Argos Final Document

Ben Kaap, Shailesh Kochhar, Scott Stuttle

May 7, 2004

Contents

List of Figures

Chapter 1

Background

This project has been brought to you
by the number 802.11, the letter B,
and users *just like you*.

Team 1

1.1 What is Argos about?

Imagine – if you will – a situation where you've got a wireless network that is exhibiting very strange behavior: packet loss is abundant, signals get dropped, nodes become silent for minutes at a time, even in the face of a lot of traffic. You have a feeling something is amiss in your network setup, so you spend hours poring through packet logs measuring in the hundreds of megabytes, possibly even gigabytes. You end up manually comparing multiple files to each other, line by line, byte by byte, in order to glean any understanding of what the symptoms are, let alone the solution to your problem. Sound fun?

Welcome to the world of your average wireless network administrator. Before ARGOS[1], that was the only way to diagnose problem areas. Sure, "regular" network administrators had tools like Ethereal[1] to work with, but none of those tools are catered towards the up-and-coming, wave-of-the-future **802.11**[2] trend. We plan to change that.

1.2 Argos is a One Of A Kind

After scouring the Internet for any sort of tool that would aid us in our quest to make life easier for the aforementioned administrator, we came up empty. This fact is both blessing and curse:

- On one hand, it means we have no working codebase to build upon and improve to add our own features and functionality into. This means that all of the code that is going to be a part of the project has to be written by us and is not already out there.

- On the other hand, it means we're also not constrained into whatever language choice/programming paradigm/existing featureset that is already in a pre-existing

[1]Argos Panoptes (meaning 'All-Seeing') was a demi-god from Greek mythology. He was a herdsman and his body was covered with eyes, so he was considered "the watcher." Hera commissioned him to ensure that Io, a beautiful woman from the city – who was later turned into a black and white heifer, which is why ARGOS was the perfect person/demi-god to watch over her – wouldn't be seduced by her husband Zeus. Much like ARGOS was a "watcher" who was charged with the task of watching **Io**, *our* ARGOS is designed to watch over network traffic, or – in other words – **I/O**.

[2]Note: This text, like all other text in **bold-faced** type is defined in glossary

code base. This freedom allows us to ensure that everything is done "right" the first time, and does not[3] require us to spend a lot of time maintaining ugly or broken code. Besides, no one really wants to deal with and maintain 3 year old **COBOL** code, for example.

As we will soon see, ARGOS aims to be the link between administrator and network. ARGOS facilitates in-depth analysis of all aspects of network traffic through a sophisticated yet easy-to-use interface.

[3]Thankfully.

Chapter 2

Requirements

Good, Fast, Cheap... Pick two.

Unknown

2.1 Introduction

Argos[1] is a 802.11b[2] tool that analyzes wireless network traffic. Its goal is to monitor network traffic on the 2.4GHz radio frequency in order to help the user analyze traffic patterns and thus optimize performance. Argos will be delivered to the customer at the conclusion of the project both in executable and source code form. This should hopefully allow for Argos to be quickly equipped to perform its task. This quick deployment is another informal requirement of the project, since the area of wireless network management is growing more important with each passing day.

2.2 What is Argos?

We designed Argos to optimize a wireless network in terms of the interference it experiences. This interference can be both internal and external, as well. Internal interference is interference that is caused by the nodes inside the network. When individual nodes have too much power, their signal can travel much further than it needs to and be overheard by nodes that shouldn't be receiving them. Once these eavesdropping nodes hear the traffic, they release the medium and wait until it is open again. This causes unnecessary lag in transmission, since this waiting period should not be happening. External interference is similar in that there is waiting that is unnecessary, but is caused by nodes in other surrounding networks. This distinction between internal and external *is* necessary, however. Although little can be done to remedy external interference, much can be done to fix the problem of internal interference, since the administrator controls both ends of the transmission.

Over a user-specified interval of time, Argos can convert **TCPDump** log files of network traffic into a graphical visualization. This visualization will not only allow an at-a-glance perspective of network traffic and performance, it will also precipitate in- depth analysis of

[1]nee "Monitoring the CU wireless system"

[2]the IEEE standard for wireless network communication, operating at a frequency of 2.4GHz with a maximum throughput of 11Mb/s

packet transfer and specifically problematic areas. The visualization will also allow the user to synchronize network logs that were collected. This synchronization helps the user to get a very accurate image of past network traffic.

The customer expects us to have a usable starting point with the ability to analyze packets and display the necessary data. The customer does not expect that all features be implemented however, so long as the potential to do so is there.

2.3 Unique Benefits

ARGOS will allow our customer to adequately and efficiently monitor and analyze traffic on the wireless network. Other tools exist to *discover* and *join* wireless networks, but none exist whose sole purpose is to ensure satisfactory performance by taking into account outside variables. The simple fact is that the ARGOS project is on the forefront of modern networking technology, and is attempting to bring something new to the table that has been previously unheard-of in the area of 802.11.

The major benefit of ARGOS is that it will allow the customer to quickly visualize network traffic in such a way that precipitates an easy understanding of the big picture. It is an invaluable asset to be able to see – at a glance – a sequence of communications between two nodes in terms of the packets transmitted, received, and replied-to. Without this graphical overview of the traffic, the user must resort to looking at those million-line binary packet logs from multiple sources to try and get a mental picture of the situation without any assistance or automation.

Once again, there are crude brute-force tools that already exist for the more standard ethernet architecture, but 802.11 is uncharted territory in this area. In fact, the tools available for ethernet often have so few features that they're not very useful to diagnose standard networking problems, either. Most end up being glorified **packet sniffers** capable of little more than just a slightly more user-friendly version of the text provided by TCPDump's standard output.

It is this graphical traffic visualization alone that makes Argos truly unique.

2.4 What Problem Does Argos Solve?

The Champaign-Urbana Wireless (CUW) Network is having problems with the speed and reliability of its 802.11 network. After some in-depth inspection of these problems, they've concluded that they fall under several categories. These categories are: outside traffic, internal network crosstalk, and packet collisions. All of these problems can be properly diagnosed and solved using Argos and the visualization it provides.

Outside traffic is a problem when other local networks are transmitting on similar frequencies without any or with insufficient courtesy to the CUW network. These sources need not be simply computers operating "outside" of the CUW network; they may be entirely unrelated to wireless networking as a whole. Some examples may include portable phones or even microwave ovens.

These signals are being heard by CUW's network and create contention . When this courtesy is not returned it results in an inability for CUW's network to find space in the now-congested traffic to communicate. This behavior can be diagnosed by noticing that logs have large periods without traffic, even though attempts are made by individual nodes to clear the channel and send data to each other.

Internal network crosstalk occurs when there are more than two nodes on a network. Each node need only be close enough to communicate with one other node on the network. This allows for the possibility that one node may not be able to 'hear' one or more nodes and can try to send messages at the same time. This can result in corrupted data received or loss of entire transmissions. Crosstalk can also create deadlock situations, where a chain of nodes exists and each node is waiting for the next to transmit because the previous node has no idea the other nodes exist. This problem is exhibited in Argos by the fact that packets from particular nodes are visible in the logs of other nodes, but the responses to those packets are not.

Packet collisions occur when two nodes on a single network try to send data simultaneously. This differs from internal network crosstalk in that these two nodes are generally within 'hearing' distance. The problem that arises is that wireless network nodes can have long pauses in transmissions while they wait for another node to complete its communication. This results in very slow transmission rates. Packet collisions are even more clear when Argos is being utilized. If both problem nodes are being examined simultaneously in Argos the colliding packets are clearly visible and in competition by their locations on the timeline. Since the timeline is divided clearly by the desired unit, any packets sent in the same location on their timelines are obviously being sent at the same time in reality.

2.5 How Does Argos Solve the Problem?

Through data extraction from TCPDump logs, and information extrapolation, Argos presents a visualization of network traffic. This allows the user to quickly identify the source of each problem. By showing information regarding past and current traffic patterns[3] in microsecond detail, Argos pinpoints problem areas to minimize the costs by eliminating ineffective solutions.

With side-by-side display of packet transmissions, data transmissions and receptions, it will be easy to identify areas of prolific packet collisions. Areas where there is data waiting to be sent and there isn't any traffic on the network will illustrate outside traffic issues. Being able to change which nodes are being viewed can aid in the resolution of internal

[3]information such as packet order, number of packets, wait times, transmission times

network crosstalk.

Although the final step in solving these network problems is modifications done to the network infrastructure by the user, Argos will be a valuable tool in the problem identification process.

2.6 Who Will Use Argos?

Our proposed users would be people like network system administrators. Anyone who has an interest in seeing how they can enhance network performance and has access to sufficient traffic logs will find Argos worthwhile.

This can include everyone from the average computing hobbyist who would like to set up a wireless network around his own home to a coffee shop owner who would like to provide wireless internet access to his customers to a highly- paid consultant contracted to solve a walk-up wireless computing problem for a large university or corporation.

However, to truly delve deeply into the more advanced capabilities of the system and actually be able to *use* the information presented by Argos, one would have to be an individual who is very experienced in the area of networking, especially that of wireless networking.

2.7 Usage Principles/Constraints

Argos should be capable of handling nearly any and all traffic data, as long as it's of an acceptable format. An "acceptable format" is simply any standardized format that includes all of the information that is necessary to accurately examine the traffic at the level of detail desired.

Regardless of whether the data is valid or logical, Argos should handle it gracefully. Its job is to display data, not analyze it itself. That's the user's responsibility. It is in this way that Argos keeps a tight rein on the scope of its usage. Since Argos provides simply more information related to the situation, this more hands-off approach keeps it generic enough to not hamper problem-solving with any artificial bias.

Argos also needs to be as intuitive as possible in order to increase the ease-of-use factor for the user. It is intended to be a relatively transparent helper that allows the user to get the information he/she desires without intruding.

2.8 Input

The eventual goal of Argos will be to automatically perform a TCPDump of the **AP**s which the user wishes to monitor. However, although the world of wireless networking is constantly and quickly changing, the capability of using an AP to both perform an active role in the network as well as collect traffic data is still a ways off.

Therefore, Argos initially requires the user to perform these dumps manually and properly locate the data according to a pre-defined structure.

As long as the input is in a standardized form, it will be accepted by Argos, regardless of correctness, again due to the role Argos plays in the process of problem-solving.

Also, the user will be able to load pre-constructed saved-state files which allow for the user to view specific scenarios saved previously. That way, the time spent to evaluate a certain situation won't have to be wasted, and any progress already made will still be there when the scenario is loaded.

2.9 Output

The final and overall output of this program will be a graphic visualization of the packets at each AP. Each AP will have its own 'timeline' which will display the incoming and outgoing packets. The user will be able to manipulate the packets to account for any erroneous data. Since this visualization is the bulk of the project from both development and user standpoints, we will explain the specifics in more detail.

Each packet (or packet-like unit) is represented by a shape decided by its packet type and then placed chronologically on a linear timeline. If more than one APs output is desired, another timeline is added below the previously selected AP(s).

