Implementation and Performance Analysis of Wireless Sensor Network Protocols

Department of Electrical & Computer Engineering, University of Florida, Gainesville

Keerti Randhawa, Punam Mahato, Shagun Arora

Abstract: **Wireless sensor nodes have limited processing power, storage capacity and energy resources. Hence, they need specifically designed routing protocols with minimum energy expenses so that nodes have longer lifetime. All nodes follow the specified rules in order to communicate and transmit data to the respective sink node and among themselves in an energy efficient and time synchronized manner. Many protocols have been developed which have their own advantages and disadvantages. For our project we have implemented Gossip, SPIN and Flooding and evaluated their performances in terms of total messages passed, overall bandwidth required and time required for the dissemination of information to all the nodes in the network.**

Keywords: Flooding, Gossip, Scala, SPIN, WSN protocols

I. INTRODUCTION:

Wireless sensor network is a pool of distributed sensor nodes which can measure physical and environmental parameters such as temperature, humidity, light, sound, pressure, etc with applications in areas including health care, utilities, and remote monitoring. A single node in itself is incapable of collecting all the desired data and performing all the computation needed, so they coordinate with each other and work cooperatively for the information to finally reach the sink or gateway. With increasing use of wireless sensor networks for sensing and monitoring, and the emergence of Internet of Things, a number of sensor-node platforms have been built in the past few years. Operating systems suitable exclusively for wireless sensor nodes have been developed and some important ones are TinyOS, Contiki, Mantis and Lite.

Initially we worked on implementing SPIN protocol in nesC for TinyOS based motes. TinyOS is an open source OS which is based on the component descriptor language nesC, is event driven and provides modularity [1].

Our work includes the implementation of SPIN, Gossip and Flooding protocols for wireless sensor networks using Scala. TinyOS is an open source OS which works on the component descriptor language nesC while providing modularity and flexibility, thereby making it the obvious choice for wireless sensor nodes [1]. Although working with TinyOS makes it very easy to write applications in WSNs but working with it is not trivial.

Our work includes the implementation of SPIN, Gossip and Flooding protocols for wireless sensor networks using Scala. We have analyzed the behavior of these protocols for two different topologies Line and 3D grid and made a performance comparison among them. SPIN is basically a data centric protocol which fits under the category of event-driven and data delivery model. SPIN has two stages, negotiation and data exchange. Negotiation between nodes takes place before the actual transmission of data through the exchange of control packets like ADV, REQ. The second protocol in consideration is Gossip. This protocol basically imitates the gossip in social networks. It is especially useful in distributed systems where the basic network either has a tiresome structure or is very large [3]. The third routing protocol in consideration is Flooding. Here, data packet is flooded into the network, neighbors after neighbors. This makes the process very fast but at the cost of resource and bandwidth utilization.

The rest of the report is organized as follows. A brief description of TinyOS, nesC and the routing protocols is given in section II. Experimental description is described in section III along with code snippets. Implementation and analysis of SPIN, Gossip and Flooding are given in detail in sections IV. Detailed graphs are included in the

analysis section that describe the individual performance of these protocols along with comparisons with its counterparts. The report is concluded in section V.

II. REFERENCE PAPER DESCRIPTIONS:

TinyOS: TinyOS was designed to meet the requirements of networked embedded systems which has support for real-time applications. Based on event driven programming approach, the program keeps on listening for an event and the response is triggered only if a particular event occurs. Applications based on TinyOS are assembled as a set of modules that communicate among themselves as well as the scheduler and the hardware to produce output in the form of a graph. Before deploying TinyOS applications, they can be run and tested on a simulator called TOSSIM.

NesC: It is an event driven, component-based software programming language that is commonly used for building applications based on TinyOS motes. It is extended verion of C with modules bound together. Each component has its own program and these programs are then assembled together to form a complete application. [6].

SPIN: Sensor protocol for information via negotiation is a protocol that was developed with the aim of disseminating information efficiently in a WSN because the conventional protocols like flooding and gossiping create redundancy of data packets and waste the already limited energy and other resources in sensor nodes.

