Sensor Networks

Anupama Lath, Rajeswari Harikrishnan and Shane Eisenman

Project for ELEN 6951: Wireless and Mobile Networking II

May 6, 2002

Part I: Reliable Multicast

Anupama Lath and Shane Eisenman
Advisor: Chieh-Yih Wan

Part II: Small World Problem

Rajeswari Harikrishnan
Advisor: Chieh-Yih Wan

Part I: Reliable Multicast

1. Introduction:

A sensor network can be defined as a self-organizing collection of peer nodes that work together to perform some application-specific task. The main task of a sensor network is to gather information on events of interest, possibly perform aggregation or other processing, and channel this information to a data sink. Other works have proposed paradigms, such as directed diffusion [1] and SPIN [2] to govern communication of data from information sources to information sinks. However, each of these protocols provides unacknowledged packet transfer at the transport/application layer. This project investigates one method of providing reliable dissemination of data from source to sink using SRM (Scalable Reliable Multicast) [3]. SRM uses three messages for reliable delivery, Session, Request and Repair. Briefly, session messages are sent by each node of a multicast group to inform members of the last data message received by this node. Session messages are time-stamped and their exchange is also used to calculate the delay between pairs of nodes. Request messages are multicast by a node when it discovers that it is missing a data message. Its complement is the repair message, which fills the missing data indicated by a request, and may be sent not only by the original source but by any node able to fill the request. SRM figures to make a good match with sensor networks due to the common application-specific nature of each.

2. Implementation:

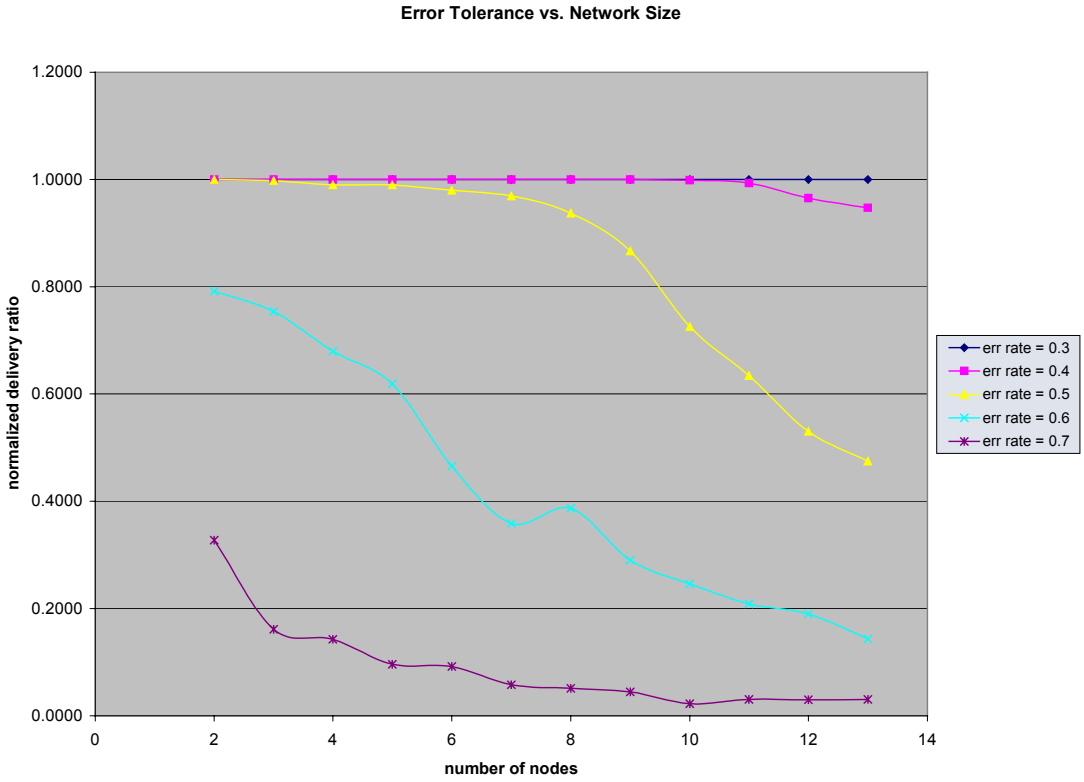
This investigation makes use of the NS-2 network simulator, version 2.1b8a, which is the most recent release at the time of this investigation. With this release, NS-2 only supports SRM for wired networks. As sensor networks are not wired but use either a wireless or optical transmission medium, we have extended the capabilities of the simulator to support wireless SRM. Specifically, we have layered those portions of the current SRM implementation that deal with reliable delivery of data over an omniscient multicast routing layer. Omniscient multicast offers the advantage that it is not dependent on fixed multicast trees, as could be defined in fixed, wired topologies, but routes packets based on definition of information sinks. At each node, when the router layer receives a packet, it decides whether to pass the packet up the stack based on the sink table it maintains. Omniscient multicast is unrealistic in that it assumes all route information is available at no cost. We use it here as a first approximation of SRM performance for wireless sensor networks. Sink tables are defined separately for different data types, which has the potential to act as the data naming schemes in directed diffusion and SPIN do. In this, our initial investigation we use only one data type with a single source. We use an 802.11 MAC layer and a wireless physical layer. The code-level implementation details can be found in section 4.

