An Ad-Hoc Wireless Communication System

by

Hector Yuen

B.E. Telecommunications Engineering
National Autonomous University of Mexico, 2003

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of

Master of Science in Media Arts and Sciences

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2006

Author ....
Program in Media Arts and Sciences
August, 2006

Certified by ....
Dr. Andrew B. Lippman
Senior Research Scientist
Thesis Supervisor

Accepted by ....
Dr. Andrew B. Lippman
Chair, Department Committee on Graduate Students

An Ad-Hoc Wireless Communication System

by

Hector Yuen

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
on August, 2006, in partial fulfillment of the
requirements for the degree of
Master of Science in Media Arts and Sciences

Abstract

This thesis studies the challenges of providing load balancing and fault-tolerant external links between ad-hoc multicast mesh networks.

The work is the gateway component of a research platform called *FluidVoice*, a wireless audio communication system. This system consists of nodes forming a broadcast mesh based on 802.11. Some of these nodes called Stargates have the capability to communicate to the external world. The problem is that these gateways can fail or lose capacity unexpectedly. In this work we explore the ways to provide communications to the external world under unexpected gateway node failures, and variance of load.

We propose and evaluate a distributed algorithm designed to form this robust and balanced interconnection. The algorithm is designed with robustness in mind, and takes into account failures in the outbound links as well as between the gateways, and it is focused to support real-time applications running over it.

In this thesis we show that by adopting this algorithm, we can provide a reliable connection to the end-user even as gateways presence or capacity varies. The prototype version has about 20ms of additional transmission time in average, with an overhead of about 5% to 35% depending on the packet size, and a recovery time of 1 to 3 seconds. The redundant traffic generated in intermediate steps of the optimization problem can grow up proportionally to the number of participating gateway nodes, and reduces quickly to only the required amount of traffic.

Thesis Supervisor: Dr. Andrew B. Lippman
Title: Senior Research Scientist

An Ad-Hoc Wireless Communication System

by

Hector Yuen

The following people served as readers to this thesis:

Thesis Reader [illegible]

Dr. David P. Reed

Adjunct Professor

MIT Media Laboratory

Thesis Reader [illegible]

Dr. Ted Selker

Associate Professor

MIT Media Laboratory

Acknowledgments

I would like to thank Andy Lippman for his encouragement and guidance during my time at grad school.

David Reed for all the time we spent having discussions related to the thesis, and other topics, I really enjoyed learning from you.

Ted Selker for helping me polish this work and encouraging me to build a demonstrable version.

To Kwan, Jamie, Fulu, Aggelos, Dimitrios, Dean, Ilia, Grace, Durga, Dawei and Polichronis for making the lab an enjoyable place.

To Eric, Ashtu and Kevin for making the apartment a place I wanted to return to during schooldays.

To Boris Escalante for encouraging me to do research.

To my parents for teaching me the important things in life.

To Jenny and Joel for all the support and love.

Gal 1:24 And they glorified God in me

Mat 6:11 Give us this day our daily bread.

Mat 22:14 For many are called, but few are chosen.

Psm 119:56 This I had, because I kept thy precepts.

Psm 25:4 Shew me thy ways, O LORD; teach me thy paths.

Luk 24:48 And ye are witnesses of these things.

Pro 8:33 Hear instruction, and be wise, and refuse it not.

Rot3

Contents

Chapter 1

Introduction

Ad-Hoc cooperative wireless networks are a perfect experiment to explore the capabilities of peer-to-peer systems. In these systems, intelligence is uniformly distributed between all the entities that form the network.

Static routing breaks down in setups where the node mobility is high, such as in the cooperative wireless networks.

In this thesis I will describe the gateway component of *FluidVoice*, a system that takes advantage of inherent radio capabilities of wireless systems to perform communications inside a mesh network, and sporadically requires to communicate to nodes in the external world.

The part of the system that provides external communications is formed by some special nodes called gateways. In the border, the various gateway nodes that serve as packet forwarders between the internal and external world require an efficient and robust way to allocate traffic. Is in this part of the system that our attention will be focused, on a way to perform this task properly so nodes in the two sides of the network can rely on having a good connection for any real-time application they wish to perform even if some of the participating links fail unexpectedly.

The problem for which this thesis provides a solution is on how to distributedly do traffic allocation over several links, taking into account their capabilities. Additionally, we are considering that links are faulty, so our aim is to improve the overall link reliability by enabling the system to recover from unexpected failures on each one of their components.

In this system, the application running over it is a two-way communication system, so latency is a parameter of major importance. We are focusing on minimizing the delay introduced by the action of adding redundancy to the links to increase availability.

1.1 Contributions

In this thesis I present a solution for load balancing in a multi-gateway environment. The traffic to be allocated comes from the internal FluidVoice nodes, which are connected through an unreliable network. This network is the same that the gateway system uses to communicate.

The solution consists on a distributed algorithm that performs a traffic distribution between all the nodes that participate in the calculation. This algorithm is robust to communication errors, as well as node failures.

As a side problem, it was required to perform flow fingerprinting for the packets that go through the network. Since this is a hard problem, and the general solution is still not completely solved, a generic framework for the analysis of flows was built, and a classifier for the particular case of the traffic generated by the current system was written.

1.2 Thesis Overview

Chapter two collects some of the work relevant to the thesis, in this section we will give a brief overview of the state of art in wireless mesh networks, review some ideas used for load balancing, and see which are the main contributions of the algorithms for routing in wired networks.

Chapter three describes the problem we are trying to solve, and enumerates a list of questions that came out while designing and implementing the final solution.

In Chapter four, we describe the architecture design, define the requirements for the system, as well as the algorithms to solve the distributed gateway load balancing problem.

Each component of the system is studied in detail so the expected behavior is well simulated and corresponds to the real one. We describe the reasoning behind the gateway algorithm, and explore all the possible scenarios that it can face, showing that it successfully performs under them, providing the robustness required.

Chapter five describes the implementation details of the system, we explore the technical difficulties and how they were solved to build a working system. These problems range from setting up a telephone network to deal with bandwidth limited links.

Next, in chapter six, we perform theoretical and experimental analysis on the system build under the design principles described in the thesis. We conduct several experiments to see the behavior under stress tests.

Finally we conclude by gathering the lessons learned from building this system, as well as the future problems that came out while thinking on the solution of it.

Chapter 2

Background

In this section we will describe *FluidVoice*, a system that is being developed at MIT Media Lab. We will give a brief description of the general aspects of the system, which are required to understand and bring context to the the gateway problem we are trying to solve.

Following that, we will give an overview of other ongoing work, describing their main features, showing how this project is similar or contrasts them. These include load balancers, wireless mesh networks, and gateway coordination algorithms.

2.1 FluidVoice

A local set of nodes form a *ground mesh*, where each one of the participating nodes can route data, voice and instructions between them. This configuration allows for continuous connections and reconfiguration around blocked parts by hopping from node to node.

This mesh is formed to provide communications to a group of users that need to deploy connections in an unplanned environment.

This setup can happen without having any previously built infrastructure, and the time frame for setting up the system is on the order of minutes.

A set of first responders to a disaster is an example set of users –The fire, ambulance and police all need to be in close contact with each other. In addition, they need to communicate outside that close-knit community, for example with hospitals and support personnel.

Another example might be traders in a real-time financial market. Stock traders want to know what is going on with different markets in real time. They establish connections with around eight to sixteen different stock information sources. These sources give short messages from time to time. When a stock trader is interested to listen about a certain market, he will tune into it, but always keeping the other information in the background.

Similarly, a group of tourists may want to wander around an area and jointly look for a restaurant for dinner, or a military mission may need to coordinate separate forces while maintaining contact with a command and information center.

A last example could be a group of people with cellphones, all connected through an alternative network such as 802.11, with each node having different characteristics, such as number of free minutes, dialing plans, etc. By taking advantage of that diversity, we would be able to avoid the cellular connection for communication between nodes, and route calls originated at the mesh through the phone with the biggest amount of free minutes, or to the one with the best reception in the moment.

We assume that the underlying network provides communications in a broadcast mode by default. This is, each message is delivered to all nodes on a best effort basis. There are no peer-to-peer messages. This experiment is designed in such a way because we want to study the capabilities and limitations of wireless communications. In 802.11 almost all the antennas in the devices are omni-directional. By taking advantage of the inherent broadcast capabilities of these devices, the goal is to reduce the amount of retransmissions required to send information to all the surrounding nodes. Work outside the scope of our thesis is to create a decentralized energy-efficient broadcast network.

In order for the system to be able to communicate to the external world, we need to build the mechanisms for these nodes to reach outsiders and accept incoming messages. This thesis addresses the means for a set of randomly located peripheral access points to be used to guarantee that the mesh will maintain this outside contact.

The issue here is to maintain contact with the highest possible number of gateways and through at least of one of these reach the outside and delay this process the smallest amount of time.

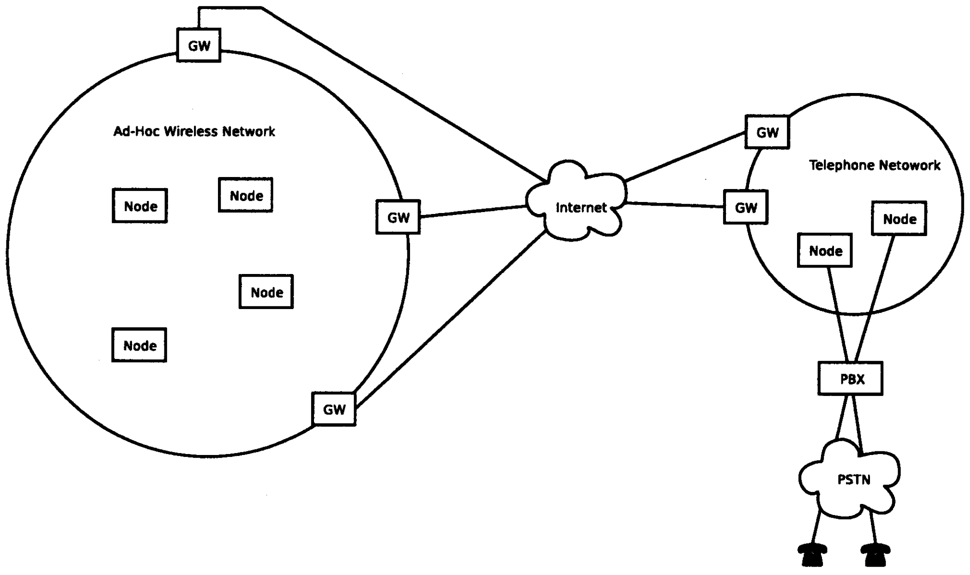

Figure 2-1: Example Configuration

In Fluid Voice, the local mesh is designed to ensure that all communications reach all other nodes by any path rather than by the best one. Right now communications happen through broadcasts. Eventually we want to change it to use a set of redundant multicast trees, or any other useful mechanism such as the algorithms that MANETs[8] use. With this abstraction in mind, we assume that nodes are one hop away from each other.

There is a proxy converting data from the telephone network format to the one used inside *FluidVoice.* This provides means to create FluidVoice nodes that pipe information from a conventional telephone. With this, we can add conventional telephones to act as external FluidVoice nodes, that connect to an Autonomous System formed by the wireless nodes.

A sample configuration is depicted in in figure 2-1. We have several internal nodes that can communicate between them, and are connected to the external world through the available set of gateways. In another mesh, a proxy translates telephone calls to FluidVoice conversations, which connect to the external world through their own set of gateways.

2.2 Related Work

In this section we compare the ideas used to develop the thesis project with the ongoing work similar to it. We present a brief description of the main ideas of these projects, and show how they are similar or related to the current system.

2.2.1 Wired Routing Algorithms

Conventional wired network algorithms for routing make use of periodic broadcasts to the vicinity of the node, and then collect information to build a tree, so any node inside the network can be eventually reached. Algorithms like RIP and OSPF [4] use these mechanisms.

The problem of applying these algorithms to wireless systems is that wired links are much more reliable than wireless links. Separate cables are electrically isolated from each other. In the wireless case, the medium is shared and it is necessary to deal with co-channel interference, these problems can be reduced by using techniques such as MACAW [5], where by adding some control packets before sending big frames, the delivery reliability can be increased, and throughput is increased.

On the other side, wireless communications have the inherent capability of working in broadcast mode. With this, we can increase reliability by replicating information along the network, and reconstruct the final packet without asking the sender for retransmission.

This has some drawbacks, such as an increased amount of required bandwidth, a buffer is required to store packets in intermediate nodes, and this can in the end cause some delay, which can play an important role in the delay caused on two way communication systems.

Even that our focus is on the gateways, since the communications between the gateway nodes are performed through the internal network, we need to know the behavior, so our system is adaptive to the conditions inside the network.

The main difference between this system and the conventional systems is that the amount of border gateway routers is small and route reconfiguration tends to be uncommon.

Examples of this behavior can be seen in the computers with just one default gateway entry, which has a single address and most of the times it is just one entity. Even that maybe at the core there is an active load balancing system going on, errors in the local network need to be fixed for that specific default gateway when it fails. In our system, we provide several outgoing gateways so in case of an outage, the down-time is minimal.

