GSoC'24 Proposal : Lablua

By Mohammad Shehar Yaar Tausif

Basic Information

Preferred Email address : sheharyaar48@gmail.com
Github : https://github.com/sheharyaar
Matrix ID : @sheharyaar:matrix.org
Academic Background : I am a pre-final year undergraduate student at Indian Institute of Technology Kharagpur, India.
Other time commitments : I have no other time commitments till July end.

Experience

Programming Languages : C, C++, Go, Lua, Python, Javascript, Bash Scripting

Tools for Development : Vim, VSCode, Git,

Debugging and Tracing tools : GDB, Valgrind, strace, ltrace, kprobes, uprobes and eBPF

Familiarity with Lua : I have experience in Lua and LuaJIT. At my previous internship at API7.ai, I worked on fixing bugs, writing FFIs and adding features to APISIX API Gateway that is written in LuaJIT.

Software Development Experiences

Yes, I have developed software in a team environment before.

Internships

1. Software Engineer Intern at Neverinstall (1 year 8 months)
    - Worked as a core dev team member to scale the startup from ~150 DAUs to over 2500+ DAUs.
    - Enhanced the observability using eBPF scripts and modularity of the desktop streaming service by separating the control and data plane over Unix sockets.
    - Patched system programs to work out of the box in Linux containers without systemd support.
    - Improved service reliability by creating thread-safe Foreign Function Interfaces (FFI) for Xorg and Gstreamer C libraries in Go to support concurrent goroutine access.
    - Integrated AI and LLM to the OS using IPC over DBus between side-car containers.
    - Developed APIs to expose clipboard contents, X11 windows, filesystem and shell information for AI context using X11 APIs.
2. Software Engineer Intern at Securethings.ai (5 months)

- Designed a framework to process large data (~1Gbps) from an OEM device over Unix sockets.
- Developed an event-driven notification system to fan-in alerts and execute user-defined hooks.
- Enhanced security of embedded devices by integrating custom HSM with Azure provisioning.

3. Backend Development Intern at API7.ai (3 months)
    - Develop API7 Enterprise products and contribute to Apache APISIX repo with over 13,000 stargazers.
    - Add support for GRPC-web transcoding and HTTP proxy for Apache Dubbo to APISIX gateway.
    - Improve the control plane and enhance load-balancing algorithms for enterprise customers.
    - Profile, benchmark and generate CPU flame-graphs for production running servers.

Competitions and Hackathons

1. Inter IIT Tech Meet 2022 (1st position scored by the team)
    - Built a scraping system and handled large volumes of data over AWS S3 using AWS SDK.

2. Intra IIT OpenSoft 2021 (1st position scored by the team)
    - Built an end-to-end complete hybrid cloud web application and set up cloud infrastructure using Kubernetes and Helm Charts.
    - Implemented optimal load distribution and reverse proxy by setting up NGINX and ensured efficient system monitoring and visualization using Prometheus and Grafana.
    - Developed a scalable backend API using Swagger, and built the server in Go with Postgres as the DB.

Open Source Involvements

- metakgp/iit-kgp-network: I donated this project. Original: sheharyaar/iit-kgp-network
- add cache metrics for NGINX plus : nginxinc/nginx-prometheus-exporter #540
- kubearmor/kubearmor-client : #388 #262
- kubearmor/KubeArmor : #899
- add fetchSysPath : bendahl/uinput #29
- Remove support for DSA Keys : libssh/libssh-mirror #231
- cache original_nodes with nodes : apache/apisix #10722
- add plugins/reload to control api : apache/apisix #10905
- grpc-web trailers : apache/apisix #10851
- update dubbo-proxy doc : apache/apisix #10822
- add BIT support to BITPOS command : apache/kvrocks #2170

Full Resume :
https://drive.google.com/file/d/1RG_AIxfx1Ui-oat2TscFX09STdYXsg5T/view?usp=sharing

GSOC

- Yes I have participated as a student in GSoC before in 2022. Unfortunately, I was **not** selected.
- I have also applied to Apache KVRocks for my second GSoC proposal.

