Google Summer of Code 2017 Proposal

LabLua

Name: Nikhil. R
Email: rnikhil96@outlook.com , rnikhil2751996@gmail.com

BASICS

1)What is your preferred e-mail address?

E-mail 1: rnikhil96@outlook.com
E-mail 2: rnikhil2751996@gmail.com

2)What is your web page / blog / GitHub?

Blog: https://rnikhil275.github.io/
Resume: https://rnikhil275.github.io/projects/
GitHub: https://github.com/rnikhil275/
LinkedIn:https://in.linkedin.com/in/rnikhil275

3)What is your academic background?

- M.Sc Economics and B.E Manufacturing at Birla Institute of Technology and Science Pilani, Pilani Campus graduating May 2018

- Higher Secondary School: Narayana Junior College (M.P & E.V English Medium School), Visakapattinam
- Secondary School: The Hindu Colony Chellammal Vidyalaya Senior Secondary School, Chennai

4)What other time commitments, such as school work, another job (GSoC is a full-time activity!), or planned vacations will you have during the period of GSoC?

I'm completely free from May 5th to August 30th. I have no other commitments/jobs. I may take one or two day break which will be informed ahead in time. I will work every day and hence can easily target about 50 hours a week.

Experience

1)What programming languages are you fluent in? Which tools do you normally use for development?

- Programming Languages: Proficient in C, Python, Lua and PHP. Familiar with x86 Assembly, Javascript and HTML/CSS.
- Web Frameworks: Sailor, Django, Flask, Slim, Laravel
- Databases: MySQL, MongoDB, Redis, Elasticsearch
- Operating Systems: Linux, Windows
- DevOps: AWS, Git, Ansible, Docker, Heroku
- Reversing Tools: OllyDBG, GDB, Bless(hex editor), Hopper Disassembler

I use Mac OS X Seirra(10.12.3) with a virtual machine running a vagrant box (Ubuntu 14.04) but I also boot into windows for domain specific work. I work in terminal most of the time and my preferred choice of text editor is

Visual Studio Code with minor edits done using vim. I use Git for version control.

2) Are you familiar with the Lua programming language? Have you developed any projects using Lua?

Yes, I am familiar with Lua programming language. I can use most of the features of the language like tables, metatables effectively.
I have done a few lua projects:
Sailor-admin center: https://github.com/rnikhil275/sailor-admin
Luagoogle: https://github.com/rnikhil275/luagoogle
Sailor: https://github.com/rnikhil275/sailor

I also participated with Lablua as part of GSoc 2016 working on Sailor which is a lua MVC framework. Apart from this, I have also written a few scripts for my campus DC++ client.

3) Have you developed software in a team environment before? Any projects with actual users?

Yes, I have lot of experience developing software in teams before.

- Google Summer of Code 2016 with LabLua - Sailor Web Framework

Integrated Elasticsearch with Sailor so that ES indexes can now be stored/searched as Sailor models providing interoperability between both systems. Developed a centralized configuration editor to change config settings on the fly without reloading the entire application. Also working on the centralized plugin system to integrate third party extensions into Sailor

Technologies Used: Lua, Elasticsearch REST API

- Vedanta Resources Ltd. (Summer 2015)

Analyzed and automated the data reporting and logging system at the Jharsuguda Plant saving around 70 hours of manual labor monthly. Personally appreciated by the Vice President(Operations) for the aggregation system and it was immediately put into production.

Technologies Used: Python, Python-excel packages like openpyxl, xlutils etc

- Technical Lead SSMS BITS Pilani (April 2015 - May 2016)

Led a 4 member team to develop a billing and inventory management system which is being put into use daily at all messes and canteens to handle transactions of about ₹6 Million monthly for all 4500 registered students.

Technologies Used : LAMP stack.

All these projects have actual users.

4)1) What kinds of projects/software have you worked on previously? (Anything larger than a class project: academic research, internships, freelance, hobby projects, etc.)

Projects I have worked on in no particular order.

- PintOS

Extended the OS by adding virtual memory, message passing queues, POSIX threads and OS-level semaphores. Implemented “fork” and “exec”

system calls and the ability to run user program. Implemented indexed filesystem, hierarchical directory structure and buffer cache.

Technologies Used: C

- DoctorsNow

Built the entire appointment booking and video calling system for doctors and patients. Patients can schedule appointments in advance, book slots for video calling and review their previous visits. The product has been now ported to Android and IoS and the startup launched as Visit.

Technologies Used: PHP, Slim Framework, WebRTC for the video calls

- Tracker

It is to help developers jump start developing any kind of mobility tracking applications. Tracker let's any type or number of GPS enabled endpoints to send data to a NodeJs server which then maps their location history using Google Maps API.

