Hindawi Publishing Corporation
Journal of Computer Networks and Communications
Volume 2013, Article ID 970597, 9 pages
http://dx.doi.org/10.1155/2013/970597

Research Article

DS$^+$: Reliable Distributed Snapshot Algorithm for Wireless Sensor Networks

Gamze Uslu, Kemal Cagri Serdaroglu, and Sebnem Baydere

Department of Computer Engineering, Yeditepe University, Istanbul, Turkey

Correspondence should be addressed to Sebnem Baydere; sbaydere@cse.yeditepe.edu.tr

Received 29 August 2012; Revised 9 November 2012; Accepted 1 December 2012

Academic Editor: Yueh M. Huang

Acquiring the snapshot of a distributed system helps gathering system related global state. In wireless sensor networks (WSNs), global state shows if a node is terminated or deadlock occurs along with many other situations which prevents a WSN from fully functioning. In this paper, we present a fully distributed snapshot acquisition algorithm adapted to tree topology wireless sensor networks (WSNs). Since snapshot acquisition is through control messages sent over highly lossy wireless channels and congested nodes, we enhanced the snapshot algorithm with a sink based reliability suit to achieve robustness. We analyzed the performance of the algorithm in terms of snapshot success ratio and response time in simulation and experimental small test bed environment. The results reveal that the proposed tailor made reliability model increases snapshot acquisition performance by a factor of seven and response time by a factor of two in a 30-node network. We have also shown that the proposed algorithm outperforms its counterparts in the specified network setting.

1. Introduction

A distributed system consists of spatially separated computing machines that are able to communicate with each other using messages. These messages are passed over communication channels. A state of a distributed computing machine is characterized by its event history. This consists of the history of the local activities at a node and the message passing events on the communication channels. The global state of a distributed system is the collection of these properties. The global states are used to determine some stable properties such as termination detection and deadlock. In addition, they are used for protocol specification and verification and discarding obsolete information.

A meaningful global state is obtained if the nodes under the distributed system record their states simultaneously [1]. This situation is technologically unfeasible because there is no global system clock for the distributed nodes. So, a distributed snapshot should be obtained with a coordinated manner. This can be possible when each node is able to receive a message which is used to alert it to take its local snapshot using a coordinated checkpointing algorithm. In addition, local states should be delivered in messages to a central node via reliable channels within a short time frame in order to generate meaningful global states.

WSN can be considered as a distributed system consisting of a set of sensor nodes communicating via wireless channels. Yet WSN is not a complete distributed system since it lacks providing the abstraction which is an essential property of a distributed system. The WSN nodes periodically collect information about physical phenomenon and transmit them by using those channels. In addition, multihop data delivery is often used among WSN nodes. However, the data links are lossy and the communication among the nodes are unreliable.

WSN nodes contain batteries which cannot be replaced due to the fact that they tend to be deployed in unattended environments. For this reason, WSN nodes have to be self-configurable which makes traditional distributed algorithms hard to adapt to WSN [2]. To be able to tailor these distributed algorithms in accordance with WSN, we need to monitor the state of WSN nodes which necessitate taking the snapshot of the system in a distributed manner.

In this paper we propose a robust distributed snapshot algorithm which can capture the global state of a WSN that is running an environmental monitoring application. Snapshot

information is acquired from the network in a reliable manner seamlessly without interrupting or pausing the current application which is not enforcing reliable delivery for sensor readings. Gathered state information can then be used to obtain a global map of the available network resources, such as remaining energy level of the network. Our work is inspired from [3] which is a simple-tree algorithm that is designed for reliable wired infrastructures. However, the proposed algorithm operates in a fully distributed manner and addresses the arising reliability issue due to congestion and link failures. Besides, in order to handle snapshot and sensing operations together, we modified sensor node operation in a way to synchronize sensor readings, snapshot acquisition, and packet transmission events. Hence, the proposed algorithm takes the snapshot of a WSN that is running a monitoring application where sensor nodes sense the environment periodically and transmit their readings towards the sink node. To create a global state, sink initiates snapshot acquisition by sending a *marker* message to its children. The *marker* message is propagated to every other node in the network via their parent nodes. Each child receiving this message sends its state back to its parent which is then forwarded towards the sink over the path. Without reliability suit, sink node is not certain to receive snapshot control messages from all functioning nodes. With our reliability module, sink detects the missing control messages within a defined acquisition time frame and forces those specific nodes to resend their backed-up states. Sensor nodes keep their state messages intact for a predefined period of time so that in the case of loss it can be resent. State messages are propagated in the network along with periodic sensor readings.

We evaluated the performance of reliable (DS^+) and unreliable (DS) versions of our snapshot algorithm with respect to snapshot success rate and snapshot response time. We carried out tests in a simulation environment where the nodes transmit their sensor readings and local state messages to sink node over a tree topology network. A small chain topology in a real WSN test bed is also implemented for proof of concept. In both environments, the sink node processes sensor messages and delivers state information to the snapshot application through a socket interface. We compared our results with two related studies: Robust and Efficient Snapshot Algorithm (RES) [4] and Propagation of Information with Feedback (PIF) [5] as stated in their respected publications to show the superiority of DS^+ in the given network setting.

