Implementation of Byzantine Algorithm in Multi-Hop Networks Using Power Based Routing

Shashank Juyal[1], Herat Gandhi[2], Shatadru Chattopadhyay[3],Gaurang Raval[4]
Institute of Technology, Nirma University
Ahmedabad, Gujarat, India
[1]07bit017@nirmauni.ac.in, [2]07bit012@nirmauni.ac.in
[3]07bit005@nirmauni.ac.in,[4]gaurang.raval@nirmauni.ac.in
[1]B.Tech Student, [2]B.Tech Student, [3]B.Tech Student, [4]Associate Professor

***Abstract*—Due to the increasing use of wireless sensor networks, Byzantine failure has assumed significant importance. In this paper we present a solution to the Byzantine failures occurring in multi-hop networks with power based routing. We present a Byzantine fault tolerant algorithm implementation on multi-hop wireless sensor networks. Byzantine fault tolerant algorithms are in general meant to be implemented upon mesh networks, but here working along with many other protocols such as Collection Tree Protocol (CTP) and Dissemination Protocols, we have an efficient algorithm that has been implemented on multi-hop networks.**

The implementation on multi-hop networks with power based routing has been based on the power demands of the motes. The packet is routed on the basis of the RSSI (Received Signal Strength Indicator) of the packet. When a packet can be routed through various motes, the mote with the maximum RSSI is chosen to be the next mote. And once the entire topology is identified then the algorithm is applied.

***Index Terms*—Sensor Networks, Nodes, Byzantine, Multi-hop, CTP, Dissemination Protocol, Power based Routing.**

I. Introduction

Sensors integrated into structures, machinery, and the environment, coupled with the efficient delivery of sensed information, could provide tremendous benefits to society. Potential benefits include: fewer catastrophic failures, conservation of natural resources, improved manufacturing productivity, improved emergency response, and enhanced homeland security. However, barriers to the widespread use of sensors in structures and machines remain. Bundles of lead wires and fiber optic 'tails' are subject to breakage and connector failures. Long wire bundles represent a significant installation and long term maintenance cost, limiting the number of sensors that may be deployed, and therefore reducing the overall quality of the data reported. Wireless sensing networks can eliminate these costs, easing installation and eliminating connectors.

Sensor networks-based monitoring applications range from simple data gathering, to complex Internet-based information systems. Either way, the physical space is instrumented with sensors extended with storage, computation and communication capabilities, the so-called motes. Motes run the network embedded programs that mainly sleep, and occasionally acquire, communicate, store and process data. There are various types of motes like mica2, micaz and telosb, all the testing that has been done of the algorithm discussed in this paper has been done over Crossbow micaz motes.

Wireless networks have various architectures like single hop, mesh or multi-hop. A multi-hop is a network where all the motes are not connected to one other and generally have a tree type structure. A mesh network allows for any node in the network to transmit to any other node in the network that is within its radio transmission range.

The term Byzantine[1] was first used for the type of failure where faulty motes may exhibit completely unconstrained behavior. The term Byzantine is intended as a pun–the battle scenario takes place in ancient Byzantium, and the behavior of some of the traitorous generals of the Byzantium Empire can only be described as 'Byzantine.'

The Byzantine Problem assumes that the network is an n-mote connected undirected graph with motes 1. . . n, where each mote knows the entire graph. Each mote starts with an input from a fixed value set V in a designated state component; we assume that, for each mote, there is exactly one start state containing each input value. The goal is for the motes to eventually output decisions from the set V, by setting special decision state components to values in V with the possibility that a limited number (at most f) of motes might fail.

The purpose of the paper is to explain implementation of byzantine algorithm on multi-hop wireless sensor networks. This kind of implementation is helpful in

[1] The term Byzantine was first used for the type of failure where faulty motes may exhibit completely unconstrained behavior. The term Byzantine is intended as a pun–the battle scenario takes place in ancient Byzantium, and the behavior of some of the traitorous generals of the Byzantium Empire can only be described as 'Byzantine'

solving the byzantine failure problem which generally occurs in wireless sensor networks.