3. Results:

Several simulations were run to gain some insight into how SRM performed in delivering data. For this project we present results in three areas: error tolerance as a function of topology size, message overhead as a function of channel error rate, and network delay as a function of channel error rate. Error tolerance is represented by the average data packet delivery ratio per node. Results were obtained for various channel error rates. The average total packets sent in the process of delivering the transmitted SRM data packets represent message overhead in our simulations. The numerical results are normalized by the number of data messages sent. Finally, the network delay is measured as the time elapsed from the transmission of the first data packet until the reception of the last packet by the last node in the topology. A line topology was used in all simulations.

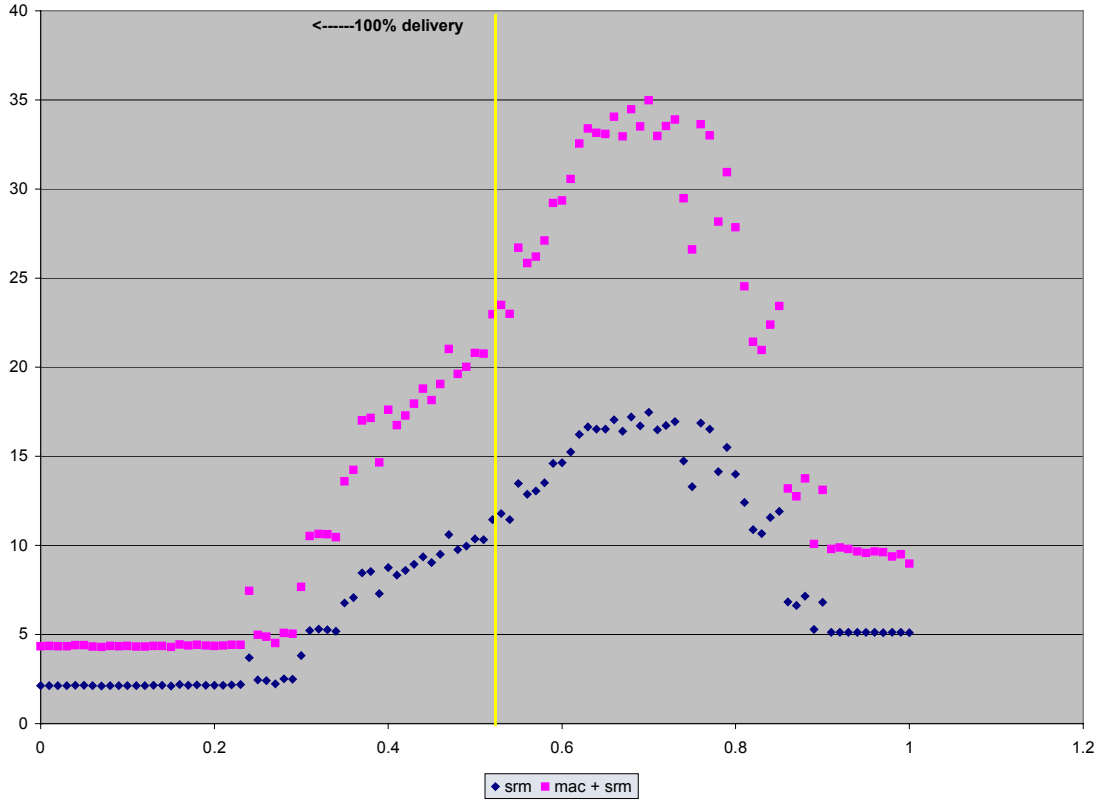
In each case, simulations were run five times and the results were averaged. The SRM source started sending data packets at a constant rate 2 seconds after the simulation began and stopped after 0.5 seconds, sending 50 packets in all. The simulation ran for 50 seconds after the source stopped sending data packets. This arbitrary cutoff point was chosen as time after which delivery of data would be worthless. Of course, this is an application-specific parameter.