Project

I am applying for the project “**Lunatik binding for Netfilter**”

Background

Netfilter is a tool inside the Linux operating system that helps control the flow of data packets moving through the computer's network. It uses hooks within the network's journey to check and manage these packets. Programs can be attached to these points to inspect and decide what to do with the data as it passes by. These programs can be added or removed as needed to change how the data is handled.

NFLua is an add-on for Netfilter that lets people use Lua to manage these data packets. The problem with NFLua was that it executed functions on behalf of the user instead of using callbacks. The goal of this project is to make a new tool that connects Netfilter with Lua, allowing people to create new Netfilter programs using Lua and attaching **callbacks** to the Netfilter hooks.

Motivation

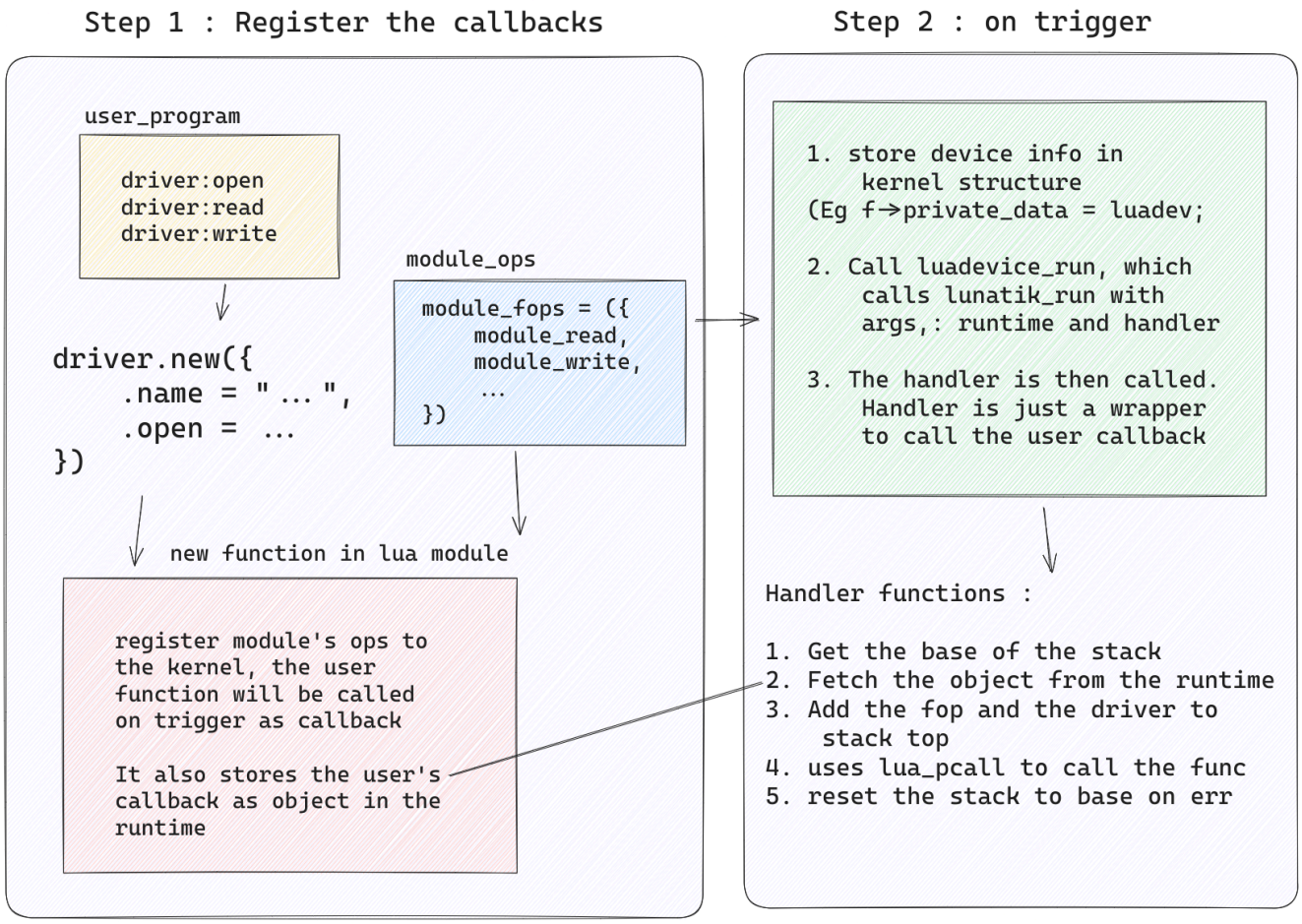
I am motivated to work on this project because :

