Implementation of Transport Protocol over Wireless Sensor Nodes

Jigar D Parekh
jdparek2@ncsu.edu
North Carolina State University

Pratik D Jain
pdjain@ncsu.edu
North Carolina State University

Nikhil N More
nnmore@ncsu.edu
North Carolina State University

Subramanian A Aayakkad
saayakk@ncsu.edu
North Carolina State University

Date: April 19, 2015

***Abstract* - An ad hoc network is a group of devices that can communicate directly with each other without needing a top-down or hierarchical communication structure or a centralized access point.**

Wireless sensor network (WSN) is a group of spatially dispersed and dedicated autonomous sensors for tracking, monitoring and controlling and recording the physical conditions of the environment and then organizing and taking an appropriate action on the collected data at a central location. WSNs can measure environmental conditions like temperature, sound, pollution levels, humidity, wind speed and direction, pressure, etc.

The primary purpose of a transport protocol is to provide end-to-end connectivity by delivering data in form of encapsulated packets from the source host to the destination host. Some protocols also implement flow control, traffic control, multiplexing, connection-oriented communication and same order delivery.

In this project we have implemented a simple transport protocol on sensor platform in tinyos-2.1.2 - mica2 motes environment using Stop and Wait, where the sender sends one packets and has to wait for the acknowledgement before sending the next one, as well as implemented Go Back N protocol, where the sender can transmit multiple packets but doesn't have to wait for an acknowledgment, till it reaches a certain maximum allowable number of unacknowledged packets in the pipeline. The transport protocol that is implemented by us provides reliability via cumulative acknowledgement and negative acknowledgement via duplicate acknowledgement. The transport protocol implemented also has features such as congestion avoidance and flow control.

TCP works well for wired networks but possesses some issues like reduced throughput in wireless networks because it considers congestion as sole reason for packet loss, whereas in wireless networks, congestion isn't the only reason for the loss.

Hence, the primary goal is to test the performance of a transport layer protocol in wireless sensor network. We are going to check the performance parameter throughput of the protocol in a multi hop environment.

***Keywords:* NesC, TinyOS, Mica2, MIB600, Split-TCP**

I. INTRODUCTION

Wireless Sensor Networks can work without wiring. We don't need any fixed infrastructure like access points etc. The implementation is cheap and can easily add newer devices to the current network. Ad-hoc networks are very good solutions for places that are not easily accessible such as across the sea, mountains, rural areas or war zones. Accessing the network is easily done using a centralized monitor.

The mica motes that we are using have design specification such that we can maintain a low cost with a minimum power consumption and a considerably small [2].

Wireless networks are prone to losses because of congestion, bursty or random channel error, high bit-error rates, fading, and during handoffs. The most important aspect in wireless communication for sensor networks is that we can't afford to lose data since the motes switch on periodically, collect data, send it to the base station and switch off the sensor. As per [9], if the data sent is less, it is has more information and hence is very precious.

An easy solution to solve this issue such that it will guarantee delivery in sequence is by increasing power of transmission or by using high gain directional antennas. This falls flat because saving power and staying on for a very long time is one of the primary objectives of wireless sensors. eg in war zone areas.

Hence a feasible solution to this issue can be to introduce congestion and flow control using a layer 4 protocol. Also, the aim is to increase the throughput by changing the protocol so that it can have with minimum overhead and send maximum data on a multi hop wireless sensor network.

In traditional wired networks, the main reason for packet losses and delay is considered to be congestion in the network. The strategy that TCP adopts in such cases is: The receiver sends cumulative acknowledgments to the sender. The TCP sender uses them to decide which packets have reached the receiver, and thus ensure reliability by retransmitting the lost packets.

The sender realizes that packets have been lost when it sees duplicate cumulative acknowledgments or no acknowledgment for the packet within a certain timeout interval.

When TCP sees packet losses, it reduces its transmission window size before resending the packets, in our case it makes equal to 1, starts congestion control or avoidance mechanisms like slow start [10], Additive Increase Multiplicative (AIMD) and use back off retransmission timer Karn's Algorithm [11].

Because of this strategy, there is a reduction in the traffic load on the links, thus congestion in the network reduces.