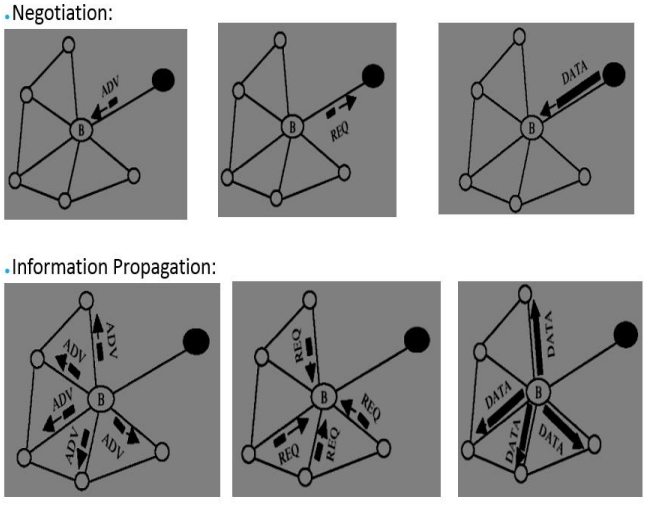


Figure 1

SPIN distributes information from every node in the network to every other node and it has two aspects associated with it. Before the actual transmission of data, nodes undergo negotiation through control packets which have the metadata information in them. The nodes then request for the data packet only if after checking the metadata they find that they don't have the updated data. The second aspect of SPIN involves each node monitoring the consumption of energy. The nodes participate in negotiation only if their available energy is more than some pre-set threshold value. The messages used in the SPIN protocol are ADV, REQ and DATA. The nodes send out the advertisement (ADV) packets as a broadcast to all the neighboring nodes. If a node is interested in the advertised data, it sends back a request (REQ) packet after which the advertising node sends the actual data packet. Due to this ADV, REQ, DATA mechanism, SPIN protocol is best suited for applications that are event-driven and require avoiding redundant communication due to resource constraints.

GOSSIP: Gossip based algorithms are gaining eminence with an increase in demand for scalable communication in distributed systems. It has a very simple underlying principle where each node selects a random neighbor and sends its information to it. This node again chooses another random node from its neighbors and passes the information to it. Therefore, over a period of time, the information is spread across the system like an epidemic [7]. The advantage of using gossip based protocols is that they are robust i.e. they can withstand message losses and node crashes without affecting the reliability of the system. Since each node initiates only a single message at a time, there is no problem of load balancing unlike in systems with central controller. Another advantage of gossip based protocols is that they are fairly easy to implement because of the simplicity of the underlying principle which does not impose any implementation and coordination of control packets. The protocol is successfully scalable to a large number of nodes, which makes it attractive for large area monitoring. It can also be used for notification of occurrence of an event at one point to other point of the network. [8]

FLOODING: Flooding is a fundamental choice for networks where there is a need for propagating updates in the network, and there is a need for reliable delivery of information from source to every other node in the network.[11] Each node passes its information to all the neighbors and this forms a chain action. This helps in quick dissemination of information throughout the network with a very high diffusion speed.

III. EXPERIMENT DESCRIPTION:

This section explains the software programs developed in Scala for different protocols.

(a) Implementation of SPIN:

For implementation of SPIN protocol, Sink.scala picks a random node which measures the data. The sensing node sends ADV packets to all its neighbors. This contains the sensing time as meta-data. The neighbor nodes compare this sensingTime_metadata with lastDataCollection_Time and accordingly send REQ packet to sensing node. The sensing node sends the DATA packet to all the requesting nodes. These node then again start the advertising process to the respective neighbors. Thus, the information propagates through negotiation. Upon receiving each message, the sink node is notified and the respective message counters DATAMESSAGESCOUNT and CONTROLMESSAGESCOUNT are incremented and the required bandwidth is calculated.

```
case ADV(nodeID, sensingTime_metaData) =>
sink_actor ! IncrementControlMessageCounter()
if ((sensingTime_metaData -
lastDataCollection_Time ) > 0.0 ){
println("\nNode: "+index+" : Received ADV
packet from node: " + nodeID)
sender ! REQ(index, sensingTime_metaData)
}
else {
println("\nNode: "+index+" : Received ADV
packet from node "+ nodeID+". But have updated
data already!")
}

case REQ(nodeID, sensingTime_metaData) =>
sink_actor ! IncrementControlMessageCounter()
println("\nNode: "+index+" : Received REQ
packet from node "+ nodeID)
sender ! DATA(index, newData,
sensingTime_metaData)

case DATA(nodeID, updatedData,
sensingTime_metaData) =>
sink_actor ! NodeReceivedData(index)
sink_actor ! IncrementDataMessageCounter()
lastDataCollection_Time =
sensingTime_metaData
newData = updatedData
var neighbour = null
for (neighbour <- myNeighboursList){
println("\nNode: "+index+" : New data received
from node "+ nodeID+" and passing ADV packets
to all neighbours.")
neighbour ! ADV(index, sensingTime_metaData)
}
```