The results for error tolerance are shown in Figure 1. Note that, as one might expect, the average delivery ratio decreases as channel error rate increases due to the increased packet loss. Also as the number of nodes increases the average delivery ratio decreases because of increased congestion.



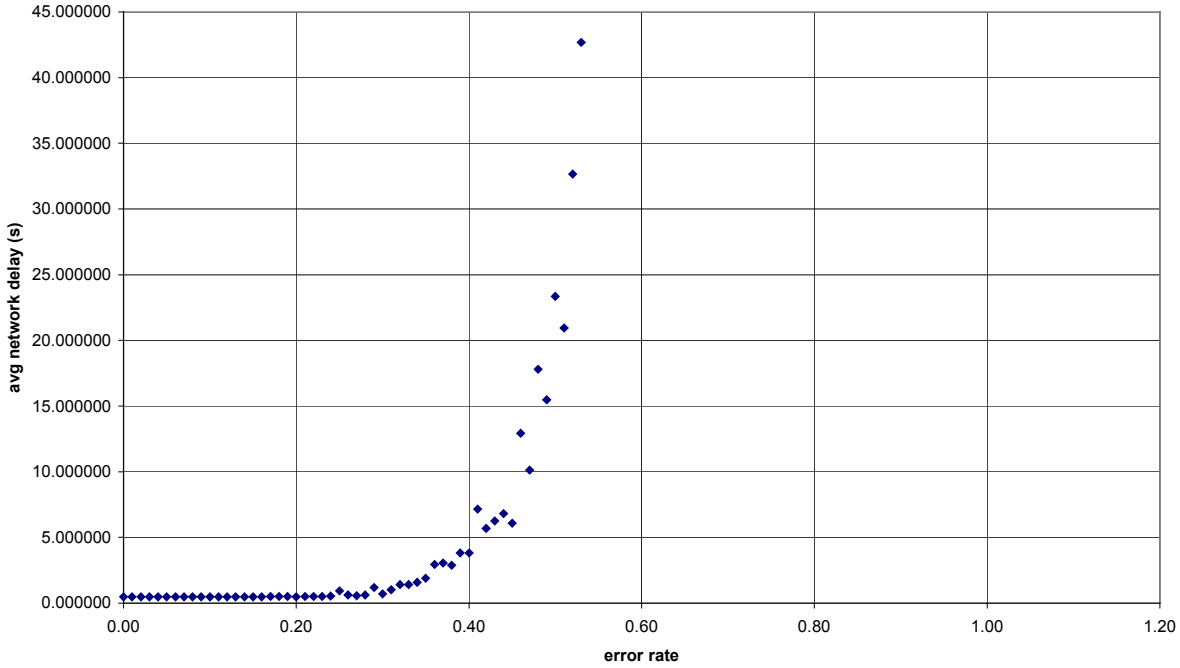
T

The results for message overhead are shown in Figure 2. There are four distinct portions of the curve visible. From 0 to .30 error rate the curve is linear with 0 slope; from .30 up to .70 channel error rate the number of messages sent increases exponentially; thereafter, the number of messages sent decreases linearly to a some absolute minimum around .90 error rate; from .90 to 1 the curve is again linear with 0 slope. Further, the 100% delivery point is marked on the plot with a vertical yellow line. This marks the error rate, .52 in this simulation, at which all nodes in the topology did not receive all data packets by the time the simulation ended. These results can be explained as follows. In the first part of the curve, the MAC layer mechanisms (RTS, CTS, ACK) are able to handle all the packet loss and there is no need for the SRM layer mechanisms (REQ, REP) to be used. From about .3 to .5 there is an increase in messages as packet loss overwhelms the MAC layer mechanisms and the reliable delivery mechanisms of SRM must be used to maintain 100% delivery. After this point, message growth continues, but despite SRM all packets cannot be delivered to all sinks. After about .7, the number of messages steeply declines. Because the error rate is so high, the MAC layer RTS-CTS exchange is rarely completed and, therefore, data packet transmission becomes much more sparse. Note that this simulation was done using a 3-node topology.



The results for network delay are shown in Figure 3. As expected, the delay shows sharp exponential growth as error rate is increase, reflecting the packet loss incurred both by increased congestion due to MAC and SRM retransmissions and the increasing error rate itself. As defined, network delay only has meaning when all nodes receive all data packets before the simulation stops. Thus, we see no points after approximately .50 error rate because from this point onward, not all nodes receive all the packets. Note that this simulation was done using a 3-node topology.

Figure 3. Network Delay



4. Code-Level Implementation Details:

Following is a detailed description of what changes were made to the various associated NS-2 files, including line numbers. Files are listed in alphabetical order. Printouts of these files are included as an appendix. In sum, 1632 lines were added.

arp.cc:

--(L63-L65)-- #include statement added to support ARP message trace
--(L253-L255)-- ARP message trace printf statements added
--(L348-L350)-- ARP message trace printf statements added

cmu_trace.cc:

--(L555-L569)-- added function format_srm() to support different colors for different SRM traffic types in NAM
--(L640-L643)-- added call to function format_srm()
--(L766-L768)-- added PT_SRM case (support for SRM header type) to function CMUtrace::format()
--(L649,L662)-- corrected to remove extra space at end of line that was causing error when running NAM

cmu-trace.h:

--(L116-L117)-- added entry for funtion format_srm() in CMUtrace class definition

errmodel.cc:

--(L58-L61)-- #include statement added to support ERR message trace

--(L114-L115)-- initialize error message counter variable in ErrorModel class constructor

--(L186-L191)-- ERR message trace printf statements added (to trace corrupted packets

errmodel.h:

--(L96-L97)-- added ErrorModel class variable corrupt_pkt_cnt_

god.cc:

--(L225-L297)-- added several print statement in function God::Dump() to enhance readability and understanding of the output

--(L1008-L1015)-- added ability to call God::Fill_for_Source from Tcl script, for debug purposes

--(L1016-L1023)-- added ability to call God::Fill_for_Sink from Tcl script, for debug purposes

mac-802_11.cc:

--(L54-L56)-- #include statement added to support MAC message trace

--(L182-L184)-- added Tcl binding and initialization of MAC message counter variable to the Mac802_11 class contructor

--(L690-L692)-- MAC message trace printf statements added

--(L745-L747)-- MAC message trace printf statements added

--(L787-L789)-- MAC message trace printf statements added

--(L826-L828)-- MAC message trace printf statements added

--(L862-L864)-- MAC message trace printf statements added

--(L920-L922)-- MAC message trace printf statements added

--(L1046-L1047)-- increment MAC message counter variable

--(L1233-L1235)-- MAC message trace printf statements added

--(L1266-L1268)-- MAC message trace printf statements added

--(L1312-L1314)-- MAC message trace printf statements added

--(L1407-L1409)-- MAC message trace printf statements added

mac-802_11.h:

--(L402-L403)-- added Mac802_11 class variable mac_sent_cnt_

diffusion/omni_mcast.cc:

--(L62-L64)-- #include statements added to support RTR message trace and Tcl call function "getsrmHandle

--(L194-L196)-- added Tcl binding and initialization of MAC message counter variable to the OmniMcastAgent class contructor

--(L224,L225,L289-L304)-- modified OmniMcastAgent::ConsiderNew() function to either pass packet up to SRM if it is new and not generated by THIS_NODE, or send to MAC for broadcast if new and was generated by THIS_NODE

--(L409-L411)-- RTR message trace printf statements added

--(L418-L419)-- increment RTR message counter variable

--(L603-L607)-- added Tcl command interface to OmniMcastAgent::dumpRoutingTable()

--(L611-L615)-- added Tcl command "getsrmHandle" to return Tcl object handle of SRM agent created in srm-omni.tcl

--(L667-L678)-- added Tcl command interface to OmniMcastAgent::addsink() and addsource() methods

--(L718-L741)-- added command method dumpRoutingTable() to print the local OmniMcast agent routing_table variable

--(L743-L767)-- added OmniMcastAgent::addsink() and addsource() to manipulate the local OmniMcast agent routing_table variable

diffusion/omni_mcast.h:

--(L151-L153)-- added class variable srmHandle_ to support Tcl command "getsrmHandle" and class variable sent_msg_ctr_ to log data

--(L180-L182)-- added entries for methods dumpRoutingTable(), addsink() and addsource() to OmniMcastAgent class definition

tcl/test/scen-800x800-2-500-1.0-1:

--(L1-L17)-- created this file to define node position and shortest path information for 2 nodes

tcl/test/scen-800x800-3-500-1.0-1:

--(L1-L22)-- created this file to define node position and shortest path information for 3 nodes

tcl/test/scen-800x800-4-500-1.0-1:

--(L1-L27)-- created this file to define node position and shortest path information for 4 nodes

tcl/test/scen-800x800-5-500-1.0-1:

--(L1-L34)-- created this file to define node position and shortest path information for 5 nodes

tcl/test/scen-800x800-6-500-1.0-1:

--(L1-L42)-- created this file to define node position and shortest path information for 6 nodes

tcl/test/scen-800x800-7-500-1.0-1:

--(L1-L51)-- created this file to define node position and shortest path information for 7 nodes

tcl/test/scen-800x800-8-500-1.0-1:

--(L1-L61)-- created this file to define node position and shortest path information for 8 nodes

tcl/test/scen-800x800-9-500-1.0-1:

--(L1-L72)-- created this file to define node position and shortest path information for 9 nodes

tcl/test/scen-800x800-10-500-1.0-1:

--(L1-L84)-- created this file to define node position and shortest path information for 10 nodes

tcl/test/scen-800x800-11-500-1.0-1:

--(L1-L97)-- created this file to define node position and shortest path information for 11 nodes

tcl/test/scen-800x800-12-500-1.0-1:

--(L1-L111)-- created this file to define node position and shortest path information for 12 nodes

tcl/test/scen-800x800-13-500-1.0-1:

--(L1-L126)-- created this file to define node position and shortest path information for 13 nodes

srm.cc:

--(L46-L48)-- #include statement added to support AGT message trace

--(L97-L103)-- added Tcl binding and initialization of AGT message counter variable and other data logging and packet tracking variables to the OmniMcastAgent class contructor

--(L173-L175)-- AGT message trace printf statements added

--(L182-L183)-- call SRMAgent::addDiffusionHeader() to add necessary diffusion header fields for use with omniscient multicast routing layer