II. Related Work

There have been many research works going on in the field of TCP over wireless. In [12], the authors have stated that networks with wireless and other lossy links suffer from significant non congestion related losses. But TCP responds to all losses by invoking congestion control and avoidance algorithms, resulting in degraded end-to-end performance in wireless. In this paper, authors have compared several schemes designed to improve the performance of TCP in such networks such as end-to-end protocols, link-layer protocols, and split-connection protocols.

In [13], the authors stated the implicit assumption TCP makes that losses indicate Network congestion is not valid. They classified wireless TCP implementation as the one which hides non congestion related losses from the TCP sender and second one which makes the TCP sender adaptable to realize that some losses are not due to congestion, using selective acknowledgments or explicit loss notification.

In [13], the authors have come out with a Markov Chain model to analyze the system and they state that when the number of hops increases from one, the TCP throughput decreases first, and then stabilizes when the number of hops becomes large.

In this paper we show the implementation and throughput comparison of Link Layer ARQ and Split TCP over wireless sensor networks. These are the modifications of TCP to run over wireless. Split-TCP splits the wired and wireless connection into two parts. There are independent Congestion and Flow control mechanisms running on both the connection. We also, analyze the throughput of Multi-hop wireless networks and compare it with that of the single hop implementation.

III. Problem Formulation

A. Components:

1. TinyOS:
An Operating System, designed to run on low-power wireless embedded devices used in wireless sensor networks. It is written in **nesC** that is a variant of C. It is a work scheduler and a collection of drivers for microcontrollers and ICs that are used in wireless embedded platforms.

2. nesC:
Network Embedded Systems C is a component-based, event-driven programming language that is used to build applications for TinyOS. In nesC, components are "wired" to each other to run applications on TinyOS [4].

3. MICA2:
This Mote is a third generation mote module used for enabling low-power, wireless and sensor networks. It uses TinyOS Distributed Software Operating System and supports wireless remote reprogramming. It has more than 1 year battery life for AA batteries [2].

4.MIB600
This board provides Ethernet connectivity to a MICAZ /MICA2 family of motes. The MIB600 can serve as a bridge between wired and wireless segments of a network. The MIB600 helps us to remotely access sensor network data via TCP/IP as in [3].

B. Stop and Wait Protocol:

The sender can send a single data packet and has to wait for an acknowledgement. Receiver window size is 1. Hence if the packet numbered ‘x’ doesn’t arrive before timeout or if the packet arrives out of sequence, then the receiver sends a negative ACK, ie. a duplicate ACK such that the sender now has to send the packet number ‘x’ again.

C. Go Back N Protocol:

The sender can send up to size N number of data packets without having to wait for an acknowledgement, since we are using a wireless channel is in half duplex and sliding window protocol such that when the transmitter is sending the data it does not have to receive the acknowledgements simultaneously. Remember that the receiving window size is only 1. Hence if the packet numbered ‘x’ doesn’t arrive before timeout or if the packet arrives out of sequence, then the receiver sends a negative ACK, ie. a duplicate ACK such that the sender now has to send the entire window size again, this time counting the failed packet as the first one to be sent. Packets received after x are discarded.

Once the entire window is sent, now the sender goes back to the receiver mode and waits for the ACKs. Here the ACKs are cumulative, i.e. ACK number 5 means every packet till number 4 have been received. When a negative ACK appears, then the window size is reduced to half whereas of timeout occurs, the entire window size number of packets are sent again to the receiver.

TCP connection between the source and the destination is called as connection oriented because it forms the connection by implementing the 3 way handshake before the transfer of data begins.

D. Packet Format:

The hardware of Mica2 motes is created such that it does not support very large packet size[2]. Thus, to ensure that the overhead is minimum, the TCP header we have designed is only 10 Bytes.