2.2.2 Wireless Mesh Networks

The internal network of the Fluid Voice project is realized as a Mesh Network. There are various different approaches to multi-hop wireless networks, most of them work well under very specific conditions, or still don't have a practical implementation.

Networks that have mobility capabilities are usually classified as MANETs[8], and there are some algorithms developed for routing inside these networks. The performance of these algorithms depend highly on the degree of mobility of the nodes. Another important requirement for our mesh network is to have low latency, which may not be achieved when some algorithms sacrifice delay for high throughput. Some of the most relevant algorithms for wireless mesh networks are AODV [9], TORA[12], and DSR [10].

The idea of having an Ad-Hoc network was inspired mainly from the Roofnet [6] project, where a set of computers with wireless network interfaces form a mesh that provides network connectivity to a small part of a city. Although this experiment is more focused on studying the behavior of the internal network, having an effective connection to the outside world is a key feature in the implementation.

In this network, the gateway is a single node. All the traffic that goes in and out of the mesh has to go through it, since the proxy adds a special layer to try new ideas to give robustness to communications. This is feasible in a setup like the one used for the real system, since the gateway node is inside a very reliable place, where it can be fixed and checked constantly. However it is a single point of failure.

The main difference between this project and Roofnet is that the amount of border gateways is bigger in our system. By having more than a single node, a routing problem needs to be solved inside the mesh, and if we add the fact that some of these nodes can appear and disappear suddenly, the problem becomes even more complicated.

Another problem not addressed by Roofnet is how to provide real time communications. Roofnet aggregates packets into batches, the sender node broadcasts the entire batch, hoping that the receiver gets the highest amount of packets. Intermediate nodes will retransmit the missing packets in the batch. These intermediate nodes are chosen depending on who is the best candidate given the ETX metric.

This approach highly increases the efficiency of TCP-like protocols, where resending a lost packet can jeopardize the entire flow. The only problem is that in the process of batching packets, the delivery of them may be too slow for real-time two way communications such as voice and video conferencing.

Ad-Hoc On Demand Distance Vector (AODV) uses on-demand route discovery to provide the best path on certain moment to a destination. Whenever there is a need to find an unknown route, a route request is broadcasted, and the answer is sent back through one of the branches of the broadcast tree.

This algorithm is one of the most popular ones for static mesh networks. It is used in several deployments to provide wireless networks in cities. Examples of those are Seattle Wireless [2]. However, performance decreases considerably when mobility increases. This happens because routing tables cannot change as fast as the location of the nodes, that move between several obstacles for the radio signals.

In a setup as the one described above it is very often that intermediate nodes realize that the path is broken, and there is a need to recompute the route. The time required to know that the path is broken and to fix it has serious effects on higher level protocols, by making them drop their performance to an unusable level.

For our system, we are considering just broadcasts, so we don't need to worry about intermediate route reconfiguration. The drawback is the amount of scalability in the size of the mesh, and probably the amount of power required to keep the system running, since each sending node needs to talk directly with the destination one, causing it to output higher power, and probably interfering other communication pairs.

2.2.3 Load Balancers

One of the most common applications of Load Balancing is to provide high availability to common services, such as web servers, or e-mail servers. The motivation for having replicated services is because one single server is usually not enough to handle all the load from the clients. The server is a single point of failure, so having a cluster of servers connected by a fast network can be the solution to provide redundancy to the system.

The Linux Virtual Server Project [17] is a project that provides a basic framework to build highly scalable and highly available network services using clusters of commodity servers.

The architecture of the system includes a load balancer, a pool of servers, and a back-end service such as storage. The load balancer handles incoming connections from the outside world using IP load balancing techniques, then selects servers form the server pool and maintains the state of concurrent connections as well as forwards packets.

This virtual server can do load balancing through a NAT, IP Tunneling, or Direct Routing. These mechanisms are implemented depending on the size of the deployment.

In the NAT case, the advantage is that any TCP/IP compliant machine can be behind it, and the setup can be made using a private address space. The drawbacks of this setup is the low scalability. The reason behind this is that requests and responses have to go through the same server, so the link is saturated with a fairly small amount of requests.

Another option is to put an IP tunnel, where one machine listens for requests, and forwards them to one machine from the pool of servers. The answer to the request is sent directly from the assigned server to the client, without the need of the load balancer. This approach requires the servers to support tunneling in their kernel, which is becoming more common everyday.

Direct Routing is when the load balancer processes the requests and forwards it directly to the idle server. It avoids to have the overhead of a tunnel, but requires that the server OS have the ARP response disabled, since all of them respond to the same address.

For this setup, the load balancer may become a single failure point of the whole system. In order to prevent the failure of this node, a backup node is setup next to it. Both machines exchange heartbeat messages through a specialized channel. If the master node seems to be dead. The backup node will use ARP spoofing to take over the virtual IP address to provide the load-balancing service.

In this thesis, we are going to extend the ideas provided by the LVS project by taking away the hierarchical structure of the system. All the previous work was done considering two or more kinds of nodes, and redundancy was provided by having several machines waiting to take care of the jobs of a failing machine.

We remove the single point of failure caused by the load balancer by running a distributed algorithm in each one of the participating nodes, making disturbances in any random gateway node to cause the same effect on the system, that other nodes can take care of.

The algorithms that the Linux Virtual Server uses are of major interest in this project. These algorithms are very well known, and can be explained intuitively. However, implementation details can represent a challenge for the correct execution of these algorithms.

LVS uses Round Robin, or Weighted Round Robin to do task assignment. The first one assumes all servers are identical, and by adding a weight function, we can address issues as different capacities.

The Least Connection metric, weighted or unweighted, assigns tasks depending on how busy is each of the servers. This approach can be taken as long as there is a metric that reflects the amount of traffic on each of the links.

Source and destination hashing is a technique used in large deployments, has the advantage of running in constant time, but won't be considered here because it assumes all servers of equal characteristics, and it is a static mapping, where redundancy is usually added by putting several servers listening to the same key space.

To provide more robustness, we could implement the algorithms that DHT's use, where nodes join and leave at random times, and the key space is partitioned between the available servers.

Instead of just using the network to store and retrieve key entries, we could use it to register services on each of the machines conforming the network.

But since the amount of individual traffic is very small compared to the usual hash table spaces, and we would like to have explicit information about the load of every link, this is not a practical way to approach our problem.

Chapter 3

Problem Statement

In this chapter we will describe the problem we are trying to solve, the challenges and limitations when trying to provide a feasible solution to the problem, and the consequences of having such solution.

Inside the network, there can be two type of nodes: ones with connection to the external world, and ones without such access. This configuration will build up several groups of nodes that intend to form Autonomous Systems (AS), that can be connected to the external world through the available gateway nodes, that conform a subset of them. These gateway nodes will provide redundant links to the Internet or to other AS's.

The problem we are trying to solve is on how to do the best allocation of the traffic generated by the internal nodes among the existing gateway nodes. The goal of the project is to build a system of redundant gateways that will provide uninterrupted communications from any node inside the FluidVoice network to the external world. This means that gateways have to be prepared to handle unexpected outages in the network, or of other gateway nodes.

There are some important challenges that the solution must meet to be effective. The most relevant ones that occurred while developing this thesis are the following:

- Gateway failure

  Gateway failures can happen in a variety of ways, ranging from a lost beacon packet that is used to update the state of each gateway, to losing the link to the external network link and needing to reestablish it, to the worst case of losing the gateway completely due to a physical damage.

We need to find out which are the best mechanisms to be aware of the real status of the gateways. These state updates must be accurate and be delivered on time for the rest of the system to react to the sudden changes and minimize the amount of lost packets.

- Internal Network failure

  As showed in the following sections, the algorithm that we design is a distributed one, so intergateway communications play a fundamental role in the correct execution of the system. Since the objective is to provide reliability under any difficult condition, the algorithm will use redundancy to deal with lost messages.

  The number of expected transmissions required to successfully deliver a message between all the participating nodes depends on the probability of successfully transmitting packets inside the network, so we provide a solution that when lost packets exist, the system will be in a suboptimal state, but still working.

- Which are the parameters to optimize?

  Our solution focuses on two parameters, availability and packet delay. Availability can be defined as the percent of time that the links can be used, and packet delay is the time it takes for a packet to get from any node inside the mesh to the external world. Other solutions may be required if other parameters were important. For example, in the case that delay is not as important as high throughput, packets could be stored in batches in intermediate nodes, and sent when the conditions are more favorable for increasing reception chances. Such batch algorithms would not support real-time voice very well, however.

  If we want to minimize the packet delay that results from providing this redundancy, we need to focus on the algorithmic section, because delay caused by the hardware is not really a big component of the end-to-end delay.

The time required for the gateways to reach agreement is the principal component of the recovery time. So the main feature of this algorithm is to do this computation concurrently with traffic forwarding. This is achieved by sending redundant traffic when the state of the system is suboptimal, and then send the exact amount of traffic when agreement has been reached.

In order to provide availability, the algorithm running on each one of the nodes will forward traffic that it believes nobody else is forwarding. This will give an impression to applications running over it that the link is always available, thanks that the underlying protocols are running to coordinate between the available gateways.

- Where should route mantainance be performed?

  For the system to work properly, it needs to take decisions about traffic allocation. The place where decisions are taken makes a huge impact in the complexity of the deployment. The question is where to put the decision making engine, so the system achieves optimality as fast as possible, without sacrificing to much processing.

  One of the lessons learned from early designs of the Internet architecture is that any design must be simple, even if it in a state where not all the parameters are optimized. By having a suboptimal solution, we leave a gap for future improvements. Extremely optimized solutions have the problem that will be very specific to the particular case that needs to be solved, and won't work for any slight variation in the input.

  By designing protocols in such a flexible way, we will try to accommodate connections that were not thought in the design stage. This is the main goal behind *Fluid Communications*, where senders and receivers can interact between them even if they were not designed specifically to do so.

  For this reasons, the solution needs to be provided as a generic framework for solving the allocation of traffic into multiple entities, not just the one that we are encountering right now.

- How much are we changing the original networking stack?

  There are various options to present the set of gateways depending on how intelligence should be partitioned. In a first setup, all the intelligence is put in the set of gateways, so the end nodes will broadcast to the biggest amount of gateways possible, leaving them the job of agreeing who should send the packet.

  Another totally different setup could be putting an agent in the end node, and let it decide which gateway to send it to, so gateways will forward packets by default.

  The third setup is a combination of the two setups stated previously, an agent is going to be in the end node, and gateways will coordinate between them.

  For this particular case, the first implementation is going to be chosen, since the focus of this problem is to solve the border gateway coordination one, and we are not so interested in adding more complexity to the internal nodes.

- Configuration takes time

  Coordination must be done fast enough so the system does not overload before agreement has been reached. This is the reason for making the coordination process asynchronous and separate from packet forwarding.

  There are a huge number of variables that can describe the state and behavior of a gateway. By taking some of the most relevant ones, we can generate a metric that reflects the channel conditions depending on the interface capabilities, the current load of the line, and other characteristics that are found to be useful. Additional to that, it is possible to collect long term statistics about the link, so we will be able to do some kind of prediction on the channel.

  The design of this part of the system is not targeted just to this particular application, which requires low latency and the fewest dropped packets possible. Other applications may require other characteristics, so the cost function will be modified.

  Each time a packet arrives to the gateways, or a subset of them, the gateways that successfully receive the packet need to decide who is the best candidate to push the packet to the outside world.

- Stability of the output

  The frequency of the gateway calculation process has to be high enough to adapt to the dynamic conditions generated by varying number of flows, and lose the minimum amount of forwarded packets. At the same time it cannot be extremely fast because we need to give some time for the system to stabilize.

  If the system tries to recalculate too fast, routes may tend to oscillate, so we need to be aware of this kind of behavior during implementation.

  The speed of recalculation may depend on the probability of packet delivery, because it affects directly the expected number of trials before having a successful transmission.

- Network configuration complexity

  There are a huge number of variables that can describe the behavior of a gateway. By taking some of the most relevant ones, we can generate a metric that reflects the channel conditions depending on the interface capabilities, the current load of the line, and other characteristics that are found to be useful. Additional to that, it is possible to collect long term statistics about the link, so we will be able to do some kind of prediction on the channel.

  Choosing which set of significant parameters is a challenge, since we need to include all the parameters that reflect the state of the system, so it can correctly react to sudden changes in it.

  The design of this part of the system is not targeted just to this particular application, which requires low latency and the fewest dropped packets possible. Other applications may require other characteristics, so the cost function will be modified. In the implementation section, we will try to isolate the computation of this value as much as possible, so other knowledge that is found to be useful for future applications can be added without the need of changing the entire implementation.

- Traffic fingerprinting, and granularity of it

  In a real-world setup, gateway capability variations can be big enough that it will be convenient to isolate separate sessions through the same gateway.

  There is a need to identify which packets belong to the same session. Once we define that, we can treat them as individual conversations, for which we would allocate resources.

  A way to identify a certain flow of information can be applying a Hash function such as SHA1 to a subset of parameters that distinguish it, such as a combination of IP address-port pairs, or additional information that can be get from the messaging protocol of the application.

