Running HIP-VPLS in infrastructure mode

Dmitriy Kuptsov
2024

Table of contents

CHAPTER 1

Introduction

CHAPTER 2

Background

In this section, we are going to describe some background material. We are going to start with the cryptographic primitives, such as symmetric key encryption/decryption algorithms, and then move on to the discussion of the Virtual Private LAN Services and what kind of problems they solve. We will then conclude the discussion in this chapter with the basic information on Host Identity Protocol as it is in the core of the solution which we are discussing in this document. To make the background material more or less complete, we are going to touch alternative on Transport Layer Security (we will here mention why this protocol is not used in the core of our architecture and is only used for the control-plane communications between the HIP-Switches and the HIP-controller). With these final words, we are going to conclude the current chapter of this work.

2.1 Cryptography

Cryptography forms the basis for secure telecommunications nowadays. SSL, TLS, SSH, Tacacs+, IPSec, DKIM, and DNSSec are only a few well-known telecommunication protocols that use cryptography to prevent such well-known attacks as eavesdropping, tampering, denial of message origin, etc. Modern cryptography is based on hard-core mathematics and non-trivial algorithms (such as random number generation, discrete logarithm problems, rings, fields, Euclidean algorithms, factorization of big numbers, etc.)

2.1.1 Symmetric cryptography

Symmetric key cryptography is just perfect for data-plane traffic as it offers high processing times (when compared to asymmetric key cryptography). As the name implies, symmetric key cryptography uses the same secret key to encrypt and decrypt messages. On one hand it is the main reason why these algorithms are so fast. On the other hand, and this is the main limitation of the type of cryptography: symmetric keys are hard to distribute and revoke without using more sophisticated symmetric key schemes.

As of today, several symmetric key cryptography algorithms, such as Advanced Encryption Standard (AES), Triple DES (3DES), and Twofish, offer advantageous processing speed and sufficient security levels. In our prototype implementation of HIP-VPLS we are using AES with 256 bits keys to perform encryption and decryption of data-plane traffic. Moreover, since NanoPI R2S - hardware that we employ to run our Software Defined Network (SDN) code - has support for on-chip instructions to boost the encryption and decryption of arbitrary long message blocks. In other words, we perform AES operations directly in the tiny computer's Central Processing Unit (CPU). We are going to devote a separate section on the implementation of the hardware accelerated AES encryption and decryption routines by the CPU.

2.1.2 Asymmetric cryptography

Asymmetric key cryptography, as the name suggests, uses two separate keys to encrypt and decrypt the messages. Since the encryption uses big number exponentiations (such as RSA) and multiplications (such as EDDSA), as well as modular arithmetics, the performance of these algorithms is considerably worse compared to symmetric cryptography algorithms.

However, since one is allowed to expose the public part of the key to anyone, and since this key is only required to encrypt the message and only the person who holds the private part of the key (secret part of the key) can decrypt the message, efficient key distribution and revocation can be organized, at the cost of extra CPU cycles. Moreover, digital signature schemes can be implemented with no hassle by encrypting the message with the private key and then making decryption plausible only with a public key (exposed to everyone). Many distributed versions of signature/encryption also exist in the literature broadening the application landscape of this type of cryptography. In our work, we use RSA and ECDSA in HIP protocol to generate message signatures. The HIP protocol also uses public keys to derive permanent identifiers for the HIP switches.

2.1.3 Hash functions

Hash functions are one-way mathematical functions used to generate the so-called fingerprints of a message (MACs). In other words, given arbitrary long input messages, a fixed size universally unique message (typically 128, 160, and 256 bits) is produced. Ideally, it should be computationally impossible to reverse the function to find the original message (or image) given the hash (or fingerprint). Hash functions are used in digital signatures to compress the message before signing it with public key cryptography algorithms. Modern hash functions should guarantee that no collisions are possible for the hash function (in other words, it should be hard to find two distinct images that will both produce the same hash code).