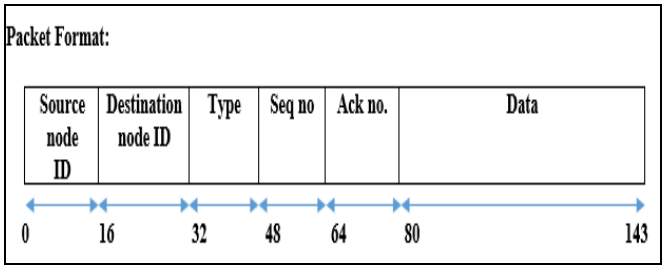

Figure 1. Packet Format

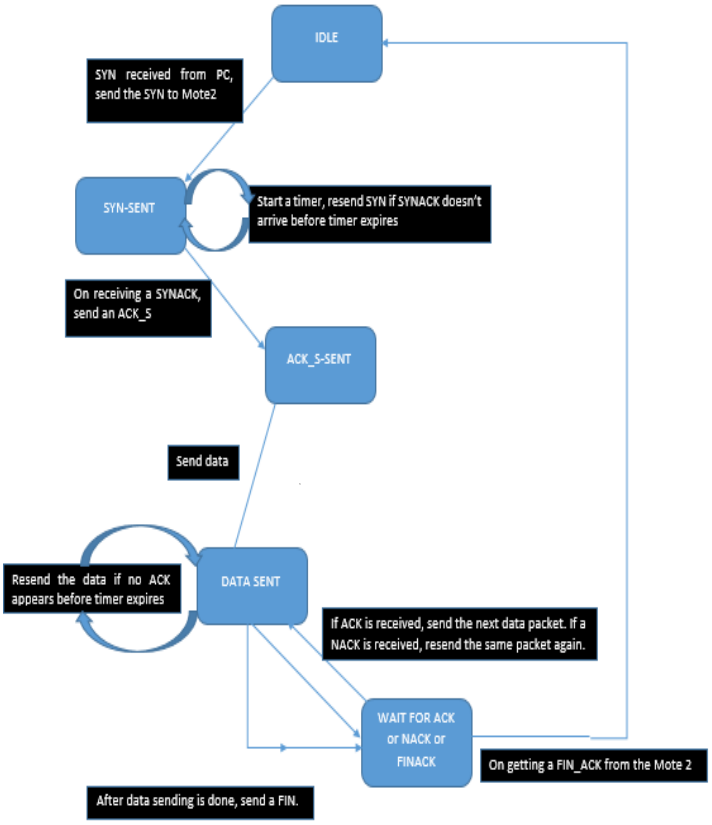

Figure 2. FSM-Base Station (Mote1)

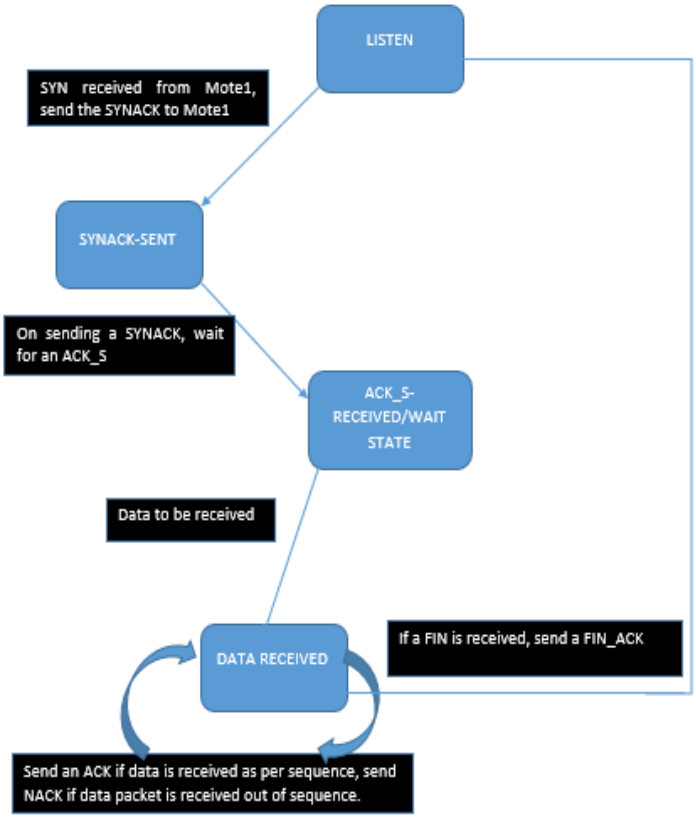

Figure 3. FSM-Destination

IV. IMPLEMENTATION

1. Connection Establishment:

This is initiated at base station (mote 1). The base station which wants to begin the communication sends a SYN to the Destination (mote 2) and starts a timer, thus going from IDLE state to SYN SENT stage.

If the Destination(mote 2)doesn't reply with a SYNACK within the timeout period, may the reason be that the SYN was lost or the SYNACK was lost, the base station resends a SYN.