Since there is no real way to ensure proper temporal synchronization between APs (because of hardware intricacies in some cases, and idiosyncrasies of the protocol in others), these timelines are capable of being dilated or compressed, as well as shifted. These movements – which can be performed using keyboard commands or "drag 'n' drop" style mouse manipulation – allow the user to compensate for a lack of data, or possibly erroneous data resulting from the lack of synchronization issue.

It will also be possible to save the current state of a visualization, preserving modifications in timeline synchronization and compression for later use. These saved objects are the only real "output"[4] which the program creates.

The overall real substantial output of the program is the enlightenment of the user, which helps to facilitate the quick resolution of their problem.

[4]in terms of raw, physical bits of data in a file

2.10 How is Argos Used?

2.10.1 Use Case 1

User wishes to visualize overall network traffic.

He/she runs ARGOS using his/her personal traffic data, and ARGOS displays the data along a timeline with different kinds of packets represented as color-coded[5] shapes[6] whose dimensions are determined by packet length or transmission time. The location of each shape on the timeline is relative to real-time as well as the other packets adjacent to it. Thus, there is both an ordinal component of the layout as well as a cardinal one. Packets are placed in the order they are sent or received, and the sizes of their shape representations are representative of overall packet size or duration of transmission. This facilitates easy viewing and comprehension and the user can see how nodes are communicating over the network.

2.10.2 Use Case 2

User's traffic logs from different APs are not synchronized, so it is impossible to see delays or lags by default.

He/she runs ARGOS using the data again. ARGOS presents the same output as earlier. Since the data is no longer synchronized as in Case 1, the packets that are visible may be in the wrong locations on the timeline. This time, the user manipulates the displayed timelines by moving them to the appropriate interval of time sliding them around either the keyboard or mouse so they can become synchronized based on some landmark. User may now view traffic as in Case 1.

[5]specific colors to be defined later
[6]specific shapes for packet types to be defined later as well

2.10.3 Use Case 3

User wishes to visualize broad trends in network traffic.

He/she follows the steps as in Case 1 above, but "zooms out" by compressing a larger time interval (for example an hour) into a single snapshot so that the desired interval can be seen at once. In order to fit more packets on a single timeline, each packet will shrink horizontally, but keep the same height. With a wide enough time frame, the display may begin to resemble a histogram of packet traffic.

From this broader scope, larger trends (e.g. daily, weekly) become more obvious, as groups of packets and gaps are still clearly visible. The only issue, then, is that individual packets become more cumbersome to manipulate. However, it can be safely assumed that if a large amount of data is desired, this invidual manipulation isn't a high priority.

2.10.4 Use Case 4

User wishes to use the minimum amount of power required to successfully transmit between nodes in the network to save power and decrease interference.

He/she follows the same steps as in previous cases, but watches the packets be displayed on-screen in real-time via a real-time parser. This enables the user to modulate power levels on the fly while examining whether or not packet transmissions are successful.

As fewer packets arrive at their destination, the user is able to see what power levels are required to succesfully complete transmissions. From this data, the user can decide whether or not packet loss is within acceptable limits.

2.11 Software Architecture

The architecture for the project is to be designed to be extremely functional while keeping maximum modularity in mind. Given the nature and area of use of the software we are creating, it is not unrealistic to expect some (and possibly crucial) elements of the existing resource base to change. These changes, such as advancements or changes in technology or different equipment, might provide an improved dataset[7], or perhaps one that is radically different. All design decisions that we shall make are made keeping this guiding principle in mind.

The software is to built upon a two-tier structure namely:

- The **parser** or the bottom layer.
- The **GUI** or the top layer.

This structure, though similar to the classic two-tier client/server architecture[8], has a key difference. The two tiers, or modules as they may be called, will communicate with one another through a middle layer, dubbed "the interface". The objective of the interface is to provide a level of abstraction between the PARSER and the GUI, that is not otherwise available. This ensures that the PARSER and the GUI may function independently of one another, while simultaneously ensuring that changes that are made in one will not affect the other. Keeping in mind the interests of hassle-free development, the interface will be defined as broadly as possible and will encompass as much data as possible to ensure that redesign is not required at later stages. Also, having the interface between the PARSER and the GUI leaves room for modifications to the file format and possible changes in parsing methodologies.

[7]The data we are dealing with currently is restricted by the radios that are in use which are nascent in terms of sophistication. If and when more sophisticated radios are used, there would be better data available in terms of signal strength, transmission time and duration and latency.

[8]Two tier architectures consist of components distributed in two layers: client (requester of services) and server (provider of services). The two tier design allocates the user system interface exclusively to the client, and places data management on the server, creating two layers

When the software is run, the PARSER processes a certain set of files, converting the input text stream into a organized set of objects defined by the interface. The interface is designed to lay out a structure that will be convenient for the GUI to utilize, in order to maximize efficiency. The interface defines the information that is obtainable by the GUI and will act as the medium through which the parsed and structured data (in the form of objects) is made available to the GUI. When the GUI is loaded it will take this data and display it to the user in the form of a timeline allowing the user to modify and adjust the views as required.

2.12 Technical Constraints

The following applications/platforms are required for ARGOS to perform:

- TCPDUMP or equivalent - to generate the traffic data to be analyzed
- An **OS** capable of running a TCPDUMP or equivalent, as well as a **Java Virtual Machine**[9].

[9]Luckily, this includes nearly all known OSes

Chapter 3

Design

To invent, you need a good
imagination and a pile of junk.

Thomas A. Edison

3.1 Design Motivations

3.1.1 Structure

As discussed earlier, modularity was the primary motivation while designing Argos to ensure easy reconfigurability and reusability. Thus each Argos design decision was made keeping in mind the premise of maintaining and increasing modularity. The reason for this emphasis on reconfigurability stems from the key area that the software package addresses, namely network visualization over the 802.11 layer achieved by examining packet logs generated by TCPDump. These fundamental considerations around which Argos is built, allow us to compensate for the inherent fluidity with regard to both the underlying components that our software relies on and the output it generates. From the outset we are aware that a number of critical parameters, such as the format used by TCPDump to log data, the radios used for communication, the variety of information to be visualized and other similar requirements, might change in the near future. This would most certainly result in a change in the quantity and nature of the data available. In such a scenario, lack of flexibility in the structure would necessitate a major code rewrite and would effectively render the software crippled.

To ensure that the software continues to function effectively, and can be used for a reasonable length of time, we designed Argos into a set of disjoint modules which when combined, represent a classic client/server two- tier architecture.

- The Parser

  The parser is responsible for converting a log file into Java objects that are then available for the GUI to display. In our case the log file is a TCPDump file, however, with the modular design, the log file could just as well be in any other format, XML is one such example, and it would be able to mesh perfectly with Argos. The parsing component represents the lowest level of the architecture. It is comprised of the following class(es) and interface[(s):

1. `AbstractParserFactory`
2. `ArgosParser`

- The Middleware or API

  The middleware for our implementation consists of an API or a standard utilized by the GUI to extract data from a set of log files that the user specifies. Given this standard, it is easily possible for a user to customize or perhaps even replace various components of the framework, including the parser, and still use the GUI for visualization purposes. This middleware is most closely tied to the parser, yet it maintains abstractness as far as possible. As long as a new component conforms to the API expected by the GUI, it can be seamlessly integrated into the software. In this manner the API works to provide the functionality for the GUI to work efficiently. The API level of the architecture is comprised of the following class(es) and interface(s):

  1. `Log`
  2. `ArgosLog`
  3. `Node`
  4. `ArgosNode`

- The GUI or Interface

  The purpose of the GUI is to provide the user with the functionality and display options that he/ she expects from Argos. The GUI will effectively be isolated from the data and will merely provide the user with means of manipulating it. Here again the modular design plays a significant role in shaping the ability to augment the current components or to completely replace them with other more sophisticated visualization tools. To accomplish this we implemented an Observer/Observed hierarchy, where a visualization component that is observing a Node receives information when the Node's state changes. The GUI level of the architecture is comprised of the following class(es) and interface(s):

1. `ArgosGUI`
2. `VisualizationFrame`
3. `TimelinePanel`
4. `NodeObserver`
5. `Timeline`

- Data Storage Design

  Argos has two primary data storage structures, the `Addr` object which represents a MAC Address and the `Packet` object which represents the information contained within a network packet. The `Packet` object takes its constructor several data members which are then immutable for the lifetime of the packet. Since the composition of this information is not set in stone, there exists a method that allows a user to add more information to the Packet other than what it presently contains.

3.1.2 Language Choice

Since a large part of the parser component of the project consisted of manipulating bits within larger length strings, Java was a natural choice of language to implement the project in.

This decision also provides the added functionality of allowing the project to be portable. Despite the fact that our present client[1] has a homogenous setup running NetBSD, it allows Argos to be run in a variety of environments thereby enhancing its usability outside of the narrow scope that it is presently intended for.

[1] CU Wireless

3.2 Parser Design

The parser class is designed to take a log file (perhaps generated by a utility such as TCPDump) and convert the information contained within it into Packet objects. These object can then be manipulated by the GUI components to render them effectively on the scree. The current implementation of our project does not contain a fully functional TCPDump parser but instead relies on simulated data that the `ArgosParser` class feeds to the GUI components when prompted.

In the typical deployment scenario, the parser shall take a log file that is output by TCP-Dump, and then convert this into the Java objects required for the GUI.

The format of the information that the parser is looking for is described in the 802.11b specification[2]. Specifically, the parser seeks to identify the three main types of frames:

- Control Frames

  These frames such as RTS (Request To Send), CTS (Clear To Send) and ACK (Acknowledge) frames manage the distribution of the physical layer and distribute usage amongst the various nodes that comprise the network.

- Management Frames

  These frames such as Association and Disassociation requests, Probes and Beacons help control allow access to the layer and help ensure that the integrity of the network is not reached.

- Data Frames

  These frames such as Data + CF (Contention Free), plain Data and other are responsible for the actual transmission of data from one node to the next.

3.2.1 Parser Components

This design objective is accomplished by the following class structure.

- The `AbstractParserFactory` Interface

[2]See Appendix B on page 61 for more detailed information

The parser design consists of an abstract interface named `AbstractParserFactory` that defines methods that any parser to be used in ARGOS must implement.[3] These methods allow the client code to request packets of information that are relevant to the particular node and time-interval being visualized.

- The `ArgosParser` Class

  In our implementation, the `ArgosParser` class performs the functions that are required of the parser. It creates simulated information in the form of `Packet` objects which it then provides to its clients as requested.

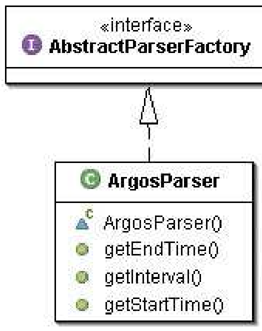


Figure 3.1: ARGOS Parser Hierarchy.

The parser class is instantiated when the user chooses to visualize a log. Each log format shall have a parser component associated with it, and when the user chooses a log to parse, the appropriate parsing engine shall be invoked.