Keyed versions of hash functions are also widely spread. For example, Hash MAC (HMAC) is used in a symmetric setting when both parties share the key. This type of algorithm is used for the authentication and identification process. HMACs are typically used to protect the data-plane traffic from the forgery attacks.

2.1.4 Key exchange protocols

Key exchange algorithms are crucial to many security solutions. Key exchange algorithms allow parties to negotiate secret keys over insecure communication channels. Diffie-Hellman and the improved variant Elliptic Curve DH are well-known key exchange algorithm variants. We have implemented both variants of the algorithms. Note that, in general, key exchange algorithms need to be protected by signature algorithms to exclude Man-In-The-Middle attacks.

2.2 Virtual Private LAN Service

Virtual Private LAN Services (or VPLS) are pretty standard nowadays. Companies build VPLS to provide Layer-2 services for branch offices: VPLS are typically built as overlays on top of Layer-3 (IP). For example, when a frame arrives at VPLS provider equipment (PE), it is encapsulated into an IP packet and is sent out to all other VPLS network elements comprising emulated LAN. Security of such overlays is important for obvious reasons: customers do not want their corporate traffic to be sniffed and analyzed. In this work, we build such a secure overlay with Host Identity Protocol. We also introduce a specially crafted control-plane protocol to configure the nodes centrally.

2.3 Host Identity Protocol

Internet was designed initially so that the Internet Protocol (IP) address has a dual role: it is the locator, so that the routers can find the recipient of a message, and it is an identifier so that the upper layer protocols (such as TCP and UDP) can make bindings (for example, transport layer sockets use IP addresses and ports to make connections). This becomes a problem when a networked device roams from one network to another, and so the IP address changes, leading to failures in upper-layer connections. The other problem is the establishment of an authenticated channel between the communicating parties. In practice, when making connections, the long-term identities of the parties are not verified. Of course, solutions such as SSL can readily solve the problem at hand. However, SSL is suitable only for TCP connections, and most of the time, practical use cases include only secure web surfing and the establishment of VPN tunnels. Host Identity Protocol, on the other hand, is more flexible: it allows peers to create authenticated secure channels on the network layer, so all upper-layer protocols can benefit from such channels.

HIP relies on the 4-way handshake to establish an authenticated session. During the handshake, the peers authenticate each other using long-term public keys and derive session keys using Diffie-Hellman or Elliptic Curve (EC) Diffie-Hellman algorithms. To combat the denial-of-service attacks, HIP also introduces computational puzzles.

HIP uses a truncated hash of the public key as an identifier in the form of an IPv6 address and exposes this identifier to the upper layer protocols so that applications can make regular connections (for example, applications can open regular TCP or UDP socket connections). At the same time, HIP uses regular IP addresses (both IPv4 and IPv6 are supported) for routing purposes. Thus, when the attachment of a host changes (and so does the IP address used for routing purposes), the identifier, which is exposed to the applications, stays the same. HIP uses a particular signaling routine to notify the corresponding peer about the locator change. More information about HIP can be found in RFC 7401.

CHAPTER 3

Architecture

Overall, system architecture is shown in Figure 3.1. Apart from the HIP-VPLS switches we have also implemented special control-plane protocol on top of the SSL protocol. According to the protocol, on one hand, every HIP-VPLS switch reports to the central controller (and authenticated using HMAC algorithm together with the shared symmetric master secret). On the other hand, every HIP-VPLS switches obtains the configuration from the central controller (such as mesh configuration, HIT resolver information, firewall rules and MAC-based ACL). In the future, we also plan to support traffic shaper functionality.

Although we did not implement traffic shaper feature in our HIP-switches and HIP controller, we believe that it is still valuable for the future work. For example, different hosts can be served differently (have more bandwidth) than some other hosts by using traffic shaping. For example, if some hosts in HIP-VPLS network send delay sensitive traffic, for example, curtain rules can be configured on HIP controller to give needed advantage over other hosts in the network. We leave this for the future discussions and work.

