Dumb Distributed Network

Quinn Gieseke (V00884671), CSC 466

April 12, 2021

1 Introduction

Historically standard networking protocols have relied on the development of an underpinning network. Initially this consisted of dedicated phone lines and operated on a per-connection basis. In an effort to reduce the overall cost of the network, packet switching was adopted in order to allow many connections to share one link. However, there are many use cases where building a dedicated network would take to much time, or too many resources to be economically viable. Additionally, fixed network topologies can introduce many points of failure and support protocols that do not adapt well to shifting link states, or link loads. To overcome these issues, Mesh Networking has been introduced, a network paradigm under which the network must self-organizes, configures, manages loads, and remains resilient in the face of node failure.

The mechanism of choice to meet these goals was wireless communication. Radio allowed for wireless mesh networks to be set up instantaneously even in hostile environments, and the overall resilience lead to much military interest in developing wireless mesh networks. Today we see similar networks cropping up where similar needs arise. Recently protesters used FireChat, a messaging app utilizing solely local bluetooth and Wi-Fi direct to deliver messages, even in the face of official telephone networks being taken down. Other attempts include Zigbee, a proprietary mesh network intended to create a cheap local wireless network, able to cover more distance and be less expensive than traditional Wi-Fi approaches [1]. Such networks would be more suitable to home automation and related IOT devices, as they would provide greater connectivity and more reliable use of network-connected smart devices.

Additionally, extensions to existing IP protocols have been created specifically for the use case of wireless mesh networks, with one open protocol being 6LoWPAN, an acronym for IPv6 over Low-power wireless Personal Area Networks [2]. The goals of such projects are to allow for low powered devices to utilize the same protocols as more traditional fixed network devices. They accomplish this by reducing the required data transfer rate to maintain a stable connection, as mesh networks are often comprised of large amounts of lower powered radios that are unable to match the expected speeds of a traditional copper or fibre packet switched network.

2 Problem Statement

All these protocols, however, share a common weakness. Since a necessity of mesh networks is to be able to pass information through intermediate nodes, each peer node must constantly be listening for incoming data, in order to possibly relay it to one of its peer nodes. This leads to high power use, even in "idle" states, meaning that nodes must either have steady access to power, or be limited in lifetime while supplied power via battery. Additionally, enforcing the symmetry of receiving and broadcasting for every node means that for a network to perform well, every node must have a radio receiver and transmitter capable of matching the total data throughput for the node, at risk of bottlenecking the entire network otherwise. Each node must also be smart enough to decode, decipher, and forward packets on, requiring at least a basic microcontroller that increases both complexity and cost.

2.1 Functions

I have designed my implementation of a dumb distributed network to meet the following conditions:

- Each dumb node should be made as simple as possible
- The number of dumb nodes should far outweigh the number of smart nodes
- Sensor values should be replicated as quickly as possible across the network
- Communication between all nodes should have as little overhead as possible

- Connections should be entirely stateless, no fixed network topology is assumed
- Data becomes stale, and stops being replicated, reliably
- The network should be highly scalable

Additionally, I have made the following assumptions about the network:

- Smart nodes only care about the most recent value produced by a given dumb node
- Node communication status can be determined solely through the distance between nodes
- All nodes stay online throughout the simulation, although their location can allow them to be out of range of all other nodes
- The sole purpose of a smart node is to act as a relay for dumb node data

2.2 Use Case

Going back to the military roots of wireless mesh networks, such a network topology would allow a large number of dumb sensors to be embedded within a battlefield where they could operate for an extended period of time without maintenance, broadcasting useful information to soldiers or vehicles carrying more powerful and less populous smart nodes that form a more traditional mesh network layer that exchanges the data gathered from their local dumb nodes in a more traditional manner over a larger area.

3 Dumb Distributed Network Description

The general solution to this problem is to create a second class of node in the wireless mesh network, a "Dumb node" whose purpose is to be as simple as possible while retaining the ability to participate in the mesh network. To do this, I introduce the concept of a broadcast only communication. By disallowing any response packets, the dumb nodes are capable of shutting off during idle, only consuming power to generate data and broadcast it. This will allow them to have increased longevity while operating completely on batteries, as well as drastically reducing their cost. In this way, the possible data loss through link

failure could be mitigated, not through rebroadcasts and acknowledgements, but rather through dumb node replication and a denser network, while maintaining or reducing overall network costs.

3.1 The Dumb Node

While limiting the processing power and power consumption of the dumb nodes precludes them from taking an active role in mesh networking, they still retain a use. One such use case that I utilized in my example implementation was as a simple sensor. In this case, each dumb node would be solely responsible for waking up at a set time interval, sampling its sensor value, and broadcasting that packet. However, some overhead is required to maintain the usefulness of the data. One, a unique node ID must be assigned to the sensor, and broadcast with every packet. This allows the network to remain stateless and for multiple smart nodes to corroborate the values generated by a single dumb node. Two, a monotonic counter is included as a logical clock, to be incremented at every broadcast and whose value should be included in the packet. This allows a for a per-sensor ordering of data, and the ability to determine an approximate age for the sensor value.

