Performance of ZigBee PRO Mesh Networks with Moving Nodes

Chester Hamilton
Texas A&M University
Email: chamilton09@tamu.edu

Varun Sampath
University of Pennsylvania
Email: vsampath@seas.upenn.edu

July 6, 2010

Abstract

Radio modules based off of the ZigBee PRO specification provide cheap and low power wireless communication, via usage of the IEEE 802.15.4 PHY and MAC layers and a network layer mesh routing protocol. In this paper, we describe various implementation aspects of ZigBee PRO and present performance data on a ZigBee PRO mesh network for point-to-point and multi-hop transmission using XBee-PRO ZB modules. In particular, we present findings on network performance when a node is constantly moving and changing routes. We found that in the worst case where packets are transmitted immediately after the old route is no longer physically possible, extreme packet loss occurs as the network cannot perform route maintenance operations in time. With more gradual movement, however, the network operates without packet loss.

1 Introduction

The ZigBee Specification provides support for developing applications and services for low power nodes in a variety of network topologies, such as mesh

or star networks. The specification adopts a layered approach similar to the Open Systems Interconnection (OSI) model. It specifies an application layer to provide services with, and a network layer for routing services amongst nodes. It also mandates that ZigBee nodes comply with the IEEE 802.15.4 Low-Rate Wireless Personal Area Network (LR-WPAN) standard [1]. IEEE 802.15.4 is a specification for the physical (PHY) and medium access control (MAC) layers for use in low power and low cost wireless modules. The PHY layer specifies several bands and their associated data rates for use. The modules that we shall use operate at 2.4GHz with a maximum throughput of 250kbps. The MAC layer specification discusses network association and dissociation techniques along with one-hop communication in star or point-to-point topologies.

The ZigBee network layer specifies the protocol for joining and leaving networks as well as various routing techniques. In the latest standard, a new specification called ZigBee PRO was created that changed network layer operation. ZigBee PRO removed the use of cluster-tree routing and introduces source, or many-to-one, routing. Addresses are also assigned randomly instead of being based off of tree topology [4]. As the ZigBee modules we will use for testing, XBee-PRO modules, implement the PRO specification, we will focus on their specific network layer, and in particular the mesh routing implementation. The mesh routing algorithm is based off of the Ad-hoc On-Demand Distance Vector Routing Algorithm [9]. In the remainder of this introduction, we shall discuss portions of the ZigBee stack that are relevant to our problem analysis.

1.1 A Discussion of the Ad-hoc On-Demand Distance Vector Algorithm

This section describes the operation of the Ad-hoc On-Demand Distance Vector algorithm (AODV) in order to facilitate the description of the ZigBee network layer (which is based on AODV). AODV employs reactive discovery mechanisms to route data through a mesh network. AODV is designed to accommodate a network consisting of all routing-capable nodes, and promises significant bandwidth and node memory savings over earlier distance vector algorithms.

These claims are derived from the on-demand nature of the algorithm. This implies that each node may not necessarily know routes to all other

nodes, nor will it always participate in the discovery or failure of all routes. The only routes that are refreshed periodically are those between a node and all of its neighbors, i.e. routes one hop away. These routes are maintained via one hop broadcast messages called "hello" messages. The failure to receive acknowledgements from a certain number of "hello" messages constitutes a link failure, which will be discussed later in this section.

Each node in a mesh network employing AODV contains a routing table, with entries corresponding to the different destination nodes it knows how to reach. A routing table entry is indexed by the destination address, and includes the address of the next hop in the path, the hop count to travel, a destination sequence number, a route neighbor list, and an expiration time. Should a node try to transmit data to another node that is not in its routing table, the node will attempt path discovery. Path discovery is performed by broadcasting a route request (RREQ) packet. The RREQ packet contains several fields, namely source and destination addresses, source and destination sequence numbers, a broadcast ID, and a hop count. The source address and broadcast ID together uniquely identify a RREQ packet (each node has its own broadcast ID that it increments every time it issues a RREQ packet). Once the RREQ packet arrives at a node, the node first verifies that it has not received this particular RREQ packet; if it has, it drops the packet. It then checks its own routing table to see if it knows of a valid route to the destination. A route is valid if the routing table entry's destination sequence number is larger than the one present in the RREQ packet. If the routing table does not contain such a route, the node increments the hop count field in the RREQ packet and rebroadcasts it.

If a node does contain a valid route to the destination in its routing table, it unicasts a route reply (RREP) packet to the source node. Every node receiving the RREQ packet so far has kept in memory the address of the node that directly sent it the RREQ packet. This information is used to construct a reverse route back to the source for the RREP packet to follow. The RREP packet contains fields for source and destination addresses, destination sequence number, hop count, and lifetime. The source and destination fields correspond to the same source and destination fields of the RREQ packet. Each node that receives the RREP packet updates its routing table and retransmits the packet towards the source. If there is more than one node with a valid route, though, multiple RREP packets will be sent to nodes on the reverse path. To reduce bandwidth use without sacrificing quality results, a node will only retransmit a RREP packet if the destination sequence

number is greater than the one stored in its routing table (if they are equal, it will retransmit if the hop count value in the packet is lower). The source can act once it receives an RREP packet; if another comes it can update its routing table for another transmission, and the nodes along the path can always use the most updated path in their routing tables (since routing tables only store next hops, not the entire path). Meanwhile, nodes that received the RREQ packet but not the RREP packet (not part of the valid route) will timeout, erasing their reverse path memory.

AODV still works in the presence of link failure. Link failure is detected locally when a node does not receive "hello" messages from its neighbor. If a node detects link failure, it transmits upstream a special RREP packet to all nodes part of the route neighbor list in the routing table entry (a list of all neighboring nodes that participated in this route). Nodes that wish to transmit to the node with a failed link will have to perform path discovery.

AODV shows its advantages in providing functionally correct mesh networking without high control overhead. Due to the heavy use of destination sequence numbers, routes are always loop-free (a looped RREQ packet would drop because of a lower number) and use the latest available path information. AODV works without periodic control message flooding or data packets with large control overhead (i.e. storing whole paths in data packets) [7].

1.2 Overview of Mesh Routing in the ZigBee Network Layer

The ZigBee mesh routing algorithm is based off of the premises of AODV. We shall now proceed to discuss the implementation of this algorithm in the ZigBee PRO specification, which introduces a focus on symmetric links and eliminates the tree routing mechanism. We assume a network of only a coordinator and routers. If end devices were present, their parents would handle all routing work, as they do not have that capability.

One particular change from AODV is the substitution of sequence numbers and hop count for route status and path cost, respectively. Route status is an indication of the usability of the route, with values such as ACTIVE, DISCOVERY_UNDERWAY, and INACTIVE (other values are omitted for simplicity). Path cost is the sum of the cost of each one hop link in a route. Link cost is a measure ranging from 0 to 7 calculated by the following func-

tion:

$$C\{l\} = min\left(7, round\left(\frac{1}{p_l^4}\right)\right)$$

Where p_l represents the probability of packet reception on the link l. Path cost is the measurement used at each node during path discovery to compare options.

Each of the ZigBee routers and the coordinator node contains a neighbor table, a routing table, and a route discovery table. The neighbor table contains entries for all nodes within one hop. Each entry also contains a value for outgoing link cost, which is measured by the neighbor. The routing table performs the same functionality as in AODV, except with the use of route status. The route discovery table is a temporary space, as entries only exist for the duration of a route discovery. A route discovery table entry contains data obtained from the route request frame (the RREQ equivalent), such as route request ID (the broadcast ID equivalent), address of the route request initiator, sender address (address of the previous hop), forward cost, residual cost, and an expiration time. The forward cost is the sum of link costs from the source to the current node, and the residual cost is the sum of link costs from the destination to the current node.

The routing process begins similarly to that of AODV. The source node first checks the neighbor table for a destination match and then the routing table for a valid route to the destination (i.e. one with a status of ACTIVE). If there is no route in the table and a discovery for this route is not already underway, a path discovery process is initiated.

The path discovery process begins similarly to AODV's, with the broadcast of a route request with an incremented route request ID. The route request contains similar fields to the RREQ, with the substitution of path cost for hop count.

The node that receives the route request computes a new path cost by adding the maximum of the incoming and outgoing link costs for itself to the path cost value in the route request. It then compares this value with the forward cost value present in the appropriate route discovery table entry (or creates a new entry if one does not exist). If the new calculated path cost is greater than the forward cost, the route request is terminated. New routing table entries are created for both the forward and reverse routes. The entry for the forward route is initiated with a status of DISCOVERY_UNDERWAY, while the other entry is set to ACTIVE. The route request is then rebroadcasted if the node is not the destination.

If the node receiving the route request is the destination, then a route reply is created. The route reply contains source, destination, and path cost fields like the route request. The destination's link cost replaces the path cost field of the route reply. The route reply is then unicasted to the source via the sender addresses present in the route discovery table entries.

The node that receives the route reply compares the residual cost field in the appropriate entry of the route discovery table with the path cost field of the route reply. If the value in the route reply is smaller, then the routing table forward entry is updated with the correct next hop address. If the node is not the source, then the path cost is then updated in the route reply frame and the route reply frame continues to move upstream. If it is the source, then the route reply terminates and the route status changes to ACTIVE.

Route maintenance procedures in ZigBee mesh routing also match closely those of AODV. A failure counter is kept for outgoing links to neighbors, and link status messages are periodically sent to neighbors. If the failure counter indicates a link failure, and the forwarding of a packet fails, a network status command frame is sent back to the source node of the packet. The source node then removes the routing table entry for that node [1].

The differences between ZigBee mesh routing and AODV have their benefits and costs. The use of path cost allows the network to be more aware of physical conditions, instead of treating all links equally. Additionally, the inclusion of the reverse path in the routing table removes some control overhead. However, the removal of sequence numbers requires that the route request must travel all the way to the destination, which is not required in AODV.

2 Problem Description

Our goal is to evaluate the performance of a mesh network consisting of XBee-PRO modules, particularly with moving nodes. Our initial hypothesis was that the constant moving of nodes should affect ZigBee mesh routing operation. We believed that movement at high enough speeds could cause frequent link failures, which would negate the benefits of routing tables as new paths would be needed. In the worst case, links would have to be repaired via path discovery upon every transmit, which could significantly increase latency.

We conducted experiments measuring performance data for point-to-

point and multi-hop networks, as well as two tests with a moving node to test our hypothesis.

3 Related Work

Previous works [6, 8] have done simulation and experimental performance analysis of the IEEE 802.15.4 PHY and MAC layers. Additionally, work has been done in analyzing ZigBee network performance in star and point-to-point configurations [2]. We could not find research on analyzing ZigBee multi-hop performance or performance with moving nodes.

Since we are also interested in the performance of ZigBee networks for use in UAVs, we felt it necessary to mention SensorFlock. A team at University of Colorado, Boulder developed SensorFlock, "an airborne wireless sensor network of micro-air vehicles," to study toxic plume dispersion [5]. SensorFlock entails a set of five micro-air vehicles, each weighing less than 500 grams. Each plane contains an autopilot board with a XBee-PRO module. Instead of using the ZigBee network layer, the team implemented their own routing algorithm that appears to based off of AODV. They performed measurements of received signal strength indication (RSSI), for air-to-air, air-to-ground, and ground-to-ground communication, and found air-to-air communication to have the highest RSSI as distance between nodes was varied. Packet loss was also found to increase with distance more rapidly with ground-to-ground communication over air-to-ground communication. They did not measure multi-hop performance, however, as all vehicles were within distance of the ground control station.

4 Experimental Setup

To conduct our experiments, we used XBee-PRO ZB modules from Digi International (model XBP24-ZB). Each module had an RPSMA connector with a 2.2dBi duck antenna attached. We had one coordinator node and several router nodes. The coordinator was running Digi's latest API firmware (version 2170). The router nodes were either running the latest API firmware (version 2370) or the latest AT firmware (version 2270). We shall address the differences between API and AT firmware in our analysis of the experimental results. All nodes were set to use the same PAN ID and to have a baud rate

of 115200. All measurements were conducted after our control network was fully formed (confirmed via the XBee module's Join Verification setting).

We used FTDI USB to Serial integrated circuits to interface the modules with x86 PCs. Data packets were always sent by the coordinator module to a single router module. To send packets and perform measurements, we wrote Java programs that used Andrew Rapp's open source XBee API [10] to interface with the module. Our benchmarking code is also open source and is available at `http://github.com/wjwwood/au-proteus/tree/master/xbee-api/src/com/GCS/xbee_test/`. Our benchmark logs are also available at the root folder of this github repository.

When conducting experiments, we used packets with the maximum data payload size. For a ZigBee PRO network using the mesh routing protocol, this is 84 bytes. The packets with control overhead, though, have a size of 128 bytes. The data payload consisted of a sequence number and a repeated constant value. We describe the total transmission time as the elapsed time starting immediately before sending a packet and ending when an acknowledgement frame (ACK) has been received. Throughput is calculated as the total packet size (128 bytes) divided by the total transmission time. We declared an error when we failed to receive an ACK frame for a transmission or when the received packet had an inaccurate sequence number.

4.1 Point-to-Point Experiments

We first conducted experiments on one-hop packet transmission from a coordinator module to a router module approximately 1.5 meters apart. Since we have only one-hop transmission, we specified the "NH" parameter in the firmwares of the XBee modules to be 1 to minimize unicast timeout. We used "synchronous" and "asynchronous" transmission methods in different experiments. In synchronous transmission, the program would not request another packet transmission unless an ACK frame had been received for the previous transmission. In asynchronous transmission, the program did not wait for ACK frames and requested another transmission after a specified delay. The receiving router module was connected to a PC that verified the data payload of each packet. In each iteration of an experiment, we sent 1000 128 byte packets. We conducted 3 iterations of each experiment and averaged the results (for 27ms- and 28ms-delayed asynchronous tests, we performed 4 iterations to catch transmission errors).

4.1.1 Synchronous Transmission with API Router

In this experiment, we collected transmission latency, error count, RSSI, and throughput measurements when transmitting 1000 packets from an API coordinator module to an API router module. The transmissions were synchronous, so transmission would only continue if an ACK frame was received for the previous transmission. This experiment was conducted to have a performance baseline for a ZigBee network.

4.1.2 Synchronous Transmission with AT Router

This experiment was conducted in the same manner as the previous, except using AT firmware instead of API firmware for the router. As the XBee API only works with XBee modules having API firmware, we could not verify the received packet's data payload contents using a Java program. This experiment only served to compare the performance of the different firmware versions.

4.1.3 Synchronous Transmission with API Router and no APS ACK

In order to see the impact of ACK packets on network performance, we disabled application support sub-layer (APS) ACK packet transmission. This is a unicast ACK packet that travels from the destination node to the source node, and is requested by the ZigBee APS layer and delivered by the ZigBee network layer. Due to API firmware, the source node still receives an ACK frame after the unicast timeout.

4.1.4 Asynchronous Transmission with API Router

We used asynchronous transmission with varying delay times to measure the impact of transmission delay on error rate. We delayed the transmission thread by 1ms intervals from 31ms to 20ms. The delay does not include the time to send the transmit request frame through the serial link to the source node. When collecting asynchronous data, the latency was redefined as the time elapsed in just sending the packet itself.

4.2 Multi-Hop Experiments

We measured multi-hop performance to compare results with both point-to-point measurements and measurements with moving nodes. To ensure multi-hop transmission, the destination router module had its antenna removed and was moved away from the source coordinator until no signal could be established. A middle router node (with AT firmware) was then placed at this point, which reestablished the signal. The NH parameter was also tuned to allow a greater unicast timeout. We measured the performance of synchronous packet transmission to the end router module two hops away. We conducted three experiments, each with three iterations of 1000 synchronous packet transmissions.

4.2.1 Multi-Hop Synchronous Packet Transmission

This experiment provided results for baseline multi-hop performance, with 1000 packets being synchronously transmitted two hops away.

4.2.2 Multi-Hop without APS ACK

This experiment was conducted with no APS ACK packets as comparison to baseline multi-hop network performance.

4.2.3 Multi-Hop without 16-bit Addresses

In this experiment, we sent all packets without explicitly specifying the 16-bit network address of the recipient in the ZigBee Transmit Request frame. Such a practice could either force address discovery or an address table lookup. We conducted this experiment to look at those effects.

4.3 Moving Node Experiments

We finally conducted two moving node experiments to witness the performance loss with a multi-hop network. Our test network consisted of a coordinator module, two stationary router modules, and a moving router module. All modules had antennas except for the moving router. The two stationary routers were placed in perpendicular hallways. This placement was made such that when the moving router was next to one of the stationary routers,

it could not establish a link with the other router. The moving router could also not establish a one hop link to the coordinator module.

4.3.1 Explicit Packets

In the first moving node experiment, we moved the end router node between the two stationary routers every two synchronous packet transmissions. Each packet transmission was initiated manually at the coordinator module. This experiment illustrated the worst case scenario where a node "instantaneously" moved to a different part of the network in between transmissions.

4.3.2 Walking Test

In our other moving node experiment, we walked the moving node through a path while synchronous packet transmission was constantly occurring. The path was always out of range of the coordinator, and started with being in range of only one router. The middle of the path was in range of both routers, and the end of the path was in range of only the other router. This test was meant to emulate a more real world scenario of a moving node.

5 Results and Discussion

In this section, we present our experimental data in tabular and graphical forms, along with our interpretation of the data.

5.1 Synchronous Point-to-Point and Multi-Hop Transmission

Table 1 shows our collected data for synchronous transmissions in both point-to-point and multi-hop configurations.

5.1.1 Point-to-Point Performance Analysis

The data for the point-to-point transmission is a baseline standard for our mesh network's performance. As the transmissions are synchronous and the coordinator receives acknowledgements, there is no queuing delay and no potential for packet loss. The total transmission time is made up of several other components, however. The most dominant components of the total

Setup	Avg. Transmission Time (ms)	Error (# of packets)	RSSI (dBm)	Throughput (kbps)
4.1.1	39.65	0	-41.33	25.67
4.1.2	31.86	0	-39.00	31.90
4.1.3	22.06	0	-42.33	45.96
4.2.1	58.62	0	-84.33	17.44
4.2.2	42.13	9	-86.67	24.56
4.2.3	62.53	0	-81.67	16.39

Figure 1: Data from Point-to-Point and Multi-Hop Synchronous Packet Transmission Experiments

transmission time are the transmission time of the ZigBee Transmit Request frame to the source module via the UART and the transmission time of the ZigBee Receive Packet frame to the PC from the receiving module via the UART. The serial connection has a bandwidth of 115,200bps, so it is the bottleneck in data transfer (as the physical layer has a bandwidth of 250kbps). As our ZigBee transmit frames are 102 bytes in size, and the ZigBee receive frames are 100 bytes in size [3], UART transmission for each ideally takes 7.1ms and 6.94ms, respectively. Additionally, reception of the ZigBee Transmit Status frame (i.e. the ACK frame) via the UART ideally takes 0.76ms. Transmission time of the packet, assuming an ideal 250kbps physical layer, is 4.1ms. Transmission time of the ACK packet could be as low as 0.32ms. Our expected total transmission time is then 19.22ms, leaving us an error of 20.43ms. We can attribute this time difference to error (such as bandwidth being lower than specified), and the node processing time that we currently cannot quantify.

The measurements using an AT router somewhat validate our notion that UART delay is the dominant transmission time factor. There is a latency difference of 7.79ms with the previous measurement. In an XBee module with an AT APS layer, the XBee module is programmed with AT commands and appears as a simple serial communication link to a connected device (e.g. PC, microcontroller). On the other hand, XBee modules with API firmware communicate with frames, encapsulating all data for more advanced features and simpler programmability [3]. Since the AT router does not encapsulate the received packet in a frame, the UART receive overhead drops from 6.94ms

to 5.93ms (the time to just transmit the 84 byte packet payload). However, this only explains 1.11ms of the difference. The rest of the discrepancy could lie in decreased processing time for AT routers over API (which has frame checksumming and encapsulation [3]).

Our results in point-to-point packet transmission without APS ACK packets provides more insight into the timing of the system. Ideally with such a setup, the ZigBee Transmit Status frame should be received immediately after the packet has been transmitted via the radio. This then implies a total transmission time of 12ms, which is a discrepancy of 10ms from the experimental data. However, since we do not involve the receiving node in calculations when no APS ACK packets are present, we can isolate that transmission time period by comparison with our previous measurements. Doing this shows that 17.6ms is needed in processing and receiving the ZigBee Receive Packet frame and transmitting an APS ACK packet. An additional interesting result is that our results reported no transmission errors. We can explain this by the lack of queueing time. Since we only transmit after receiving a Transmit Status frame, the transmission queue will always be empty and thus not drop packets.

5.1.2 Multi-Hop Performance Analysis

Our multi-hop experiments provide baseline measurements for comparison with the results of the moving node tests. Synchronous transmission over two-hops results in a 32% throughput performance decrease over the analogous one-hop transmission. It is not a full 50% decrease as the middle router module does not output anything on the serial link. The additional latency comes from the transmission and processing time of the data packet and of the ACK packet by the middle router. Additionally, performance loss is expected as the RSSI of the last hop is significantly worse in the multi-hop experiment than in the one-hop experiment.

Multi-hop results without APS ACK packets show our first sign of packet loss with synchronous transmission. Our total transmission time is also unreasonably higher than the point-to-point data with no APS ACK, which should not be the case as the coordinator node is still doing the same job. However, a closer look at our log data revealed the discrepancy. The packet loss was due to 5 second transmission timeouts, because of link failure with the middle router module. These 5 second timeouts significantly increased the average total transmission time. However, the median of total transmis-

sion time over 3000 packets is 25ms, which is a much more reasonable value. Even this 3ms difference between the point-to-point data can be explained by the low RSSI, which could result in lower physical layer throughput.

The removal of the 16-bit address in the ZigBee Transmit Request frame was meant as a measure to force the coordinator to perform address or route discovery operations. Address discovery is necessary because all ZigBee routing is done using 16-bit network addresses instead of each node's unique 64-bit IEEE address. Address discovery involves a broadcast transmission of the 64-bit address until the transmission reaches the correct node, so theoretically it is a slow operation. However, each XBee coordinator and module possesses an address table that it uses in the APS layer to lookup 16-bit addresses from 64-bit addresses. The 4ms increase in total transmission time could then be explained by these lookups instead of constant address discovery. Our log files show that no address discovery was performed, which may confirm this. Since the table is in the APS layer, the network layer can operate with a 16-bit address and avoid route discovery.

5.2 Asynchronous Point-to-Point Transmission

This section provides data and analysis for the experiment detailed in Section 4.1.4. As can be seen from Figure 2 as the delay time between sequential transmissions decreases, the throughput of the network increases linearly. This makes sense because as the delay time decreases, more data is sent over the network in any arbitrary block of time. However, as Figure 3 shows, there are almost no errors in data transmissions up until the total transmission time is 27 ms. After that point, the network suffers significant packet loss since the receiving node cannot keep up with the amount of data being sent. Since the delay is initiated after the data is sent across the UART to the transmitting XBee, the delay represents the amount of time it takes for the data to arrive and be processed by the receiving XBee. The data packet was 128 bytes in size, ideally making the amount of time it took to traverse the link 4.1ms. This leaves 20.9ms of unaccounted delay. To determine if there was a queuing delay the equation for traffic intensity applies: La/R, where L is the number of bits in a packet, a is the number of packets/s, and R is the transmission rate in bits per second. Thus, with a total transmission time of 27ms (i.e. 37 packets/s), a packet size of 128 bytes, and a transmission rate of 250kbps the traffic intensity is 0.14787, which is much less than 1. Thus, we expect no queuing delay and no dropped packets. However, in practicality we

