Non-Naive Flooding Mesh Network

James Adams (ja5g14, 26824744)

Advanced Computer Networks, University of Southampton. Source available at https://github.com/csjames/acn

***Abstract*—Flooding is used in many networks to build routing tables, for example in Open Shorting Path First routing. This is often the prerequisite to sending routed packets which are not received by all nodes in the network. By creating a mesh network which uses only flooding, this project endeavors to achieve a low computational and spatial overhead, leading to low node power usage and efficient broadcasting, as well as a simple implementation. This efficient flooding may see usefulness in home automation settings where messages are often relevant to all nodes in the network.**

I. Introduction

Low power mesh networks dominate wireless home automation, with protocols such as Z-Wave and Zigbee as well as more ubiquitous consumer protocols such as WiFi being used. Often the radio modules required to produce devices compatible with these networks are expensive due to licensing fees and high component cost.

As discussed by Schmid, T etal, as of 2010, the cost of mesh networks for home automation is four times the predicted price. By using low cost, off-the-shelf, one dollar radio modules and micro-controllers, we endeavor to build a very low cost mesh network, for simple home automation.

Additionally, the stack size of mesh network protocols such as Zigbee are often large and complicated, requiring higher performance, higher cost, micro controllers. This project aims to avoid the high complexity by not implementing a complicated standard such as IEEE 802.15.4, as well as minimizing code size through eliminating maintenance of routing tables and the associated logic. To avoid this, a wait-listen-repeat approach to routing packets is taken, with an intentional dependence on increased memory leading to increased network performance.

II. Background Research

A. Z-Wave

Z-Wave is a proprietary protocol, used primarily in home automation. Radio modules are produced exclusively by Sigma Designs.

The protocol describes a source-routed mesh network, meaning that the sender specifies the route the packet shall take; a route which is computed at install time[1]. This creates the disadvantage that when devices are moved, the reliability of the network is majorly compromised, and the paths must either be recomputed (requiring a manually initiated reconfiguration), or routed by attempting all other routes.

Z-Wave achieves very low resource requirements and hence requires lower performance devices with a far smaller flash size, when compared to Zigbee devices[3].

B. Zigbee

Zigbee, unlike Z-Wave is an open source network protocol capable of forming mesh and star networks. It uses 802.15.4 for Layer 1 (PHY) and Layer 2 (MAC), and implements its own network and application layer[7]. It operates on a range of different frequency bands depending on required application, and is far more resilient than Z-Wave.

The mesh network element of the protocol is self configuring, and self healing, hence nodes can be moved around, join and leave the network, without compromising reliability due to static routes[3].

Zigbee specifies different types of devices in its topology. Each network has a single coordinator which manages security and policy, and up to 65535 routers and end devices. Unfortunately a single coordinator adds a single point of failure to the network; a problem which is often dealt with through manufacturer specific protocol extensions.

C. Flooding

Flooding is a simplistic routing algorithm in which incoming packets are re-transmitted by each receiving router. If flooding is uncontrolled this causes packets to be re transmitted continuously, creating a broadcast storm, potentially rendering the network unusable[6]. Hence a controlled flooding approach is necessary for a stable mesh network.

Subramanian, J produced an exemplary flooding wireless mesh network, UFlood[5]. It was designed to distribute large over non-IP based protocols. This method uses a complex statistical model to determine link probability to its one hop neighbors, and, like ZWave, assumes nodes are stationary. Additionally the UFlood protocol also assumes few nodes can communicate directly, which is unlikely in a home automation setting.

This solution demonstrates many ideal flooding mesh network properties, however, pays little attention to power consumption and protocol overhead, instead focusing on high throughput.

III. RFM69 Radio Module and Development Board

The RFM69 is a 'one-dollar' radio transceiver. It can send raw or packetised data, and contains a hardware CRC engine for error detection, data whitening (as discussed in V), AES-128 and an on-board temperature sensor[4]. Encryption (though symmetric) along with 868MHz transceiver make it appropriate for in home mesh networking due to greater wall penetration and little need for high data rate. Furthermore, the module supports CSMA, making it appropriate for multi-radio

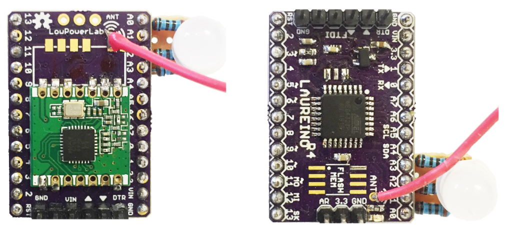

Figure 1. Moteino

This board contains the bare minimum for an ATmega 328p and RFM69. An optional regulator is soldered to the board but was bypassed for low power testing. In addition for power testing, the internal oscillator was used, and the board was down-clocked to 8Mhz.

networks. When compared to the NRF24L01[1] we observed a greater indoor and outdoor range, along with a lower power consumption when sleeping.