Initially, destination mote is in the listen state. On getting a SYN from the base station, it sends a SYNACK and starts a timer while waiting for an ACKS (ACK for SYNACK) from the base station. If the base station doesn't reply with a ACK within the timeout period, may the reason be that the SYNACK was lost or the ACKS was lost, it resends a SYNACK. Once a SYNACK is received, the base station sends an ACK to the Destination and state changes to ACK SENT and thus the connection is established.

2. TCP Session:

At Base Station (Mote 1), once the connection is established, it will send the entire window sized number of packets to the Destination (Mote 2), it being 1 at the beginning and a threshold size of 8, and enters the DATA SENT stage. Base station sends the next packet again and enters DATA SENT stage. This happens till the window size is reached after which it enters the "waits for the ACK" state. As an ACK is received, it sends double the size of the window allowed. This Slow Start continues till the threshold value is reached. Then it enter Congestion Avoidance state. Once a packet loss occurs, a NACK (Duplicate ACK) is received and the congestion window size drops to 1 and again the slow start continues. This process will go on till an Active Close arrives from the system. A FIN Packet is sent to the receiver and the Base Station enters the Connection Release state.

At Destination (Mote 2), the window size of the base station receiver is always 1. It is waiting for data in the WAIT state.

Once it receives data, send an ACK if data is received as per sequence or send NACK if data packet is received out of sequence. It stays in the DATA RECEIVED STATE till it gets a FIN. Then it sends a FIN ACK and goes into the Connection Release Stage.

3. Connection Release State:

Termination of the TCP process isn't a three-way handshake like establishment, it is a pair of two-way handshakes.

At Mote 1 after the sending is done, and receiving an ACTIVE-CLOSE form the system, the Mote 1 sends a FIN packet and goes into WAIT FOR ACK or NACK or FINACK, Once it received FINACK, it will change its state to LISTEN state again. If no ack is received in the stipulated time, then mote 1 will resend the FIN packet.

At Mote 2 when it receives FIN packet it will send an FINACK and goes into Connection Release state.

After this, it goes into IDLE state. How will you know that an ACK was delivered successfully? Because if mote_1 won't get it, it will send FIN again after its timer expires, as such the mote_2 will resend a new ACK.

A. Step-by-step Approach:

1. Install TinyOS in Ubuntu machine[1] and learnt coding in nesC [6].
2. Interfacing of MICA motes and installing a basic Blink Program to check the functionality [100].
3. We found the IP address of the MIB600 programming board using Wireshark.
4. To compile and burn the code, we inserted following commands:

   make mica2

   make mica2 reinstall,<moteID>eprb,<IP address of the programming board >
5. For PC to Mote serial communication:
   i. We modified TinyOS provided application TestSerial to send and receive packets.
   ii. To establish connection we have to set an environmental variable MOTECOM using the command:

   export MOTECOM=network@<IP Address of the packet source>:<portNo>

   Then we just run the java application using:

   java TestSerial
6. Initially we implemented basic packet transmission between motes over radio using Mote to Mote Radio Communication.
7. We then gradually implemented the project in sub-problems as follows:
   i. Implement Connection Establishment (3-way handshake)
   ii. Implement sending data packets with Connection establishment and Connection Termination
   iii. Implement Go-Back-N and Congestion Control

V. EXPERIMENTAL SETUP

The bandwidth available while using mica2 Motes is 19.2kbps [reference of data sheet]. In the experimental setup the mica2 Motes are in close proximity and in Line of sight of each other. Thus in the results found, there is no packet loss. In case of packet loss the throughput will be much less than the obtained throughput in our experimental setup.

For evaluating the performance of TCP in wireless network, we analyze the throughput and efficiency of both Link Layer ARQ and Split TCP.

The throughput and efficiency for the following scenarios were determined:

1. Stop and Wait ARQ (Single hop)
2. Stop and Wait ARQ (Multi hop)
3. Go Back-N and Congestion Control

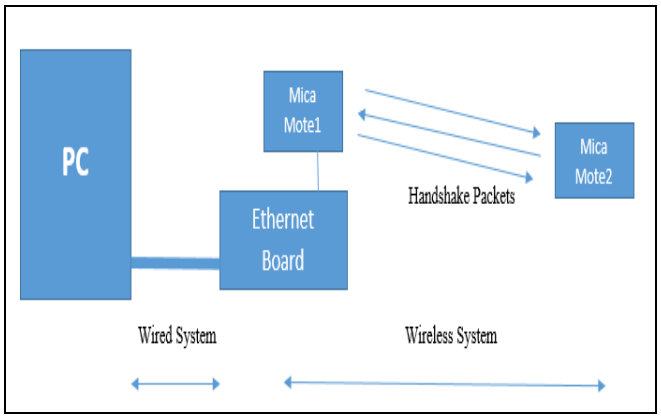

Figure 4. Single-Hop Topology

The throughput calculations were done by adding timestamp at the base station (sender) side. The timer was started when the first data packet is sent by the base station mote and the timer was stopped when the last ACK from the receiver is received by to the sender. This gives us the total time taken for the data to be transferred (neglecting the connection establishment and termination phase). Thus, dividing the total number of bits (header + data) by the total time taken gives us the throughput in bits per second.

$$Throughput = \frac{DataSize(bits)}{Total\ time\ taken}$$

Example: For 50 packets, the total time taken in Go-Back-N with Congestion control, the Data size of single packet is 144 bits and the time taken is 540msec. Thus the

$$Throughput = 50 * \frac{144}{540}\ kbps\ =\ 13.33\ kbps$$

The efficiency is calculated as:

$$\eta\ = \frac{Throughput}{Available\ Bandwidth}$$

While using mica2 Motes, the available bandwidth is 19.2 kbps. Therefore for above example the efficiency is :

$$\eta\ = \frac{13.33}{19.2}\ = 0.6943$$

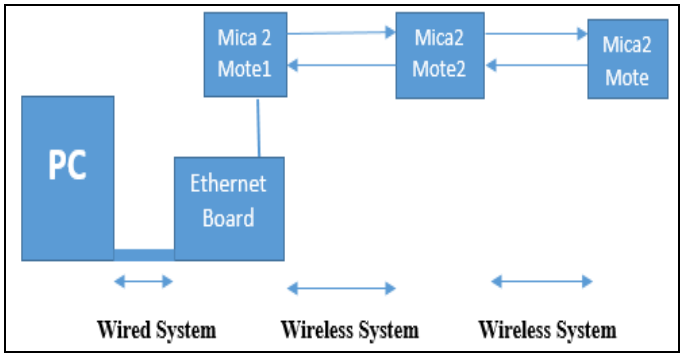

Figure 4. Multi-Hop Topology

VI. Results and Analysis

Number of packets	Time Taken (in msec)	Throughput (in kbps)	Efficiency (%)
10	235	5.45	28
50	1231	5.2	27
70	1764	5.08	26.4
100	2581	4.96	25.83

Table 1. Stop-&-Wait (Single Hop)

Number of packets	Time Taken (in msec)	Throughput (in kbps)	Efficiency (%)
10	135	10.667	55.56
50	540	13.33	69.43
70	722	13.96	72.7
100	976	14.75	76.82

Table 2. Go-Back-N

Number of packets	Time Taken (in msec)	Throughput (in kbps)	Efficiency (%)
10	883	1.456	7.58
50	4377	1.467	7.64
70	6233	1.438	7.49
100	9097	1.41	7.34

Table 3. Stop-&-Wait (Multi Hop)

Number of Hops	Throughput (in kbps)
1	5.2
2	1.467
3	0.78
4	0.443

Table 2. Throughput Comparison-Multi Hop (50 packets)

For Link Layer ARQ (Stop and Wait) scheme, the expected theoretical throughput is 9.2 kbps. This is because, in this scheme, the data packet is sent only when it receives an ACK for that packet. Hence, one data packet is transmitted per Round trip time (RTT). Thus, the link utilization for this scheme is 50% and hence the throughput is half the available bandwidth in case of no packet loss.

For Go-Back-N and Congestion control, the expected theoretical throughput depends upon the change in the window size according to the number of packets. But the efficiency is expected to be more than 80% in case of no packet loss. This is because, number of packets equal to the window size is sent and the cumulative ACK is taken into consideration before sliding the window. This increases the utilization of link more than that utilized by Link Layer ARQ scheme.