The content of the following sections is as follows. In Section 2 related work is reviewed. In Section 3 proposed snapshot algorithm is explained in detail. Section 4 describes the experimental setup including both simulation and real test bed revealing performance results. Section 6 concludes the paper with future work.

2. Related Work

There are various reliable transport layer protocols proposed for sensor networks which provide distributed or centralized congestion control [6]. Sensor Transmission Control Protocol (STCP) [7] provides distributed congestion control by initiating the flow in the network through a session initiation packet. This packet informs the base station about transmission rate and required reliability. After storing the information obtained from session initiation packet, the base station sends an acknowledgement to the transmitting node. Following this, other nodes start transmission. Flush [8], being another reliable transport protocol with distributed congestion control, employs sink as the main congestion control module. The sink requests packets from the sender until there are no unreceived packets. During this procedure, Flush estimates bottleneck bandwith by means of a dynamic rate control algorithm. Rate-Controlled Reliable Transport (RCRT) Protocol for Wireless Sensor Networks [6] employs a centralized congestion detection and rate adaptation mechanism. In this protocol, lost packet detection is the responsibility of the sink and upon lost packet detection, the sink makes the sensors resend those packets. Sink infers the existence of congestion if loss recovery takes much more than a round trip time. Upon congestion detection, the protocol decides on what the new traffic rate should be, then it decides on the new transmission rate of the sources to achieve the estimated traffic rate. Price-Oriented Reliable Transport Protocol (PORT) [9] supports centralized congestion control by utilizing neighborhood information. Nodes report packets to the sink and their transmission rate can be adjusted during the reporting phase. Sink informs the nodes about at what rate they should report as well. Assigning priority to different sensors is another approach followed to implement reliable transport [10]. Collaborative Transport Control Protocol (CTCP) [11] makes a distinction between communication losses and buffer overflows, along with achieving congestion detection and control with the contribution of all nodes. All the above mentioned studies address reliable transport as a layer in the WSN communication stack. They are not designed with an aim to take the global state of a sensor network while running an unreliable monitoring application.

In [2], the authors propose an enhancement on Chandy and Lamport's distributed snapshot algorithm [12] to obtain the energy map of a WSN. This algorithm collects the local states of the nodes to a central node and generate energy map of the network. The algorithm is made resilient to node failures by using a special packet type named "death warning." A node that is close to death sends a warning message to its neighbors and the neighbors take the odd node out of the topology map. However, the algorithm assumes that the network messages are never lost neither due to congestion nor due to channel conditions which is an unrealistic assumption for WSNs. Our proposed solution also differs from [2] in two senses; first it is capable of handling sensor readings and state messages simultaneously. Second, it handles packet losses due to congestion and link failures in the network. Hence, our failure model is based on a more realistic assumption for resource constraint nodes in tree topologies.

Lian et al. [13] studies on a way of acquiring global snapshot without snapshots of all nodes. They intend to reduce energy consumption in applications where sink needs

to keep snapshot history. They partition the network into regions where sensed data in one region lie in a predefined range so taking the snapshot of nodes creating the boundary would be adequate to form the global snapshot. But this algorithm needs sensor data obtained from different nodes to be similar. Another drawback of this approach is that some nodes consume more energy than others which attributes different roles to some nodes thus reduces the fault tolerance of the network. This is not the case for DS$^+$ where all nodes have the same role except sink initiating snapshot acquisition.

RES [4] addresses packet loss tolerance in WSN. Both of our approach and RES make the sink initiate the snapshot and messages are transmitted through network by propagation but there are a number of issues where distinction between these two algorithms is worth noting: Firstly, RES employs flooding for the marker message which is avoided in DS$^+$ for the sake of efficiency. Secondly, RES considers only sensor messages whereas DS$^+$ processes both snapshot messages and sensor readings having different reliability requirements. Moreover, RES makes each node to take its snapshot but transmission of snapshot messages is delayed until all local snapshots are aggregated together in a single message. Although aggregation reduces the overall network traffic, it has the potential to increase overall acquisition time. Besides, due to aggregation delay, when a node eventually transmits its snapshot, its actual state might have changed, meaning that transmitted snapshots are no more valid. Finally, RES defines only how a single global snapshot is taken whereas DS$^+$ defines how consecutive global snapshots can be taken.

Segall [5] presents distributed network protocol named PIF, which operates on tree topology similar to DS$^+$. However, this protocol maintains one path for each node to transmit its packets to the sink, so a packet loss cannot be overcome.

