Table of Contents

SARATOGA: Simple and Reliable Internet over Long-Range Radio

Kyle Birkeland, Kris Brown, Joseph Connor, Parker Diamond,
Tyler Marshall, Sara Mousavi, Jared M. Smith

University of Tennessee, Knoxville
`saratoga@utk.edu`

1 Introduction

In this work, we present SARATOGA, a solution to connecting off-the-grid locations with the rest of the Internet by building on top of existing HAM radio architectures in order to provide scalability, reliability, and low latency while focusing on keeping the solution *simple, generic, and limited*. Our project team, though large, worked effectively to submit a viable solution to the Mozilla WINS challenge and support the solution with two simulations, one in the OmNet++ simulation framework, and functioning with a custom live simulation framework working with tens of thousands of actual site data. In our live evaluation, we can suffer from 10% bit-level error and fully recover the original radio transmissions. Furthermore, the high noise levels that cause greater than 20% bit-error after 10% recovery are unrealistic in practice. We also show that greater than 99% of requests are serviced with less than half a second latency for 25 requests per second for varying amounts of clients.

In the rest of this report, we present our submission to Mozilla WINS in Section 2, we describe our OMNeT++ implementation in Section 4 and our live implementation in Section 5, the individual team member contributions in Section 6, and conclude with future work in Section 7.1.

2 WINS

In the following sections, we present our submission to the Mozilla WINS request for proposals for the first design phase. [5]

2.1 Challenge

Off-The-Grid Internet

2.2 Solution name

SARATOGA: Simple and Reliable Internet over Long-Range Radio

2.3 Public Documentation

- Github Organization: `https://github.com/wins-saratoga`
- Public website: `https://wins-saratoga.github.io/SARATOGA/`
- Live Simulation: `https://github.com/WINS-SARATOGA/LiveSimulation`
- OmNet++ Simultation: `https://github.com/WINS-SARATOGA/rbn`

2.4 Problem Statement

Connecting geographic regions that have faced natural disasters or otherwise adverse circumstances continually plagues developed countries, like Puerto Rico after Hurricane Maria. Existing solutions attempt to reconnect regions via store-and-forward systems, while others attempt to drop portable networking devices into a region from above. However, these systems scale poorly and fail to connect broad regions that are isolated from the broader Internet without massive infrastructure overhead.

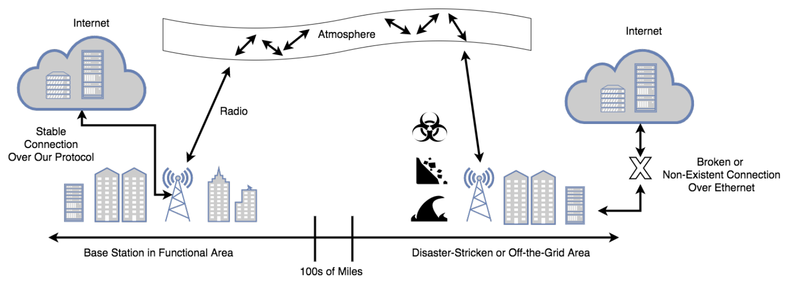

Internet
Atmosphere
Internet
Radio
Stable Connection Over Our Protocol
Broken or Non-Existent Connection Over Ethernet
Base Station in Functional Area
Disaster-Stricken or Off-the-Grid Area
100s of Miles

Fig. 1: SARATOGA deployed in a real-world environment

2.5 Solution Statement

SARATOGA aims to connect large, geographically distant and unconnected areas to the current Internet infrastructure through the use of long-distance radio network bridges combined with a protocol based around data compression, caching, and forward error correction. By placing radio endpoints at an area that is well-connected to the Internet and at another disconnected area, the endpoint at the disconnected area can relay information to the other, which will pass it to the Internet at large.

2.6 Users

Since SARATOGA supports conventional network communications, it services general internet users with typical client devices, like laptops and mobile phones. SARATOGA is designed to benefit users with minimal existing connectivity without the overhead of fiber/wire infrastructure. For example, immediately following a natural disaster, emergency services will be able to set up these nodes rapidly, allowing users with devices to connect and communicate. As the reconstruction of communications infrastructure is costly and time consuming, SARATOGA aims to provide service in the interval in which users can make full use of the Internet.

SARATOGA seeks to fulfill users connectivity needs by being portable, simple, and cost-effective. Nodes have low power requirements and can be easily transported. These nodes can also be distributed in adverse environments or generally unconnected areas to provide Internet-connectivity. Stronger nodes can be established where there is more power supply with multiple low-cost nodes scattered in a network to connect isolated groups. The use of compression and caching maximizes bandwidth and throughput allowing service to the maximum number of users possible.

2.7 Technical Feasibility

SARATOGA uses two long-distance radios to establish a network connection between users in a disconnected area and an endpoint in a connected area that will relay traffic to the high-speed Internet available in modernized regions. All amateur radio endpoints have two interfaces – one for sending/receiving conventional network data (802.11 WiFi, Ethernet) and another for sending/receiving data on frequencies used for long-distance transmissions. The endpoints will translate data from the conventional network interfaces into data on long-distance radio frequencies and send that data to each other, allowing geographically distant areas to communicate without the infrastructure overhead of large transcontinental cables.

The connected area endpoint will operate as an Internet service provider for the disconnected area(s). When the endpoint in the connected area receives data over the long-distance frequency, it will translate this data to conventional network data. The endpoint will route this data and send the response back across the long-distance radio bridge to the disconnected area. The disconnected area endpoint will act as a typical router/switch on a computer network – it will accept connections from individual devices and translate and send their data over the long-distance radio frequency to the corresponding connected endpoint. A working prototype will ideally support point-to-point communication over 10 kilometers at upwards of 1 megabits per second.

There are several hurdles to constructing a network bridge over amateur radio frequencies:

- Bandwidth, Distance, and Power Trade-Off
- Low Signal to Noise Ratio
- High Latency
- Regulations on Radio Frequencies and Transmissions

each of which is discussed in the following sections.

2.7.1 Bandwidth, Throughput, and Distance To alleviate the bandwidth restrictions and noise in radio transmissions, the project implements a network protocol on top of this infrastructure that uses data compression, forward error correction, and caching. When an endpoint sends data over the long-distance frequency, it will first compress the data using Gzip's data compression algorithm. Other common techniques for reducing traffic on the network bridge also scale well with the proposed protocol. Common network traffic can be cached at the disconnected endpoints in order to reduce the traffic on the radio bridge. Additionally, different connections can be multiplexed across different channels, increasing the throughput of an individual bridge. Directional and omni-directional antennas on the endpoints can also reduce the overlap of transmissions sent into different regions while increasing the power/sensitivity of senders/receivers can improve the distance over transmissions.

2.7.2 Low Signal to Noise Ratio To counteract the noise in long-distance transmissions, the compressed data will be encoded with a turbo code, a form of forward error correction which can be used to correct data errors even if they affect relatively large amounts of the data. This will reduce retransmission requests on protocols like TCP, which could further encumber the radio bridge. Compression and strong forward error correction provide the most efficient transmission of information, when information must be retrieved from outside the disconnected area.

2.7.3 High Latency Since SARATOGA operates over long distances through radio waves, there is an inherent latency overhead since the transmissions must travel upwards of 10 kilometers. For static web content, this is a non-issue since the request and response are sent only once at a time, typically in lock-step. Dynamic content, however, requires a persistent connection. The difficulty therein lies with network protocols like TCP that have a built-in timeout which will close a network connection if a response is not received within a given time window. To counteract these problems, SARATOGA's protocol uses aggressive TCP acknowledgements at the radio bridge endpoints. When a radio bridge endpoint receives a TCP segment, it will automatically send an acknowledgement in order to keep the connection alive and forward the data to the recipient. If the recipient does not send an acknowledgement back to the endpoint, then the packet is assumed lost and the connection subsequently closed. Otherwise, the acknowledgement is simply discarded.

2.7.4 Regulations on Radio Frequencies and Transmissions Some countries place restrictions on the type of data that can be transmitted over certain radio frequencies. The United States of America's Federal Communication Commission (FCC), for instance, restricts the use of encryption on some amateur radio bands as part of Title 47. There, however, exist radio frequencies where this ruling does not apply that can be used for point-to-point encrypted transmissions. Some alternatives include WiFi over Long-Distance (WiLD) and the

ISM (Industrial, Scientific, and Medical) band for long-distance encrypted radio transmissions. For countries without such regulations, more optimal frequencies can be used.

2.7.5 Status of Current Work Currently, SARATOGA is focusing on the implementation of the network protocol to support efficient transmission of data. At this time, the project has no hardware prototypes but has software support for forward error correction of network data via open-source libraries. Currently the work focuses on how the project differs from its predecessors, specifically in the inclusion of compression, forward error correction, and caching. Previous projects using long-distance radio to facilitate Internet connectivity in off-the-grid locations have had moderate success, which can be furthered by the changes we propose.

2.8 Community/Location

SARATOGA is intended for any environment that is (1) disconnected from the Internet and (2) does not currently have the resources or ability to connect directly to the public Internet. Using long-range radio coupled with caching, channel multiplexing, compression, and forward error correction, SARATOGA can supply connectivity to any region with the ability to power a small router-like device with the ability to send out signals similar to HAM radio. Since these devices can be powered via solar power or generators, endpoints can be setup in disconnected locations and provide a way to connect back to the Internet at the connected endpoint, while providing the reliability needed to transmit messages without data loss. Limitations do exist, however. Long-range radio becomes unfeasible without massive amounts of power at the connected endpoint when attempting to send more than several hundred kilometers; therefore, SARATOGA is limited to providing connectivity to regions several hundred kilometers away.

2.9 Differentiation

Several past projects have connected disconnected regions with varying amounts of success. The Latin American Networking School Foundation in Venezuela, for example, created a 279 kilometer WiFi connection. The TIER project at the University of California, Berkeley created a similar long-range WiFi network and successfully supported video conferences between patients and doctors that were geographically distant. More recently, Google's Project Loon provided Internet to Puerto Rico after Hurricane Maria.

These projects have had moderate success, which SARATOGA seeks to improve upon with a novel combination of existing technologies alongside a different network topology. Unlike the aforementioned projects, SARATOGA uses a point-to-point bridge between regions. Rather than place the radio nodes far from the users, like in the previous projects, this project keeps the endpoints close the users to minimize latency and error. This results in only one streamlined high-latency, moderately error-prone connection across the two regions. This project should support higher bandwidth operations than previous projects as well by compressing and caching data and reduce re-transmission with forward error correction.