3.2.2 Parsing

1. Parsing begins when the user chooses a log file to visualize. At this step the parser is invoked and it begins by reading through the file collecting information about the nodes that are represented in the log.

[3]For a more thorough discussion of the interface and required methods, consult Appendix C on page 63, which lays out the UML diagram and internal structure of ARGOS components.

2. This information is used by the GUI to allow the user to make selections regarding the nodes he/she wishes to visualize.

3. The actual parsing process is done by reading a line of text from the log file, which is then converted into its corresponding `byte` stream.

4. Time information is extracted from the TCPDump header portion of the stream, after which the TCPDump section is discarded.

5. The remainder of the stream represents a portion of the packet that has the 802.11 header around it. The first two bits (0-1) of the header are examined to determine the packet type.

6. The remainder of the byte stream is examined to extract other information such as the packet subtype, the duration of the transmission, the source and destination addresses and the BSSID.

7. The packet subtype helps in the identification of the remainder of the data which is then extracted from the byte stream, and the process continues with the next line of the log.

3.3 API Design

The viewpoint behind a separate API for the project that lies between the GUI and the parser is to disassociate the user-interface from the parsing process. Our aim is to make the GUI independent of the data structures and formats associated with the process of parsing the log file. This will allow the user to replace the current parsing component with another in the future, if the structure or the source, of logged data changes.

Thus, if for example, the format in which data is logged by TCPDump changes, or if the data available from the radio transmitter increases or changes in anyway, then a new parsing component can be written to replace the existing parser. The user could also write a packet sniffer in another language, say Perl, then by writing a “translation” module the user can continue to use our software. The role of the “translation” module will be to implement the required interface methods that will allow ARGOS to utilize the data generated by the new packet sniffer. In this manner, it is no longer necessary for the GUI to have intimate knowledge of the structure of the parsed data. When a user wishes to visualize a log, the GUI makes the requisite API calls for the corresponding sections of data, the implementing class performs any translations that are required and hands the information to the GUI, which then displays it according to the user’s preferences.

This gives us maximum modularity while allowing the GUI component to be reused in an unrelated project that has a requirement to visualize data similar in nature.

3.3.1 API Components

To accomplish these goals, the design of the classes that comprise the middleware is as follows:

- The `Log` classes and interface:

  The abstract `Log` class encapsulates the concept of a log file. A `Log` is instantiated with a `AbstractParserFactory` object that performs the parsing when necessary. The `Log` has information about the `Node` objects that can be found in the log, as well as information about the time interval that is represented by the log file. The `Log` class provides the interface used by the GUI extract data. It provides methods that return packets based on time and node information.
  In our current implementation `ArgosLog` provides much of the functionality that the Log class requires.

- The `Node` classes in interface:

  The abstract `Node` class encapsulates the concept of a node in the network. A `Node` object is closely related to the actual screen display. A `Node` may contain multiple logs since it is theoretically possible for more than one log to contain information about a specific packet. The `Node` object also contains information about the entire range of time intervals that are contained in all its logs. The GUI directly manipulates a `Node` object, which in turn performs operations on the `Log` objects that it contains to provide `Packet` information to display.

In our current implementation `ArgosNode` provides the functionality demanded by the abstract `Node` class.

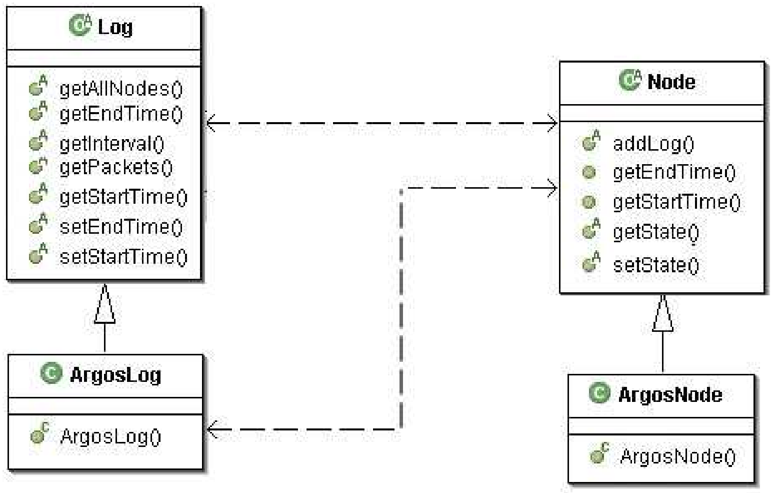


Figure 3.2: ARGOS API Hierarchy.

In a typical session, given a combination of ⟨log file, `Log` object, `Node` object⟩, the GUI shall be able to extract all the relevant information without any hinderance.

3.4 GUI Design

In comparison to the other components the design for the GUI is far more involved and required considerable effort. Since the functions of the GUI are more detailed and more complex, the structure of the class(es) that provide these functions needs to be more flexible and robust. The underlying objective has been to expose a working API for the to make it possible to develop components that can plug into ARGOS and provide additional visualization capabilities. This has allowed us to concentrate on the extensibility of the software, rather than focusing on implementing a limited set of features that could not be added to at a later time.

Primarily, the structure consists of the following classes:

- `ArgosGUI` Class

  This class is the driver of the entire application. It provides the basic interface that allows the user to create and manipulate visualizations. The `ArgosGUI` class instantiates a `VisualizationFrame` class when prompted to create a visualization.

- A `VisualizationFrame` Class

  The visualization frame class encapsulates a visualization to which the user can add or remover nodes and log files. The visualization frame provides the ability to manipulate a single timeline view of a node or multiple timeline views at the same time. The visualization frame takes as a parameter in its constructor a `Log` object that it uses to draw node information from.

- A `TimelinePanel` Class

  The `TimelinePanel` Class allows the user manipulate and view multiple timelines at a single time. It acts as a container class that takes special actions from the user and applies them to all the `Timeline` objects that are contained within it.

- A `Timeline` Class

  This class defines a particular timeline. It contain instances of packets in the form of `DisplayObject` instances and has functions and methods that allow the user to manipulate packets that are being visualized. These methods include methods that navigate the timeline, perform zoom-in and zoom-out functions and identify individual packet information.

The Timeline class implements the `NodeObserver` interface which allows it access to information regarding the change of state of a particular node. In essence it is possible to design to different visualization components that observe the same `Node` object, and have them aware of changes made to the state of the `Node` by the other.

- A `DisplayObject` Class

  This class manages the display of a single packet transmission. It is different from the Packet class in the lower level architecture in that its primary purpose is to manipulate packets on a timeline rather than encapsulate data. It provides methods that render the packet on the timeline based on the packet type and subtype, and methods that align the packet to a particular point in the timeline as well.

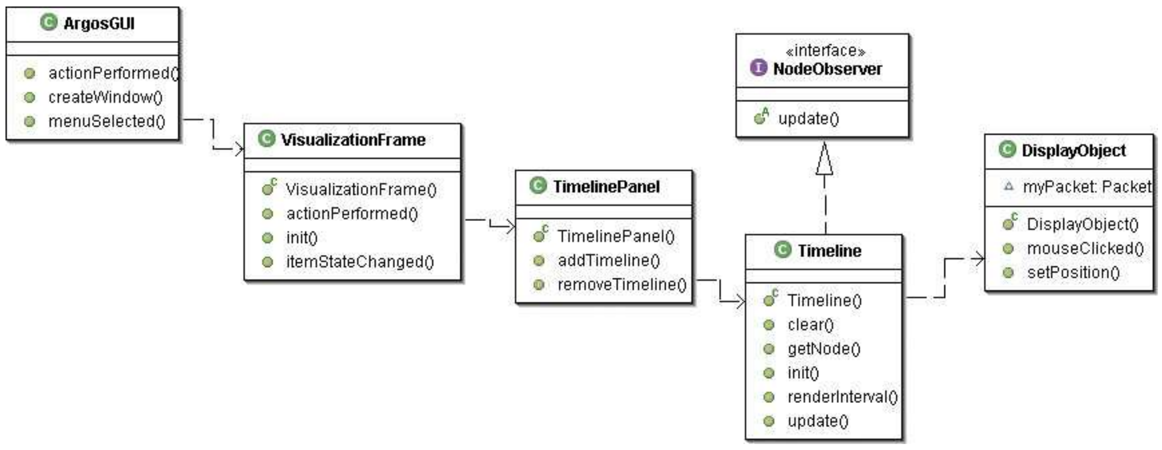


Figure 3.3: ARGOS GUI Hierarchy.

Chapter 4

GUI

mmmmm, goooey

Homer J. Simpson

4.1 GUI Design

Since JAVA is the language of choice for ARGOS we needed a GUI platform that would work with JAVA without any major hassles or integration issues. With that in mind **Swing**[4] was chosen. SWING is already integrated with, and is a core part of, the JAVA 2 platform. With components such as Combo Boxes, Sliders, and Drag & Drop support, SWING is ideal for accomplishing what the GUI needs to do.

Because the GUI is what sets ARGOS apart from all other network debugging applications, that is what the bulk of the design of this project has been aimed toward. We have gone through several iterations of the GUI already, refining the design as we progressed to optimize both usability and functionality. That series of GUI iterations is displayed in the following section.

- Stage 1
  - Our initial mockup was very text-based and a very rough representation of what ARGOS was planned to display.
  - It was essentially non-interactive and more of a visual representation of data, rather than a display capable of being manipulated.
  - It also did not utilize any of the capabilities afforded to us by the use of Java and Swing.
- Stage 2
  - This mockup was created in Visual Basic as the basis for the finished GUI.
  - Nearly everything contained in this mockup will hopefully be part of the final product.
  - This iteration contained a much more standard interface, as well as more added functionality that allowed for the manipulation of packets, as well as some of the more advanced features such as auto-analysis of traffic problems.

- Stage 3
  - This iteration was created in Java using Swing, and is the current state of the GUI.
  - It may seem at first glance to be feature-light, but the majority of the work done to create this GUI allows for modularity and future functionality.
  - One major addition to the GUI design over the mockup in Stage 2 is the ability to monitor a varying number of nodes not limited to only two.
- Future
  - This iteration is a logical extension of the current state of the GUI, and would be what ARGOS would look like if given another month of work.
  - It contains a realistic number of features that can be found in the Visual Basic GUI, but transplanted into our current Swing GUI.

Since ARGOS is a visualizer, the GUI is the centerpiece of this project. It allows the user to choose any number of TCP dumps to be displayed and create a visual timeline model of a default range of network traffic. Since there are times where the sequence of multiple dumps will not align properly, the GUI allows the user to adjust, or slide, one visualization over to accommodate. To view different sections of the timelines ARGOS can be adjusted via buttons to slide the range in the GUI temporally. This can also adjust the "zoom" of the visualization which will also be an option in ARGOS.

4.2 GUI Function

Beyond simply moving the dump timelines around, our GUI will have some minimal functionality. Although much can be done in the visualization process without the GUI actually handling any calculations, there are some things that ARGOS will do that the GUI will be required to do. These will include minimal processing of user requests, basic interface error handling, as well as saving and reloading the state of the project.