But the Throughput determined is less than the expected value (Refer Table 1 and 2). The reason is that while calculating the expected value, the propagation and processing delays are not considered. The transmission takes over in a wireless medium, which can also induce unwanted delays. Thus the determined experimental value of throughput is less than that of the expected. Hence the efficiency decreases than expected**.** For Multi-hop due to the extra hop the packet travels the throughput decreases. The throughput is much less than that of Single hop Stop and Wait (Refer table 1 and 3). The other observation is that as number of hops increases the throughput decreases significantly. (Refer Table 4).

Further, the throughput for Stop-&-Wait remains more or less constant with increase in number of packets, while with Go-Back-N, it increases in the absence of packet loss. Since Stop-N-Wait follows a uniform pattern of sending a packet and waiting for its ACK, no change in throughput should be expected. However, for Go-Back-N with Congestion Control, the window size increases multiplicatively until a threshold post which it enters Congestion Avoidance phase and increases slowly. With this increase in window size, the number of packets that are sent back to back increases, thereby increasing link utilization after every window cycle and hence overall throughput increases. (Refer table 1 and 2).

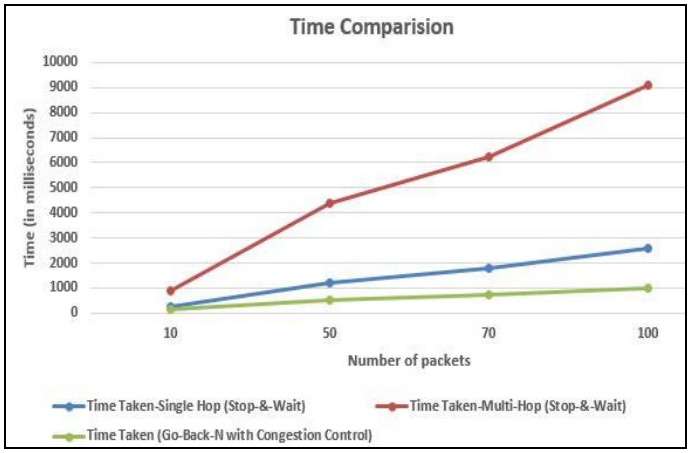

Figure 6. Time-taken Comparison

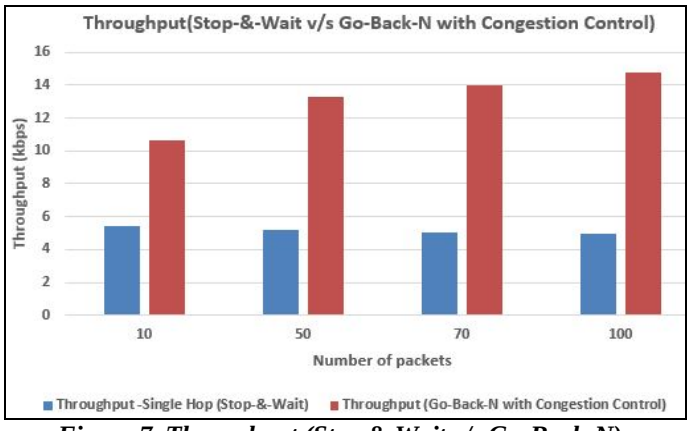

Figure 7. Throughput (Stop & Wait v/s Go-Back-N)

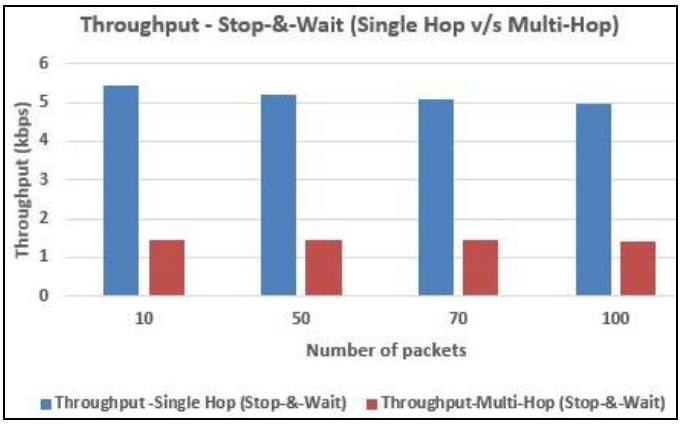

Figure 8.Throughput (Single-Hop v/s Multi-Hop)