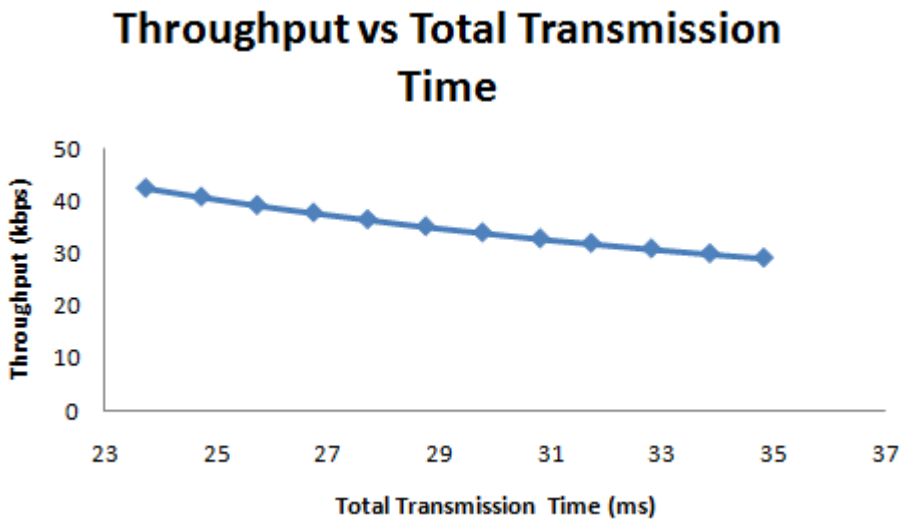

Figure 2: Throughput vs. Total Transmission Time in Asynchronous Point-to-Point Transmission

observed that there must be queuing delay since packets are being dropped at the receiving node. This can once again be attributed to the fact that the network does not come close to reaching reliable 250kbps transmission rates, thus making the traffic intensity peak with less data traversing the network.

5.3 Moving Node Performance Discussion

We do not show numerical data in this section, but all of our logs are online as previously mentioned. The results for the experiment mentioned in Section 4.3.1 indicate that data transmission is guaranteed successful only when the mobile node first associates with the network. After the initial association, when the node is moved, the route from the coordinator to it is no longer valid, and the middle router must respond with a network status command frame indicating a broken link. If the next attempted transmission occurs before all of the routers broadcast their link status messages, the broken link is only discovered after a transmission is attempted and then fails after a timeout. A route discovery is then initiated. However, the packet is never received at the receiving node and the data is ultimately lost. If the

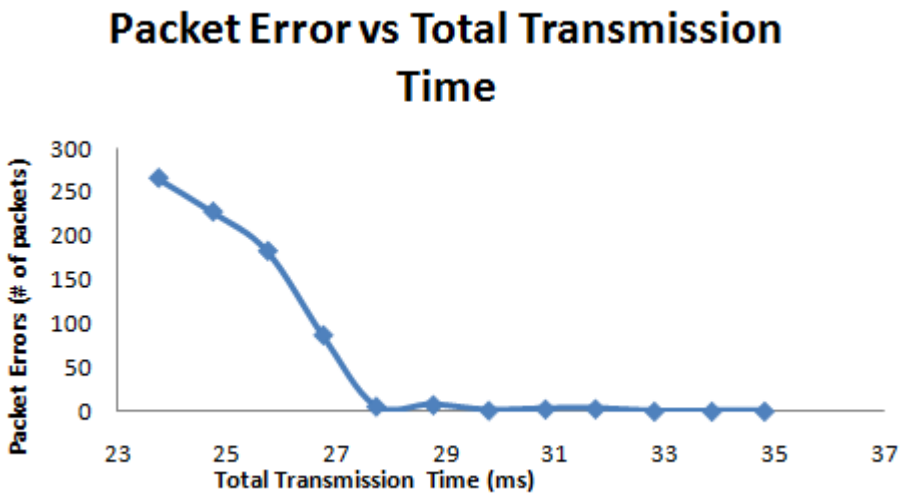

Figure 3: Packet Error vs. Total Transmission Time in Asynchronous Point-to-Point Transmission