Each time a user requests a different set of data, an additional dump visualization, or a different range of the current timeline, the request must be handled. By implementing an extra layer of abstraction in our middleware design we relieve the GUI from much of the processing. It will only be required to make requests that the middleware will carry out. The minimal amount of functionality still required of the GUI, such as formatting the requests to the standard of the middleware, will be negligible.

As far as processing goes, one thing that cannot be passed off is interface error handling. That is, making sure the user requests are valid, within limits, and warning the user if extensive amounts of resources may be required to complete each request. Out of range values will need to be met with error messages. Redundant requests will need to be handled to avoid conflicts. And, of course, we wouldn't want the user exiting Argos without saving the user's changes.

Saving changes will be one of the lower priorities as far as implementing the GUI, however it will be an important one. If a user does a few hours of work to set up the timeline visualizations properly, Argos will be able to save enough information to recreate the project at the point the user stopped. This implementation will also require us to write the loading side of the saved states. By loading this saved state Argos will bring up the visualization just as the user recalls it so they can restart where they left off.

4.3 GUI Future

The fate of the GUI is now in the hands of future developers. Due to the highly-modular design of the GUI, it should prove relatively straightforward for any interested developer to add on to and modify the existing GUI framework.

Some of the extra "features" we envision being added include Drag & Drop and Gravity support as well as a possible Auto-Analyze feature. Dragging & Dropping will be useful to the user for simple addition or removal of dumps or switching of visualizations. Gravity is the concept where as the user drags items near an object or grid line the item will "snap" to the object or line and not leave small arbitrary gaps between the items or objects. This not only helps the user, but also reduces errors by eliminating the small gaps between where a user intended to place and object and where it is placed in the GUI.

Among the more ambitious of our final goals is an Auto-Analyze feature that will look at the timelines and find obvious errors that can be identified by the order of TCP packets. For instance if a data packet is sent over and over this indicates a traffic error which may be caused by previously-discussed problems. This Auto-Analyze feature will be difficult to implement and may end up being an applet separate from Argos's GUI. The Stage 2 screen shot following this section has another example of what the Auto-Analyzer may be able to detect. The red circle brings attention to crosstalk from a node not seen by the node that is currently trying to transmit data. This could be the cause of major network traffic slowdown but may be difficult to identify without the use of tools such as Argos.

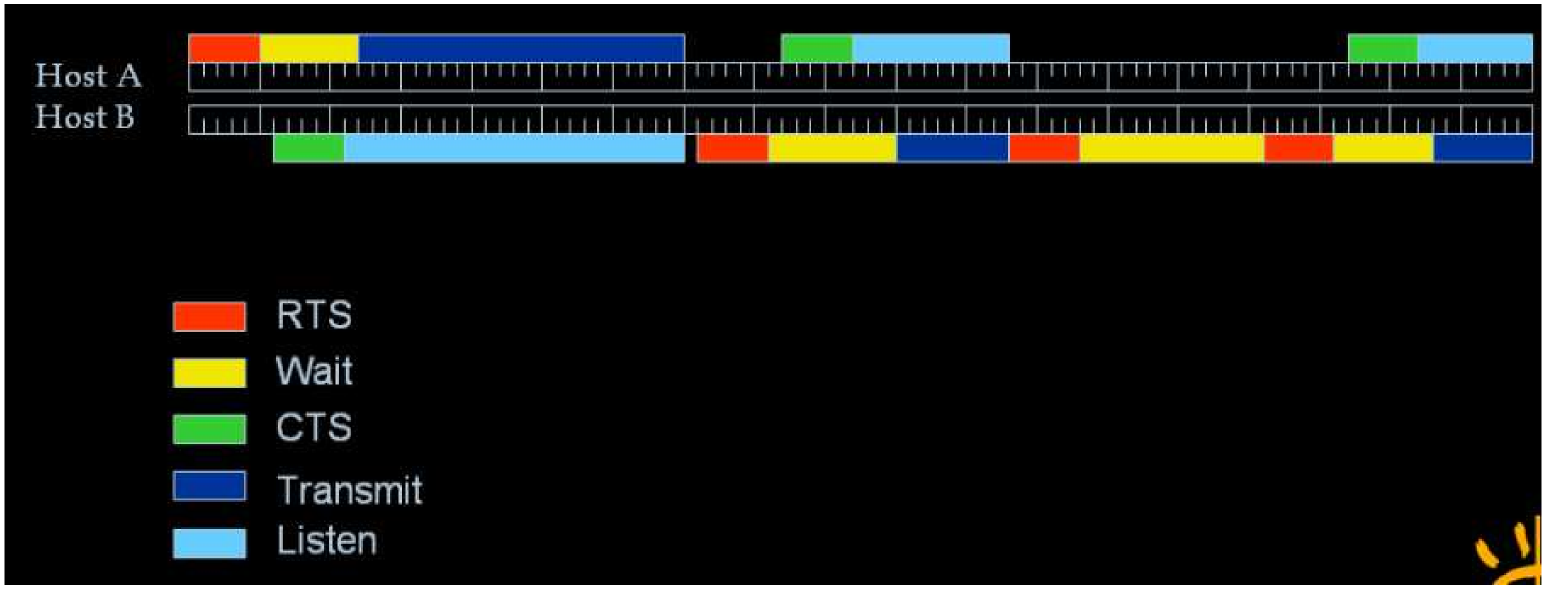
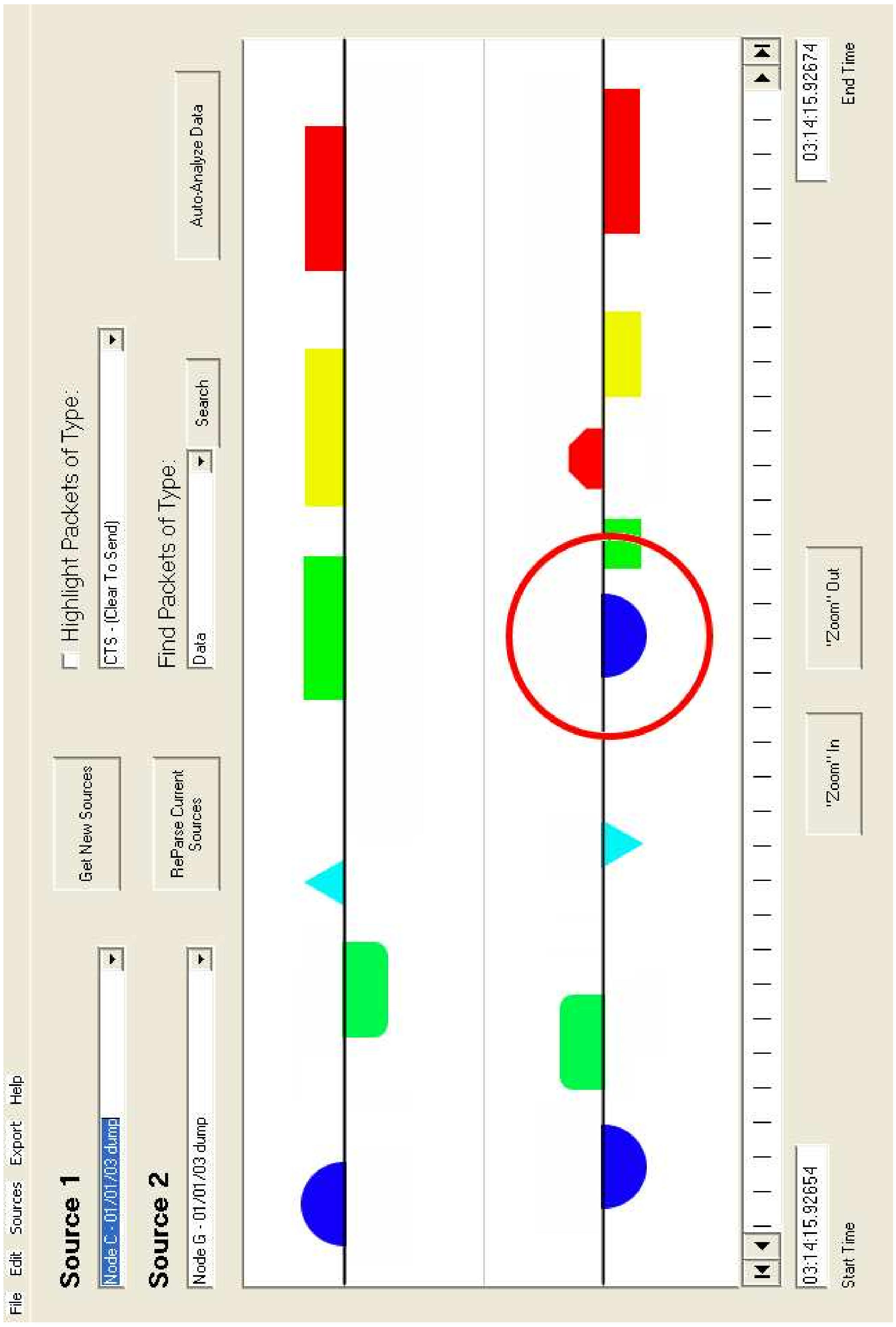


Figure 4.1: Argos Stage 1 Screenshot



Source 1: Node C - 01/01/03 dump

Source 2: Node G - 01/01/03 dump

Highlight Packets of Type: CTS - (Clear To Send)