Chapter 4

System Overview

In this section we will describe the components that form the system. We will give a brief overview of the requirements of the system, as well as the algorithms that run in each element of the system.

4.1 Architecture

The system used to provide communications between the inside and the outside nodes is formed by a group of nodes that have external access. We call this nodes Stargates. These nodes behave as internal nodes, and also have a connection to the external network, that allows them to communicate all the nodes in the mesh with the outside world.

The difference between this system from the conventional ones is that new routes can be added and deleted much more often than in a wired network. The communication channel between all the gateways is not very reliable, so often polling between gateways is required to know the status of each one of the gateways in the system.

Several outside links are established, the reason is to provide redundancy and reliable communications under this circumstances. With this, it can be seen that each mesh network can be treated as an Autonomous System, since internal nodes can communicate seamlessly between them. Work is being done to provide this reliable connectivity. In the generic case, we are dealing with the problem of communicating autonomous systems through several redundant links 4-1.

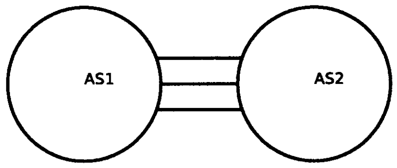

Figure 4-1: Generic Architecture

This work focuses on optimizing the architecture for real-time applications such as voice, but the results can be applied to any generic data network. We can envision any group of closely connected computers as an AS, that communicates between each others through redundant links. An example of this could be the computers behind the NAT of every house, that could conform a set of ASs running as an overlay network built over the Internet.

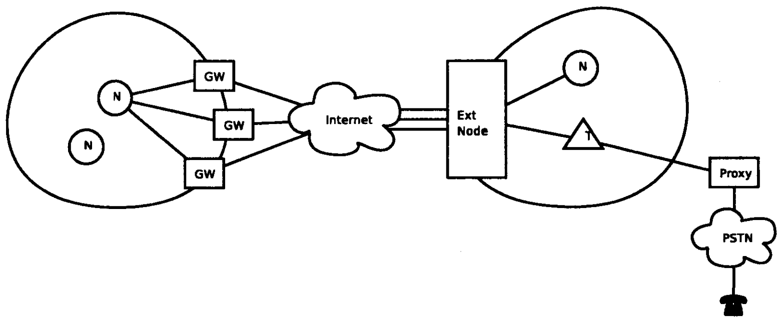

Figure 4-2: Main elements in the architecture

Nodes are randomly positioned within the mesh. In order to provide robustness to the system, there is no hard coded topology between the set of nodes. The set of nodes will decide at runtime which one should serve each of the clients requesting service. Each one of the Stargates will provide an availability index to be compared.

The availability index is computed with static and dynamic parameters. Potential static parameters can be the theoretical maximal bandwidth, expected latency in the link, between others, and in the range of dynamic parameters, we can consider parameters such as current load of the system, the instantaneous throughput and delay of the network. These parameters will be used to feed an estimator, in order to give an idea of how good it is to use the gateway at a given time.

The gateways need to perform two tasks basically: the first one is how to choose the optimal gateway to communicate a certain flow of data, and the second one is how to provide route reconfiguration when a a gateway that is currently sending data stops responding.

In order to communicate data through the link, packets that a gateway gets need to be tunneled to be sent, and reconstructed and delivered in the other side, so nodes in both sides can treat each other as in the same subnet.

The service where the links go through can vary in characteristics depending on the provider, and the type of link. The ideal case will be to provide a real Internet point-to-point link, but sometimes this does not happen, and the IP addresses provided are behind NATs. In these cases, it is required to use a NAT traversal technique.

Important decisions have to be made in the way packets are being tunneled. This is because the size of the packets are small, comparable to the header size of an IP/UDP packet, so by stripping down redundant information, we can construct our own header so occupies only the required amount of space. By doing this, and enabling packet header compression, we can have a relatively smaller packet size, which is critical in bandwidth-limited links.

The main challenge of this system is how to perform the tasks described in a robust way, come out with a solution in the shortest amount of time, and perform this operations in a distributed fashion. To make a compelling voice demonstration, it was considered to add PSTN capabilities, so a proxy was built to translate telephone calls to VoiceMesh compatible packets.

The rest of this chapter describes the main characteristics of each of the elements that conform the system. In the next chapter we will describe the current setup from where measurements are being taken.

4.1.1 Internal Node

A node consists on a computer with a wireless network. The setup is in this way because a computer is flexible to program and prototype, but any 802.11 enabled device that is fast enough to handle voice compression and generic GUI display can be used.

The networking stack is a typical IP network, running under a Linux machine. The Link Layer is for now based on 802.11b, with some modifications to allow mesh capabilities. These capabilities are implemented between link layer and network layer, and they are still on experimental stage.

There is still not a definitive solution on creating an optimal wireless mesh network. There are some attempts to create them , the most relevant ones are AODV[9], DSR[10], and ExOr MAC[7], between others.

The implementation of these mesh algorithms will provide us a convenient abstraction for the nodes inside the network, since all of them are going to be one hop away.

Communications within an AS are broadcasted on a best effort basis. For our purposes we do not need to know whether the broadcast is done via repeaters or by direct broadcast.

The reason to use broadcast communications is to recognize that radios in networking naturally multicast. Antennas used in radio networks are omni directional, which makes it very attractive to use for broadcast mode since radio waves are already being broadcasted by default.

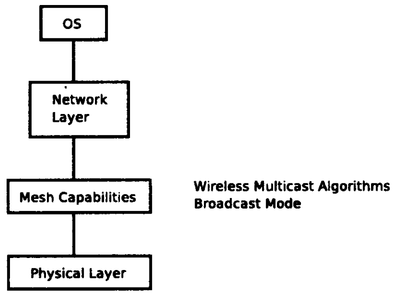

Figure 4-3: Node architecture

Right now routing is done in the node granularity, and not in the application one. That means that in the case with a single stream of audio going in and out from the node, there is no problem, but in setups with multiple inputs, where each one should be routed in a different way, this may cause problems, by forcing all the flows through the same routing rules. In that case, the fingerprint function must be changed to account for this types of traffic, which may require to add some kind of message identifier in the data packets.

4.1.2 Gateway Node

The design of the gateway node is based on the architecture proposed for the internal node. The gateways are extensions of these internal nodes by adding them capabilities to establish a connection with an external link.

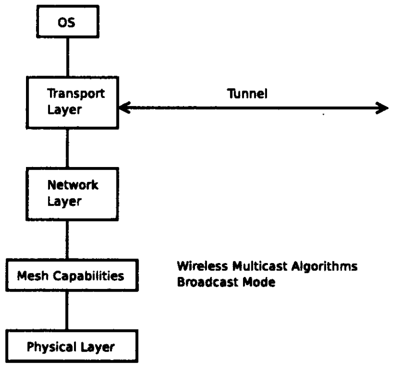

Figure 4-4: Gateway architecture

The gateway node listens to packets originated from nodes inside the mesh, or itself. If the received packet is intended to be broadcasted, then it is then forwarded to the external node through the link connection it has established.

In order to send a packet through the link, it first needs to be encapsulated, so the link connection can forward it as one of its data packets. The initial encapsulation chosen was IP over UDP. The reason was because some of these links require NAT traversal,and UDP packets work well under these conditions. It was seen that although it works satisfactory for high bandwidth links, overhead is too big, in extreme cases, the required bandwidth was more than double. In the implementation section we will describe in detail the structure of the pseudo header and the reasoning behind its selection.

Gateways can reach outside the AS via connections of different nature, thus their bandwidth, latency, or other important characteristics are different from those within the AS. Even for links with the same theoretical capacity, the physical conditions at the time of establishment can give variations on their characteristics.

Since all gateway nodes are able to receive the broadcast packets within the AS, it is necessary to coordinate their packet forwarding so the load allocation happens in an optimal way. The algorithm for allocation is described in the following section, and the key feature of it is that flow traffic does not get interrupted by the agreement process.

An inter-gateway network is established, where the participating nodes can send messages containing information about their own state, so other gateways can gather information from the rest and make a correct assessment of the load in the entire network.

4.1.3 Link handler

When gateway nodes initialize, they connect to an external node to setup a connection with it. We will describe the characteristics and requirements that this node has.

This node is considered to be in a safe place, so availability is much higher than any of the gateways. That is the reason for having one physical node in it. However if some extra redundancy needs to be added, it can be added through establishing a master-slaves scheme, where the master node receives task requests and redirects them to the available slave nodes.

For simplicity, the link handler is designed as a multi-threaded program, with a parent thread listening for requests from external gateways, and spawning children nodes whenever a link needs to be established.

The link handler has two tasks: The first one is to handle incoming packets from the links, reconstruct a valid IP header from the provided information, discard the duplicate packets, and rebroadcast the rest of them to its internal network. The second is to gather the packets from its internal network and send them through the link with most availability back to the mesh network.

A garbage collector runs in the external node. It periodically goes through each one of the children to see if the gateway that is connected to is still alive, if not, it will free the process and reassign outgoing flows to other active nodes.

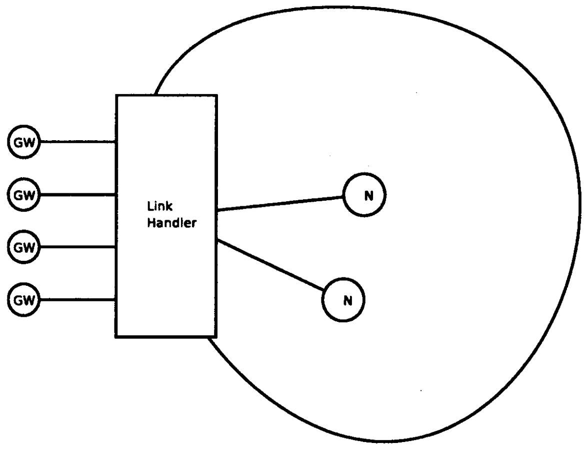

Figure 4-5: Link Handler

4.1.4 Telephone Interface

To make a compelling demonstration with voice communications, it is required to be able to accept incoming calls that can participate in the communications.

One of the big complications for making this happen is to have access to a telephone switch that can transform PSTN calls into VoIP calls. The infrastructure that is being used right now is the MIT PBX that can make this transformation for external calls to campus.

An Asterisk[1] server setup so any call to a range of extensions will arrive to this server.The job of the Asterisk box is to give answering services to the system. The format of audio is usually G771, which is not suitable for transmission over bandwidth limited links, so there is a need to compress the audio, and at the same time reformat it to something that FluidVoice understands. The Open Source Shtoom [3] package is a soft-phone implementation that handles SIP calls to interface with Asterisk, and at the same time handles G711 and Speex [16] compression, which are the formats that we want to be able to translate. A modified version of Shtoom along with an Asterisk server were used to build the proxy.

The modified Shtoom client takes the VoIP capabilities and the Networking layer of FluidVoice. It registers to the local Asterisk server, so when there is an incoming telephone call, the extension number is mapped to the correct client. Voice packets get translated to Speex encoded packets with FluidVoice headers. These packets are

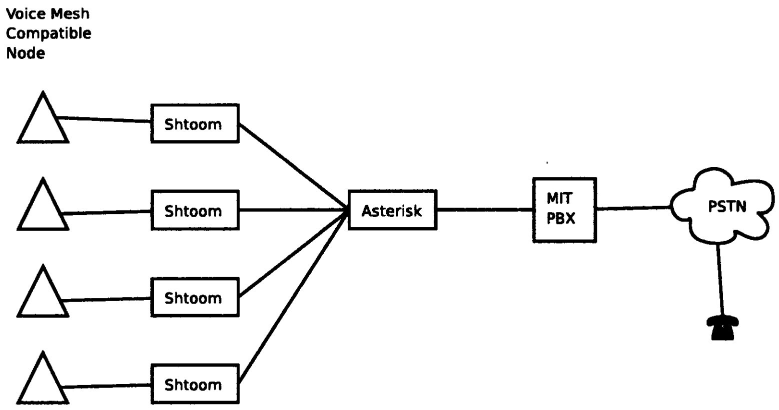

Figure 4-6: Telephone Interface

connected to the external node interface, so it takes care of sending information to the nodes inside the mesh.

4.2 Algorithms

The algorithm used to compute the flow allocations is described in this section.It consists of two parts, the one handling outbound traffic from the mesh, and the one handling inbound traffic. First we will define the requirements and design principles for this algorithm, then we describe the algorithm itself, and finally we explore the possible conditions it will encounter and make sure that it behaves correctly under all the possible conditions.

4.2.1 Design Considerations

We are looking to design an algorithm that does load balancing of one or more flows through one or more gateway nodes. A useful algorithm for this application requires it to be completely distributed. The instances running on every node have to coordinate between themselves and decide the optimal resource allocation.

It requires to have a fast coordination, since we cannot delay the packets too much, because that would cause timeouts that can cause problems with the applications running over this network.

Optimal allocation may not happen very fast, since the priority of the algorithm is output a flow that needs to go out of the network. Simultaneous flows can occur if gateways still haven't reached an agreement, and these duplicate packets can be discarded in the external endpoint.

Since the algorithm is going to run in the same network as the one used for data, it needs to provide its own mechanisms to work correctly under flaky communications.

