THE INTEL® MOTE PLATFORM: A BLUETOOTH*-BASED SENSOR NETWORK FOR INDUSTRIAL MONITORING

Lama Nachman, Ralph Kling, Robert Adler, Jonathan Huang, Vincent Hummel

Corporate Technology Group
Intel Corporation
Santa Clara, CA, USA

Abstract--The Intel Mote is a new sensor node platform motivated by several design goals: increased CPU performance, improved radio bandwidth and reliability, and the usage of commercial off-the-shelf components in order to maintain cost-effectiveness. This new platform is built around an integrated wireless microcontroller consisting of an ARM*7 core, a Bluetooth radio, SRAM and FLASH memory, as well as various I/O options. The Intel Mote software architecture is based on an ARM port of TinyOS. Networking and routing layers have been created on top of the TinyOS base to provide Bluetooth-based multi-hop functionality. The network is self-organizing on startup and has mechanisms to repair failed links and circumvent failed nodes. A reliable high bandwidth streaming transport layer has also been created.

The Intel Mote was deployed in an equipment monitoring application using industrial vibration sensors. This application was chosen since it benefits from the increased platform capabilities and network bandwidth of the Intel Mote platform. The paper presents a detailed analysis of the observed network operation, packet transfer rates, and power consumption.

Keywords—*wireless sensor networks; Intel mote; condition based monitoring; Bluetooth motes*

I. INTRODUCTION

Sensor networks have been built for a number of different applications and environments. Traditionally, motes have been based on 8-bit microcontrollers such as the Atmel*-128 and single channel radios in the 300-900 MHz range [1]. These platforms worked well for initial deployments that were focused on environmental monitoring [2]. Typical sensors used by those applications required fairly low bandwidth for measuring temperature, humidity or barometric pressure data. The sensor network hardware was sufficient to handle the requirements of such sensor inputs.

Increasingly, sensor networks are being deployed in industrial monitoring solutions. This is a new application area with a number of different requirements. Some of the desired measurements, such as vibration and acceleration, require higher bandwidth and longer sampling periods. This in turn necessitates platforms that can acquire and transmit larger data streams efficiently. In addition, local CPU computational capabilities can be used for data compression as well as initial classification and analysis.

II. INTEL MOTE PLATFORM

The Intel Mote [3] was designed to address the above mentioned application requirements by introducing a new platform design. Further design goals included improved radio reliability, high level of integration, and cost-effectiveness. An important aspect of the platform design was to achieve increased performance while still offering competitive battery life. To satisfy these requirements we chose an integrated wireless microcontroller module from Zeevo*, Inc. that incorporates an ARM7TDMI core and a CMOS Bluetooth radio [4] [5]. Of particular concern was that the chosen module fully supports the Bluetooth "scatternet" operation in order to be able to build mesh networks. In addition, the hardware needed to support the Bluetooth low power modes as well as an overall platform sleep mode.

The Intel Mote is built on a 3x3cm circuit board that integrates the Zeevo module, a surface-mount 2.4GHz antenna, various digital I/O options using stackable connectors and a multi-color status LED. The heart of the platform is the integrated 12MHz CPU, BT Radio, 64KB SRAM, 512KB FLASH module that is contained within an 11x13mm BGA package. The connectors expose 2 UART ports (one without flow control), USB client, GPIOs and power. The radio's range is approximately 30 meters with the built in antenna, however, we were able to extend the range up to a 100 meters using an omni-directional external antenna. By simply removing the two matching network components, an ultra-small coaxial connector (Hirose U.FL) can be soldered in their place to allow for an external antenna solution.

When choosing an operating system environment for the Intel Mote we decided to use TinyOS [6]. It provides the basic functionality needed for the intended applications and benefits from a large and active user community. We therefore ported TinyOS to the ARM architecture and wrote a set of platform drivers that make the various platform I/O components available to the application programmer. The OS and radio stack leave about 11 KB of free SRAM to be used by the application.

In order to facilitate sensor network research, the Intel Mote was designed as a modular platform. This allows custom sensor boards, battery boards, interface boards and debug boards to be attached to the system in a flexible manner. As shown in Fig. 1, this flexibility is provided via a set of pass-through connectors on the main board.

