directly from TinyOS code [Levis et al. 2003]. The real sensing data was replaced by a dummy sensor function. The simulation parameters are listed in Table VIII. The default values of all MAC-layer parameters were used with the exception of `maxIterations` which was initialized to 10 since the default value (0) sets the number of radio back-offs to infinity. A noise trace collected from Meyer Library, Stanford University was used to model RF noise and interference. The topology generator and the fault modeler were written in Python 2.7.

5.3.1. Accuracy. A program was written to generate random network topology files. The networks were generated for various combinations of number of nodes N, network length h, and mean connectivity d_μ. The parameters for the sparse networks are listed in Table X and those for dense networks are listed in Table IX. In all cases, the sink node was placed in the middle of the network with ID 1. Node IDs were then allocated sequentially. Nodes with even IDs were designated as SNs. Nodes with odd IDs were designated as DRNs. Hence, the SN ratio α was 0.5.

Table IX. Parameters: Dense Networks

Network Size (N)	MaxHop (h)	Mean Connectivity (d_μ)
40	5	2.8
75	6	3.1
150	8	3.85
200	10	7.2
300	10	9.3
400	10	10.5

Table X. Parameters: Sparse Networks

Network Size (N)	MaxHop (h)	Mean Connectivity (d_μ)
25	5	1.5
50	6	1.7
100	8	1.85
150	10	1.91
200	10	1.98
250	10	2.1

Table XI. Detection Accuracy: Sparse Topologies

Network Size (N)	Detection Accuracy (%)			
	Node Failure	Link Failure	Node Reboot	Overall
25	88.8	88.8	100	91.1
50	93.3	93.3	88.8	92.4
100	90	93.3	86.6	90.6
150	93.3	91.1	87.5	91.2
200	90	95	93.3	92.6
250	90.6	93.3	92.3	92.1

In each simulation, the number of faults injected was set as a percentage of the network size N. Experience from WSN deployments, such as Arora et al. [2004] and Beckwith et al. [2004], suggests that node failure and link failure are more likely than node reboot. Hence, over the course of the simulation, 10% of nodes suffered node failures, 10% of links suffered link failures, and 5% of nodes suffered node reboots. The timing of each fault was determined randomly. Three rounds of simulation were performed for each topology and the results were averaged. SNs sent a packet to the sink periodically every 250 milliseconds. The simulation duration was 100 seconds. On average, each SN transmitted 395 packets to the sink. The network experienced a Packet Drop Rate (PDR) of around 4.2% because of the noise modeled in the simulation. SBFD was applied, as described in Section 4.5 with $m = 3$ and $T_{th} = 750$ ms.

Tables XI and XII and Figure 8 report the accuracy of fault detection. Overall detection accuracy is expressed as a weighted average taking into consideration the probability of each fault type.

We consider the results for the sparse topologies first. Detection accuracy for node failure was bounded between 88.8% and 93.3% and was unaffected by network size. Even for large values of N (i.e., 250), accuracy was greater than 90%. The same trend was noticeable for both link failure and node reboot faults. The overall accuracy was close to 91.5% for all values of N. The high accuracy of SBFD, regardless of network size, is due to the distributed nature of IPT and the localization of faults using control messages in the FI step.

Detection accuracy for dense topologies was higher for all types of faults. In the case of node failures, for instance, accuracy ranged between 91.6% and 97.5%. As for sparse

Table XII. Detection Accuracy: Dense Topologies

Network Size (N)	Detection Accuracy (%)			
	Node Failure	Link Failure	Node Reboot	Overall
40	91.7	100	83.3	93.3
75	91.6	95.8	91.6	93.3
150	93.3	95.6	91.7	93.9
200	96.7	95.0	86.7	94.0
300	95.6	94.4	91.1	94.2
400	97.5	94.2	91.7	95.0

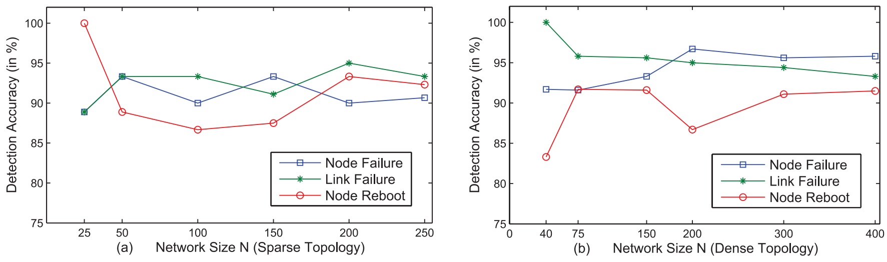


Fig. 8. Detection accuracy: (a) sparse topologies; (b) dense topologies.

topologies, the accuracy in dense networks was not affected by network size. Figures for overall accuracy were close to 93.9%, giving an average improvement of 2.3% relative to the results for the dense topologies. The improvement is due to the higher number of alternate routes in the dense networks which allows for better fault discrimination using control messages.

