WPI

CS 3516 - Computer Networks

Project 3 (100 points)
Assigned: Friday, November 25, 2022
Checkpoint: Tuesday, December 6, 2022 at 11:59pm
Due: Tuesday, December 13, 2022 at 11:59pm

Project 3: Overlay Networking

The goal of this project is to create an overlay network, which is the same technology used to implement several peer-to-peer networks. You will implement both the overlay end-hosts and the overlay routers. Fortunately, unlike a network-layer router, an overlay router is implemented as a user application and does not require kernel modifications. We will provide you with a configuration file for an overlay network topology. Your job will be to create appropriate headers, implement packet forwarding functionality, queuing, and TTL processing. Your program must be written in C/C++ and must work on a standard Ubuntu 20.04 Linux virtual machine (which is what the graders will be using).

Project Specification

The first step will be to set up an overlay network of end hosts and overlay routers. We will provide you with a configuration file to do this. Each machine participating in the overlay network will read the locally stored configuration file and figure out its role in the overlay network. Machines allocated the role of overlay routers will use the configuration file to start construct a forwarding table while end-host machines will simply look for files to transmit and wait for data to receive.

To avoid writing kernel level code, we will use UDP sockets for basic data transmission. All machines running your program will use the same UDP port number for sending and receiving packets. UDP is simply a transport mechanism for this project.

In each of the messages we transmit, we will create our own IP header, UDP header, and payload. Your router will use the IP header in this payload, perform a lookup, and determine the next hop for that packet. The router will then send the packet on to that next hop.

Configuration File

Lines in the configuration file are of five types: global configuration options (TYPE=0), router identification (TYPE=1), host identification (TYPE=2), router-to-router links (TYPE=3), and router-to-host links (TYPE=4).

```
0 QUEUE_LENGTH DEFAULT_TTL_VALUE
1 ROUTER_ID REAL_NETWORK_IP
2 END_HOST_ID REAL_NETWORK_IP HOST_OVERLAY_IP
3 ROUTER_1_ID ROUTER_1_SEND_DELAY ROUTER_2_ID ROUTER_2_SEND_DELAY
4 ROUTER_ID ROUTER_SEND_DELAY OVERLAY_PREFIX END_HOST_ID HOST_SEND_DELAY
```

The configuration file has been written to make parsing easy. Each entry is delimited with the newline character (`\n`) and each field on the line is delimited with a space character. Below is an example configuration file for the network shown in Figure 1. Note, the IPs in the 10.0.2.0/24 network represent actual IP addresses in a real subnet network (in this case, individual virtual machines).

```
0 10 3
1 1 10.0.2.101
1 2 10.0.2.102
1 3 10.0.2.103
```

```
2 4 10.0.2.104 1.2.3.1
2 5 10.0.2.105 4.5.6.1
2 6 10.0.2.106 7.8.9.1
3 1 100 2 110
3 2 200 3 200
3 1 500 3 500
4 1 1000 1.2.3.0/24 4 2000
4 2 120 4.5.6.0/24 5 120
4 3 1000 7.8.9.0/24 6 1000
```

In the first line of the configuration, we are setting the queue length to be 10 packets for each router's egress interface (end-hosts are assumed to have infinitely long queues). We further specify that each end-host will supply a TTL value of 3.

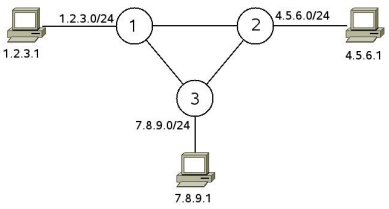

Figure 1: Example network

In the second line of the configuration, we are setting up router 1, which has the real network IP address 10.0.2.101. This IP is supposed to represent an actual machine on the real network. When the program is run on the machine, it must find its IP address in the configuration file. Other overlay routers will send packets to it on this IP address.

In the fifth line of the configuration, we are setting up an end-host (the upper left one in the diagram), which has the real network IP address 10.0.2.104. This IP is supposed to represent an actual machine on the real network. When the program is run on the machine, it must find its IP address in the configuration file. As an end-host, this host must be able to play traffic generated from a file, as needed. Further, it must write to a file any traffic that it receives. The end-host has the overlay IP address of "1.2.3.1", which it must use as the source in any overlay IPv4 packets it generates.

In the eighth line of the configuration, we are setting up a connection between two routers (routers 1 and 2). Router 1 can send a packet once every 100 ms to router 2. Router 2 can send a packet every 110 ms to router 1.

In the eleventh line of the configuration, we are setting up a connection between router 1 and the upper left end-host. Router 1 can send a packet every 1,000 ms to the end host. Further, an entire overlay prefix, 1.2.3.0/24, is reachable through this interface on router 1, even though we actually only have one host connected through it (you can imagine that the router and end-host are connection via a overlay switch that has no other end-hosts connected). The end host can send a packet every 2,000 ms to router 1.