3. WSN Snapshot Algorithm

Let us define a wireless sensor network as an undirected graph $W = (N, E)$ consisting of a set of k nodes (accept sink) and a set of m edges. S_0 represents the sink node and the set of nodes denoted by $N = \{S_1, S_2, \ldots, S_k\}$ represent ordinary sensor nodes periodically sensing the environment and sending their readings. The set of edges denoted by $E = \{S_{ij}, S_{iv}, \ldots\}$ represent symmetric communication links between 1-hop neighbors; that is, node S_j and node S_v are in the vicinity of node S_i and vice versa. Given that the maximum number of children a node can have is denoted by b, the following legend is used in Algorithms 1–4:

C_i: child set of sensor S_i

M_{red}: red message, a.k.a. marker message

$M_{\text{white},i}$: white message sent from node S_i to its parent

Z: set of nodes whose snapshot message is not received by the sink at timeout

M_{rt}: snapshot retransmit message

t_s: snapshot acquisition period

$t_{c,i}$: timeout for creating a new snapshot message by S_i

B_i: set of ids of nodes at depth i

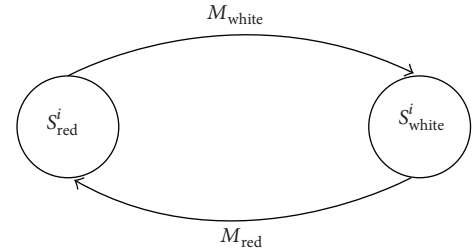


Figure 1: FSM modeling the node states.

sendB(S_i, C_i, M): event of sending message M from S_i to sensors in C_i

receive(S_i, S_j, M): event of receiving message M by S_i from S_j

$\lfloor (i-1)/2 \rfloor$: id of sensor which is parent of S_i

sendU(S_i, S_j, M): event of sending unicast message M from S_i to S_j.

3.1. Coordinated Checkpointing without Reliability Suit. Distributed snapshot component of our system employs a snapshot acquisition algorithm which is modeled as a Finite State Machine with *red* and *white* states representing node states as illustrated in Figure 1. Assuming that the sink node knows the number of sensors; k in the network, the sink sends a red message to its children initiating the snapshot process. When a node receives a red message, it moves to the red state. A node in the red state first forwards the red message to its children then acquires its own state. A node in the red state moves to white state when it transmits its local snapshot message to its parent. Snapshot messages are forwarded by their recipient nodes until the message is received by the sink. When sink node receives all snapshot messages from the network, it combines the gathered information to form a global checkpoint and the snapshot algorithm terminates.

3.2. Reliable Data Delivery for Distributed Snapshot. Distributed snapshot algorithm acquires snapshot through messages so ensuring message delivery necessitates a reliability suit. In our proposed reliability solution the number of nodes is assumed to be known by the sink within a snapshot time frame. Termination of one snapshot and initiation of a consecutive snapshot is allowed only when snapshot messages are received by the sink from all functioning nodes. The underlying topology is assumed to be fixed during the snapshot process. However it may change between consecutive snapshots. Sink ensures reliable data delivery by keeping a table indexed with node id's as explained in Algorithm 2. A slot is set to one when sink receives snapshot message from the node having the corresponding node id otherwise it contains zero. After sending the snapshot initiating marker message, sink starts waiting until timeout t_s expires. After timeout expiration, sink checks the content of the table to see from which nodes it has not received snapshot message yet and sends M_{rt} message only to those nodes to force them to retransmit. Before sending snapshot messages, nodes back up their acquired snapshot *sn*. When they receive M_{rt}

```
for i = 0 → d do
    for j = 0 ∈ B_i do
        sendB(S_j, C_j, M_red)
    end for
end for
```

Algorithm 1: Snapshot initiation.

```
While t_s not expired do
    if receive(S_i, S_⌊(i−1)/2⌋, M_red) = true then
        While i ≠ 0 do
            Wait for t_c,i
            sendU(S_i, S_⌊(i−1)/2⌋, M_white,i)
            i ← ⌊(i − 1)/2⌋
        end while
    else if receive(S_i, S_⌊(i−1)/2⌋, M_rt) = true then
        if M_white,i · payload = null then
            Take snapshot sn
            M_white,i · payload = sn
        end if
        While i ≠ 0 do
            sendU(S_i, S_⌊(i−1)/2⌋, M_white,i)
            i ← ⌊(i − 1)/2⌋
        end while
    end if
end while
```

Algorithm 2: Reliable transmission of snapshot messages.

```
While t_s not expired do
    if receive(S_0, S_i ∈ N, M_white,i) = true then
        A[i] ← 1
    end if
end while
Restart t_s
for i = 1 → k do
    If A[i] = 0 then
        m ← 0
        j ← findD(i, m)
        While j ≠ i do
            sendU(S_m, S_j, M_rt)
            m ← j
            j ← findD(i, m)
        end while
    end if
end for
```

Algorithm 3: Reliability procedure run on sink.

```
while !((u = b ∗ nodeId + 1) ∨ (u = b ∗ nodeId + 2)) do
    u ← u − 1
    u ← u/2
end while
return u
```

Algorithm 4: findD(u,nodeId).

message, they resend their backed up *sn*. If they do not have a backed-up *sn*, this indicates the reason why sink does not have their snapshot message yet since they did not receive M_{red} message. This procedure continues until M_{white} messages from all nodes are received by sink and afterwards the consecutive snapshot can be initiated.