Find Packets of Type: Data

Start Time: 03:14:15.92654

End Time: 03:14:15.92674

Figure 4.2: ARGOS Stage 2 Screenshot

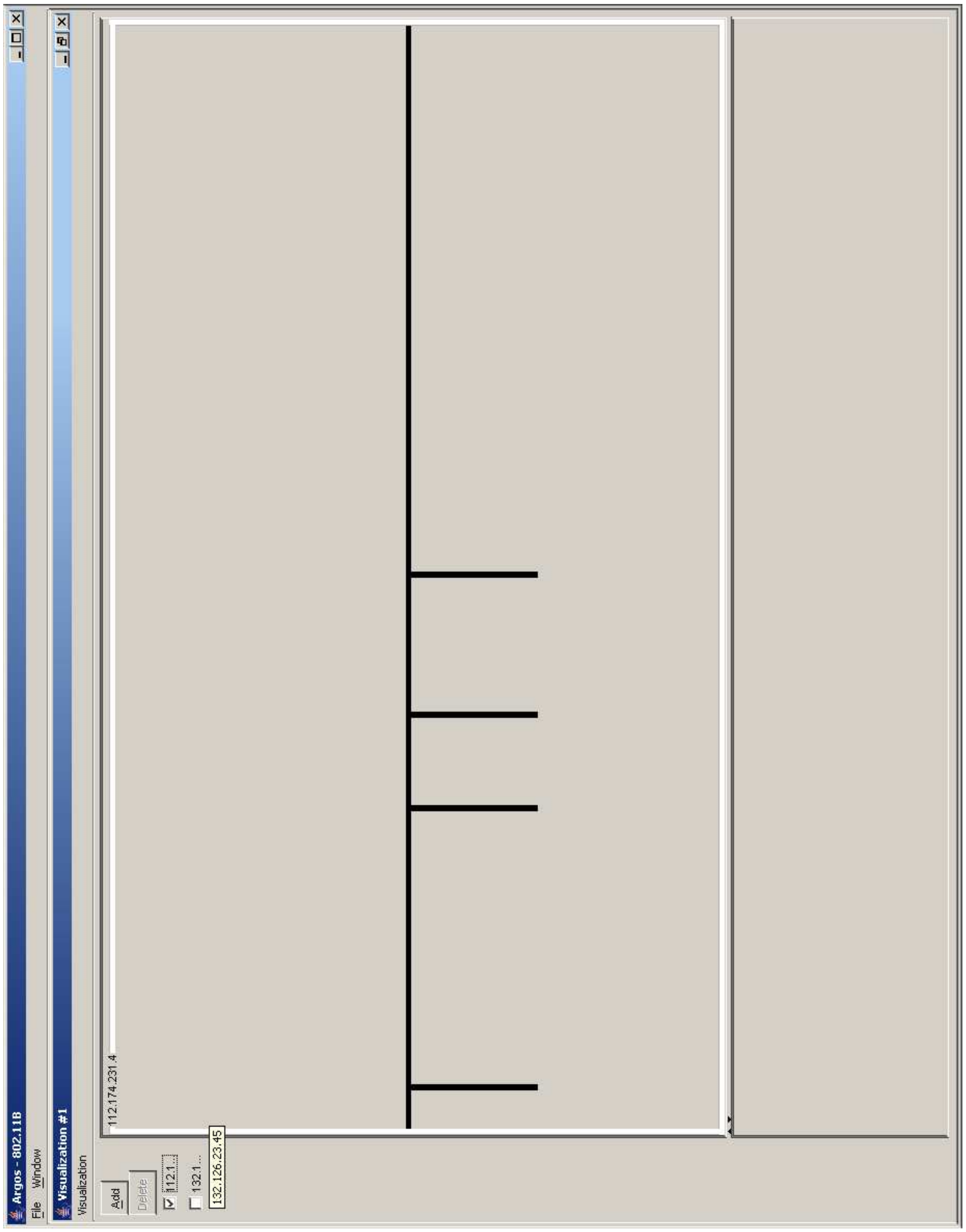
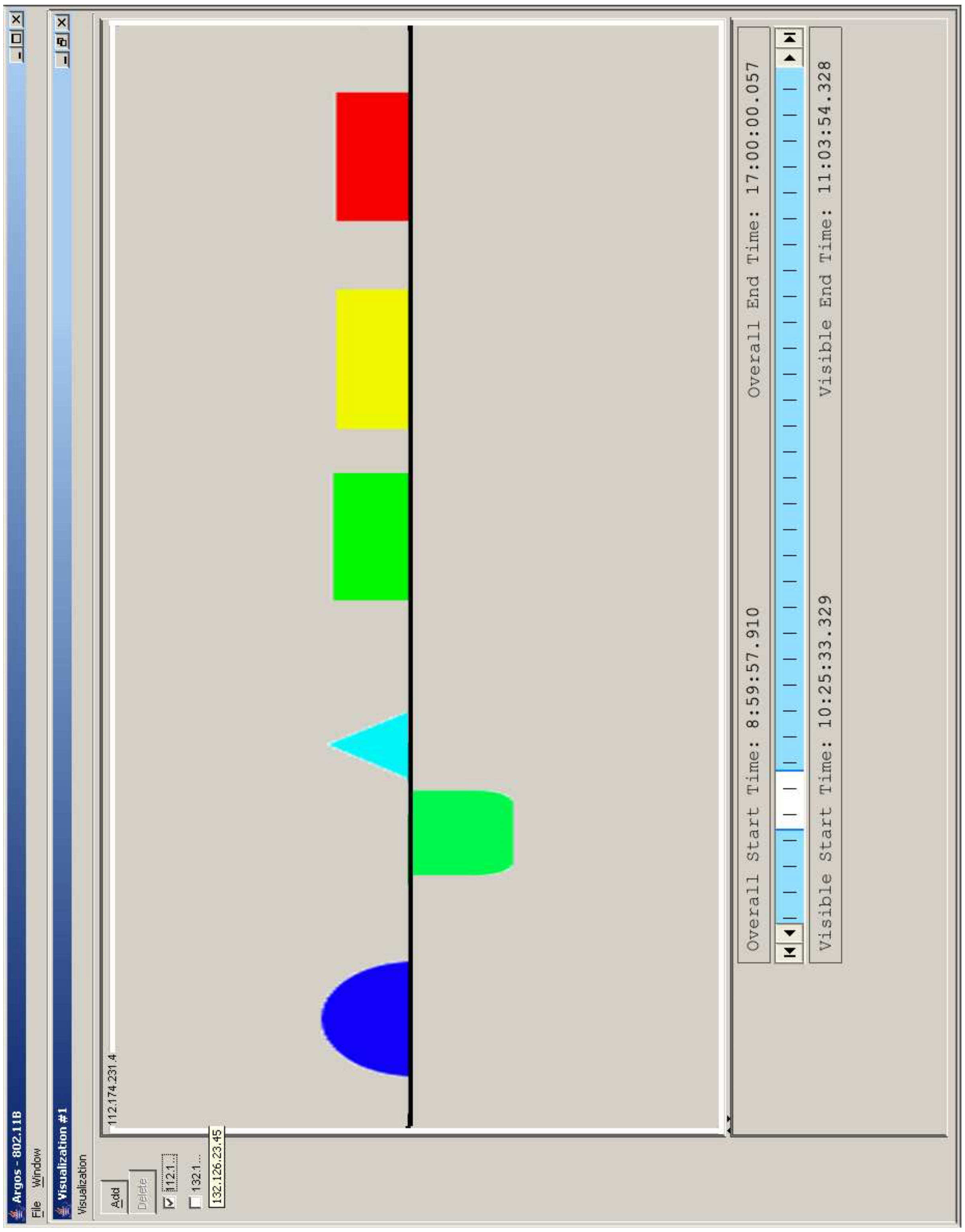


Figure 4.3: ARGOS Stage 3 Screenshot



```
Overall Start Time: 8:59:57.910          Overall End Time: 17:00:00.057
Visible Start Time: 10:25:33.329         Visible End Time: 11:03:54.328
```

Figure 4.4: ARGOS Future Screenshot

Chapter 5

Implementation Issues

The devil is in the details

Anon.

5.1 Introduction

As with any project or undertaking of this size, some issues regarding implementation arose. In the specific case of Argos, two major issues came about, as well as a host of minor related problems. Here, we will show not only the problems encountered, but also their solutions[1].

We shall first address those issues created due to the nature of the project itself, and then those which occurred due to our particular implementation approach.

5.2 Inherent Issues

Logically, the first issue that was encountered was that of packet acquisition. It was decided that the acquisition of packets from different access points on the network would be performed by running TCPDump on each node, which allows us to track all network traffic through and around that node. TCPDump was chosen for three major reasons:

- Timestamping

  allows for accurate temporal ordering of packets

[1]as well as projected solutions for those yet to be resolved

- Updatable Formatting

  allows for ease in tailoring the logs in a format that can be manipulated

- Widespread Portability

  ensures[2] portability to many different platforms

5.2.1 TCPDump Issues

Timestamping

The timestamp within the packet will give only the time the packet was sent from that particular host. Since this system must work within the context of a wireless packet routing scheme, with APs transferring packets to other APs. This creates so many issues that it was infeasible to compute the actual arrival time at a particular host.

TCPDump's timestamp at acquisition creates issues of its own. The timestamp is applied after the packet has left the network's buffer space. This is only accurate to the millisecond of receiving time[5].

We would desire the time of transmit as well as the time the packet exists the network's buffer space. Without building the packets ourselves and creating our own wrapper class between the 802.11 encapsulation and the TCP/IP encapsulation. This was deemed infeasible within the scope of this project. It has been documented as a viable solution with the continuation of the project and functionality has been included within the current structure for such a solution.

Updatable Formatting

Since we are using a third party software system to perform our dumping for us, we must assure that a change in that software's encoding of the packet information will not break our system. It is for exactly this reason that we decided upon TCPDump. TCPDump uses the

[2]along with the fact that our project is implemented in Java

C library '`libpcap`'[6] which has been ported to JAVA in JPCAP[7], giving all functionality which is associated within C and C++ to a JAVA based system. These functionalities include: acquiring information from a dump created with TCPDUMP, or any other dumping mechanism which uses `libpcap` for packet captures.

Widespread System Portability

TCPDUMP has been ported to include a wide variety of systems including: Sun SPARC, PC x86 (both Windows & Linux), HP 9000/C110, SGI/CRAY 02000, Solaris 2.6 SPARC, Solaris 2.5.1 x86, HP-UX 10.20, and IRIX 6.5.7m. The only systems not ported to are Apple machines at the time of this document.

5.3 Design Issues

The second type of issue that was encountered was one that came about due to choices that we made in our implementation, rather than a problem inherent in the goal of the project. The major one of these was the interface design.

5.3.1 Interface Design Issues

After intense research into the 802.11 specification, it became clear that the way the packets and headers were structured lent itself very well to a system revolving around objects. Due to the several slight variations in the classes of 802.11 packets mentioned earlier, it was decided that a hierarchical object-oriented system was the best and most appropriate solution. This solution created two additional minor issues, however: pre-parsing vs. run-time parsing, and language choice itself.

Pre-Parsing vs. Run-time Parsing

There are only two distinct ways of attacking the issue of parsing the data into something easily displayed. One is to pre-parse, and the other is to parse at run-time.

- Pre-Parsing

  Pre-parsing consists of reading the input log at the beginning of the process and storing the data in object form on the disk, since many logs will be far too large to be held in memory. This would only happen once, because the objects created have only the necessary and relevant data in them. After pre-parsing, the GUI would pull the already-parsed data off of the disk and perform its functions with no extraneous information to sift through.

  This possible solution has its strength in its simplicity of implementation. Its weakness, however, lies in the fact that this serialization of log data to JAVA objects is relatively slow.

- Run-time Parsing

  Run-time parsing consists of the same basic steps as pre-parsing, but the actual parsing is postponed until the GUI explicitly needs it. In other words, there would be a minimal amount of parsing at first, to store the log data in binary form on the disk, which would later be parsed and objectified as necessary. Since the objects aren't being serialized to disk, we avoid the speed issue involved with pre-parsing, and the implementation itself is arguably no more complicated than pre-parsing in its design. However, we have found that its implementation is not as straight-forward as it once seemed.

It would seem obvious that run-time parsing is the way to go, which is why that is the avenue we chose to pursue for ARGOS . We chose run-time parsing partly because of the more open-ended approach available to us in terms of parser design. One advantage is that with a run-time parser, the ability to create a real-time parser is possible. This goes a long way toward our goal of modularity, although it requires much more work in the long-run.