destination node continuously hops between routers after each transmission attempt, it could have 100% packet loss. However, if the transition were gradual as the experiment mentioned in Section 4.3.2, the node would "smartly" associate with the closest router and facilitate reliable communication; this is because the neighbor table ages, deleting old entries periodically. It is only when the transition is sufficiently fast that the neighbor table is not refreshed and the network status command frame (indicating a broken link) is not sent that there is significant packet loss. According to the XBee-PRO manual, link status messages are sent 3-4 times a minute [3]. We verified that no packet loss would occur if the broken link was reported in time by waiting 15 seconds after moving the node before sending another packet. When doing this, we received all packets. Even given this, the maximum throughput of a moving node is slightly less than that of a static multi-hop network due to the high latencies at the fringes of the router range before the mobile node reassociates with the stronger link.

6 Conclusion

The goal of this paper was to evaluate the performance of moving nodes in a ZigBee PRO mesh network using XBee-PRO modules as the nodes and measuring the maximum throughput and latency of the network. The results indicate that the mesh network does not respond favorably to nodes that are moved from one router to another before the routers send their network status command frames to indicate a broken link. We observed the possibility for 100% packet loss should the movement occur in between neighbor table refreshes.

We would also like to note that the performance of our multi-hop network could have possibly been attributed to the low RSSI of the link between the source and middle modules. We can attribute this low RSSI to our indoor setting, as there was a door between the source and middle modules, and the modules themselves were approximately 20 meters apart. However, since ZigBee modules are often used for home automation purposes [1], we find our results relatively disconcerting.

In relation to our work with UAV mesh networking, we find through our results that ZigBee mesh routing with the necessity of multiple hops will result in undesirable performance. Alternatives such as many-to-one routing may provide better performance, and may be a subject of future research.

Acknowledgment

The authors thank Drs. Saad Biaz and Richard Chapman of Auburn University for their guidance and help in securing materials for research. This work was funded by NSF Award #0851960.

References

[1] ZigBee Alliance. Zigbee specification. Technical report, ZigBee Alliance, January 2008.

[2] M. Armholt, S. Junnila, and I. Defee. A non-beaconing ZigBee network implementation and performance study. In *Communications, 2007. ICC '07. IEEE International Conference on*, pages 3232 –3236, 24-28 2007.

[3] Digi International. *XBee/XBee-PRO ZB RF Modules*, April 2010.

[4] Bob Gohn. The ZigBee PRO feature set: More of a good thing. `http://www.embedded.com/design/205100696`, December 2007.

[5] Ahmad Bilal Hasan, Bill Pisano, Saroch Panichsakul, Pete Gray, Jyh Huang, Richard Han, Dale Lawrence, Kamran Mohseni, Ahmad Bilal Hasan, Bill Pisano, Saroch Panichsakul, Pete Gray, Jyh Huang, Richard Han, Dale Lawrence, and Kamran Mohseni. SensorFlock: A mobile system of networked micro-air vehicles., tr-cu-cs-1018-06, u. of colorado at boulder, 2006.

[6] Mikko Kohvakka, Mauri Kuorilehto, Marko Hännikäinen, and Timo D. Hämäläinen. Performance analysis of IEEE 802.15.4 and ZigBee for large-scale wireless sensor network applications. In *PE-WASUN '06: Proceedings of the 3rd ACM international workshop on Performance evaluation of wireless ad hoc, sensor and ubiquitous networks*, pages 48–57, New York, NY, USA, 2006. ACM.

[7] Charles E. Perkins and Elizabeth M. Royer. Ad-hoc on-demand distance vector routing. In *IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, 1999.

[8] M. Petrova, J. Riihijarvi, P. Mahonen, and S. Labella. Performance study of IEEE 802.15.4 using measurements and simulations. In *Wireless Communications and Networking Conference, 2006. WCNC 2006. IEEE*, volume 1, pages 487 –492, 3-6 2006.

[9] Peng Ran, Mao heng Sun, and You min Zou. ZigBee routing selection strategy based on data services and energy-balanced ZigBee routing. *Asia-Pacific Conference on Services Computing. 2006 IEEE*, 0:400–404, 2006.

[10] Andrew Rapp. xbee-api. `http://code.google.com/p/xbee-api/`, May 2009.