--(L188-L194)-- AGT message trace printf statements added

--(L253-L254)-- call SRMAgent::addDiffusionHeader() to add necessary diffusion header fields for use with omniscient multicast routing layer

--(L258-L260)-- AGT message trace printf statements added

--(L282-L283)-- call SRMAgent::addDiffusionHeader() to add necessary diffusion header fields for use with omniscient multicast routing layer

--(L285-L288)-- AGT message trace printf statements added

--(L299-L303)-- increment received data counter and log time to Tcl if all received

--(L369-L370)-- call SRMAgent::addDiffusionHeader() to add necessary diffusion header fields for use with omniscient multicast routing layer

--(L371-L373)-- AGT message trace printf statements added

--(L428-L433)-- AGT message trace printf statements added (SRMinfo->distance calculation message)

srm.h:

--(L44-L45)-- #include statement added to support SRMAgent::addDiffusionHeader() method

--(L59-L63)-- added various SRMAgent class variables for data logging purposes

--(L102-L120)-- implemented SRMAgent::addDiffusionHeader()

tcl/mcast/srm.tcl:

```
--(L51-L52)-- added Agent/SRM variable data_ctr_ to log received
  SRM data messsages
--(L126-L127)-- commented reference to join-group in the start()
  method
--(L203-L207)-- in recv-data: increment data_ctr_ and return
  simulation time if last dat message received
--(L535-L539)-- in recv-repr: increment data_ctr_ and return
  simulation time if last dat message received
```

tcl/test/srm-omni.h:

```
--(L1-L15)-- created to manage message trace for printf
  statements added to srm.cc, omni_mcast.cc, arp.cc, errmodel and
  mac-802_11.cc
```

tcl/test/srm-omni.tcl:

```
--(L1-L443)-- created this file to act as top level simlulation
  script (create ns, god, nodes, config nodes, run, etc.)
```

tcl/test/traffic.tcl:

```
--(L1-L57)-- created this file to define traffic sources and
  sinks, etc.
```

5. Future Work:

This project represents only the first step in the investigation of SRM as a legitimate means to achieve reliable multicast in sensor networks. Several interesting enhancements can be made to the existing code to broaden the scope of the results. These include running simulations: with a more random topology configuration than just the line topology simulated here, without the support of the RTS-CTS MAC mechanism and while introducing node movement.

6. References:

[1] C. Intanagonwiwat, RC. Govindan, D. Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks," 6th Conf. on Mobile Computing and Networking, August 6 - 11, 2000, Boston, MA, USA.

[2] W. R. Heinzelman, J. Kulik, H. Balakrishnan, "Adaptive Protocols for Information Dissemination in Wireless Sensor Networks," Proc. Mobicom '99, pp 174-185.

[3] S. Floyd, V. Jacobson, C.-G. Liu, S. McCanne, L. Zhang, "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing," IEEE/ACM Transactions, December 1997.

[4] K. Varadhan, K. Fall, "The *ns* Manual (formerly *ns* Notes and Documentation)," The VINT Project, http://www.isi.edu/nsnam/ns/ns-documentation.html.

[5] S. Eisenman, "Packet Loss and Recovery in NS-2 SRM," Internal Project Memo, 2002.

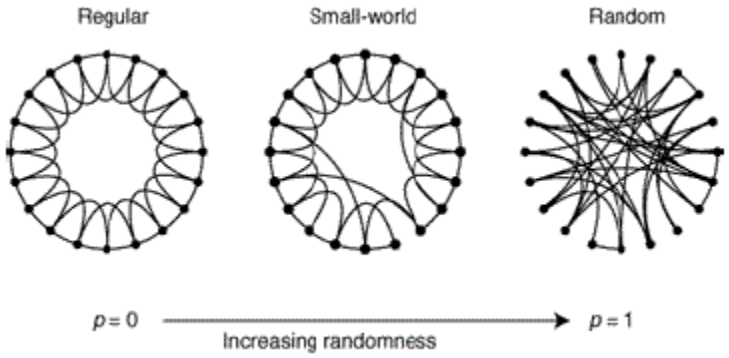
[6] A. Lath, "Omniscient Multicast," Internal Project Memo, 2002.