Technologies Used: Javascript, NodeJS, Google Maps API

- Portsmith

A secure Port Knocking Implementation in Python using single packet authorization. Uses TCP and IND-CCA secure requests to open a port on the server. Uses hping3 to craft TCP packets. The knock packet is encrypted using the key transferred from the server and then sent to the knockport. It gets logged into kern.log which is read by Portsmith. It is then decrypted and the required port is then opened for the sourceIP using a

custom iptables command. There is also an integrated SOCKS proxy to perform knocks before routing application traffic. Currently working on a kernel module implementation using Netfilter hooks, nftables and cryptographic primitives instead of high level libraries.

Technologies Used: Python, Cryptography Library(Fernet Symmetric Encryption)

- Peer to Peer Streaming Protocol

Was involved in the building a statistics module for an application layer streaming protocol in Python. It records information related to the stream, number of peers (audience) per minute, in/out of peers, total peers, etc. Wrote an graphical interface for viewing the numbers built using d3.js.

Technologies Used: Python, HTML/CSS, Javascript (D3.js)

- MicroTracer

System Call tracer for Linux based systems which displays the syscalls used by a particular program in a neatly formatted manner without the complex details as shown by programs like strace.

Technologies Used: C

Implemented a Carrier Sense Multiple access with collision avoidance (CSMS/CA) layer for a generic protocol stack which was presented at APOGEE 2015.

I have done a bunch of other personal projects also which can found on my GitHub profile (https://github.com/rnikhil275/)

2) In particular, are you (or have you been) involved with any open source development project? If so, briefly describe the project and the scope of your involvement.

Some of the projects above are open source. I have been working with Lablua since last summer and I intend to continue doing so.

Project

1)Did you select a project from our list? If yes, which project did you select? Why did you choose this project? If you are proposing a project, give a description of your proposal, including the expected results.

Yes. I have selected a project from the list.

I want to work on “Improve the HTTPS module of LuaSec”. I really liked this project because it lets me delve on two very important things. I have been working on a lot of computer networking related projects recently and this one gives me a chance to work on implementing HTTP protocol which helps me understand it’s internals better. Also, I get to work on a very important low level module which would be used by other applications and I also find myself using this in the future. This project also matches my skillset and I would really enjoy working on this.

Improve the HTTPS module of LuaSec

I am going to split the project into three subtasks:

1) Integrate more HTTP functions into LuaSec
2) Add support for HTTP/2
3) Work on features like proxy, SNI, curve selection API, advanced HTTP/2 features and documentation based on time left.

4) All code would be documented and tested at the end.

1) Integrate more HTTP functions into LuaSec

Right now LuaSec depends on Luasocket for it's HTTP functions. This task would involve getting those functions from Luasocket and integrating it directly into LuaSec. This would solve the integration problems.

This can split into subtasks:

1) First check this pull request to find a basis for starting the work on http.lua(https://github.com/brunoos/luasec/pull/38). Also, use this pull(https://github.com/diegonehab/luasocket/pull/133) as reference.
2) Based on the investigation above, start importing a few functions directly into luasec. Remove dependency on LuaSocket wherever possible.
3) Check if the redirect problem is solved and whether the SNI support is fixed.
4) Test support for proxy using HTTP CONNECT method.

2) Add support for HTTP/2

This RFC(https://httpwg.github.io/specs/rfc7540.html) shall be the reference for this section. I also use this (https://hpbn.co/http2/) for understanding the spec and enumerating the functions. This can be split into subtasks:

1) Write HTTP/2 connection module

    HTTP 2.0 is a binary protocol that multiplexes numerous streams going over a single (normally TLS-encrypted) TCP connection. The contents of each stream are HTTP 1.1 requests and responses, just encoded and packed up differently. HTTP/2 adds a

number of features to manage the streams, but leaves old semantics untouched. HTTP 1.1 is very latency sensitive, partly because HTTP Pipelining is still riddled with enough problems to remain switched off to a large percentage of users.

HTTP Pipelining is a way to send another request while waiting for the response to a previous request. It is very similar to queuing at a counter at the bank or in a supermarket. You just don't know if the person in front of you is a quick customer or that annoying one that will take forever before he/she is done: head of line blocking.

Since HTTP/2 is a binary protocol, to inspect the traffic, something like curl or wireshark HTTP/2 dissector have to be used. I have experience with both of them. HTTP/2 sends binary frames. There are different frame types that can be sent and they all have the same setup: Length, Type, Flags, Stream Identifier, and frame payload. There are ten different frame types defined in the HTTP/2 spec and perhaps the two most fundamental ones that map to HTTP 1.1 features are DATA and HEADERS.