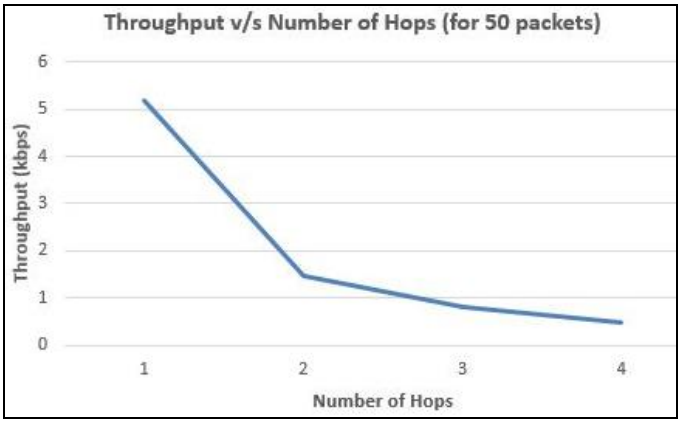

Figure 9. Throughput v/s Number of Hops

We observe that the time taken in multi-hop is much higher when compared to that of single hop. This is expected, because with every additional hop, the packet will be intercepted by the intermediate motes, processed and then transmitted to next hop or the destination accordingly. Thus, there is an added processing and transmission delay. Similar addition of processing and transmission delay during the return path should also be considered. Further, with multi-hop, there is a likelihood of increase in overall distance between source and destination, which increases propagation delay. (Refer figure 6)

Finally, we observe that the time taken by Go-Back-N is much less than that of Stop-&-Wait (Refer Figure 7). This is again, as expected, because with Stop-&-Wait, the next packet is transmitted only after the ACK for current packet is received. On the other hand, with Go-Back-N, packets are continuously transmitted until the window size, without waiting for every ACK like Stop-&-Wait. After one window of packets is sent, the window moves forwards depending upon the last received ACK. As it does not wait to transmit after every packet, the time taken by Go-Back-N is less.

VII. Future Work

The main problem with TCP over wireless networks is that congestion in the network is assumed to be the reason for a packet loss always which most of the times is not true for the wireless media. The path loss, interference due to other wireless media etc. are the main causes for the packet loss. To overcome this to some extent, the concept of having a middle node and marking of packets can be used.

One of the methods is to use an enhanced TCP [14]. In this, the base station keeps a note of the packets incoming from the wired sender node. If this marked packet's acknowledgement is not received by the base station, it sends a special acknowledgement back to the sender stating that this packet loss is due to wireless media losses (as this was transmitted by the base station over the wireless media), so no congestion control mechanism to be run for this packet. Hence, the window size is not dropped and thereby performance is enhanced.

Another method of improving TCP performance in wireless networks is to use a Snoop protocol [15]. In this, the base station (known as the snoop agent) takes care of losses by firstly caching the packet received from the fixed host and resending the packet if there is a packet loss in the wireless media. Thus, there is a local retransmission mechanism in place by not affecting the ends, resulting in improved TCP performance.

Also, there are other means of improving TCP over wireless by having reliable link layer (use ARQ and/or FEC depending on the application)[16] or one can enhance TCP performance by optimizing the number of link-layer retransmissions[17]

VIII. Conclusion

The throughput and efficiency of Transport Control Protocol over wireless network depends on the following parameters:

1. The number of hops between the source and destination

   From the analysis, as the number of hops between the source and destination increases the throughput goes on decreasing. This is because larger the number of hops (motes) in the network, more would be the RTT of a packet as each mote would have its delays.

2. The surrounding radio environment

   Greater the number of devices using the radio interface (around 900Mhz as mica2 works around 900Mhz), large will be the interference, more would be the packet losses and hence less would be the TCP throughput.

3. The flow control technique used

   Greater the number of successively transmitted packets, larger would the pipe be filled and thus more would be the throughput for a lossless environment. The analysis section proves the Go Back-N ARQ would work better than the Stop and Wait ARQ. Also, the Selective Repeat ARQ would work the best and have a much larger throughput and efficiency among all the sliding window techniques for TCP over wireless media as it has lesser retransmissions.