Unlike Zigbee radios, the RFM69 does not build on a predefined networking standard, and is instead a far simpler module, helping to keep cost and complexity to a minimum.

The Open-source hardware Moteino[2] design was used for development. Printed circuit-boards were fabricated using OSHpark[3] and assembled by my co-author and I. A UART arduino boot-loader was used so we had access to the standard arduino tools, and could program the modules quickly.

IV. Protocol Design and Implementation.

The following features were implemented based on the properties of the aforementioned protocols, as well as the features we desired for the system.

- A single node type to avoid master- router-sleepy architecture.
- Acknowledgments for reliability.
- Controlled flooding of packets as opposed to routing
- Message repeating to reach nodes out of range
- Broadcast messages for creating a system with shared state, e.g. multiple lights in a room being set to the same color.
- Low overhead and compiled size to allow for deployment on low cost micro-controllers.
- Radio Independent protocol, such that the protocol can be used with a range of low cost modules e.g. NRF24l01, RFM12B, NRF905.
- A more simple forwarding heuristic when compared to UFlood.

Unlike Z-Wave, protocol does not need to be aware of 'sleepy' nodes, as there is no route calculation, however flags in the library have been created so that space is not reserved for packet history.

[1]NRF24l01 http://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01

[2]Moteino: https://lowpowerlab.com/guide/moteino/

[3]OSHpark PCB: https://oshpark.com/

Figure 2 describes the packet structure that was produced.

0	1	2 to 3	4	5 to 31
Packet Type	Spare Field	UID HB	Origin	Payload (27 Byte)

Figure 2. Packet Structure

The payload length of the packet is configurable, up to a total packet size of 64 bytes. The radio module does support packets up to 255 bytes however reliability is drastically reduced. As encryption is done in 32 byte blocks, we opted for a 32 byte packet size to avoid the need for padding. The reader may notice that source and destination, along with CRC are not contained in the packet header. This is due to the fact these properties were handled by the Radio Module using register configuration. The spare field was used for debugging, and has been left unused for future development.

Packets were represented as a C++ structured object. By using struct inheritance we were able to seperate the header from the packet structure, for efficient storage of message information, and easy discarding of payload. Additional fields were defined inside this packet, which are used by the send-with-retry and repeating mechanism. Only the fields in figure 2 are marshalled into a byte array.

```
typedef struct {
    uint8_t origin;
    uint8_t source;
    uint8_t destination;
    uint8_t tries; // number of send attempts
    uint32_t lastTry; //time of last send attempt
    uint16_t uid;
    char packet_type;// A, B, U (ack, broadcast, unicast)
} rfheader_t;

typedef struct rfpacket_t : public rfheader_t {
    uint8_t data[MAX_PAYLOAD];
};
```

A. Unicast Packets (UCAST)

Experiments in a household showed that most places were reachable by simple unicast communications. Because of this we determined that this should be prioritized before any repeating occurred. Because of this the initial transport mechanism uses a send-wait-retry method to ensure message delivery. If the built in send with retry failed, the message to be sent was re en queued and sent after a small delay.

When a node hears a communication not for itself, it records the packet, and waits for an acknowledgment (ACK) to be seen. If an ACK is not seen within a certain amount of time , it re transmits the packet. Again attempting a unicast communication with the target node. This process is detailed in figure 3. The time spent waiting to retry is calculated using the simple heuristic below.

1) Repeating Heuristic:

A simple RSSI based heuristic is used for repeating. The delay before repeating a message is inversely proportional to the RSSI of the received packet is lower, the node that transmitted the message is likely further away/ less reachable, and hence is more appropriate for immediate repeating, due to the likelihood that the unicast communication failed. This also means that the delay is likely to be different for nodes repeating the same message, acting to avoid collisions. If one of these nodes receives the ACK for the message during this delay time, the packet is not repeated.

To avoid broadcast storms, nodes do not repeat a message they have already seen. This means that the size of packet memory must increase proportionally with network traffic, as otherwise the message seen buffer will overfill, incorrectly marking packets as not seen, and potentially causing broadcast storms. This does however create an easy way to increase the performance of the network, simply buy microprocessors with larger memories.

Figure 3 provides a visualization of this heuristics data flow. An example topology for each case is also provided.