Choice of Language

As previously noted, Java was taken to be our language of choice for this project. This decision was not made lightly, however, as many different facts and trade-offs, and advantages and disadvantages had to be understood and balanced before the appropriate language could be chosen. Once again, these issues are described in detail. This time, however, from the implementation point of view, rather than simply the design standpoint.
Advantages:

- Platform independence

  As previously stated, Java is completely platform agnostic. This is a definite plus, as it allows the audience of our application to be much larger than it would be otherwise

- Experience

  Each team member working on the project has had fairly extensive experience working with and writing Java code. This allowed us all to become involved in the design process as well as the creation process, and no one was left behind.

- OO-ness

  Because Java is structured entirely around the concept of objects, it fit the design model perfectly, as it, too, was designed around an object-oriented framework.

Disadvantages:

- Speed

  In almost all circumstances, Java tends to be anywhere from slightly to extremely slower than other languages like C and C++

- Abstraction

  Other languages allow for complete access to the hardware when necessary, but Java shrouds that from the user in a thick layer of abstraction

5.4 Implementation Issues

Despite our very elaborate and in-depth design, we still ran into a series of roadblocks when it came time to implement both the GUI and the parser.

5.4.1 Parser vs GUI - Clash of the Titans

When we began the project, we believed the parser to be the most difficult and important issue to tackle in our implementation of the design. The GUI, on the other hand, we felt would be relatively straightfoward in its design, but slow and time-consuming in its implementation due to the many idiosyncrasies of GUI programming.

As we found out much later, however, our belief that the parser was both the most difficult and most important part of the project was unfounded. The difficulty we were experiencing in the implementation of the parser was due more to the fact that none of us had written a parser than any inherent difficulty in the design. Furthermore, a parser infrastructure is already in place in many other forms, allowing the user to use ARGOS for more than what our initial design was based around. Writing our own proprietary parser would also limit the portability of the GUI, and thus diminish its widespread use throughout the networking world.

5.4.2 Argos Uncertainty Principle

Another implementation problem that we ran into was a conceptual one. That is, in currently available wireless hardware, it is infeasible for an active participant in transmission to log all traffic it can detect. This requires there to be a monitoring node that must make several assumptions about the reception of packets by network nodes. For example, although a monitor node may be perfectly capable of intercepting packets in transit that are destined to reach a particular node, there is no guarantee that the destination node is actually receiving them. In other words, because of our monitor's incomplete view of the network traffic, we must make assumptions that alter the model we use to display said traffic.

5.4.3 GUIs are Hard. Let's Go Shopping.

Although we knew coming into this stage of development that GUI programming could be tedious and time-consuming, none of us had attempted to create an interface of this magnitude, so we grossly underestimated the investment of time required. This assumption led us to a much lower than optimal level of preparation. Such an inadequate grasp of the language specifics required us to spend valuable time researching Swing constructs in parallel with the actual writing of GUI code.

One of the early stumbling blocks we encountered was the manner in which Swing objects interact with each other. We found ourselves implementing large chunks of code based on our intuition of the system's inner workings, and later realized that, in actuality, Swing was designed in a completely different fashion. Due to this conflict, we found ourselves second-guessing our methods of coding, prompting us to consult the Swing specification at every step, thus slowing the coding process.

Another stumbling block that we encountered later on in the development of the GUI was that of **painting**. The same method that Swing uses to manipulate video buffers to eliminate flickering also complicates the process by which a programmer is able to draw objects to the screen. This problem manifested itself whenever we attempted to draw a packet or timeline to the display, or visually manipulate a packet or timeline that has already been drawn to the display. Instead of simply redrawing the screen whenever we affected the display area, we found that the only way to get our desired result was to minimize the window and force a refresh.

All in all, this underestimation of time requirements forced us to re-evaluate the true difficulty of GUI programming. We now know to devote more time to the GUI stage of future project implementation.

5.4.4 Data Structure Woes

The final major problem we encountered in the implementation of the GUI was that of data structures. Since we knew that we had to have a data structure capable of containing mappings between nodes and their names, the logical choice was to use a built-in hash map. We soon found out, however, that that was ill-advised at best.

Since a hash map is defined by its ability to hash objects to their proper positions in the map, it requires knowledge of equality between objects in order to ensure that position. Java's built-in hash map seemed like it was capable of performing this task adequately at first. However, we soon found out that since we were using custom objects, that notion of equality was unknown to the virtual machine. We spent a large amount of time trying to alter the way that Java could interpret equality between our nodes, only to find out that it was impossible.

After this realization, we simply created an extension to the standard Java hash map, which we then modified to utilize our own notion of equality. This cumbersome, but eventually successful, endeavor became part of the final product. Similarly, we created modified hash sets with the same reasoning.

Chapter 6

Future Plans

Roads? Where we're going we don't need roads.

Dr. Emmett Brown

6.1 Plans for the Future

As you've already no doubt seen, there has already been a substantial amount of design work done thus far. However, with a series of several setbacks in development, progress has slowed in most cases, and regressed in a few. We firmly believe that the base of the project is solidly defined and provides for a good point at which to allow for further development on the GUI portion alone.

Below is a basic overall glimpse at what has been accomplished (the text with the strike-through), and what has yet to be accomplished.

- Parser
  - ~~Design Abstract Parser classes~~
  - Write Parser classes
  - Serialize objects[1]
  - Thoroughly test parser[2]
- Protocol/Interface/Middleware
  - ~~Define protocol specifications~~
  - ~~Design overall structure~~
  - ~~Implement object classes~~
- GUI
  - ~~Create mockup for design~~
  - ~~Identify overall hierarchy~~
  - ~~Write classes and primitive GUI structure~~
  - Enhance and test GUI[3]
    - Auto-Analyze - The feature in a previous iteration of the GUI design that allows ARGOS to do a bit of the grunt work of traffic analysis on its own
    - Displaying Packet Information - The ability to click on a specific packet and see packet-specific information, including possibly the data in its payload
    - Timeline Shifting - A major goal, which is to be able to shift packets around the timeline via drag-n-drop to help visually align packets
    - Overview Timeline - A larger timeline display that encompasses all traffic on all nodes; allows the user to zoom, shift and scale the display
    - State Saving/Loading - The ability to save the current state of the display in order to be able to recall it later without re-doing all of the work required to set it up

[1]Again, this is a possibly defunct goal, depending on serialization tactic to be decided later

[2]With the change in focus of the project, the parser has taken the wayside, but it is still a necessary part of the application and, as such, requires thorough testing

[3]Again, this is potentially an eternally ongoing plan, as – much like a work of art – no GUI is truly finished

Chapter 7

Closing Remarks

M-I-C... (See you real soon)...
K-E-Y... (Why? because we like
you)... M-O-U-S-E.

Mickey Mouse Club

7.1 Conclusion

In conclusion, we feel that Argos has the potential to redefine the way that administrators of wireless networks can do their jobs. With Argos , one will be able to quickly and succinctly diagnose and treat any networking problems, no matter how large or small, using its intuitive interface. As of this writing it may be a pale representation of that potential, but there is little doubt that it's there lying under the surface.

It is a monumentous task, but the hardest part (designing the whole application, including the intricate class structures and hierarchies required to represent all necessary data) is done, and the task of finishing the implementation is the only thing remaining. We have made great strides toward the goal of the project, and have a very concrete concept that simply needs to be realized. This is not to say that it will be a breeze from here on out. GUI implementation is one of the hardest things to get done, let alone to get *right*. Even though we have a proper GUI skeleton upon which to base all further development, the task of rendering all of the desired data to the screen is a non-trivial one.

We still look forward to the final product since we all have an interest in wireless as well, and we realize that the need for a tool like Argos is increasing as each day passes. This project is as much for us and everyone else that is interested in 802.11 as it is for our client alone, which makes it all the more important that we succeed in our goals of making an intuitive and efficient application.

Although our project has not reached its final stage of completion and we have officially ceased development as the semester has come to a close, we plan to release the source code we have in the form of a sourceforge project in the coming weeks. This will allow us, and anyone else who is interested and motivated in the project to work after graduation, and for the collaboration of everyone in the Open Source community for aid in both testing and development. That collaboration will help find and remedy more bugs and get more varied input than our group and client can come up with alone.

Appendix A

User Manual

You can fool some of the people all the time, all the people some of the time, but you can't fool all the people all the time

Abraham Lincoln

A.1 Getting Started

A.1.1 Launching Argos

Once the setup process is complete and Argos is available on your system, executing the application brings you to this stage:

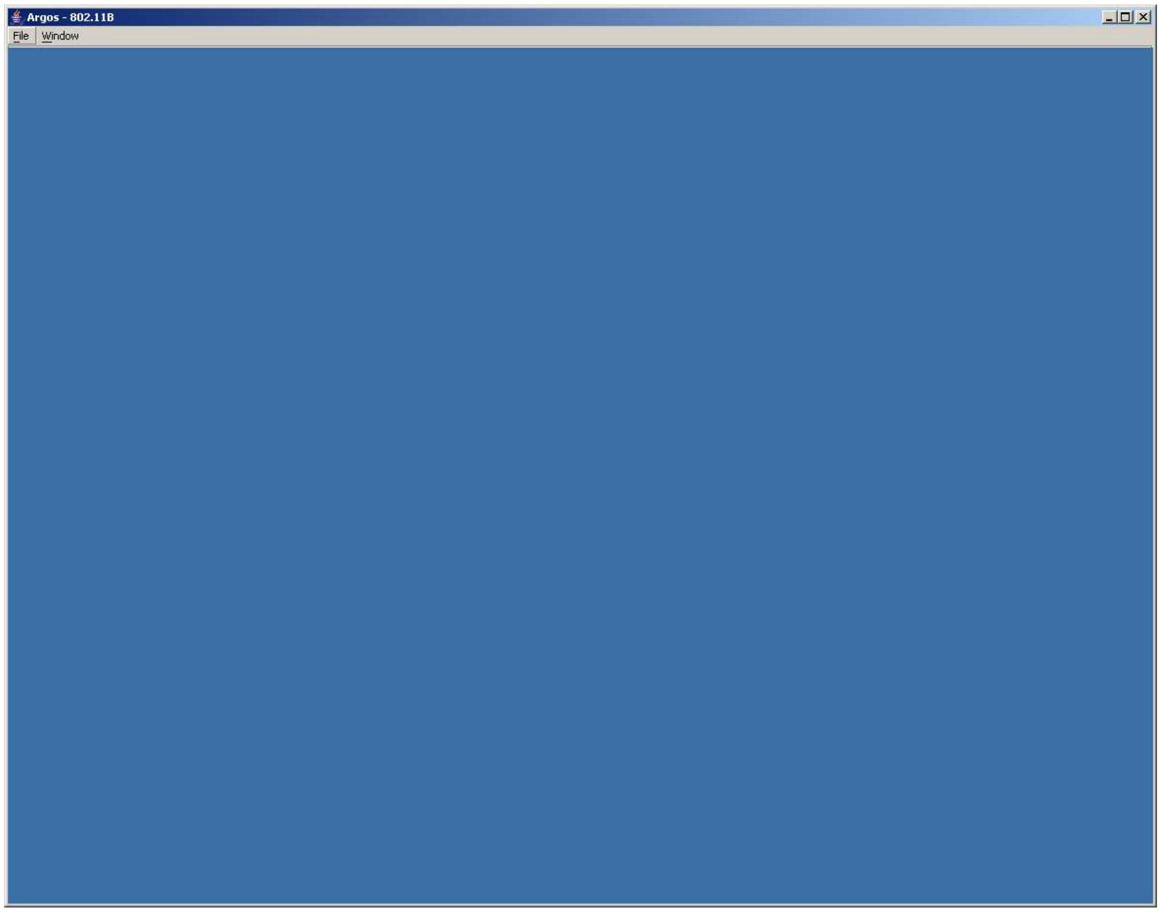


Figure A.1: Argos Launching Screenshot

This is the starting point for all of Argos's functions.