2.10 Affordability

SARATOGA aims to re-purpose existing wireless communication devices for longer-range communication, in combination with software techniques to overcome existing limitations. Therefore, SARATOGA uses only low-cost, off-the-shelf hardware, which minimizes the cost for a single node. A single node requires, at a minimum, only a consumer router capable of running the necessary software such as the Linksys WRT5GL, which is available online for about $50 as of November 2017 (https://www.amazon.com/Linksys-WRT54GL-Wireless-G-Broadband-Router/dp/B000BTL0OA). This router can run open-source firmware like OpenWRT, which supports modification of network protocols. Antennae can substantially increase range and cost between $100 up to thousands of dollars; using a small antenna, we estimate ranges of up to 10 miles with good line of sight. Amplifiers increase range at an additional hardware cost, and amateur radio amplifiers also come at a wide range of power levels and cost, from the low hundreds of dollars into the thousands. In total, we estimate that a minimal functioning node could be built for less than a $100, while an ultra-long-range node could cost several thousand dollars.

2.11 Social Impact

This project provides a hot-plug solution to Internet connectivity in disconnected communities. First, the simplicity of being able to plug in two devices and have the system work makes the solution approachable, even by people who are not technically literate. This means that even extremely small communities can gain connectivity by setting up a few nodes themselves without having to pay expensive network technicians. Second, application layer programs using the network will see almost no change in network behavior since the endpoints communicate with client devices with conventional network protocols.

The SARATOGA nodes are low-power and can run on low-cost commercial-off-the-shelf hardware, which is ideal for growing communities that do not have the stability or money to bring fiber/wire infrastructure to their location. Moreover, these nodes can be routed into a mesh network for increased redundancy of cached data or multiplexed for increased throughput. All of these features make this solution optimal for disaster struck areas, or simply growing communities, to quickly connect to the Internet and scale their connections as they grow until they can attain a more stable and efficient hardware solution.

2.12 Scalability

Since SARATOGA only needs to alter conventional network traffic at one point during the transmission process, it scales with current network devices. The endpoints support 802.11 and Ethernet connections with clients, so devices available in connected regions operate without any issues. Included caching, data compression, and forward error correction reduce the bandwidth constraints on the long-distance radio connection between disconnected/connected regions, so links should support moderate amounts of users for each disconnected location. The support for normal devices (i.e. phones/laptops), allows users in the disconnected region to join the connected world with existing technology. This generality allows transferability of new developments between these regions and allow the disconnected regions to develop technologically alongside the connected world. Moreover, additional channels/substations can be added to the network topology to support more concurrent users as the community grows. If the individual P2P links become overwhelmed, the wireless nature of the topology allows transitioning to other topologies, like a mesh network, for congestion-aware routing. In an extreme case, a dense enough mesh of this protocol could implement a high-latency border-gateway protocol (BGP), much like the modern Internet.

2.13 Openness

We have documented our project through public documentation on GitHub, including a public facing website consisting of information from the application, and source code in our ongoing implementation of SARATOGA. Further work remains to be done to document our software and evaluation in conference publications to be submitted to networking conferences such as ICDCS, ICNP, and SIGCOMM.

2.14 Previous Awards

None.

2.15 Which of the following Internet health issue areas will your solution address?

- **Open Innovation**: By making SARATOGA open-source, this project hopes to promote innovation in alternative internet technologies.
- **Digital Inclusion**: This project aims to include those not in the Internet community by giving them access to a reliable and moderate speed Internet connection
- **Decentralization**: The use of multiple, lightweight radio endpoints allows small organizations or even individuals to the support their own Internet connections.
- **Privacy and Security**: The agile nature of radio Internet is that it can route around malicious nodes and quickly adapt to a changing cybersecurity environment without the overhead of infrastructure.

3 Off-The-Grid Internet Challenge

3.1 Portability

A standard SARATOGA node can be incredibly lightweight, power efficient, and portable. Each Linksys router is compact and weighs under a pound. As the power demands are light for the router, relatively small batteries, under two pounds, can be used to power the endpoint for up to ten hours. Therefore, it is reasonable to assume that a fully functioning node could weigh under 5 pounds and would be very easy to place in any location regardless of the transportation difficulties. Naturally, more powerful endpoints would require significantly added weight requirements due to amplifiers, batteries, solar panels, and antennas. However, the added equipment could be easily distributed among a few individuals and transported with ease.

3.2 Power

The home routers used in SARATOGA are expected to consume on the order of a few watts of power. For example, the Linksys WRT5G data sheet specifies 0.5 A of current at 12 V, or 6 watts [3]. Other comparable home routers are expected to have a similar power draw. This is the minimum required to run a small, short-range node; however, depending on the required range and bandwidth additional power may be needed to run an amplifier. Amplifiers used in amateur radio use power measured in the hundreds of watts.

A small node could therefore function for less than 10 watts, while a much larger node could draw up to 1000 watts. As with hardware costs, the exact power usage will vary significantly depending on the exact choice of hardware for the node. The hardware will depend on the deployment conditions, and again, this allows the user to readily scale power consumption with the range desired in a particular deployment scenario.

3.3 Applications

An estimate of the bandwidth provided by long-range radios is 1 megabit per second. Without compressing data packets, messengers such as Zangi, WhatsApp, and many other communication applications with low data usage will be able to function at nearly full capacity using the proposed off-the-grid internet. Other than messengers, streaming applications such as Netflix and YouTube can also be used; however, watching video over a 1Mb/s means that either the video will be low quality (like watching VHS movies) or a large amount of time will be spent buffering the video at high quality.

In order to decrease traffic over the radio bridges, the data is compressed with Gzip's DEFLATE algorithm. Compression effectiveness varies by data, but assume that the algorithm can reduce the size of packets by 60%. That means more than twice as much data can be sent and more data can fit into the cache. Due to the high latency of traffic, however, real-time applications will not work well over this network topology. SARATOGA is intended to provide low-overhead connectivity for minimally interactive web function with minimal infrastructure – ideal for post-disaster and unstable environments.

4 OMNet++ Implementation

We used OMNeT++ to simulate a radio based network (RBN). In our simulation, the RBN is a network that simulates a group of users who were disconnected from the Internet but have been connected using SARATOGA nodes. The nodes in the network that refer to the users we call "people" nodes. The SARATOGA nodes are either called "converter" or "radio" nodes. Lastly, the nodes that represent the rest of the Internet, we call "grid" nodes. The people can communicate with each other using wifi links. However, when they are going to communicate with the rest of the Internet, the packets must be sent through the radio nodes. To do this transmission, the converters have been placed between the people and the radios and between radios and the grid. They act as a barrier between regular wifi connections and radio links. They are responsible for the data transformation: translating between radio and wifi, compressing packets, forward error correction, and TCP caching.

4.1 Building an OMNeT++ project

OMNeT++ simulations are comprised of these components:

- .msg files – these files are where messages are defined
- .ned files – these files are where the structure of the network is created, including the topology as well as node properties (gates, connections, etc)
- .ini files – these files define parameters of the nodes in the network
- .cc or .h files – the .h files include the module dependencies and define structures. The functionalities of modules are defined in the .cc files.

First, .msg files are optional because there is a basic, default message that can be sent. More complicated messages/packets can be specified in .msg files. Messages can have properties such as a size, properties that simulate bit errors, or properties that indicate the next hop node on the path to its destination. The .msg files are translated into C++ code using the opp_msgc. compiler. All C++ sources are compiled and linked with the simulation kernel and user interface library to form a simulation executable. The .ned files are dynamically loaded from their original text forms at the simulation run time.

The topology of the network is defined in .ned files. The nodes are specified along with how they are connected to each other and how they interact with each other. Properties of each node and properties of the connections between them are also specified in the .ned file. To facilitate the modeling of communication networks, connections in OMNeT++ can be tweaked to model physical links. Connections support parameters such as data rate, propagation delay, bit errors, and packet error rate, all of which may be enabled/disabled. These parameters and the underlying communication are encapsulated into channel objects. An exception to putting the topology and channel properties in a .ned file would be the dynamic network builder. Using the dynamic builder, one can use one or more files to specify a network and the properties of nodes in the network. The dynamic builder can then create an OMNeT++ network based on the information in the file(s). This is especially convenient for creating large networks because the network files can be generated by an outside program/script. Specifying large networks manually could be quite cumbersome.

The .ini file allows one to specify properties of components in the network. The .ini file allows one to quickly tweak properties of the network (such as transfer rates or packet generation frequency) and rerun the simulation to compare results. It also allows properties to be specified from a random distribution (e.g. specifying the delay between nodes as an exponentially distributed random variable). The .ini file also allows the user to specify which network topology to simulate. Multiple topologies can be within the same .ini file to allow one to test custom network nodes in different network topologies.

4.2 Simulation Design and Assumptions

4.2.1 Node Structure Each node in our network is composed of 3 basic layers: an application layer, a routing layer, and a queuing layer. The application layer is responsible for generating packets to be sent through the network. The application layer is also responsible for consuming packets when they eventually arrive at their destination. The routing layer accepts packets from the application layer, uses its routing table to determine the port of the next hop, and finally sends the packet to the output queue attached for that port. Lastly, the queuing layer is responsible for buffering between the routing and physical layers. Each queue holds on to packets that are to be sent out through a specific port. These connections between the 3 layers also work in reverse order. The queuing layer holds onto incoming packets from a specific port. The queuing layer sends the packets to the routing layer. The routing layer checks if the node is the destination. If the node is the packet's destination, the routing layer forwards the packet to the application layer. If the node is not the destination, the packet is not sent to the node's application layer. Instead, the routing layer uses its table and sends the packet to the outgoing queue for the next hop. A visualization of this layout is shown in Figure 2.

Connection properties To simulate some of the physical the limitations of using radio links, the connections were configured to have a greater latency and a slower data transfer rate between nodes using radio communication. Only the radio-radio links and radio-converter links reflect these limitations.