Algorithm 1 shows snapshot initiation process. Sink node sends M_{red} message to its children and upon receiving M_{red}, each node propagates the message to its own children.

The robustness of message delivery is achieved by using Algorithm 2 as follows: when a node receives M_{red} from its parent, it waits for $t_{c,i}$ period to reduce congestion and sends its snapshot message $M_{\text{white},i}$ to its parent. A node, receiving snapshot message from its child, forwards this message to its own parent. Forwarding continues until snapshot message is received by the sink. If a node receives M_{rt} from its parent, it means that sink did not receive its snapshot message yet. The reliability algorithm handles two cases causing unsuccessful delivery. The first case is that M_{red} message is not received by the node, so snapshot is not taken and a message is not generated. The second case is that M_{red} is received and snapshot is taken but snapshot message does not reach the sink though it is generated. In order to handle the first case, node i takes its snapshot and generates $M_{\text{white},i}$. For the second case, node i does not generate $M_{\text{white},i}$ and sends its backed-up $M_{\text{white},i}$ message to its parent. Each node, receiving a snapshot message from its children, forwards the message to its own parent.

The sink controlled reliability protocol is given in Algorithm 3. When sink receives a snapshot message, it sets the slot belonging to the sender node indicating successful reception of the state information. When t_s expires, if there are slots containing zero, sink sends M_{rt} to the nodes in Z, that is, the nodes from which the snapshot message is not received yet. M_{rt} is propagated using findD routine as explained in Algorithm 4. When a node needs to send a message to another node which is more than one hop away, it needs to know to which intermediate node it should send the message so that using the same knowledge other intermediate nodes can make the ultimate receiver have the message. Given the id q of node from which sink did not receive snapshot message within t_s, findD calculates the ids of intermediate nodes between sink and the node q.

Transmission patterns belonging to the nodes in the network partition surrounded by a dashed rectangle in Figure 2 are shown in Figure 3(a) in the case where no packet loss occurs and in Figure 3(b) where sink does not receive snapshot message from $node_2$. The messages exchanged in the reliable distributed snapshot algorithm is shown on a time chart where nodes are structured as a binary tree with depth two. In the figures, sink is the parent of $node_0$ and $node_0$ is the parent of $node_1$ and $node_2$.

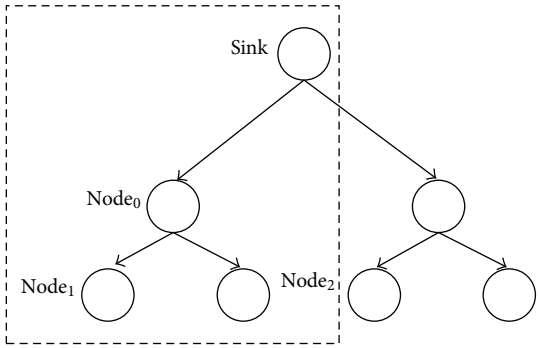


Figure 2: An example network (depth = 2, nodes = 6)

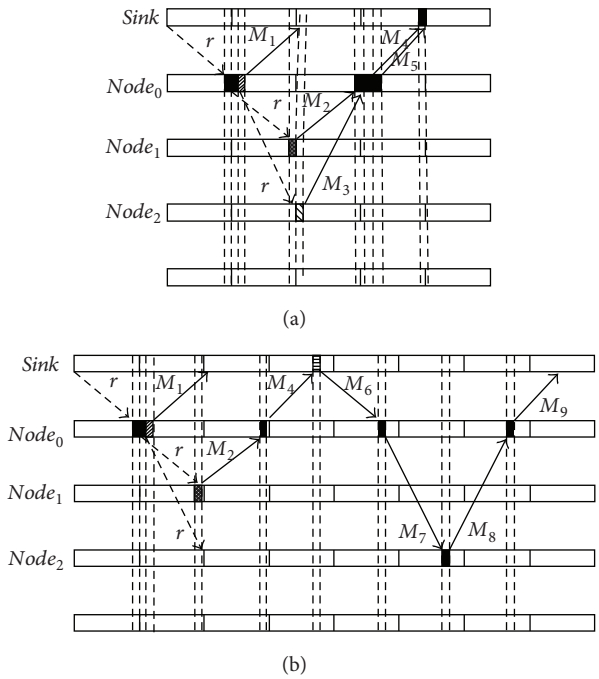


Figure 3: Message exchange: (a) no packet loss and (b) sink does not receive state message from $node_2$.