Two further factors were found to impact on overall accuracy. First, since the topology was built randomly there were some idle nodes and idle links, especially at the network edge. Clearly, failures in idle nodes and links cannot be detected based on persistent path change. In the case of fixed rate transmissions from the SNs to the sink, idle SNs can be detected by checking the NAT for SNs with timestamps greater than the transmission period. This technique could be applied in the simulation but was not used since, to allow more general usage, the algorithm assumes that the SN packet transmission rate is not known a priori (see Section 3). Alternatively, the concept of periodic heartbeat message as described in Section 4.6 could be exploited to detect these idle nodes. We did not apply this concept in simulation since SBFD is primarily designed for WSN applications with frequent data reporting. Second, loss of control messages and responses impacts on detection accuracy. Thus detection accuracy is somewhat dependent on the average network Packet Drop Ratio (PDR). To investigate this further, we performed a series of simulations with a fixed network (dense $N = 150$; see Table IX) and increasing workload, so as to introduce congestion and increase PDR. Simulation duration, noise injection, and network protocols were identical to the previous simulations. Workload W was defined as

$$W = N\frac{1}{T_s}\alpha, \tag{4}$$

where T_s is the sensing period.

The workloads used in the simulations are listed in Table XIII. PDR and overall fault detection accuracies were measured and are plotted in Figure 9. The figure shows that higher network traffic leads to higher PDR which reduces accuracy. Initially, PDR was

Table XIII. Workload Parameters

Network Size (N)	SN Ratio (α)	Sensing Frequency (T_s) in ms	Workload (W)
150	0.50	250	0.30
150	0.60	150	0.60
150	0.70	100	1.05
150	0.75	75	1.50
150	0.80	50	2.40

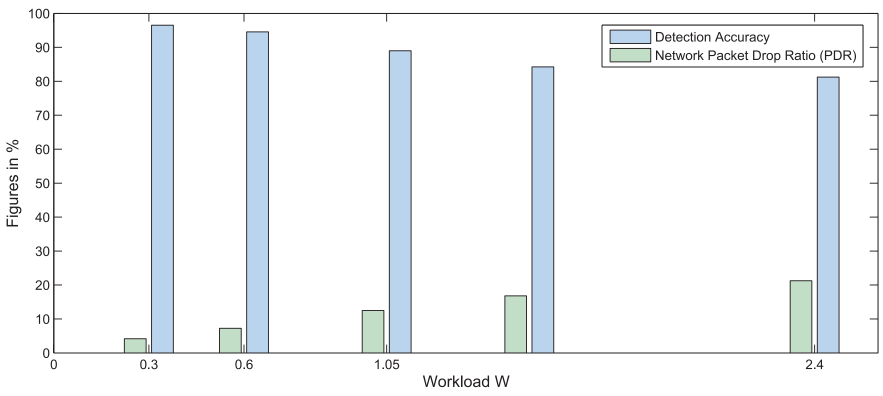


Fig. 9. Overall fault detection accuracy under varied workload (network size N = 150, dense topology).

4.1% and accuracy was 96.5%. A PDR of 21.2% is associated with an accuracy of 81.2%. This result indicates that SBFD's accuracy also depends on the routing performance of the network.

5.3.2. Message Overhead and Detection Latency. In Section 5.2.2, we considered the impact of IPT on node lifetime. The network overhead of FI is also affected by the number of control messages sent by the sink when suspect nodes and links are detected. This section evaluates the impact of SBDF in terms of additional message exchange for the purpose of fault detection.

The number of control messages sent by the sink is presented in Tables XIV and XV for the sparse and dense networks, respectively. Two scenarios are considered: a network simulation with zero faults and a simulation with faults injected as described previously.