4.2.2 Node Behavior The nodes in the network interact with each other differently. To simulate users communicating with the grid, packets are randomly generated by the application layer of people nodes. Users will also be receiving responses from the grid. The responses may not be immediate and it may be

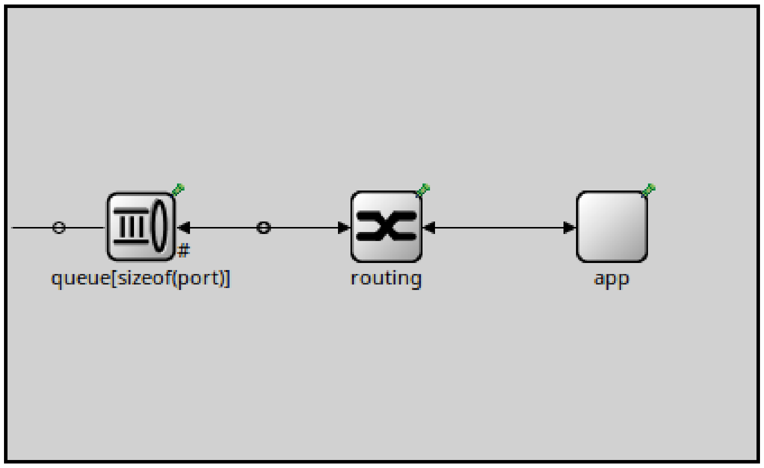


Fig. 2: A closer look at how each node in the network is structured. This shows the connections between the 3 layers within each node. Notice the connections go in both directions.

possible for grid nodes to send multiple response packets, so the application layer of the grid also randomly generates packets to simulate responses from the grid. The frequency of quickly packets are generated is a parameter that can be adjusted with each run. The destination of all packets is either going to be a grid node or a people node. Neither radio nor converter nodes generate or consume packets. They are only responsible for bridging the gap between the people and the rest of the Internet.

4.2.3 Network Topology To create the network for the WINS competition, we considered 4 types of nodes, each with different behavior. The four nodes types are: people, grid radios, and converters. The radio and converter nodes are the two non-standard nodes that we are proposing as part of SARATOGA to solve the problem. The radio nodes are nodes that have long distance radio links to other radio nodes or to converter nodes. The converter nodes act as a barrier between regular wifi connections and radio links. The converters are the nodes that are responsible for the "heavy lifting" in this project. They translate packets from wifi to radio, they compress, they provide forward error correction, and they provide TCP caching. The caching and forward error correction components are done separately in another simulation by a different subset of our group.

We built two different networks in OMNeT++:

– A small fixed/static network. That means that the network's nodes and connections were specified manually in the .ned file. In this case we had 9 nodes. Their behaviors were entered manually at run time. Figure 3 shows the resulting network.

– A large dynamically generated network. For this network, we provided two files: one that specified the the nodes and another that specified the connections between them. The files were processed and the network was automatically generated accordingly at simulation run time. Figure 4 shows this network.

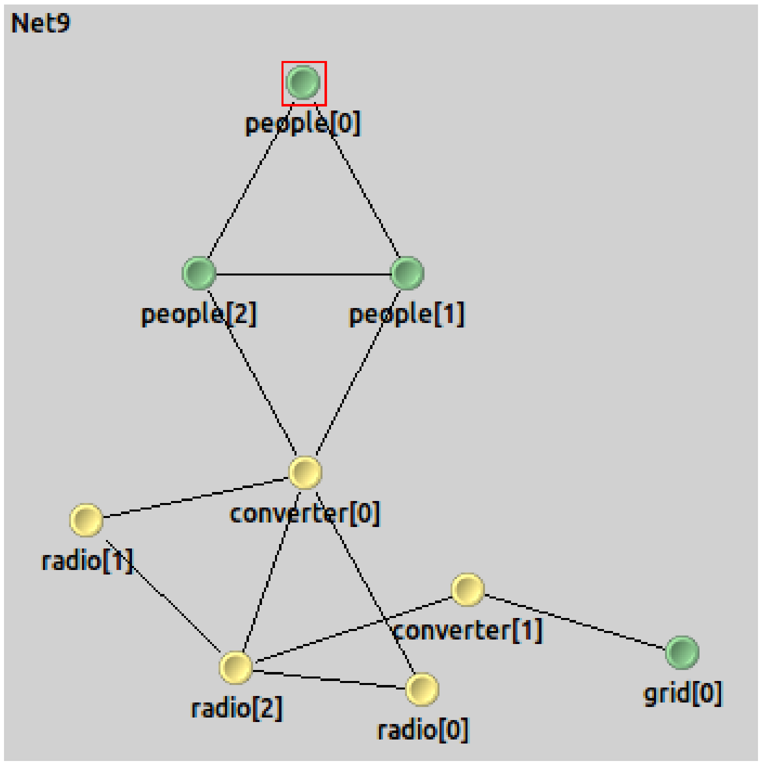


Fig. 3: A network with 9 nodes. The name of the nodes shows the behavior of each node. This small network was mainly used for testing/debugging purposes.

4.2.4 Routing The foundation routing used in this simulation comes from the routing example in the OMNeT++ simulator. In order to rout we need to be able to handle these scenarios:

– Finding a route for a packet from its source to its destination.
– Broadcasting packets over radio links to all neighbors.

The first step is finding a route in order to send a packet to its destination. To do so, routing tables are calculated for each node at the beginning of the

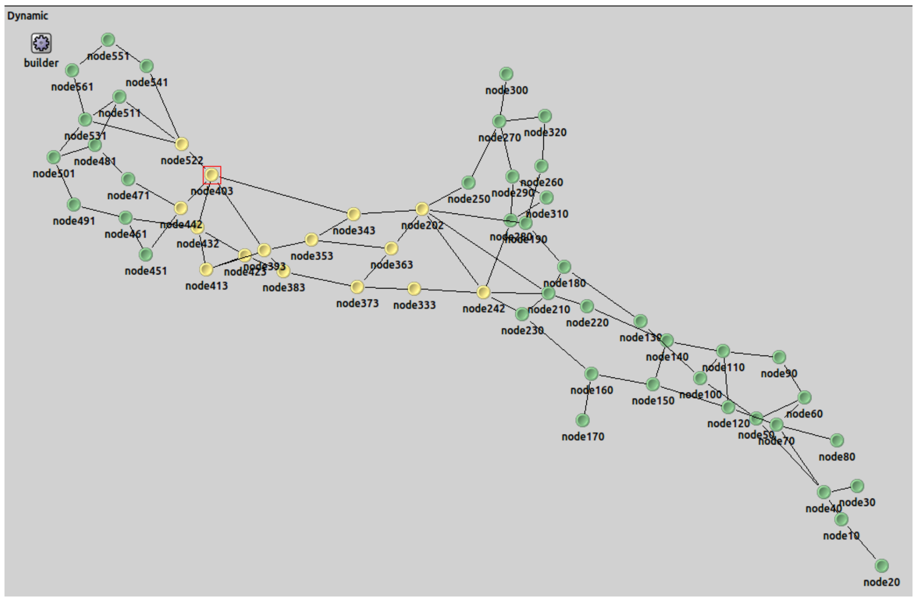


Fig. 4: The network that generated using the dynamic builder. The green nodes on the left represent grid nodes. The yellow nodes in the middle represent SARATOGA nodes (converters/radios). The green nodes on the right represent people nodes. The name of the nodes contains the address and the behavior of each node. Nodes names that end with 0 represent people nodes, 1 for grid nodes, 2 for converter nodes, and 3 represents radio nodes. This network was created in attempt to provide a more realistic network after ensuring the network functioned properly using the network in Figure 3.

simulation. This is done using the cTopology class. Nodes use the cTopology class to know the topology of the entire network. Since the topology of the network doesn't change during execution and it is static, the routing tables can be set once at the beginning and do not need to be updated again. They will always reflect the most up to date state of the network. To compute the routing table, each node iterates through every other node in the network and considers each to be the destination of a packet. Using the topology class, nodes calculate the next hop for each destination node. The next hop is saved into a map. The routing table maps destination nodes to next hop nodes.

People and grid nodes are the only nodes that generate packets. These nodes have knowledge of other people and grid nodes so they know which other nodes accept packets. Packets are generated with random intervals. When a node generates a packet, it randomly picks one of the nodes listed to be a destination address. Note that this means nodes can send packets to themselves. This is fine though because some applications communicate with each on the same host using the network layer.

Because of how radio antennae work, whenever a message is sent, all other nodes with radio antennae in the surrounding area receive the message, even nodes who are not the intended recipient. Because there is no way for a node to know if a message is intended for it or a different node, each node must accept the packet and inspect it to determine the intended recipient. Only then may the node discard the message and use system resources to process other messages. We decided to model this broadcasting behavior because it requires extra processing for each node.

The routing table tells a node the next hop based on the destination address. Each node needs to know what type of nodes it is connected to so it can either send the message directly or broadcast the message over radio waves. In a real life situation, the type of the connected node would be trivial to determine. If the node is reachable via a wifi antenna, the connection is a normal connection. If the node is reachable *only* through the radio antenna, the connection is a radio link. In the simulation, all connections are symbolic of one of these links. Using the type of both the current node and the neighbor node, the type of link can be determined. In order to get the type of neighbor nodes, another table was added to each node in the simulation. The original routing table—*rtable* in the code— maps destination addresses to outgoing ports. The new table, which we call a neighbor table—*ntable* in the code—maps outgoing ports on a node to pointers to that node's neighbor object. This allows every node to access the information about all its neighbors. Using this information, each node can look at the type/behavior of its neighbors to determine the connection type. People and grid nodes do not have radio antennae, so all their connections must be regular connection. Radios are SARATOGA nodes that are only connected to other SARATOGA nodes (they provide no services other than forwarding). Because of this, all of their links must be radio connections. Finally, converters determine their connections by examining their neighbors. If the neighbor is a grid or people node, the connection must be a wifi link. If the neighbor is a radio, the link can either be a wifi or radio link. In our simulation, we consider these links to only be radio links. 1) this allows us to analyze the worst case scenario of the network and 2) maximal network coverage is achieved by placing radios far enough from converters that they are only reachable via radio. The assumption is that users of the system want to maximize coverage which both minimizes network cost and maximizes service area.

4.3 Evaluation

In this section the behavior of both networks is analyzed. Because the radio links are slower, the performance of each network is considered with and without radio links. The goal of this is to hopefully provide some insight on the stability and performance of a network using radio links in comparison to the same network togology using traditional links.