II. Previous Work on Byzantine

In 1980, Pease et al. introduced the problem of reaching agreement among a group of correctly functioning processors in the presence of arbitrarily faulty processors; they proved that a minimum of 3t + 1 processors are needed to tolerate t faults. Two years later, they christened this the Byzantine Generals Problem, as it has been known ever since. A large body of research has addressed the problem of Byzantine agreement, the first decade of which is well surveyed by Barborak and Malek. In the mid-to-late 1990's, several researchers combined Byzantine-fault-tolerant protocols with state machine replication to produce toolkits such as Rampart, SecureRing, and BFT. These toolkits provide a replication substrate for services written as deterministic state machines. The substrate guarantees that the service will operate correctly as long as fewer than a third of its replicas are faulty. An unfortunate property of these toolkits is that their throughput scales negatively: As the group size grows, the system throughput shrinks, which is the exact opposite of the behavior desired for scalable systems.

All earlier systems all exhibit two properties: (1) nonpositive throughput scaling and (2) all-or-nothing failure semantics, meaning that failures beyond the tolerated threshold can cause the entire system to fail. In the absence of a Byzantine-fault-tolerant substrate that provides positive throughput scaling, researchers have built systems that partition their workload among multiple machines. However, as the system size grows, so does the expected number of faulty machines, which in turn - given all-or-nothing failure semantics - leads to an increasing likelihood of total system failure. We observe that this problem could be assuaged if there were some means to limit the spread of Byzantine faults. Three avenues of research are related to this problem:

First, several researchers have isolated Byzantine faults in distributed problems of academic interest, such as dining philosophers, vertex coloring and edge coloring . Their solutions employ self-stabilizing protocols to guarantee that correct results are eventually obtained by all nodes that are beyond a specified distance (the 'containment radius') from faulty nodes. The formal notion of fault containment for self-stabilizing systems was introduced by Ghosh et al. , who applied it only to transient faults. Such transient-fault containment was achieved by Demirbas et al. for the problem of tracking in sensor networks. None of this research offers a broad approach to containing Byzantine faults.

Second, a number of researchers have investigated ways to limit Byzantine corruption when performing broadcast, multicast , or gossip . These closely related problems have no computational aspect; they merely propagate data. Furthermore, they have the property that correct operation implicitly replicates all of the data to all machines. The resulting redundancy enables machines to vote on the data's correctness, as in the original Byzantine agreement problem.

Third, some researchers have tackled specialized subclasses of the general problem. Merideth proposed a proactive fault-containment system that relies on fault detection, a well-known specialization of fault tolerance problems. Krings and McQueen employ standard Byzantine-fault-tolerant protocols only for carefully defined 'critical functionalities'. The TTP/C protocol isolates only a constrained subset of Byzantine faults, namely reception failures and consistent transmission failures.

Thus, every known technique for building systems that resist Byzantine faults has at least one of the following weaknesses:

- Its throughput does not increase with scale.
- It addresses only a narrow academic problem.
- It does not support computation.
- It does not address general Byzantine faults.

III. Problem in detail

A reliable computer system must be able to cope with the failure of one or more of its components. A failed component may exhibit a type of behavior that is often overlooked-namely, sending conflicting information to different parts of the system. The problem of coping with this type of failure is expressed abstractly as the Byzantine Generals Problem.

This problem is built around an imaginary General who makes a decision to attack or retreat, and must communicate the decision to his lieutenants. A given number of these actors are traitors (possibly including the General.) Traitors cannot be relied upon to properly communicate orders; worse yet, they may actively alter messages in an attempt to subvert the process.

When we're not dwelling in storybook land, the generals are collectively known as processes (running in motes), the general who initiates the order is the source process, and the orders sent to the other processes are messages. Traitorous generals and lieutenants are faulty processes, and loyal generals and lieutenants are correct processes. The order to retreat or attack is a message with a single bit of information: a one or a zero.

In general, a solution to an agreement problem must pass three tests: termination, agreement, and validity. As applied to the Byzantine General's problem, these three tests are:

1. A solution has to guarantee that all correct processes eventually reach a decision regarding the value of the order they have been given.

2. All correct processes have to decide on the same value of the order they have been given.
3. If the source process is a correct process, all processes have to decide on the value that was original given by the source process.

Note that one interesting side effect of this is that if the source process is faulty, all other processes still have to agree on the same value. It doesn't matter what value they agree on, they simply all have to agree. So if the General is subversive, all lieutenants still have to come to a common, unanimous decision.

IV. Collection tree Protocol

A collection protocol builds and maintains minimum cost trees to nodes that advertise themselves as tree roots. Collection is address-free: when there are multiple base stations, it sends to the one with the minimum cost without knowing its address. In this paper, we assume all data packets are simple unicast frames. Rapidly changing link qualities cause nodes to have stale topology information, which can lead to routing loops and packet drops. This section presents two mechanisms that enable a routing protocol to be robust to stale route information and agile to link dynamics while also having a low overhead when the topology is stable.