In the external node side, it is required to have a buffer containing the most recent packets, so it can have information to decide if the packet is a duplicate one. The size of the queue must be chosen correctly depending on the characteristics of the traffic.

How to send packets back to the network is an important issue, the algorithm needs to take into account the current load due to the outbound traffic, and based on this additional information, it will run a similar calculation for the inbound traffic allocation.

Dead gateway detection is done by running a garbage collection algorithm that periodically samples the gateways for liveness, and deletes them. If the gateway hasn't had activity for a long enough period of time, this garbage collector process will deallocate the resources assigned for the certain link.

4.2.2 Gateway Algorithm

The input of the algorithm is a set of M gateways $G = \{G_1, \ldots, G_M\}$, and a set of N flows $F = \{F_1, \ldots, F_N\}$, and the output is a correspondence table between gateways and flows $\{(G_i, F_j) \mid i \in M, j \in N\}$.The flows are the result of applying a fingerprint function to every packet that passes through a gateway.

Initialization

1. Each gateway G_i is identified by its index i. The gateway contains three sets: GF_i, that contain the flows that the gateway is forwarding, AF that contains all the flows that are being forwarded by all the gateways that G_i can see, in the ideal case, all the gateways of the system, and B, that contains the control beacons received from neighboring nodes. These three sets are set to be empty at the beginning.

Packet handling

1. When a packet arrives to G_i, it will calculate the fingerprint F_j to determine which flow does it belong to, and record the time of arrival T_s.

2. If the flow F_j is not in AF, it will forward the packet and append (F_j, T_s) to GF_i.

3. If the flow F_j is already in AF, it will check if it is in GF_i. If true, it will forward the packet, and update (F_j, T_s) in GF_i.

4. Else, drop the packet. This situation happens because the packet is already being forwarded by another gateway.

InterGateway Communication

1. Every T_b, each gateway G_i sends a beacon packet containing

   $B_i = \{G_i, C_i, \{(F_0, T_0), (F_1, T_1), \ldots, (F_N, T_N)\}\}$

   This describes the gateway ID G_i, the current capacity C_i, and an array containing the fingerprint and timestamp of each one of the flows $(F_j, T_j), j \in N$.

2. After sending the beacon signal, each G_i will empty B, and listen for beacons for a period of time T_b. It will append all the received beacons B_i to its own B, and construct a list containing the flows and the gateways that are forwarding them.

3. If in B there is a flow with more than one gateway forwarding, the one with highest C_i is chosen, and if there still exists more than one gateway in the list, the one with lowest G_i is chosen. The resulting array is AF, and we take all the flows whose gateway is equal to G_i to construct GF_i.

With this algorithm it can be guaranteed that packets are forwarded as long as they reach at least one of the gateways. In practice they should arrive to all the gateways thanks to the infrastructure inside the mesh to provide local communications.

In case there are lost beacon signal packets, a suboptimal allocation is achieved, where more than one gateway is sending the same type of traffic. This can be eventually fixed when communications are fixed again.

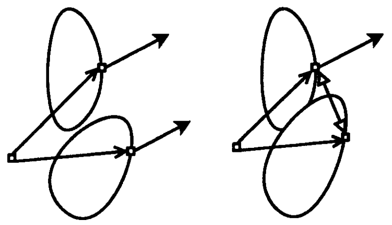

Figure 4-7: Gateway consensus with lost control packets

Figure 4-7 describes this situation. In the left figure, packets are lost in the network, so the set of gateways partition into two subsets. Each one of these subsets reaches an agreement of who should forward traffic, and select a gateway from each group. This is the suboptimal case where more than one gateway is forwarding traffic, with unnecessary redundancy happening.

Eventually gateways will be able to communicate again, and will reach to the state described in the right figure. In this case, where a global agreement can happen, there will be just one gateway selected to forward a certain flow.

Another case is when communications are partial, meaning communications only happen in one direction. Say gateways G_i and G_j are part of the system, as shown in the left side of figure 4-8. G_i can listen to the beacons coming from G_j but not the other direction. In this case, G_i has information about G_i and G_j, and G_j just of itself. When the algorithm runs in each one of the nodes, the one lacking information from the other node may be forwarding extra packets of a flow, but there is very little information because of gateway coordination failures.

The ideal case happens when communications are perfect, such as in the right side of figure 4-8. In this case, each gateway has information of the other one, so when the algorithm runs in each one of the nodes, the optimal allocation is achieved.

When a gateway dies, the outage perceived by the application will oscillate from 0 to the period of a beacon T_b. This short hiccup can be handled by upper layers, which in our case can be the audio codecs, that in the absence of a packet, interpolation

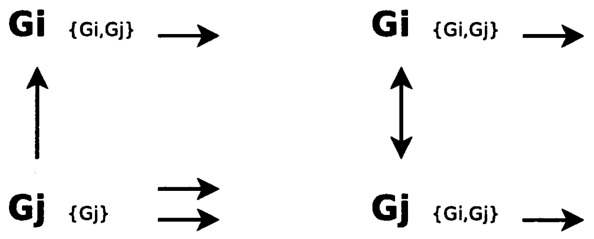

Figure 4-8: Gateway consensus with lost control packets

can be performed with the data available, making the transition more smooth.

Recovery happens because right just before beacons are sent, information about who is forwarding which flows is gathered, and the flows that the disappeared gateway was forwarding will appear as orphan flows, will be picked by several gateways, so the application will see packets again, and another period of time, when the next beacon is sent, agreement will be reached and just one gateway will be forwarding the flow.

4.2.3 Flows inbound at gateways

In the previous section we described the algorithm used to route outbound traffic. Now we will describe an algorithm that deals with inbound traffic allocation on several available links to the AS.

Each external endpoint is in charge of receiving the packets, discard duplicates, reconstruct them to the IP format, and rebroadcast them in its local network. The external node keeps a list of recent received packets, which uses to find out if the incoming packet has already been sent into the AS.

When the external node receives packets from the mesh, it also receives information about the capacity of the link. By constructing a table with information about the state of the active links, the external node will be able to decide which link should be used to send the packet.

This problem is slightly different from the one in the internal gateway, because the algorithm does not need to run in a distributed fashion. All the information is gathered by one entity, and there are no worries about lost packets between inter-gateway communications.

Redundancy in this gateway node can be provided by having backups of this node that can enter the system to replace the dead node. This can be done by sending heartbeat signals from the current working node to the backups.

The objective of this algorithm is to provide an optimal allocation of the traffic going from the external world to the mesh. This allocation is done such that the nodes with least link usage are the first on being used.

The input of the algorithm is a set of M gateways $G = \{G_1, G_2, \dots, G_M\}$ connected to the external node, and incoming flows $F = \{F_1, F_2, \dots, F_N\}$. The output of the algorithm is a correspondence table between gateways and flows $\{(G_i, F_j) | i \in M, j \in N\}$.

Initialization

1. Queue L will contain the last received packets is initialized, and set to the maximal size L_{max}. A table C will contain the capacity of each one of the links that are established.

Input Packet Handling

1. When a packet arrives, a hash is computed over it, and stored into H_n.
2. If H_n is contained in L, the packet is discarded and H_n will be moved to the end of the queue.
3. If H_n is not contained in L, it is appended to the end of the queue, and the packet is forwarded.
4. If L is larger than L_{max}, the front element of L is dropped.
5. If the packet was successfully forwarded, the value of the current capacity as well as the timestamp is updated in $C[in]$.

Output Packet Handling

1. The packet gets the fingerprint F_j.
2. If F_j is in F, we retrieve the pair (F_j, G_j), and send the packet through G_j.

3. If F_j is not in F, select the gateway G_{max} that maximizes $C[in][j] - C[out][j], j \in M$. Send the packet through it, and increase $C[out][G_j]$.

Garbage Collector

1. Every T_g look in F , if $C[in][i]$ has a timestamp older than T_{old}, remove F_i from F.

The minimal size of L_{max} needs to be chosen carefully, since a small one may allow duplicate packets if there are many packets coming from the links. A safe amount to choose could be the number of packets that can come during a period of inter-gateway broadcasts times the maximum amount of gateways.

Chapter 5

Implementation

This chapter gathers the lessons learned from the implementation of the ideas described in the Design section. The implementation is focused in constructing a framework to study new ways of routing between wireless networks taking into account their unique characteristics.

The prototype implementation needs to be easy to modify so new traffic models can be added. At the same time, it has to be efficient enough to perform real-time tasks. The Python scripting language was chosen for the implementation because it combines the flexibility required for prototype development, and good runtime speed for VoIP applications, as reported in [3].

5.1 Gateway Network

In this section we will describe the laboratory setup of the current implementation. This system is built according to the design guidelines stated in previous chapters. The high level diagram of the setup is in figure 5-1.

The implementation consists of five different entities, each one of them is going to be described here. This particular implementation has internal nodes communicating between them, some gateways that provide redundant communications to the outside world, and finally some telephone extensions whose calls are translated to be able to participate in a conversation inside the mesh.

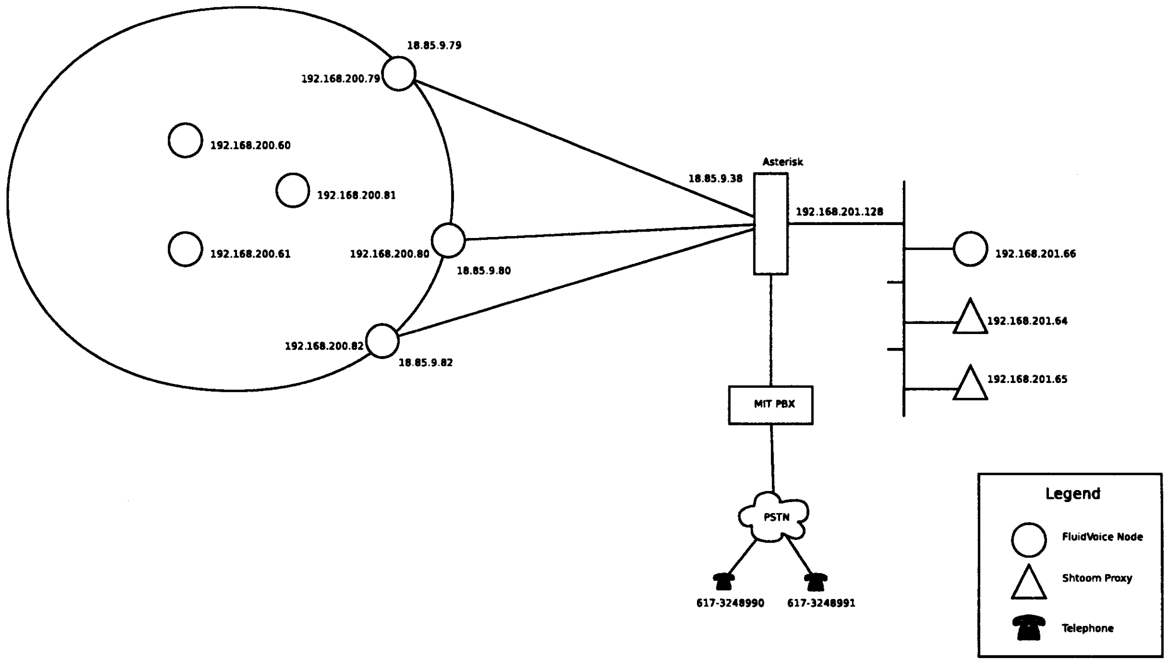

Figure 5-1: Experimental Setup

5.1.1 VoiceMesh Node

This is an internal node of the Voice Mesh. It consists on a Linux computer with 802.11 networking enabled in Ad-Hoc mode.

This node is intended to be as simple as possible, so there is no need to have any extra configuration over it. The application runnning on top of it uses broadcast communications by default, so there is no need to setup a default gateway (there are more than one usually). It is going to broadcast packets to its vicinity, and receive broadcast packets from its neighbors, so the application is in charge to handle them.

5.1.2 Gateway Node

These nodes are the similar to the internal nodes in the mesh, except that it has an additional network interface that is used to establish external communications.

The gateway node has a sniffer running on it, this is done not because it requires to listen in promiscuous mode, since all the packets are intended to arrive to every node, but to listen to its own outgoing packets.

Gateways communicate between them through broadcast packets in a special port intended for inter-gateway communications, where they send messages to calculate load allocation.

The job of a gateway is to encapsulate and de-encapsulate packets according to the pseudoheader structure used in the links.

5.1.3 Asterisk and Link Handler

In this node, traffic from FluidVoice to the Telephone Network is handled. This node has two network interfaces, as described in the diagram, which enables it to handle outbound and inbound traffic.

It establishes connections with the gateway nodes through its public network interface, receives packets, and forwards them to its own private network. Source addresses are rewritten, so nodes in both sides of the network believe that they are in the same subnet.

This address translation is not a requirement, since we could assign same subnet addresses to both sides of the mesh, however to keep order of the different internal networks it may be useful to have them uniquely identified.

Duplicate packet handling is also done in this node. Packet identification must be done in an efficient way, since the arrival rate is relatively high and that can cause it to be the bottleneck of the system. For this reasons, a hash is computed over the packet and stored as a reference. The probability of hash collisions is low enough that we can consider it none.