4.3.1 Data Received by Each Node In Figure 5 there is a histogram of the amount of information received by each node. As mentioned in Section 4.2.3 radios are solely responsible for transferring data, whereas the converters are responsible for transforming to radio waves to wifi and vice versa. The topologies of each network is set up so that the people are separated from the grid by SARATOGA nodes. This configuration implies that all packets sent between people and grid nodes must go through the radios and converters. Because radios and converters do not consume packets, each packet that is sent across the radio bridge must be forwarded by all the nodes on the bridge. This means that when a packet is sent across, the number of bytes sent for *each* node on the bridge increases by the packet size. This explains why there are peaks for the number of bytes received by converters and radios.

There are not the same peaks for all the radios. This is due to the broadcasting of the radios. Radios must broadcast packets to each of their neighbors. Thus at the places in the network where the radios have higher connectivity, they are more likely to receive packets. This phenomenon is shown in the plot. For example, looking at Dynamic.node403 and Dynamic.node333, Dynamic.node403 has a higher number of bytes received. If we reference the topology in Figure 4, node403 has many more more connections compared to node333. The reason that there are different values for queues is that nodes do not receive information in an uniform distribution. This is due to the routing protocol. The non uniform distributed of bytes received is discussed in the section "Network Traffic Patterns."

4.3.2 End to end delay In this section we examine the end to end delay in the two network topologies discussed in section 4.2.3. Figure 7 shows the maximum end to end delay for nodes people and grid. There is a gap in the histogram which corresponds to the converters and radio nodes (because they don't generate packets). This histogram shows that end to end delay is is function of distance and the more distance between a node and the rest of the network

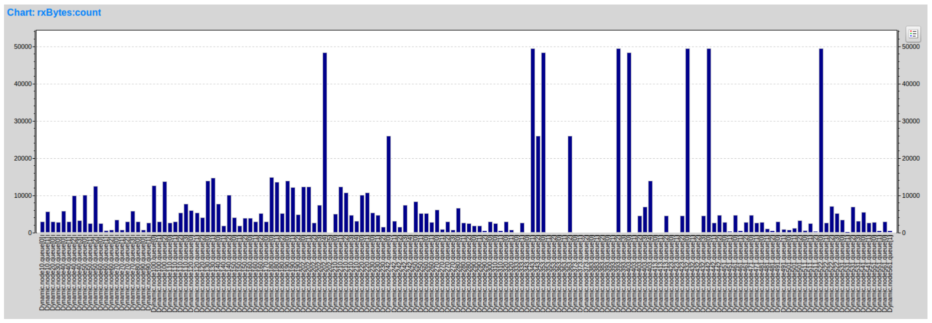


Fig. 5: Number of received bytes for each node. This histogram is the result of running the simulation for the network topology shown in figure 4 for 3000 seconds. The name of each node is shown on x axis. Node names that end with 0 are people, those that end with 1 are grid, those that end with 2 are converters and those that end with 3 are radios. The number of received bytes is shown on y axis.

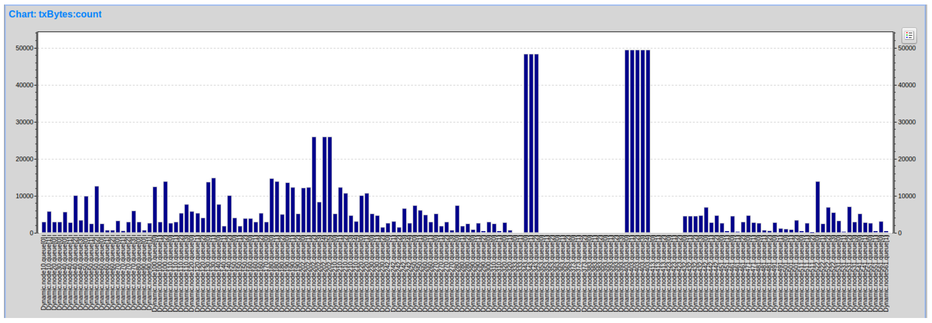


Fig. 6: Number of bytes sent for each node. This histogram is the result of running the same simulation Figure 5.

the more higher the end to end delay would be for that node. For instance, node 20 is located in most far location than the rest of the network and as we can see has the highest delay as well.

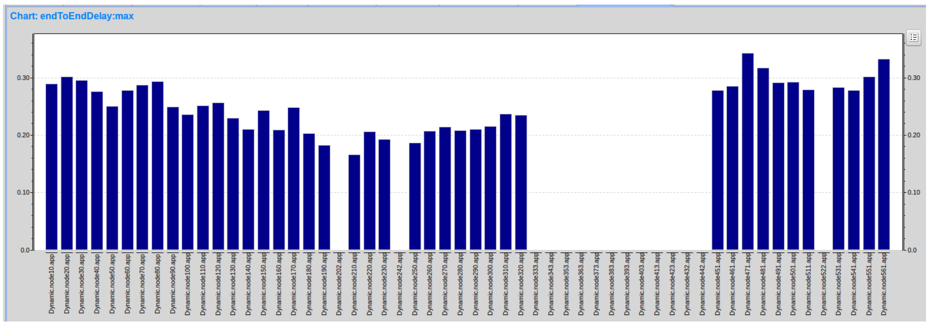


Fig. 7: Maximum End-to-End delay

4.3.3 Network Traffic Patterns To help visualize the traffic patterns and potential places for bottlenecking, the busy time of each node is shown in Figure 8. The busy time is shown for each queue within each node. As we can see in this plot, all the queues of the radios and converters are more busy compared to the regular nodes. This is due to the fact that the radio links are slower and thus take more time to send the same amount of data. The radio nodes only have radio links, so all of its communications are slower. Converter nodes might broadcast data over the radio or they might forward over a normal link, depending on the next hop of the packet. This results in the busy times of each queue in converter nodes depending on the type of connection. We can see this difference by comparing the queues within the same converter. The queues that have large busy times are the radio connections while the other smaller times are the regular links.

Some nodes have a busy time of zero while a few nodes that have large busy times. This difference is due to the routing algorithm. The routing algorithm only routes packets through the shortest path from the source to the destination. What ended up happening is that in the particular network topology, the shortest paths between any two people and grid nodes were all the same path. This meant that all cross traffic—traffic flowing from grid nodes to people nodes or vice versa—traveled through the same nodes. The result was that a few SARATOGA nodes that were on the shortest path were responsible for handling all cross traffic. The large amount of traffic to forward resulted in more busy time. Nodes that were not on the shortest path were never tasked with forwarding packets, so they were never busy transmitting messages. Due to broadcasting, these nodes still received messages, but because they were not on the shortest path, the messages were never intended to go to them and were dropped. Overall, this results in nodes receiving a large number of packets from other nodes that are on the shortest path, but receiving no data from other nodes that are not on the shortest path.

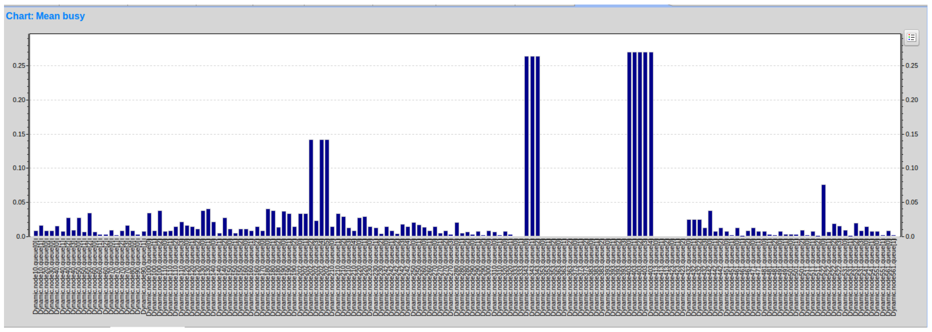


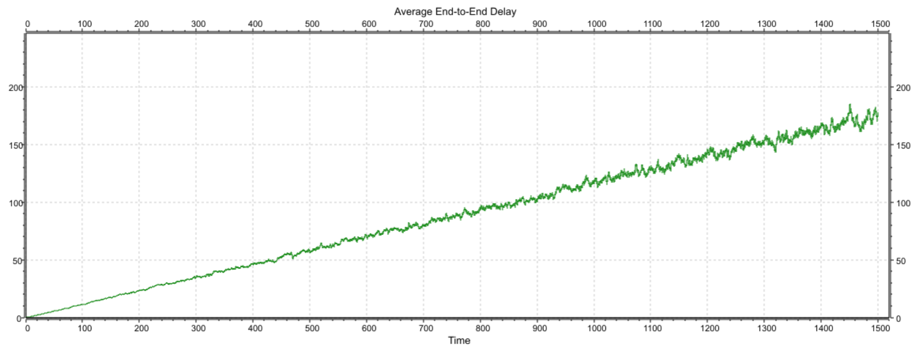
Fig. 8: Busy time for each node.

4.4 Evaluating radio links performance

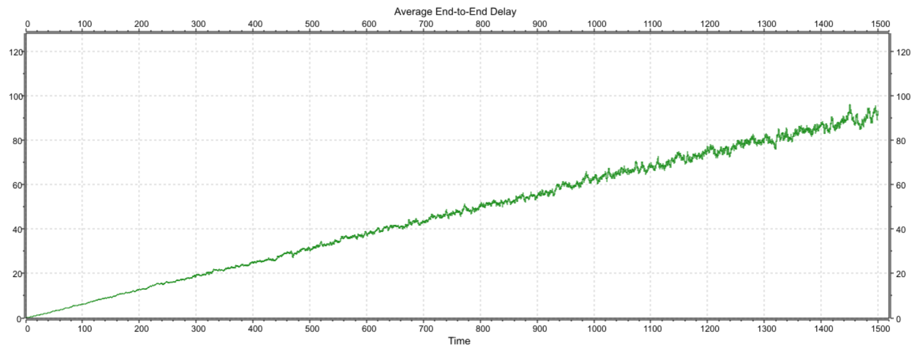
In this section we compare the proposed Radio Based Networks (RBN), to networks that have only wifi links. Figure 9a and 10a show the average delay for RBN typologies shown in Figures 3 and 4 and Figures 9b and 10b show the average delay for wifi linked networks for the same typologies. These plots confirm that the proposed network preserves the stability of a regular network with wifi connections, only adding latency to the network. Even with the added latency, the average eventually stabilizes. If the links were too slow for the network to handle, what we would see is that the buffers would be overloaded and delays would continually increase to infinity as more and more packets get dropped.

