Voice Streaming over a Sensor Network

Navraj Chohan, Wei Zhang
{nchohan, wei}@cs.ucsb.edu

Abstract

Sensor networks have become a major area of research for low power wireless embedded development. This paper takes the approach of proposing and implementing a system which goes against traditional low-duty sensor network applications. We propose and implement a prototype for the end-to-end streaming of voice communication over a multi-hop sensor network.

Keywords

Sensor Networks, Multi-hop Routing, Voice Streaming

1 Introduction

Current communication methods used in the military employ a top-down implementation based on satellite communication. The communication system's limitations are based on the nature of satellites. These limitations include: the need for line-of-sight with the end device, satellite latency, and lack of adaptation. In this paper we will propose and implement a bottom-up approach through the use of sensor nets. This approach, in contrast, is cost effective and can be employed on an ad-hoc basis.

The focus of this paper will lay on the trade offs between packet reception and quality of service (QoS) of a voice channel on a sensor network, as well as the limitations and workarounds of current sensor net hardware. The architecture will be built using two different types of protocols: a stream transmission protocol and multi-hop routing protocol. This paper is organized as follows: Section 2 will discuss the hardware platforms and the limitations associated with using such hardware. Section 3 will describe and detail the multi-hop sensor net application. Section 4 will describe the end-to-end application. Section 5 will give our results from testing our system. Section 6 will discuss related and future work, and we will conclude in Section 7.

2 Hardware Platforms

The sensor network is comprised of MICAz motes produced by Crossbow. These motes have the following attributes:

- 2.4 GHz, ZigBee compliant 802.15.4 radios
- 250kbps data rate radio
- Atmel Atmega128 microprocessor
- 512kB flash memory

The sensor network communicates via programming boards to the end-points. The programming boards are MIB510 which are attached to a mote via UART. Our endpoints are laptops which have both larger battery supplies and greater processing power. All computation heavy processes take place at the end-points.

3 Multi-hop Application

We developed our mote firmware based on TinyOS, which is the leading operating system in wireless embedded systems. It provides an event driven programming environment and a number of libraries written in NesC to support agile wireless sensor network development.

Our multi-hop protocol is based on the TinyOS standard routing libraries [8]. It is composed by the following modules:

MultiHopEngingM: This module assembles all other low level communication components. It accepts messages from both end-point applications and the network while doing necessary reception and forwarding.

MultiHopLEPSM: This module is the implementation of multi-hop algorithms. It maintains the multi-hop network information and provides link estimation and path selection (upstream and downstream) to *MultiHopEngingM*.

QueueSend: This is the standard TinyOS library for packets buffering and sending.

GeneticComm: This combined component provides a unique interface for framed packet transmission over the serial port and the radio.

3.1 Link estimation

The goal of link estimation is to design a component, the link reliability estimator, which is responsive, stable, reasonably accurate, simple with little computational requirement, and memory efficient. The estimation values are calculated based on the packets received/loosed status in a past time window T. Individual nodes use these estimations to build multi-hop routing structures.

3.2 UpStream

The task of upstreaming is simple: select a parent and forward. After the link estimation, we need to maintain a set of good neighbors in memory to select the parent node. Neighborhood management has three operations: insertion, eviction, and reinforcement. For each incoming packet which a neighbor analysis is performed, the source is considered for insertion or reinforcement. If the source is represented in the table, a reinforcement operation may be performed to keep it there. If the source is not present and the table is full, the node must decide whether to discard the information associated with the source or evict another node from the table.

Shortest Path (SP and SP(t)) algorithms are the conventional distance-vector approach where each node picks a minimum hop-count neighbor and sets its hop count to one greater than its parent. For SP a node is a neighbor if a packet is received from it. For SP(t) a node is a neighbor if its link quality exceeds threshold t. Thus, a neighbor is selected as the parent from those who have the least hop count towards the base and can provide the best link quality.

3.3 DownStream

The legacy multi-hop routing protocol provides only one way upstreaming communication (from remote node to base). This one way communication does not satisfy our requirements because the two endpoint applications must be able to respond each other. In order to provide two-way voice streaming, we have added downstream routing to the multi-hop protocol.

- Each node maintains a downstream routing table, which records the reverse path for upstream transmission.
- This table is updated when upstream messages pass through. A simple hash table is used to accommodate the high data rate of voice streaming; facilitating fast lookups and updates.
- In order to keep this table up to date, each node generates an upstream packet sent out every 5 seconds.

3.4 Application

The application at the mote level handles voice stream packets coming from the serial port. Packets are transmitted into the network using the interfaces provide by multi-hop routing library. Components on the multi-hop application level are:

- **VoiceTest**: This is the main entrance of our TinyOS software
- **UARTFramedPacket**: This is a TinyOS standard component for serial communication
  - HDLC packet framing and CRC checking are implemented here
  - Accepts input from the serial port byte by byte
  - Once a complete packet is received it will signal *VoiceTest* to make further process on the received packet

The base station will always treat the input from the serial port as a downstream packet; other nodes accept serial packets as an upstream message and send it back to the base.