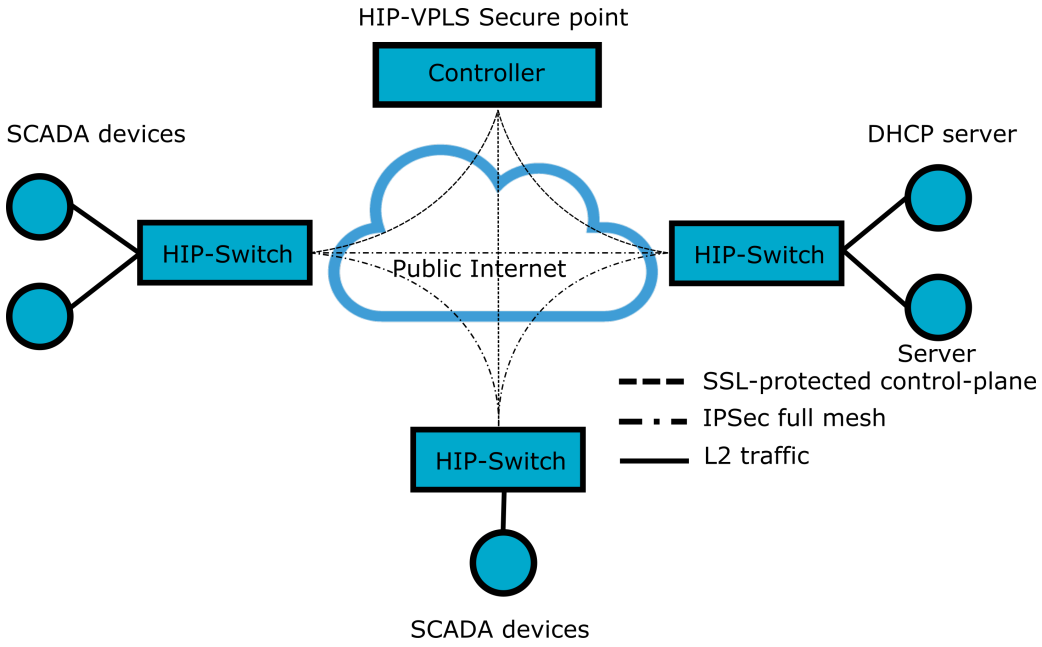

Figure 3.1: System architecture

CHAPTER 4

Proof-of-a-concept implementation

In this chapter, we are going to discuss the proof of a concept, or prototype, implementation of the HIP-VPLS (we are going to discuss how HIP-switches and controller were implemented). In our work we have used the Python language as it offers simplicity at the cost of extra CPU cycles to do the job.

The experimental test-bed is shown in Figure 4.1.

Our prototype implementation consists of roughly 10K lines of code. Overall, the deployed architecture is shown in Figure 3.1. The communication between the HIP-VPLS switches is secured with HIP and IPSec protocols. The communication with the HIP-VPLS controller is secured with SSL protocol. We have chosen HIP protocol to secure the data plane traffic as it does not rely on the TCP, hence reduces the wasted bandwidth.

The communication with the HIP-VPLS controller is authenticated using self-signed certificates. In addition client authenticates itself to the controller using HMAC and preshared master secret. The format of the control packets can be looked up in the HIP controller source code found in our Git repository [?].