When there are no faults, SBFD sends few control messages because the false positive rate at the Fault Detection (FD) stage is typically low. Occasionally false positives are due to a node's parent change because it discovered a better parent in terms of connectivity. This is due a greedy approach in the underlying routing protocol. These unnecessary control messages constitute an average overhead of 0.01% and 0.04% in terms of network traffic for sparse and dense networks, respectively, as shown in Figure 10. After subtracting this baseline activity, SFDB, on average in the faulty network simulation, sends 2.3 and 3.3 control messages per fault detected in sparse and dense networks, respectively. The number is higher for dense networks since the FD algorithm generally issues d_μ control messages to detect node failure and reboot faults. For higher values of d_μ the algorithm restricts it to a maximum of Q_{max}.

In the presence of faults, the control message overhead was 0.164% and 0.239% of the total data traffic at the SN for the sparse and dense networks, respectively. Overall, SFDB incurs very low overhead in terms of additional network traffic.

Table XIV. Message Overhead of SBDF in Sparse Networks

		Zero Fault Scenario		Faulty Scenario		
Network Size (N)	Total Data Packets received by the Sink	Control Packets sent	Control Packets sent (%)	Control Packets sent	Control Packets sent (%)	Mean Control Packets per Fault
25	9875	2	0.021	19	0.192	2.4
50	19750	2	0.01	31	0.157	2.2
100	39500	4	0.01	61	0.154	2.2
150	59250	4	0.006	95	0.160	2.4
200	79000	8	0.012	126	0.159	2.3
250	98750	8	0.008	163	0.165	2.4
Average:	–	–	0.01	–	0.164	2.3

Table XV. Message Overhead of SBDF in Dense Networks

		Zero Fault Scenario		Faulty Scenario		
Network Size (N)	Total Data Packets received by the Sink	Control Packets sent	Control Packets sent (%)	Control Packets sent	Control Packets sent (%)	Mean Control Packets per Fault
40	15800	3	0.019	32	0.203	2.9
75	29625	6	0.020	60	0.203	2.7
150	59250	19	0.032	135	0.228	3.1
200	79000	30	0.038	214	0.271	4.9
300	118500	35	0.030	319	0.269	6.3
400	158000	50	0.032	414	0.262	6.7
Average:	–	–	0.028	–	0.239	3.3

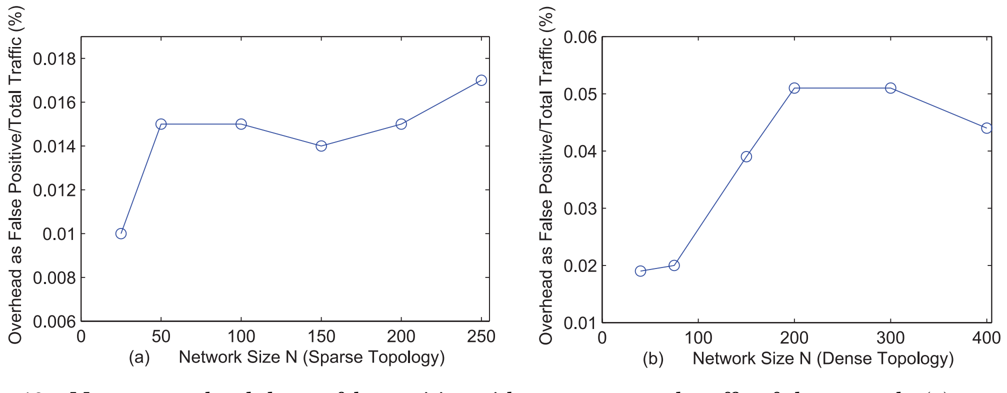


Fig. 10. Message overhead due to false positive with respect to total traffic of the network: (a) sparse, (b) dense.

The latency of fault detection in SBFD was also assessed. It is primarily bounded by $T_{th} + T_{search}$ where T_{th} is described in Section 4.5 and T_{search} is the processing delay at the sink which is dominated by the NDB lookup during NPA. Since $T_{th} = mf$, the value of T_{th} largely depends on the sensing frequency which is typically set according to application requirements. NDB lookup delay was measured for the first 100 packets sent by 5 randomly selected SNs in the dense network simulation with $N = 400$. The size of the NDB for this network was 100.4MB. The Empirical Cumulative Distribution Funciton (ECDF) of the search time is shown in Figure 11. In over 80% of cases, the delay was in the range of 1 ms to 5 ms. The results indicate that fault detection

