Problem 1 (220 pts)

1.1 Overall design

We will build an overlay network utilizing application layer routing to forward packets through paths that go through designated IP devices when packets are sent from source to destination. The executable binary, `zigzagrouter`, runs on a subset of Linux PCs in both lab machines (pod and amber). For example, pod1-1.cs.purdue.edu, pod3-4.cs.purdue.edu, amber02.cs.purdue.edu, amber05.cs.purdue.edu. For readability, the symbolic domain names of the IP addresses are used but your code should assume IP addresses in dotted decimal form. On each machine where `zigzagrouter` executes, three UDP sockets are created that bind to three unused ports. The port numbers are output to stdout so that the human operator knows which simplifies coding. At the sender, in our case the UDP ping client, `mypingcli`, from Problem 1 of lab2, an overlay network configuration binary, `zigzagconf`, is executed. To each IP device running `zigzagrouter`, `zigzagconf` sends a management UDP packet to the first port bound by the device informing it how to forward UDP packets arriving on the second and third ports bound by `zigzagrouter`. In networking parlance, we refer to the first port number as the control plane of the overlay network through which management information is communicated.

For example, `zigzagconf` running on amber07.cs.purdue.edu may send a UDP packet to pod3-4.cs.purdue.edu on the first port bound by `zigzagrouter` specifying

```
55000 39000 amber05.cs.purdue.edu 40000 50001 pod1-1.cs.purdue.edu
```

which instructs `zigzagrouter` on pod3-4.cs.purdue.edu to forward a UDP packet arriving on port 55000 (its second port) to amber05.cs.purdue.edu:39000. Similarly, UDP packets arriving on port 40000 (its third port) should be forwarded to pod1-1.cs.purdue.edu at port 50001. We refer to ports 55000 and 40000 at pod3-4.cs.purdue.edu as comprising part of the overlay network's data plane. The final destination of the `myping` UDP app from lab2 runs the `myping` server app, `mypingsrv`, say on pod2-2.cs.purdue.edu at port 55555. By sending instructions over the UDP-based control plane to each of the application layer routers pod1-1.cs.purdue.edu, pod3-4.cs.purdue.edu, amber02.cs.purdue.edu, and amber05.cs.purdue.edu they can be configured to forward `mypingcli` UDP packets from amber07.cs.purdue.edu to `mypingsrv` running on pod2-2.cs.purdue.edu through four intermediate hops, say, amber02.cs.purdue.edu -> pod3-4.cs.purdue.edu -> amber05.cs.purdue.edu -> pod1-1.cs.purdue.edu.

The return path from `mypingsrv` running on pod2-2.cs.purdue.edu to `mypingcli` on amber07.cs.purdue.edu which uses the third port of the data plane need not be symmetric. For example, the return path may be pod1-1.cs.purdue.edu -> pod3-4.cs.purdue.edu -> amber05.cs.purdue.edu -> amber02.cs.purdue.edu. In this specific overlay network, the first hop (amber02.cs.purdue.edu) and last hop (pod1-1.cs.purdue.edu) of the forward data path will switch roles in the return data path. That is, pod1-1.cs.purdue.edu is the first hop of the return path and amber02.cs.purdue.edu the last hop of the return path. Since `mypingcli` binds to an ephemeral port number and reuse `mypingcli` as a legacy app as is (without changing its code to output its port to stdout), `zigzagconf` will send the following UDP management packet to the first hop on the forward path amber02.cs.purdue.edu

```
25000 55000 pod3-4.cs.purdue.edu 26000 0 0.0.0.0
```

In the above, 25000 represents the second port bound by amber02.cs.purdue.edu and 26000 its third port. The values 0 and 0.0.0.0 specify that `zigzagrouter` running on amber02.cs.purdue.edu should remember the IPv4 address and port number of the first UDP packet received on port 25000 (i.e., forward path) and replace 0.0.0.0 and 0 with their values so that the response sent from `mypingsrv` can be delivered to `mypingcli` running on amber07.cs.purdue.edu. To forward a UDP means to extract its payload and transmit it as the payload of a new UDP from application layer router to its next hop (an application layer router, source, or final destination).