To deploy the system, we have prepared the bash script. Overall the deployment is trivial. The only things that are needed to change is the master secret and MySQL password. Otherwise, the administrator needs to execute the following commands in the server's console (note, we have tested on Ubuntu 22.04.2 LTS). So, to deploy the system run the following (note, that one needs to change the MySQL password in deploy.sh and config.py files (for both controller and configurator)):

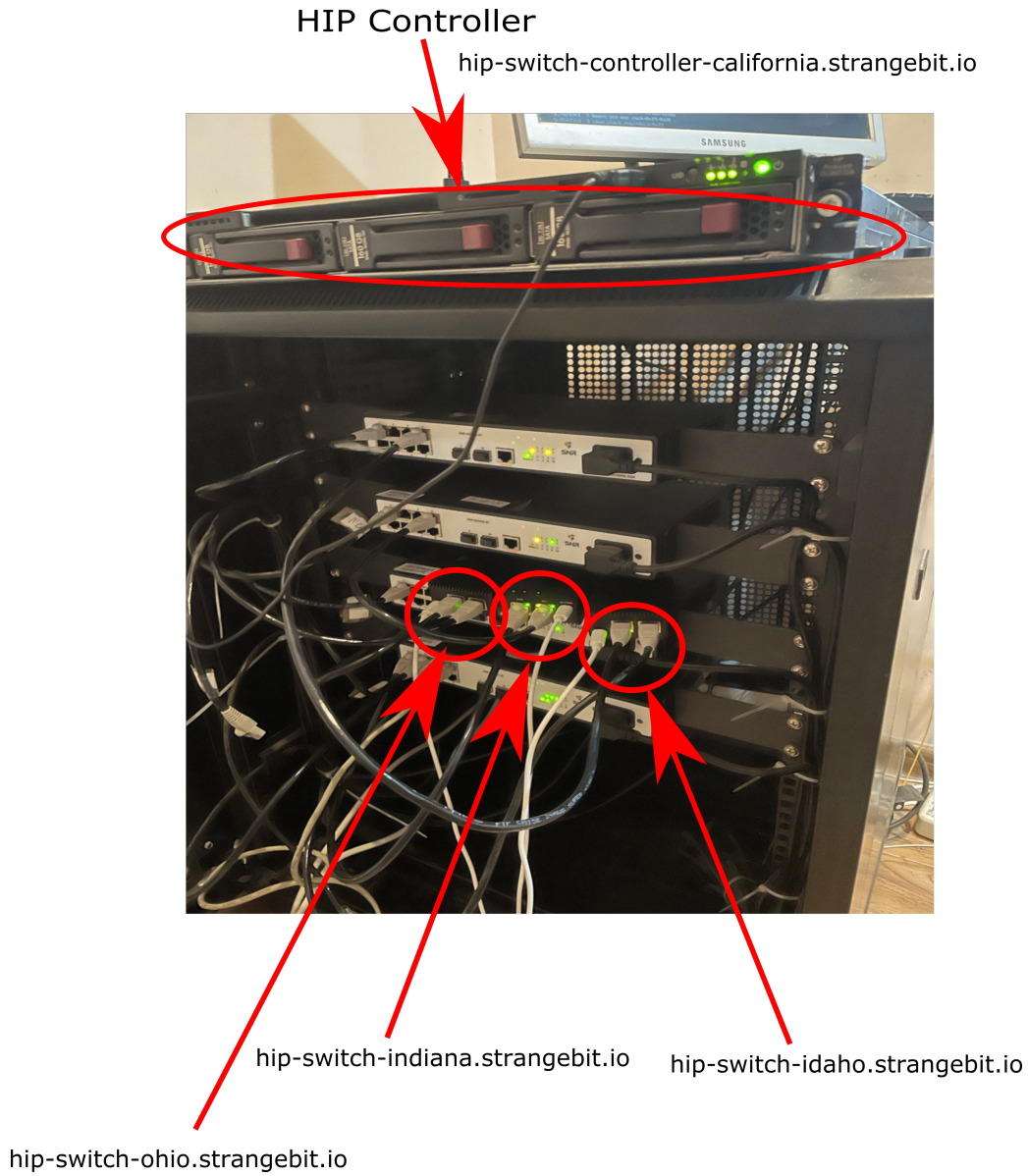

Figure 4.1: Testbed setup

```
$ git clonse https://github.com/strangebit-io/hip-vpls-controller.git
$ cd hip-vpls-controller/deployment
$ sudo bash deploy.sh
```