4. Experimental Setup

We tested our snapshot algorithm on TOSSIM simulator [14] which is a TinyOS [15] emulation environment for sensor nodes. We designed our experiments for a varying number of nodes as given in Table 2 where nodes conform with binary tree topology as illustrated in Figure 2. We repeated each experiment 100 times and the mean values are plotted in the graphs. In the simulations, the radio attenuation value for each channel between the nodes is set to 30 dBM so that BER can be neglected. It is also assumed that packet losses due to radio channel are handled by the underlying MAC protocol.

We tested the algorithm's performance in the case where network packets are heavily dropped by the nodes due to congestion. In our setup, sensor nodes do not prioritize sensor readings and control messages ($M_{\text{white},i}$ and M_{red}). They transmit and/or forward the packets in the order they arrive. Simulation parameter values used in the test environment are given in Table 1.

Table 1: Simulation parameters.

Max number of nodes	62
Max depth	5
Topology	Regular binary tree
Scheduling policy	FIFO
Bandwidth	256 Kbps
Radio attenuation	30 dB
Sensor sampling frequency	2, 5, 10 sec

Table 2: Furthest node latency versus depth.

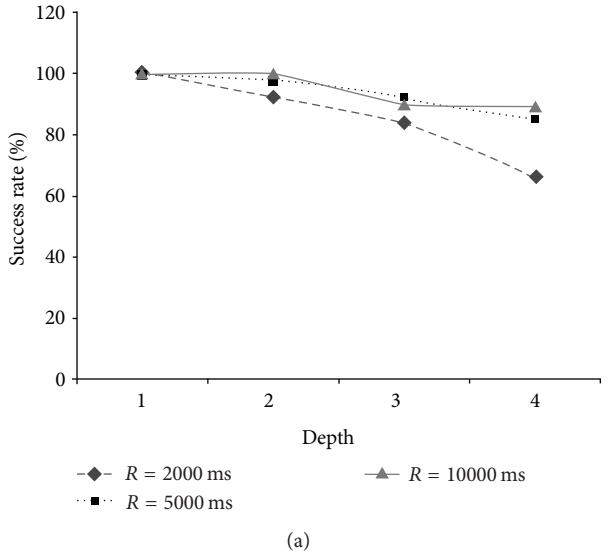
Depth	Number of nodes	Furthest node latency (ms)
1	2	11
2	6	22
3	14	36
4	30	49

As given in Algorithms 2 and 3 there are two parameters t_s and $t_{c,i}$ which affect the overall performance of the system since they change the ratio of control packets to all packets including sensor data. The lower this ratio is, the better the algorithm performs. We tested the system for varying sensor reading periods denoted by R being 2, 5, and 10 sec. The snapshot acquisition time, t_s, and timeout between consecutive snapshots; $t_{c,i}$ are given in (1) and (2), respectively, where k denotes the number of nodes except sink and n being id of any node. In (1), t_s is designated as $t_{s,R}$ where $t_{s,R}$ is the time corresponding to related R and F values where F stands for *furthest node latency*. This is a parameter experimentally determined representing the maximum amount of time sink waits for the snapshot messages from all nodes. In other words, F is the amount of time passed until sink receives snapshot message from the furthest node meaning that the higher the depth of the tree is, the greater F becomes. Upon running distributed snapshot algorithm without using reliability suit, we obtained F values tabulated in Table 2 and used them to specify t_s in the reliability suit. The node with id i waits for the duration of $t_{c,i}$ before sending its $M_{\text{white},i}$ message. As shown in (2), nodes with greater ids are delayed more as a way of reducing congestion:

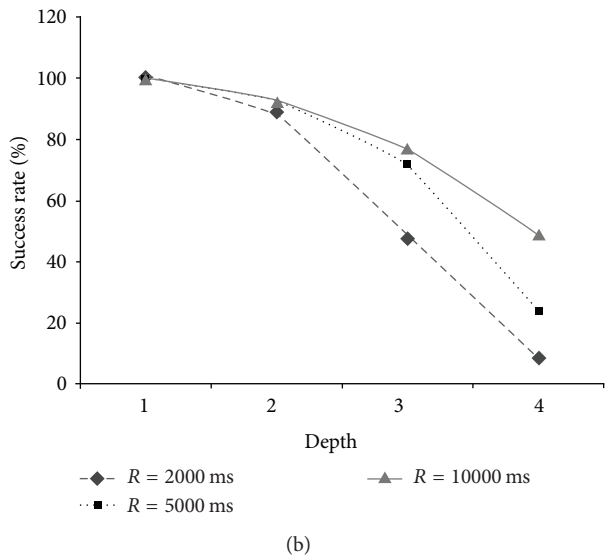
$$t_{s,R} = F + R, \tag{1}$$

$$t_{c,i} = \left(\frac{R}{k}\right) * n. \tag{2}$$

4.1. Performance Analysis. We defined a snapshot as successful if sink receives $M_{\text{white},i}$ messages from all nodes in a controlled experiment. Snapshot acquisition success rate is the percentage of successful snapshots over all snapshot attempts. In the first group of tests, we compared our reliable snapshot algorithm (DS^+) with its version without reliability extension (DS). As illustrated in Figure 4, DS^+ improves the success rate of snapshot acquisition compared to DS by a factor of seven. Both DS^+ and DS deteriorate as sensor



(a)



(b)

Figure 4: Snapshot success rate for (a) DS$^+$ and (b) DS.