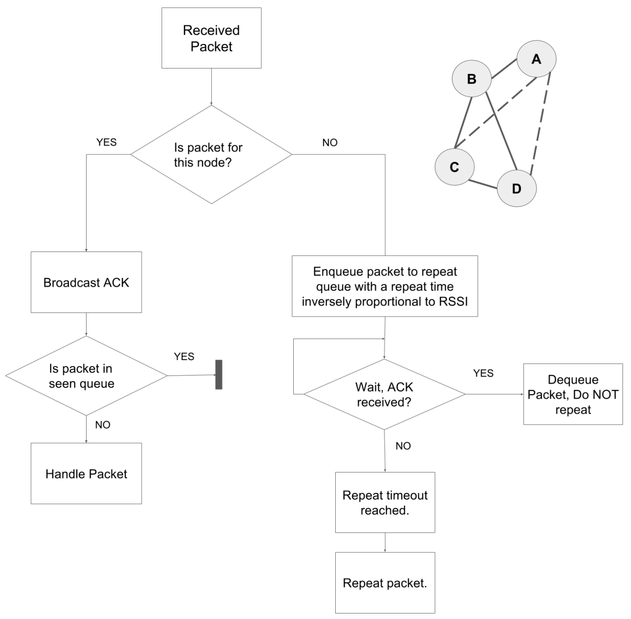

Figure 3. Hierarchy and Supported Topology.

Nodes with solid lines between them can communicate directly without repeating. Dotted lines show nodes which cannot directly communicate.

If the distance between nodes is taken to be the sole indicator of RSSI, a message from node D to node A would be repeated by node B with a higher priority than a message from node C to node A, as the transmitted packet would have a lower RSSI.

B. Broadcast Packet (BCAST)

Broadcast packets are sent on address 254, and nodes who receive a BCAST message repeat if they have not yet seen the message.

C. Acknowledgment Packet (ACK)

Acknowledgment packets are often smaller in size than the payload bearing data packets. Due to this being a demonstration project we opted to still use a payload as although it adds packet bloat, increasing the probability of message failure, it also enables the transmitting of debug information.

When using the RFM69, these ACK packets are broadcast in conjunction to the built in hardware ACKs. This is because RFM69 modules cannot receive hardware acknowledgments not sent to their address, and these ACKs can only be sent to the last received from address. By keeping ACK packets small, we increase the chance of a successful ACK message, hence ACKS contain the first 5 bytes of the header in figure figure 2.

V. Difficulties

Initially when testing the network, the payloads sent were empty, and a large packet size was used, resulting in a large sequence of zero bits in the radio packets. This was further exacerbated by the fact we initially weren't using encryption and hence most packets were 90% zeroes.

This caused a very poor reliability, with a 70% error rate. After much research it was discovered that the RFM69 has built in data whitening, utilizing linear feedback shift registers (LFSR) to avoid long sequences of the same bit value. When this feature was enabled, packet transmission through walls increased to above a 99% success rate when sent with two retries.

Data whitening does not completely solve the equal consecutive bits problem, for example, one could construct a packet which contains the same values as the LFSR, causing all zeroes to be transmitted[2]. Additionally there may be other factors at play reducing reliability, such as distance, and high background noise.

Some time was spent tweaking the default transmit properties of the Radio to get the best reliability inside our home. This involved changing retry time, increasing radio sensitivity, and reducing the bit rate.

VI. Visualization and Debugging

For debugging, an LPC11u68 Cortex M0+[4] board was used with the MBED framework. This module has a far larger memory, higher clock speed, and a very fast UART interface, making it suitable for observations of all messages sent through the network.

A. Architecture

For persistence and speed, Redis was used to store previous messages and discovered nodes. This was done due to the many easily accessible data structures available, for example circular queues, allowing the easy storing of the N most recent messages.

Web Sockets were used for communication between the server and web interface. This allowed node states to be dynamically updated as messages came though.

VII. Visualization

Cytoscape.js, a graph visualisation library[5] was used to display the mesh network.

[4]LPC11u68 Board: http://uk.farnell.com/nxp/om13058/lpcxpresso-board-lpc11u68-mcu/dp/2399860

[5]Cytoscape.js - A graph visualization library: http://js.cytoscape.org/

References

[1] *Z-Wave Technical Basics*, 2011.
[2] Grant Christiansen. Data whitening and random tx mode. *TI Design Notes*, 2010.
[3] Lou Frenzel. What's the difference between zigbee and z-wave? web, 2012.
[4] HOPERF. *RFM69HW ISM TRANSCEIVER MODULE V1.3*, 1.3 edition.
[5] Jayashree Subramanian. *Efficient Flooding for Wireless Mesh Networks*. PhD thesis, MASSACHUSETTS INSTITUTE OF TECHNOLOGY, 2012.
[6] Andrew Tanenbaum. *Computer Networks*. Prentice Hall Professional Technical Reference, 4th edition, 2002.
[7] Ankur Tomar. Introduction to zibgbee technology, 07 2011.