After HIP controller and configurator were deployed, one needs to deploy the HIP

switch code on NanoPI R2S (remember you need to copy the certchain.pem - self-signed certificates and change the master secret to match the one specified for HIP controller. Also, administrator needs to change the public, routable in the Internet, IP address in the configuration and specify the switch name):

```
$ https://github.com/strangebit-io/hip-vpls-hw-with-controller.git
$ cd hip-vpls-hw-with-controller/
$ sudo bash deployment/deploy.sh
```

Once important aspect. The clocks on NanoPI R2S needs to be set to UTC and should be synchronized with the controller (otherwise, certificate verification will fail during the SSL handshake).

Once everything is set open the browser and open the location http://192.168.1.3:10000/ (or, whatever the IP address of the HIP controller node). You should see the following screen (see Figure 4.2). The credentials can be found in schema.sql file in database folder.

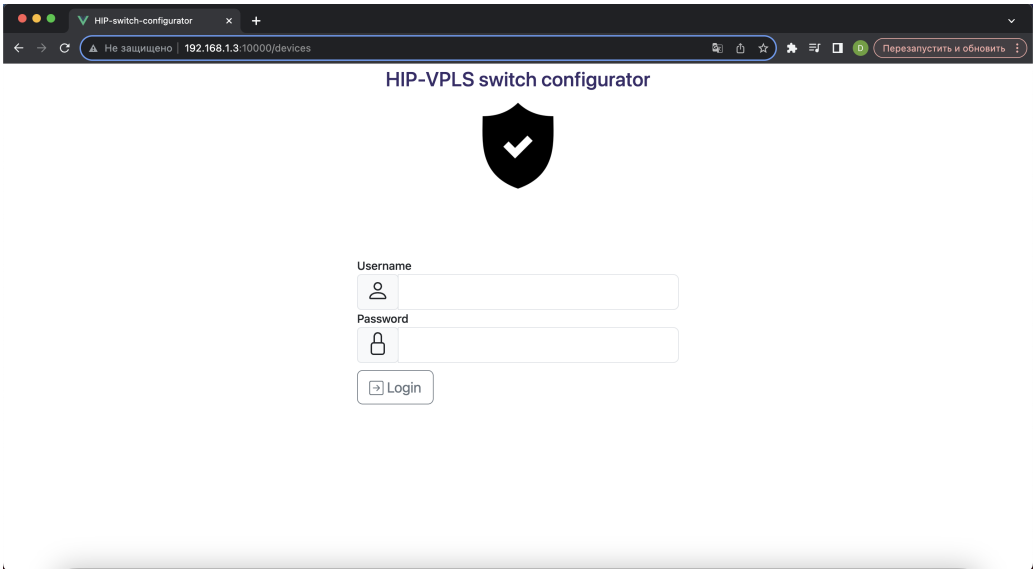

Figure 4.2: Prototype: login screen

After successful login the device status page will open. Here, all registered (registration is performed automatically) HIP switches will appear. You can check the status, HIT and IP of the switches.

Next, one needs to configure the mesh for distributed network. In our setup we simply listed all possible pair of HIP-switches. That is the typical configuration option. For example, see Figure 4.5.