Part II: Small World Problem

1. Introduction:

The small world phenomenon is the principle that no matter how large a network may be, any pair of nodes is linked by short chains of acquaintances. This concept, originally defined for a network of people (by Harvard social psychologist Stanley Milgram), can be applied to any network of "relationships" with "six degrees of separation" between any two members of the network [1]. A number of network models have been proposed to study this phenomenon, one of which is the Watts-Strogatz (WS) model. Watts-Strogatz proposed a model in which a uniform ring lattice, where every node is connected to *k* of its neighbors (*k/2* on either side), is "rewired" with probability *p* i.e. an existing edge from a node to any of its *k* neighbors is replaced by another edge to a node in some other neighborhood [1].

Figure 1[1]. Rewiring of uniform lattice to form small world network;
p = rewiring probability



The effect of rewiring on the network is analogous to the case of relationships among people, where an individual has several friends in his immediate vicinity and neighborhood, but also has contact with other friends in more distant locations (though fewer in number).

Rewiring has some interesting consequences on the network. As the probability of rewiring increases, the average shortest distance between any two nodes in the network starts falling – possibly significantly. This can be understood by interpreting rewiring as establishing "short-cuts" between distant nodes that were hitherto not connected. Thus creating a single new connection or "short-cut" between two distant nodes in the network causes several shortest paths to collapse, thereby reducing the average shortest distance in the network.

[1] Source : [1]

We used the above idea of the small world phenomenon in sensor networks. Applying the idea to wireless networks is appropriate because nodes are typically aware only of their immediate neighbors, within a certain radius (dictated by the transmission capabilities of the nodes). Communication between two nodes that are not directly connected can be established via a sequence of hops (A can talk to B, B can talk to C, C can talk to D; hence A can "talk" to D). The shortest path from A to D then would be the path from A to B in the least number of hops. Note that these hops are logical and not entirely physical.[2] Also, note that we are dealing with a directed graph of nodes because communication is *not* bi-directional i.e. if A can talk to B, B need not necessarily be able to talk back to A and the same applies to "talk"ing as well i.e. if a shortest path exists from A to D, it is not necessary that a shortest path exists from D to A.

This is an outcome of the limitations imposed by the transmission capabilities of the nodes and their physical separation or location in the network. Thus, in spite of the small world phenomenon, it may still not be possible for every node to be able to communicate with every other node in the network; also, some nodes may be isolated, groups of nodes may be isolated and some nodes may be able to receive messages but not transmit (and vice versa). Thus the average shortest distance in the network which is defined as

Average shortest distance = (sum of all finite shortest distances between any pair of nodes in the network) / (the number of such pairs)

would be quite meaningless in itself because it would not convey any information about the presence of isolated nodes or isolated groups or about the network topology.

Hence, new system metrics were defined to gain a better understanding of how the shortest distances between nodes in the network are affected when rewiring is performed. The rewiring in this case was effected by increasing the number of powerful transmitters in the network, thereby increasing the number of nodes that would fall within their transmission radius (and hence establishing new connections).

Since communication is directed, and not all nodes can talk to each other, some being able to only receive or only transmit, the nodes were classified as *transmitter*, *receiver_only*, and *isolated*. A node is a *transmitter* if it can transmit to at least one other node i.e. at least one other node falls within its transmit radius. A node is *receiver_only* if it cannot transmit to any other node but can receive from at least one *transmitter* i.e. no other node falls within its transmit radius but it falls under the transmit radius of at least one other node. An *isolated* node is one which can neither receive nor transmit from/to any other node in the network. In addition to the above classification of nodes, some nodes can be

[2] Though the hops are logical, they are not entirely independent of physical distance because limitations to transmission capabilities are imposed by distance in the wireless environment.

