**Team No.:** 22

**Team Members:** Kameron Bielawski, Caleb Bryant, Steven Hu, Guhyoun Nam, Neel Patel

**Project Name:** The Watcher - https://github.com/neelpatelbiz/watcher_frontend

**Project Synopsis:** A webpage that displays network traffic information and packets captured by a computer with access to all network traffic in a local network.

**Project Description:**

For our project we are creating a website that will present data about network traffic in a local area network to its visitors. The site will be hosted on a machine performing packet capturing within the LAN by using a NIC in promiscuous mode. The value this project provides is giving users an easy way to gain at-a-glance insights into the traffic crossing their network. Graphs, charts, warnings, alerts, and other notifications are easy ways to engage the user while providing useful data.

This project has the potential to assist professionals and hobbyists in spotting issues learning about their networks. For instance, IT professionals and network administrators could discover nefarious traffic and unencrypted data being transmitted on a workplace's network. This would help protect the data of employees. Additionally, data center professionals could find the use of this application beneficial for sampling the types of traffic being sent within their fleet. When many requests are being retransmitted, this could signal an issue with a single machine or service and help system administrators diagnose problems.

By the end of this project, we hope to have a functioning traffic sampler that is capable of interpreting packets and aggregating network statistics. Depending on the rate of traffic, performance of the OS, and hardware we are using, the traffic analyzer may be able to analyze all packets being sent across the network and give interesting analytics with the full data set. Constructing a GUI for this analysis is a top priority and we hope to have a usable and convenient interface for understanding the data as well as providing information about possible security concerns.

**Project Milestones:**

**First Semester:**

-Find a packet processing framework for capturing network traffic in promiscuous mode (10/7)
-Capture traffic using a packet capture library(10/14)
-Document Packet Capture functionality (functions, classes for packet capture PCAP back end)(12/3)
-Generate statistics relating to number of packets captured of a specific protocol(10/21)
-Document Request Handler functions, httplistener class (REST api back end) (12/3)
-Format statistics into graphs to be displayed in a web page (10/25)

-Document react components, asynchronous functions, file hierarchy (Front end) (12/3)