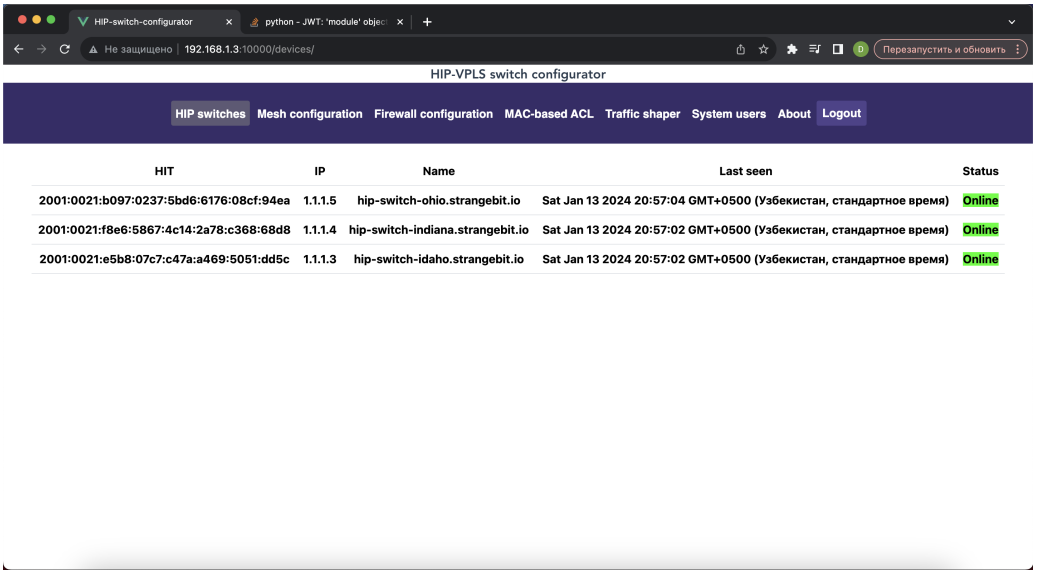

Figure 4.3: Prototype: devices status information

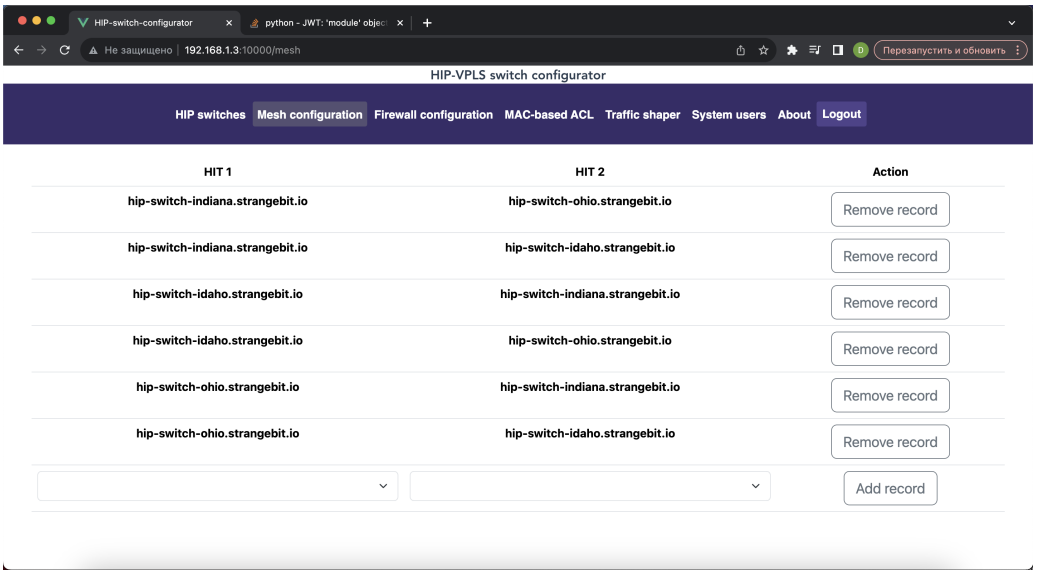

Figure 4.4: Prototype: mesh configuration

Then we need to specify the HIP firewall rules. This operation should be done on firewall tab. See Figure 4.5. Again, a typical setup should have all pairs of HITS.

The final step is to configure MAC address-based access control (see Figure 4.6).