A.1.2 Creating a visualization

Once Argos is up and running, you can create a new visualization by going to the `Visualization` menu and choosing `New Visualization`.

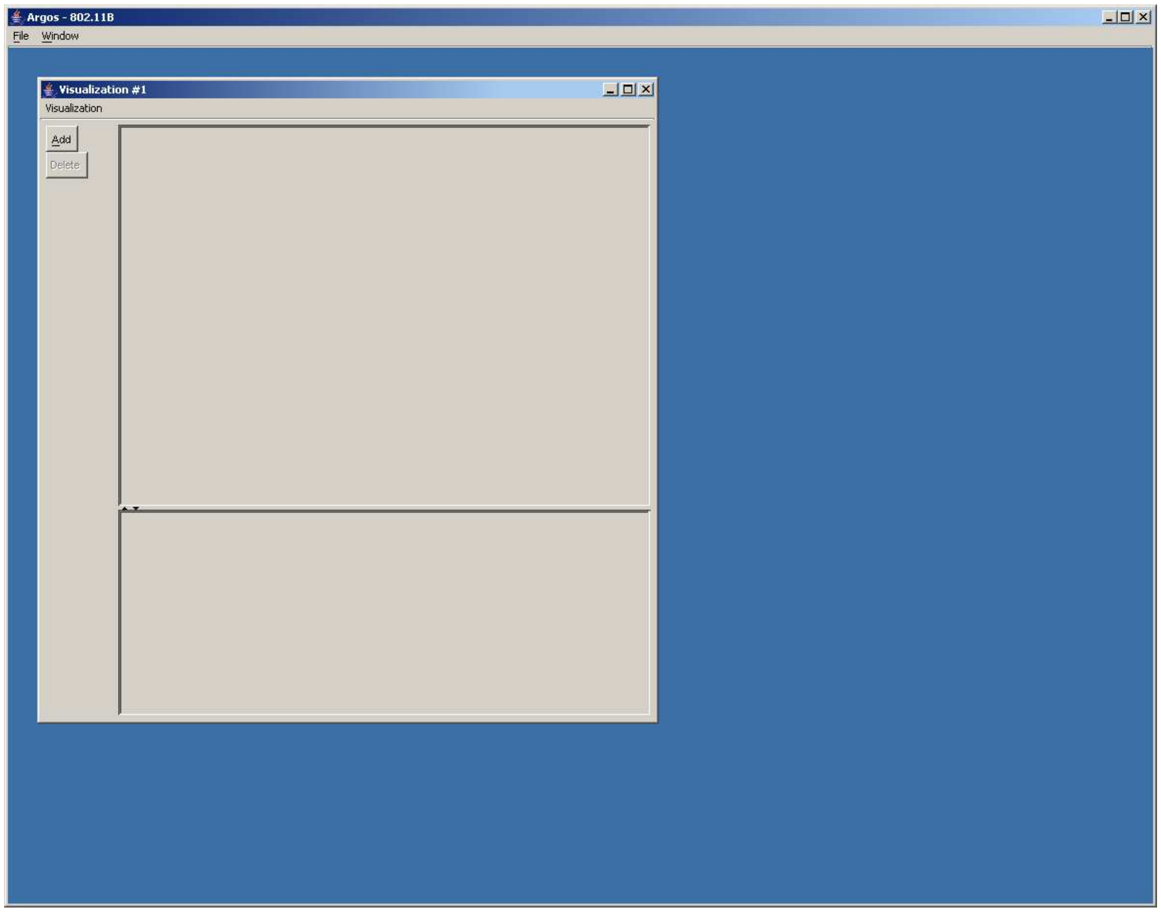


Figure A.2: Argos with a new visualization

Once this visualization is created, the real work can be done.

A.1.3 Adding Nodes

To add nodes, simply click the `Add` button on the panel on the left.

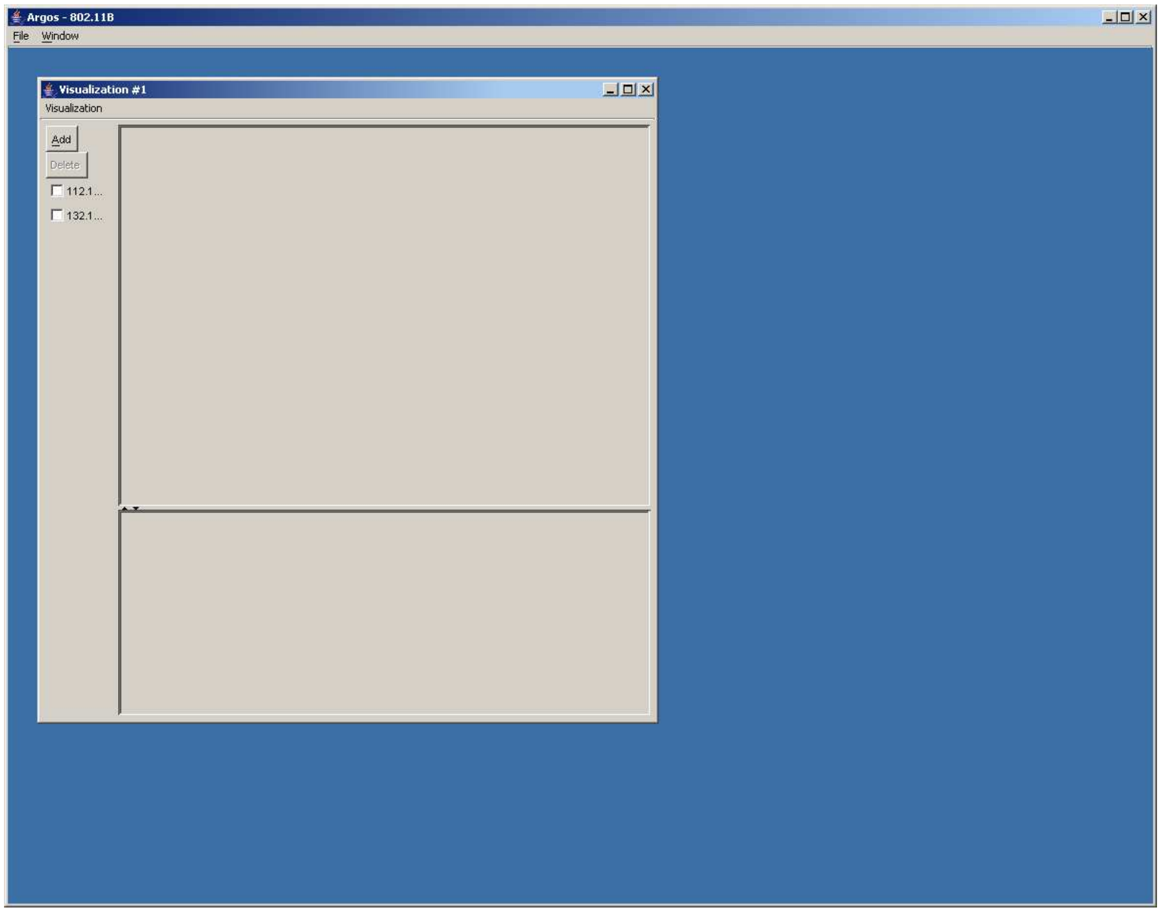


Figure A.3: 2 Nodes Added

Argos then parses through your available logs and displays a list of all of the nodes it has encountered.

A.2 Using Argos

A.2.1 Selecting a Node for Display

Now that ARGOS has found all of the available nodes in your logs, you can click on the checkbox next to a node to view a single timeline with its data.

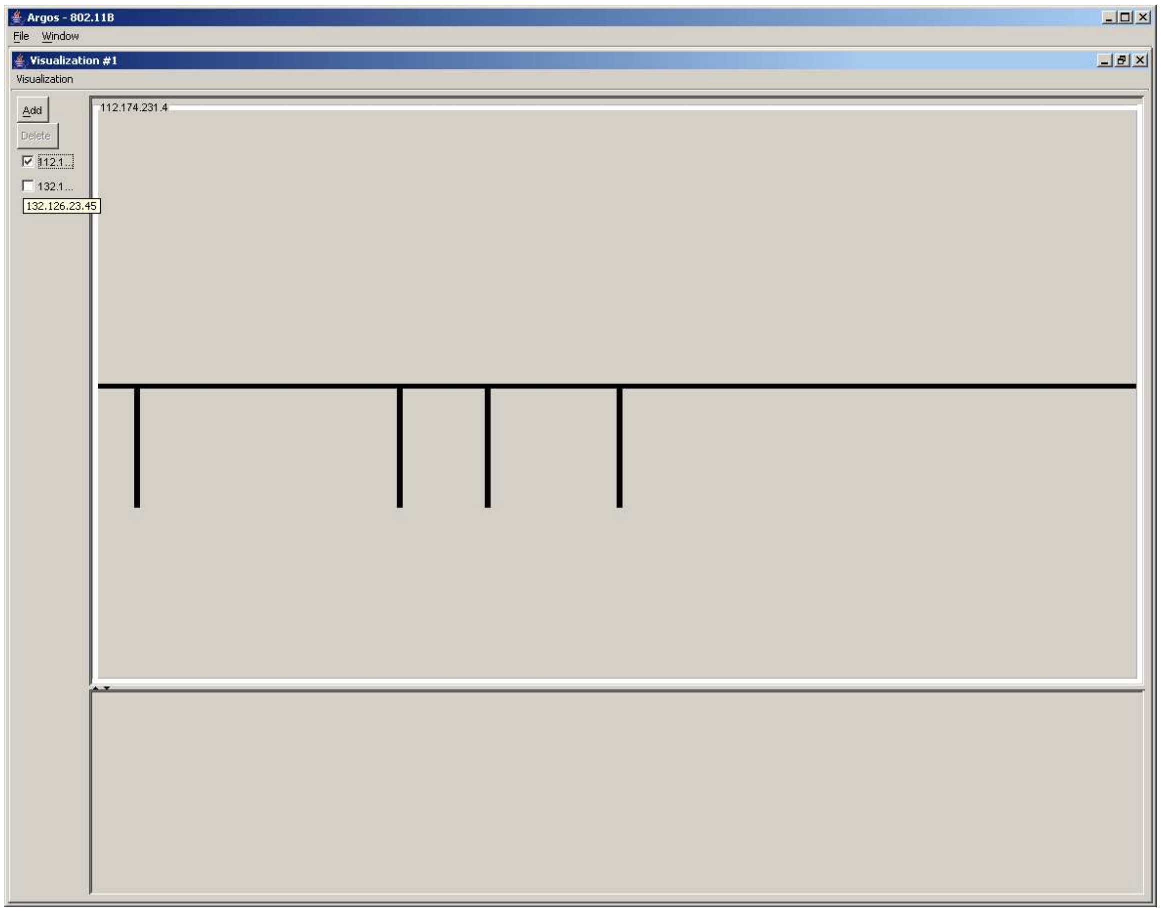


Figure A.4: Viewing a single timeline

From here, packet display and manipulation are possible.

