Hardware-Accelerated Real-Time Packet Filtering and Multimedia Output System

Michael Grieco (mag2346), Adwyck Gupta (ag5016), Harry Zhang (hz3000)

Embedded Systems Spring 2025

I. Goal

Our main goal with this project is to design a hardware accelerator to process a stream of incoming network packets and then produce a graphical or auditory output based on the input stream. We intend to have the initial packets pass through a hardware accelerator to filter out duplicates or unauthorized data. Software can then look through the reduced dataset and produce data to be presented to the user either on the screen through VGA or as noise through the Audio Codec. In either route, we will implement a hardware block to interface between software and the output.

The filter accelerator will have two main processing elements: (1) the decryption piece to decrypt incoming traffic and (2) a counting Bloom filter to provide count data for our target bins; this will also help filter out packets based on a criteria (e.g., throw away duplicates, throw away requests that fall into specific bins). This derives from concepts of a Smart Network Interface Controller (SmartNIC) which strives to process data as close to the network interface as possible in hardware.

II. Constraints

A. Stream processing

We choose to focus on a stateless streaming application to minimize the memory footprint of the system. Stateless processing elements do not require knowledge of the previous data so a processing element need not buffer data in the on-board memory blocks. Additionally, in avoiding using the larger SDRAM, the hardware accelerators can be more autonomous as they need not go through the HPS to access memory.

B. Streaming data source

The streaming data on which we decide must include some raw impurities to necessitate a filter. As a result, we will focus on real-time datasets which go through as little processing as possible. For example, a real-time camera feed would include noise in the image from obstructions and extraneous events.

Additionally, the data should benefit from further processing in our system before presenting to the user. With the above real-time camera feed, we can clean up noise but simply presenting this to the user would not be very novel.

Taking this into account, we have three avenues to further explore. The first is real-time transit data. The MTA provides various APIs[1] to read real-time updates from the system. The advantages with this data set is that there are numerous ways to filter then present the data. We can implement a location filter so that a user can focus on disruptions in a specific region. However, that type of processing requires inspecting the packet contents and parsing the GTFS format[2] beyond decryption and typical IP headers, which would increase the hardware accelerator complexity. Another possibility is taking weather data from a network of sensors[3] and filtering old data that had previously been lost and then creating a scrollable heatmap for temperature or another value based on the user input.

C. Data presentation

We want to present meaningful results to the user either through audio, a visual, or a combination of both. Since we want to target a streaming application, having a real-time counter that triggers a notification bell noise could indicate to a user that some threshold has been reached. Furthering the thread of monitoring, we can present a bar graph or heatmap of the input categories to the user, and have a bell noise associated with a threshold being reached in one of the categories.

D. Hardware-Software interface

Three interfaces comprise our hardware-software interface. The first is passing data from the ethernet to our filter accelerator. The software will have a light background process that streams its received ethernet packets to the hardware so the accelerator can enqueue them before processing. This is required because the only on-chip network interface is in the HPS, so all packets must first pass through the HPS. One optimization we can make here to reduce bandwidth is when calling the *read* function on the socket file descriptor, we can have the destination be a memory location mapped to the accelerator's input queue. This way, the ethernet memory space can send data directly to the accelerator and the processor's local memory.

The second is the filter returning packets that are valid and should be processed further. While the implementation for this is not yet determined, we envision this as the filter writing the valid packets to a FIFO in an on-chip BRAM and then having the software receive an interrupt when new data is available in the FIFO.

Finally, once the software performs complex processing on the data to actuate graphical (and maybe auditory) information; it writes this to the VGA (and maybe audio codec) driver which will then display it on the screen (and maybe play the audio).

We want to avoid creating too complex of a path in the application flow as this will negate any benefits from accelerating the filtering and VGA/audio rendering in hardware.

E. User Interface

To interface with a USB peripheral, we will allow the user to control the processing using the keyboard.

III. Major Tasks

Data source (first priority): Find continuous sources of online data to provide requests that can probe our system. We need to make sure that the accelerator and program can parse and process data packets in the application format. This will determine the specifics of the later implementations.

Hardware filter: This covers the implementation of the preliminary filter in hardware. It should be configurable to allow for the user to control what is thrown away.

VGA controller: This will likely derive mostly from lab 3, but we will need adaptations to make it work in our system.

Audio controller: This will be a new module that interfaces between the software driver and the audio output. We will have to study the Audio protocol and likely insert a digital-to-analog converter somewhere in the pipeline.

Independent testbenches: Separate testbenches in Verilator for each of the hardware modules so that we can test them individually.

Linux network driver: This consists of the first HW-SW interface connecting the ethernet port to the filter queue. As mentioned above, the largest challenge will be reducing the load on the software to send packets to the accelerator.

Linux filter driver: This is the second HW-SW interface which will read from the FIFO queue containing the filtered packets once they have passed through the first accelerator.

Linux VGA/Audio driver: This is the final HW-SW interface which allows software to write graphical/auditory information to the drivers to then render to the user.

IV. References

1) MTA APIs: https://www.mta.info/developers
2) GTFS documentation: https://gtfs.org/documentation/overview/
3) weather data from National Weather Service: https://www.weather.gov/documentation/services-web-api
4) Realtime Weather data source: https://xmacis.rcc-acis.org/