1.2 Legacy compatibility

The UDP ping server, `mypingsrv`, runs as is, and the UDP ping request will appear as originating from the last hop pod1-1.cs.purdue.edu in the above example. The UDP ping client, `mypingcli`, does not run fully in a legacy compatible manner since its command-line argument must specify the coordinates of the first hop in the forward path. That is,

```
mypingcli amber07.cs.purdue.edu amber02.cs.purdue.edu 25000
```

instead of the final destination's coordinates. In production systems, functionality of zigzagrouter are implemented as dynamically loadable kernel modules in operating systems such as Linux and Windows. The kernel modules act as hooks in its protocol stack that allow intercepting and modifying (or mangling) messages depending on where the hooks are installed. For example, in a simple VPN where a single intermediate hop acts as the conduit -- i.e., packets are not bounced around multiple times as in a pinball machine -- a legacy client app such as a web brower works as is with final destination IP addresses. However, transparent to the client app the kernel module will tunnel client messages through the VPN server which will then appear to the server that the request is originating from the VPN server.

In the case of Purdue's campus network, some services are only honored if the source IP address belongs to one of the Purdue's CIDR IP prefixes. By default, these services would not be accessible off-campus through ISPs since their IP addresses are not part of Purdue's IP prefix blocks. An on-campus VPN server that acts as a conduit can allow the same services to be accessed off-campus. Since modifying kernel behavior to implement network protocols is outside the scope of the lab assignments, we are approaching the problem from a less transparent and elegant manner. Linux and Windows kernel programming to implement these features is not difficult but requires additional background and skills beyond system programming.

1.3 Implementation details

The overlay network configuration app, `zigzagconf`, should read from a configuration file (ASCII text file), `zzoverlay.dat`, to whom on the control plane what forwarding configuration information should be transmitted. For example, one entry in `zzovrelay.dat` for the application layer router `zigzagrouter` running on pod3-4.cs.purdue.edu would be

```
pod3-4.cs.purdue.edu 33333
55000 39000 amber05.cs.purdue.edu
40000 50001 pod1-1.cs.purdue.edu
```

where pod3-4.cs.purdue.edu specifies the IP address (in dotted decimal form) of an application layer router and its first (i.e., control plane) port number 33333, and 55000 39000 amber05.cs.purdue.edu 40000 50001 pod1-1.cs.purdue.edu specifying the forwarding instructions on the forward and return paths. The first entry of `zzoverlay.dat` is an integer specifying the number of application layer routers, followed by three lines separated by newline. The IP addresses (dotted decimal) and port numbers should be separated by spaces.

A UDP management packet transmitted over the control plane has the format: 2 bytes for second port number, 2 bytes for next hop port number, 4 bytes for IPv4 address for the forward path, 2 bytes for third port number, 2 bytes for next hop port number, 4 bytes for IPv4 address for the return path. Hence a total of 16 bytes.

Whenever a control plane message is received, `zigzagrouter` should output to stdout a timestamp followed by the content of the message after installing it in a data structure that represents its forwarding table. When a data plane packet is received, `zigzagrouter` should output to stdout a timestamp followed the sender's IP address (in dotted decimal form) and port number, and the payload of the UDP packet. When `zigzagconf` transmits a control plane UDP packet, print to stdout the timestamp, the destination IP address and port number, and the payload of the packet. The output will help monitor and debug `zigzagrouter` and `zigzagconf` actions. When `zigzagrouter` executes, it creates a socket to be used for the control plane. Instead of binding to an ephemeral port number, provide the port number to be used for the control plane as a command-line argument of `zigzagrouter`.

`zigzagrouter` needs to handle three socket descriptors (1 for control plane, 2 for data plane). Use the `select()` system call which facilitates monitoring events on multiple descriptors.

1.4 Testing

Test the overlay network application that utilizes multiple hops across both pod and amber lab machines. Make sure to use IPv4 addresses in dotted decimal form, not their symbolic (i.e., domain) names. That is, `128.10.25.214` in place of pod3-4.cs.purdue.edu. Run the UDP `myping` app without intermediate application layer routers to gauge RTT. Then run the app with the same parameters specified in `pingparam.dat` (keep it simple). Test and verify that the overlay network application works correctly. The UDP ping app should report larger RTT values that increase with the number of hops, but otherwise work as before. Discuss your finding in lab6.pdf. Submit your work in `v1/` following the convention of previous labs (e.g., `Makefile`, `README`).

> Note: This problem may be tackled as a group effort involving up to 3 people. If going the group effort route, please specify in lab6.pdf on its cover page who the members are, who did what work including programming the various components, performance evaluation, and write-up. If you participated in a group effort in lab5, the members of lab6 cannot overlap. Whether you implement lab6 as an individual effort or make it a group effort is up to you. Keep in mind the trade-offs: group effort incurs coordination overhead which can slow down execution, especially for a 2-week assignment. Benefits include collaborative problem solving and some parallel speed-up if efficiently executed. Regarding late days, for a group to use `k (= 1, 2, 3)` late days, every member of the group must have `k` late days left.