This node also acts as an Asterisk server, which handles call connections with the telephone network. The task of handling calls requires the node to do audio transcoding in real time, which han be processor intensive.

5.1.4 Telephone Proxy

The conversion between SIP/RTP format to the one used in FluidVoice is made in this part of the system. The underlying platform used is Shtoom, which is originally intended to be a VoIP client with Speex capabilities.

An instance of the proxy running for each extension in the telephone network. On initialization, the proxy registers with the Asterisk-NAT server, so in case of an incoming call, it is redirected to it.

When an incoming call is established, the proxy begins sending beacon signals about the availability of the node. It will translate G.771 format to Speex coded packets and add VoiceMesh headers to them.

Conversion in the other direction is also performed by this entity, where Speex packets are converted to RTP format, so it can be delivered to Asterisk, that subsequently forwads it to the PBX and it can be converted to voltage amplitudes that the telephone understands.

5.1.5 Telephone

This is a plain telephone node, the extension was provided by MIT IS&T Department, and the service they provide is a translation between conventional analog lines to packetized networks that are forwarded to our Asterisk server. The output is the typical SIP/RTP format, which is a widely known standard for VoIP. The coding used is G771, which uses μLaw.

5.2 Flow Identification and Isolation

For the routing algorithm to make a decision whether or not to forward a certain packet, it requires to know to which conversation does that packet belong to. Traffic fingerprinting can be a challenging subject since it is necessary to find which parameters of the packet reflect its identity.

Popular approaches make use of unsupervised machine learning techniques to classify flows that require big amounts of packets to be processed before a good decision can be made [15]. These approaches are useful when the type of traffic is unknown and can vary unexpectedly.

Unsupervised techniques start by selecting some parameters that are believed to be distinctive of the class and preserved over time, such as connection ports, and begin to do clustering.

This is useful for traffic generated by applications that choose random ports on run time to send information. Common applications are the ones running over TCP, which by nature will spawn the next available port for every instance of the original service.

For our particular application, since the application runs over UDP packets, most of the communication happens in a well defined set of ports, so after trying some combinations of parameters, the most successful one was the Source IP Address + Data port. A SHA1 hash is computed over the concatenation of the bytes that conform these fields.

The probability of hash collision in SHA1 is about 2^{-80}, which can be considered essentially zero for the current application. By using this particular flow fingerprinting algorithm, we are restricted to have at most one flow per device, which works without problems right now, but if a future application has another configuration that makes this flow identification useless, more information can be added or removed from the hash, to reflect uniqueness in the flow, and nothing else needs to be changed in the routing algorithm.

5.3 InterGateway Messaging Protocol

In this section we will provide the implementation details of the inter-gateway protocol, which was described in the algorithm section.

This protocol runs over UDP, so no state is kept in the network stack. This makes it much more convenient to program.

Every gateway node sends a periodic beacon signal, the period T_b is set to be one second for the implementation. The value was chosen by considering that such a timeout due to lost packets is significant enough to be noticed by the end user. This value can be increased or decreased depending on the rate of the packets arriving from the internal nodes.

Considering that voice packets are sent every 20ms, the amount of traffic forwarded before every consensus time is about 50 packets. This value is big enough to notice when a packet was failed to deliver because of a missing gateway, and not because of a random intermission in the network.

The structure of the packet is shown 5-2. The beacons contain the following fields: Gateway ID, current capacity, and finally a list with FlowID, timestamp pairs.

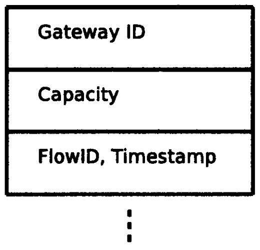

Figure 5-2: Generic Beacon Packet Structure

In FluidVoice we are relying on the UDP protcol to check for errors in packets, so we do not need to add any checksum into the beacon signal, and can consider the packet delivery an atomic operation, duplication or loss of a packet gets handled elsewhere.

One problem that happens in communications is how to synchronize the clocks between all the nodes. This problem was solved by considering some assumptions that often hold for wireless networks inside a local network.

By enabling RTS/CTS handshake and pseudo-broadcast [11] inside the wireless network, packet collision can be reduced to a rare case, even for broadcast packets. This is very important specially for UDP packets, where retransmissions do not happen unless the protocol running over it takes into account lost packets. The delay of the packet then depends on the amount of traffic circulating around the network. This delay is in the order of milliseconds, and is small compared to the beacon interval time.

So we can assume that beacons come spaced about T_b, with small variations due to packet delivery delay. Since it is impractical to send acknoledgement packets back to the sender, it is necessary to define an epoch for which all the beacons received during that period of time will be added up to calculate the traffica allocation.

After a beacon has been sent, the epoch starts, and ends just before the next beacon is sent. Just after sending packages, the gateway will be listening for incoming beacon signals from other nodes, and add them to an array that holds several beacon signals.

The array for holding incoming beacon signals needs to be two epochs long, this is done because it is necessary to consider to gateways sending beacons to each other at the same time. There is no problem of lost packets since the MAC protocol will take care of that. But if two packets from the same gateway come in the same epoch, that means that the second one is too close to the end of the epoch and belongs to the next epoch rather than to the current one. This is illustrated in figure 5-3.

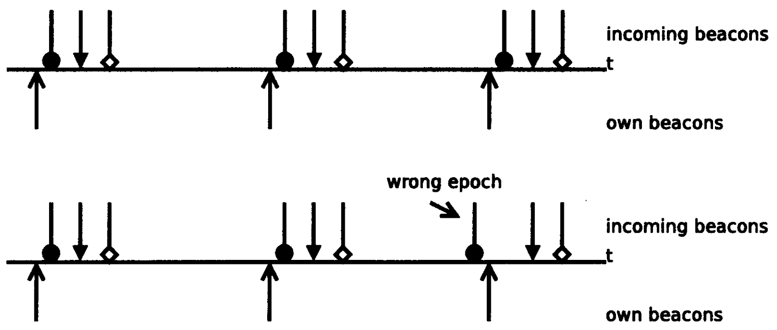

Figure 5-3: Beacon Message Timing Diagram

5.4 Packet Header Compression

For voice applications the average packet size is small, and sent in a relatively high rate. This packet size is comparable to the size of the headers, so including a poorly designed header can easily increase the amount of bandwidth required in the link.

Some of the links considered in the design have a limited amount of bandwidth, to the point that the smallest ones can hold exactly one voice conversation coded using a proprietary codec. Under these conditions, IP header compression together with an efficient packet encapsulation are required for the link to work.

In this particular application we are using UDP packets with fragmentation disabled. We also know that packets are broadcasted, so the concept of a destination IP address is irrelevant.

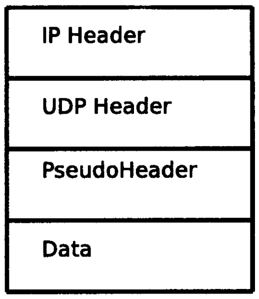

Figure 5-4: Generic Encapsulation

With this information in mind, the IP ID field can be discarded since in a local network, modern switches do not allow fragmentation, and even in the Linux IP implementation, this field is discarded and padded with zeroes.

A final simplification is that checksums are not sent with the packet, so when a packet arrives to the other end, it will be checked to have a consistent structure, such as having expected values in each one of the fields. If the packet seems to be correct, an UDP checksum will be calculated over it. In case this packet was really erroneous, higher layer protocols will notice that and take care using their own error correction mechanisms.

Some of the links that compose the system are services that work behind a NAT, so in order to apply the NAT traversal algorithms, it is necessary to use UDP packets. The current implementation is done by encapsulating IP packets over UDP ones. Taking into account these special characteristics of the packets that the system is going to handle, a smaller pseudoheader can be built, thus reducing the overhead. The general aspects of the encapsulation are shown in 5.4.

The header for outbound traffic is different from the one for inbound traffic, this is because the algorithm requires to know some information about the current state of the gateways.

The pseudoheader structure is shown in figure 5.4. The first field is the last octet of the Source IP Address, followed goes the UDP source port, which happens to be the same value for the destination port. The encapsulation in the forward path is different from the reverse one. In the path from the gateway to the external server there is an extra field about the current capacity of the link. This header is 4 bytes

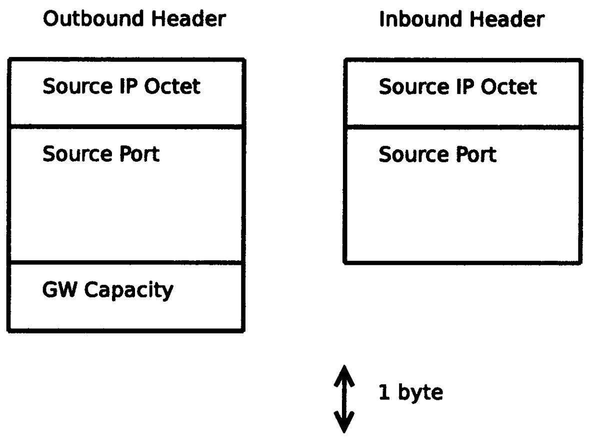

Figure 5-5: Encapsulation Header

long going outside the mesh, and 3 bytes going inside.

Considering the different audio compression factors that can be handled by the voice codec, we can have an idea of the size of the packet before header compression techniques. Some common values are shown in table 5.1.

For example, the average packet size for encoding near telephone quality using Speex compression is 44 bytes for every 20ms of sound.

Table 5.1: Overhead introduced by encapsulation

Packet Size[bytes]	Outbound Size Increase [%]	Inbound Size Increase [%]
44	8.3	6.3
22	15.4	11.5
8	33	25

5.5 NAT Traversal

The fast growing of the Internet led to a shortage of IP addresses. To deal with this problem, NAT (Network Address Translation) was invented, this method allows to connect private IP address spaces to public routable Internet addresses. This is a very common practice between organizations where there can be thousands of systems connected through a few public IP addresses.

The problem with using this kind of schemes is that hosts behind a NAT do not have true end-to-end connectivity. Services that require the initiation of connections from the outside network such as TCP, or stateless protocols such as UDP, cannot work behind a NAT.

However, the lack of bidirectional connectivity can be regarded sometimes as a "feature" instead of a limitation. This is because the NAT depends on an internal machine to initiate a session, so it can prevent malicious activity targeted to a host behind the NAT. This can enhance reliability of local systems by not allowing scans.

Some of the links that participate as alternative options to send packets through, are provided by ISP's that establish a connection behind a NAT, this is true at least for GPRS connections as well as Satellite Modems. In this case, it is necessary to punch a hole in this NATs to allow incoming and outgoing packets through it.

The nature of the data generated by the nodes is very simmilar to standard VoIP applications, and thus suffers from the same problems related to NATs. There are well known techniques to traverse NATs.

We have chosen STUN [14] as the way to traverse NAT boxes. STUN is a lightweight protocol that allows applications to discover the presence and types of NATs and firewalls between them and the public internet. It also provides the ability for applications to determine the public IP addresses allocated to them by the NAT. STUN works with most of the existing NAT boxes and does not require any special behavior from them.

Once the client running in each one of the gateways has been able to determine if there is a NAT in the middle, and what kind of NAT it is, is possible to traverse it by sending packets through the port where we are expecting to receive information. The usual configuration is a port restricted cone NAT, so it is required to send packets through the ports we want to use before the state needs to be kept. A keepalive signal is sent every minute through the NAT in case there is an absence of packets to keep the state in the NAT box.

Chapter 6

Analysis and Evaluation

In this section we analyse and evaluate the performance of the algorithm running on the gateways. We first analyse the behavior of the gateways under load variation, and provide the upper bounds for the possible variation in input traffic that the system can handle.

6.1 Gateway Behavior Analysis

The nature of the algorithm used in the set of routers sacrifice optimality for reliable packet delivery. In this section we will give an analysis of the implications of this.

The algorithm is designed in such a way that when an unknown flow arrives, it is forwarded by default. This is done until gateways coordinate after a period T_b, in the ideal case where intergateway messages are not lost.

6.1.1 Load Variation

Consider a system of M gateways that communicate intergateway messages every T_b. The total load is λ_i packets/s at $t < 0$, and a changes to λ_f for $t > 0$. This load is sent from the internal nodes of the mesh network to all the available gateways. Our interest is to predict the expected load experienced by the external link.

- If $\lambda_f > \lambda_i$. The link experiences a transient period where the usage increases from λ_i to an overshoot value $\lambda_o = \lambda_i + M(\lambda_f - \lambda_i)$. After at most T_b, the gateways reach an agreement and the final load is λ_f.

- If $\lambda_f < \lambda_i$. If the gateways reduce the amount of traffic, the effect is immediate in the link. After some period of time, the garbage collector in the gateway will notice that traffic is not being sent anymore, and will remove the allocated resources for it.

Both cases are illustrated in figure 6-1.

