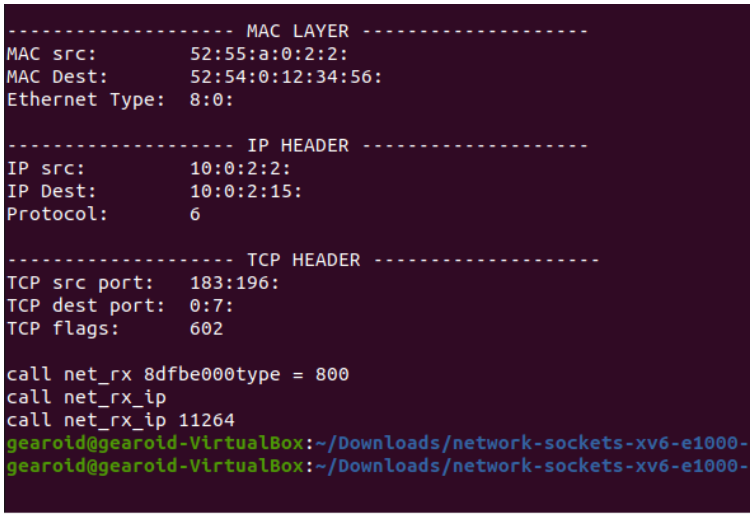
Assignment Three Theory Questions

Gearóid Sheehan R00151523

Operating Systems Engineering COMP8050

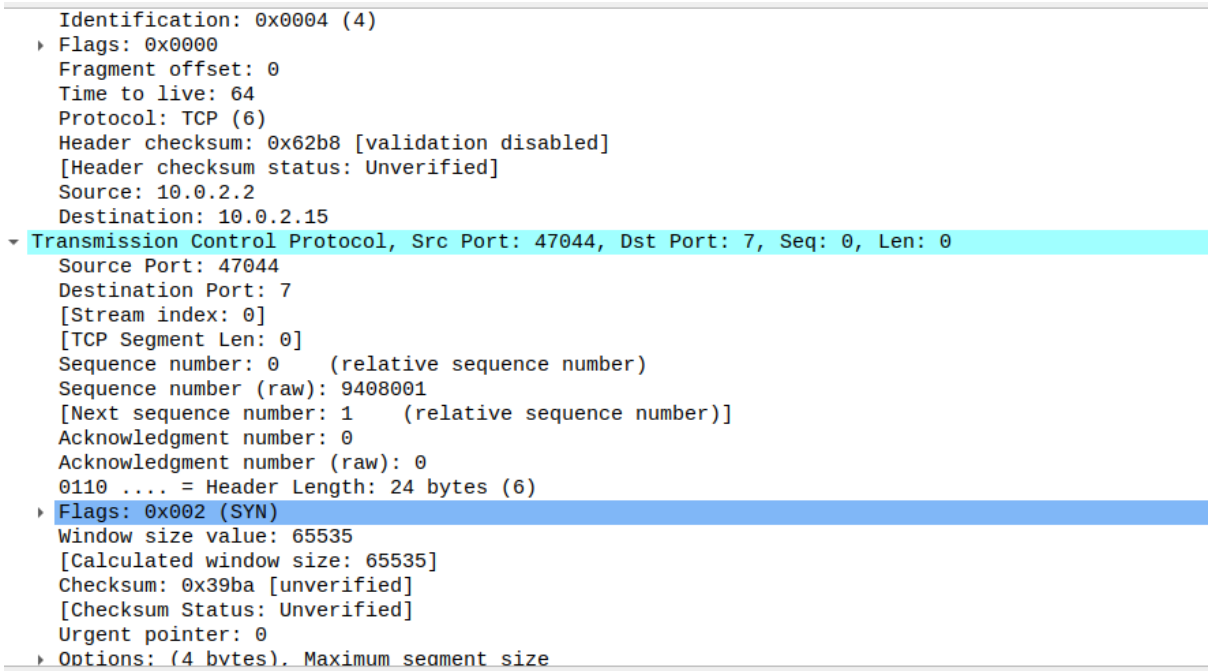
1) - Write a parse function in the file e1000.c in the code in network-sockets-xv6-e1000-lab.zip to give a human readable dump of received packet details.