References

[1] https://letstalkgyan.wordpress.com/2013/02/22/install-tinyos-2-1-2-on-ubuntu-12-04-and-blink-leds-on-telosb-mote/

[2] http://www.eol.ucar.edu/isf/facilities/isa/internal/CrossBow/DataSheets/mica2.pdf

[3] http://www.electronicsdatasheets.com/pdf-datasheets/memsic/mib600/

[4] http://tinyos.stanford.edu/tinyos-wiki/index.php/Mote-mote_radio_communication

[5] Computer Networking – A top down approach by Kurose 6th edition

[6] http://www.tinyos.net/tinyos-1.x/doc/tutorial/lesson1.html

[7] [http://tinyos.stanford.edu/tinyos-wiki/index.php/MotePC_serial_communication_and_SerialForwarder_%28TOS_2.1.1_and_later%29

[8] http://tinyos.stanford.edu/tinyos-wiki/index.php/Modules_and_the_TinyOS_Execution_Model

[9] S. Haykin. Digital communications. Wiley, 1988

[10] V. Jacobson. Congestion Avoidance and Control. In Proc. ACM SIGCOMM 88, August 1988.

[11] P. Karn and C. Partridge. Improving Round-Trip Time Estimates in Reliable Transport Protocols. ACM Transactions on Computer Systems, 9(4):364–373, November 1991.

[12] HariBalakrishnan et al.,"A Comparison of Mechanisms for Improving TCP Performance over Wireless Links", Computer Science Division, Department of EECS, University of California at Berkeley,1996

[13] Sonia Fahmy et al.,"TCP over Wireless Links: Mechanisms and Implications", Purdue University, Report Number: 03-004, 2003

[14] Deddy Chandra et al,"TCP PERFORMANCE FOR FUTURE IP-BASED WIRELESS NETWORKS", School of Electrical and Computer System Engineering Royal Melbourne Institute of Technology Melbourne, Australia 3000

[15] H. Balakrishnan, "Improving TCP/IP Performance over Wireless Networks," Proc. ACM MOBICOM '95, Berkeley, CA. ACM, USA, Nov. 1995

[16] [R. Abdelmoumen, M. Malli, and C. Barakat, "Analysis of TCP Latency overWireless Links supporting FEC/ARQ-SR for Error Recovery," Proc. ICC '04, vol. 7, pp. 3994 – 3998, Paris, France, Jun. 2004.

[17] F. Vacirca, A. D. Vendictis, A. Todini, and A. Baiocchi, "On the Effects of ARQ Mechanisms on TCP Performance in Wireless Environments," Proc. IEEE GLOBECOM '03, vol. 2, pp. 671–675, CA, Dec. 2003.

[18] TinyOS Tutorial #1 - How to install TinyOS on Ubuntu, Electronics Engineering Tutorials, https://www.youtube.com/watch?v=AJYjy4bSaHw

Appendix

Below stated are the problems faced by us during the implementation of this project. These problems required a significant amount of time for debugging and for their solutions to be reached.
The following challenges helped us a great way in not only implementing the TCP on the mica motes, but also knowing the minor details associated with it.

Sr. No.	Problem faced	Problem Description	How we overcame the problem
1	Installing TinyOS	'No rule to make target' error was encountered	Had to include a 'MAKERULES' line in ~/.bashrc file.
2	Interfacing of Mica motes	We were not able to find the Ethernet MIB600 boards' IP address to interface the motes.	The Ethernet board and Ethernet ports were in different subnet. So we found the IP address of board using Wireshark and capturing the Gratuitous ARP. Then we set the IP address of our Ethernet port in the same subnet to communicate with the MIB600 Board.
3	Mote-to-Mote Communication	The Motes were not able to communicate over the radio as their frequencies were not set to their default frequency. (MICA2 = 918 MHz)	Edit the CC1000Const.h file in /opt/tinyos-2.1.2/tos/chips/CC1000. Set the frequency as CC1k_918_998_MHZ
4	Mote-to-PC Communication	The Motes were not able to serially communicate with the PC via Ethernet through the MIB600 board.	The MOTECOM environmental variable was configured to communicate serially via Ethernet to the MIB600 board. This was done by writing on the terminal in UBUNTU: ***export MOTECOM=network@<IP address of the packet source> : 10002*** Make the necessary changes in the Makefile by including commands to create a TestSerialMsg class which contains getters and setters to get and set the packet parameters accordingly.
5	Packet transmission	The packet was always being broadcast, we wanted it to unicast considering the multi-hop scenario	Each mote was assigned a unique ID using the following command: ***Make mica2 reinstall,<mote ID> eprb,<IP address of the board>*** Also, corresponding changes were made in the send function of each mote.