INFORMATION TECHNOLOGY JOURNAL

Information Technology Journal 6 (7): 941-953, 2007
ISSN 1812-5638
© 2007 Asian Network for Scientific Information

Using Emulation Instead of Simulation to Analyze Synchronized Higher Layers Performance in Wireless Networks

Fei Hu, Amish Rughoonundon and Ramesh Krishnaram
Department of Computer Engineering, Rochester Institute of Technology, Rochester NY, USA

Abstract: The emulation is different from the simulation because the former uses real hardware to build an environment similar to the physical testbed. We have used RF hardware to create a wireless networking emulation platform that is easy to use and modular enough to allow users to quickly adapt the emulator to their own use. Specifically we have tested the communication performance of a routing protocol (Ad-Hoc On-Demand Distance Vector (AODV)) and a transport protocol (Pump Slowly Fetch Quickly (PSFQ)). Present results show that the ability to use a realistic wireless device provides additional constraints that need to be taken into consideration when designing a communication protocol.

Key words: MANET, AODV, PSFQ, Skipjack, Emulator, cryptography

WHY EMULATION

Experimentation in wireless protocols and technologies is at the core of implementations of any new wireless network solution. The most common kind of experimentation is the simulation of wireless protocol design using network simulation tools such as OPNET, ns-2 and OMNET++. Although these software-based simulations could provide a good background on the possible advantages of wireless protocols, they do not cover all the problems that may be encountered in the real world.

A tested is defined as a platform on which a range of experimental products can be deployed and allowed to interact in real-time. Testbeds provide the best possible solution to real time testing of communication protocols but they do carry a heavy price. Testbeds that have been used in the past have shown that they are difficult to set up and some of the experiments could not be reproduced faithfully. Testbeds have also been expensive in terms of the monetary cost of setting them up. As stated by Maltz *et al*. (1999). Most of the testbeds currently in use cost thousands of dollars and are so complex that they require a lot of man power to perform one experiment.

Emulators, although not the best solution, do provide advantages as compared to testbeds. They cost less to setup and use, they take much smaller space than testbeds. And the manpower needed to operate an emulator is much less than a tested. They could therefore be easily moved to other locations to check the effect of a real environment on the wireless communications. There have been a number of emulators created over the years for wireless network testing, such as JEMU (Flynn *et al*., 2002) which is a radio replacing emulator and MobiEmu (Zhang and Li., 2002) which uses packet filtering to emulate mobility. Although all the emulators are different in their architecture, they all have the following things in common: All of them use Linux as their base Operating System to run the emulation software on; They all use WLAN 802.11 as their MAC and physical layer; TCP/IP is the transport and network protocol of choice when designing the emulator; They all use IP chains to virtually kill the link between different nodes. IP Chains have been a set of rules in Linux that allows the operating system to filter IP packets based on the content of the packet such as MAC address, source IP address or destination IP address. The issue with this kind of architecture is that 802.11 implements a high data rate wireless protocol which may not exist for all Ad-Hoc networks and the emulator software is restricted to being used under the Linux operating system.

In this research, an emulator has been designed that will be highly reconfigurable and provide a user with the ability to modify the emulator to their specific needs. The emulator is created to run on the Windows operating system to differentiate it from the common trait of past emulators. Since IP chains are not available in Windows, a new type of packet filter has been created to virtually kill the wireless links between the hardware devices. The emulator is not created to be used with a single hardware device. This will provide the user with the ability to test a protocol with multiple hardware devices.

Corresponding Author: Fei Hu, Department of Computer Engineering, Rochester Institute of Technology, Rochester NY, USA

Since this work will deal with low data rate wireless networks, two low data rate radio devices are considered to be used in this research: (1) TelosB motes from XBow Inc. and (2) XBee-Pro RS-232 RF modem from MaxStream Inc. TelosB motes are highly reconfigurable devices that consist of processor and low data rate radio boards. The board could also be customized with different sensory devices. A more in depth specification of TelosB are as follows: IEEE 802.15.4 compliant; 250 kbps, High Data Rate Radio; TI MSP430 microcontroller with 10kB RAM; Integrated onboard antenna; Data collection and programming via USB; Open-source operating system. In Fig. 1, we can see that a TelosB mote needs to talk with a gateway board in order to report the sensor data to the laptop.

Xbee-Pro (Fig. 2) is an off-the-shelf RF modem. Much of the modem's settings are easily reconfigurable for different power levels as well as different channel frequencies by using user-friendly software that comes with the device. A more in depth specification of XBee-Pro can be found below: Zigbee/802.15.4 compliant RF modem using ISM 2.4 GHz; RF maximum data rate of 250, 000 bps; Uses Direct Sequence Spread Spectrum for modulation; RS-232 interface; Interface maximum data rate of 115, 200 bps; Range of a 100 m indoor.

Both radio devices have more or less the same hardware settings. Having used the Xbow Inc. TelosB motes in the past, we knew that it is tricky for a user to reconfigure the programs running on the motes. Any little change to the hardware needed such as having the radio run at different power levels or changing the channel frequency will have required a very good understanding of the underlying code and considerable testing. The XBee-Pro provided a more easily reconfigurable device.

As shown in Fig. 2, a RF modem needs to connect to a laptop to achieve the following purposes:

- To emulate a mobile node: we could move the laptop freely. The RF modem has a wireless communication range of 10~100 meters (depending on the setup of RF sending power level);
- The RF modem can only provide lower layers (Physical layer and Data Link layer) functions. The laptop could provide higher layers (Routing layer and Transport layer) emulation analysis.

Here we can see the benefit of using emulation compared to simulation and tested:

- The simulation software cannot reflect the lower layer radio propagation characteristics. It cannot accurately simulate wireless transmission error statistical distributions.