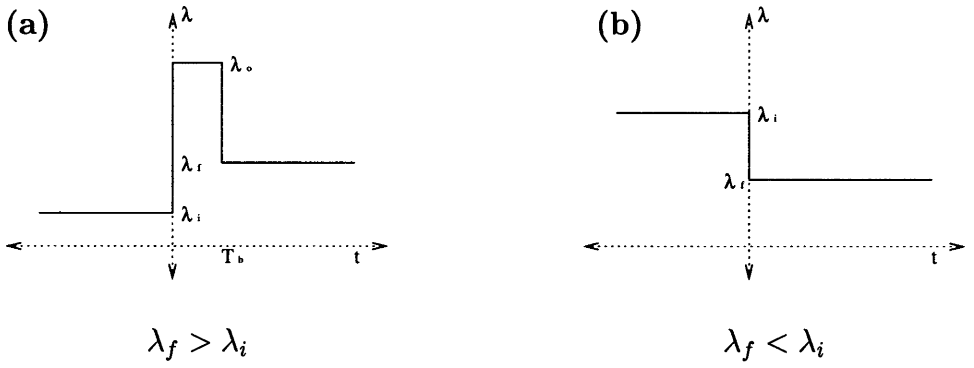

Figure 6-1: Load variation behavior

Having presented our assumptions about the nature of the variations in load, we are prepared to do an analysis of the maximal variations allowed by a certain link. The transient bandwidth usage is defined as

$$\lambda_o = \lambda_f M - \lambda_i(M-1) \tag{6.1}$$

In the limit case, where $\lambda_o = BW_{max}$, we can calculate the maximal achievable load given an initial one. A sample plot with some values of M is shown in figure 6.1.1.

For a typical case with around 10 conversations of about 20kb/s simultaneously, running in 5 gateways, and an external link of about 100Mbps. For this case, the system can handle another extra 130 conversations without overloading in the transient period.

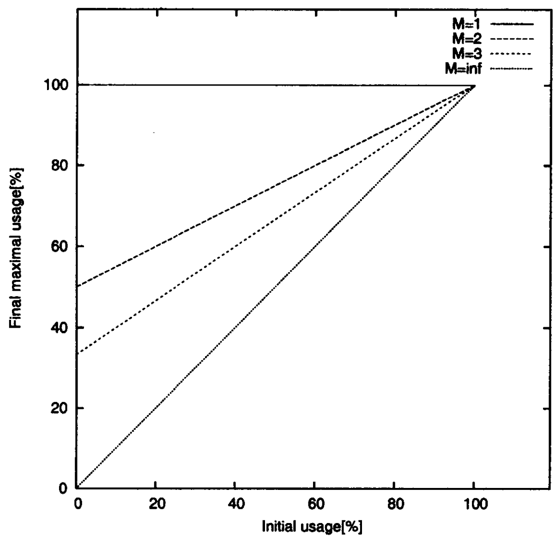

Figure 6-2: Upper bounds for load variation

6.1.2 Time to reach a stable state

The convergence time of this algorithm depends highly on the reliability of communications. Consider a gateway system of M nodes, where the probability for a node sending a message and reaching all the other nodes is p. Beacon signals are sent periodically, every T_b.

For this part of the problem, a simulation was run on a setup of M nodes. Figure 6.1.2 shows results for an interval from $p = 0.1$ to $p = 1$. We can see that the number of transmissions required to deliver the message successfully decreases as the probability is near to 1.

An interesting fact is that the amount of required transmissions does not increase significantly with the number of nodes, as the simulation shows.We can see that for low packet delivery probability, adding a large amount of nodes does not increase the number of transmissions significantly.

In out tests so far, experiments have shown that the packet delivery probability is higher than 80%, so we will focus on that region of interest. With this information, and considering that the number of gateways we work are less than 10, we can see that the expected number of transmissions before reaching all nodes is around 3, which can be mapped to expected delay during gateway reconfiguration. Such information can be very useful for a voice codec, so it can know the worst case of delay, and apply countermeasures to minimize it, such as redundant transmission of frames, or forward error correction codes (FEC), etc.

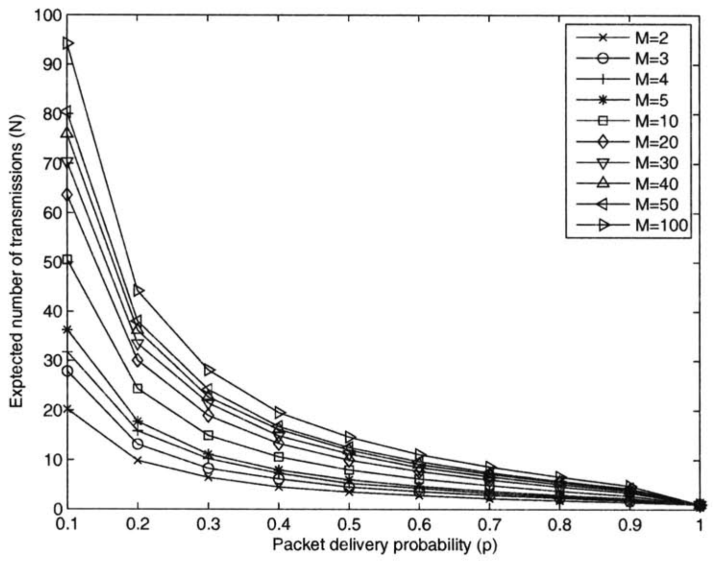

Figure 6-3: Convergence Time

6.1.3 Recovery Time

The original goal of the algorithm is to reach a stable state, and ultimately an optimal stable state. In case that there are errors, this process can imply going through suboptimal states, that can be stable or not.

In this section we will talk about how to recover from errors causing the system to go from an optimal state to a suboptimal one. This recovery is focused on delivering packets properly, and not other kinds of higher level recovery, such as Forward Error Correction codes, which interpolate information of a lost packet through their neighbors.

The most important aspect of this system is resiliency. We are focused from the very early design stages to construct a system that can tolerate failures and recover from them in the shortest amount of time.

Consider a system with M gateways, transferring packets at a λ_i rate, with one of

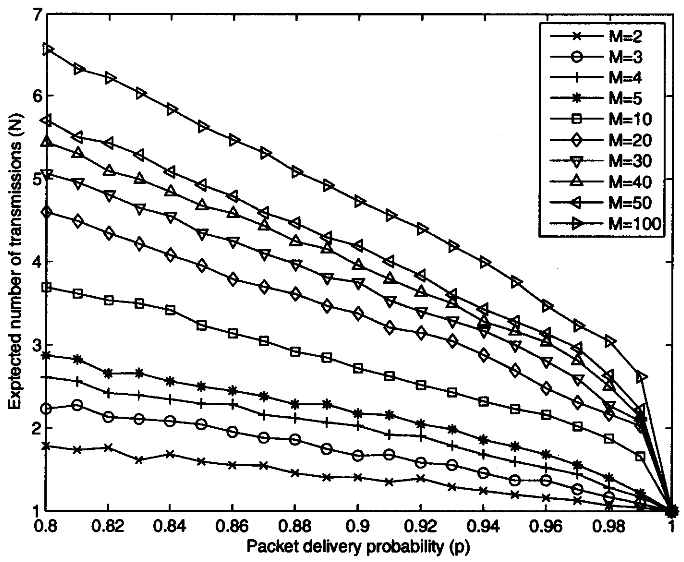

Figure 6-4: Convergence Time

them failing. We are interested to know which is the expected time before information goes back through the link, and also the time for the system to converge after this happens.

There is a periodic beacon sent every T_b by the nodes in the network. We can model the moment of failure as a uniform distribution in $0 < t < T_b$. With this information we can find out the expected time for the system to realize that one node has died, which is $T_b/2$. This is the amount of time for which traffic is lost.

The expected time it takes for the system to converge again is $3/2T_b$, because we need to wait for the next beacon to be sent. If the traffic that the lost gateway was transferring is equal to λ_m, we can predict that the overshoot will be $\lambda_o = \lambda_i + M\lambda_m$, before stabilizing to the original value λ_i.

6.1.4 Route Stability

An important aspect of a routing algorithm is how long a route follows the same path. The behavior of the routes differs from its wired counterpart in the sense that most wired networks are setup in specialized ways, where the routers are in special environments, such as data centers, with shielded connections.

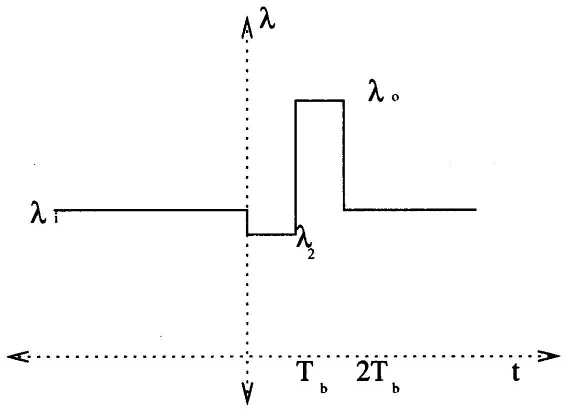

Figure 6-5: Example of Recovery

In fixed topology wired networks, routes are stable for long times [13], in the order of hours to days. This happens because once routers have solved the optimal route problem, and if there is no change in the topology, the optimal has been achieved and there is no need to switch to another configuration.

The routing problem is an search for a better solution, and the reason routes can vary if there are not drastic changes in the conditions is if a better optimal solution is found on the system. This happens for example when an optimal solution is found but more information is added to the optimization problem.

In our case, the algorithm that runs on each gateway executes a global optimization algorithm, so we can expect that routes last for a considerable amount of time unless gateways drop unexpectedly.

6.2 Gateway Measurements

In this section, we present some experimental data gathered from running probes on the network testbed. The main objective of these measurements is to see how close we hold from the theoretical analysis presented in the previous section.

The main differences between the measurements and the theoretical prediction are due to simplifications done on the model, such as no propagation time of messages, immediate response time of the systems, and perfect internal mesh behavior.

We will see that most of the model behaves, however important phenomena were observed, from which some should be added to the design considerations.

6.2.1 Gateway behavior under load variation

We present a series of measurements for the traffic observed inside the mesh network. An interesting observation of this measurements are the scalability issues for internal broadcast methods inside a wireless mesh network.

Although the packet speed ranges around 11-54Mbps, which is several orders of magnitude higher than the traffic generated by the internal nodes, packet collision happens when the amount of nodes increases.

Figure 6-6 shows plots for four independent nodes turned on sequentially. Even that the final cumulative traffic is about 20kbps, which is less than 1% of the theoretical maximal capacity of the channel, collisions start happening with very high probability.

The total traffic seen at the gateway is in figure 6-7. A possible explanation of this behavior is that nodes do not know when to send information. Packets are being broadcasted, so the concept of receiving a CTS packet does not exist.

Since 802.11 does not protect packets in broadcast mode, and UDP packets do not have reliable delivery protection in the network layer, information is sent through a much lower rate, generally at 1Mbps. This can be avoided in the Atheros chipset by changing some parameters in the driver, however this is just for experiments.

One possible way to avoid this kind of behavior, could be to send unicast packets to a node, and have the rest of them listen in promiscuous mode. With this we can guarantee that the RTS/CTS process happens, so reliable delivery can happen between two nodes, and at the same time, the other ones can listen to the conversation.

6.2.2 Link behavior under load variation

In this experiment, we setup three gateways connected to the external link, and begin running FluidVoice clients that generate traffic incrementally in random internal nodes. There is a period of time that load has to wait in order to be stabilized. These

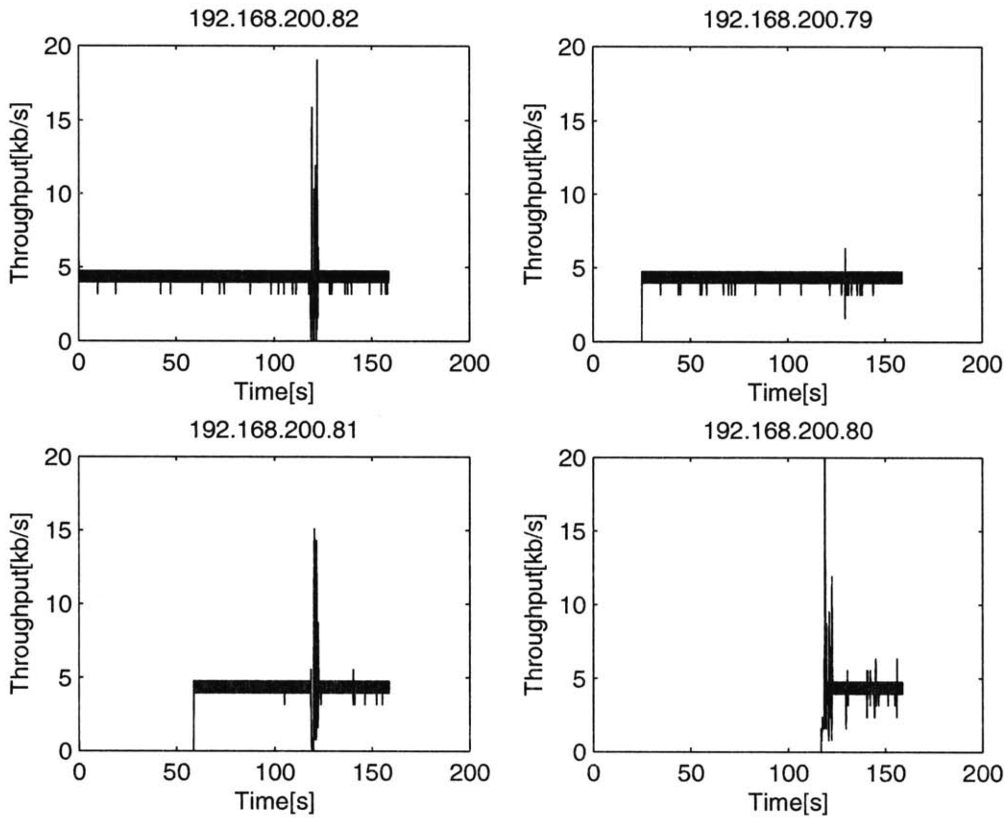

Figure 6-6: Individual Flows seen from a gateway inside the mesh

plots can show how information gets routed from one side of the mesh to another.

