

Running HIP-VPLS in infrastructure mode: The prototype implementation

Dmitriy Kuptsov
2024

Table of contents

1	Introduction	5
2	Background	7
2.1	Cryptography	7
2.1.1	Symmetric cryptography	8
2.1.2	Asymmetric cryptography	8
2.1.3	Hash functions	9
2.1.4	Key exchange protocols	9
2.2	Virtual Private LAN Service	9
2.3	Host Identity Protocol	10
2.4	Secure socket layer (SSL)	10
3	Architecture	13
4	Proof-of-a-concept implementation	15

CHAPTER 1

Introduction

The roots of the Internet are back in 70's. Back then the Internet was rather small network and was interconnecting major universities, military and governmental organizations. Today the Internet is a sophisticated network comprising billions of devices.

As the Internet evolved so did the need for the security. Modern Internet contains colossal number of legacy devices which are not capable to perform cryptographic operations and so special solutions are required to protect sensitive data which is passed over the wire from such attacks as Man-In-The-Middle, message forgery, impersonation attack, etc.

The solution is to employ the so called Virtual Private LAN service with message encryption and authentication. With this idea in mind in this work we are going to demonstrate the HIP-VPLS solution which uses Host Identity Protocol (HIP) and IPsec for securing data-plane traffic, and SSL for communication with central server - configurator - for protecting the control plane traffic.

We have already described the HIP-VPLS solution in our previous work (for reference see [5]). Here, the main focus is on the complete architecture and configurator user interface.

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 Secure Socket Layer (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 problem, rings, fields, Euclidean algorithm, factorization of big numbers, etc.)

2.1.1 Symmetric cryptography

Symmetric key cryptography is just perfect for data-plane traffic as it offers low 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 [7]. 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 [7]. 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 [7].

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) [7]. 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 [7]. 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. More on HIP-VPLS can be found here [1].

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. More on the protocol can be found in [6].

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.

2.4 Secure socket layer (SSL)

Secure socket layer (SSL) [4] is an application layer solution to secure TCP connections. SSL was standardized in RFC 6101. And was designed to prevent eavesdropping, man in-the-middle attacks, tampering and message forgery. In SSL the communicating hosts can authenticate each other with help of longer term identities

- public key certificates. SSL is greate for building VPN tunnels and protecting upper layer protocols such as HTTP.

CHAPTER 3

Architecture

The system architecture is shown in Figure 3.1. Apart from the HIP-VPLS switches, we have also implemented a unique control-plane protocol on top of the SSL protocol.

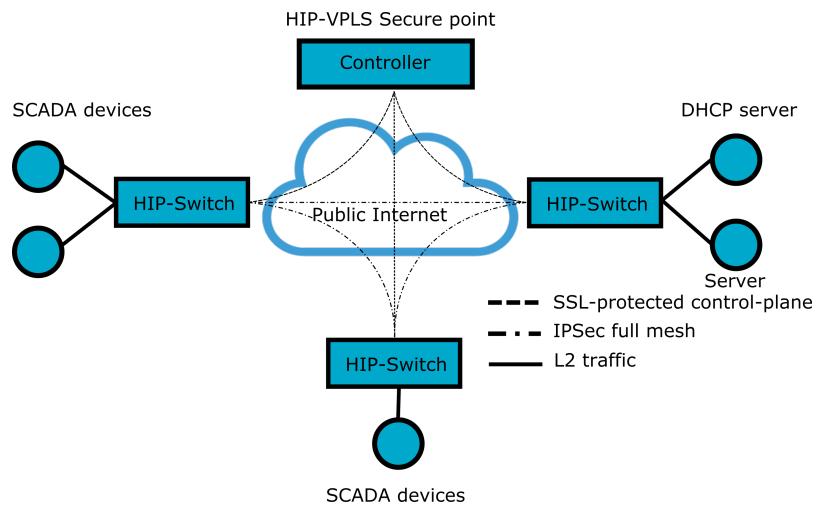


Figure 3.1: System architecture

According to the protocol, on the one hand, every HIP-VPLS switch reports to the central controller (and is authenticated using the HMAC algorithm together with the shared symmetric master secret). In the implementation switches report their

presence every 5 seconds. On the other hand, every HIP-VPLS switch 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 a traffic shaper feature in our HIP switches and HIP controller, it is still valuable for future work. For example, different hosts can be served differently (with more bandwidth) than others by using traffic shaping. If some hosts in the HIP-VPLS network send delay-sensitive traffic, for example, certain rules can be configured on the HIP controller to give a needed advantage over other hosts in the network. We leave this for future discussions and work.

In the testbed, we had a multihomed server (with one IP facing the public network so that HIP switches will be able to connect to the controller in the Internet, and one IP in the private range), several legacy microcomputers, IP camera, and DHCP/DNS server.

CHAPTER 4

Proof-of-a-concept implementation

In this chapter, we will discuss the proof of a concept, or prototype, implementation of the HIP-VPLS (we will 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. The source code for the HIP-VPLS infrastructure is available in the following repositories: [2, 3].

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 reducing the wasted bandwidth and minimizing the delays (since IPSec does not rely on guaranteed delivery of a packet).

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 [3].