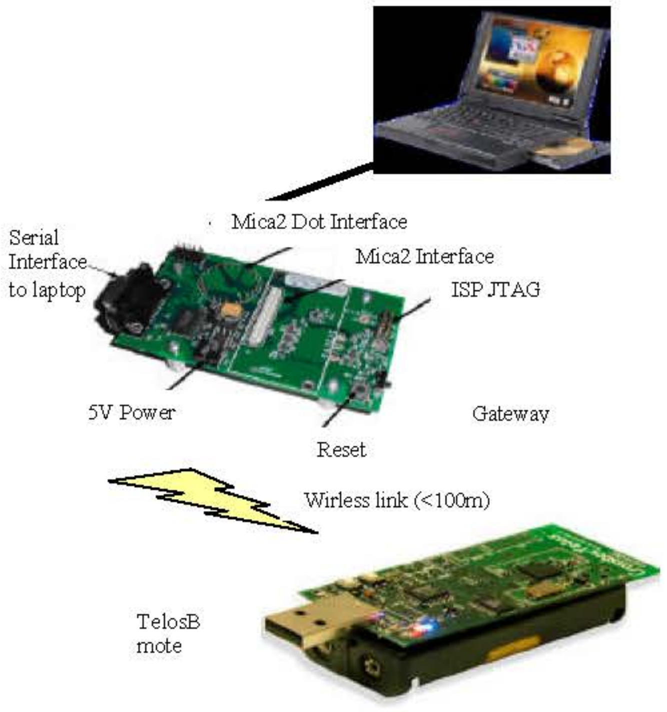


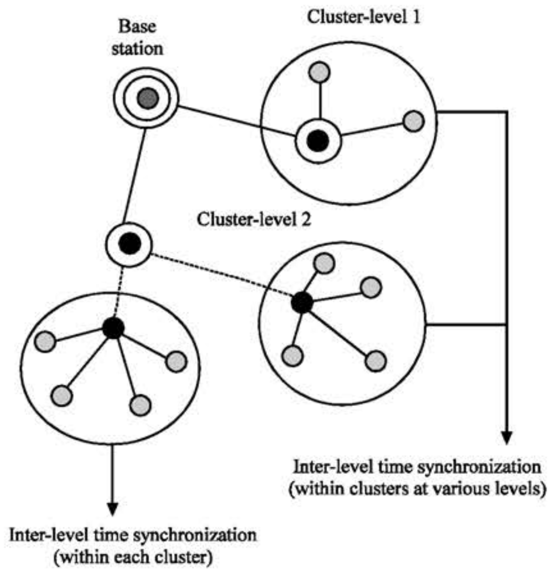
Fig. 1: Use TelosB motes

Fig. 2: XBee-Pro RS-232 RF modem

- The simulation cannot give us a comprehensive Data Link layer function. Our RF Modem has a built-in IEEE 802.15.4 medium access control module that can avoid the wireless channel conflict and hidden terminal problem. It is very time-consuming and also difficult to use pure software to implement all IEEE 802.15.4 details.
- The tested needs to use an independent, commercial wireless device that has all wireless stack functions (from application layer to physical layer). Those devices are expensive for academic research. And most commercial products are for specific applications (such as multi-point conferences) and do not allow a user to change most of their software modules. This is not convenient for academic research.

In this research, we have used XBee-Pro RF modems to build a cluster-based wireless network topology (Fig. 3). It consists of over 20 nodes. Those nodes are organized into several clusters with one node as the cluster head. All nodes only send data to the cluster heads. And the base-station only communicates with all cluster heads.

We have added clock synchronization software in the laptops to achieve accurate time synchronization among all RF nodes. Our synchronization software is based on RBS scheme (Elson, 2002). RBS protocol exploits the broadcast property of the wireless communication medium. In this protocol two receivers located within listening distance of the same sender will receive the same message at approximately the same time. Also if each receiver records the local time as soon as the message arrives then they can synchronize with a high degree of precision by comparing the local clocks on when each message was received (Fig. 4). The offset and skew is calculated based on a sequence of synchronization messages and this protocol exploits the concept of time-critical path (Sundararaman, 2005), that is, the path of a message that contributes to non-deterministic errors in a protocol.


Base station
Cluster-level 1
Cluster-level 2
Inter-level time synchronization (within clusters at various levels)
Inter-level time synchronization (within each cluster)
Fig. 3: Cluster based network architecture

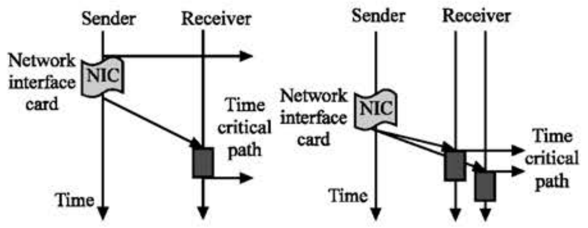


Fig. 4: Time fritical path (left: Traditional 1-to-1 scheme; right: RBS scheme)

The accuracy of a synchronization protocol is greatly affected by nondeterministic transmission delays because it is difficult for a receiver to estimate time at which a message was sent and vice versa. The four factors that are involved in sending a message i.e., send time, Access time, Propagation time, Receive time, are all factors that can vary non-deterministically.

However RBS considers only the times at which a message reaches different receivers and hence it directly removes two of the largest sources of non-determinism involved in message transmission, namely the send time and the access time. The simplest form of RBS can be explained in three steps. A transmitter broadcasts a reference packet to two receivers. The receivers record the time at which the message was received according to their local clocks and exchange their observations.

RBS has following advantages: The largest sources of error (send time and access time) are removed from the critical path by considering only the times at which a message reaches different receivers. Estimation of clock offset and skew are independent of each other. Since local clocks are never modified, clock correction does not interfere with either estimation. Post-facto synchronization prevents energy from being wasted on expensive clock updates and by using nodes belonging to multiple neighborhoods RBS provides supports for Multi-hop networks. However, it also has some shortcomings. For instance, for a single-hop network of *n* nodes, this protocol requires O (n2) message exchanges, which can be computationally expensive in the case of large neighborhoods. Since a larger number of message exchanges, the convergence time, which is the time required synchronizing the network, can be high. The reference sender is left unsynchronized in this method and if the reference sender needs to be synchronized, it will lead to a significant waste of energy.

WIRELESS NODE EMULATOR

The goal of our design is to create a wireless network emulator that could work with any wireless portable device at the physical level. As long as the communication interface between the device and the emulator is kept the same, our emulator could be able to function with any current wireless devices. The emulator could run under the windows operating system, which differentiates it from other emulators created before. Since the design of an emulator should be as modular as possible, our emulator is created as a Dynamic Link Library (DLL). In this way if any part is changed, a whole re-write of the emulator is not needed.

The breakdown of our emulator modules is showed in Fig. 5. The filter modules as well as the serial communication modules are designed to be DLLs. The emulator itself is another DLL. Currently the RS232 communication protocol is used to transfer data from the emulator to the wireless devices. In the future USB or I2C protocols could be used instead in order to transfer the information much faster. The filters shown in Fig. 5 are special software modules that are designed to perform various tasks on the packets being passed through them. For example, the distance filter could compute the distance between two nodes using information in the packet and decide if the packet should be silently dropped or passed onto the next filter. An additional filter called errorInsertion is also created that allows a user to inject a certain percentage of incoming packets with errors. If a real wireless environment is tested, error injection will not have been necessary but it provides a means to test the protocols under varying errors in the incoming packets.

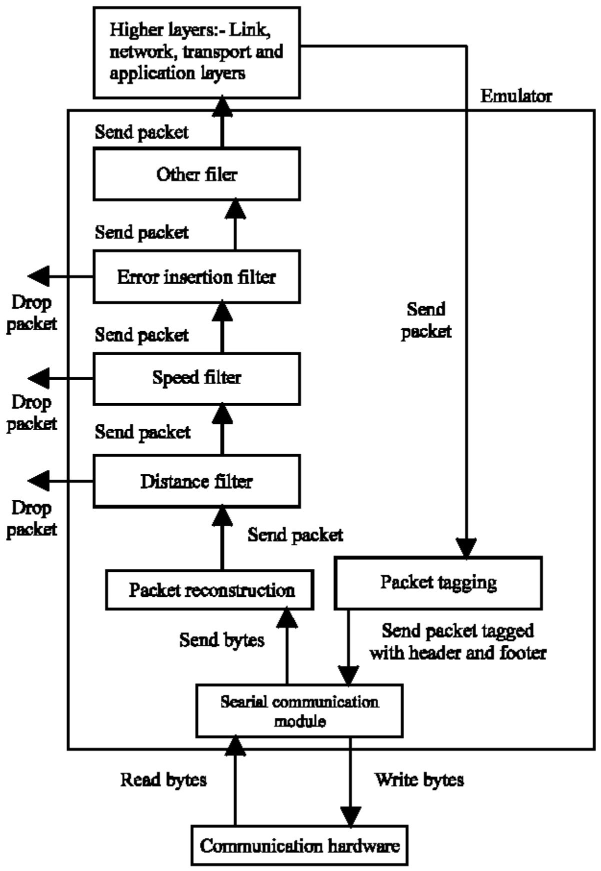


Fig. 5: MoNoE modules interconnection

More filters can be created and added without the need to modify the emulator. This could be done by using a text file that will tell the emulator which DLL needs to be loaded at runtime. The distance filter used a file called distanceFilter.data similar to the ns2 simulation file to determine the position of each node in the virtual landscape. The content of one such file is shown in Fig. 6.

In Fig. 6, the first line is added to allow a human user to understand the content of the file. The filter will first bypass the first line and then use the second line to know how many nodes will be in use during the current emulation. The filter will use the third line to obtain the total simulation time. Note that all the times are set in seconds.

The next 4 lines contain the information for each node in the emulation. The line for each node contains information on the node identification number followed by the start position of the node in x and y coordinates. Alas only positive x and y coordinates could be used as the start position. The first number after the equal sign is the

```
NodeID,x start position, y start position=range,start
time,end time,direction,speed=range,start time,end
time,direction,speed,...
Number of nodes = 4
Simulation time = 60
1,0,0=5,0,60,0,0
2,10,0=5,0,5,270,1=5,6,10,90,1=5,11,15,270,1=5,16,20,90,
1=5,21,30,270,1=5,31,40,90,1=5,41,50,270,1=5,51,60,90,1
3,0,10=5,0,5,180,1=5,6,10,0,1=5,11,15,180,1=5,16,20,0,1=5,
21,30,180,1=5,31,40,0,1=5,41,50,180,1=5,51,60,0,1
4,0,-
10=5,0,5,0,1=5,6,10,180,1=5,11,15,0,1=5,16,20,180,1=5,
21,30,0,1=5,31,40,180,1=5,41,50,0,1=5,51,60,180,1
```

Fig. 6: DistanceFilter simulation file

virtual transmission range of the node. This has to be greater or equal to 0. The number could be used to emulate real wireless communication where the transmission range of 1 node may be greater than the transmission range of another node.

The transmission range is followed by the duration for which the range is valid. The duration consists of a start and stop time. The start time should always be 1 second more than the previous stop time except for the first one in which case it is 0. The corresponding stop time should at least be 1 second more than the current start time. The next two numbers are the direction in degrees ranging from 0-360° and speed of movement.

In this case there are 4 nodes running. The distance filter will use this information to calculate the position of each node for each second of the simulation. The calculation is done before the simulation is started so that the filter does not take too long to determine if the packet is valid or not during runtime. The information is then kept in an array for use during runtime. The downside of such an algorithm is that the memory usage will increase proportionately to the simulation time. It should not cause any problems if the program is run on personal computers but if the program is ported to smaller devices such as PDAs, it may run out of memory very quickly. In this case it may be better suited to have the algorithm compute the data during runtime to save memory.

Since the emulator is supposed to recognize any kind of data stream used by the physical device, it is necessary to add some overhead to create a packet that will be recognized on the receiving end. The overhead consists of header and footer bytes. The header bytes consist of the byte 0xFE followed by the Identity of the sender twice followed by 0xFE again. The footer bytes consist of the byte 0xEF followed by the Identity of the sender twice followed by 0xEF again. This is shown in Fig. 7. The size of ID is a byte. This means that at this time at most 251 nodes could be used altogether. ID 0x00, 0xFF, 0xFE and 0xEF are reserved with 0xFF used for broadcast.

0xFE ID ID 0xFE	**Packet sent from application layer**	**0xEF ID ID 0xEF**

Fig. 7: Emulator packet content

This sequence of characters is to ensure that the packet re-constructor on the receiver side knows the boundary of the packet. Although this adds 8 bytes to the total packet length, it is necessary so that the receiver could create the packet again from the byte stream of the physical device. There is no error correction at this level. If either the header or footer is changed during transport, the packet will not be recognized. Error correction will be a nice addition if ever the emulator is upgraded in the future. This issue may cause the re-constructor to drop the next packet as well while it tries to locate the start of a new packet. This shortcoming is unavoidable due to the fact that the re-constructor has to look at each byte coming in one at a time.

The sender function is used by the application layer to send packets to different part of the network. On the receiving end, the serial buffer is continuously monitored by a thread for new data. As the new data arrived one by one, the data is fed to a re-constructor function that places it in the appropriate location in a packet object. If a packet is reconstructed correctly, it is then sent to the filters.

To prevent any bottleneck from occurring and the possibility that the serial buffer may fill up faster than the re-constructor can create packets, intermediate circular buffers have been added with independent threads on each side of the buffer. The thread on one side will be responsible for grabbing data from the serial buffer and sending the data to the re-constructor thread on the other side. The re-constructor will create a packet and put it in another buffer. On the other side of the buffer, another thread will take care of sending the packets to the filters.

Since the receiving thread is always monitoring the serial port, the sender and receiver thread have to be synchronized appropriately. In this emulator, the sender thread has priority over the receiver thread. This means that if the application needs to send any data, the emulator will stop the receiving thread, send out the data and then resume the receiving thread.

Since the emulator is created to be transparent to the higher layers, it could not be set to know in advance the type of packets that the higher layers will be sending. Therefore collecting statistics about routing or transport layer packets could not be done at the emulator level. Data collection will therefore have to be done at the user level. The programmer will need to insert code to collect specific data regarding his/her protocol. The same thing will apply to the filters used in the emulator. Each filter used can have coded instructions to collect the data that it will be filtering.

WIRELESS PROTOCOL EMULATION

Routing layer emulation: Ad-Hoc On-Demand Distance Vector routing has had many different flavors in the industry. There has been Kernel AODV provided by the National Institute of Standard and Technology, AODV-UU by the Uppsala University and UoB JAdhoc by the University of Bremen just to name a few. Some are created to be exclusively used on the Linux platform whereas others created in JAVA could be used anywhere the JAVA virtual machine is running. A lot of AODV implementation assumed that packet loss will be minimal as viewed from the routing protocol since the Link layer will take care of packet consistency and retransmission. In this case though, the Link layer used does not provide any of those services. Therefore the AODV protocol that is implemented does not assume anything about the lower layers. Another point is that the AODV protocol will need to work with the emulator created. The emulator has certain restrictions due to the size of the node ID and the total size of the network. Therefore a new AODV protocol is implemented called AODV_RIT that does not assume that any underlying protocol is performing any packet consistency check. The original specification by Perkins *et al*. (2003) is used to create the protocol. The protocol specification does not take into account some more intermittent extreme conditions that can sometimes occur in some wireless networks. It is therefore modified somewhat to be better suited to the conditions that it will encounter in this case. These modifications have been outlined below.

The original AODV protocol does not work very well when used in a very lossy environment with wireless nodes that broadcast every message to all other nodes in the network. Some changes need to be made:

- AODV_RIT does not keep track of the active neighbors using a route. In the original protocol, if a route is invalidated, only the active neighbors are notified of the change. During testing, it is found that unicasting messages to all the active neighbors do not provide the best solution. Due to the lossy environment, a lot of messages get lost and many neighbors still sent messages to the node even after an inactive route notification is sent. On top of that, since the hardware nodes always broadcasted their data, even unicasting really means broadcasting the data to all the nodes. Since the same message is sent multiple times to each active neighbor, the overhead is greatly increased and the bandwidth taken by all the messages is more than is really required. The solution is to broadcast the message once and let the receiving nodes take care of deciding whether the message should be used or not.
- Our second enhancement is to use local connectivity management on all packets received by the node and not only hello messages. This is due to the fact that many messages do not actually reach their destination and it will happen that although a node is still a neighbour, none of its hello messages reaches another neighbour node. Since other messages such as RREQ and RREP are still heard by neighbours, these are also used to perform local connectivity management.

Transport layer emulation: MANET devices are everywhere we can think of. They could be inside home appliances relaying data regarding the state of the device, used as sensing applications in security devices or to monitor hazard areas and even used inside the human body to monitor the health of a patient. One of the problems has been that the code used in the machines may need an upgrade from time to time. It could be due to an error that needed to be fixed or maybe the function of the device needed to be modified. This may pose a problem if the wireless device is found in a hazardous or unreachable environment. It is therefore important to have a system that will allow the devices to be reprogrammed remotely. Reprogramming wireless devices involved many different tasks as described by Wang *et al.* (2006b) that cannot all be discussed here. This work will therefore concentrate on the aspect of code dissemination at the MANET Transport layer.

Existing MANET Transport protocols fall into two categories: Congestion Control Protocol and Reliability Guarantee protocol (Wang *et al*., 2006a). PSFQ falls into the second category. It is a transport protocol used for the propagation of downstream data. It has been advertised as being able to deliver data from a master node to slave nodes across multi-hops with close to a 100% reliability factor. PSFQ does not take a traditional end-to-end approach to error recovery. Since reliability decreased quickly with increasing number of hops, PSFQ is designed to be a hop by hop transport protocol. This means that the data will need to be consistent across 1 hop before PSFQ will allow it to be sent to the next hop.

PSFQ is designed to send regular packets to neighbor nodes slowly (PUMP slowly). This allows neighbour nodes to quickly request any missed packets in between the regular packets (FETCH quickly). The request is done using Negative Acknowledgment (NACK). This means that a node will only request for a packet resend after getting a new packet and if there is a sequence gap between packets. The protocol relied heavily on timers to keep track of when to send the regular packets and how fast to request for packet resend.

Since PSFQ usually broadcasted its data to all neighboring nodes, one important drawback that needed to be considered is the problem of message implosion. To prevent message implosion, the protocol will not send a certain packet if it hears the packet being broadcasted by other nodes a certain number of times. This is also true for NACKs. Each regular packet is sent at time intervals between T_MAX and T_MIN whereas the fetch operation is done a certain number of times before T_MIN. This is to ensure that the missed packets will be obtained before the next regular packet is sent as well as to prevent message implosion by allowing nodes to hear other broadcasted packets. T_MAX and T_MIN can be adjusted depending on the state of the wireless medium.

Another important feature of PSFQ is that only packets with continuous sequence numbers are forwarded to the next node. This is to prevent a missed packet error to propagate to other nodes which could cause unnecessary NACKs transmissions.

A third important function in PSFQ is the report operation. The master node needs a way to know that the end nodes have received the packet properly. PSFQ therefore uses a report code to request that the slave nodes tell the master node if they receive all packets or not. To prevent message implosion, the furthest node from the master node is the only one to reply and all intermediate nodes piggyback their answer on that reply.

In original PSFQ protocol, it was stated that any packet with the most significant bit of the TTL set to high, will act as a report request. In our emulation design, instead of using 1 bit in the TTL field, a separate REPORT packet is sent to request a report from the node. This is done since the maximum number of nodes allowed in the emulator is 254. Therefore the TTL field is set to use 8 bits and 1 bit could not be used as a report bit.

This PSFQ implementation does not really guarantee 100% packet delivery since the protocol description does not give any details what happens when the master node receives a report that says that a node has not received all packets. In this case, it is decided to only measure the number of data that actually reach all the nodes. Further implementation may use the report as a mechanism to start sending the missing packets again.

EMULATION PERFORMANCE ANALYSIS

Our tests are carried out at different locations with different radio fading characteristics. The delay statistics is shown below in Fig. 8. The average delay time almost remains the same for various values of the number of messages. However the maximum delay time and standard deviation delay time both show an exponential increase with the number of messages which implies that bit error rate has a influence on the transmission messages through the channel and as the number of messages increases the probability that a bit is correctly transmitted decreases. The average packet error rate statistics (when the number of sent messages is increased) is shown below in Fig. 9.

In order to observe our time synchronization performance, we first achieve synchronization in Application layer and illustrate the relationship between Master times, Slave time and predicted Master time in

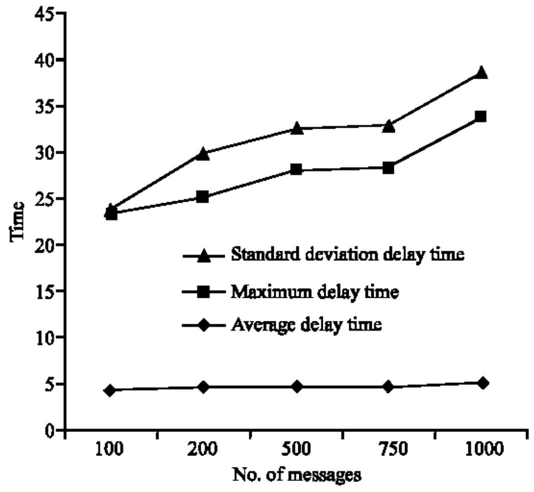


Fig. 8: Delay statistics

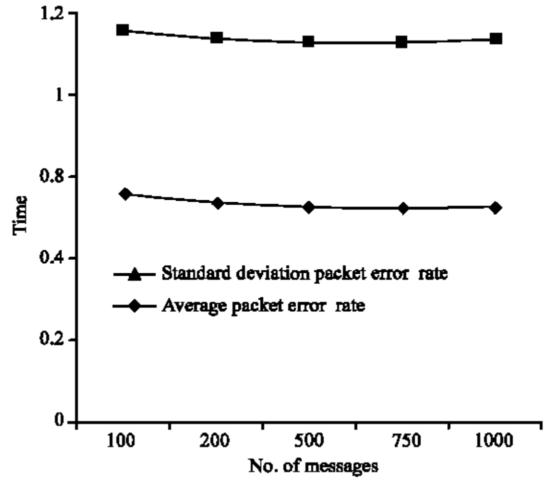
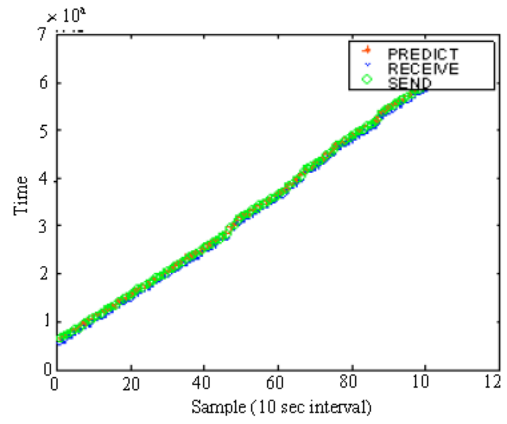


Fig. 9: Packet error statistics



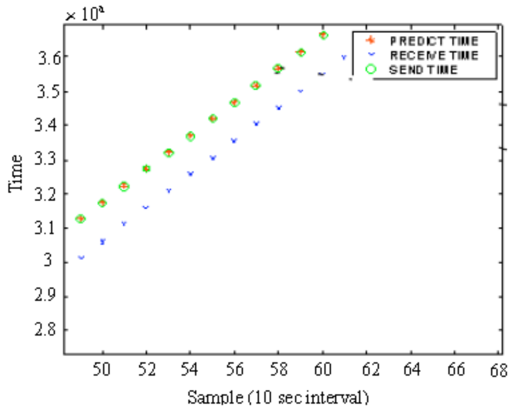


Fig. 10: Predict time versus receive versus send time

Fig. 10. Each figure represents the same data on different time scales. However from the graph it can be clearly seen that the predicted Master time does not exactly match the actual Master time but the prediction is very close.

Besides Application layer synchronization design, we have also created the MAC layer (i.e., Data Link layer) time stamping algorithm using a fine-grained clock, MAC layer time stamping with several jitter-reducing techniques to achieve high precision. This approach eliminates the send, access, interrupt handling, encoding, decoding and receive time errors, but does not compensate for the propagation time. Multiple timestamps with linear regression are used to estimate clock skew and offset. The average error of the algorithm for a single hop case using two nodes was 1.48 μs.

The Fig. 11 below illustrates the error readings for Application layer and MAC layer synchronization approach. Similarly, the offsets from those two synchronization schemes were plotted on the same figure to provide insight to any differences in offset as shown in Fig. 12.

It can be seen from figures above that the error from Physical layer synchronization scheme increased faster than the error from MAC layer scheme. The MAC layer time stamp provided a more accurate time estimate. The

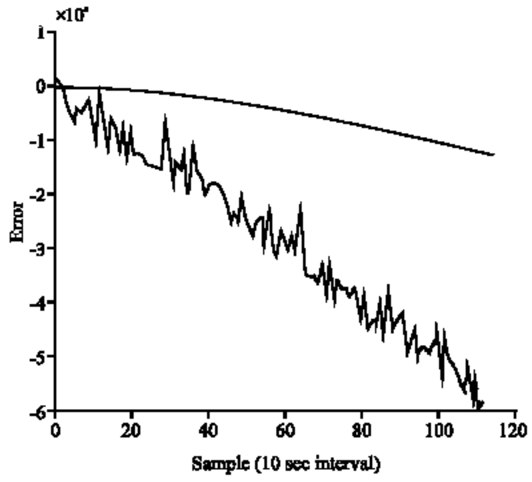


Fig. 11: MAC layer vs application layer timestamp error

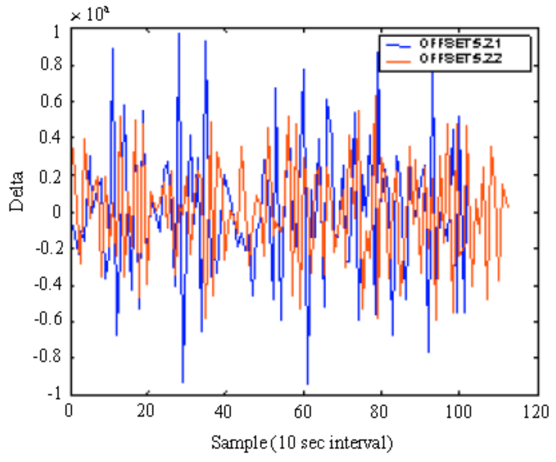


Fig. 12: MAC layer vs application layer timestamp offset

error from Physical layer synchronization also had more jitter when compared to the error from MAC layer scheme. The offset from both these sections were similar. Thus the MAC layer time stamping has a higher degree of precision since it uses multiple timestamps with linear regression to estimate clock skew and offset and a fine-grained clock. It also eliminates the send, access, interrupt handling, encoding, decoding and receive time errors, but does not compensate for the propagation time.

The next experiment used MANET nodes to send 1000 packets of 50 bytes from node ID = 1 to ID = 4. This is done to obtain results regarding end-to-end error rate and throughput of the system.

As shown in Fig. 13, the percentage of overhead packets versus data packets increases with the Packet error rate per link. Since the overhead takes into consideration RREQ, RREP and Hello messages, it is expected that as the number of nodes increases, the number of hello messages will increase proportionately.

The RREQ and RREP mostly contribute to the additional overhead packets at high error rates where a lot of control packets are lost and routes become invalidated frequently. This causes the sending node to request a route over and over again.

As shown in Fig. 14, the end-to-end packet lost increased with an increase in Packet error rate per link although at very low and very high error rates, the number of lost packets congregates to the same amount. This is probably due to most of the packets being lost on the first hop with only a few packets making it to the next hop.

In our next experiment, MANET devices are used with all nodes in a linear fashion. We have focused on 4 of those nodes (Fig. 15): node 1 is within range of node 2 only and node 3 is within range of node 2 only. Node 4 is initially within the range of node 3 only. Node 1 is the sender and node 4 is the receiver. Once a route has been set up, node 4 then starts moving slowly towards node 1 stopping within range of the middle node for an increasing amount of time. This is done to force node 1 to reconstruct the route.

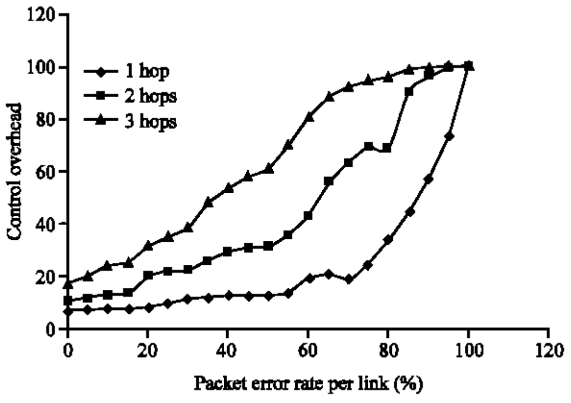


Fig. 13: Overhead percentage versus packet error rate per link

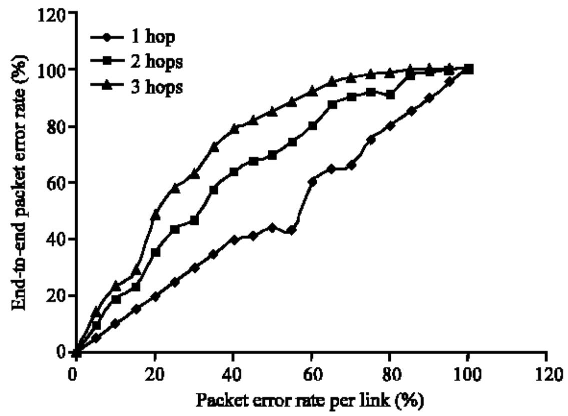


Fig. 14: End-To-End Packet error rate versus packet error rate per link

The emulation file for such a movement is as follows. In this case node 4 will move back to its original position at the end:

```
NodeID,x start position, y start
position=range,start time,end
time,direction,speed=range,start time,end
time,direction,speed,...
    Number of nodes = 4
    Simulation time = 7200
    1,0,0=3,0,7200,0,0
    2,3,0=3,0,7200,0,0
    3,6,0=3,0,7200,0,0
    4,6,3=3,0,60,0,0=3,61,63,270,1=3,64,94,0,0=3,95,
97,270,1=3,98,128,0,0=3,129,131,90,1=3,161,0,
0=3,162,164,90,1=3,165,7200,0,0
```

In Fig. 16, the number of rounds indicates how many times node 4 has moved towards node 1 and back to its original position. For example, for 4 rounds, node 4 has moved towards node 1 then back to its original position 4 times. This particular experiment does not take into account the speed of movement but the amount of time the node is moving. From the data collected it is observed that stopping for a longer time within range of the intermediate node has no effect on the number of packets

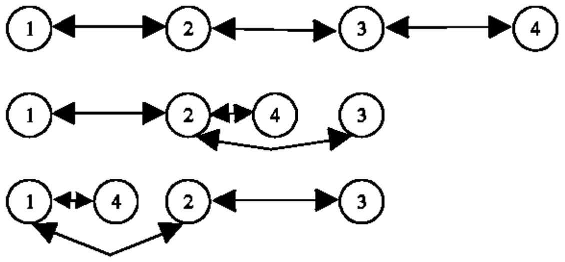


Fig. 15: Destination node 4 moving closer to sender node 1

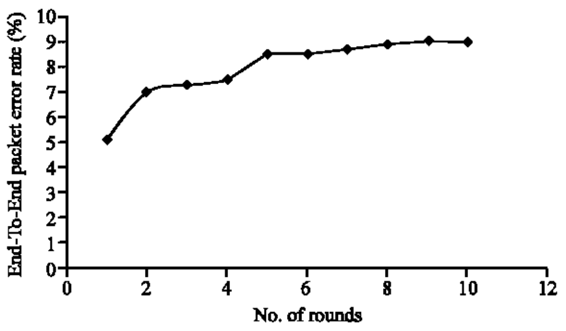


Fig. 16: Packet error rate versus node movement

received. The movement of the node is the major cause of packet lost in this case. This is probably because the nodes have some delay before sending hello messages and there is the possibility that some hello messages are lost. While the nodes are trying to reconfigure the paths, some packets are definitely lost in the meantime.

Decreasing the time between hello messages will certainly improve data loss due to movement. The downside will be that it will increase the interference and packet collisions. Another side effect will be that since only 1 function can access the serial port at any one time, sending a lot of hello messages could actually increase the time it takes to send other packets out.

We then test the performance of PSFQ_RIT protocol in our emulation hardware. For simplicity, among all MANET devices, we have concentrated on the traffic characteristics of 4 nodes. Each experiment is stopped after the master sent the last packet to the 4th node, requested a report and obtained a report reply message. The amount of packets received is then tabulated. The tests are done as follows:

MANET packet size is 50 bytes. Three tests are done with increasing T_MIN, T_MAX and constant T_NACK values. The number of end-to-end packets lost with respect to number of hops and packet error rate per link is measured. The results are shown in Fig. 17-19.

From the above experimental results, we can see that PSFQ's performance is much better than that of sending packets without any hop-to-hop error recovery (up to a certain error rate of about 65%). After that there is a significant lost of data. Since the Timing used in PSFQ is critical to the proper use of the protocol, different intervals are used to send packets out. As T_MIN is increased and T_NACK stays the same, more NACKs are sent before the next valid packet (thus allowing the next node more chances to recover lost packets). The number of NACKs is approximately equal to T_MIN/T_NACK. The downside is that with more overhead data, the chance of packet collision and interference increases.

It can be observed from Fig. 19 that the most improvement is seen for the node 3 hops away from the sender. The end-to-end packet error rate actually decreased from 15% to about 1% at a packet error rate per link of 60%. This shows that this protocol is highly dependent on the timing of its packet departure time and recovery time. It may be beneficial to modify these times for different kind of wireless medium to obtain an optimal solution.

During the above experiments, it is observed that NACKs are not being received once the NACK packets become too large. This is shown in figures above with high packet error rates per link above 65%.

To remedied this issue, in the next set of experiments, the original PSFQ protocol is enhanced so that NACKs will not contain all the gaps in the current file. Instead, multiple different NACKs will be sent with

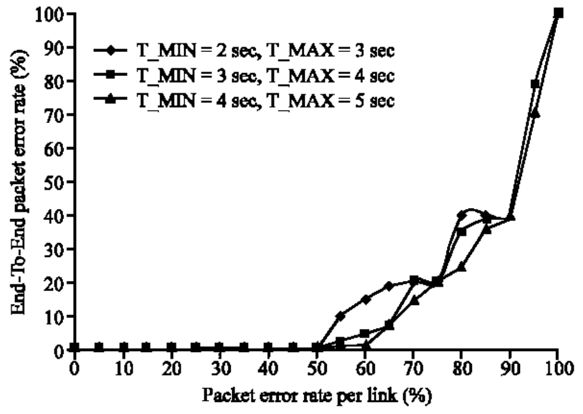


Fig. 17: End-to-end packet error rate vs packet error rate per link for node 1 hop away

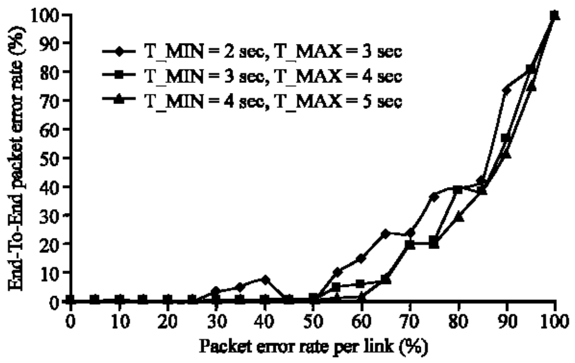


Fig. 18: End-to-end packet error rate vs packet error rate per link for node 2 hops away

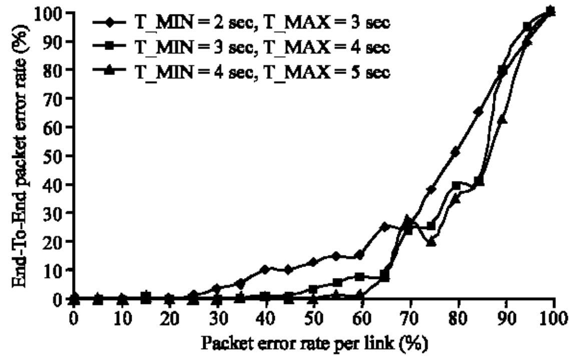


Fig. 19: End-to-end packet error rate vs packet error rate per link for node 3 hops away

different segment gaps. This is to ensure that the NACK packets do not become too big. Packets with too many bytes may overload the hardware buffer of the receiving module and cause packets to be dropped unnecessarily.

The optimizations outlined above regarding the size of the NACKs are then added to the emulation program and another set of experiment is run with the PSFQ properties as follows: T_MIN = 1; T_MAX = 2; T_NACK = 1.

As seen in Fig. 20, it shows an improvement from 25% end-to-end error rate to close to 1% for a packet error rate per link of 65%. This shows that even simple modifications can have huge impact on the overall performance of the protocol when the limitation of the hardware is taken into account.

These types of issues are only noticeable when real emulation hardware is used. A programmer or tester could have overlooked such an issue if only using a simulation. The most probable case is that the programmer may think that there is a bug in the software that is overlooked or that the wireless interference is just too great. With our emulator, a programmer will immediately notice that the issue is not the program since he/she can test the program as they are writing it using the hardware. In this case using an emulator on top of simulation will be more cost effective in the long run.

This set of experiments are done to compare the use of multiple intermediate nodes to help recovery. For simplicity, again we have concentrated on four devices among all MANET nodes (Fig. 21), with two nodes in between node 1 and 4. The two middle nodes will be set not to communicating with each other. The timing of the protocol is as follows: T_MIN = 1; T_MAX = 2; T_NACK = 1.

In next experiment, node 2 and 3 are set to communicate with each other (Fig. 23). The timing of the protocol is as follows: T_MIN = 4; T_MAX = 5; T_NACK = 1.

As we can see, both graphs (Fig. 22 and 24) from experiment 3) and 4) start at 70% since any packet error rate per link less than that showed a 0% packet loss. Experiment 3 and 4 showed that additional intermediate nodes do in fact help in packet recovery albeit very little. The interconnection between node 2 and 3 helps both nodes to retrieve missing packets better. There are a couple of outliers in both graphs that is due to the hardware used and random nature of the wireless medium.

The results from all the experiments are then compared side by side. This is shown in Fig. 25 and 26. It is found that PSFQ worked well with small data under good wireless conditions. Once the packet error rate goes beyond 80 percent per link, PSFQ has a tendency to perform poorly.