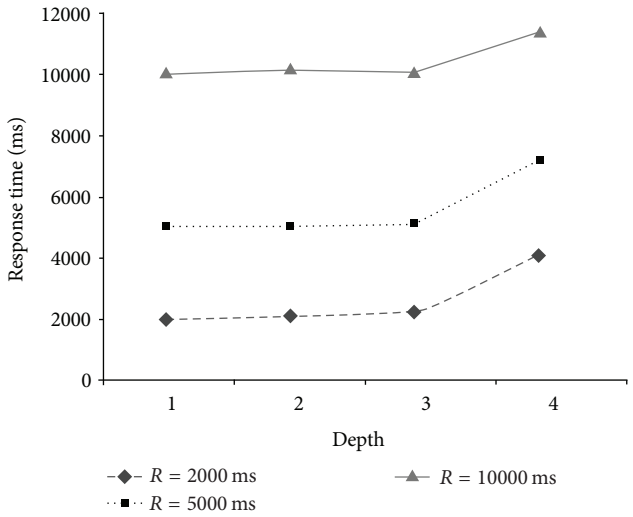


Figure 5: Response time of DS$^+$.

Table 3: Response time comparison for depth four and five in DS$^+$.

R (ms)	Depth four	Depth five
2000	4085	16303
5000	7207	16782
10000	11377	22010

Table 4: Response time comparison for depth four and five in DS.

R (ms)	Depth four	Depth five
2000	2071	2083
5000	5072	5083
10000	10073	10088

reading period decreases and depth increases but decrease in the success rate of DS$^+$ is less than that of DS.

DS does not exhibit much variation in response time as depth increases as can be seen in Table 5 where minimum and maximum response time values of all depths are shown along with population standard deviation (σ). Minimum and Maximum values belong to depth one and depth four, respectively. Lack of variation in response time is caused by the delay formula shown in (2), $t_{c,i}$ values are set in such a way that furthest node is exposed to a delay which lasts as long as sensor reading period. As can be seen in these results and Figure 5, average snapshot acquisition time is higher in DS$^+$ than DS in successful acquisitions. This is an expected result due to retransmission overhead caused by congestion. The difference between the two cases increases as the depth of the tree increases. This is due to the delivery of M_{rt} and the retransmission of $M_{\text{white},i}$ messages. However, acquisition success rate is much higher in DS$^+$ revealing the success of the algorithm.

Reliable distributed snapshot algorithm is successful in small-to-medium scale networks. Between response time values for network with depth four and network with depth five (62 nodes), a sharp difference is observed as shown in Tables 3 and 4 where response time values specified are in milliseconds. The reason why reliable version is not scalable after depth four is that reliability is achieved in a sink oriented manner.

We compared the performance of our algorithm with RES and PIF based on their published results. In Table 6, we present the snapshot success and response time values for 15 and 35 node cases. The results yield that DS$^+$ outperforms RES and PIF in terms of response time when sampling frequency is R = 2000 ms. Note that DS$^+$ performs even better with R = 5000 ms.

5. Real Testbed

Due to the restrictions of simulation environment, proficiency of DS$^+$ in terms of congestion handling is experimented on a real environment. We designed a real test

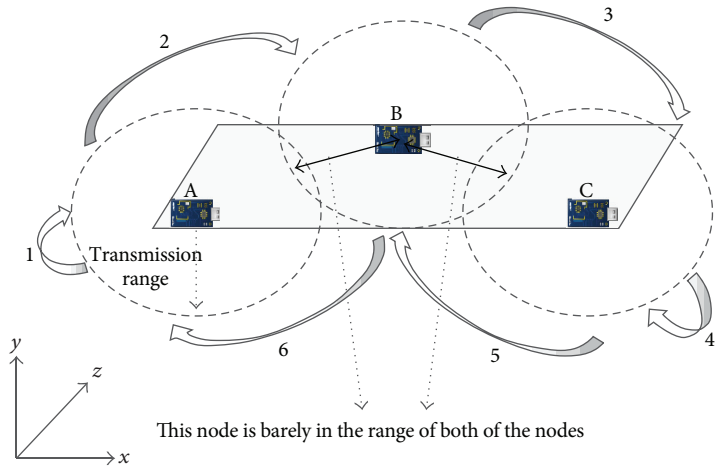


Figure 6: Test environment for three nodes.

Table 5: Response time variation for DS.

R (ms)	Min (ms)	Max (ms)	σ (ms)
2000	2017	2071	19.659
5000	5017	5072	19.862
10000	10017	10073	20.463

Table 6: Comparison of RES, PIF, and DS$^+$.

	RES	PIF	DS$^+$	RES	PIF	DS$^+$
Node count	15	15	14	35	35	30
Success rate (%)	90	72	92	77	54	85
Response time (ms)	12000	20000	2272	27000	40000	4085

bed using battery powered TmoteSky [16] and sensor nodes running TinyOS [15]. We programmed sensor nodes with NesC language in a way that they periodically transmit their sensor readings and respond to snapshot messages in the meanwhile. We tested our algorithm on a chain topology which is again a strict form of tree. Each node propagates local messages as well as the messages it receives from its child. The test bed has a clear line of sight between nodes in order to homogenize environmental effects on the communication channels as illustrated in Figure 6.

In the test bed, sensor nodes introduce an additional conversion delay due to analog to digital converters (ADCs) which is ignored in simulations. Interrupt handling is highly affected by these delays and in turn can cause packet drops since when a new packet arrives at the radio, the node might be busy with sensing. Besides, TmoteSky has a single processing unit responsible for all computations so a task driven architecture manages interrupts considering their priorities. Hardware interrupts such as ADC operations have a higher priority than any other software interrupts, so message receiving tasks can be delayed until hardware related tasks are completed which is another cause of packet losses. Because dynamic buffers are not allowed in TinyOS, we implemented separate message buffers structured as static circular queues for each message type: sensor data and control message. We analyzed the impact of multiple buffers on the performance of the snapshot algorithm.

Figure 7 illustrates the results of reliability tests without buffer support. Sensor nodes are structured in a chain topology of four hops. 3000 packets are received at the sink in total. In the experiment, packets are propagated just immediately after they are received if the channel is available, otherwise dropped. Packet losses does not occur at the channel because of the clear line of sight and signal power setting. The main observation is that the packet losses because the congestion increases as the nodes get closer to the sink in terms of hop number as expected.

In the second experiment, we tested the system with multiple message buffers. The number of packets received and packet response times are illustrated in Figures 8 and 9, respectively. Since there are separate buffers for local state messages and periodic sensor messages, no local state messages and marker messages are lost. The results present time elapsed during the receiving procedure and packets received by the sink from each node.

According to the test results, congestion increases as the depth of the network increases. On the other hand, the time required to receive 5000 packets gets smaller as the number of nodes increases. This is due to the task handling mechanism of TinyOS. Since the tasks are posted one after another, the operating system pushes all the tasks that is in the internal queue to the processing unit. Since the buffers are statically allocated, buffer size has an effect on the packet loss distribution as well. If the ratio of buffer size over the total number of nodes gets smaller, then the number of received packets per node decreases.

6. Conclusions and Future Work

Distributed snapshot of WSN is an assistive tool in terms of observing momentary conditions which may become obstacles for WSN functioning such as termination and

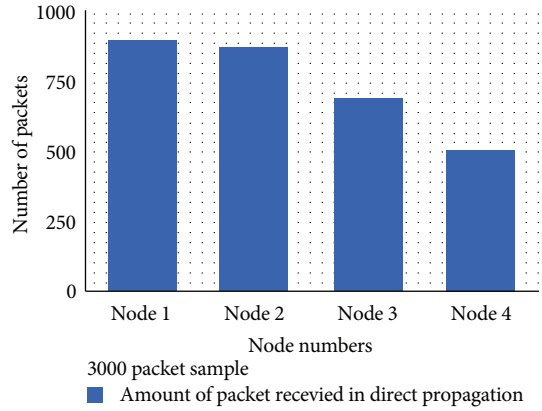


Figure 7: Direct packet propagation results for four nodes.

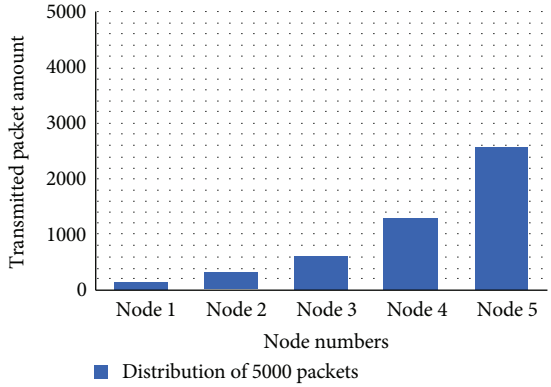


Figure 8: Packets received from two, three, four, and five hop topologies.

deadlock. In this work, we propose a reliable distributed snapshot algorithm suitable for resource constrained WSNs. We show that the algorithm is successful in terms of response time and snapshot success rate.

Reliability integration to distributed snapshot algorithm has distributed and centralized implementation options. Centralized implementation has higher communication cost but it has less storage requirements and lower implementation complexity whereas the distributed version has lower communication cost with higher implementation complexity. There are two reasons why distributed reliability will cause less traffic than centralized version. In centralized case all snapshot messages are propagated to a sink node while in distributed case each parent node in the tree can merge snapshot messages coming from its children and propagate a single message to its own parent. Second, in centralized version, sink will send M_{rt} message to each node from which it does not receive a snapshot message so in the worst case M_{rt} message will be propagated to a leaf node in the tree whereas in distributed version, sending M_{rt} message does not require message propagation throughout the network. For a node whose snapshot message is not received by its parent, parent itself will send M_{rt} message which requires only one hop message exchange.