For the implementation of gossip, Sink.scala picks up a random node which measures the data. The sensing node picks a random neighbor and sends the message to that neighbor. Upon receiving the message, this neighbor node sends the data to one of its neighbors which is picked up randomly. Thus, this chain of message passing continues and the information propagates. Upon receiving each message, the sink node is notified and the respective message counter defined as TOTALMESSAGESCOUNT is incremented and the required bandwidth is calculated. Gossip protocol does not have control messages.

```
case BeginGossip(gossipMessage) =>
sink_actor ! IncrementMessageCounter(index)
val r =
(scala.util.Random).nextInt(myNeighboursList.len
gth)
println("\nGossip started & passing message to
neighbour: " + r +"\n")
myNeighboursList(r) !
PassMessage(gossipMessage, index)


case PassMessage(gossipMessage, nodeID) =>
sink_actor ! IncrementMessageCounter(index)

println("\nNode: "+index +" got message from
node: " + nodeID)
val r =
(scala.util.Random).nextInt(myNeighboursList.len
gth)
println("\nNode: "+index +" passing message to
neighbour: " + r)
//println("MygossipMessageCount: " +
gossipMessageCount)
myNeighboursList(r) !
PassMessage(gossipMessage,index)
```

(c) Implementation of Flooding:

For the implementation of flooding, Sink.scala picks up a random node which senses the data and broadcasts this data to all its neighbors except the one from which it originated. These nodes again send this data to all their neighbor nodes. Thus, there is information implosion resulting in multiple copies of same data at multiple destinations. Each time data packet is received by a node, TOTALMESSAGESCOUNT is incremented and bandwidth required is calculated. Like gossiping, even flooding protocol does not have any control messages.

Bandwidth is calculated as:

[(TOTALMESSAGESCOUNT) * 1000] / Time taken

The code snippet for Flooding is enclosed on the next page.

```
case BeginFlooding(floodingMessage) =>
sink_actor ! IncrementMessageCounter(index)
println("\nFlooding started & passing message to all
neighbours.\n")
for (i<-0 until myNeighboursList.length){
myNeighboursList(i) ! ForwardMessage(floodingMessage,
index)
}


case ForwardMessage(floodingMessage, nodeID) =>
sink_actor ! IncrementMessageCounter(index)

println("\nNode: "+index +" got message from node: " +
nodeID + " \t\t Passing message to all neighbors. ")
for (i<-0 until myNeighboursList.length){
myNeighboursList(i) ! ForwardMessage(floodingMessage,
index)
}
```

IV. SIMULATION DESCRIPTION:

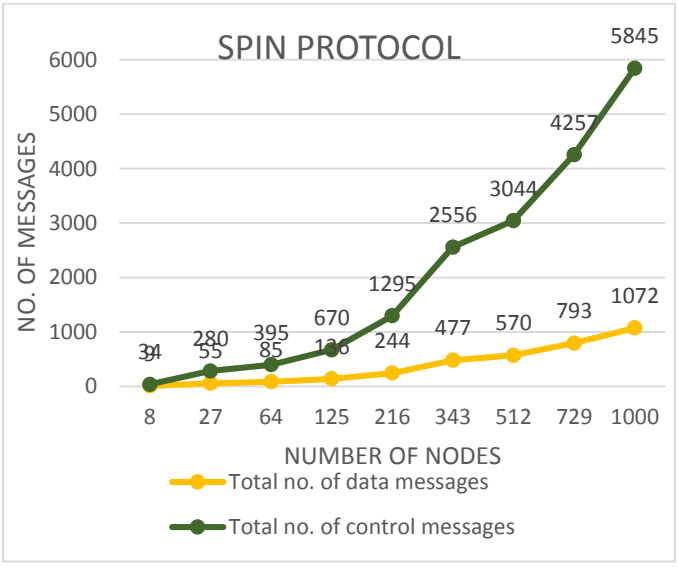
TOSSIM: (TinyOS simulator) is usually used for simulation of TinyOS applications. The advantage of using TOSSIM is that it bridges the gap between design, implementation and deployment of an application. It can capture the performance of a mote at a very granular level. Event driven approach of TinyOS makes it compatible with the discrete event simulation of TOSSIM and each event can be recorded and seen on the simulator for debugging and analysis. WSN programs need to be tested for different magnitudes of network and in different environments so as to identify their behavior under different conditions. Therefore, it is not feasible to place sensor motes at actual fields for testing purposes because even if one of the sensor fails, setting it up again is a large overhead for the entire network [9]. TOSSIM lets the user debug, test and analyze the algorithms in a controlled and repetitive manner.