The first is datapath validation using data packets to dynamically probe and validate the consistency of its routing topology.

The second is adaptive beaconing, which extends the Trickle code propagation algorithm so it can be applied to routing control traffic. Trickle's exponential timer allows nodes to send very few control beacons when the topology is consistent, yet quickly adapt when the datapath discovers a possible problem.

CTP is a tree-based collection protocol. Some number of motes in a network advertises themselves as tree roots. Nodes form a set of routing trees to these roots. CTP is address-free in that a node does not send a packet to a particular root; instead, it implicitly chooses a root by choosing a next hop. Nodes generate routes to roots using a routing gradient.

The CTP protocol assumes that the data link layer provides four things:

- Its throughput does not increase with scale.
- It addresses only a narrow academic problem.
- It does not support computation.
- It does not address general Byzantine faults.

CTP assumes that it has link quality estimates of some number of nearby neighbors. These provide an estimate of the number of transmissions it takes for the node to send a unicast packet whose acknowledgment is successfully received. It is best effort, but a best effort that tries very hard. CTP is designed for relatively low traffic rates. Bandwidth-limited systems might benefit from a different protocol, which can, for example, pack multiple small frames into a single data-link packet.

CTP uses expected transmissions (ETX) as its routing gradient. A root has an ETX of 0. The ETX of a node is the ETX of its parent plus the ETX of its link to its parent. This additive measure assumes that nodes use link-level retransmissions. Given a choice of valid routes, CTP SHOULD choose the one with the lowest ETX value. CTP represents ETX values as 16-bit fixed-point real numbers with a precision of hundredths. An ETX value of 451, for example, represents an ETX of 4.51, while an ETX value of 109 represents an ETX of 1.09.

Routing loops are a problem that can emerge in a CTP network. Routing loops generally occur when a node choose a new route that has a significantly higher ETX than its old one, perhaps in response to losing connectivity with a candidate parent. If the new route includes a node which was a descendant, then a loop occurs.

CTP addresses loops through two mechanisms. First, every CTP packet contains a node's current gradient value. If CTP receives a data frame with a gradient value lower than its own, then this indicates that there is an inconsistency in the tree. CTP tries to resolve the inconsistency by broadcasting a beacon frame, with the hope that the node which sent the data frame will hear it and adjust its routes accordingly. If a collection of nodes is separated from the rest of the network, then they will form a loop whose ETX increases forever. CTP's second mechanism is to not consider routes with an ETX higher than a reasonable constant. The value of this constant is implementation dependent.

Packet duplication is an additional problem that can occur in CTP. Packet duplication occurs when a node receives a data frame successfully and transmits an ACK, but the ACK is not received. The sender retransmits the packet, and the receiver receives it a second time. This can have disasterous effects over multiple hops, as the duplication is exponential. For example, if each hop on average produces one duplicate, then on the first hop there will be two packets, on the second there will be four, on the third there will be eight, etc.

Routing loops complicate duplicate suppression, as a routing loop may cause a node to legitimately receive a packet more than once. Therefore, if a node suppresses duplicates based solely on originating address and sequence number, packets in routing loops may be dropped. CTP data frames therefore have an additional time has lived (THL) field, which the routing layer increments on each hop. A link-level retransmission has the same THL value, while a looped version of the packet is unlikely to do so.

V. Dissemination Protocol

There are various kinds of Dissemination Protocols in existence. The major ones that are used with sensor motes are

A. Drip

It is the most basic of all dissemination protocols. Each data item is independently advertised and disseminated. Metadata is not shared among data items, meaning nodes do not need to agree on data sets a priori. The application itself is responsible for enabling or disabling the radio communication. Trickle timers are used and the data size should be less than the message payload size. It is the most efficient since the motes do not transmit values to the next mote till they receive the changed value from the last mote hence this is the most efficient protocol.

With the ever spreading and heightening of interest with this technology larger networks are emerging leaving Drip as an unmanageable protocol in these scenarios. On smaller scale applications however Drip can be seen to be more efficient than some of the other more advanced protocols, saving on energy and messages transmitted due to their more advanced and costly algorithms and scanning techniques, leaving Drip room to be a contender to be used in some edge cases and topologies.