5 Live Simulation

Beyond implementing SARATOGA in OMNeT++, we also wrote an end-to-end simulation that simulated a setup in which two radio nodes service requests for clients: one local node connected to several off-the-grid clients, and one remote node with a reliable internet connection. The local and remote nodes communicate over long distance radio to provide internet connectivity to the clients. Whereas the OMNeT++ simulation simulated routing behavior in a larger network, the end-to-end simulation examined cache, coding, and compression performance in the more specific context of clients requesting pieces of content via a single local radio node.

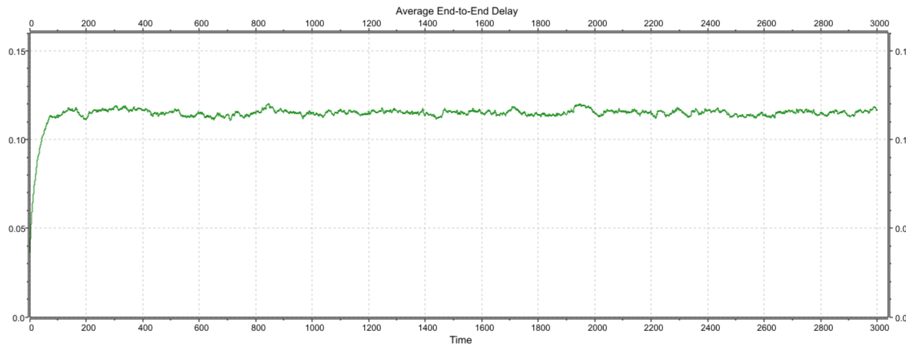


(a) Moving average of end to end delay for the small network of 9 nodes with radio links.

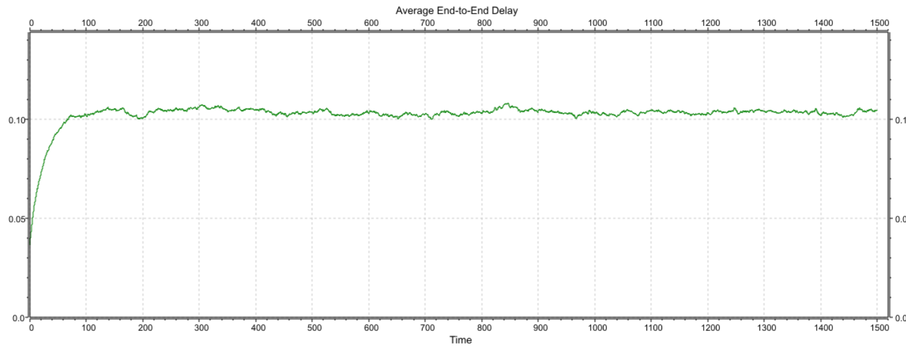


(b) Moving average of end to end delay for the small network of 9 nodes without radio links.

Fig. 9



(a) Moving average of end to end delay for a network with 58 nodes with radio links.



(b) Moving average of end to end delay for a network with 58 nodes without radio links.

Fig. 10

5.1 Forward-Error Correction Implementation

5.1.1 Background and Theory Forward-error correction is similar to erasure coding – encode data and send the encoded data such that errors in the code can be detected and (hopefully) corrected by the recipient at arrival. There are various forward-error correcting codes, but only two seemed to fit our purposes: turbo codes and low-density parity check (LDPC) codes. Both are probabilistic decoding algorithms that can almost reach the Shannon information capacity of a channel. They are so robust that they are used to satellite transmissions and LTE communication.

Most real-world Turbo coder implementations are hardware-based, and finding an accessible software framework for Turbo coding proved difficult. Originally, we planned to use the CommPy library in Python (`https://github.com/veeresht/CommPy`), but the documentation was sparse and required some prior knowledge of Turbo codes in order to use effectively; it fairly slow as well. PyLDPC is another Python library that uses LDPC codes as part of a simulation framework – it is fairly easy to use and was kept as a fall-back in case no other frameworks proved useful. Eventually, we found the IT++ library, which is a digital communication simulation and modeling framework written in C++. The functions are fairly accessible, and the documentation, although sparse, was available for the functionality we needed.

Turbo codes rely on many of the same mechanisms in erasure coding but vary in implementation. The typical example of a Turbo coder has 2 parallel recursive systematic coder (RSC) blocks, which are generator matrices. Generator matrices can be expressed as an identity matrix with the bottom most rows being polynomials over a Galois field with terms equal to the number of bits in the word size of the system.

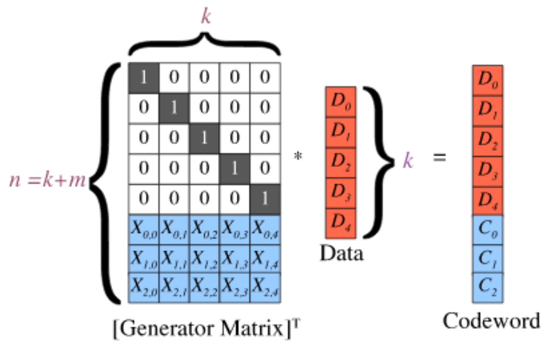


Fig. 11: Image from "A Complete Treatment of Software Implementations of Finite Field Arithmetic for Erasure Coding Applications" by Dr. James Plank et al.

One RSC receives the input bits, performs a matrix-vector multiplication with the input bits and the generator matrix, and outputs a set of codewords. The second RSC receives a permuted set of input words (the output of an "interleaver") and performs the same operation. The recursive part of these systems is that the initial output of the RSCs is fed back into the next set of input bits, typically by XORing the input of the next word. These encoders can be combined in various ways through serial or parallel concatenation and through recursive or non-recursive encoding.

The decoding process is a bit more complicated, and its implementation is not discussed in detail here. Decoding uses an iterative soft-decision approach to determine the value of bits. The decoder receives the bits and generates a range of values indicating the likelihood that a bit is a particular value. The decoders then exchange these values (if they differ) and attempt to decode using these likelihood values until the two decoders agree. In this way, message bits can be reconstructed with minimal overhead despite errors in transmission.

5.1.2 Implementation Fortunately, IT++ abstracted away many of these implementation details. The structures that we needed to define include:

- Generator Matrix
- Interleaver
- Maximum Iterations

For the generator matrix, only two polynomials were used (two rows of blue blocks in the diagram), corresponding to two codewords per input word. MATLAB's Turbo Code implementation and other reference documents all referenced polynomials corresponding to 013 and 015 (in octal), which is $x^3 + x + 1$ and $x^3 + x^2 + 1$, respectively. Since these numbers seemed ubiquitous, we used them for the generator matrix in `Transcoder.cpp`.

The interleaver is provided by IT++, uses a word size of 320 bits, and is based on the one used for Wideband Code Division Multiplexing Access (WCDMA). The interchange of decoding information usually has a hard maximum of 8 iterations, but due to the amount of information being processed at one time, this was turned off in favor of an adaptive stop. The adaptive stop parameter immediately terminates the decoding iterations when the decoders do not change any of the information in their decoded values between iterations (similar to convergence).

5.2 Simulator Design and Assumptions

5.2.1 Overall Design The software simulated a number of clients connected to a local off-the-grid long-range radio station via a reliable, high-bandwidth, low-latency link. Requests from clients flow through the local radio station which communicates with a second, internet-connected radio station. The link between the two radio stations was modeled to simulate a reasonable bandwidth achievable over long-range radio (1 MB/s), with varying bit error rates to simulate loss in the presence of noise.

The simulation also incorporated models of the strategies employed by SARATOGA to mitigate the possible issues with long-range radio: forward error correction, caching, and data compression. All simulated traffic flowing between the radio stations was first compressed using the GZIP compression algorithm and encoded using error-correcting codes. The local radio station also cached all responses from the remote station. When a client sent a request for which the local station had a cached response, the local station sent the cached response immediately over the high-performance, reliable local network.

The simulation was divided into discrete rounds. During each round, a random number of requests were generated from clients, modeled as a Poisson distribution assuming a constant average rate of requests. The content requested was selected randomly from a body of websites with a non-uniform distribution; the probability of selecting a site was proportional to the site's reported popularity, as measured by the number of referring subnets. When the request traffic exhausted available bandwidth for a discrete round, further requests are dropped.

5.2.2 Forward Error Correction To simulate the effectiveness of forward error correction, all requests were encoded with a forward error-correcting code (turbo codes). The encoded data was subjected to random corruption at a constant rate of 10%, to simulate bit errors in the presence of noise. The resulting code was then decoded and checked for integrity using a CRC32 checksum. If corruption in the final result was detected, then the receiving party ignored the packet. To model the effectiveness of the Turbo codes in the presence of noise, the `GaussianNoise` program was written; its results are given in the next section.

5.2.3 Retransmissions Reliability was achieved through a very simple retransmission mechanism: the local station retransmitted requests until a response was received, and the remote station transmitted a response for every request received. In this way, if either the original request or the response is lost, then the entire conversation must be replayed. Although this approach guarantees that a successful conversation will almost certainly occur eventually, is not optimal; by packetizing requests and responses and adding the capability to indirectly reference packets through a unique ID in a manner similar to TCP sequence numbers, more efficient retransmission could be achieved.

5.2.4 Measuring Latency The latency of a request was calculated as the sum of the latencies of any re-transmissions, plus the time required to send the data over the wire (the product of the request size and the bandwidth), plus the internet latency: latency of the request from the remote radio station to the requested web site over the internet. The time spent compressing, encoding, and decoding the data was assumed to be negligible compared to the latencies of network communication.

A list of the top 10000 websites with a corresponding popularity measure (number of referring subnets) was used as a model for a large body of content with realistic contents and a skewed distribution of popularity.

In a realistic body of content it is expected that the popularity of certain pieces of content will be heavily skewed, and therefore that the most popular pieces of content will account for a disproportionate number of requests. Since the effectiveness of the cache will depend on the rates at which the same content is requested, it is important that our simulation use realistic content popularity measures. As Figure 12 shows, the distribution of site popularities in the real data set has the expected skewed distribution where a very few large sites have popularities several times more than the vast majority of sites.