TOSSIM is started automatically whenever an application is compiled by entering the application directory (eg. 'tinyos-root directory'/apps/Blink). Applications can be compiled for a simulated mote by the command: *make micaz sim*, or *make telosb sim*. [10]

Scala: Another software that we used for this project is Scala which stands for Scalable Language. Scala libraries and toolkits suited our requirements to program the sensor nodes and their message passing feature. We used Akka toolkit for programming each node as an actor (program unit) and data and control messages were sent to the respective actor's message queue which were then processed accordingly. Scala is a statically typed language and because of its functional style of programming, Scala systems are easily scalable. The Actor module of Scala is very efficient for the development of communication protocols. Another positive aspect of Scala is that programs developed in Scala can be compiled on JVM, thereby, eliminating the need of a separate compiler.

We have tested the performance of SPIN, Gossip and Flooding protocols by choosing various network sizes. Effect of increasing the number of nodes is observed on parameters like total time taken, total number of messages (for Gossip and Flooding), total number of data and control messages (for SPIN protocol) and bandwidth. For the analysis, 3D grid topology was used. Hence, the number of nodes were increased with cubic values like 8, 27 etc. Therefore, even if the user passes a non-cubic value at run time, the program will automatically change the number of nodes to the nearest cube possible.

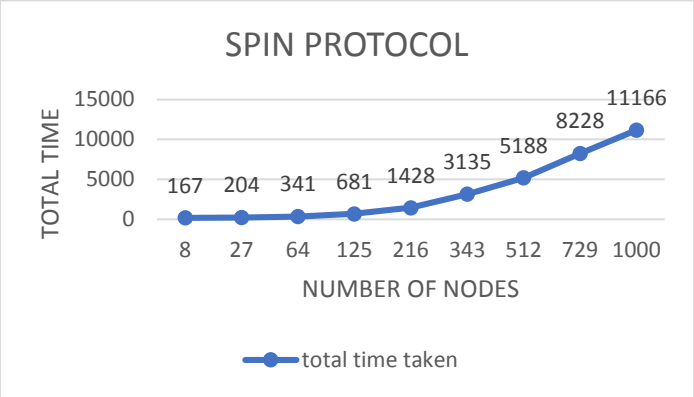
(a) Analysis for SPIN:



Graph 1

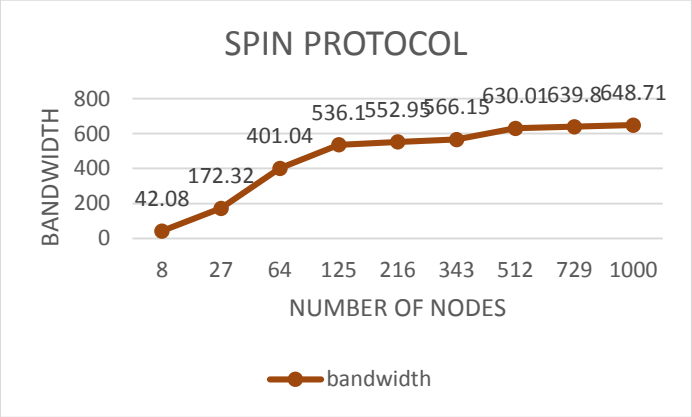
For SPIN protocol using 3D grid topology we see that the number of control packets are large in comparison to data messages. This is because as we increase the number of nodes, the number of control messages increase because of the negotiation (ADV, REQ) packets in SPIN protocol. The second parameter in consideration for SPIN protocol is total time taken. Graph 2 shows that as the number of nodes increases, the time for information propagation across all nodes in the network increases. This proves that SPIN is relatively slow as compared to other protocols

because a lot of time is spent in negotiation among nodes before the actual data transfer.