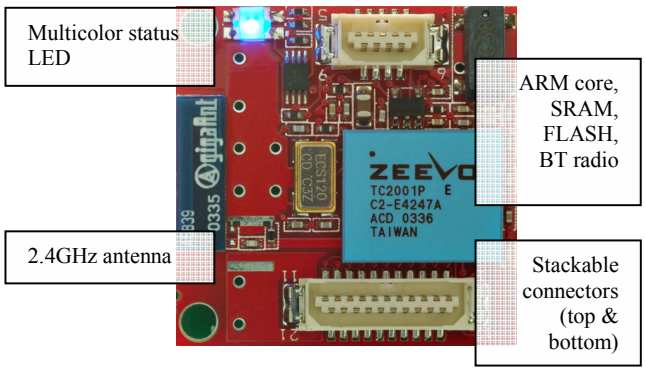

Figure 1. Intel Mote platform

III. NETWORK STACK

A significant difference between the Intel Mote networking stack and the networking stacks of previous motes is that the Intel Mote network is built on top of a point-to-point Bluetooth baseband. In contrast, most other mote platforms broadcast on a common communication channel using a CSMA protocol [7]. As a result, we implemented new network formation and routing algorithms adapted to connection oriented radios.

In order to support the data integrity requirements of the above mentioned applications we created a number of new communication components on top of TinyOS that are optimized for the Bluetooth master/slave and piconet/scatternet operation. The network is self-organizing on startup by employing a distributed node discovery and connection procedure. After basic network establishment, routing information is exchanged between the nodes. We also provide automatic network repair in case of node or link failures and a low power mode that maintains network connectivity.

A. Network Formation

We have implemented a scatternet formation algorithm which creates a network with a tree topology using a pre-determined node at the root of the tree. This is a simplified version of the algorithm described in [8]. At initialization each node randomly enters either an inquiry scan or inquiry state. Each node then alternates between these two states until it discovers another node or another node attempts to form a connection with it. When a connection is made, the two nodes exchange information including a flag which indicates whether each node has a route back to the root node. If neither node has a path back to the root, the connection is broken and both nodes resume searching.

Any node connected directly to the root node will learn that it has a path back to the root node during the initial handshake. For every new connection, the node initially connected to the root node becomes the master in the local piconet and the originally free node becomes the slave. After a node has joined the tree, it stops performing the inquiry operation and remains in an inquiry scan state. By stopping the inquiry process for nodes already in the tree connected to the root, this algorithm avoids creating loops in the tree. After the root node connects with another node, all other connections are initiated by unconnected nodes called free nodes. Free nodes continue the process of alternating between inquiry and inquiry scan states. When a free node discovers another node, it determines whether the other node has a path to the root node. If it is not connected the free node disconnects and continues its search. If the other node is connected, the free node joins the other node's piconet as a slave. This maintains the tree topology for the network in which each master in every piconet is closer to the root than its slaves.

B. Network Routing and Repair

For the topologies discussed in this paper, a simple routing algorithm is used to maintain a path from the root to each node and from each node back to the root. The root periodically sends out a routing beacon to each of its slaves. This beacon continues to flow down the tree, with each master transmitting it to each of its slaves. As the beacon flow progresses each node caches the next hop back to the root and echoes a route reply back to the root node. Intermediate nodes snoop these replies and refresh their route tables for descendent nodes.

The lower level Bluetooth stack periodically exchanges maintenance packets across active links. When a link fails and these maintenance packets are not received for a pre-determined period of time, the lower level stack breaks the connection and signals to the upper level that a connection was dropped. The scatternet formation algorithm receives these signals and updates its view of the network accordingly. If a master receives a signal indicating that it lost a link to a slave, it updates its routing table to indicate that the slave and its descendants are no longer reachable across that link. If a slave receives a signal indicating that it lost its connection to its master, it assumes that it can no longer reach the root of the tree and returns to the free node state in which it randomly alternates between the inquiry and inquiry scan states. It also disconnects from all of its slaves so they can also search for a new path back to the root node.