B. DIP

It is a modification of the Drip protocol. Advertisement messages are used for a fixed data set meaning all nodes must agree on a fixed set of data item identifiers before dissemination. The application itself is responsible again for enabling or disabling the radio communication. All the data items meant for various items use a single trickle timer and are combined into one.

Efficiency is improved by using hashes over key space ranges to determine there are differing versions and then a bloom filter is applied to locate the differing values. Theoretically this works well, however real life networks tend to yield some problems with the Search algorithm over a less realistic model. A big issue is packet loss; this can make navigation of the tree difficult. The second issue stems from this packet loss issue, inordinate advertisements are generated when packets are lost. This is where the dynamic nature of the DIP protocol comes into play.

C. DHV

It is the newest protocol and was added as a protocol in 2009. Advertisement messages are used for a fixed data set meaning all nodes must agree on a fixed set of data item identifiers before dissemination. DHV automatically starts the radio. It also uses a single trickle timer for all data items.

This protocol is based on the consideration that when new code is being propagated around a network, the older and newer codes version numbers only vary in several least significant bits of their binary representation. By virtue of this observation DHV provides the functionality to a node of selecting much less data bits and transmitting these throughout a network when wishing to see if the network is up to date or needs an update.

In wireless sensor networks, power used by motes is the most important parameter to be taken into consideration. Motes run on batteries so if they use more power than it would be necessary to change batteries frequently. Motes are deployed on very large area like forest, desert, etc. Changing batteries will be costly. In order to optimize power usage developer must make application power efficient.

All three alternatives: DIP, DRIP and DHV deliver packets from one mote to all other motes in the network. But we need to select most efficient one. The graph in Fig-1 shows power analysis of three protocols. We can infer from this graph that DIP uses the least power from three of them. So we have used DIP in our application.

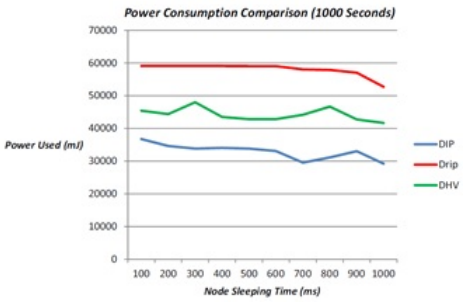

Fig. 1. Comparison between DIP,DRIP and DHV

VI. Receive Signal Strength Indicator (RSSI)

RSSI is an acronym for Received Signal Strength Indication. It is a measure of the signal power on the radio link, usually in the units of dBm, while a message is being received. It can be used for estimating node connectivity and node distance (although the relation between distance and RSSI is noisy and not straightforward), among other things. Another usage of RSSI is to sample the channel power when no node is transmitting to estimate the background noise, also known as noise floor.

According to TinyOS' HAA the RSSI data is not provided by the standard platform independent Hardware Interface Layer. RSSI must be accessed by a platform-specific HAL interface, as in the case of the CC2420 or by a specific field of the message_t struct as defined in the platform_message.h like in the case of the CC1000.

The RSSI values given by TinyOS are usually not in

dBm units, and should be converted by the platform-specific relation to get meaningful data out of it.

For CC2420: The RSSI is a 5-bit value indicating the receive power in the selected channel, in steps of 3 dB. No attempt is made to distinguish between IEEE 802.15.4 signal and other signal source, only the received signal power is evaluated. The RSSI provides the basis for Energy Detection measurement.

Using the Basic Operating Mode, the RSSI value is valid at any RX state, and is updated every 2 microsec. The current RSSI value is stored to the PHY_RSSI register. Note, it is not recommended to read the RSSI value when using the Extended Operating Mode.

The RSSI value is always averaged over 8 symbol periods (128 microsec). The RSSI_VALID status bit indicates when the RSSI value is valid, meaning that the receiver has been enabled for at least 8 symbol periods.

The RSSI register value RSSI.RSSI_VAL can be referred to the power P at the RF pins by using the following equations: P = RSSI_VAL + RSSI_OFFSET [dBm]

where the RSSI_OFFSET is found empirically during system development from the front end gain. RSSI_OFFSET is approximately -45. E.g. if reading a value of -20 from the RSSI register, the RF input power is approximately -65 dBm.

VII. POWER BASED ROUTING

The byzantine fault tolerant algorithm is generally implemented over mesh network because this algorithm requires a mote to broadcast a packet to all motes in the network. In mesh network all motes are directly connected to each other so this kind of broadcasting will be very easy. If we have multi-hop network then global broadcasting is not possible because all motes are not directly connected to each other. In multi-hop networks local broadcast is possible in which each mote sends packet to its neighbors.