## Watcher Development Gantt Chart

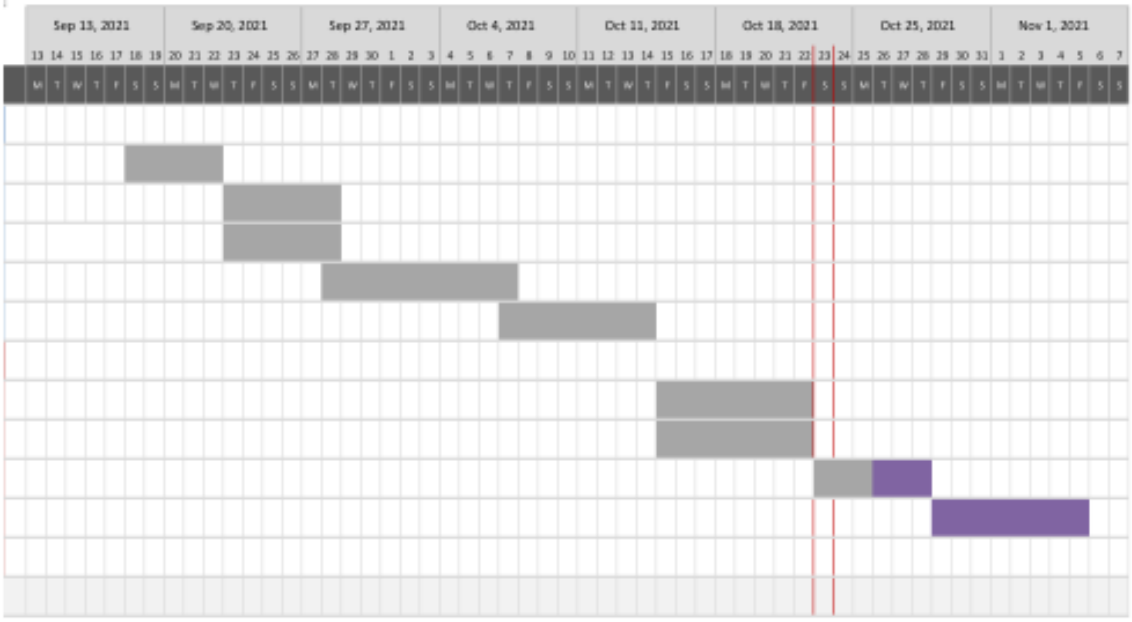
Team 22

Project Start: Sat, 9/18/2021

Display Week: 1

| TASK | ASSIGNED TO | PROGRESS | START | END |
|---|---|---|---|---|
| **Initial Planning/Research** | | | | |
| Gather Initial Basic Requirements | All | 100% | 9/18/21 | 9/22/21 |
| Research Packet Capturing Software | Neel, Caleb, Kam | 100% | 9/21/21 | 9/28/21 |
| Research Front End Frameworks and Tools | Steven Hu, Guhyoun Nam | 100% | 9/21/21 | 9/28/21 |
| Design Front End and Back End Interconnection | All | 100% | 9/28/21 | 10/7/21 |
| Front End and Back End Design Decisions | All | 100% | 10/7/21 | 10/14/21 |
| **Implementation Phase** | | | | |
| Implement Live Packet Capture | Neel, Caleb, Kam | 100% | 10/15/21 | 10/22/21 |
| Landing Page UI design | Steven Hu, Guhyoun Nam | 100% | 10/15/21 | 10/22/21 |
| Test web page interaction with back end using npm server and json-server | ALL | 50% | 10/23/21 | 10/28/21 |
| Research and Implement Json-Server/Rest-API within BackEnd | All | 0% | 10/29/21 | 11/5/21 |
| | | | | |

Insert new rows ABOVE this one




**Second Semester:**

-Filter packets based on protocol type, MAC address, or IP address fields (1/16)

-Write packets captured to a pcap file for later use or analysis and display in front end (1/27)

-Document PCAP file features (2/5)

-Determine additional features (different packet capturing methods/ data uses) (2/24)

-Implement possible new features and test features (4/24)
-Document new features (4/29)
-Create a test network using virtual machines for simulating LAN traffic (3/23)

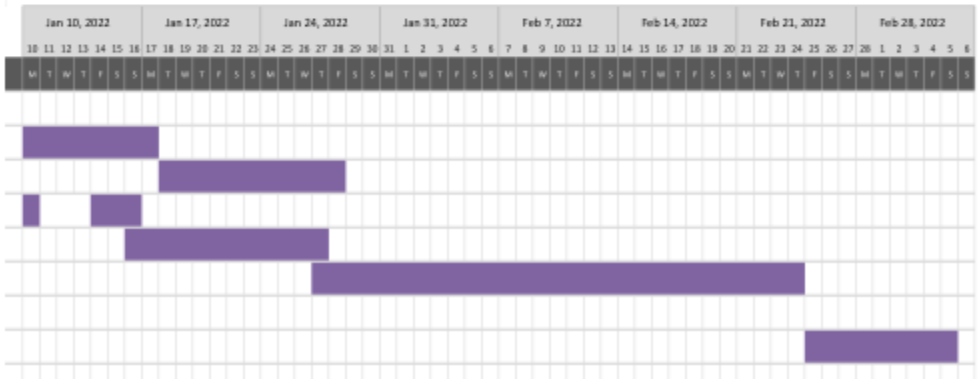
**Semester 2 Gantt Chart**

Team 22

Project Start: Sat, 1/1/2022

Display Week: 3

| TASK | ASSIGNED TO | PROGRESS | START | END |
|---|---|---|---|---|
| **Additional Features / Front End Remodelling** | | | | |
| Web Page Sketches / Modelling | All | 0% | 1/1/22 | 1/17/22 |
| Completion of Front End Diagrams | All | 0% | 1/18/22 | 1/28/22 |
| Allow User to Select Packets for Capture from Front end | All | 0% | 1/1/22 | 1/16/22 |
| Displaying PCAP information in front end | All | 0% | 1/16/22 | 1/27/22 |
| Research and Determine Additional Features (Frontend and Backend) | All | 0% | 1/27/22 | 2/24/22 |
| **Extended Functionality / Performance Enhancements** | | | | |
| Research Test Networks and Virtual Machines for testing the watcher | All | 0% | 2/25/22 | 3/11/22 |
| Test Application using virtual machines or test network | All | 0% | 3/12/22 | 3/23/22 |
| Test Other Packet Capturing / Processing Frameworks using PCap++ | ALL | 0% | 3/24/22 | 4/6/22 |
| Improve Performance with Alternate Packet Processing Framework | All | 0% | 4/7/22 | 4/14/22 |
| Implement Additional Features TBD | All | 0%% | 4/14/22 | 4/24/22 |
| *Insert new rows ABOVE this one* | | | | |




**<u>Project Budget:</u>**

Estimated Budget: $0

Tools we will need:

- Computer with NIC (promiscuous mode enabled) <- can be virtualized (using virtualbox)
- Ethernet switch with mirror port <- can be virtualized
- Other Computers for testing functionality <- can be virtualized

Special Training: Learning about Networks, the Internet, OSI model, Packet Types

**Software Design:**

*<u>How the software works:</u>*

The software we are designing is intended to run on a server within a rack of a data center or within a LAN. The software will utilize common features provided by switching hardware and ethernet adapters to perform packet capture on all packets on the LAN. Our software, "The Watcher", requires that the computer it is running on is connected to the mirror port of the switch so that it can receive all multicast and unicast traffic directed to any computer connected to the switch. To recognize this traffic, the NIC must also support "monitor mode". To create a rich user experience with multiple useful features, a front-end and back end must be constructed. The back-end must expose multiple services each providing a different functionality. Asynchronously serving the web page itself is important, but the site is not very useful without a variety of other asynchronous services providing data to be consumed by the client. These services must utilize data provided by the back end and provide the data in a way that allows the front end to consume it and display it in an understandable way for the user. In this way, the development of our front and back end is tightly coupled due to the cohesion needed between the services provided by the "Watcher" and their consumption by the client-side.

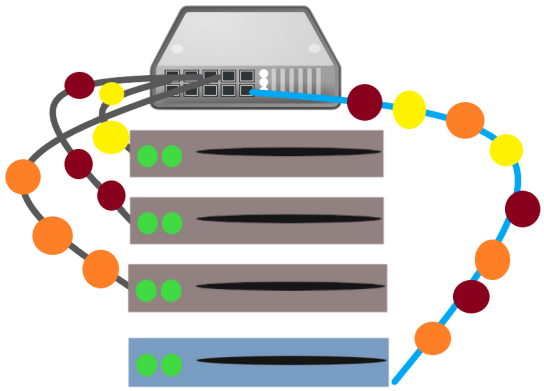


*Fig 1.) All Traffic Directed to Watcher (Bottom Server) via. Port Mirroring in the Switch*

Construction of the back-end was made possible using two main libraries. For the live capture of network traffic, we made use of the PCap++ library. This library provides API's for many useful packet capturing frameworks and libraries such as DPDK, PF_RING, and libpcap. For our purposes at the moment, libpcap is sufficient as it provides all of the packet analysis and network capture tools we need. It's performance in terms of latency, however, is limited to the performance of the linux network stack's raw sockets. PCap++ is capable of opening any interface on the machine for packet capture, capturing traffic on a specific interface, sending

packets from an interface, as well as filtering packets. We utilize the packet capture functionality provided by PCap++ as well as the packet filtering capabilities. Additionally the library allows reading and writing to pcap files, so that we can allow the user to choose certain types of packets they would like to view later. We assume that the server running the applications and services is utilizing a NIC in promiscuous mode, and that the required port mirroring capabilities are provided by the switch distributing traffic throughout the LAN or server rack. This will allow the duplication of all packets received by any computer on the network and the redirection of these duplicates to the network traffic analyzer. It is important to note that the "Watcher" is still fully functional without using a NIC without promiscuous mode enabled, but it will present a limited data set due to the NIC's inability to receive packets destined for alternate mac addresses.

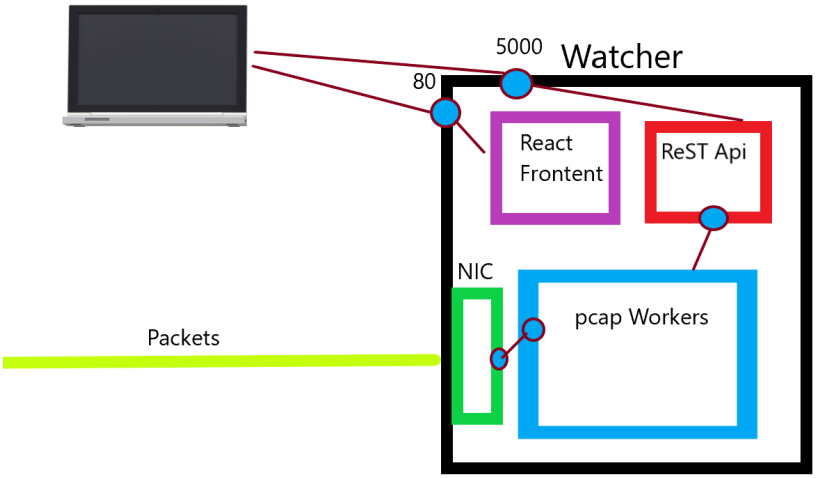



*Fig 2.) Watcher Software Structure; A machine in the LAN connects to the Watcher to inspect traffic (loads web page from port 80 and asynchronously received data via port 5000)*

The bulk of the work done by the application involves the capturing, processing, and filtering of packets received on the interface of the "Watcher". This is accomplished using the PCap++ library, which allows the live capture of packet data in multiple modes. Although PCap++ offers both asynchronous and blocking methods for capturing packets, we opted for the former since the main thread has other responsibilities besides capturing of packets. The main thread cannot be blocked while capturing a set number of packets since updating of data structures and other services must take place at the same time. Utilizing a callback function on the reception of every packet incurs a lofty overhead, however, which is not ideal for an application which may be processing large amounts of traffic destined for every node in the

LAN. Thankfully, PCap++ offers an asynchronous capture method that allows capturing of packets to take place on a separate thread while the main thread still has access to captured packets. We are then able to utilize the main thread for the other services.

The data captured by the packet capture thread is exposed to the front end using JSON since Javascript objects are utilized extensively for rendering the React Components on the site. This also allows complete separation of the front and back ends in terms of development as well as usage. One could utilize the front end with an alternate packet processing framework exposing the data, as long as the JSON data provided is the same. One could also create their own unique front end using the watcher thread application's ReST API. These design decisions allow flexibility regarding usage of the limited services currently provided by the back end and a standard for expanding the services provided as development continues. While the site could be loaded from a server different from the one the ReST API resides on, we assume that both the site and the asynchronous services reside on the same machine, the "Watcher".

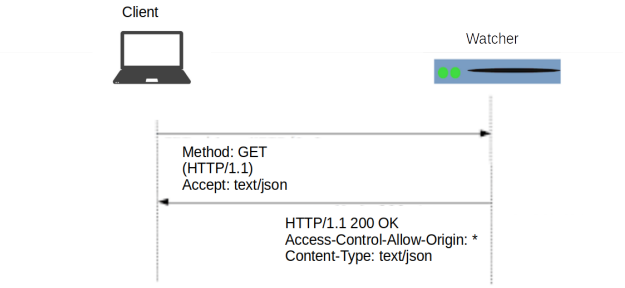



*Fig 3.) Client Asynchronously Requests Data being collected by the watcher (Data is received in json format)*

To implement the ReST API we utilized the C++ REST SDK created by Microsoft which allows the creation of asynchronous APIs using C++. This simplified the design of the backend by allowing the creation of a single multithreaded process that accomplishes a multitude of tasks. Namely, the process is able to capture packets on a given NIC, process the packets, store data related to the packets in memory, and deliver the data using a client-server communication model. The client-server communication model is made simple due to the fact that we are using ReSTful services utilizing the HTTP protocol. All data we intend to expose can be accessed using HTTP Get requests and the responses are provided by the backend process. The payload of the responses contain the JSON data that the client is requesting. The data is then utilized by the page loaded in the browser to create graphs, charts, and other analytics.

*Design Constraints:*

Designing asynchronous services in C++ can be challenging due to the language design. Compared to a language like Javascript, which was designed to support asynchronous

applications utilizing features like callback functions and promises, C++ leaves a lot to be desired. Thankfully, C++ REST SDK assists us in many aspects of asynchronously responding to HTTP requests, but we are limited to the classes provided by this library. The library supports assigning callback functions to http requests using the GET, PUT, POST, and DELETE methods, and creating completion of asynchronous actions, but the code is far less intuitive than similar ReST servers implemented in Javascript due to C++ REST SDK's use of function objects and utility classes for converting arrays, objects, and primitive types to and from JSON. Although this library has assisted us greatly in development, more complicated asynchronous scenarios, like updating specific pieces of data, or manipulating specific fields, could be a challenge.

We are also constrained by our schedule. A full understanding of traffic patterns within data centers or local networks would take time and research to obtain. Creating an application that creates notifications for every possible security threat or vulnerability is therefore out of the question, since our timeframe and understanding of network security is limited. Our goals for the application must be obtainable relative to the timeframe we are given. The services we can provide will be limited in scope, but if they are fully functional they will still provide valuable insights into the network traffic. Additionally, creating a scalable backend that exposes useful APIs will make expanding the front end website's capabilities easier as well as allow further additions after the deadlines imposed by the senior design lab.

**Ethical Issues:**

*Ramifications of Inspecting Network Traffic:*

When logging and inspecting network traffic, it is important to be aware of the potential consequences. Because unencrypted traffic and sensitive data could be received by our application, we must ensure that the data's usage is ethical. This is also encouraged by the ACM's code of ethics which (General Ethical Principles 1.2) states that a computing professional should "seek to minimize the possibility of indirectly or unintentionally harming others, [by following] generally accepted best practices unless there is a compelling ethical reason to do otherwise"(ACM). Our goal is to help others understand the network traffic on their LAN, and our application does not explicitly provide tools for extracting sensitive data from unencrypted packets or penetration testing of any kind. These features could be implemented by derivatives of our work, however. This is not our responsibility, though, since our implementation only provides ethically gathered information that is intended for use by system administrators.

*Ethical Ramifications of future features:*

If development on the "Watcher" continues into the future, a variety of features could be added that process many different fields of the packet data. If misused, an aggravated employee or system administrator, could utilize the tool in a way that is nefarious. For instance, if we provide a general purpose packet filtering tool within our application, the user could save packets utilizing potentially insecure protocols and inspect them later. Examples of this include HTTP packets not encrypted using the Secure Sockets Layer (SSL) and the telnet protocol. Someone

with access to a local network's "Watcher", could filter for these packets and inspect the potentially insecure data. Our intention in creating the "Watcher" is to allow system administrators to see if sensitive data is potentially being exposed and allow them to take measures to stop it. If large amounts of unencrypted data are being sent, system administrators would be able to use the "Watcher" to discover this. In this way, we intentionally follow the ACM's general principle that encourages respecting privacy. We are providing a tool that is best suited for ethical usage which, with the addition of this functionality, could prevent sensitive data leaks.

*Authorization to access computing/communication resources:*

Because of the nature of network monitoring software, computing professionals may use our software and find that action needs to be taken to protect their network. The ACM specifies that computing professionals should not access another's computer system or data without a compelling belief that it is consistent with the public good. While our application provides information that can be used to profile a network, it is done to assist ethical professionals. We are not ethically responsible for a computing professional who takes it upon themselves to resolve an internal issue by accessing a system without authorization, or harming others through inhibiting the functioning of an important system. Our intentions align with principle 2.8 of the ACM's code which encourages accessing communication resources only when authorized. This is because our application must be installed by a user with root access to the system and physical access to the network switch to provide the needed port mirroring and promiscuous capture features.

**Intellectual Property Issues:**

*Software Licensing (Use of external Libraries/Frameworks):*

Both the C++ REST SDK and React Framework use the MIT License which allows commercial use, modification, distribution, and private use of all the software provided. Because of the permissive nature of this license, we could use these libraries for creating a commercial application and distribute copies of this code. We could also sublicense this code with a more restrictive license allowing us to "copyleft" the software we create. The PCapPlusPlus library is licensed under the Unlicense which is just as permissive as the MIT License. We are not required to distribute source code with our product and we are free to use all three of the aforementioned pieces of software commercially and distribute them for use. These licenses provide us with the freedom to use and redistribute all of these pieces of software in any way we may choose in the future.

*Our Choice of License:*

With the freedoms given to us by the licenses of the code we are including in our project, we have no obligation to provide our code under the same license. Our choice of license could be more restrictive, like the GPL, which would require all derivations of our code to be copylefted and require that the sources be released. We could also create proprietary software, disallowing the modification or redistribution of our project. Our choice of project could lend itself to a proprietary license if we intended to sell our software for profit, and we did not want others to redistribute it for free. This is not our intention, however, since other network traffic analysis tools already in commercial use provide far more features than our project. Some implement advanced threat detection, machine learning, and behavioral analytics, to provide a far more robust solution than we are currently offering. With this in mind, we intend to license our project using the MIT License that the React Framework and C++ Rest SDK libraries use. This is because, after our development for the semester is over, we would like to see more changes made to the front and back end of our network traffic analyzer. Derivatives of our code with alternate front ends, advanced packet capturing, and improved data analysis are possible, and we would like to see what others can create.

*Commercial Use Possibilities:*

Because our software is licensed under the MIT license which is very permissive, others can utilize our work, and build more complex packet analysis tools and websites. This was our intention when deciding to release under this license. Companies and businesses could also utilize our software and redistribute it for profit under this license. While it is not in a state that would be competitive with alternatives already on the market, some modifications made by cybersecurity and data center professionals could turn our project into a viable tool. We would hope to be able to learn from any modifications or derivatives to our software, but we must allow others to redistribute our software without inclusion of the source code under the permissive MIT license. One strategy to see all derivations of our work would be to use a license like GPL, so that anyone using the software would have to release the sources along with any product they distribute. Although this would allow us to see anything created using our project as a starting point, it is not our intention to limit others in this regard. The tools and libraries that were instrumental for us in creating our project were all licensed under the MIT license or were in the public domain, which encourages us to provide the same availability and permissions.

**Change Log:**

*1)*

Previously we did not have a complete understanding of how network traffic would reach the watcher and what it would do with the traffic. After researching common local network practices and modern network switches, we formally specified how network traffic would reach the "Watcher"

How our application will receive traffic from the network:

-The computer will be connected to a port providing port mirroring functionality, passing along all traffic destined for any port to the "Watcher"

-The NIC of the "Watcher" will support promiscuous mode to capture traffic destined for mac addresses besides its own

*2)*

We now have a specification for how data will be transferred between the front end and back end as well as the format of the data.

-The data will be sent from the backend in a JSON format to be used by the React Front End. This is useful because the front end utilizes javascript, so receiving data in a form that can be easily transformed into a javascript object is important.

3)

Determining a method for querying data from the back end is important to our application. We decided that the front end will use simple HTTP requests to get data from the backend. This involves sending an HTTP get request, and awaiting a response from the backend, which contains JSON data in its payload.