C. Reliability Protocol

We have defined an end-to-end reliable transport protocol to support reliable transmission of large datagrams in sensor networks. This was needed for the vibration sensing application described below, as each collection required the reliable transmission of a 6KB buffer, which does not fit into even the largest Bluetooth packet. A major requirement of this protocol was reduction of communication overhead, as well as memory requirements in each node. The data is divided into multiple fragments which may be transferred across multiple hops, may arrive out of order, or may be dropped by intermediate nodes. To be able to support different link types, the fragment size is negotiated in the connection setup process. This protocol relies on the receiver to periodically send a list of missing fragments and controls a fixed size sliding window. Since the receiver is aware of the total datagram size, it is able to verify the receipt of the complete datagram. The sender will not retransmit any fragments until it receives a NACK packet. It will only retransmit the fragments that the receiver has requested. The protocol has three phases, a connection setup, a data transfer and a final acknowledgement phase.

D. Low Power Mode

We implemented a low power protocol to enable the Bluetooth links to be put in low power mode while keeping the network in the connected state. This is extremely useful in cases where the network response time is required to be in the order of minutes. When the root node decides to put the network into a sleep state, e.g., after a collection cycle, it issues a "sleep" command to each of its slaves. Those store the sleep flag and also retransmit this command to each of their slaves, hence propagating it one level at a time down the tree. Once the sleep command reaches a leaf node, it sends an ACK back up the tree to its master. When a master receives an ACK from all of its slaves, it places the Bluetooth links to each of them in "hold" mode for a time period T_{hold}, currently set to 20 seconds, and sends an ACK to its master. These ACKs are propagated upwards until they reach the root node. Once a node has all its connections in hold mode, it sleeps itself for T_{hold}. When T_{hold} expires, the master of each piconet checks the sleep flag, if it is still set, it puts all of its slaves on hold again. Note that since a master puts all its slaves on hold at approximately the same time, they also wake up concurrently, hence enabling the master to wake up only once per T_{hold} interval to synchronize all its slaves.

When the cluster head decides to wake up the network for the next data collection phase, it sends a "wake" command through the network. Since all the connections are still maintained, the wake command will be sent to the first level slaves once T_{hold} expires, causing them to reset their sleep flags. As a result, these nodes will not attempt to put the links to their own (next level) slaves back in hold mode. The process repeats, traversing one level of the tree every T_{hold} period. Once the wake command reaches the leaf nodes, ACKs will be sent back up the tree, until they reach the root node indicating that the network is now awake. As a result, the network wake up time is approximately the tree depth * T_{hold}.

IV. INDUSTRIAL EQUIPMENT MONITORING

Equipment health monitoring is a typical industrial application that can benefit from a sensor network deployment. Currently, sensors are attached to critical equipment, their readings are manually collected and the data is analyzed off-line by computer. The data needs to be collected and analyzed frequently enough to proactively detect future equipment problems and perform preventative maintenance. This can be very critical because shutting down a factory to fix broken equipment can be very costly, in addition to the possibility of losing some of the product in the process. The benefits of using a wireless sensor network for an equipment monitoring application are numerous: First, by eliminating the labor intensive operation of manual data collection, cost savings can be achieved. Second, the automation of such repetitive tasks reduces the probability of human errors. Finally, using a wireless solution, in particular when retrofitting existing buildings, is significantly more cost-effective than using a wired solution.

We have performed a test deployment of Intel motes in one of Intel's central utility buildings. This building supplies wafer fabrication facilities with the resources needed for their operation. Data from vibration sensors connected to water pumps, compressors and chillers is regularly monitored. For ease of network management and reduction of maximum hop counts, the sensor network is divided into clusters. Each mote cluster has a head node that is responsible for communication scheduling and power management. It is connected to a line powered local "Stargate" gateway node [9] via a serial link. All gateways are connected to a root gateway via an 802.11b network. Finally, the root is connected to a central server that runs data analysis software. This test deployment was designed as proof of concept for sensor networks in real industrial applications and environments. In addition to the Intel Mote clusters, we also had MICA2 based clusters deployed in the same building. This enabled us to compare the features of the two platforms with respect to this industrial monitoring application.