Output of e1000.c parse function when accessing http://localhost:20001/



Output of Wireshark

```
▾ Frame 7: 58 bytes on wire (464 bits), 58 bytes captured (464 bits)
    Encapsulation type: Ethernet (1)
    Arrival Time: May 25, 2021 17:42:52.534007000 IST
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1621960972.534007000 seconds
    [Time delta from previous captured frame: 10.016021000 seconds]
    [Time delta from previous displayed frame: 10.016021000 seconds]
    [Time since reference or first frame: 17.648836000 seconds]
    Frame Number: 7
    Frame Length: 58 bytes (464 bits)
    Capture Length: 58 bytes (464 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:tcp]
    [Coloring Rule Name: Bad TCP]
    [Coloring Rule String: tcp.analysis.flags && !tcp.analysis.window_update]
▾ Ethernet II, Src: 52:55:0a:00:02:02 (52:55:0a:00:02:02), Dst: RealtekU_12:34:56 (52:54:00:12:34:56)
  ▸ Destination: RealtekU_12:34:56 (52:54:00:12:34:56)
  ▸ Source: 52:55:0a:00:02:02 (52:55:0a:00:02:02)
    Type: IPv4 (0x0800)
▾ Internet Protocol Version 4, Src: 10.0.2.2, Dst: 10.0.2.15
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 44
    Identification: 0x0004 (4)
  ▸ Flags: 0x0000
```



2) - Read the description below and also the "File descriptor layer" section in the xv6book and describe how the user is able to send and receive packets to/from the E1000 device with simple system calls such as read and write.

The socket layer in xv6 works very similarly to the file descriptor layer. In order to send packets, the write() function is used, and to receive packets the read() function is used. These are the same read() and write functions used when writing to a file. Firstly, confirmation of a successful connection is established by calling the ping() function in nettests.c. The connect() function is used here and returns an integer if the ping is successful. The connect() function is a user level function and takes the parameters of destination address, local port and remote port. When it is called, a socket is allocated to the incoming connection request. These sockets are an internal data structure called 'sock' in xv6 and are stored within a linked list. Inside each socket data structure, a buffer exists called 'rxq' of type mbufq, which is also a linked list. This buffer holds the queue of packets waiting to be received.

When packets are being sent to the device, the write function goes through the layers to sys_write() and filewrite() where the incoming files descriptor is checked. If the file is of type FD_SOCK then the sockwrite() function is called. A new buffer is allocated, and the incoming data is passed into the buffer. The net_tx_udp() function is then called to add the UDP header on top of the data. The IP and ethernet headers are then added, before the e1000_transmit() function is used to send the packet.

Similarly to when packets are being sent, when packets are being received by the device, the read function goes through the layers to sys_read() and fileread() where the incoming files descriptor is checked. When packets are being read from the rxq buffer, the buffer is checked to see if it is empty or not first using mbufq_empty(). If the buffer is empty, then a sleep() function is used to wait until the buffer receives packets.

3) - Give an overview of firecracker and its use of virtio for networking. In your answer focus specifically on virtio-sock.

Amazon Firecracker is a virtual machine monitor, or VMM, which uses the Linux kernel-based virtual machine to create and manage microVMs. It provides lightweight virtualization for serverless computing and removes the trade-off of choosing between hypervisor-based virtualization and Linux containers. With the provided minimal Linux guest kernel configuration, it offers memory overhead of less than 5MB per container, boots to application code in less than 125ms, and allows creation of up to 150 MicroVMs per second per host.

Virtio is an abstraction layer over devices in a paravirtualized hypervisor. In layman's terms, it is a virtualization standard for network and disk device drivers where just the guest's device driver "knows" it is running in a virtual environment and cooperates with the hypervisor. Firecracker uses Virtio for networking, as it is simple, scalable, and offers sufficiently good over-head and performance.

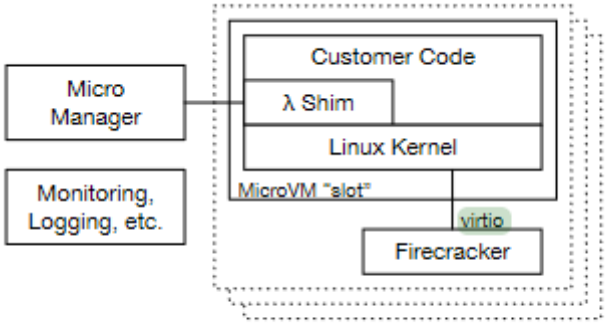


Figure 3: Architecture of the Lambda worker

The virtio-vsock device provides a zero-configuration communication channel between guest agents and hypervisor services independent of the guest network configuration. QEMU, Firecracker and the Linux kernel have virtio-vsock vhost support. The Firecracker vsock device aims to provide full virtio-vsock support to software running inside the guest VM, while bypassing vhost kernel code on the host. To that end, Firecracker implements the virtio-vsock device model, and mediates communication between AF_UNIX sockets (on the host end) and AF_VSOCK sockets (on the guest end).

In order to provide channel multiplexing the guest AF_VSOCK ports are mapped 1:1 to AF_UNIX sockets on the host. The virtio-vsock device must be configured with a path to an AF_UNIX socket on the host. There are two scenarios to be considered, depending on where the connection is initiated – Host Initiated Connections and Guest Initiated Connections.