- It aligns with my interests in working on projects at low-level, closer to the kernel.
- This also presents a unique opportunity for me to apply my programming skills in C and Lua, my knowledge of Operating Systems and my debugging qualities to this project.
- Initial design and working on this project forced me to research and understand the foundations of netfilter and iptables projects on which the documentation is very limited. This project will help me push my boundaries and help me gain knowledge in this area of the kernel.
- This project also involves a lot of thinking around namespace, per-cpu isolation, netfilter hook context, locks in the kernel, etc., thus making it a good project to gain valuable experience.

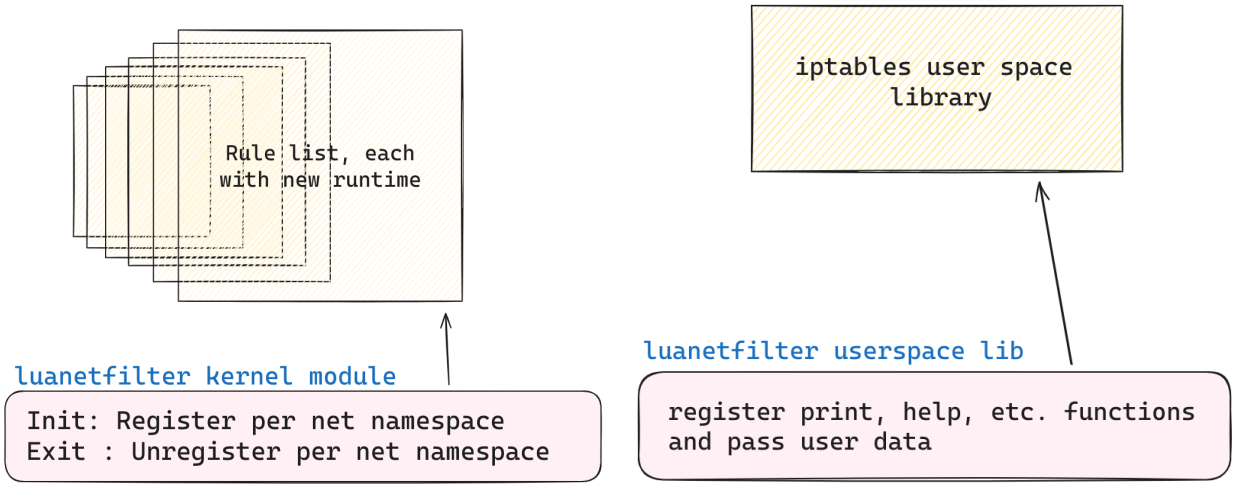
Design of the Implementation

The design components of the implementation of luanetfiler, luapacket and other deliverables using the new lunatik architecture are :