4 End-to-End Application

Although it has been shown that voice encoding and decoding has been performed on the mote level in [3], our system takes that burden to the endpoints which have larger battery supplies and higher computation power. Our system looks at a sensor network much like a PC looks at the Internet. It is to provide end-to-end connectivity. Yet a deployed sensor network is very application specific and thus information from the router level can be forwarded to the end point to make ad-hoc decisions. The information can serve as a facilitator towards power efficiency to allow us to maximize the lifetime of the network. The streaming protocol is a lightweight RTP implementation [10]. This section will discuss our design decisions of the end point application as well as the trade-offs of making such decisions.

4.1 Application Requirements

The PC side software must have the following requirements met:

- Serial port communication
- Input and output queuing of messages
- CRC checking
- Streaming data messages
- Detection of packet loss
- Robustness to packet loss
- Encoding/Decoding

In addition to the requirements the software also has the following features either for debugging or reliability:

- Flow control
- Text messaging
- Reliable control messaging
- Statistics on packet loss and routing

The serial port communication with the sensor network is done through a MIB510 programming board at 57600 baud. The board serves us for both programming the motes as well as communication—communication up and down stream through the sensor network. The programming boards communicate via UART to connected motes. These connected motes serve as our gateway to the sensor network, where packets which come from the radio are forwarded to the UART, and packets from the UART are forwarded onto the radio.

Getting started with serial port communication proved difficult because our Keyspan USA-19HS USB-to-serial adapter showed read and write inconsistencies in the Cygwin windows environment. Several accounts of fellow sensor-net researchers also attested to the difficulty they had face with this adapter, as well as an account on the TinyOS FAQ [6]. After switching to a serial-to-serial connection or using a different adapter, such as the ATEN UC-232A USB-Serial converter, our serial port woes disappeared.

The protocol for the PC to mote communication is based on RFC 1662 [9]. There is a byte flag to mark the beginning of a packet, followed by a byte to specify the packet type (ack/no ack). An escape byte is used to differentiate which bytes are data and which bytes are special characters (i.e. the framing byte). The application must deal with framing and unframing the packets as well as doing CRC checking on the packets to ensure good data.

The application has three queues which are shared between four different threads, requiring mutex locks and conditional variables. The queues are:

- Outgoing queue
- Incoming data queue
- Incoming command queue

The four threads which use these queues are:

- Output to serial port thread
- Input from serial port and stdin thread
- Control thread
- Data thread

The mutex locks ensure that there are no deadlocks or race conditions between shared queues and the conditional variables are used to wake up waiting threads. A thread can be waiting on data from a queue, data from the serial port, or standard input from the user. Each queue holds unframed packets and thus the serial input and output queues must take care of framing and unframing the packets. Moreover, the input queue manager must be able to read several packets at a time and maintain synchronization with the framing. User commands are also dealt by the input thread and are converted to command messages. A flag is set in the message to distinguish between local commands and remote commands.

4.2 Voice Encoder

Our selection of a decoder had the requirements that it had to give us flexibility in bit-rate, do narrow-band encoding for voice (300Hz to 3300Hz), be robust in the face of packet loss, and ultimately give us good voice quality. We choose the Speex library for this need [1]. The benefits of using this library is that it is open source, free to use, has a variable bit-rate, is written in C, can deal with packet loss, and is used exclusively for voice (whereas using vorbis would have been excessive). The Speex encoder deals with 20ms frames of 160 bytes of raw PCM audio. The encoded form of the audio is 20 bytes per 20 ms frames (8kbps), which gave us the option to reasonably do two frames per packet while still being able to stay under our bottleneck of 57600 baud. Our packet structure for a voice packet had:

- 5 bytes of the TinyOS header
- 7 bytes for the multi-hop protocol
- 1 byte for message type
- 1 byte for sequence number
- 20 bytes for the encoded audio data
- 2 bytes for the CRC

It can be seen that there is much overhead per packet at a 8:10 ratio of overhead to audio data. Section 5 will discuss our options on how to lower this overhead ratio through the use of additional frames per packet.

The decoder functions can deal with packet loss by "guessing" what the intended voice was. In order to determine that there was packet loss we used one byte of our audio message as a sequence number. This one byte gave us the ability to have up to 256 packets before looping back to sequence number 0. If a packet comes in and is not the immediate next packet from the previously decoded packet, null packets are sent to the decoder for "guessing." Depending on the current sequence number count, there is a range of sequence number values that are considered to be out-of-order packets. Currently these packets are dropped and decoded packets are freed after being played, but in the future we intend to store the sound to playback without jitter and delay.