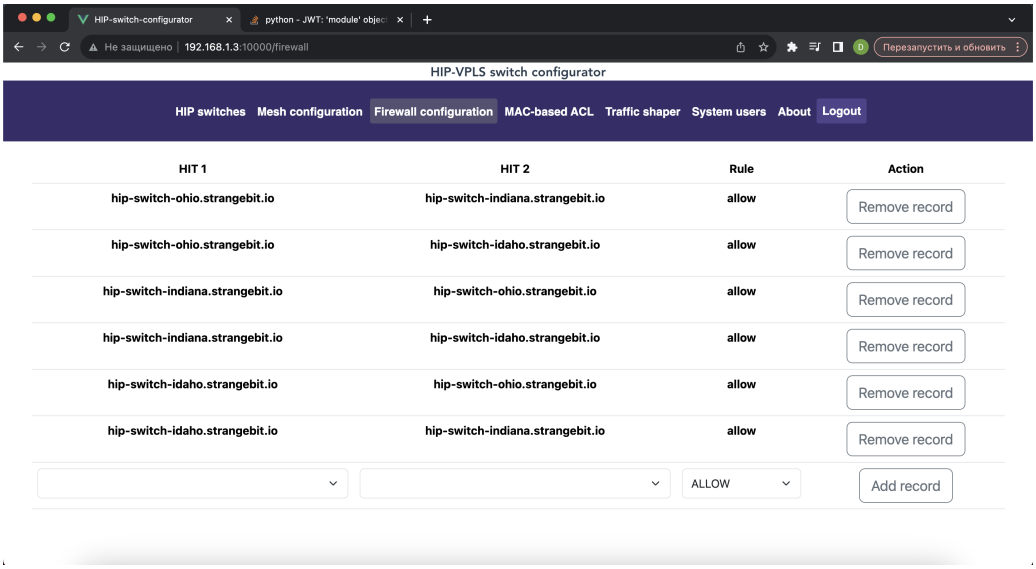

Figure 4.5: Prototype: HIP firewall

Here we need to specify (in outgoing direction) necessary MAC address pairs of hosts in the network. Remember this step need to be completed for each HIP switch separately.