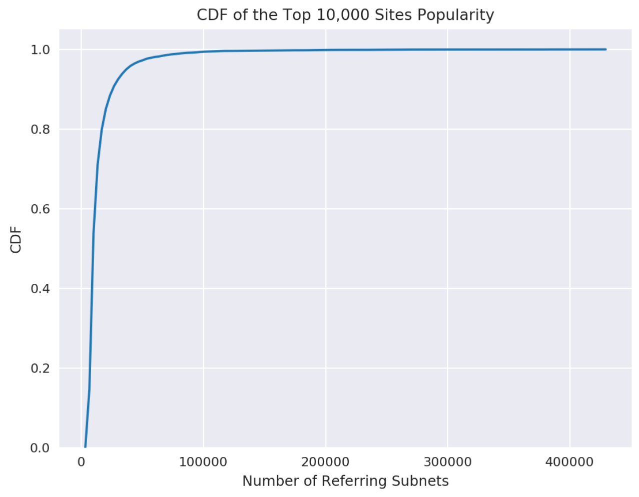


Fig. 12: CDF of site popularity from the dataset used to bias random site selection.

Requests were generated based on the URL of the requested website, in the form of a standard HTTP request. For the response content, we scraped the actual content of all reachable sites in the list (approximately 85% of the sites were reachable at the time of our simulation). Likewise, the internet latency of the request was modelled as the actual latency in scraping the site contents. Realistic site contents are important for measuring data compression performance, as the effectiveness of the compression algorithm depends heavily on the type of data being compressed.

5.3 Evaluation

Several metrics were used to evaluate performance. In each simulation, a measurement was made for each metric at each round as a running average. The change in metrics over time demonstrates the effects of the cache. As expected, caching significantly improved performance for all metrics for all simulation parameters.

5.3.1 Hit rate The cache hit rate is the most direct measure of the cache's performance. It is calculated simply as the number of requests found in the local node's cache divided by the total number of requests. For larger cache sizes, hit rates as high as 30% were observed after several rounds. This is in part due to the biased selection of sites; since the few most popular pieces of content account for a disproportionate number of requests, a cache is highly effective in servicing those requests.

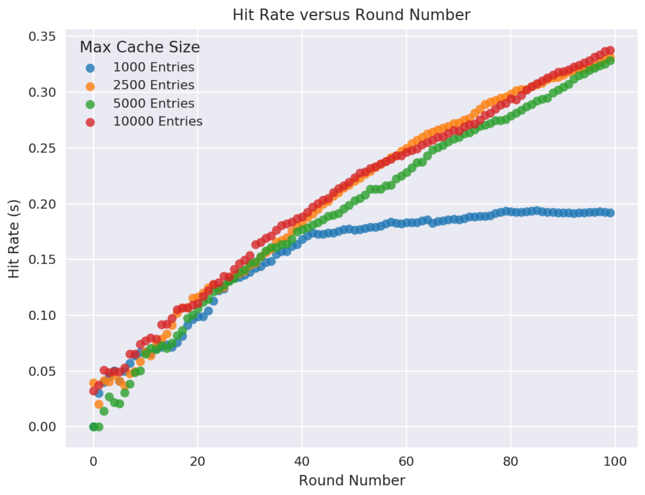


Fig. 13: The moving average of cache hit rates for different cache sizes.

5.3.2 Retransmission rate Retransmission rates measure the number of times a request or response had to be retransmitted per request. Retransmission of a request or response impacts performance both by increasing the round-trip latency of the individual request and using bandwidth for the transmission, which may prevent the node from servicing other requests. Retransmissions occur whenever data loss is detected in the request or response via a CRC32 checksum.

In general, the forward error-correction was effective for preventing data loss even in the presence of moderate bit error rates. Nearly all transmissions were successful at a 10% bit error rate. Because transmissions were typically on the order of several kilobytes, the probability of a correct transmission without error-correcting codes at a 10% bit error rate is on the order of $1 - (1 - 0.1)^{(}1024)$, which means that it is overwhelmingly likely that not a single request in the simulation would ever be answered. Although this could be mitigated by dividing requests into smaller packets, this was unnecessary with turbo codes. In contrast, with the use of turbo codes, more than 95% of transmissions were successful at the same bit error rate. This demonstrates that turbo codes are extremely effective at overcoming moderate data loss. However, at error levels beyond this performance quickly degrades. This is apparent in the results from the AWGN simulation, whose results are shown in the graph below. At 15% bit error rates and above, successful requests are so rare that simulation results could not be obtained.

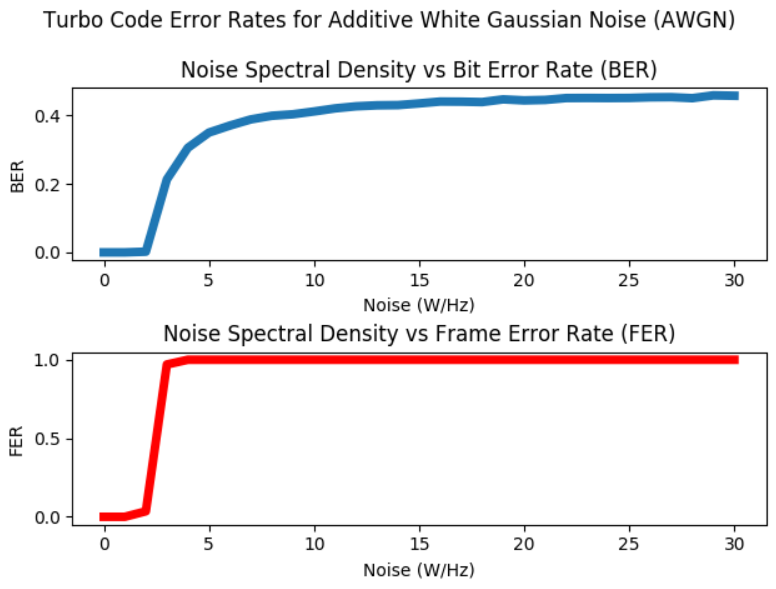


Fig. 14: Bit Error vs Noise

The bit-error rate (BER) and frame-error rate (FER; number of errors per byte or word) scales logistically. Realistically, the noise level will likely never reach the levels shown in 14 in practice – normal power of transmissions on the 2.4 Ghz band is on the order of MW, so most noise levels will be less than 1. The bandwidth is also relatively sparse in low population areas, like the disconnected regions that this project targets, so the noise likely will not be an issue.

One interesting finding is that receive error levels have a much greater impact on performance than transmit errors. This is due to the asymmetry in packet sizes, where responses are typically much larger than requests. We would expect this effect to disappear if requests and responses are divided into packets of equal sizes.

In addition, a reduction in retransmission rates can be observed over time, as more and more sites are cached. This is because a request for a cached site is not transmitted over long distance radio and is not subject to the simulated corruption due to interference. Figures 15 and 16 show the retransmission rates at different bit error levels for receiving and transmitting error rates.

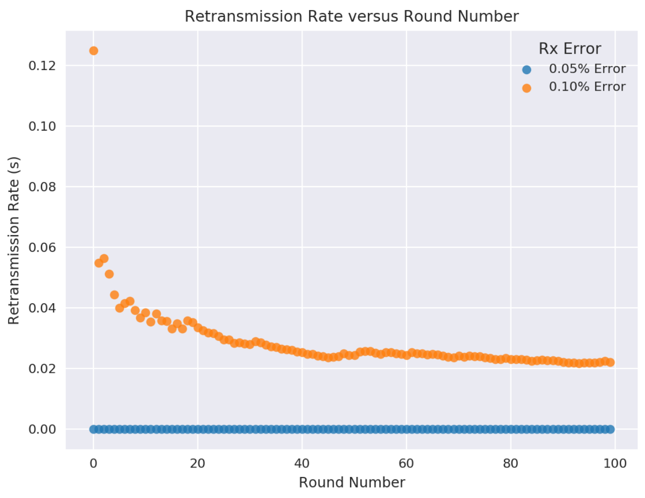


Fig. 15: The moving average of retransmission rates at different receiving bit error levels.

5.3.3 Average latency Because retransmission rates were generally low at the noise levels tested, round trip latency was dominated by the internet latency (the time required for the remote node to request content over the internet). As expected, average latency also drops over time as the cache becomes more effective, with larger cache sizes being more effective.

Figures 17 and 18 show latency graphed with both varying cache sizes and varying request rates, respectively.

5.3.4 Drop rate The drop rate measures the proportion of requests that are never answered due to an exhaustion of available bandwidth on the long-range

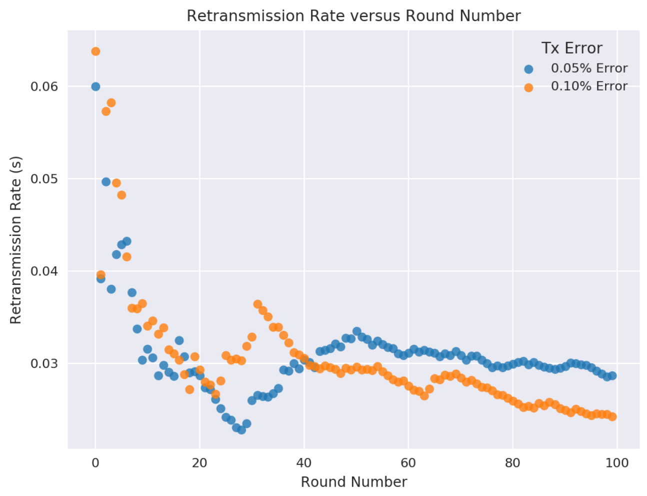


Fig. 16: The moving average of retransmission rates at different transmit bit error levels.