3.2 The Smart Node

Smart nodes are assumed to be orders of magnitude more powerful and much more reliable than dumb nodes. However, their main purpose in my proposed network is to collect and rebroadcast information generated by the sensor nodes to peer smart nodes outside the limited broadcast range of local dumb nodes. In order to achieve this, and because of the lack of limitations, smart nodes more closely reflect traditional nodes in a wireless mesh network. To enable the use of dumb nodes, my implementation keeps track of a few additional parameters. Firstly, an additional logical clock is kept at each smart node. This clock is incremented every time the node receives a broadcast containing updated information, either from another smart node or from a sensor node. This allows for an approximate measuring of the age of data in the network. Secondly, an array is kept of all the most recent sensor values that a given smart node is aware of, that includes the actual data, the sensor clock value, and an approximate local node clock value. Thirdly, the state of every sensor node in the network must be tracked. If the most recent local node clock value for a sensor is less than the current clock value by some tunable threshold, the sensor and its corresponding stored data should be considered stale, and should not be shared as it is sufficiently out of date.

Finally, the smart node is responsible for rebroadcasting updated sensor values to its peers. To achieve this I decided on regular heartbeats that include all non-stale sensor data, ensuring that even as network topologies shift, any nodes within range will be informed of that nodes most current state. However, one complication is that local smart node clocks will not agree, so that value cannot be transmitted as is. Instead, an offset is transmitted, calculated as $T_{current} - T_{stored}$, or the number of ticks that have passed since the data was received. Then, when this heartbeat is parsed at a receiving smart node, if the given sensor tick is higher than the stored sensor tick, the offset is subtracted from that smart nodes current clock. In this way, we ensure that data becomes stale at a roughly set rate, even while it is being replicated across the network.

4 My Implementation

To test the viability of my network design, I implemented a basic version of the network in C. In this implementation, each node is simulated as an independent process running on a single machine. Each process is initialized using a unique ID by a parent simulator that controls the number and communication between all nodes.

4.1 The Simulator

By far the most complicated part of testing my network, the simulator utilizes named pipes on a standard UNIX filesystem to manage the communication between all sensors and smart nodes. Each sensor has a single unique output pipe, and every smart node has two unique inputs, one for sensors and one for peer nodes, and a unique output pipe. In my simulator I created a 2D plane of a set size, and regularly randomized the position of all nodes in the network on that plane. Then I used set distance thresholds to determine if certain nodes could "hear" each other, routing the sensor and smart node output pipes into the proper smart node inputs pipes, replicating data as necessary. This allowed me to simulate the changing nature of a wireless mesh network while including cases where the network would become partitioned then reconnected, nodes could become isolated, and previously distant nodes become close. In this way, I was able to ensure that the network operated smoothly even during shifting topologies and observe how sensor data could be replicated across the network.

4.2 Sensor Node

My implementation for each sensor nodes consisted mainly of a single loop that would write a packet to its output loop, then sleep for a set amount of time. Each sensor would be terminated by an asynchronous interrupt passed from the simulator.

4.3 Smart Node

The implementation for the smart nodes was a little more complicated. To ensure maximum speed I made each node multi-threaded, with one thread dedicated to each I/O operation. Each node keeps track of sensor data using a fixed array, and of sensor status using a bitmap. Heartbeat updates were triggered roughly every second, again utilizing a delay loop inside the dedicated thread. The two additional read threads, one for sensor reading, and one for reading heartbeats from peer smart nodes. These would each process all data in their respective pipes and then block until additional data was written. All threads were terminated by asynchronous interrupt passed from the simulator.

5 Results

My network simulation worked as intended, meeting all of the conditions I wanted it to. The code for the dumb sensor nodes was less than 50 lines of unoptimized C code, the communications protocol between sensors and nodes only had 2 bytes of overhead data, and no fixed network topology was ever assumed or needed. The network simulator showed that smart nodes were able to share information gathered from dumb nodes, even in the face of changing network topologies and network partitions. While sometimes it would be impossible to keep every smart node up to date, because of isolated sensors or nodes, the network was capable of fully operating during my scale tests. Unfortunately, I was not able to adequately test the scalability of the system, as I was unable to simulate sufficiently large networks on my home PC.

However, measuring the capability of a single smart node was measured, assuming each sensor node was broadcasting a value approximately once every second, at a little over 4200 reads per second. This more than meets my goal of the quantity of dumb nodes far outweighing the quantity of smart nodes.

Due to this massive difference, however, I was not able to adequately measure the scalability of the system

as fully saturating more than 4 smart nodes was impossible on the hardware I had available, even after modifying each sensor process to report 1000 updates per second to reduce scheduling overhead. Ultimately the limiting factor was the strain of the simulator, not the smart node, so I still believe this counts as the system being fully scalable to a reasonable degree.

6 Further Steps

This implementation of a dumb node enhanced network simply experimentally proves its viability as an addition to and extension of existing mesh network topologies. Several simple modifications could be made to make it much more usable, for one increasing the data transmission of the sensor nodes from a single integer value to a small suite of useful information. Another increase would be allowing the smart nodes to have much more active participation in the network, gathering and calculating inferences on the data they have access to, or storing values over time for more nuanced applications. Additionally, there is no reason that smart nodes could act partially as sensors themself, possibly gathering more complicated data points and spreading those throughout the network themselves.

Further validation is also needed to fully prove the applicability of a dumb node enhanced network. Simulation on more powerful hardware could provide better information as to the upper limits of such an approach, and a small scale implementation over true wireless communication would provide better insights into how a network would operate in the real world.

Other extensions include encryption of communication between nodes to prevent malicious actors from either gathering important information from the network, or from poisoning the network with false data.

7 URL

Source code and all related documentation, including this report, available at:

https://github.com/QGieseke/CSC466_Distributed_Network

References

[1] https://zigbeealliance.org/

[2] Design and Application Spaces for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs) RFC 6568, April 2012