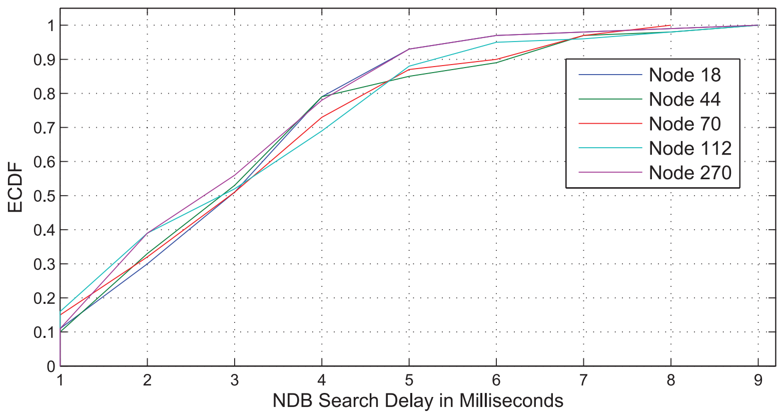


Fig. 11. Delay incurred in NDB search for the first 100 packets received in the dense network, $N = 400$.

latency in SBFD is not significantly affected by the NDB search delay even for a large network.

5.4. Comparative Study

This section presents a performance comparison between SBFD and other state-of-the-art proposals for fault detection in WSNs. Comparison was performed by means of network simulation using TOSSIM [Levis et al. 2003], and using the simulation parameters in the Table VIII and the fault model described in Section 5.3. SBFD was compared with Sympathy [Rost and Balakrishnan 2006] and PAD [Liu et al. 2010]. Sympathy is based on the principal that there is a direct correlation between the amount of data collected from nodes and the presence of failures in the network. PAD, on the other hand, detects faults based on mismatches between expected and deduced paths for each data packet. Detection accuracy was used as a metric.

Sympathy and SBFD. Figure 12 shows the results for the dense topology with varying network size N and SN ratio α. In Figure 12(a) with $\alpha = 1$, the overall detection accuracy of both schemes were similar for $N \leq 150$, that is, 84.2% and 87.5% for Sympathy and SBFD, respectively. With $N > 150$, Sympathy accuracy falls to around 80%. This is because large dense networks offer a larger number of alternate paths to the sink. If one link or node fails, the routing engine uses an alternative path without any noticeable difference in the quantity of the received data. As a result, some link and node failures were not correctly detected using Sympathy. In Figures 12(b), (c), and (d) the number of DRNs is increased by reducing the value of α. This was investigated because in many real deployments the network contains SNs, RNs, and DRNs. As α decreases, Sympathy accuracy reduces to 81.2%, 78.1%, and 75.7%. This is 8.3%, 13.9%, and 16.7% less than SBFD. This is due to the working principal of Sympathy, it detects faults based on significant deviations in expected traffic. Thus, provided there are alternative paths, it is difficult to detect faults for DRNs in Sympathy. Figure 13 shows the results for sparse networks. In Figure 13(a), with $\alpha = 1$, the overall accuracy of Sympathy was 83.4% which is 2.3% less than SBFD. In the sparse topology, nodes do not have sufficient alternate paths to the sink which makes failure detection more accurate in Sympathy. Lower values of α reduce Sympathy accuracy. For $\alpha = .25$, for instance, the accuracy of Sympathy reduces to 76.1% which is 13.4% lower than SBFD.

PAD and SBFD. We investigated the performance of PAD and SBFD with varying network size N and SN ratio α. Since PAD has no mechanism to detect node reboot, we eliminated it from our fault model. As PAD supports tree-based routing we used