I shall be using lua-http module which can found here (https://daurnimator.github.io/lua-http/#http.h2_connection) to get an idea of functions to implement.

2) Add support HTTP/2 multiplexing and stream module

When the HTTP/2 TCP connection is established, the client and server exchange SETTINGS frames first, which indicates how many streams can be open at a time(or how many parallel requests), how many bytes server is ready to receive for a stream and for the whole connection, and the rate at which data can be delivered or received i.e., window size. Once a TCP connection is established all the requests for that origin is done via that TCP connection. Multiple

HTTP/2 requests are divided into frames and assigned their respective stream ids. All the frames from multiple streams are sent asynchronously. And the server also sends responses asynchronously. Therefore if one response is taking too long then other's don't have to wait for it to finish. The client receives the frames and arranges them according to their stream id. The request and response both happen parallelly i.e., while client is sending frames the server is also sending frames back to the client. Using the PRIORITY frame, a client can also tell the server which other stream this stream depends on. It allows a client to build a priority "tree" where several "child streams" may depend on the completion of "parent streams".

The Stream Identifier associates each frame sent over HTTP/2 with a "stream". A stream is an independent, bi-directional sequence of frames exchanged between the client and server within an HTTP/2 connection. Streams can be established and used unilaterally or shared by either the client or server and they can be closed by either endpoint. The order in which frames are sent within a stream is significant. Recipients process frames in the order they are received.

To make an HTTP request, first the client divides the request into binary frames and assigns the stream ID of the request to the frames. Then it initiates a TCP connection with the server. And then the client starts sending the frames to the server. Once the server has the response ready it divides the response into frames and gives the response frames the same response stream id. Server sends the response in frames. Stream ID is necessary because multiple requests to an origin is made using a single TCP connection so stream ID makes it possible to identify to which request or response a frame belongs to.

I shall be using this as reference to get an idea of functions to implement. (https://daurnimator.github.io/lua-http/#http.h2_stream)

3) Improve the present headers module to add HTTP/2 headers

For every TCP connection the client and server maintain a headers table containing the last response and request headers and their values respectively. For the first request or response they send all the required header. But for subsequent requests client and server omit headers which are same as the previous request or response. Those missing headers are retrieved from the headers table thus decreasing the overall length of the headers data in a request or response.

I shall be using this as reference to get an idea of functions to implement. (https://daurnimator.github.io/lua-http/#http.headers)

4) Add support for server push

The idea is that if the client asks for resource X, the server may know that the client will probably want resource Z as well, and sends it to the client without being asked. It helps the client by putting Z into its cache so that it will be there when it wants it. It is something a client must explicitly allow the server to do. Even then, the client can swiftly terminate a pushed stream at any time with RST_STREAM should it not want a particular resource.

Refer to the push and push promise functions here(https://daurnimator.github.io/lua-http/#http.h2_stream)

5) Header Compression module

HPACK, Header Compression for HTTP/2, which is a compression format especially crafted for HTTP/2 headers, and it is being specified in a separate internet draft. The new format, together with other counter-measures (such as a bit that asks intermediaries to not compress a specific header and optional padding of frames), should make it harder to exploit compression.The receiver always controls the amount of memory used in HPACK, and can set it to zero at a minimum, with a maximum related to the maximum representable integer in a SETTINGS frame, currently 2^32 - 1.

This RFC(https://httpwg.github.io/specs/rfc7541.html) and this (https://daurnimator.github.io/lua-http/#http.hpack) shall be used as reference while implementing the HPACK module.

6) Add support for negotiation schemes. (ALPN first, then others)

Next Protocol Negotiation (NPN) is the protocol used to negotiate SPDY with TLS servers. As it wasn't a proper standard, it was taken through the IETF and the result was ALPN: Application Layer Protocol Negotiation. ALPN is being promoted for use by HTTP/2, while SPDY clients and servers still use NPN.ALPN differs from NPN primarily in who decides what protocol to speak. With ALPN, the client gives the server a list of protocols in its order of preference and the server picks the one it wants, while with NPN the client makes the final choice.

3) Work on features like proxy, SNI, curve selection API and documentation.

Use HTTP CONNECT method to build a HTTP tunnel for accessing HTTPS websites. This RFC(https://www.ietf.org/rfc/rfc2817.txt) and this RFC (http://tools.ietf.org/html/rfc2616) details it.