The algorithm acts in such a way that when unknown traffic is heard, the gateway will forward it by default, and subsequently will make an assessment of the correct allocation.

Figure 6-8 shows a plot for the traffic seen at the link handler, where packets are gathered from all the possible links that it could be connected to. The behavior is the one as expected, where a period of overload happens before stabilization.

An important observation that has to be made is that packets get ordered in the links, since the capacity of the link is much higher than the throughput generated by the flows. Ordering is accomplished because packets are sent to the link, where the probability of collision is very low because most of the time the link is idle.

Figure 6-9 shows the traffic behavior seen at the link compared to its theoretical

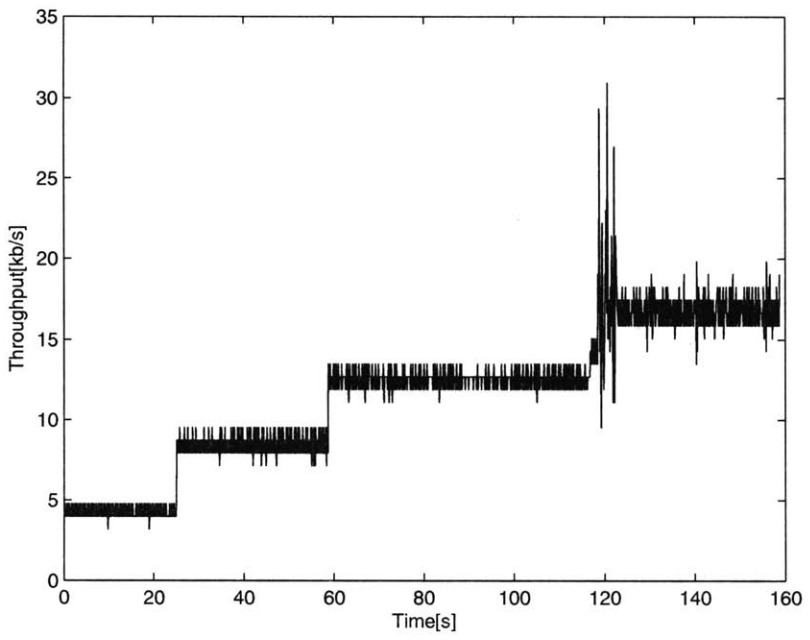

Figure 6-7: Total Flows seen from a gateway inside the mesh

behavior. The thick line is the predicted behavior and the thin line is the real one. The clock between each one of the nodes are not synchronized, so due to the small variations in response time, we can see that the result is not a perfect step function, but a low pass version of it.

We can see from the plots that the external node has to handle some extra traffic, proportional to the amount of active gateways at the time. This is the key property of the algorithm, which is its commitment to forward any unknown traffic until the nodes figure out who should really be in charge of a certain flow.

Once gateways communicate, agreement is reached, and the load perceived at the external node decreases to only the required amount. This is can be seen in the stable regions of the plot.

If we measure traffic in the side of the network where the external node has already filtered out duplicate packets, we can see that increase in network load occurs in a much regular way, just as predicted, consecutive incremental step functions, without the overshoot seen due to the redundant external traffic. This can be observed in figure 6-10.

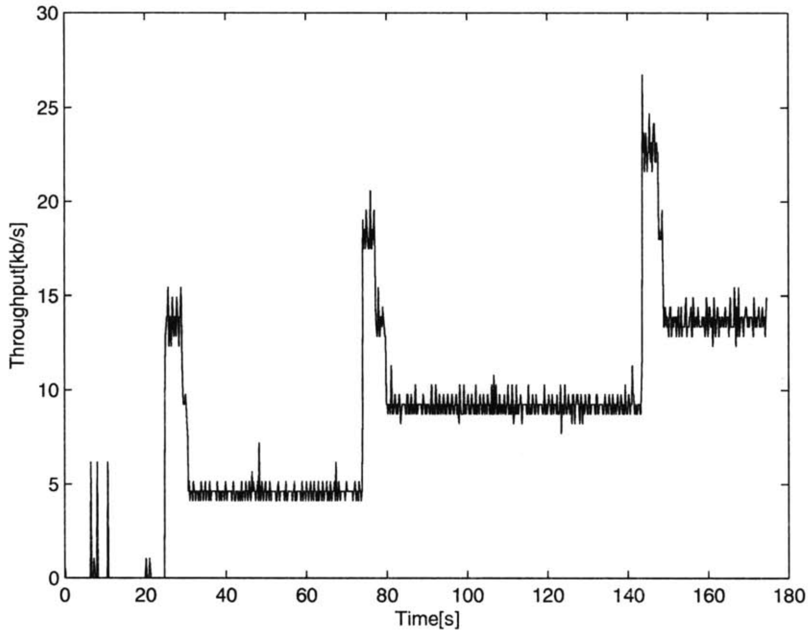

Figure 6-8: Load Variation seen at the Link

One fundamental parameter for this traffic filtering to happen is the size of the queue used to handle packets, as well as having an efficient implementation of the filter. The delay in packet processing can affect dramatically the output response for the link handler, adding delay and jitter to the packets as they cross the link.

The range there this kind of behavior can happen is limited by the maximum bandwidth of the link, and we have seen that the theoretical limits, or maximum values specified by the manufacturer are not the values that should be used for design. We need to take into account the speed of the intermediate router, as well as the nature of the traffic that is going to be forwarded. If it does not count with collision avoidance mechanisms, which is very common for broadcast communications, this can deteriorate severely performance.

6.2.3 Gateway Failure

The main objective of the project is to build a resilient gateway system, which can perform packet forwarding under severe conditions.

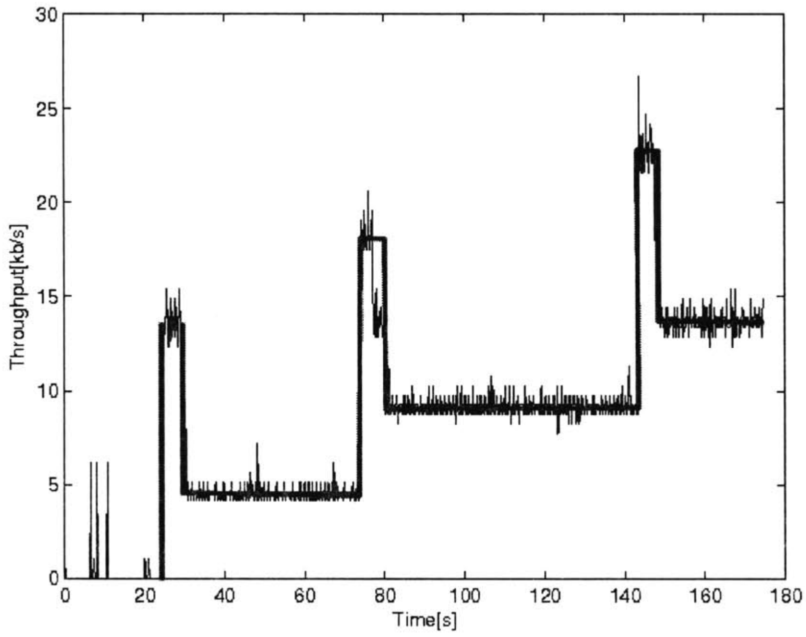

Figure 6-9: Load Variation seen at the Link, compared to the Ideal Behavior

This made the design to be focused on automatic recovery and self healing.

The sources for packet failure can be gateway death, or miscommunication between live gateways. In either case, the rest of the network tries to take care as good as possible of the remaining links.

In figure 6-11 the gateways receive load incrementally, which can be observed in the peak variations. But due to miscommunications between the gateways in the mesh, one gateway believes that another one is dead, so it will take care of its traffic, generating the sudden increase in traffic.

Once communication is re-established, load stabilizes and the system behaves normally.

6.2.4 Packet delivery probability

In this section we will describe some interesting phenomena observed in the experiment run. These are implementation dependent, and need to be considered for any future implementation of a similar system.

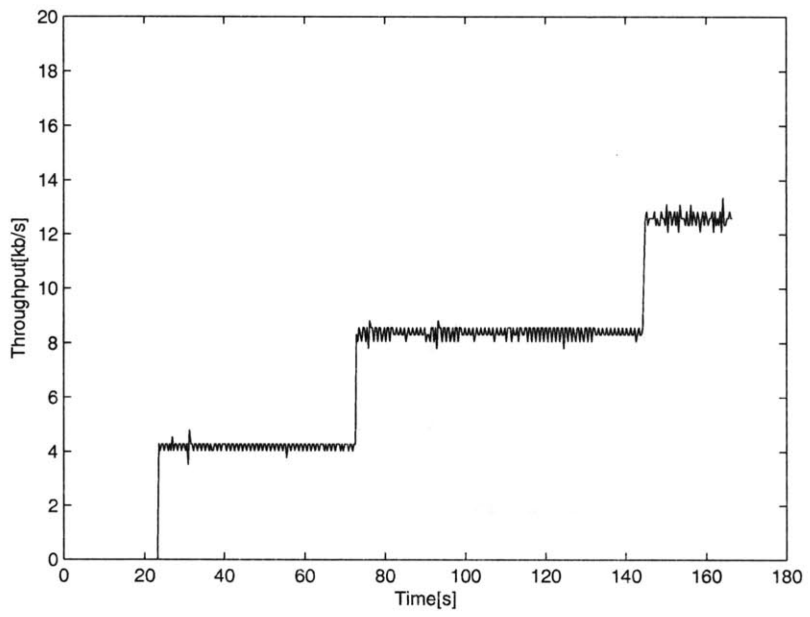

Figure 6-10: Load Variation seen in the external Network

Figure 6-12 shows a typical increase in bandwidth, and the behavior of the gateways. This behavior is the one expected in the theoretical analysis, and is very similar to the plots describing gateway load increase.

However, in this case the amount of time it takes to converge is bigger than one period of beacon broadcast. In graph 6-11 it is very clear that the number of coordination messages that have to be sent are two, which corresponds to about 85% of reliability in message delivery, as calculated in the simulations for packet delivery probability.

One application of this could be that we could set convergence time based on measurements of network conditions. With this, we could do what network weather tools predict. This information could be made explicit to higher network layer, and they could be able to take advantage of this.

6.2.5 Packet Delay

One of the main goals of this algorithm is to minimize the amount of delay introduced by adding the load balancing mechanism to the network.

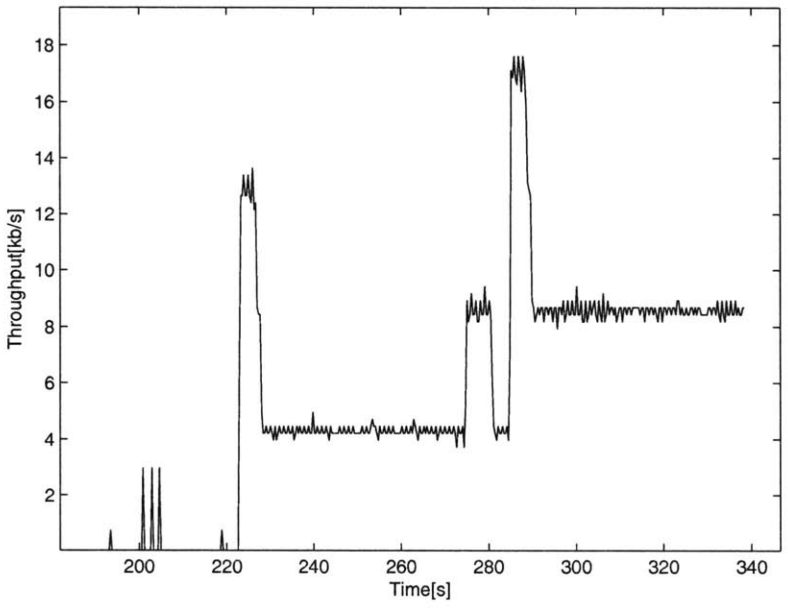

Figure 6-11: Recovery

The amount of delay caused by the gateway loss and reconfiguration was analyzed in the previous section. Here we will study the behavior of the system in stable conditions, by that I mean connections are routed through the same path during the experiment.

The performed test was the conventional Round Trip Time(RTT) test, in which a packet with a timestamp is sent from a machine inside the mesh, and reaches a machine outside the network, and this machine resends it back. Once the packet is received, its timestamp is compared with the reception time, and this value was measured.