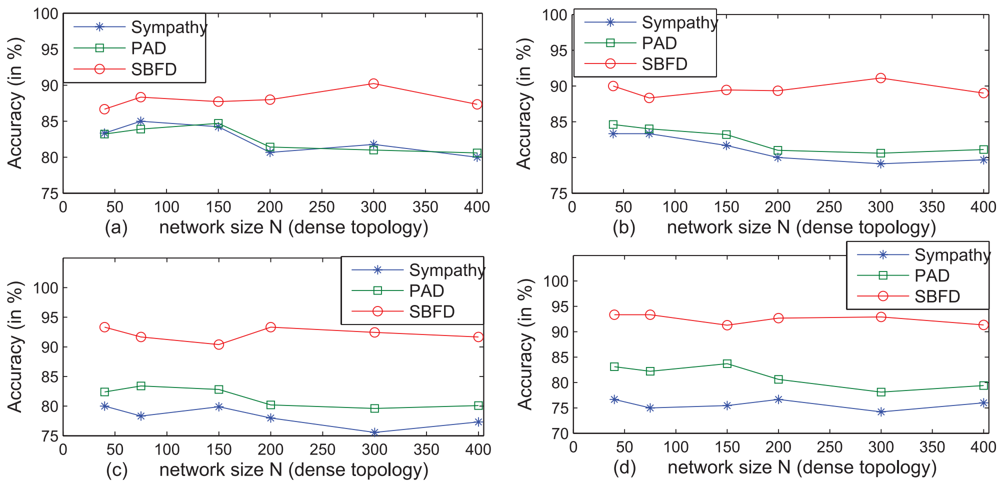


Fig. 12. Fault detection accuracy in the dense network, comparative study with Sympathy and PAD: (a) $\alpha = 1$; (b) $\alpha = .75$; (c) $\alpha = .5$; (d) $\alpha = .25$.

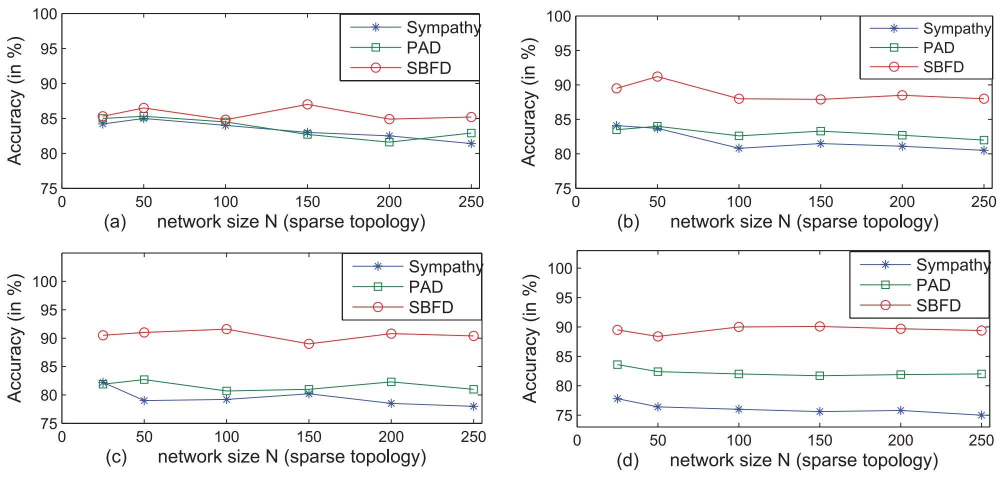


Fig. 13. Fault detection accuracy in the sparse network, comparative study with Sympathy and PAD: (a) $\alpha = 1$; (b) $\alpha = .75$; (c) $\alpha = .5$; (d) $\alpha = .25$.

CTP in the routing layer of both schemes. PAD introduces a new connection variable (i.e., C_i) for detecting failures of DRNs. The results are shown in Figures 12 and 13. Both figures show that the impact of α is not significant for PAD. For instance, the overall accuracy of PAD was 82.47% with $\alpha = 1$ whereas it was 80.84% with $\alpha = .25$ as shown in Figures 12(a) and (d), respectively. The overall accuracy of PAD in dense and sparse networks was 81.7% and 82.6%, respectively, which were 8.7% and 6% lower than SBFD. Path deduction in PAD incurs significant latency due to the synchronized, serial packet marking scheme. When a new path is initiated, PAD often fails to detect failures if the nodes or links fail before the sink has finished path deduction.

We also assessed the performance of PAD and SBFD in the presence of packet drops for a single network (i.e., dense $N = 180$). We varied the workload W to introduce packet drops in the network (see Table XIII). Figure 14 shows the results obtained. With lower PDR (i.e., below 5%) the overall accuracy of PAD and SBFD was 86% and 92.8%,