Packet Headers and Format

In order to transmit packets, your program will create overlay layers 3, layer 4, and layer 5 on top of a UDP header, as shown in Figure 2. The first three layers (in light purple) show the headers that are automatically created for you when you use the UDP socket system calls. The top three layers (in light green) show the IPv4 and UDP header you must make yourself as well as the actual packet's payload (note, you can reuse the Linux IPv4 and UDP structures from /usr/include/ that you learned about in Project 2). The source and destination IP addresses of the overlay layer 3 will stay fixed, as determined by the sending host. At each hop, the IP addresses contained in the actual layer 3 will have to modified to indicate the next hop. We have provided source code at `https://cerebro.cs.wpi.edu/cs3516/cs3516sock.h` that gives you an interface to abstract away the UDP socket details.

Since the overlay packets will be encapsulated in a UDP and IP packet, the overlay routers must determine the UDP and IP data to populate these packet headers each time they receive a packet. The UDP ports are assumed to be fixed at some constant for all sends and receives. However, the IP addresses have to change each hop to make the packet traverse the overlay router network. To determine the IP address of the destination, the router must perform a longest prefix match on a set of overlay prefixes to determine the next hop for a given overlay IP address. The data obtained from this lookup will be the destination

physical IPv4 address for the next hop machine in the overlay network. **You can think of the UDP sockets (the purple region) as analogous to the datalink layer on a regular network:** while regular routers perform a longest prefix match to determine the physical interface through which the packet should be transmitted and the datalink layer information (such as Ethernet headers) used to reach the next hop router, you instead use longest prefix matching to determine the next hop IP address in the underlying (real) IP network.

Each packet's payload is limited to 1,000 bytes. Fragmentation is assumed not to occur in the overlay network. Further, packets are assumed to not use IP options. The type of service, fragmentation flags, fragmentation offset fields, and checksum fields must be set to zero. The IPv4 and UDP length fields must be set appropriately based on the packet size. The IPv4 header length must be set to to 5, the protocol field must be set to the value for UDP (17), and the TTL value set to the value specified in the configuration file by the end-host, but then is decremented by each router. The sending end-host will set the identification field to zero for the first packet it transmits to a given destination. The sending end-host will increment the identification field by 1 for each subsequent packet that it transmits to that destination.

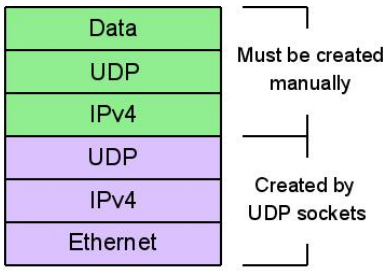

Figure 2: Packet Headers

Router Functionality

Longest Prefix Matching

As we discussed in class, longest prefix matching is often performed using the trie data structure in modern IP routers. Another approach is to use parallel hash table lookups. You may implement your lookup using either of these approaches, **but you must support correct longest prefix matching**.

Your router actually does not need to calculate routes (admittedly, a misnomer). We will only use routers in a full mesh, so each router will have a direct connection to the router servicing a prefix. Accordingly, you can just load IP prefixes and find the IP address of the router associated with those prefixes.

As an example, consider the eleventh line in the above configuration (`4 1 1000 1.2.3.0/24 4 2000`). Only `1.2.3` needs to be stored from the prefix. This can be used as the key to a hash table or trie insertion. The value of the insertion would be the IP address associated with host ID 4, which is 10.0.2.104. This would be the destination IP address that you would provide to the UDP socket (or, if you use the functions we provide, it would be the nextIP argument to the cs3516_send function).

Drop-tail Queuing

When a router has insufficient network capacity to forward incoming packets, it stores the packets in a queue based on the outgoing interface. When the queue becomes full, the router cannot accept any more packets. In the **drop-tail** queuing approach, these packets are simply dropped at the router until more space is available in the queue. We have specified sending delays in the network configuration file so that we can create resource contention on targeted routers. The delay, specified in milliseconds, indicates how long the end-host must wait before transmitting the next packet. This delay mimics the transmission time on actual end-hosts, though it is much larger in order to make the transmission perceptible during the demo. Whenever a packet is dropped because the queue is full, the router should record a message to its router log file, as indicated under "Expected Program Output."

Time-To-Live Processing

The overlay routers should decrement the TTL value in each overlay packet they process. The initial TTL value should be set by the sender end-host according to the value specified in the network configuration file.

If the TTL value becomes zero upon decrementing, the router must drop the packet. Additionally, the router must record a message to its router log file, as indicated under "Expected Program Output."

End-Host Functionality

Upon start up, end-hosts should scan their working directory for a file named send_config.txt. If such a file is found, the end-host must read the contents, which is all on a single line. The line will have three values, separated by a space. The first will be the destination IP address for the overlay IP header. The second value will be the source port for the overlay packet UDP header and the third will be the destination port for the overlay UDP header. The end host must then open a second file, send_body. It should then determine the file size (possibly via *fseek*), and transmit this file size as an unsigned 32 bit integer. The host must then transmit the contents of the file to the overlay IP address indicated, using it's own overlay IP address as the source address. The content must be divided into 1,000 byte payloads (or smaller for the last packet). The packets must be transmitted according to the rate-limit specified in its link. The sender queue is assumed never to become full (an end-host will buffer the required space), so no packets are ever dropped by the sending end-host.