4.3 Program State

For end-to-end state synchronization, where the receiver knows it is the receiver and the sender knows it is the sender, reliable messaging must be built into the system. The multi-hop protocol does support bi-directional communication and thus makes acknowledgements and reliable messaging possible. Timers are needed to know when to retransmit a reliable command message. Messages of type FLOW (the types of messages are listed below) allows communication between the end points to relay the packet reception rate, which we consider the prime indicator of the QoS. The different types of messages include:

- FLOW and FLOW_ACK
- AUDIO_DATA
- TEXT_DATA
- SETUP_ROUTE and SETUP_ROUTE_ACK
- RECEIVER_READY and RECEIVER_READY_ACK
- TEAR_DOWN and TEAR_DOWN_ACK
- ROUTE_MSG (used for statistics on multi-hop network)
- USER_QUIT

With these message types a voice channel can be built and tore down.

Each endpoint deals with three different states: SENDER, RECEIVER, and IDLE. Encoding happens in SENDER, decoding in RECEIVER, and neither in IDLE mode. For debugging purposes, we allow for text message to be sent in any of these modes. When both end points are in IDLE the voice channel is up for grabs from either end point ("push-to-talk"). A typical setup from the sender side would include the following steps:

1. Send a SETUP_ROUTE message to determine if the network is ready
2. Wait for SETUP_ROUTE_ACK and resend SETUP_ROUTE if there is a timeout
3. Send a RECEIVER_READY message to tell the endpoint to switch to receiver mode
4. Wait for the RECEIVER_READY_ACK and resend RECEIVER_READY on timeout
5. Start streaming AUDIO_DATA packets
6. Send TEAR_DOWN message and go into idle mode
7. Wait for TEAR_DOWN_ACK and resend if there is a timeout

The receiver side must also be ready to resend acknowledgements in case they were dropped during transmission.

5 Experimental Observations and Analysis

As [4] and [5] has shown, the radio model is complex and proves unpredictable in a deployed system. Our deployment was in an indoor facility which exacerbated the predictability. Our definition of a good packet is one which passes the CRC check. We experienced no dropped packets. Our observations of the unstable reception rate in different circumstances affirmed these other accounts of unpredictability. Even in the same setting the reception rate would fluctuate at different time intervals. Furthermore asymmetric links in reception rate were not uncommon. With upstreaming receiving 90 percent good packets, we would experience 20 percent downstream. There were also counter-intuitive instances where a multi-hop path was providing better reception than a point-to-point connection.

This paper reports no data for hop count versus packet reception due to the randomness of our data. No model could be constructed of the packet loss. Large scale and long term experiments are required to find some significant regulations and reportable deployment policies, which we pose as future work.

5.1 Audio Quality

What we do present is a comparison of audio quality at different levels of packet loss. In order to achieve this accomplishment we did not use our system to attain the data due to the difficulty of achieving different levels of packet loss, rather we programmed a sound simulator which was based on packet reception. One caveat of our sound simulator is that it does not simulate jitter or delay in packet reception so it is an upper bound on audio quality. While it is difficult to use metrics on sound quality due to its subjectiveness, we will provide the reader with audio samples with different levels of packet loss for analysis. These audio samples can be found at:

www.cs.ucsb.edu/~nchohan/vosn/

In the future we will aim for MOS ratings of the audio quality.

Our own subjective analysis of the simulated sound shows that at about 50 percent packet loss or above is unacceptable. In actual results from our system we found that 75 percent packet reception was still unacceptable. This leads to the conclusion that first: we experience much jitter and delay in our real system which requires implementing a buffering scheme at the cost of overall delay, and second: packets which are dropped in burst significantly hurt sound quality.

To look further into the speech encoder and decoder, we see that each frame received is encoded and decoded relative to the previous and next frames. A frame is not independently encoded like one might find with the iLBC [2] encoder (yet iLBC has a higher bit-rate as a trade off). Thus an equal distribution of packet loss is much more acceptable than burst loss. In order to verify this conclusion, our simulator also encoded and decoded frames in multiples (two or more frames per packet). Our subjective results show that the sound did deteriorate even if the packet-loss was the same for multiple frames per packet. The explanation of this is due to the nature of the speex encoder. If a frame N is "guessed" and it is wrong, then N+1 will be "guessed" based on the previous miscalculation. The sound diverges from the actual spoken word. Audio samples of this kind can also be found on the previously mentioned site.