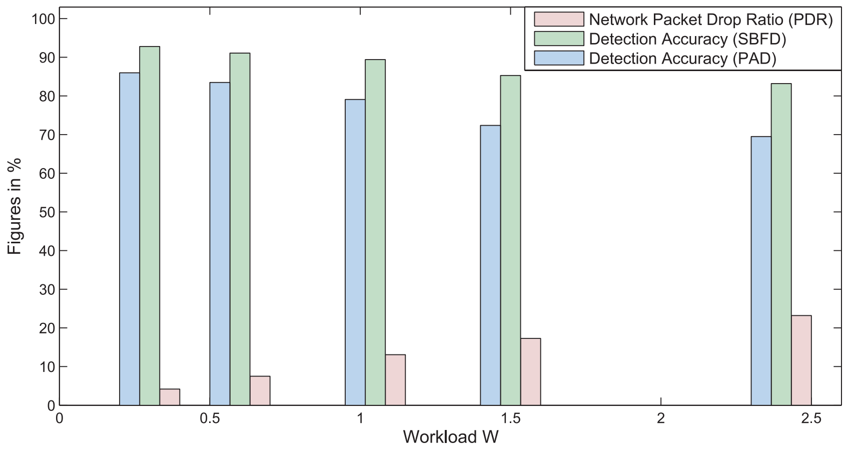


Fig. 14. Fault detection accuracy in the dense network, $N = 150$, comparative study with PAD.

Table XVI. Path Deduction Latency

		Latency in ms
PAD	Maximum	1750
	Minimum	250
	Average	1085
SBFD	Maximum	4
	Minimum	1
	Average	2.54

respectively. When PDR fell below 12% the accuracy of PAD significantly reduced to 79.1% while it was 89.4% in SBFD. Above this, SBFD maintained an average of 12.3% greater accuracy than PAD. The main factor that influences the overall performance of PAD is the order of each packet being maintained. In most WSN applications, the network intermittently experiences out-of-order packet delivery due to unavoidable factors such as congestion, failures of nodes/links. PDR also impacts on the accuracy of SBFD which has been discussed in Section 5.3.1. The results indicate that SBFD is more robust to packet drops than PAD.

We compared the latency of path deduction in both schemes. The results are shown in Table XVI. The average latencies were 1085 ms and 2.54 ms for PAD and SBFD, respectively. Packet marking in PAD incurs higher delay since it is directly dependent on hop distance h and sensing frequency f. In this simulation the average hop distance h of the selected SNs was 5.3 and the value of f was 250 ms. In PAD, the sink must wait for an additional $(h - 1)$ packets after receiving the first packet from an SN to deduce the observed path. In SBFD, on the other hand, the path deduction latency is primarily dependent on NDB lookup time. The results indicate that SBFD incurs very low path deduction latency in comparison with PAD.

6. RELATED WORK

To the best of the authors' knowledge, SBFD is the first proposal which exploits a lightweight tagging mechanism and server-side storage-intensive computation to deterministically extract the packet path and to utilize this information for efficient and accurate detection of network faults.

Considerable work been conducted on fault detection for enterprise networks. Commercial tools, such as HP Openview [HPO 2007] and IBM Tivoli [IBM 1996], independently monitor servers and routers by means of periodic message exchange. These systems are very effective for large-scale enterprise networks but are not suitable for WSNs since the tools are designed to operate on high-bandwidth, mains-powered equipment.

A number of research papers have considered the use of bipartite graphs for enterprise network fault detection. In Kandula et al. [2005], the authors propose Shrink which uses a probabilistic inference model based on a bipartite dependency graph which assesses dependency between network components. The authors of Steinder and Sethi [2002] evaluate the relationships between links to identify faults using a combination of bipartite graphs and a belief network. The major limitation of these solutions in the context of WSNs is that they require complete a priori information on the dependencies in the network, which is not feasible in a sensor network. Furthermore, they are very expensive in terms of computation and storage.

Most previous research on fault detection in WSNs uses the proactive approach whereby either a special node or debugging agent is deployed to collect information and periodic control messages are transmitted to the sink. In the paper on Sympathy [Ramanathan et al. 2005], the authors propose a predeployment architecture that periodically logs communication statistics, such as the routing table, number of packets, and the level of congestion. Sympathy is based on the principal that there should be a predictable relationship between these data statistics and the number of faults in the network. Scanning residual energy to monitor the status of nodes has been investigated by many researchers, for example, Zhao et al. [2002]. In Memento [Rost and Balakrishnan 2006] every node periodically sends a heartbeat message to its parent. In order to reduce overall traffic, it uses aggregation to summarize messages. In Khan et al. [2008], the authors propose a debugging framework, Dustminer, that identifies *culprit sequences* and uses them to pinpoint failures. Dustminer mainly targets soft faults in the protocol stack. In a recent publication, the authors of Khan et al. [2010] propose a remote monitoring system, Powertracer, which determines the internal health of unresponsive nodes in the network. It can effectively classify faults including node failure, link failure, and frequent node reboot. The downside of Powertracer is the additional requirement for a wireless power meter in every node. For a large network, the cost of deploying Powertracer may be significant. All of these methods involve periodic message exchange (either for control or statistic passing). Thus they incur a heavy penalty in terms of lifetime reduction in resource-constrained WSNs.