A. Experiment description

We deployed 3 Intel Mote and 3 MICA2 clusters. Each cluster covered a set of equipment in separate sections of the building. The cluster sizes (up to 10 motes per cluster) were dictated by the location of the equipment. Each mote had about 5 sensors attached. All motes were instrumented to log many vital networking stats (data transfer delay, network formation overhead, packet retransmissions, etc). The experiment lasted for 7 days and generated significant statistics. The motes were configured to only sleep for one hour between collections in order to heavily exercise the network. All vibration data was successfully transferred via the gateways to the backend server and imported into Rockwell Enshare software for further analysis.

We have designed a custom sensor board to interface the Intel Mote to Wilcoxon 786A vibration sensors. Much like the manual method that this replaces, multiple sensors are connected to a single sensor network node. The Intel Mote's large amount of available memory and high speed DMA-based UART enabled us to keep the design simple and inexpensive. Each collection consisted of 3000 vibration samples, of 16 bits. We were able to fit a complete 6KB buffer in the internal SRAM of the Imote. The A/D sampling rate was 19.2KHz at 16 bits. The Intel Mote's UART was used to transfer the data from the sensor board at 460Kb/s. In the case of the MICA2 sensor board, an external RAM buffer and a second processor had to be added to the sensor board due to the limited capability of that platform.

B. Network Formation Overhead

The network formation is initiated when the cluster head starts. Since connections are maintained during the sleep phase, there is no need to reestablish them before every collection cycle, hence the network formation delay is amortized over many collection cycles. The network formation phase remains in effect until either all the motes are found or a timeout value is reached. For the purpose of this experiment, we set the timeout value to 2 minutes. Over the one week run, we had a total of 163 collection cycles. The cluster went through the network formation phase only 13 times. Some of the new network formations were caused by a special watchdog that monitors network performance. The average network formation delay for a 7-node cluster was 67 seconds, if we average over all collection cycles we get a network formation overhead of 5.3 seconds per collection. This data is shown in Fig. 2.

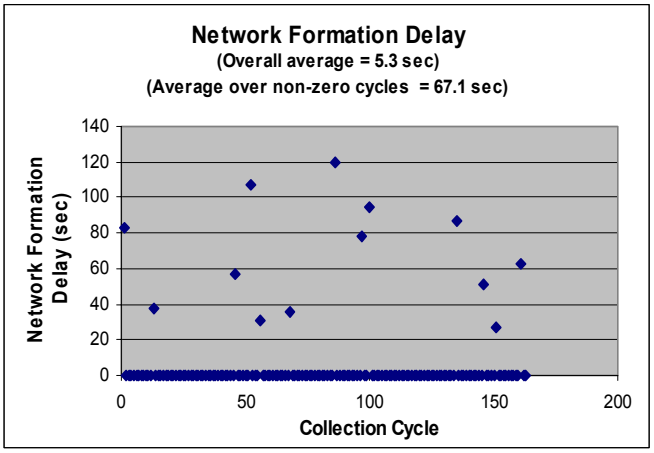

Figure 2. Network formation

C. Data Transfer Delay

Each mote transfers 3000 samples (6KB) of data per sensor using the reliable transport protocol described above resulting in a total data size of 30KB for all 5 sensors. The average transfer time for the 30KB of data is 88 seconds, with a standard deviation of 9s. By comparison, the MICA2 average transfer time was more than 10x that of the Imote, with a much larger spread as well. The number of fragments needed to transfer the 30KB of data is 320 (maximum fragment size is 94B), however, we see an average of 430 fragments transmitted per collection, this amounts to about 34% retransmitted packets. As expected, we observed that the average and variance in the number of retransmitted packets increases with the node hop count as shown in Fig. 3. We believe that most of this packet loss was due to aggressive NACK timing and buffer overflows. When we examined the number of fragments successfully received and divided that by the total number of transmitted fragments, we found that the loss rate is about 8.5%. This can be improved further by tweaking the NACK timing and optimizing the lower layer to switch faster between piconets.