1. Lunatik Architecture

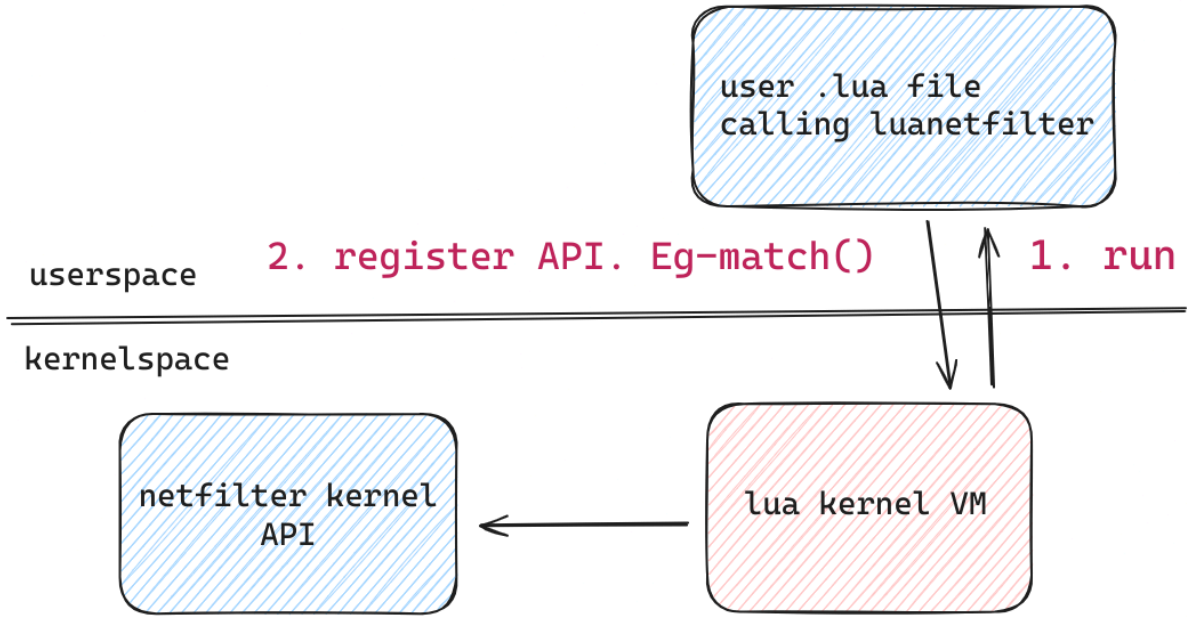


2. Kernel Module and User Space Module for `match` and `target`

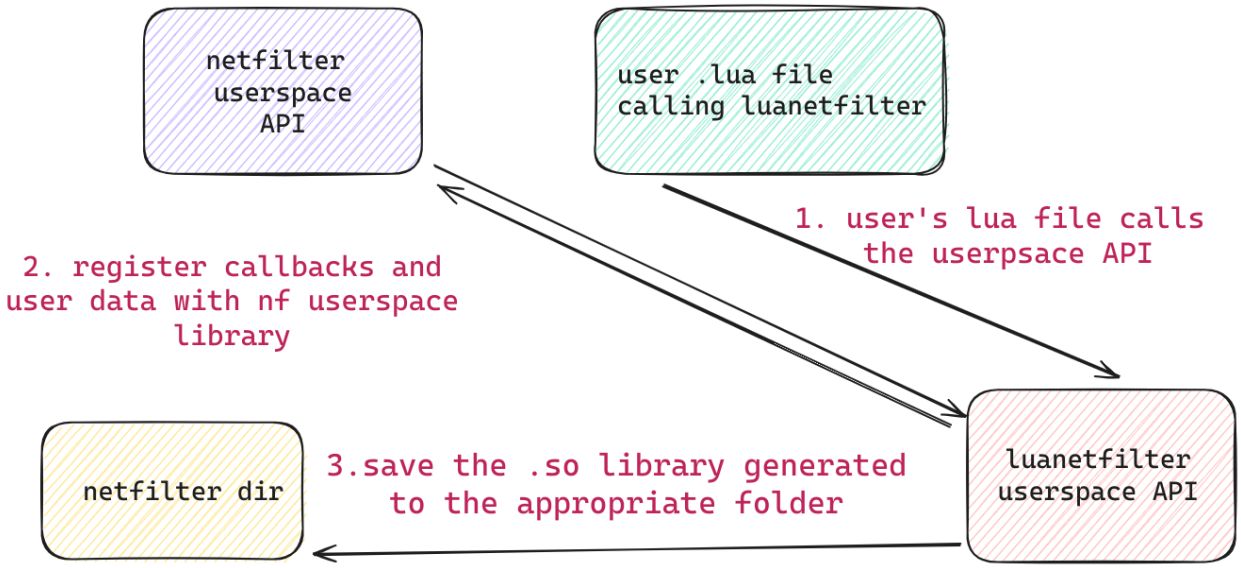


3. Interaction Between Netfilter Userspace and Kernel Space module for `match` and `target`

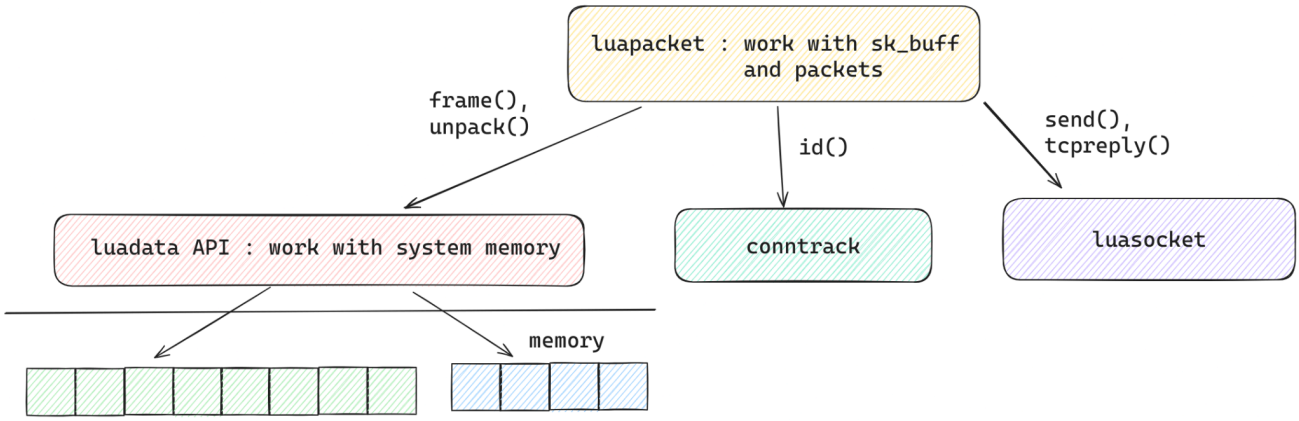
Step 1 : Registering with kernel



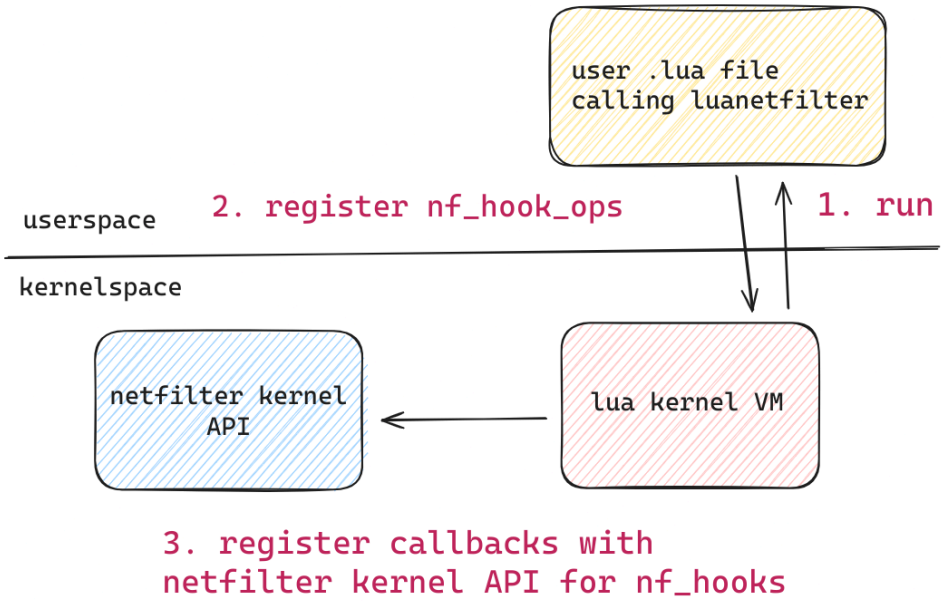
Step 2 : Registering with userspace



4. `Luapacket` over luadata and luasockets



5. `netfilter hooks` implementation



Deliverables

1. Implementing `luapacket` over luadata and luasockets. This is the proposed APIs to be implemented as of now, the detailed discussion and the draft for a proper API will be done during the Community Bonding Period
    - `close()` – Discards all resources of the packet.
    - `connid()` – Returns an integer that represents the ID of the packet's connection as maintained by conntrack module of Netfilter. This will be done after implementing conntrack
    - `frame()` – Returns a memory object which references the frame part of the packet (L1 header)

- `send([payload])` – Send the packet through the network and then closes it, thus it cannot be sent again

- `tcpreply(type, message)` – Sends a TCP reply with the string message as payload.