Using this fact we can design a power efficient protocol for global broadcast. Here the whole protocol is divided in different rounds:

- In first round, each mote locally broadcasts HELO packet.
- In second round, receiving HELO packet from neighbor mote sends ACK packet to sender mote. After receiving ACK packet mote measures its RSSI value and stores all necessary information. Each mote also stores which mote can be selected as its parent.
- In third round, information collected from the previous stage is sent to base station and base station processes this information and creates routing table.
- In fourth round, base station sends pruned routing table to all motes.

The fourth round requires somewhat explanation. Pruning refers to deleting unnecessary entries from routing table. When routing table is constructed at base station, it sends this routing table to all motes in the network. When it forwards this table to motes, they will prune entries of motes which are at same level or higher level in the given graph. This will enable motes to communicate efficiently. Using this table each mote will know how much power is required to transmit a packet and for global broadcast to which mote it should send packet. This method will allow power efficient global broadcast in the whole wireless sensor network.

VIII. Packet Structure Used

1. HELO packet
   The HELO packet is a 16bit packet that contains the node id of the mote that is transmitting the packet. Through this the base station or the root of the tree in the Collection Tree Protocol receives the mote id of the mote that sent it the HELO packet.

2. INIT packet
   This is a value that is disseminated over the network using the Drip protocol. The value in this case would be a 0. If a mote receives a disseminated value of 0 then it behaves as having received an INIT packet.

3. ECHO packet
   This is a value that is disseminated over the network using the Drip protocol and is same as the mote id of the mote being the source of the dissemination, hence if there is any value other than 0 received by any mote it is regarded as an ECHO packet.

4. FRNDS packet
   It is a packet which carries information about mote's neighbors and power required to send packet to its neighbors.

IX. Algorithm

The algorithm that has been presented in this paper implements a consistent broadcast protocol for implementing byzantine fault tolerant algorithms.

The consistent broadcast mechanism is required to satisfy the following three conditions:

1. If non-faulty mote i broadcasts message (m, i, r) in round r, then the message is accepted by all non-faulty motes by round r + 1 (i.e., it is either accepted at round r or round r + 1).

2. If non-faulty mote i does not broadcast message (m, i, r) in round r, then (m, i, r) is never accepted by

any non-faulty mote.

3. If any message (m, i, r) is accepted by any non-faulty mote j, say at round r, then it is accepted by all non-faulty motes by round r + 1.

The first condition says that non-faulty motes' broadcasts are accepted quickly, while the second says that no messages are ever falsely attributed to non-faulty motes. The third condition says that any message that is accepted by a non-faulty mote (whether from a faulty or non-faulty sender) must also be accepted by every other non-faulty mote soon thereafter.

The algorithm is divided in different stages:

- **First stage:**
  All motes send HELO packet to base station using CTP and base station finds total number of motes in network as shown in Fig-2.

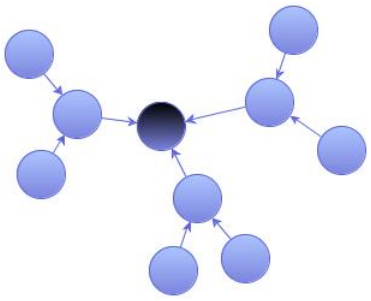

Fig. 2. Motes sending HELO packet to Base Station

- **Second stage:**
  Base station sends 'n (number of motes)' to all motes using Dissemination as shown in Fig-3.

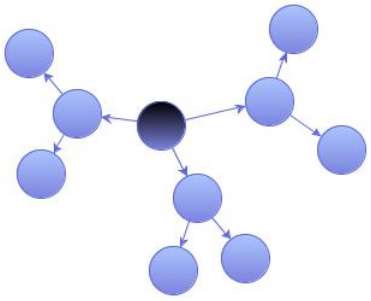

Fig. 3. Base Station sending Number of Motes in the network to all motes

- **Third stage:**
  Each mote sends HELO packet to its neighbor as shown in Fig-4.

- **Fourth stage:**
  After receiving HELO packet from neighbor mote sends ACK packet to sender mote. After receiving ACK packet mote measures its RSSI value and stores all necessary information. Each mote also stores which mote can be selected as its parent.