A.2.2 Selecting Multiple Nodes

Displaying a single node doesn't usually give you much information, however. Since the real reason Argos is used is to monitor traffic *between* a set of nodes, you can click on all of the checkboxes next to the nodes you wish to view.

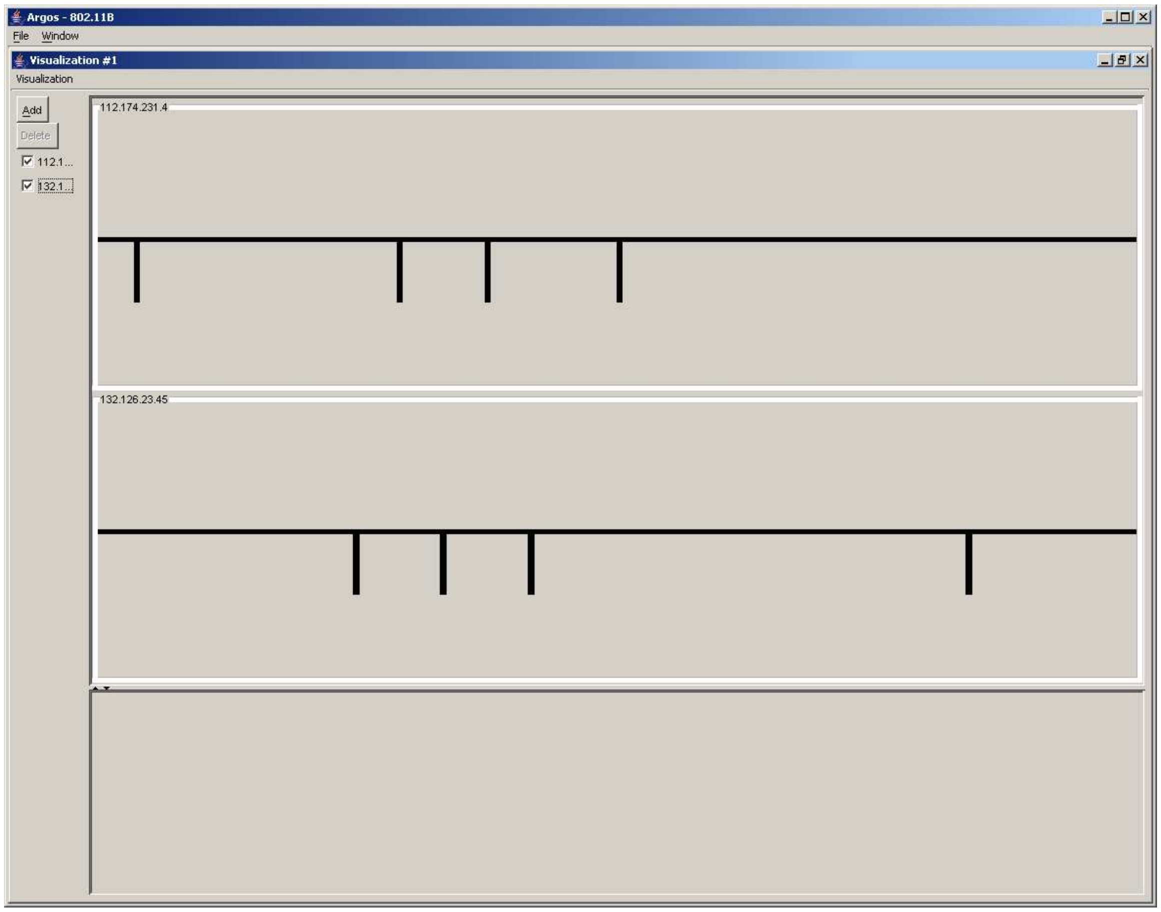


Figure A.5: Viewing multiple timelines

The only limit to the number of timelines is the number of nodes you have available to you.

Appendix B

802.11: Wireless LAN Frame Formats

B.1 Frame Formats

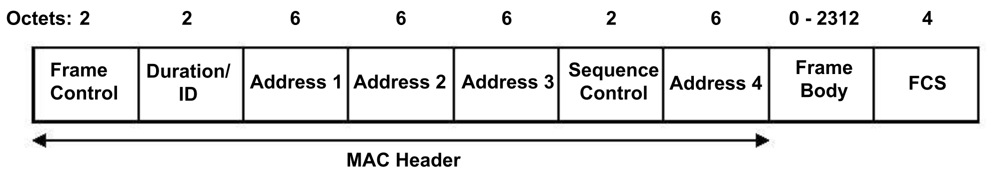


Figure B.1: Frame Formats: MAC Frame Format

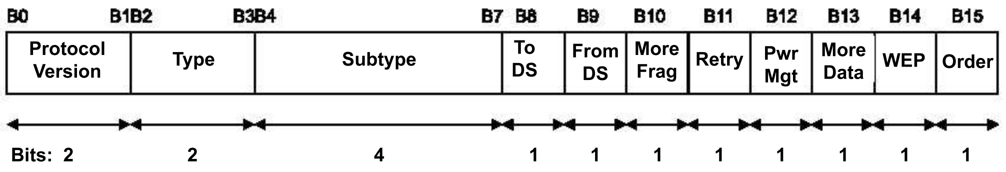


Figure B.2: Frame Formats: Frame Control Field

B.2 Individual Frame Type Formats

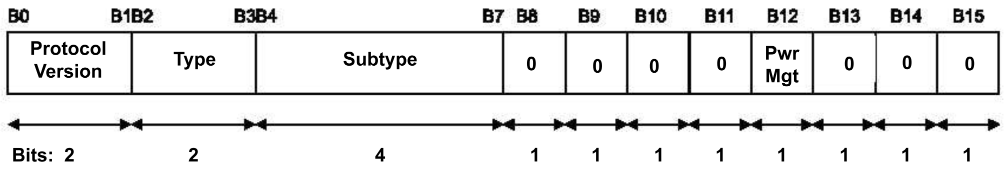


Figure B.3: Frame Control Field Subfield Values: Control Frames

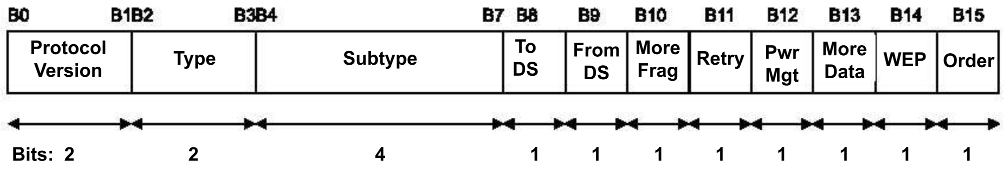


Figure B.4: Frame Control Field Subfield Values: Data Frames

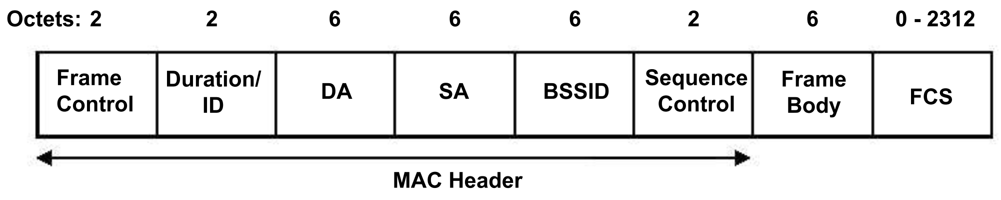


Figure B.5: Frame Control Field Subfield Values: Management Frames

Appendix C

Argos Class Structure

ARGOS's class structure can be found on the following page.

Glossary

802.11 - IEEE's official wireless networking specification

AP - Access Point; nodes on the network that allow for users to begin transmitting/receiving data from each other

COBOL - (Common Business Oriented Language) Clumsy, archaic programming language dating back to 1959.

contention - when a node detects that another node wishes to transmit, it pauses to allow the other transmission to terminate; they are therefore "contending" for use of the network

deadlock - a situation where two or more objects are present, and none can proceed, resulting in a standstill

GUI - Graphical User Interface

Java - A programming language created by Sun Microsystems that is designed to be platform-independent. This is achieved by the use of "virtual machines that allow Java programs to run on a particular operating system.

OS - Operating System

packet sniffers - tools designed to analyze individual packets and inspect their contents

painting - The rendering process that Swing uses to display objects on-screen

parser - A program that determines the structure of a string of symbols in some format (in our case a TCPDUMP file). A parser normally takes as input a sequence of tokens, and produces some kind structured data as output.

Swing - As defined by Sun Microsystems, the code name for a collection of GUI components that runs uniformly on any native platform that supports the Java virtual machine (JVM). Contrast with Abstract Window Toolkit.

TCPDump - an enumerated and structured output of all TCP packets transmitted between nodes

Virtual Machine - The mechanism the Java language uses to execute Java bytecode on any physical computer. The virtual machine (VM) converts the bytecode to the native instruction for the destination computer

Bibliography

[1] Gerald Combs "The Ethereal Network Protocol Analyzer",
`http://www.ethereal.com/`

[2] Marius Milner, "802.11b based wireless network auditing utility",
`http://www.netstumbler.com/`

[3] Mike Kershaw "802.11 wireless network sniffer",
`http://www.kismetwireless.net/`

[4] Sun Microsystems "Java Foundation Classes (JFC/Swing)"
`http://java.sun.com/products/jfc/`

[5] Yuchung Cheng to Marcos Paredes Farrera "Help with timestamp"
`http://www.tcpdump.org/lists/workers/2003/08/msg00423.html`

[6] Bill Fenner, Guy Harris, and Michael Richardson, "The libpcap Project",
`http://sourceforge.net/projects/libpcap/`

[7] Patrick Charles, "JPCap: Network Packet Capture Facility for JAVA",
`http://sourceforge.net/projects/jpcap/`

[8] Lawrence Berkeley National Laboratory "Protocol packet capture and dumper program",
`http://www.tcpdump.org`

[9] Institute of Electrical and Electronics Engineers `http://www.ieee.org/`

[10] Institute of Electrical and Electronics Engineers "802.11 Wireless Protocol Specification",
`http://slappy.cs.uiuc.edu/fall03/team1/files/802.11-1999.pdf`

[11] Open Source Development Network "World's largest open source software development site",
`http://sourceforge.net`