Recently, passive information collection has been seen as a lightweight approach to failure detection in WSNs since it extracts information on network behavior with low overhead. LiveNet [Chen et al. 2008] employs several *sniffer nodes* distributed throughout the network. Each sniffer node collects network metadata. The trace files collected are merged to provide a global view of the status of the network. Deploying sniffer nodes in the network makes this scheme hard to accommodate in many applications. The concept of a sniffer node for network data collection has also been investigated by the designers of PDA [Romer and Ma 2009]. PDA presents various options for trace collection, including an offline and an online approach. In the offline approach, the sniffer nodes collect both network information and application data. In the online scheme, a separate radio channel is established for the sniffer nodes. PDA has the major drawback of high message overhead and its scalability is not assessed in the paper.

A recent publication related to SBFD is PAD [Liu et al. 2010]. PAD is founded upon the concept of packet marking to deduce the sequence of relay nodes. A comparison in terms of fault detection accuracy and path deduction latency is presented in Section 5.4. We distinguish our approach from PAD in the following ways. First, the packet marking scheme in PAD appends two fields to regular data packets: stamping node ID (2 bytes) and hop distance (2 bytes), incurring a total overhead of 4 bytes per packet. IPT in SBFD incurs a 2-bytes overhead for storing the checksum value. Second, the PAD marking algorithm requires synchronization between all nodes in a path from source to sink. Synchronization must be maintained by the sink to deduce the path from the marking scheme. This requirement poses additional complexity, particularly when a relay node processes multiple packets from different sources within the same time slot. Third, to deduce a path using PAD the sink must receive *k* consecutive packets from the same SN, where *k* is the number of hops between the SN and sink. Thus PAD experiences higher latency in detecting a path change which, in turn, affects the overall accuracy.

7. CONCLUSIONS AND FUTURE WORK

There have been a number of interesting publications on fault detection in WSNs. However most existing methods either reduce network lifetime due to additional message exchange, or suffer from poor accuracy due to lack of up-to-date view of activity in the network. In this article we propose a fault detection framework named SBFD which is lightweight, accurate, and scalable.

The framework was implemented in a testbed network. The accuracy of the framework was measured and found to be 90% in testbed experiments. The impact of path tagging on sensor node lifetime was found to be 0.50% and additional memory requirements were less than 1.2%. Extensive simulation using a variety of network parameters was performed to assess the method's scalability. Detection accuracy in simulation varied from 90.7% to 92.7% for sparse networks while for dense networks accuracy ranged from 93.3% to 95.0%. Accuracy was found be almost constant with increasing network size. The overhead in terms of control messages was found to be 2.23 and 3.32 control packets on average per fault for sparse and dense networks, respectively. Finally we conducted a comparative study with existing solutions.

In future work, we plan to extend the basic concept to analyze network behavior during network operation, including deduction of possible reasons for failures, methods for sensor hotspot detection, and evaluation of routing protocols.

REFERENCES

A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Gouda, Y. Choi, T. Herman, S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora, and M. Miyashita. 2004. A line in the sand: A wireless sensor network for target detection, classification, and tracking. *Comput. Netw.* 46, 605–634.

R. Beckwith, D. Teibel, and P. Bowen. 2004. Unwired wine: Sensor networks in vineyards. In *Proceedings of the IEEE Conference on Sensors.* 561–564.

P. Buonadonna, D. Gay, J. M. Hellerstein, W. Hong, and S. Madden. 2005. Task: Sensor network in a box. In *Proceedings of the European Workshop on Sensor Networks*. 133–144.

B.-R. Chen, G. Peterson, G. Mainland, and M. Welsh. 2008. Livenet: Using passive monitoring to reconstruct sensor network dynamics. In *Proceedings of the 4th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS'08).* Springer, 79–98.

P. Corke, T. Wark, R. Jurdak, W. Hu, P. Valencia, and D. Moore. 2010. Environmental wireless sensor networks. *Proc. IEEE* 98, 11, 1903–1917.

Crossbow. 2012. Data sheet from crossbow. http://www.xbow.com/Products/productdetails.aspx?sid=252.