End-hosts must also be prepared to receive incoming data (busy waiting is permitted, though short sleeps are also acceptable; **consider using a non-blocking *recv* call, *poll*, or *select***). When they receive a packet, they should write its overlay IP addresses and ports, to a file called `received_stats.txt` and write the contents to a file called `received`. For both files, if the file does not exist, it should be created. However, if the file does exist, the end-host should append to it rather than overwriting the existing content. Please note: the file contents may be text or binary; your hosts and routers should not expect the data to be text.

As a note, the sending system transmits the size of the packet in advance. Using this and the incrementing nature of the ident field, the receiver should be able to detect how many packets will be sent and which are missing.

Expected Program Output

End-hosts: In addition to the output required in the "End-Host Functionality" section, your program must print transmission data, based on files it received or transmitted. For each file the end-host transmitted, if any, it must print to standard out the size of the file transmitted and the total number of packets transmitted. For each file received, if any, it must print the size of the file received, total number of packets received, and the IP identification field numbers of any missing packets to standard output.

At each router: Each router should maintain a log file in which it records its activity. This information should be written to the ROUTER_control.txt file in the current working directory. All entries to the control log should be delimited by the new line character with each line in the following format:

```
UNIXTIME SOURCE_OVERLAY_IP DEST_OVERLAY_IP IP_IDENT STATUS_CODE [NEXT_HOP]
```

This entry indicates the current Unix epoch time (in seconds), the sender's overlay IP address, the recipient's overlay IP address, the value of the IP identification field, a status code, and the next hop information, if the packet was delivered successfully. The following are valid status codes:

- TTL_EXPIRED - The TTL value in the packet reached zero when decremented by the router and was discarded as a result.
- MAX_SENDQ_EXCEEDED - The queue for the router's egress interface was full, so the packet was dropped.
- NO_ROUTE_TO_HOST - The overlay IP address in the packet did not match any of the prefixes in the routing table. The packet was dropped.
- SENT_OKAY - The packet was okay and queued. With this status code, you must also include the NEXT_HOP IP address (either the physical or overlay address is acceptable).

Road Map

The following roadmap may be useful in completing this project.

1. Send and receive messages using UDP
2. Create and read overlay IP/UDP headers at sender and receiver. Have hosts generate random packets for transmission.
3. Use command line arguments to set role as router/end-host.
4. Have router mode decrement TTL values, log and drop packets at zero TTL. Have the router hardcoded with the forwarding destination.
5. Add drop tail queuing to the router.
6. Implement longest prefix matching at the router using a trie (or parallel hash tables) data structure. Manually load prefixes into the trie for the router to use when forwarding.
7. Use the configuration file to self-determine network roles.
8. Implement the end-host directory scanning/file reading/file writing behavior.

Advice: While you may be tempted to start with steps 7 and 8, it will likely be easier to implement those steps last since you will be more familiar with your code at that point. The above order may make it easier to incrementally build and test your design.

Checkpoint Contributions

Students must submit work that demonstrates substantial progress towards completing the project on the checkpoint date. Substantial progress is judged at the discretion of the grader to allow students flexibility in prioritizing their efforts. However, as an example, the checkpoint could be satisfied by completing the first four steps of the roadmap or by implementing an equivalent portion of the Project. **Projects that fail to submit a checkpoint demonstrating significant progress will incur a 10% penalty during final project grading.**

Deliverables and Grading

When submitting your project, please include the following:

- All of the files containing the code for all parts of the assignment.
- One file called `Makefile` that can be used by the `make` command for building the executables. It should support the "`make clean`" command and the "`make all`" command.
- A document called `README.txt` explaining your project and anything that you feel the teaching staff should know when grading the project. Only plaintext write-ups are accepted.
- Any test files used to confirm transmissions.

Please compress all the files together as a single .zip archive for submission. As with all projects, please **only use standard zip files** for compression; **.rar, .7z, and other custom file formats will not be accepted**.

The project programming is only a portion of the project. Students should use the following checklist in turning in their projects to avoid forgetting any deliverables:

1. Sign up for a project partner or have one assigned (URL: `https://ia.wpi.edu/cs3516/request_teammate.php`),
2. Submit the project code and documentation via InstructAssist (URL: `https://ia.wpi.edu/cs3516/files.php`),
3. Complete your Partner Evaluation (URL: `https://ia.wpi.edu/cs3516/evals.php`), and

A grading rubric has been provided at the end of this specification to give you a guide for how the project will be graded. No points can be earned for a task that has a prerequisite unless that prerequisite is working well enough to support the dependent task. Students will receive a scanned markup of this rubric as part of their project grading feedback. Students in teams are expected to contribute equally; unequal contributions may yield different grades for the team members. Students who work in teams but do not submit a Partner Evaluation by the project deadline will incur up to a 5% penalty on the project grade.