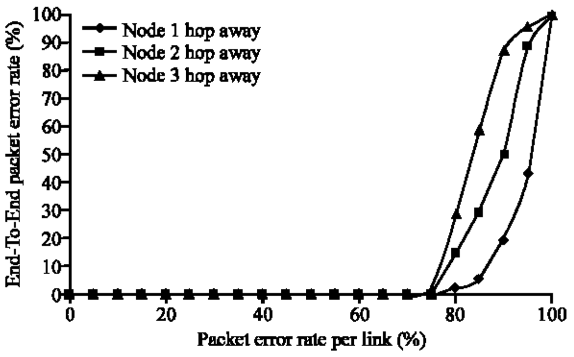


Fig. 20: Packet error rate vS medium error rate with T_MIN=1 sec, T_MAX=2 sec, T_NACK=1 sec

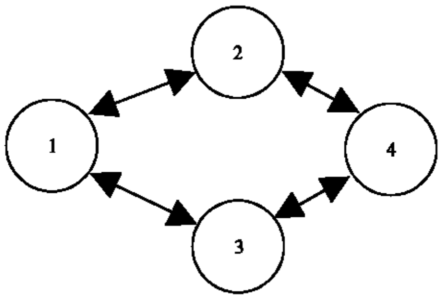


Fig. 21: 4 nodes in diamond formation

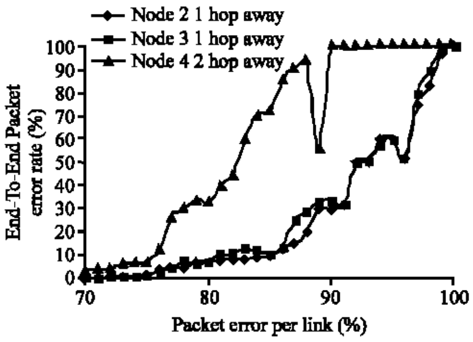


Fig. 22: 4 nodes in diamond formation

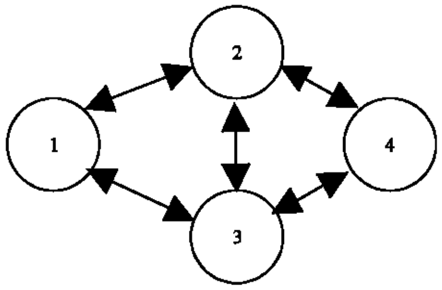


Fig. 23: 4 nodes in diamond formation

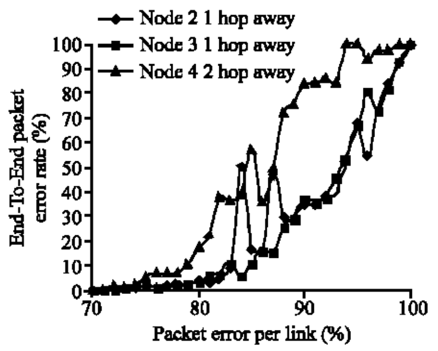


Fig. 24: 4 nodes in diamond formation

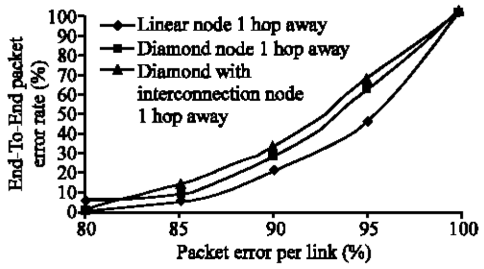


Fig. 25: Comparison of packet error rate with different formations

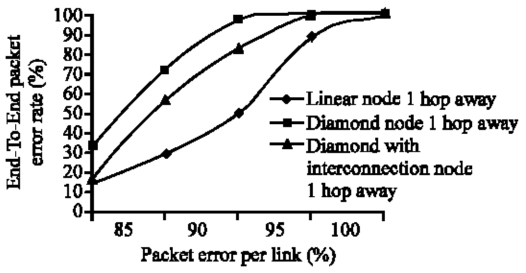


Fig. 26: Comparison of packet error rate with different formations

One critical observation is that whenever a packet is never recovered by a middle node, that node will never send any additional packets obtained to the next node. The next node will have to wait until the proactive fetch kicked in before starting to send NACKs and obtaining the data. This does not cause issues for small files but once the files become too large, the proactive fetch mechanism will take a long time before starting up. In the meantime, the nodes will be left to wait and do nothing.

As the number of nodes inter-communicating increases, so does the end-to-end packet error rate. This may be caused by an increase in overhead packets which prevent valid data packets from being sent out or an increase in interference due to other surrounding nodes or even buffer overflow occurring more often causing good packets to be dropped. This may be the major cause of the diamond formation behaving worst than the linear formation.

CONCLUSIONS

In this research, a wireless network emulator was created that is easy to use and modular enough so that it is fully expandable by allowing users to swap different part of the emulator without the need to re-compile the original software. The AODV routing protocol and PSFQ transport protocol were tested on the emulator. After experimentation and analysis of the results, an improvement to the PSFQ protocol was inferred and implemented which made the protocol more reliable particularly when used with the XBee-PRO wireless modem. This showed that it could be more advantageous for an implementer or tester to use an emulator to test a wireless network protocol on real hardware than just using a simulator.

ACKNOWLEDGMENTS

The authors acknowledge U.S. National Science Foundation (NSF) (Award No. 0511098) for supporting the development of the Pervasive Computing Laboratory at Rochester Institute of Technology. We also thank Cisco Inc. University Research Program (URP) for supporting the development of MANET security projects.

REFERENCES

Elson, J., L. Girod and D. Estrin, 2002. Fine-Grained Network Time Synchronization using Reference Broadcasts. Proc. 5th Symposium on Operating Systems Design and Implementation (OSDI 2002), 36: 147-163.

Flynn, J., H. Tewari and D. O'Mahony, 2002. A Real-Time Emulation System for ad hoc Networks in Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS, 2002), San Antonio, Texas, January 27-31, pp: 115-120.

Maltz, D., J. Broch and D. Johnson, 1999. Experiences designing and building a multi-hop wireless ad hoc network tested. CMU School of Computer Science Technical Report CMU-CS-99-116, March 5 1999, http://citeseer.ist.psu.edu/maltz99experience.html, Accessed January 19, 2006.

Perkins, C.E., E.M. Belding-Royer and I. Chakeres, 2003. Ad Hoc On Demand Distance Vector (AODV) Routing. IETF Internet draft, draft-perkinsmanet-aodvbis-00.txt, Oct, 2003.

Sundararaman, B., U. Buy and A.D. Kshemkalyani, 2005. Clock synchronization for wireless sensor networks: A survey. Ad hoc Networks, May, 3: 281-323.

Wang, C., M. Daneshmand, B. Li and K. Sohraby, 2006a. A survey of Transport Protocols for Wireless Sendor Networks. Accepted to Appear in IEEE Network Magazine Special Issue on Wireless Sensor Networking, 2006.

Wang, Q., Y. Zhu and L. Cheng, 2006b. Reprogramming wireless sensor networks: Challenges and approaches. Proceedings of IEEE Network, 20: 48-55.

Zhang, Y. and W. Li, 2006. An integrated environment for testing mobile ad hoc networks. In: Proceedings of the 3rd ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc), Lausanne, Switzerland, pp: 104-111.