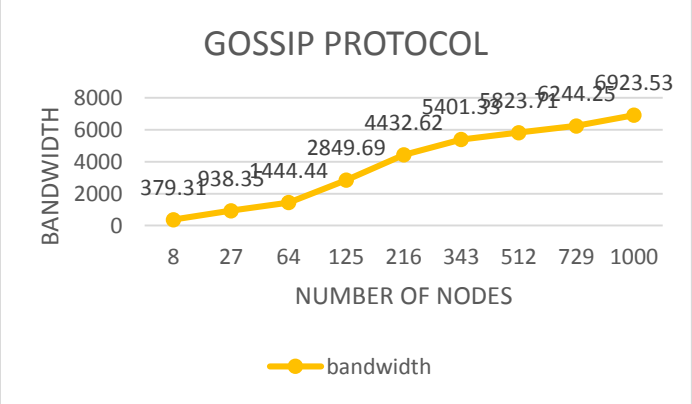
Graph 2

Graph 3 shows that for SPIN, as the number of nodes increases, the bandwidth of the network increases.



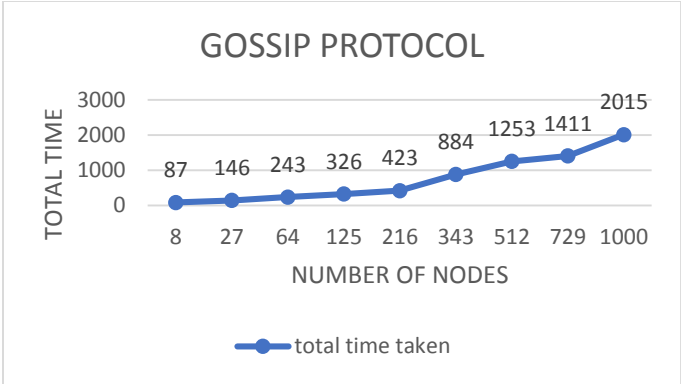
Graph 3

(b) Analysis for Gossip: Graph 4 shows that as the network scales the bandwidth increases.



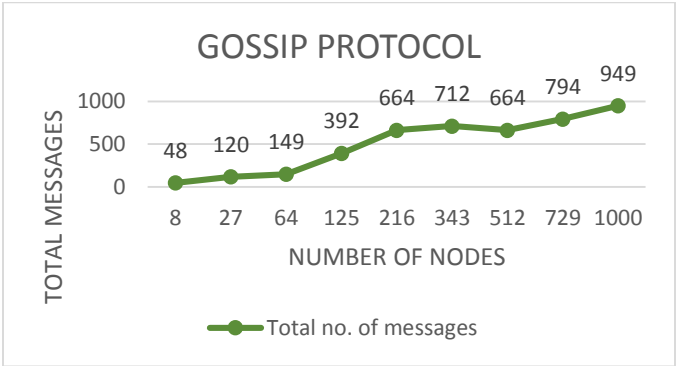
Graph 4

Graph 5 also shows an increasing pattern in the time taken as the number of nodes are increased. However, it can be seen that the time taken by Gossip is relatively lower as the information is disseminated very fast due to the absence of any negotiation.



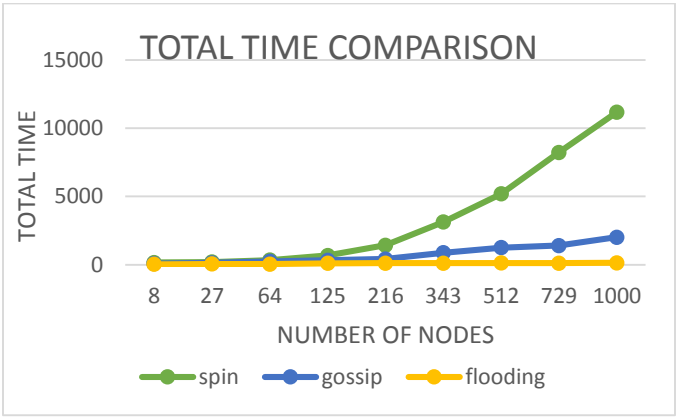
Graph 5

Graph 6 shows that the total number of data messages increases with an increase in number of nodes.



Graph 6

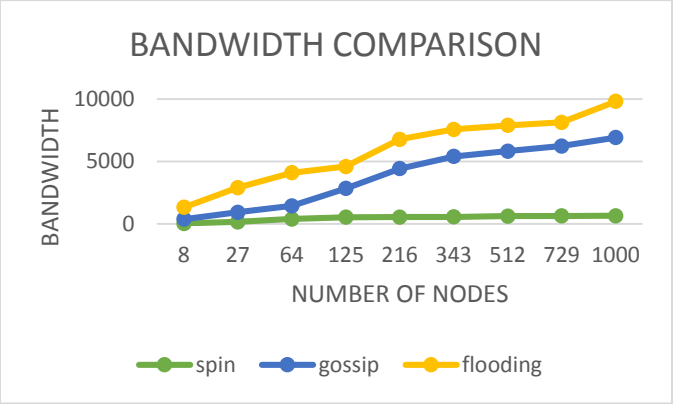
(c) Comparison between SPIN, gossip and flooding: Graph 7 is a comparison between the time taken by SPIN, gossip and flooding using 3D grid network topology. We can see the total time taken for SPIN is greater than gossip and flooding. This is because of the negotiation that takes place in SPIN. Time required for flooding is the least because it broadcasts to all the neighbors and the data is sent to all the nodes at once.



Graph 7

Graph 8 shows a comparison between bandwidth requirement for SPIN, gossip and flooding. Bandwidth needed for flooding is more than that for gossip and SPIN because it sends data to all nodes at once. The bandwidth

required for SPIN is the least because a lot of bandwidth is saved with negotiation using ADV and REQ packets and sending data only to nodes that requested it.



Graph 8

V. CONCLUSION:

SPIN is more energy efficient because there is no message explosion like Flooding & no redundant Data passing. Instead, nodes negotiate with control packets which are quite small as compared to data packets. Bandwidth requirements for SPIN protocol are much less as compared to Gossip protocol. Similarly, gossip is more energy and bandwidth efficient as compared to flooding because in flooding data packets are sent to all neighbors at every iteration whereas in gossip a node picks up a random neighbor and sends it the message which in turn picks up another node and sends message to it. Although gossip and flooding both suffer from redundancy but they are important when fault tolerance is required in case of node failures.

Thus, each protocol has its own advantages and disadvantages depending on the network condition, network dimension and fault tolerance requirements.

Division of Labor: Although there is no clear division of labor, everyone has contributed equally to the project and worked very hard to in order to complete it.

Team Member	Major tasks
Keerti Randhawa	Reviewing the literature and setting plan of action & Report.
Punam Mahato	Implementing algorithms & Report.
Shagun Arora	Analysis of data and report preparation & Report.

VI. REFERENCES:

[1] Rehena, Z.; Kumar, K.; Roy, S.; Mukherjee, N., "SPIN implementation in TinyOS environment using nesC," in *Computing Communication and Networking Technologies (ICCCNT), 2010 International Conference on* , vol., no., pp.1-6, 29-31 July 2010 doi: 10.1109/ICCCNT.2010.5591887

[2] url: https://en.wikipedia.org/wiki/Wireless_sensor_network

[3] url: https://en.wikipedia.org/wiki/Gossip_protocol

[4] Kempe, D.; Dobra, A.; Gehrke, J., "Gossip-based computation of aggregate information," in *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on* , vol., no., pp.482-491, 11-14 Oct. 2003 doi: 10.1109/SFCS.2003.1238221

[5] Huilong Huang; Hartman, J.H.; Hurst, T.N., "Data-Centric Routing in Sensor Networks using Biased Walk," in *Sensor and Ad Hoc Communications and Networks, 2006. SECON '06. 2006 3rd Annual IEEE Communications Society on* , vol.1, no., pp.1-9, 28-28 Sept. 2006 doi: 10.1109/SAHCN.2006.288403

[6] url: https://en.wikipedia.org/wiki/NesC

[7] Kempe, D.; Kleinberg, J., "Protocols and impossibility results for gossip-based communication mechanisms," in *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on* , vol., no., pp.471-480, 2002 doi: 10.1109/SFCS.2002.1181971

[8] Vigfusson, Y.; Birman, K.; Qi Huang; Nataraj, D.P., "GO: Platform support for gossip applications," in *Peer-to-Peer Computing, 2009. P2P '09. IEEE Ninth International Conference on* , vol., no., pp.222-231, 9-11 Sept. 2009 doi: 10.1109/P2P.2009.5284509

[9] Nath, R., "A TOSSIM based implementation and analysis of collection tree protocol in wireless sensor networks," in *Communications and Signal Processing (ICCSP), 2013 International Conference on* , vol., no., pp.484-488, 3-5 April 2013 doi: 10.1109/iccsp.2013.6577101

[10] url: http://www.tinyos.net/tinyos-1.x/doc/nido.pdf

[11] Long Cheng; Yu Gu; Tian He; Jianwei Niu, "Dynamic switching-based reliable flooding in low-duty-cycle wireless sensor networks," in *INFOCOM, 2013 Proceedings IEEE* , vol., no., pp.1393-1401, 14-19 April 2013 doi: 10.1109/INFCOM.2013.6566933