The measurements are shown in figure 6-13. The values are RTT against Trial Number. The RTT experienced by the application is in the order of 20ms. This value is acceptable for real-time two-way voice communications, much less than the 80-100 msec tolerance considered acceptable in the Bell Telephone network.

In a comparable wired network, the RTT experienced between two hosts in the same AS is in the order of 0.5ms, which can be neglible,and between single hop wireless networks, the amount of delay is also around 0.5ms. With this knowledge,

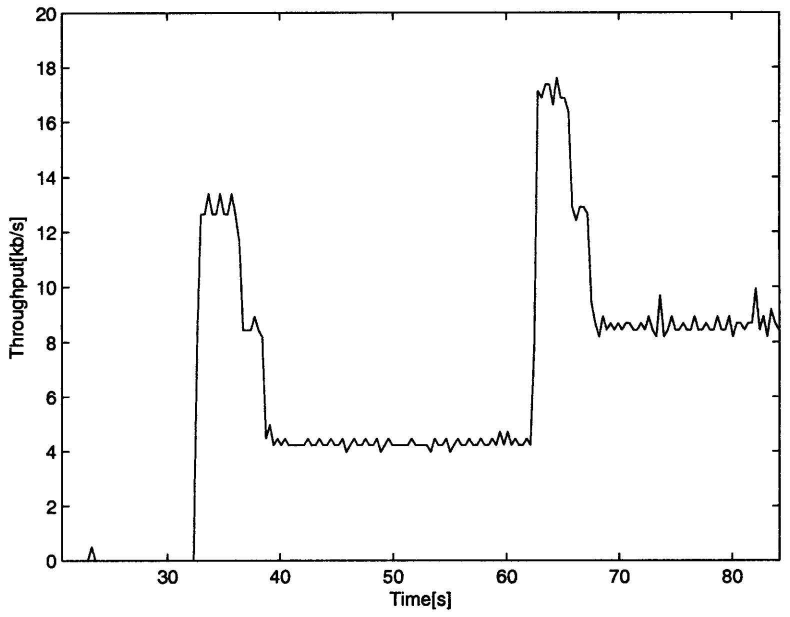

Figure 6-12: Individual Flows

we can conclude that the major component of the delay is due to the gateway code. Any variation shown is ideally due to the processing overhead in the gateway network.

In these set of measurements, even that points are spreaded due to random noise in the environment, there is a clear tendency to increase the lower bound linearly.

Even that the slope of this bound is relatively small, and may not interfere with the overall behavior of the system, it is clear that it reflects the existence of some state in the gateway system.

The system was measured in a stable state. We arranged this by running the system configuration for about one day before measurements were made. This was done to make sure that the errors introduced by transient states could be discarded. The setup includes four nodes transmitting information and three acting as gateways too.

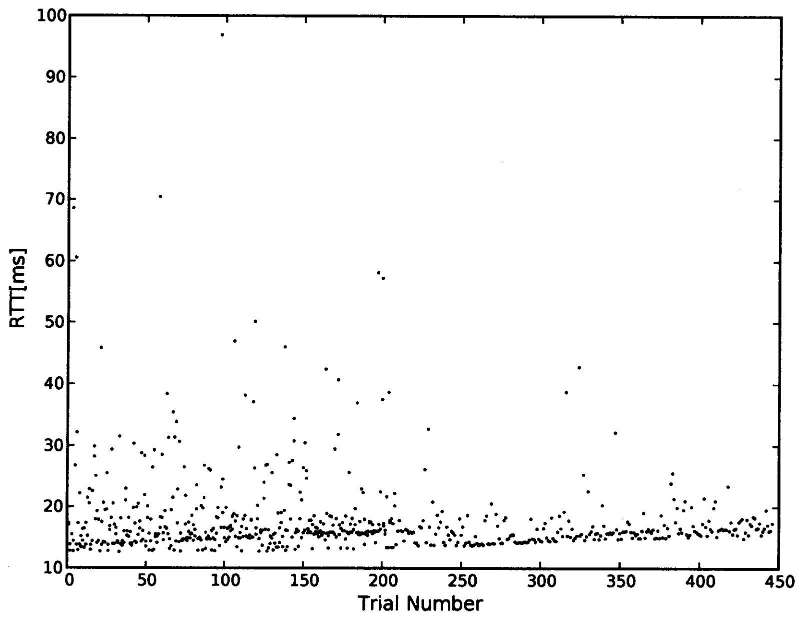

Figure 6-13: Round Trip Time $\mu = 17.37[ms], sd = 5.67[ms]$

Chapter 7

Conclusions and Future Work

This thesis reports on the design and implementation of a distributed algorithm for connecting mesh radio networks using multiple available gateways. Our goal is to be resilient, robust, and efficient. This setup of redundant gateways aims to provide an overall reliable link between two Autonomous Systems.

Reliability and resiliency are the primary goals, given that any of the underlying links may fail unexpectedly, and their connectivity to the mesh may vary substantially.

The objective of providing a reliable system has been achieved, as shown in the analysis and experimental section, where it was shown how the algorithm works under some of the most common scenarios.

Gateway failure was handled successfully by the algorithm, as we can show in the experimental measurements. The amount of downtime is around 1 to 3 seconds, and this his depending on the period for broadcast reconfiguration. It can be lowered down to even smaller values, but the stability and convergence of the system can be threatened.

Internal network failure was studied through simulations, resulting that in order for a message to reach all the other nodes in the mesh, it is required to send packets around 2 times in the average case for 85% packet delivery rate in a network consisting of around 10 nodes. With this value we could tune the intergateway beacon period so we can control the outage time. This outage time resulted to be of 1 to 3 seconds depending on the network conditions.

The traditional approach to provide redundancy in big systems is to have a replacement substitute a faulty node and perform its task, and in a hierarchical system such as in master-slave ones, specific replacers are required. In our approach, we are using a flat approach, where any node can be replaced any other one. This approach can bring more robustness to distributed systems, specially the small ones, where having hierarchical structure can be a waste of resources.

7.1 Measurements

The measurements done show that depending on network conditions, packet delivery probability can vary substantially, making delivery rate and probability play an important role in the performance of the algorithm. The algorithm can/may be in a suboptimal state in the case of partial communications. Under the most common or normal conditions, the transient overload period after a crowd of new communications or lost gateway, where the bandwidth consumption increases suddenly, would last one beacon period. In the other case, where communications are flaky, and some beacon packets are lost, bandwidth consumption will be more than the optimal, but for the end user, the problem would be imperceptible.

The parameters that this system optimizes are the packet delivery latency and system downtime. The average packet latency of the algorithm in a typical setup is about 15ms to 20ms, which is much higher than a conventional wired link, which is about 0.5ms. Even that this latency is several times bigger than without adding the system, it is still acceptable for real time communications, and it is probably caused by implementing it on Python. The system downtime is in the order of seconds, as stated before, and this when gateways fail to forward traffic. In the case of lost beacon packets, the effect is not seen as a lost packet, but as some extra use of bandwidth, proportional to the number of Stargates that realized that a certain flow is unattended.

The changes in the networking stack were minimal for the internal nodes, which are just required to broadcast packets by default. For the Stargate nodes, the only difference was that they were taking the relevant parameters of the headers to construct their own pseudo header based over IP/UDP. The overhead introduced by adding a tunneling layer into the packets for forwarding through the links is in the range of 5% to 35%, depending on the packet size. If packets are very small, they are comparable to the size of the header of the protocol where they are encapsulated. Some of the fields in the IP header are redundant, so just the useful information was extracted to construct a pseudo-header.

This algorithm has the advantage of doing a distributed allocation given a partial or total input to each one of the gateway nodes. The downside of this is that during recovery time traffic grows proportional to the number of participating gateway nodes, that quickly slows down to only the required amount of traffic when agreement has been achieved. The time for this traffic to slow down depends on the packet delivery probability, and in our case is about two or three gateway beacon periods.

Stability of the routes were measured as the number of times a flow had to change the gateway it is using. The stability of the allocation depends directly on the availability of gateways. Once a flow is taken by a certain gateway, it will remain allocated there until the gateway disappears, or there is a miscommunication between gateway nodes. In that case, a route will be replicated by other nodes, and once communication is reestablished, allocation will go back to the original node that was forwarding traffic.

Traffic fingerprinting and flow identification were done by taking IP addresses and ports. This could be done since traffic flows in very well defined ports and do not tend to change. However in more complicated cases such as encrypted traffic, or multiple clients running on the same machine, more complex ways to fingerprint needs to be done.

7.2 Lessons Learned

The main lesson learned from this thesis is the fact of isolating a specific problem and solve it, instead of trying to solve all the small side problems that increase performance in a small fraction. This is true specially for the gateway problem, where it was very important to set a limit on the problem, and just solve the necessary networking problems required to have a working solution.

As an important implementation issue, it was seen that even using an interpreted language such as Python, the development time was decreased drastically without a substantial performance sacrifice. But for future applications on a more larger scale development or a production development, it would be better to write the primitives in a compilable language, and use Python bindings to have the flexibility to test different approaches without sacrificing performance.

7.3 Future work

As future work, it would be interesting to add some unsupervised machine learning techniques to analyze more complex traffic. In this case, ports do not vary, but in a broader range of applications, it would be critical to know which packets belong to the same logical entity, so they can be treated atomically.

A very important issue that has not been considered in the design of this algorithm is scalability. This algorithm clearly does not scale very well for a large number of gateways. The original goal of the problem is to provide reliable communications through any available node. This still works very well in practical setups, where no more than a couple of gateways are present, such as in our experiments, where there were about three to four gateways. However, in a bigger system, such as a campus scale network with dozens of links with the external world, overhead introduced by gateway coordination could degrade performance severely.

A more efficient way to solve the problem would be to create several internal multicast trees, so routing would be performed under a geographical basis depending on parameters such as position or signal strength. This implies integrating the mesh network with the gateway network, using the same internal routing capabilities on both types of nodes. With this assumption, load could be balanced between several gateways, without overloading the links during transient periods. The downside of this is that in case of gateway failure or intermediate node dead, the time to reconstruct the tree could cause lost packets.

This prototype was built for routing traffic generated by FluidVoice, which is a subset of the traffic that networks can carry. As a more ambitious project, it would be interesting to explore ways to handle general Internet traffic, and try to couple them to higher layer protocols like TCP, which try to provide fairness and handle congestion.

Bibliography

[1] Asterisk. `http://www.asterisk.org/`.

[2] Seattle wireless. `http://www.seattlewireless.org/`.

[3] Anthony Baxter. Scripting language my arse: Using Python for Voice over IP. Technical report, 2004.

[4] Dimitri P. Bertsekas and Robert G. Gallagher. *Data Networks.* Prentice-Hall, 1987.

[5] Vaduvur Bharghavan, Alan J. Demers, Scott Shenker, and Lixia Zhang. MACAW: A media access protocol for wireless LAN's. In *SIGCOMM*, pages 212–225, 1994.

[6] John Bicket, Daniel Aguayo, Sanjit Biswas, and Robert Morris. Architecture and evaluation of an unplanned 802.11b mesh network. In *Proceedings of the 11th ACM International Conference on Mobile Computing and Networking (MobiCom '05)*, Cologne, Germany, August 2005.

[7] Sanjit Biswas and Robert Morris. Opportunistic routing in Multi-Hop wireless networks. In *Proceedings of the ACM SIGCOMM '05 Conference*, Philadelphia, Pennsylvania, August 2005.

[8] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A performance comparison of multi-hop wireless Ad-Hoc network routing protocols. In *Mobile Computing and Networking*, pages 85–97, 1998.

[9] Ian D. Chakeres and Elizabeth M. Belding-Royer. AODV Routing Protocol Implementation Design. In *Proceedings of the International Workshop on Wireless Ad-Hoc Networking (WWAN)*, pages 698–703, Tokyo, Japan, March 2004.

[10] David B. Johnson, David A. Maltz, and Yih-Chun Hu. The dynamic source routing protocol for mobile ad hoc networks (dsr). Internet-draft, IETF MANET Working Group, July 2004. Expiration: January 2005.

[11] Sachin Katti, Dina Katabi, Wenjun Hu, Hariharan Rahul, and Muriel Medard. The importance of being opportunistic: Practical network coding for wireless environments. Technical report, MIT, 2005.

[12] Vincent D. Park and M. Scott Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *IEEE Conference on Computer Communications, INFOCOM'97, April 7-11, 1997, Kobe, Japan*, volume 3, pages 1405–1413. IEEE, IEEE, April 1997.

[13] Vern Paxson. End-to-end routing behavior in the Internet. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, volume 26,4 of *ACM SIGCOMM Computer Communication Review*, pages 25–38, New York, August 1996. ACM Press.

[14] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. STUN - simple traversal of user datagram protocol (UDP) through network address translators (NATs). RFC 3489, IETF, March 2003.

[15] Grenville Armitage Sebastian Zander, Thuy Nguyen. Self-learning IP traffic classification based on statistical flow characteristics, 2005.

[16] Jean-Marc Valin and Christopher Montgomery. Improved noise weighting in CELP coding of speech - applying the vorbis psychoacoustic model to speex. In *Audio Engineering Society*, May 2006.

[17] Wensong Zhang. Linux Virtual Server for scalable network services. Technical report, National Laboratory for Parallel & Distributed Processing.