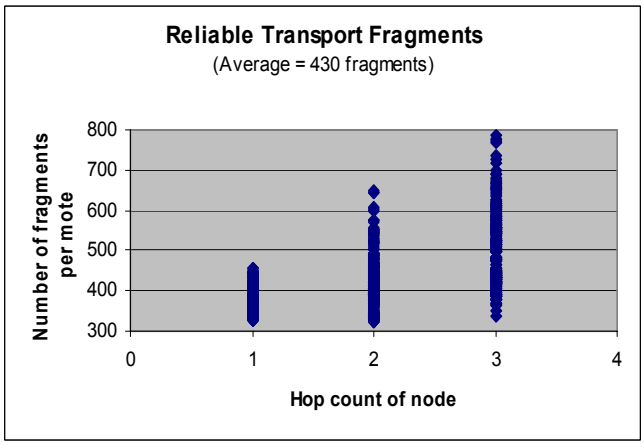

Figure 3. Reliable transport

D. Radio Performance

Two of the Intel Mote clusters had their cluster heads inside the building whereas the sensor motes were connected to equipment outside the building. The performance of these external clusters was very similar to the internal one, despite having to go through walls as well as longer hop distances. In addition, the performance of the Imote clusters seemed much more consistent across the different collection cycles compared to the Mica2 clusters. Fig. 4 shows the cumulative distribution function of the data transfer time for a Mica2 and an Imote cluster over the total number of collection cycles. It can be seen that the variance in the Imote performance is much less than the Mica2.

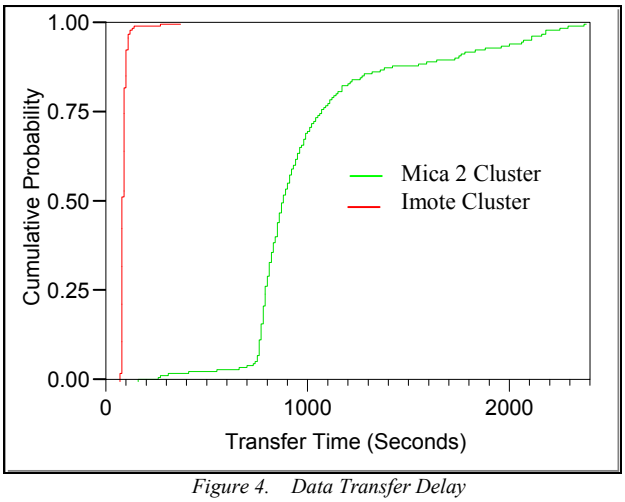

Figure 4. Data Transfer Delay

To investigate the effect of physical distance on transmission reliability, we performed a separate experiment: two nodes were connected directly and the slave transmitted a 3KB buffer to the master using the reliability protocol. The distance between the two nodes was increased in steps of 3 meters. At each location, 8 transmissions of the 3KB buffer were performed. Fig. 5 shows the min, max and average transmission times as a function of distance. The average transmission time for distances smaller than 30m is about 1.2 seconds. Since the sender is configured to send 100 byte packets every 40ms, 1.2s is the minimum achievable transmission time. As the distance is increased beyond 30m, we start seeing longer transmission times, as expected. However, even at 60m, we still get reliable transmissions (at the expense of higher power consumption). The above experiments were performed indoors, in a cubicle environment with people moving around and in the presence of 802.11b networks. In all of these experiments we hardly see any transport layer NACK packets sent, however, the transmission times vary quite a bit. This is a result of the Bluetooth MAC layer retries. As the link quality decreases, more packets fail the CRC check and don't get acknowledged. The link controller on the transmitter side will retry these packets multiple times.