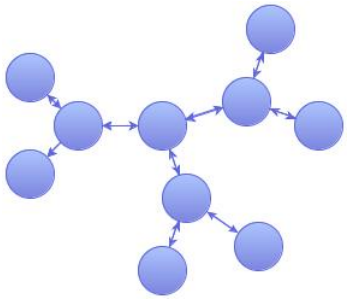

Fig. 4. Motes sending HELO packet to each other

- **Fifth stage:**
  Information collected from the previous stage is sent to base station through FRNDS packet and base station processes this information and creates routing table.

- **Sixth stage:**
  Base station sends pruned routing table to all motes.

- **Seventh stage:**
  In this stage one node will be randomly selected and will disseminate INIT packet using pruned routing table.

- **Eighth stage:**
  After receiving INIT packet, motes send ECHO packet. Motes keep counter of ECHO packets.

- **Ninth stage:**
  If any mote has received n-f ECHO packets in previous round then it establishes successful communication.

X. Results

The algorithm was simulated for a varied number of motes which were 10, 25, 50 and 100 in number. The chief parameter of the mote that is analyzed is its Power and the time consumed for the entire algorithm to execute. Energy is measured using PowerTOSSIM-Z and is then graphically plotted in Fig.5:

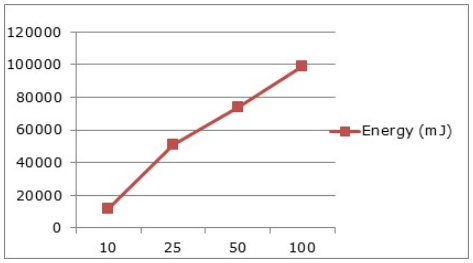

Fig. 5. Energy vs. Number of Motes

The energy has been measured in millijoules (mj). Y-

axis in the graph indicates Energy consumed by the motes while X-axis indicates number of motes used for simulation. We can discern from the graph that as the number of motes increases, the total energy consumption of the motes also increases. As more and more motes are added, the energy consumed increases non-linearly. The Time consumed during the algorithm is measured and graphically plotted in Fig.6:

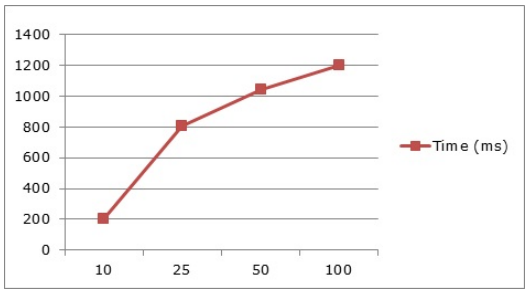

Fig. 6. Time vs. Number of Motes

Y-axis in the graph indicates the time taken by the motes for simulation while X-axis indicates the number of motes used for simulation. As the number of motes increases, this graph clearly shows the increase in the amount of time consumed by the algorithm in seconds.

XI. Conclusion

Our algorithm implementation detects faulty motes in multi-hop networks using power based routing and is scalable upto a large number of motes. Though it has been tested over Crossbow micaz motes, it is applicable over motes of any make. This type of algorithm would help ensure that the Byzantine problem that plagues huge sensor networks in general can be reduced, and motes that are faulty, be isolated and yet the network may function as a robust network.

References

[1] David Gay and Philip Levis, *The NesC language: A holistic approach to network embedded systems*, ACM 2003
[2] Nancy Lynch, *Distributed Algorithms*, Morgan Kaufmann Publishers, Inc., 1997
[3] Philip Levis, *TOSSIM: Accurate and Scalable simulation of entire TinyOS applications*,
[4] Enrico Perla, Art O Cathain, *PowerTOSSIM z: Realistic Energy Modeling for Wireless Sensor Network Environments*,
[5] Jessica Staddon, Dirk Balfanz and Glenn Durfee, *Efficient Tracing of Failed Nodes in Sensor Networks*, Palo Alto reasearch Center, October 1 2002.
[6] Thomas Clouqueur, *Fault Tolerance in Collaborative Sensor Networks for Target Detection*, IEEE Conference, March 2004
[7] Carolos Livadas and Nancy A. Lynch, *A Reliable Broadcast Scheme for Sensor Networks*,
[8] Nicolas Burri, *YETI : A Tinos plugin for eclipse*, June 19,2006
[9] H.A. Ali, *An efficient relative broadcast algorithm in Adhoc networks based on Self-Pruning*, Egypt