- `unpack(fmt [, pos])` – Returns the values encoded in position pos of packet's payload (see diagram below), according to the format fmt (see the Lua manual).

2. Implementing `user-space library` for xtables
   - Implement callback registering for xtables userspace
   - Handle data sharing between user space module and kernel module.
   - Handle generating of `.so` files and placing them in `XTABLES_LIBDIR`.

3. Additional routines that need to be implemented to luapacket for working with packets :
   - `skb_clone()` – copy an skb, but the data remains shared (“hardlinked”).
   - `skb_copy()` – copy an skb and its data.
   - `skb_copy_expand()` – copy an skb and its data, and additionally expand its size.
   - `skb_header_pointer()` – returns a pointer to the start of the layer-3 header.
   - `skb_linearize()` – make an skb linear.

   - `skb_make_writable()` – make the skb writable for the given length; required for targets.

   - `skb_pull(struct sk_buff *, unsigned int length)` – pull `skb->data` towards the right — **increments** `skb->data` **by length** and **decreases** `skb->len` **by the same amount.**

   - `skb_pull_tail(struct sk_buff *, unsigned int length)` – pull `skb->tail` towards the right.

   - `skb_push(struct sk_buff *, unsigned int length)` – push skb->data towards the left — **decrements** `skb->data` pointer by length and **increases** `skb->len` by the same amount.

   - `ip(v6)_hdr` – returns a pointer to the IP(v4/v6) header.
   - `ip_hdrlen(struct sk_buff *)` – size of the IPv4 header in this skb.

4. Implementing `match` and `target`
   - `register_pernet_subsys()`
   - `xt_(un)register_match(es)()`
   - `xt_(un)register_target(s)()`
5. Implementing `nf_hook_ops`
   - `nf_hook_ops`

- Netfilter verdict helpers (`NF_ACCEPT`, `NF_DROP`, `NF_STOLEN`, `NF_REPEAT`, `NF_STOP`)
- Implement `nf_(un)register_hook(s)`

6. Implementing `conntrack`
   - Implementing `ip_conntrack_info` and its identifiers.
     - IP_CT_ESTABLISHED
     - IP_CT_RELATED
     - IP_CT_EW
     - IP_CT_IS_REPLY
     - IP_CT_ESTABLISHED_REPLY
     - IP_CT_RELATED_REPLY
     - IP_CT_NEW_REPLY
     - IP_CT_NUMBER
   - Implement `nf_ct_get` to get the info for a particular sk_buff / luapacket.
   - Implement `nf_conn_acct_find` to get the counter informations.
   - Implementing `nf_conntrack_helper_register` to register helpers for netfilter conntrack.

7. Implementing `netlink` message support over luasocket
   - Support for `nlmsghdr` & `genlmshdr` structures, netlink control message types and netlink families (not present in the current luasocket implementation).
   - Support for netlink flag options for `nlmsg_flags`, `GET` requests & `NEW` requests and support for netlink socket options.

8. Adding example scripts like the following :

```
local netfilter = require("netfilter")
local luapacket = require("luapacket")

local function match_func(pkt) -- pkt is an instance of luapacket
    if pkt == nil or type(pkt) ~= "table" then
       return nf.ACCEPT
    end

    local len = pkt:length()
    if len < 14 then
        return nf.DROP;
    end
    return nf.ACCEPT;
end

local ops = {name = "mymodule"}
local nf = netfilter.new(ops)
local match = nf:match(match_func)
```

9. Adding benchmarks and test scripts for per network subsystem registration and evaluation.

Pre-GSoC Work

- I have understood the new architecture of the lunatik library and I have explored the previous year's GSoC Pull Requests for outlining the deliverables and as a starting point to build a PoC.

- I have also attempted to port the `match` function from xtables to lunatik. This is just a PoC. This re-uses a lot of nflua code which will also be re-implemented using the new lunatik libraries or will be ported to lunatik during the GSoC period.