> Note: For students who have earned 100+ extra credits (bonus problems in lab1-lab5 and late days unused), you may consider solving a simpler problem where instead of `zigzagconf` communicating forwarding instructions over the control plane, the information is read from a configuration file, `zzone.dat`, and support for a single application layer router (i.e., simple VPN) is sufficient. For example, if pod3-4.cs.purdue.edu were the application layer router where `zigzagrouter` runs, `zzone.dat` would contain
>
> ```
> 55000 39000 128.10.112.135
> 40000 0 0.0.0.0
> ```
>
> Thus when `zigzagrouter` starts it reads the forwarding information from `zzone.dat` and is ready to forward UDP packets arriving on the data plane. The simplified problem counts as 120 points toward lab6's total of 220 points. Hence solving the simplified problem plus 100 points of extra credit earned would yield 220 points. If you decide to solve the simplified problem, please indicate so on the cover page of lab6.pdf. In a group effort, all members must solve the same problem, full or simplified.

Getting Started

Design the routing graph and prepare the `zzoverlay.dat` file mentioned above. For example:

```
2

127.0.0.1 10000
10001 20001 127.0.0.1
10002 0 0.0.0.0

127.0.0.1 20000
20001 30001 127.0.0.1
20002 10002 127.0.0.1
```

Any reachable IP address is acceptable.

Build `zigzagrouter` and `zigzagconf` with the `make` command in the `/v1` directory.

```
make
```

Start all `zigzagrouter`s first to get ready for forwarding data.

```
./zigzagrouter <control-port>
```

After the `zigzagrouter` is running, build (with `make`) and start the `mypingsrv` in [`lab2/v1/`](lab2/v1/).

```
../../lab2/v1/mypingsrv <server-ip> <server-port>
```

Note that you should set the IP/port of `mypingsrv` according to your `zzoverlay.dat` design.

Now, run `zigzagconf` to update the graph design to all routers.

```
./zigzagconf
```

After the update, start pinging with the following command.

```
../../lab2/v1/mypingcli <client-ip> <server-ip> <server-port>
```

To stop a running router or server, send `SIGINT` with `ctrl` + `c` on Linux.

See `README.md` in `lab2/v1` for details about the configuration of pinging.

Project Structure

`zigzagconf.c`

Provide the functionality to update routing plan to all associated `zigzagrouter`s.

`zigzagrouter.c`

Provide the functionality to do application-layer routing. Be able to manage receiving ports update from `zigzagconf`.

`print_payload.c`

Print payload contents in hex code.

`read_overlay.c`

Read `zzoverlay.dat` file and provide a clean interface to get the configuration.

`socket_utils.c`

Wrap the low-level socket system calls with safer functions.

`zzconfig_codec.c`

Encode and decode the message packets between `zigzagconf` and `zigzagrouter`s.

Performance Analysis

Sample Configuration

- `./mypingsrv 128.10.25.207 10901` runs on `128.10.25.207`.
- `./mypingcli 128.10.112.137 128.10.112.132 10501` runs on `128.10.112.137`.

`zzoverlay.dat`

```
4

128.10.112.132 10500
10501 10601 128.10.25.214
10502 0 0.0.0.0
```

```
128.10.25.214 10600
10601 10701 128.10.112.135
10602 10702 128.10.112.135

128.10.112.135 10700
10701 10801 128.10.25.201
10702 10502 128.10.112.132

128.10.25.201 10800
10801 10901 128.10.25.207
10802 10602 128.10.25.214
```

`pingparam.dat`

```
3 1 0 1
```

Statistics

The following data shows the RTT in 9 independent trials.

0 `zigzagrouter` RTT (ms)	1 `zigzagrouter` RTT (ms)	2 `zigzagrouters` RTT (ms)	0 `zigzagrouters` RTT (ms)	4 `zigzagrouters` RTT (ms)
0.879	1.640	2.272	2.853	3.523
0.748	1.582	2.476	3.329	4.017
0.572	1.380	2.210	3.133	3.919
0.800	1.675	2.426	3.235	4.019
1.047	1.797	2.364	2.950	3.532
0.932	1.723	2.377	3.059	3.755
0.918	1.770	2.536	3.333	4.028
1.045	1.894	2.565	3.245	4.110
1.042	1.758	2.518	3.222	4.052

As expected, the pinging RTT increases when the number of involving `zigzagrouter` increases as it needs more UDP transmission to pass the data. The "0 `zigzagrouter`" scenario is a typical P2P pinging only involving physical routing. Other cases are more complex and slower as each application routing are built on top the physical one.