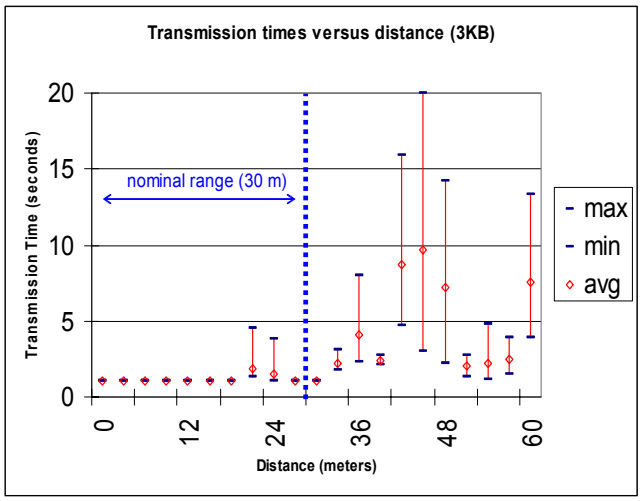

Figure 5. Radio range

E. Current Consumption

We have measured the current consumed in the different phases of operation. The following table shows the amount of current consumed and time spent in each mode. Note that we assumed the Imote radio is always on during an active cycle, this is a conservative estimate (Imote with radio off consumes about 9 mA). The table sets the sleep cycle to 1 hour to match the deployment. As the collection frequency is decreased, the sleep current dominates the current consumption. The battery solution that was used in this deployment (4 C-cells) has a capacity of ~9100mAh at 6V nominal. For this configuration, the resulting network lifetime is roughly 2 months. Since this deployment was a retrofit, supplying power hookups to the motes was deemed too expensive by the facilities team which might not be the case in new installations.

TABLE 1. Current consumption

Modes	**I_{supply} [mA]**	**t_{cycle} [s]**	**Avg. I_{supply} [mA]**
Sleep mode	3.0	3600.0	2.53
Imote + radio	20.7	591.0	2.87
Imote+Sensor	54.6	75.5	0.97
Imote+Sensor+A/D	58.8	1.5	0.02
Total cycle	-	4268.0	6.39

There are multiple lessons to take away from this experiment: First, the current consumed by the sensor and board exceeds that of the mote even in full power. Second, the current consumed to perform a collection cycle was less than one tenth that of the MICA2 clusters, due to the faster transfer times. This is where Bluetooth shows to be a much better fit for higher data rate applications.

As the collection frequency decreases, the sleep current dominates the current consumption. We had a higher average sleep current due to keeping the network connected at all times, in addition to a few software problems that need to be addressed. We discuss these possible optimizations below.

Currently the processor wakes up frequently to service some high resolution timers (reliable transport layer) that are not needed in the low power mode. The average low power mode current can be reduced from 3mA to at least 1.5mA by stopping all of these timers before we enter the sleep cycle. This will increase the average lifetime to more than 6 months in the daily collection case.

Another way to reduce the current consumption is to tear down the network completely in-between collections and re-establish the network at the beginning of each collection cycle. The sleep current can be reduced to <700uA in this case. However, this means that we will now encounter the network formation overhead in every collection. We chose to go with the current design to allow for on-demand data collection and reduce the network response time to less than one minute. However, if this feature is not needed and we assume that we can only access the network through the preprogrammed collection cycles, we can increase the battery lifetime substantially. Fig. 6 compares the average current consumption and the battery lifetime (shown in months) of both approaches for varying sleep duration. Note that network formation overhead has been added for the disconnected network case. It can be seen that even for the one hour sleep duration, the disconnected network approach consumes less power.

The current consumption can be reduced even further if we use an external real time clock solution and completely shut off the processor during the sleep cycle, which can bring the sleep current down to the 10uA range. In this case and with a daily collection cycle we would get about 4.5 years of battery life.

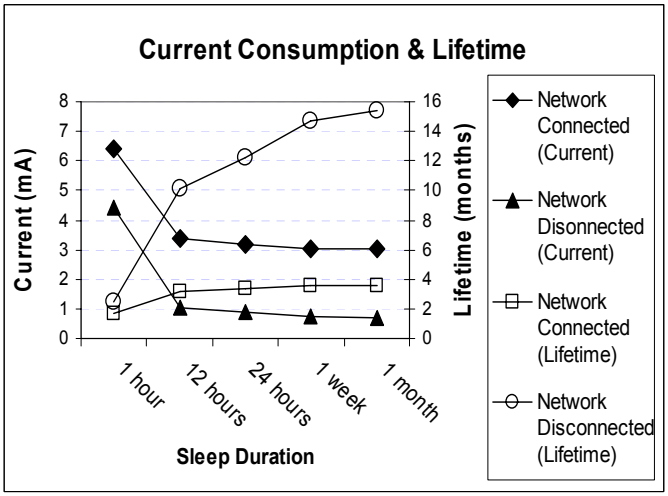

Figure 6. Current consumption & Lifetime

V. CONCLUSIONS AND FUTURE WORK

This paper describes the development and deployment of the Intel Mote. The key contributions are:

- Created an advanced mote with increased capabilities based on low-cost components
- Successfully used Bluetooth scatternet mode to create self-organizing, multi-hop networks
- Demonstrated Bluetooth and platform low power modes to enable battery powered network operation
- Showed the advantages of a more capable mote design using the equipment monitoring application

Our development of Bluetooth based motes and the appropriate networking components have shown that this protocol can be

successfully deployed in real sensor network applications. In order to take advantage of the Intel Mote platform's capabilities and its Bluetooth radio, an application should benefit from the increased computational resources and radio bandwidth. The equipment monitoring application that we are currently piloting is a good example thereof.

We have also shown that it is possible to use the Bluetooth scatternet mode to create multi-hop sensor networks. These networks can be self-organizing and self-healing in a similar way to traditional sensor networks. The nominally higher power consumption of Bluetooth radios can be amortized over the higher network bandwidth and higher link reliability. As has been shown previously, the per-bit power consumption of Bluetooth radios can nominally be lower than for single frequency solutions. In addition, the overhead of forming network connections as required by the protocol can be negligible in semi static network configurations. The high tolerance of channel variations achieved by the frequency hopping schema enables stable links without the necessity for frequent reconnections. The tight clock synchronization at the MAC level allows piconets to operate very efficiently as the transmitter and receiver can be precisely turned on for their respective communication slots.

We are working on a number of optimizations in the network formation and maintenance algorithms that are expected to reduce connection overhead at least two-fold. In addition, we also expect to speed up the reliable data transmission protocol by a factor of 2-4, thereby further increasing network throughput and reducing transmission times. In addition, new platform software and hardware currently under development will reduce leakage and extend battery life by an expected factor of 2-3.

ACKNOWLEDGEMENT

We would like to thank the Intel Research Berkley team, in particular Wei Hong and Phil Buonadonna for their help in porting the Intel Mote Software to TinyOS 1.1. In addition, we would like to thank the folks at Intel Research in Oregon, namely Jasmeet Chhabra, Mark Yarvis, Lakshman Krishnamurthy and Nandakishore Kushalnagar for their cooperation on the reliable data transfer protocol and the mote deployment. Major thanks to Mick Flanigan and his team from Intel facilities for helping us with the deployment. Finally, we would like to thank Zeevo, Inc. for their technical support of this research project.

REFERENCES

[1] Crossbow, Inc., MICA2 mote, *http://www.xbow.com/Products/productsdetails.aspx?sid=72*.

[2] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless Sensor Networks for Habitat Monitoring", *ACM International Workshop on Wireless Sensor Networks and Applications 2002*.

[3] "Intel Mote", Intel Corporation Research, *http://www.intel.com/research/exploratory/motes.htm*.

[4] "TC2001P Product Brief", Zeevo, Inc., *http://www.zeevo.com/pdf_files/TC2001P_HCI_prodbrief_v1.2.pdf*.

[5] "Bluetooth Core Specification v1.1", Bluetooth Special Interest Group, *http://www.bluetooth.org/spec*.

[6] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System Architecture Directions for Networked Sensors", *Architectural Support for Programming Languages and Operating Systems 2000,* pages 93-104.

[7] A Transmission Control Scheme for Media Access in Sensor Networks, Alec Woo, David Culler, *Mobicom 2001*.

[8] "An Efficient Scatternet Formation Algorithm for Dynamic Environments", Godfrey Tan, Allen Miu, John Guttag, and Hari Balakrishnan, MIT Laboratory for Computer Science Cambridge, MA 02139.

[9] Crossbow, Inc., Stargate gateway, *http://www.xbow.com/Products/productsdetails.aspx?sid=8.*