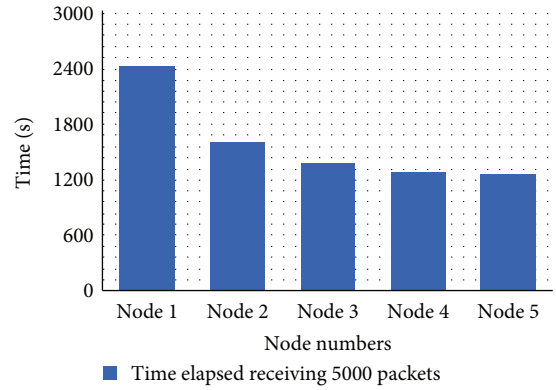


Figure 9: Durations for receiving 5000 packets.

Our reliable distributed snapshot algorithm is highly successful in small-to-medium scale networks since reliability is sink based. However, hop based distributed reliability scheme may achieve better scalability.

As a future work, we plan to implement our distributed snapshot algorithm using the above mentioned distributed reliability suit where each parent keeps a list of snapshot messages it receives from its children and participate in the retransmission of unreceived messages from its children.

Acknowledgment

The authors gratefully acknowledge Ugur Arpaci for coding the message exchange in NesC language and setting up a TmoteSky testbed as a part of his engineering project.

References

[1] A. D. Kshemkalyani, M. Raynal, and M. Singhal, "An introduction to snapshot algorithms in distributed computing," *Distributed Systems Engineering*, vol. 2, no. 4, article 005, pp. 224–233, 1995.

[2] A. da Silva, F. Teixeira, R. Lage, L. Ruiz, A. Loureiro, and J. Nogueira, "Using a distributed snapshot algorithm in wireless sensor networks," in *Proceedings of the 9th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS '03)*, pp. 31–37, May 2003.

[3] A. D. Kshemkalyani, "Fast and message-efficient global snapshot algorithms for large-scale distributed systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 9, pp. 1281–1289, 2010.

[4] W. Wu, H. Liu, and H. Wu, "Res: a robust and efficient snapshot algorithm for wireless sensor networks," in *Proceedings of the 32nd International Conference on Distributed Computing Systems Workshops (ICDCSW '12)*, pp. 231–236, June 2012.

[5] A. Segall, "Distributed network protocols," *IEEE Transactions on Information Theory*, vol. 29, pp. 23–35, 1983.

[6] J. Paek and R. Govindan, "RCRT: rate-controlled reliable transport for wireless sensor networks," in *Proceedings of the 5th ACM International Conference on Embedded Networked Sensor Systems (SenSys '07)*, pp. 305–319, November 2007.

[7] Y. G. Iyer, S. Gandham, and S. Venkatesan, "STCP: a generic transport layer protocol for wireless sensor networks," in

Proceedings of the 14th International Conference on Computer Communications and Networks (ICCCN '05), pp. 449–454, October 2005.

[8] S. Kim, R. Fonseca, P. Dutta et al., "Flush: a reliable bulk transport protocol for multihop wireless networks," in *Proceedings of the 5th ACM International Conference on Embedded Networked Sensor Systems (SenSys '07)*, pp. 351–365, New York, NY, USA, November 2007.

[9] Y. Zhou, M. R. Lyu, J. Liu, and H. Wang, "PORT: a price-oriented reliable transport protocol for wireless sensor networks," in *Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering (ISSRE '05)*, pp. 117–126, November 2005.

[10] A. Sharif, V. Potdar, and A. J. D. Rathnayaka, "Priority enabled transport layer protocol for wireless sensor network," in *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications Workshops (WAINA '10)*, pp. 583–588, April 2010.

[11] E. Giancoli, F. Jabour, and A. Pedroza, "CTCP: reliable transport control protocol for Sensor networks," in *Proceedings of the International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP '08)*, pp. 493–498, December 2008.

[12] K. M. Chandy and L. Lamport, "Distributed snapshots: determining global states of distributed systems," *ACM Transactions on Computer Systems*, vol. 3, no. 1, pp. 63–75, 1985.

[13] J. Lian, L. Chen, K. Nak, Y. Liu, and G. B. Agnew, "Gradient boundary detection for time series snapshot construction in sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 10, pp. 1462–1475, 2007.

[14] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: accurate and scalable simulation of entire TinyOS applications," in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys '03)*, pp. 126–137, New York, NY, USA, November 2003.

[15] U. Department, TinyOS: an operating system for sensor networks, http://www.tinyos.net/.

[16] T. Crossbow, TelosB Data Sheet, http://www.xbow.com/.