5.2 Power Awareness

The issue of power of this system has not been addressed yet. What we propose is an adaptive system which can switch on the fly based on feedback of packet reception. We aim to have two parameters: QoS_low and QoS_high. QoS_low is the packet reception that is unacceptable in sound quality, while QoS_high is sound quality that is crystal clear. Our solution to passing the QoS_low threshold is to send duplicate packets of the same frame. We double the send rate at the expense of power but our priorities are for a usable system and then power awareness. If the QoS_high threshold is passed, then we want to try to conserve power. This can come in the form of eliminating duplicate packets, or doing multiple frames per packet which would lower the ratio of overhead to audio data. Currently we are limited by TinyOS architecture of fixed packet sizes and are unable to change it on the fly, but our system allows for it at compile and mote programming time. If we did have the ability to do multiple frames per packet, there lies the trade off of lowered packet reception due to larger packet sizes, and the cost of burst frame drops.

5.3 Multi-hop Algorithm Limitations

Although we cannot draw any intuitive graphs to claim that we have exciting discoveries, we have observed some interesting network behaviors. First, when doing multi-hop test, were we removed the antennas of two nodes and place some antennaed nodes between them, our hope was to create multi-hop networks but sometimes these two nodes still choose to communicate directly with each other in the face of a bad reception rate. In fact this describes an important trade off between power efficient and link quality. For power efficiency issues, we are seeking parents from neighbors who have the least hop-counts while providing the best link quality. This is to reduce the number of hops to minimum and thus optimize the overall packets transmission and power consumption, but due to the instability of the wireless communication, such links are especially unreliable when doing high speed transmission. What if we increase the threshold of shortest algorithm? The number of hops will increase and kill the batteries rapidly. Therefore, the problem comes out: for high speed transmission, we require higher link quality than for slow sensor applications, but at the cost of high power usage.

6 Related and Future Work

Ardizzone el al. has shown a sensor network with video capabilities in surveillance and monitoring with startgates and MICAz mote while Smeaton and McHugh were able to do audio based surveillance using a sensor network. Mangharam et al. were able to do voice transmission in a sensor network by building into the motes the ability to encode and transmit voice data. Additionally, he did time synchronization with both a hardware solution and a software solution. Our implementation has extracted all the computationally heavy operations as well as synchronization to the endpoints. This allows for a simpler solution and longer battery life of the network.

Future work will include improving the voice quality of the transmission as well as attaining data of a multi-hop deployment. On multi-hop level, we need to find an adaptive protocol for such application, which select better links but reduce the redundant transmission by snooping. On the application level, voice quality can be improved with redundancy in the streaming. Yet in order to save power we may not have to send redundant packets for each set of voice frames. Determining out the rate of replication depending on the current packet loss rate needs many more experiments in both the field and simulation. Additionally, our system has a very clear distinction between the end-point and the sensor network. Mote applications are application specific and so blurring the line between the end-point application and the mote application can provide the better power awareness at the cost of the simplicity of abstraction. Lastly, our application must scale, while we transmitted voice over several hops, we wish to see our application on hundreds of nodes with multi-casting. Currently our routing layer does support many receivers and many senders, but it is untested and will prove to have new challenges.

7 Conclusion

We have proposed and implemented a system for data streaming capable of delivering voice from end-point to end-point over a sensor network. Our implementation has two distinct layers between the PC application and the mote application. While new challenges and limitations of our system are evident, problems from the current communication system in military systems are overcome. These problems include line-of-sight with satellites, large satellite latency, and a lack of adaptability. Furthermore, the cost of deploying a satellite is tremendous, while the cost of deploying a sensor network drops with the price of silicon and demand. With additional work and refinement of the software and with customizable endpoint hardware, a working, robust, and ultimately deployable sensor network is not far from the making.

8 References

[1] www.speex.org

[2] www.ilbcfreeware.org/

[3] R. Mangharam, A. Rowe, R. Rajkumar, and R. Suzuki. Voice over Sensor Networks. RTTSS 2006.

[4] A. Smeaton and M. McHugh. Toward Event Detection in an Audio-Based Sensor Network. Proceedings of the third ACM international workshop on Video surveillance & sensor networks, 2005.

[5] E. Ardizzone, M. L. Cascia, G. Lo Re, and M. Ortolani. An Integrated Architecture for Surveillance and Monitoring in an Archaeological Site. Proceedings of the third ACM international workshop on Video surveillance & sensor networks VSSN, 2005.

[6] www.tinyos.net

[7] A. Woo and D. Culler. A transmission control scheme for media access in sensor networks. In International Conference on Mobile Computing and Networking (MobiCom 2001), page 221, Rome, Italy, July 2001.

[8] A. Woo and D. Culler. Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks. In Proc. of the 1st ACM Conf. on Embedded Networked Sensor Systems, pages 14--27. Los Angeles, Nov 5-7 2003.

[9] W. Simpson, PPP in HDLC-like framing, RFC 1662, Internet Engineering Task Force (1994)

[10] H. Schulzrinne, S. Casner. R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. Network Working Group, January 1996. RFC 1889. http://www.ietf.org/rfc/rfc1889.txt