CONNECT is used between the client and the proxy server before establishing the TLS connection between the client and the end server. The client connects to the proxy and sends a request to connect to HTTPS website. The proxy opens a TCP connection to the HTTPS website and responds to the client with a 200 status code, accepting the request. After this, the connection between the client and the proxy is kept open. The proxy server relays everything on the client connection to and from the website. The client upgrades its active connection to the HTTPS webiste to an SSL/TLS connection, by initiating a TLS handshake on that channel. Since everything is now relayed to the server, it's as if the TLS exchange was done directly with the HTTPS website. The proxy doesn't play any role in the handshake (and thus with SNI). The TLS handshake effectively happens directly between the client and the end server.

Here all I need to do for allowing clients to connect to HTTPS servers is read in the CONNECT request, make a connection from the proxy to the end server (given in the CONNECT request), send the client with a 200 OK reply and then forward everything that you read from the client to the server, and vice versa.

Work on the support for SNI and curve selection API, investigate existing implementations and pull requests. Documentation and various reports shall also be dealt with here.

2)Please provide a schedule with dates and important milestones/deliverables (preferably in two week increments).

The following below is a tentative timeline for developing admin center, the elasticsearch plugin and luagoogle.

May 5 - May 19	Discuss with mentor on possible integration

	issues and figure out which HTTP functions are not working properly. Start working on the upgrade to HTTP/2 roadmap. I would like to have a clear design plan before starting to code. Take best practices from already existing implementations. Design a plan for HTTP/2 implementation after looking at various other sources(lua-http, RFC's). If possible, also discuss the implementation of the proxy, SNI and other features.
May 20 – May 31	HTTP functions would be integrated inside Luasec itself. I would be working on the first task during this period modifying https.lua. Initial current support for SNI, proxy and redirects would also be tested.
June 1 – June 14	Finish integration and since the basic HTTP functions would be ported, this would be good time to set up a test environment. I have Ubuntu 14.04 and MacOS Sierra 10.12.3 and I shall be using Apache for testing. This is also the period for the first evaluation and the deliverable would be an integrated HTTP/HTTPS module in LuaSec. The first report shall be submitted to Google regarding the same. Will also be discussing with mentor and tweaking the codebase.
June 15 – July 7	Start work on HTTP/2 implementation. Work on a

	few basic binary types and methods. Code the connection module, stream module and multiplexing. I shall be working on this first because I would like to deliver a basic HTTP/2 functions so that applications can use it.
July 7 – July 30	Work on upgrading the headers to HTTP/2 standards, implement Server Push and this is the time for second evaluation, hence the second report shall be worked on and submitted to Google. Carefully work on header compression; HPACK module. Implement basic features first.
August 1 – August 14	Support ALPN first as it's important for both TLS and HTTP/2. Support for rest would depend on time left. Investigate existing code on relevant matter for reference. This PR(https://github.com/brunoos/luasec/pull/64) would be referred to and worked upon.
August 15 – August 21	Start work on documentation for the entire project, API reference and usage tutorials.
August 22 - 29	The HTTP functions would be integrated well with HTTPS module of LuaSec by now. HTTP/2 support would be added. Finish documentation, code review, bug fixing if any and polishing of code(as suggested by the mentor) would be done. Features like Proxy, SNI support, HTTP/2 optimization like Domain Sharding, Stream Prioritization, advanced features in HPACK, etc would be worked upon depending on the time

	remaining.

3)What will be showable one month into the project?

After the first month, HTTP functions would be integrated inside the LuaSec module itself adding support for redirects, proxy and server name indication if time remains. Work on HTTP/2 would have started.

4)What will be showable two months into the project?

After the second month, work on HTTP/2 would be half done and LuaSec would support stream multiplexing, updated HTTP/2 headers and Server Push.

Depending on time, advanced features would be integrated into HTTP/2. Basic ALPN support would be worked upon and compulsory. Use lua-http(https://github.com/daurnimator/lua-http) module as reference and also have a look at this pull request for ALPN support. (https://github.com/brunoos/luasec/pull/64). Support for error module to encapsulate HTTP/2 error information, other negotiation schemes would depend on the time remaining.

GSOC

1)Have you participated to GSoC before? If so, how many times, which year, which project?

Yes. I have participated in GSoC once before in 2016 with Lablua.

2)Have you applied but were not selected? When?

Yes. I applied in 2015 to develop a simple Nodejs application which takes geolocation data through an endpoint and then plots it on a map using the Google Maps API. The project wasn't selected but the application was finished by me in the first month itself.

3)Did you apply this year to any other organizations?

No, I did not apply to any other organization. I really liked the community and decided to stick with them.