J. Fletcher. 1982. An arithmetic checksum for serial transmissions. *IEEE Trans. Comm.* 30, 1, 247–252.

O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. 2009. Collection tree protocol. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys'09).* ACM Press, New York, 1–14.

S. Guo, Z. Zhong, and T. He. 2009. Find: Faulty node detection for wireless sensor networks. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys'09).* ACM Press, New York, 253–266.

T. He, S. Krishnamurthy, L. Luo, T. Yan, L. Gu, R. Stoleru, G. Zhou, Q. Cao, P. Vicaire, J. A. Stankovic, T. F. Abdelzaher, J. Hui, and B. Krogh. 2006. Vigilnet: An integrated sensor network system for energy-efficient surveillance. *ACM Trans. Sens. Netw.* 2, 1, 1–38.

HPO. 2007. http://www.openview.hp.com.

IBM Tivoli. 1996. http://www.ibm.com/software/tivoli.

S. Kandula, D. Katabi, and J.-P. Vasseur. 2005. Shrink: A tool for failure diagnosis in ip networks. In *Proceedings of the ACM SIGCOMM Workshop on Mining Network Data (MineNet'05).* ACM Press, New York, 173–178.

M. M. H. Khan, H. K. Le, H. Ahmadi, T. F. Abdelzaher, and J. Han. 2008. Dustminer: Troubleshooting interactive complexity bugs in sensor networks. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems (SenSys'08).* ACM Press, New York, 99–112.

M. M. H. Khan, H. K. Le, M. Lemay, P. Moinzadeh, L. Wang, Y. Yang, D. K. Noh, T. Abdelzaher, C. A. Gunter, J. Han, and X. Jin. 2010. Diagnostic powertracing for sensor node failure analysis. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN'10).* ACM Press, New York, 117–128.

L. Krishnamurthy, R. Adler, P. Buonadonna, J. Chhabra, M. Flanigan, N. Kushalnagar, L. Nachman, and M. Yarvis. 2005. Design and deployment of industrial sensor networks: Experiences from a semiconductor plant and the north sea. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SenSys'05).* ACM Press, New York, 64–75.

K. Langendoen, A. Baggio, and O. Visser. 2006. Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture. In *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS'06).*

P. Levis, N. Lee, M. Welsh, and D. Culler. 2003. Tossim: Accurate and scalable simulation of entire tinyos applications. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys'03).* ACM Press, New York, 126–137.

Y. Liu, K. Liu, and M. Li. 2010. Passive diagnosis for wireless sensor networks. *IEEE/ACM Trans. Netw.* 18, 4, 1132–1144.

MySQL Reference Guide. 2012. MySQL reference manual for 5.5 version. http://dev.mysql.com/doc/refman/5.5/en/.

Oracle Database. 2012. Oracle database performance tuning guide,11g release 1 (11.1). http://www.oracle.com/pls/db111/portal.portal_db?selected=17&frame.

D. Puccinelli and M. Haenggi. 2010. Reliable data delivery in large-scale low-power sensor networks. *ACM Trans. Sen. Netw.* 6, 28:1–28:41.

N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin. 2005. Sympathy for the sensor network debugger. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SenSys'05).* ACM Press, New York, 255–267.

K. Romer and J. Ma. 2009. Pda: Passive distributed assertions for sensor networks. In *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN'09).* 337–348.

S. Rost and H. Balakrishnan. 2006. Memento: A health monitoring system for wireless sensor networks. http://nms.lcs.mit.edu/papers/memento-secon-2006.pdf.

M. Steinder and A. Sethi. 2002. Increasing robustness of fault localization through analysis of lost, spurious, and positive symptoms. In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'02).* Vol. 1. 322–331.

R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler. 2004. An analysis of a large scale habitat monitoring application. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys'04).* ACM Press, New York, 214–226.

TinyOS. 2010. TinyOS documentation. http://docs.tinyos.net/index.php/Main_Page.

M. Wachs, J. I. Choi, J. W. Lee, K. Srinivasan, Z. Chen, M. Jain, and P. Levis. 2007. Visibility: A new metric for protocol design. In *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems (SenSys'07).* ACM Press, New York, 73–86.

Y. Zhao, R. Govindan, and D. Estrin. 2002. Residual energy scan for monitoring sensor networks. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC'02).* Vol. 1. 356–362.

Received May 2012; revised March 2013; accepted April 2013