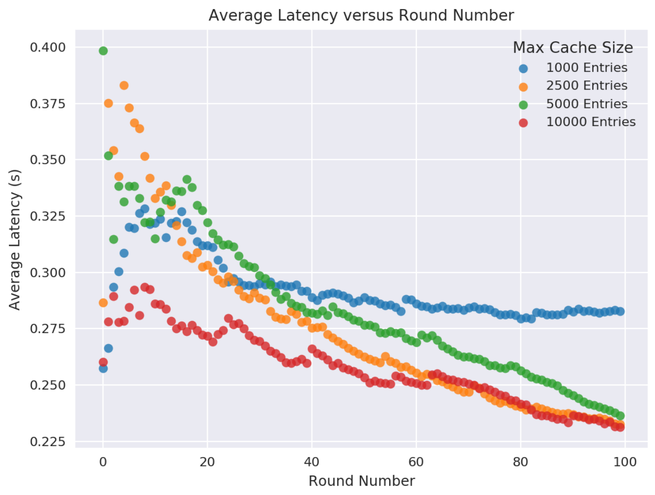


Fig. 17

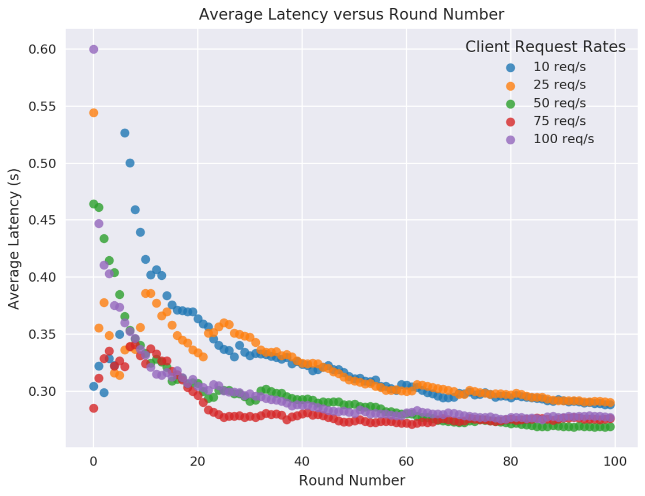


Fig. 18

radio link. Figure 19 shows the drop rates over time for different request rates. This demonstrates that a single node is capable of serving up to 25 requests/second with very low drop rates.

5.3.5 Summary The results show that through a combination compression, caching, and error-correcting codes, the system can reasonably handle up to 25 requests/second even in the presence of significant data loss due to interference at the 10% bit error level. We believe this is sufficient to serve as a form of emergency connectivity in regions impacted by natural disaster.

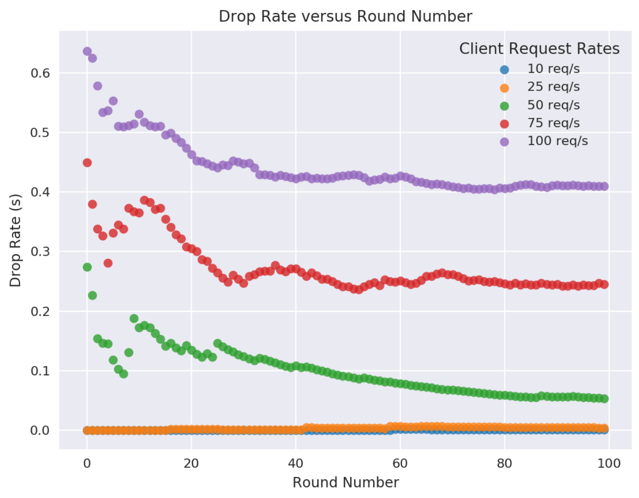


Fig. 19: The moving average of drop rates at different request rates.

6 Team Member Contributions

The members of this project included Kyle Birkeland, Kris Brown, Joseph Connor, Parker Diamond, Tyler Marshall, Sara Mousavi, and Jared M. Smith. The following alphabetically ordered section includes the contributions of each team member to SARATOGA.

6.1 Kyle Birkeland

6.1.1 Background I used much of my previous knowledge of wireless networking, routing, and higher level services from experience working as a network engineer, along with research specifically for this project. My previous work has dealt with designing and scaling a resilient and highly-available network for the University, and has included analyzing and tuning our BGP policies to avoid congestion on internet links, designing a routing infrastructure for our wireless networks which had outscaled most commodity networking hardware, and ensuring the integrity of critical networks services, such as network authentication and DNS.

6.1.2 WINS Submission For the WINS submission, I helped with the design and technical feasibility of the project. From my experience, I was familiar with the current state of how 802.11 works and how a greenfield system could be improved. Based on the previous success with long-range wireless transmission and by efficiently using links in a wireless mesh network along with caching, I believed long-range internet connectivity could be brought to places where fiber or other wired infrastructure couldn't be due to physical or economic constraints. For the paper, I helped with the editing and consistency of the paper and cowrote the following componenets:

- Solution Statement (with Parker Diamond)
- Technical Feasibility (With Parker Diamond)
- Scalability (with Parker Diamond, Jared White, and Kris Brown)
- Portability (with Parker Diamond, Jared White, and Kris Brown)

6.1.3 Live Simulation For the live simulation, I helped refine the design and implementation of the simulated transmission protocol, including adding a checksum for data integrity, fixing padding, and modeling bit error rates between two transceivers. I also helped fix some bugs and dealt with performance optimizations relating to the website retrieval simulation and the slow operations IT++ has built in for its data structures.

6.2 Kris Brown

6.2.1 WINS Submission For the WINS submission I was responsible for, in collaboration with other team members, for the sections on portability, scalability, and users. In order to write these sections I conducted research regarding the components our technical plan required. It was necessary to find information regarding the dimensions, weight, and capabilities of the hardware as it related to our project proposal. I also became familiar with the various types of hardware used in existing radio networks in addition to their power requirements and the price range for acquiring such hardware. Based on those power requirements I searched for various power supplies that would be suitable for our proposal based on factors such as price, size, and weight. I also looked into various solar options to get a sense of what someone could reasonably expect to be able to afford and set up in accordance with the power requirements that I had previously researched. For the users section, I read and researched common problems and responses in the various types of recent natural disasters that have hit the United States over the last year. From this research I wrote about how our system is ideally designed to handle the needs of people in those areas and how it can contribute to a recovery effort and provide missing internet connectivity services.

6.2.2 Background and Lessons Learned Prior to this class I had no experience or knowledge or real-world distributed systems. I have recently experimented with implementing various distributed consensus algorithms, such as the Raft protocol. However, my experience in real network programming work is confined to professional web application development. Through this project I was exposed to other aspects of distributed systems that are not usually considered outside of the work of large scale tech companies which is what I generally thought of when I thought of distributed systems work. It was a great experience to work with a team that has a different skill set than myself and to take part in the creative brainstorming to solve a real-world problem like networking in offline communities in disaster situations. I learned a-lot about different and unique approaches to networking, radio networks, the hardware involved, and the various capabilities and limitations involved in such a project.

6.3 Joseph Connor

6.3.1 Background and WINS submission I researched the expected power requirements and affordability for a long-range radio setup capable of running our proposed system. I researched existing solutions and the hardware they used, and referenced the real prices and technical specifications of the equipment. Using this information, I authored the Power and Affordability sections of our WINS submission.

6.3.2 Live Simulation I designed the architecture of our live end-to-end simulation, including the models for client requests, responses, content data, latencies, popularities, caching, compression, and error correction, as well as the metrics and graphs used to evaluate performance. I also contributed a large amount of code to the implementation, writing the code that implements the client and local radio station models as well as the simulation of long-range transmission and retransmission, except for the actual encoding and decoding of the error-correcting codes. I also solved the technical challenge of interfacing the code of our Python simulation with the available turbo codes library in C++.

I wanted the live end-to-end simulation to provide a reasonable model for the effectiveness of our complete system. Since internet over long-range-radio has never seen widespread adoption due to its significant drawbacks, I wanted to test, as best as was practical, how well our solution could overcome these drawbacks through a combination of error-correcting codes, compression, and caching. To that end, I wanted a simulation that could model all of these systems working together, as opposed to testing individual components in isolation.

For the final paper, I wrote the live simulation design and results sections, as well as the Power and Affordability sections taken from the WINS report.

6.4 Parker Diamond

For the Mozilla WINS submission, I wrote the following sections (with collaborators specified in the parentheses):

1. Portability (with Jared, Kyle, and Kris)
2. Focus Areas
3. Scalability (with Jared and Kyle)
4. Technical Feasibility (with Kyle)
5. Solution Statement (with Kyle)

6. Community/Locations (with Jared)

In general, I focused on constructing a technical solution to the off-the-grid challenge. My initial idea was to create a amateur radio network bridge since amateur radio frequencies can span extremely long distances. I spent a significant amount of time looking at implementations of similar amount of times and investigating methods to increase the throughput of traffic on high-latency, low-bandwidth channels.

6.4.1 Review of Prior Work and Brain-Storming I researched long-range WiFi networks and Ham radio internet, and found several similar projects. Broadband Hamnet, a grass-roots Internet mesh over amateur radio, was the most similar project. According to their web page (http://www.broadband-hamnet.org/), operators can achieve 10 mile consistent communication with typical Internet protocols with commercial-off-the-shelf hardware. Further investigation revealed stringent restrictions on encryption on Ham radio frequencies, however. The FCC prohibits the use of encryption on amateur radio frequencies.

I researched other frequency bands and found that the ISM (Industrial, Science, and Medical) band allows free transmission and encryption – Google may have used this frequency band for Project Loon. Additionally, I looked at projects that used existing 802.11 frequencies for long-range communication. Most notably, I found the TIER project at University of California, Berkeley, which used WiFi over Long-Distance (WiLD) in a chained bridge in order to connect patients In Tamil Nadu, India, with an eye clinic through video conferencing.

With these past successes in mind, I looked for enhancements. From past experience, I know that most internet traffic is from a limited number of hosts and that these hosts transmit mostly the same data. I suggested to the team that we use long-range radio and add caching, compression, and forward-error correction on a network layer in order to make the most of the limited bandwidth available on long-range radios. With the general solution outlined, the work was distributed across the team, and I focused on finding forward-error correction libraries since nobody else had prior experience with them.

6.4.2 Forward Error Correction Implementation All work done on the turbo codes and forward error correction in Section 5.1 was completed by me as well.

I spent a large amount of time researching Turbo code libraries, understanding how they work (somewhat), and how to implement them in software rather than hardware. The patent on Turbo codes as used in the LTE specification just ended in 2013, so there is not much accessible information on the web about how to implement them in software. Nevertheless, I found a few key resources (thanks, Dr. Plank) and libraries in order to implement at least an example solution. Since IT++ uses vectors of bits rather than byte arrays or other primitives, this library is not suited for a real-world implementation of software-defined Turbo coding. However, it is sufficient for this simulation.

As part of the simulation, I wrote the `Transcoder.h`, `Transcoder.cpp`, and `GaussianNoise.cpp` pieces. The Transcoder class is the basis of the Turbo coding within the simulation, while the GaussianNoise program simulates a Turbo encoding/decoding over a channel with additive white Gaussian noise (AWGN). The graphs produced in Figure 14 show the results of the simulation for high amounts of noise in a channel with Turbo encoded transmissions. The code is available in `https://github.com/WINS-SARATOGA/LiveSimulation/tree/master/src/simulation`. Compilation of the code requires installation of IT++, which can be retrieved through aptitude with `sudo apt-get install libitpp-dev`.