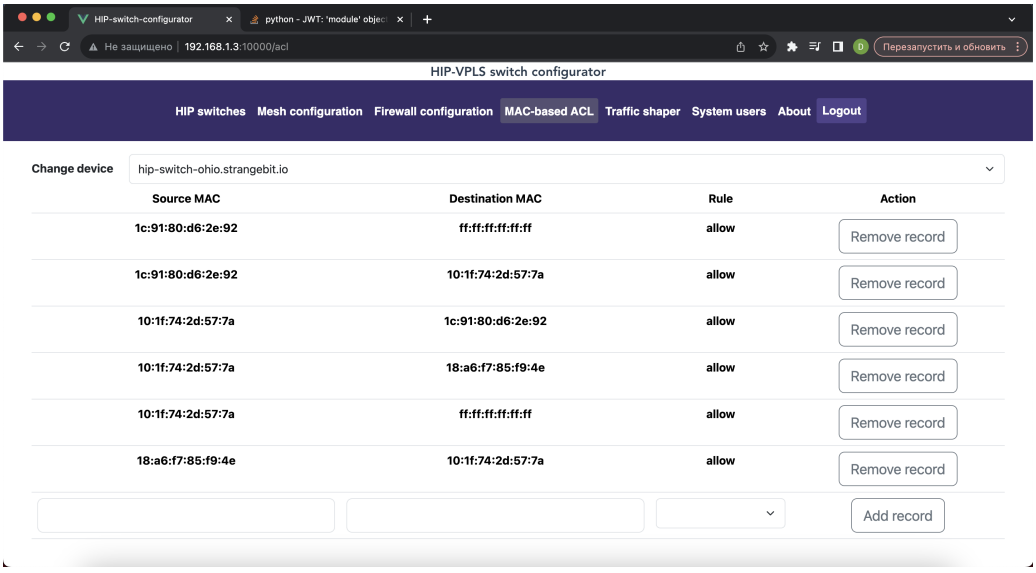

Figure 4.6: Prototype: MAC address based access control lists

Overall, we were not satisfied with the HIP switches. We were observing roughly $120 Kbits/s$ throughput using IPerf tool. So we have tried to improve the performance by implementing in C++ and assembly language AES256 cryptographic algorithm. Luckily, NanoPI R2S chip (ARM Cortex 53) supports AES256 instruction set. We have cross compiled the library and used it in our HIP switch implementation. The results for AES256 encryption is shown in Figure 4.7. The performance of AES256 implemented with special CPU instructions was 10 times better. Thus, our preliminary experiments showed that we can achieve roughly $2.3 Mbits/s$ throughput between pair of hosts.

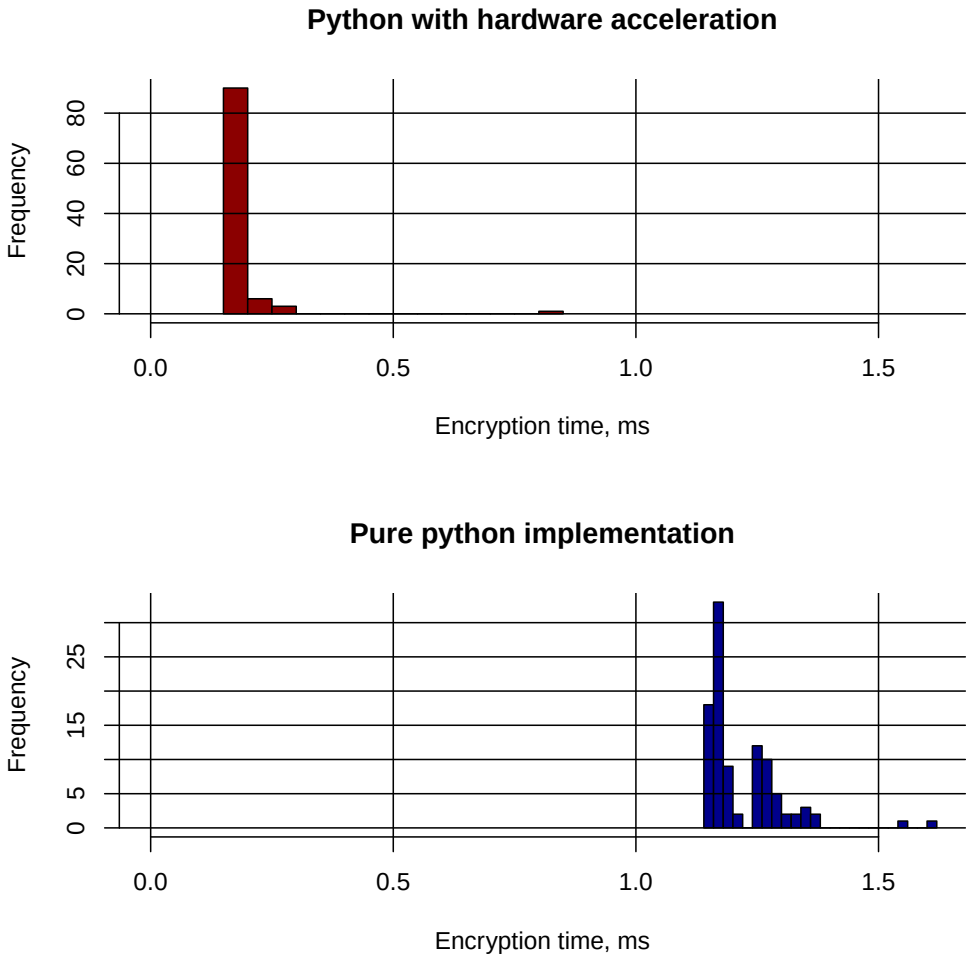

Figure 4.7: AES encryption performance

Literature