  Commit: https://github.com/sheharyaar/lunatik/commit/17cedee244dd7bab9f7d1274181c8d84637ff3e7

- I also gained the fundamental knowledge of Netfilter libraries and Netlink Protocol that is required for the completion of these tasks.

- I have carried out discussions with the mentor(s) of this project and built the proposal in accordance with their suggestions.

May 1 - May 26 [Community Bonding Period]

- The primary focus in this period will be to discuss and draft a proper API for the various deliverables that will be implemented and the schedule will be refined according to the final draft.
- I will study `PacketScript` implementation in order to improve the approach and the required APIs for the project.
- This period will also be used to get familiar with the lunatik VM in the kernel, the process of loading and pushing - variables, handlers and errors to the lua stack. This will also require me to get familiar with common functions used in lunatik from the lua C API (Example : luaL_check* functions), the libraries present in lunatik and their counterparts in nflua.
- I will set up development tools such as benchmarking scripts, QEMU VMs and kernel images for testing my changes and to improve the workflow. I will also contribute these to the lunatik repository for the community.

The order that I would be following to implement the features will be :

- Implementing `luapacket` over luadata and luasocket
- Additional routines that need to be implemented to luapacket for working with packets :
- Implementing `match` and `target`
- Implementing `nf_hook_ops`
- Implementing `conntrack`

May 27 [Coding Phase 1]

May 27, 2024 - Jun 23, 2024 (4 weeks)

In this period, I will work on implementing the `luapacket` APIs along with `match` feature for lunatik using luadata and luasocket. This includes all the helper functions and additional **IP and skbuff routines** required to be implemented for use in other netfilter libraries. The unit tests related to luapacket and these helper routines will also be completed.

The implementation of luapacket and match will be done in parallel to have a working concept which will guide us for the other features and implementations.

Jun 24, 2024 - Jul 7, 2024 (2 weeks)

During this period, I will make changes to the `luapacket APIs` and `luadata` to implement `target` features along with the required documentation, unit tests and benchmarks.

After the new features are implemented, I will write sample programs which users can refer to as a starting point. These sample programs can be :

- A simple packet dropping program that drops all DNS packets using `match.`
- A program that echoes back all the bytes written to a particular port using `target.`
- A program combining match and target to create complex rules that are updated over the cloud (example from netdev 0x14)

The test cases will include testing for per network subsystem registration and evaluation.

July 8 - July 12 [Midterm Evaluations]

This period will be used to write a detailed report on the work done in Coding Phase 1. All the work done will be uploaded and documentation will be created/uploaded to Lunatik repository. The benchmarks will also be uploaded to the repository.

July 12 - August 19 [Coding Phase 2]

Jul 13, 2024 - Aug 3, 2024 (3 weeks)

During this period, I will implement the netfilter core hook (`nf_hook_ops`), `conntrack` and `conntrack helpers` for the lunatik library. I will write unit tests, documentation and benchmarks for this feature.

I will write sample programs for users to refer to as a starting point, these programs can be :

- A program that drops all packets from a particular IP address using netfilter core hooks.
- Counting packets of a particular connection and exporting to a time-series database or cloud service.
- Implementing ftp pr custom conntrack helper in Lua.

Aug 4, 2024 - Aug 18, 2024 (2 weeks)

During these 2 weeks, I will implement `netlink` features to the luasocket library and I will work with the mentors to review the performance of the library via benchmarks. I will also work to identify hot regions using CPU flame graphs, memory usage using kernel tracers and work towards possible optimisations to make the library more efficient. This will be important for sections dealing with luapacket and luadata that handle kernel memory and perform operations on the memory.

August 19 - August 26 [Final Evaluations]

All the progress until now, will be compiled and a detailed report on the work done in Phase 2 will be done this week. I will also write a detailed article (or blog post) on the overall work done during this period which will be published on my github repository and medium. **All the promised deliverables will be completed by this period**. The documentation will be uploaded and benchmarks will be shared on the repository.

Post GSoC

After GSoC, I will submit a package for the implemented features to the OpenWRT repository. I will be moving on to other extensions that can be added to the lunatik library. I would also be very happy to become a maintainer and encourage new contributors to contribute to the repository.