6.5 Tyler Marshall

Of course, we all participated in brainstorming solutions to the WINS challenges. There were several proposed solutions that were just never chosen. For most of the project we decided to go with, Sara and I worked closely together. We were both responsible for writing the same sections in the WINS submissions: sections 2.9, 2.11, and 3.3. Other than the submission for the competition, Sara and I are both wrote section 4 of this report, the section about the OMNeT++ simulation. The code the simulation was also written by us. When writing the code, Sara and I practiced pair programming. Originally, we were both new to OMNeT++, we had never used it. Having someone to go through the tutorials with was beneficial in that 1) there was someone to hold me accountable for actually going through them and 2) we could talk out our understanding of the functionalities until we agreed we understood them. Once we were finished with the tutorials, we took what we had and just started building from there. We continued using pair programming because of its benefits. Using pair programming, one of us is able to focus "big picture" topics our project such as where it was going and the overall architecture of the project. The other is able to focus on lower level ideas such as language syntax and program flow. I must say, there were a few instances I can think of where pair programming saved us from having bugs that could have taken *hours* to find.

6.5.1 Lessons Learned Previously, I did not have much experience working with distributed systems. Before I decided on which project to work on, I played around with OMNeT++, going through the tutorials and looking online at larger, more complex examples made by others. Originally I thought of making changes to layer 3 of the network stack and analyzing the performance of networks after the changes. I went with SARATOGA as my project because I liked the idea of creating something that could 1) help others during times of disaster or 2) provide connectivity to small communities who have never had it and who are not likely to get it in the near future because it is such a small market (who else is looking after them?).

Overall, this project was useful to me in that it exposed me to the implementation details of networking (routing algorithms, transmission errors, etc). In this project, I also got to try pair programming. I have read articles in the past about it, but there are not many opportunities in academia to work closely with others on big projects. I had such a positive experience with pair programming and it was was so influential on me that it will most likely end up on my resume. Lastly, I gained experience using git. I have used it in the past, but it is more beneficial to use version control on projects with other people. I know version control is important because it is used by many in the industry. There will be projects in the future where there are many, many others working on it at the same time. Having experience collaborating with multiple people on the same project is good preparation for the future.

6.6 Sara Mousavi

After discussing the idea of using radio signals to connect off-the-grid areas to the Internet with my teammates, I decided to simulate the idea using OMNeT++. For implementing the simulation I collaborated with Tyler. The simulation had three main purposes. First, we wanted to check the feasibility of the radio-based-network idea. Second, we wanted to asses the limitations and tradeoffs of such a network. Finally, we wanted to compare the setup with a typical network in order to gain insight into its behavior.

We simulated the idea of using radios to connect the unconnected areas to the Internet using OMNeT++. We spent several weeks going through its tutorials to learn how to use the environment.

For implementing the simulation we pair programmed. At each programming session, one would write code and the other would revise and vice versa. We discussed each design choice before its implementation. In the process of developing the simulation, we had to make many design choices. Some examples are: how to define sources and destinations of packets in the network, choosing methods for realistically simulating radio links, and designing the overall topology of the network. For each of these, we offered our ideas, discussed the design options and chose a solution.

As for the write-up (Section 4), I created the overall layout in a shared Latex document and we both gradually added content to it.

6.6.1 Contribution to WINS competition What I have done for the wins competition is performing research on the data usage of communication applications that use various media such as text, video, and audio. I then researched the feasibility of supporting them using the proposed network, considering their data usage patterns. I also investigated the amount of compression that we can achieve using different methods for sent packets. Additionally, I researched on error correcting codes, caching techniques, and the limitations of HAMnet radios. The result of my contribution to the WINS submission is what we have for the differentiation (section 2.9), social impact (section 2.11) and application (section 3.3), for which Tyler and I also collaborated in writing.

6.7 Jared Smith

6.7.1 Background Work Prior to this course, my work on distributed systems was contained to working with Internet-scale simulations of BGP routing and traffic simulation for the purpose of measuring techniques to mitigate DDoS. I became very interested in off-the-grid networking once we started discussing the subject in class, as it relates heavily to being able to route around congestion on the connected Internet, in the sense that both face issues of network latency, lack of connectivity, and a lack of availability in general. When the time came to submit a proposal for the course project, I became aware of the Mozilla WINS challenge, I saw an opportunity to apply my interest in off-the-grid networking combined with my expertise in Internet-scale simulation to build a potential solution. To that end, I was responsible for forming the team for this project and providing oversight and organization for the Mozilla WINS submission and both simulation implementations.

6.7.2 Mozilla WINS Submission After facilitating the application to Mozilla WINS, I organized team meetings to divide the work on the Mozilla WINS submission, serving as the team leader for our submission. Then, I led our group discussion of the approach and technical details needed for submitting to the competition based on Parker, Kyle, and Joseph's technical research into both HAM radio and forward error correction. After dividing the work needed to be done via GitHub Issues (see `https://github.com/WINS-SARATOGA/submission/issues?q=is%3Aissue+is%3Aclosed`), I coordinated the drafting and submission of the design phase portion of the competition. In particular, I worked on the following components of the submission in detail:

- Public Website and Documentation: I created the website and public documentation for the WINS submission. Find it here: `https://wins-saratoga.github.io/SARATOGA/`
- Community and Location (with Parker Diamond)
- Portability (with Kyle Birkeland, Parker Diamond, and Kris Brown)
- Openness
- Scalability (with Kyle Birkeland, Parker Diamond, and Kris Brown)
- Problem Statement
- Edited all sections for length requirements, clarity, and content correctness (with Parker Diamond)
- Overall formatting and submission of the final document to Mozilla WINS through the Fluxx portal

After submitting to the design phase, I led the team in regular meetings to iterate on the simulations, at which point Sarah and Tyler took on the OmNet++ simulation and Parker, Kyle, Joseph, and I took on the live simulation.

6.7.3 Live Simulation To fully realize SARATOGA, Joseph and I decided to write a complete end-to-end live simulation as described in Section 5 and in the repository at `https://github.com/WINS-SARATOGA/LiveSimulation`. Joseph led the technical design and specific details of the implementation, while I served as a reviewer and software engineer for the simulation. Through his vision, we ultimately built and evaluated a live simulation of the SARATOGA system with injected loss and latency.

The specific portions of the live simulation that I built include: the simulation framework, including `sim.py`, allowing any number of rounds to be run with various configuration settings; the Internet-connected end of the simulation, `netbase.py`; and the gathering and formatting of the top 1 million sites from the Majestic Million dataset, including their content in the `data` folder which was used as our representative "Internet" throughout the simulation. We used the latency from pulling down these sites as the latency in the simulation, since having the content pre-cached on the Internet-connected end made the simulation run much more quickly without compromising the validity of the solution. My work also included writing a test framework that integrated with the rest of the simulation to run various parameter settings in parallel and gather metrics used for graphs, as shown in `test.py`. With these metrics, I used the generated results to build all visualizations, which are shown in Section 5 above under the results.

Among the parameters I varied for testing include:

- The client request rate, with values of $10, 25, 50, 75, 100$ requests per second
- The max cache size, with values of $1000, 2500, 5000, and 10000$ entries
- The receiving or Rx error, with values of $0.05, 0.1, 0.15$ percent
- The transmission or Tx error, with values of $0.05, 0.1, and 0.15$ percent

All generated graphs are shown in the HTML report in the source code repository for the live simulation at `https://github.com/WINS-SARATOGA/LiveSimulation/blob/master/src/e2e_sim/results/analysis.html`.

6.7.4 Final Report After both teams had finished the simulations, I led the effort to put together the final project report, including doing the final formatting and gathering of all source code, results, and documentation. Additionally, I wrote the introduction, conclusion, and future work sections of the report beyond what I completed for the Mozilla WINS submission included in Section 2.

7 Conclusion

In this work, we presented SARATOGA, or Simple and Reliable Internet over Long-Range Radio, a solution designed for connecting off-the-grid locations with the rest of the world via a scalable and low latency protocol with integrated robust error-correcting codes. We showed in our live evaluation that we can suffer from 10% bit-level error and fully recover the original radio transmissions. Furthermore, the high noise levels that cause greater than 20% bit-error after 10%

recovery are unrealistic in practice. We also show that greater than 99% of requests are serviced with less than half a second latency for 25 requests per second for varying amounts of clients. We presented our submission to Mozilla WINS in Section 2, described our OMNeT++ implementation in Section 4 and our live implementation in Section 5, and the individual team member contributions in Section 6.

7.1 Future Work

Beyond the current simulations, future work would include implementing SARATOGA in hardware. Forward error correct, and specifically convolutional or turbo codes are easily implemented in hardware devices such as FPGAs. With this implementation, we could then use two hardware routers with the power escalated to be able to transmit radio frequencies over the actual atmposphere. With this implementation, we could test the same metrics as our live simulation and OmNet++ simulation in practice. Beyond a hardware implementation, we could add security properties to our model by encrypting the end-to-end communication between HAM clients, though encrypting radio communications is barred by the FCC in certain areas of the United States, which would make testing our system difficult.

Bibliography

[1] Amazon. (2017). *Linksys WRT54GL Wi-Fi Wireless-G Broadband Router*. Retrieved from https://www.amazon.com/Linksys-WRT54GL-Wireless-G-Broadband-Router/dp/B000BTL0OA

[2] Retrieved from https://tukaani.org/lzma/benchmarks.html

[3] Linksys. (2017). *Wireless-G Broadbrand Router*. Retrieved from http://downloads.linksys.com/downloads/datasheet/wrt54gv8-ds.pdf

[4] TC. (2001). Retrieved from http://lartc.org/manpages/tc.txt

[5] Mozilla WINS. (2017) *Wireless Innovation for a Networked Society (WINS) Challenges*. Retrieved from https://wirelesschallenge.mozilla.org/

[6] Plank, James S., et al. A Complete Treatment of Software Implementations of Finite Field Arithmetic for Erasure Coding Applications. A Complete Treatment of Software Implementations of Finite Field Arithmetic for Erasure Coding Applications.

[7] Veeresh Taranalli, "CommPy: Digital Communication with Python, version 0.3.0. Available at https://github.com/veeresht/CommPy", 2015.