termed as *God* nodes – these are nodes that have a finite shortest distance from every other node in the network i.e. they are capable of communicating with every other node either directly or via a sequence of hops, through other nodes. *Transmitter* nodes that are not *God*s are *grouped* together to form disjoint strongly-connected groups i.e. two *transmitters* A and B belong to the same group if A can "talk"/talk to B and B can "talk"/talk to A. This division of the graph (rather of non-*God transmitters*) into groups helps us to identify clusters of nodes in the network. A group G1 is said to be able to talk to a group G2 if at least one node in G1 can "talk"/talk to at least one node in G2). (Note that if G1 can "talk"/talk to G2 $\Rightarrow$ G2 cannot "talk"/talk back because if it could, then G1 and G2 would not be two separate groups in the first place). Clearly, groups may or may not be able talk to each other, and could even be isolated.

Thus, a kind of hierarchy is observed in the network with the *God* nodes at the top, followed by groups of *transmitter* nodes (each group is strongly connected and the groups are all disjoint) and a third layer of *receiver_only* nodes that may be transmitted to by *transmitters* belonging to one or more groups. The isolated nodes can be ignored for all practical purposes.

2. Implementation:

Programs were written in C++ to generate a network of 100 nodes in an area of dimensions 100m x 100m with random node positions (uniformly distributed – a valid assumption since we are modeling a real-life scenario of sensors being scattered in the area under observation). The transmission radii chosen were 50m for powerful transmitters and 10m for other transmitters. The number of powerful transmitters was varied from 1% to 5% of the total number of nodes in the network i.e. one node was randomly made a powerful node and successive powerful transmitters were added (again picked randomly). Different system metrics were then studied and their variation with increasing number of powerful transmitters was observed.

The shortest distances for the network were computed using the Floyd-Warshall algorithm which has an $O(N^3)$. The partitioning of the graph into groups was performed using the computed shortest distances.

3. Results:

The average shortest path was computed for each of the 5 cases with number of powerful transmitters varied from 1 through 5. Though one would expect the average shortest distance to decrease with increasing number of powerful transmitters, the average shortest path actually increases. This can be explained by the fact that as we increase the number of powerful transmitters in the network, the number of pairs of nodes that can "talk"/talk to each other increases significantly (see graph XYZ). Hence the number of new distances that are being added must be incorporated because the usual definition of average shortest

path does not take this into account. Hence a new system metric *normalized average shortest distance parameter* was defined as (average shortest distance / √(Number of distance pairs)). Note that (A, B) is a different pair from (B, A) because communication is not bi-directional. As expected, the *normalized average shortest distance parameter* decreased with increasing number of powerful transmitters.

The average shortest path, normalized average shortest distance parameter, number of direct transmitter-receiver pairs (i.e. number of pairs (A, B) such that A can transmit to B directly), number of finite shortest distance pairs (i.e. number of pairs (A, B) such that there is finite shortest distance from A to B), and number of groups were plotted. As expected, when the number of powerful transmitters was increased, the number of groups decreased and the number of direct transmitter-receiver distance pairs and finite shortest distance pairs increased.

The node connectivity diagrams for each of the five cases and graphs for system metrics versus number of powerful transmitters, follow. In the node connectivity diagrams, the arrow indicates the direction of communication with the head at the receiver.

4. Future Work:

The variation in system metrics has to be observed for networks with a very large number of nodes. An ns-2 implementation has to be done to evaluate the impact of the small world phenomenon on routing performance.

5. References:

[1] R. Albert, A. Barabasi, “Statistical mechanics of complex networks”, Reviews of Modern Physics, 74, 47, 2002.

[2] F. Harary, R.Z. Norman, D. Cartwright, “Structural Models : An Introduction to The Theory Of Directed Graphs”, Wiley, 1965.

A.1 Reliable Multicast Code Printout:

All files are in alphabetical order and include page number to aid in identification of start and end of files. Lines are numbered for easy matching with the description of changes presented in section I.4.

A.2 Small World Code Printout: