# ECE496: Proposal

# Integrating Wireshark with an FPGA BPF Engine (FFShark) for Packet Analysis of High Speed Networks (100G)

Project Number: 2020174

Supervisor Name: Professor Paul Chow

Administrator Name: Inci McGreal (Section 5)

Students: Kanver Bhandal, Alexander Buck, Tobias Rozario

Date of submission: October 22, 2020

## Executive Summary

Wireshark is an open-source network analyzer commonly used to debug network connections throughout the networking industry. Currently, using Wireshark with a high performance processor allows debugging networks running up to 10G. FFShark, developed at the University of Toronto, is an open-source FPGA-based hardware tool for filtering network packets. Replacing the high performance processor with FFShark enables debugging faster networks, as FFShark has been proven capable of filtering packets at 100G.

However, the current implementation of FFShark has never been connected to Wireshark. This limits the usability of FFShark in the field as using FFShark requires a deep knowledge of its internals, a variety of scripts, and programming it using the Linux operating system running on the ARM chip connected to FFShark. This is a significant barrier for users of FFShark.

Our capstone project aims to rectify this problem by connecting Wireshark to FFShark so that anyone familiar with Wireshark can use FFShark to debug network connections. This will involve modifying or extending the Wireshark source code, allowing Wireshark to communicate with the Linux system running on the ARM chip, and implementing a simple interface to program FFShark and read its output data.

Our design will be the interface between Wireshark and FFShark. It must allow Wireshark to send filters to FFShark and FFShark to send back filtered packets to Wireshark. The design has limitations as certain unimplemented instructions must not be sent to FFShark, Wireshark cannot run directly on the Linux system running on the ARM chip, and FFShark can only transfer data through a 1G Ethernet port connected to the ARM chip. Goals of our design include minimizing the setup and installation required by the user, allowing communication between FFShark and Wireshark at the maximum transfer speed of the 1G Ethernet port, and providing support for FFShark on multiple operating systems.

FFShark is an implementation of a packet filter, but to be used in the field it requires a GUI that its users are familiar with so that it can be used effectively. Hence, the need exists for Wireshark to support FFShark.

**Table of Contents**

## 1.0 Introduction

FFShark [1] is a hardware network debugging tool developed by J.C. Vega and M.A. Merlini at the University of Toronto through P. Chow's research group. It can be used in high speed data centers capable of 100G speeds, that is, running at 100 Gbit/s. Originally, FFShark was intended to be connected to Wireshark [2], a packet capture program, but that functionality has not yet been achieved. Our project aims to complete this goal of connecting FFShark to Wireshark, thereby expanding the functionality of FFShark and increasing usability so that anyone familiar with Wireshark can use FFShark without in-depth knowledge of the FFShark internals.

## 1.1 Background and Motivation

Network analysis and debugging often involves inspecting packets, units of data, live while avoiding any disruptions to communications. Wireshark [2] is an open source software tool that aids this task by providing a graphical user interface (GUI) for capturing and viewing packets. Additionally, Wireshark allows setting filters within the program to accept only certain packets for viewing to prevent the user from being overwhelmed by irrelevant packets. While this filtering can be done at the "application level" through display filters, a means is also provided by the operating system. This reduces application runtime and memory usage compared to Wireshark receiving all packets and using only display filters in the application. As shown in Fig. 1, on Linux, the filtering is accomplished through the use of the Berkeley Packet Filter (BPF), a virtual machine in the kernel that can perform arithmetic and comparison operations to either accept or reject the packets and consequently pass them to Wireshark [3]. Thus, packets are received by the computer's network interface card (NIC) and sent to their destination program while at the same time being filtered using the BPF and may be additionally sent to Wireshark.

However, even with a high-performance processor, a computer is unable to filter packets with networks running at more than 10G due to speed limitations. As calculated by Vega *et al.* [1], a modern CPU running at its maximum frequency would need to receive, filter, and forward packets within 1.6 clock cycles to run at 100G, something not achievable in practice. Therefore, there exists external hardware that can perform 100G packet capture [4], [5]. Since these are commercial and closed source, FFShark was developed by Vega *et al.* [1] to provide an open source [6], 100G FPGA-based solution. FFShark runs on a Fidus Sidewinder 100 [7] consisting of two 100G Ethernet ports and a Xilinx Multiprocessor System-on-Chip (MPSoC) with an FPGA and ARM chip. This system is shown in Fig. 2. The FPGA implements FFShark for filtering while the ARM chip is used for communicating with FFShark. The ARM processor has an additional 1G Ethernet port for external communication.

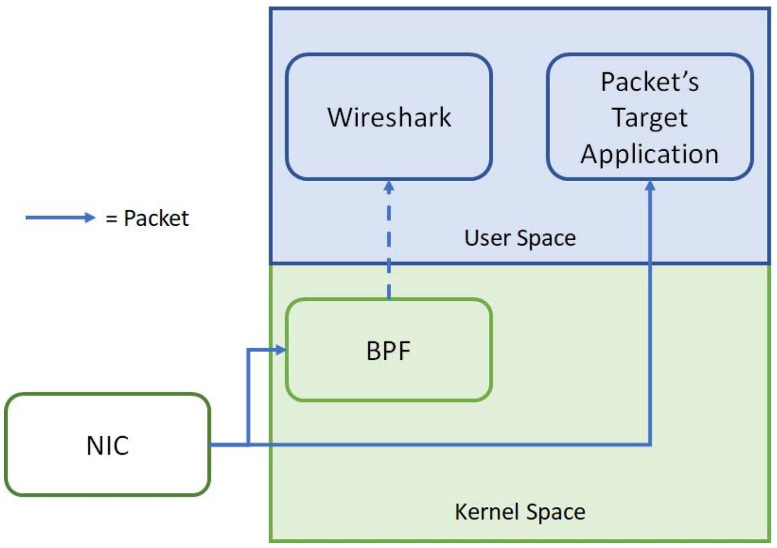


Figure 1 - The NIC sends a packet to the BPF and the target application. The BPF forwards the packet to Wireshark only if the capture filter accepts it.

Currently, FFShark does not have all the required interfaces to use it with Wireshark. Instead, users must first compile their PCAP filter [8] into machine code using a provided compiler and then load the filter onto FFShark using a shell script. The filtered packets are then output to the on-chip ARM processor that is running a terminal-only version of Linux Ubuntu. Thus, even though FFShark is compatible with Wireshark because it uses the same BPF mechanism that Wireshark relies on within the Linux kernel, it is not currently capable of being connected due to a lack of and incompatibility of interfaces.

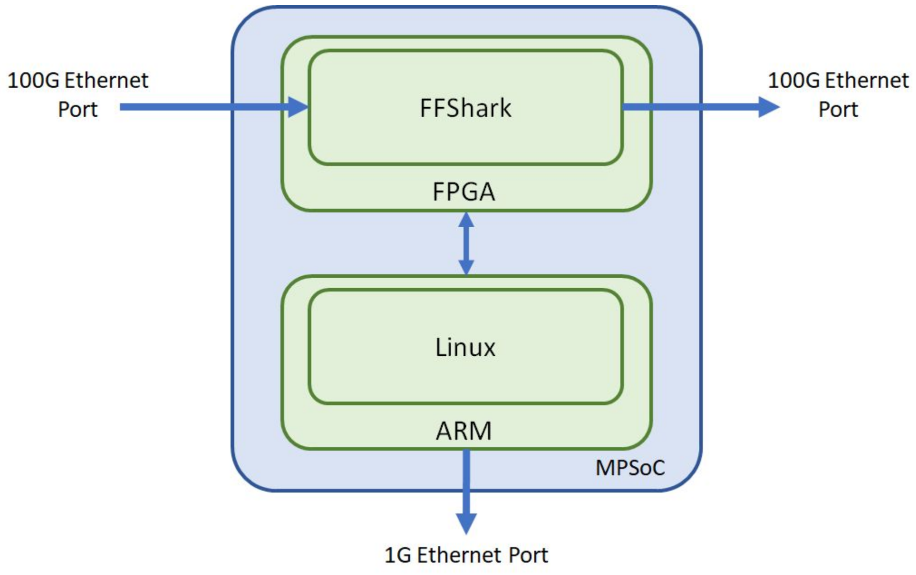


Figure 2 - A diagram of the MPSoC consisting of three Ethernet ports, an FPGA, and an ARM processor.

## 1.2 Problem Statement

While FFShark allows high speed packet filtering, the current implementation requires users to have an in-depth knowledge of the tool and accompanying scripts. Specifically, loading and changing filters requires compiling the PCAP filter syntax and then sending them to FFShark using terminal commands and scripts. Additionally, the captured packets are then sent to an ARM processor running a terminal-only version of Linux Ubuntu and therefore not readily viewable on a GUI. Due to the difficulties above with using FFShark, a network administrator will be less inclined to use FFShark for debugging as the cost of setting up and using FFShark is too high.

## 1.3 Project Goal

Our goal is to connect FFShark to Wireshark so that a network administrator can debug network connections using Wireshark with the actual packet filtering done by FFShark.

## 1.4 Scope of Work

The scope of our project is to send a packet capture filter from Wireshark to FFShark and display the filtered packets from FFShark on Wireshark, as shown in Fig. 3. This involves modifying or extending the open-source Wireshark code to recognize FFShark as a packet filter device, send capture filters to FFShark, and receiving the data or captured packets from FFShark. Wireshark will be run on a separate workstation rather than on the ARM chip directly connected to FFShark due to there being no GUI installed on the ARM chip. This requires Wireshark on a remote workstation and the ARM chip to transfer data using the 1G Ethernet port on the ARM chip. To program FFShark with a packet capture filter and read the filtered packets from FFShark, the on-board ARM chip must be used as FFShark does not expose a way to program registers without it [1]. Therefore, we will be creating an application to run on the ARM chip to handle programming FFShark. Refer to appendix A for the milestones and deliverables for our project, appendix B for the feasibility of our project, and appendix C for the system context diagram and a typical use case of our project.

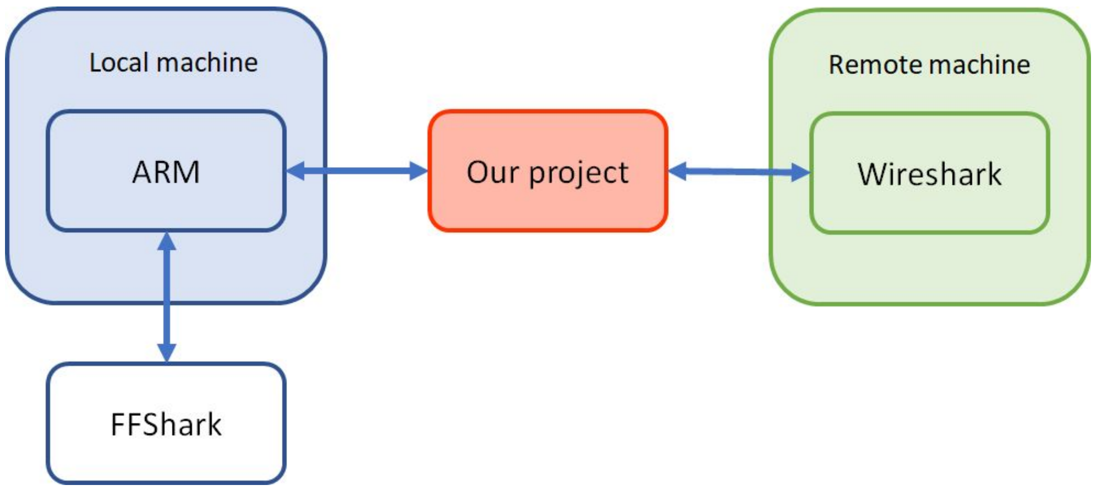


Figure 3 - Our project connects Wireshark running on a remote machine to the local machine connected to FFShark.

Our project does not consist of creating or modifying the FFShark internals as the implementation of FFShark we will be using is based on the work already done by Vega *et al.* [1]. We will not be creating a BPF compiler for FFShark as FFShark uses the standard BPF ISA (apart from *MUL*, *DIV*, and *MOD* instructions) and there are many libraries that can handle compiling BPF machine code [1], [9]. Finally, we are not creating a GUI to view the filtered packets from FFShark as we will be using Wireshark which implements a GUI.

For testing purposes it is sufficient to verify that a packet filter can be sent to FFShark using Wireshark and that Wireshark displays the filtered packets from FFShark. The displayed packets must match the packet filter provided to FFShark. There is no need to verify that packets can be filtered at a rate of 100G as that has already been proven [1].

## 2.0 Requirements Specification

Project requirements have been divided into three categories: functions, constraints and objectives. Functions, shown in Table 1, allow the design to meet the basic needs of industry professionals to debug network connections. Constraints, shown in Table 2, are restrictions the solution must follow. Lastly, objectives, shown in Table 3, are additional targets, and ways of determining the effectiveness of the design.

Table 1: Functions defining the basic criteria of the design

| ID | Function | Explanation |
|---|---|---|
| F1 | Display captured filtered packets from FFShark on Wireshark | FFShark is an FPGA design that can filter high speed network packets [1]. Wireshark needs to be used to display filtered packets to the user. |
| F2 | Send filters to FFShark via Wireshark | FFShark is a custom FPGA design, which is not supported by Wireshark. The design needs to allow Wireshark to send filters to FFShark. |

Table 2: Constraints restricting the design

| ID | Constraint | Explanation |
|---|---|---|
| C1 | Must use Wireshark packet analyzer to display FFShark | Client requires Wireshark to be used. |
| C2 | Must run Wireshark on an external workstation | The ARM chip, provided on the MPSoC, does not have a desktop environment for GUIs. As a result, Wireshark cannot be run directly on the ARM chip. |
| C3 | Must use the Ethernet port from the ARM chip to connect to the internet | The Ethernet port connected to the ARM chip is the only medium for FFShark to send over filtered packets to an external interface [1]. |
| C4 | Must not utilize the *MUL* (multiplication), *DIV* (division), and *MOD* (modulo) filtering instructions | FFShark omitted these instructions to optimize performance because they are rarely used in filtering applications [1]. |

Table 3: Objectives defining the goals of the design

| **ID** | **Objective** | **Explanation** |
|---|---|---|
| O1 | Limit the number of clicks from the Wireshark main menu in order to view FFShark captured filters (at most 2 clicks) | Two clicks were chosen as a maximum because users have to perform 1 click on the GUI to select the capture interface (Wi-Fi, Ethernet, LAN etc.) [2]. Only 1 extra click should be required at most to see FFShark captured data. |
| O2 | Limit the number of extra scripts to run per session when running our design with Wireshark (at most 3) | A maximum of 3 scripts has been chosen because many of the open-source designs interfacing wireshark to custom hardware have 2-4 [10], [11]. |
| O3 | Limit the number of scripts to run or plugins to install during the initial setup (2 at most) | Initial setup is the set of installation steps performed when running our design with Wireshark for the first time. Many custom open-source wireshark based projects require only 1-2 plugins/scripts to be installed/run [10], [11]. |
| O4 | Remove the need for users to access the Linux machine running on the ARM chip | Currently, users need to remotely access the Linux machine running on the ARM chip to program the FFShark bitstream onto the FPGA. The goal is to provide a level of abstraction such that users do not need to be aware of these steps. |
| O5 | Support 1G rate of captured filters | FFShark currently has a 1G Ethernet port connected to the ARM chip [1]. The goal is to utilize the entire bandwidth of the port to send filtered data to users. |
| O6 | Support several operating systems (minimum 2) | Wireshark supports Windows, Mac and Linux [2]. The goal is to have the design support at least two of these operating systems to provide support to more users. |

## 3.0 Conclusion

FFShark is an optimized FPGA hardware packet filtering design created for high speed network analysis. Currently, the cost of setup and usage of the tool is too high because users need extensive knowledge of the design and accompanying scripts to run it on a terminal interface. This brought upon the need of a graphical interface through which industry professionals will interface with FFShark. Based on the project requirements, it has been decided that Wireshark will be the GUI for performing packet analysis through FFShark. Our solution will connect FFShark to Wireshark.

**4.0 References**

[1] C. Vega, M. Merlini and P. Chow, "FFShark: A 100G FPGA Implementation of BPF Filtering for Wireshark," in *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM).* [Online]. IEEE, 2020, Accessed September 21, 2020. Available: https://www.fccm.org/past/2020/proceedings/2020/pdfs/FCCM2020-65FOvhMqzyMYm99lfeVKyl/580300a047/580300a047.pdf.

[2] Wireshark, "Wireshark User's Guide Version 3.3.1," 2020, [Online]. Accessed September 21, 2020. https://www.wireshark.org/docs/wsug_html_chunked/.

[3] S. McCanne and V. Jacobson, "The BSD Packet Filter: A New Architecture for User-level Packet Capture." in *USENIX winter,* vol. 46, 1993.

[4] "10/25/40/100G Packet Broker with PCAP Filtering," product Brief. [Online]. Accessed September 21, 2020. Available: https://www.bittware.com/102540100g-packet-broker-with-pcap-filtering/.

[5] "Full Line Rate Sustained 100Gbit Packet Capture," product Brief. [Online]. Accessed September 21, 2020. Available: https://www.fmad.io/products-100G-packet-capture.html.

[6] M. Merlini and C. Vega, "A versatile Wireshark-compatible packet filter, capable of 100G speeds and higher," 2020, [Online]. Accessed September 21, 2020. https://github.com/UofT-HPRC/fpga-bpf.

[7] Sidewinder: Fidus, "Sidewinder-100 Datasheet," 2018, [Online]. Accessed September 21, 2020. https://fidus.com/wp-content/uploads/2019/01/Sidewinder_Data_Sheet.pdf.

[8] V. Jacobson, C. Leres, and S. McCanne, *pcap-filter(7) - Linux man page*, Lawrence Berkeley National Laboratory.

[9] T. Carstens and G. Harris, "Programming with pcap," *Programming with pcapTCPDUMP/LIBPCAP public repository*, 2002. [Online]. Accessed September 21, 2020. Available: https://www.tcpdump.org/pcap.html.

[10] A. Cardigliano, "PF-RING - Wireshark Extcap", 2020, [Online]. Accessed September 21, 2020. https://github.com/ntop/PF_RING/tree/dev/userland/wireshark/extcap.

[11] N. Bowden, “WLAN-Pi - WLANPiShark2”, 2019,  [Online]. Accessed September 21, 2020. https://github.com/WLAN-Pi/WLANPiShark2.

[12] D. Lombardo, “sshdump,” *sshdump - The Wireshark Network Analyzer 3.2.7*. [Online]. Accessed September 21, 2020. Available:
https://www.wireshark.org/docs/man-pages/sshdump.html.

## Appendices

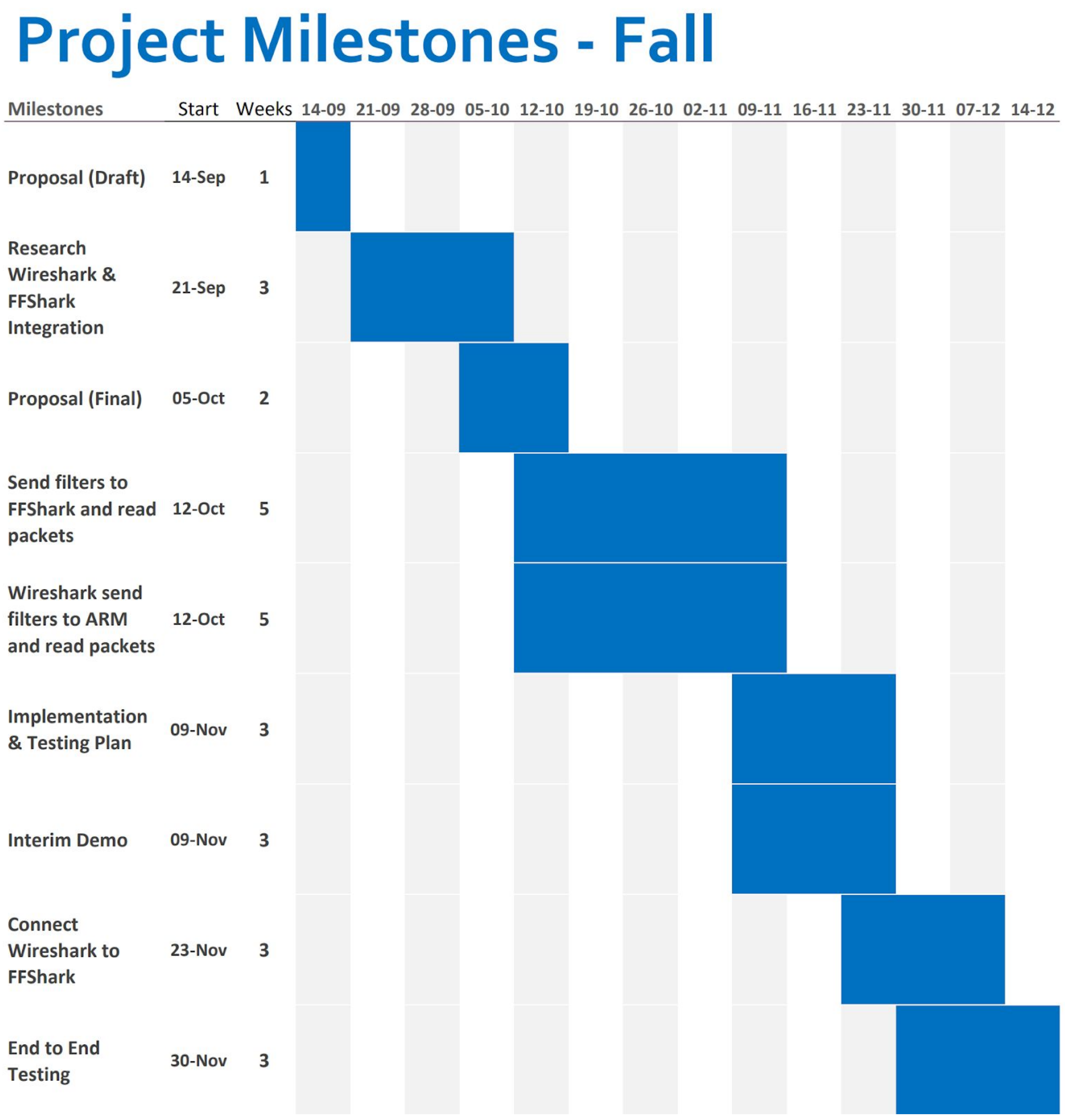
Below are appendices that provide further details on our project milestones and deliverables, the feasibility of our project, and a system context diagram to further explain our project and the scope of it.

### Appendix A: Project Milestones

Refer to Fig. 4 for the project milestones during fall semester and Fig. 5 for winter semester. Each figure contains dates to start and complete the milestones for the project and their deliverables. Project deliverables were included to help with planning time to devote to the project versus the deliverables. Some milestones can start in parallel or be done while working on another milestone as shown in Fig. 4 and Fig. 5.

Currently, the tasks and assignees to reach each milestone are not included in Fig. 4 and Fig. 5 as we do not have enough knowledge to break down each milestone into detailed tasks. This will be added as we start to research into what each milestone requires. For now we can break down the first milestone of “Research Wireshark & FFShark Integration.” Currently, Alex and Tobias are researching how to communicate between Wireshark and the ARM chip. Kanver is researching how to communicate between the ARM chip and FFShark.

For the interim demo, we plan to showcase that we have met our functional requirements i.e., created a connection between FFShark and Wireshark. This means that Wireshark can be used to send packet capture filters to FFShark and display the filtered packets from FFShark. The demo will send a small amount of non-filtered test packets (~20 or so) to FFShark and show Wireshark displaying the filtered packets from FFShark.



Project Milestones - Fall

| Milestones | Start | Weeks | 14-09 | 21-09 | 28-09 | 05-10 | 12-10 | 19-10 | 26-10 | 02-11 | 09-11 | 16-11 | 23-11 | 30-11 | 07-12 | 14-12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Proposal (Draft) | 14-Sep | 1 | 🟦 | | | | | | | | | | | | | |
| Research Wireshark & FFShark Integration | 21-Sep | 3 | | 🟦 | 🟦 | 🟦 | | | | | | | | | | |
| Proposal (Final) | 05-Oct | 2 | | | | 🟦 | 🟦 | | | | | | | | | |
| Send filters to FFShark and read packets | 12-Oct | 5 | | | | | 🟦 | 🟦 | 🟦 | 🟦 | 🟦 | | | | | |
| Wireshark send filters to ARM and read packets | 12-Oct | 5 | | | | | 🟦 | 🟦 | 🟦 | 🟦 | 🟦 | | | | | |
| Implementation & Testing Plan | 09-Nov | 3 | | | | | | | | | 🟦 | 🟦 | 🟦 | | | |
| Interim Demo | 09-Nov | 3 | | | | | | | | | 🟦 | 🟦 | 🟦 | | | |
| Connect Wireshark to FFShark | 23-Nov | 3 | | | | | | | | | | | 🟦 | 🟦 | 🟦 | |
| End to End Testing | 30-Nov | 3 | | | | | | | | | | | | 🟦 | 🟦 | 🟦 |

Figure 4 - Project Milestones for Fall Semester

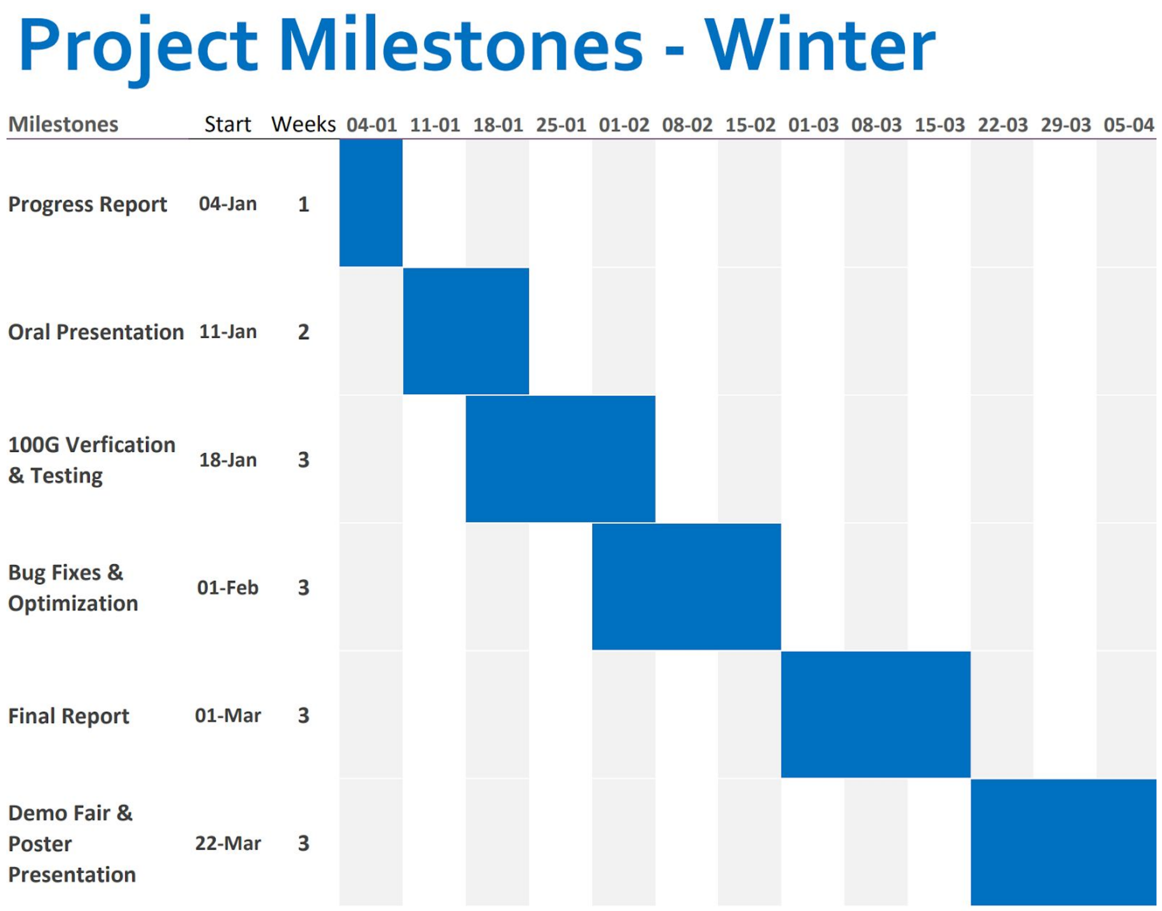


Figure 5 - Project Milestones for Winter Semester

**Appendix B: Feasibility Assessment**

This project will require knowledge and skills in programming, computer hardware, interfacing with hardware on Linux, computer networks, and understanding how Wireshark works. All three members of our team have taken or will be taking courses related to programming, computer hardware, interfacing with hardware on Linux and computer networks. We have also all completed co-op internships which required working with large code bases. Our experience in reading and learning from new code bases will facilitate learning about Wireshark through its code base. Additionally, there is a lot of documentation about Wireshark available and we can browse the source code to understand how Wireshark functions.

The resources required are all provided by our supervisor, Professor Paul Chow. He has an FPGA board that can be programmed with FFShark, an ARM chip to interface with FFShark, and three Ethernet ports (2 at 100G, 1 at 1G). He has set up a remote Linux computer that we can use for development and running Wireshark. The only concern is that there is only one FPGA board so we will need to coordinate access to it with other students who need to access it.

The biggest risk for our project was that we did not know how much work was required to modify Wireshark to capture packets remotely from a new BPF device. This presented us with a large scope for modifying Wireshark. However, we found several examples of other projects modifying Wireshark to be able to communicate with other BPF devices or work remotely as shown in [10], [11]. Using these examples, we researched into sshdump which is an extension to Wireshark that allows the capture of packets remotely using an executable running on a remote machine [12]. Using sshdump looks promising as it already interfaces with Wireshark, mitigating a lot of the risk of not knowing how much work is needed to modify Wireshark. Thus, we only have to look into modifying sshdump to communicate with FFShark and not the entire Wireshark codebase.

**Appendix C: System Context Diagram**

The system context diagram in Fig. 6 identifies the scope of our project. We will describe a typical usage of the system starting from the right side of the figure. A capture filter from Wireshark gets sent to our project as a PCAP filter. Our project will then forward this PCAP filter to the BPF compiler to convert it into machine code. The machine code is then passed to the ARM chip, which will program FFShark to accept only packets as specified in the filter. Raw packets are then output from FFShark back to the ARM chip. Next, these raw packets are converted by our project into the required format for Wireshark to view them and finally are passed back to Wireshark so a user can see the captured packets.

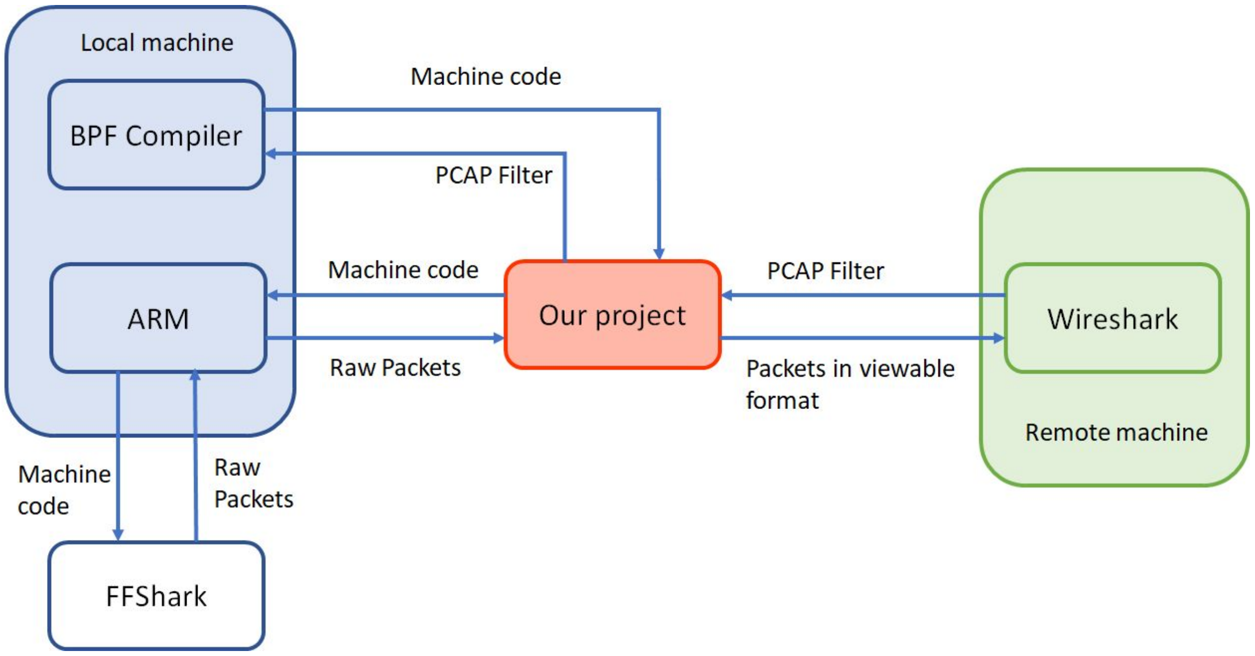


Figure 6 - Project System Context Diagram