To deploy the system, we have prepared the bash script. Overall, the deployment is trivial. The only things that need to change are the master secret and MySQL username and password. Otherwise, the administrator needs to execute the following commands in the server's console (observe, we have tested the deploy script on

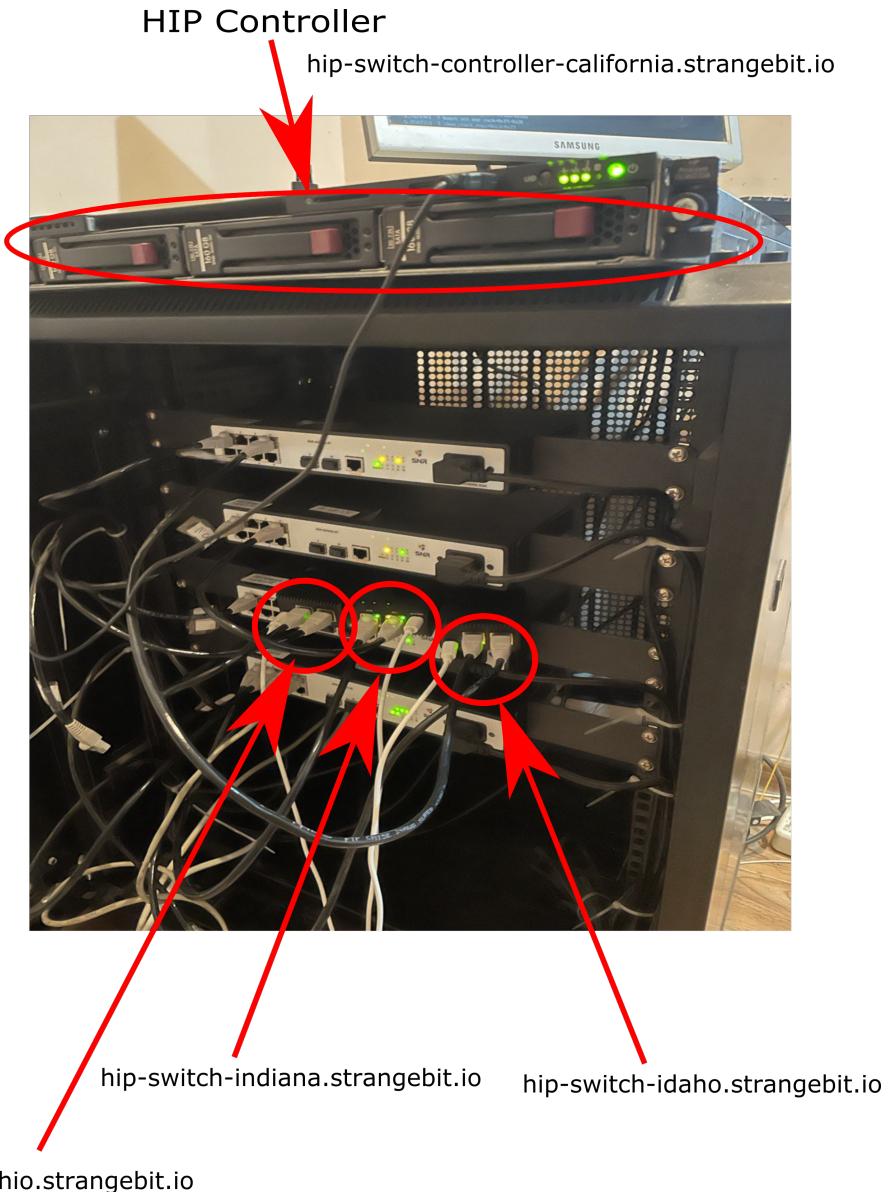


Figure 4.1: Testbed setup

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)):

```
$ git clone https://github.com/strangebit-io/hip-vpls-controller.git
```

```
$ cd hip-vpls-controller/deployment
$ sudo bash deploy.sh
```

After the HIP controller and configurator are 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 the HIP controller. Also, the administrator needs to change the public, routable on the Internet, IP address in the configuration and specify the switch name):

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

One important aspect. The clocks on NanoPI R2S must be set to UTC and 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 is, this needs to be routable in the Internet IP address). One should see the following screen (see Figure 4.2). The credentials can be found in the schema.sql file in the database folder.

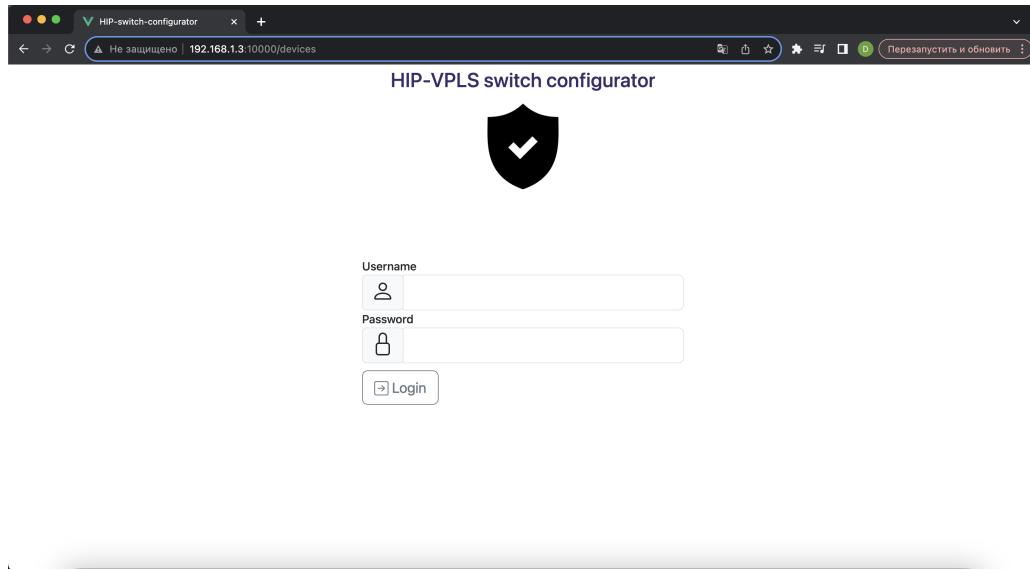
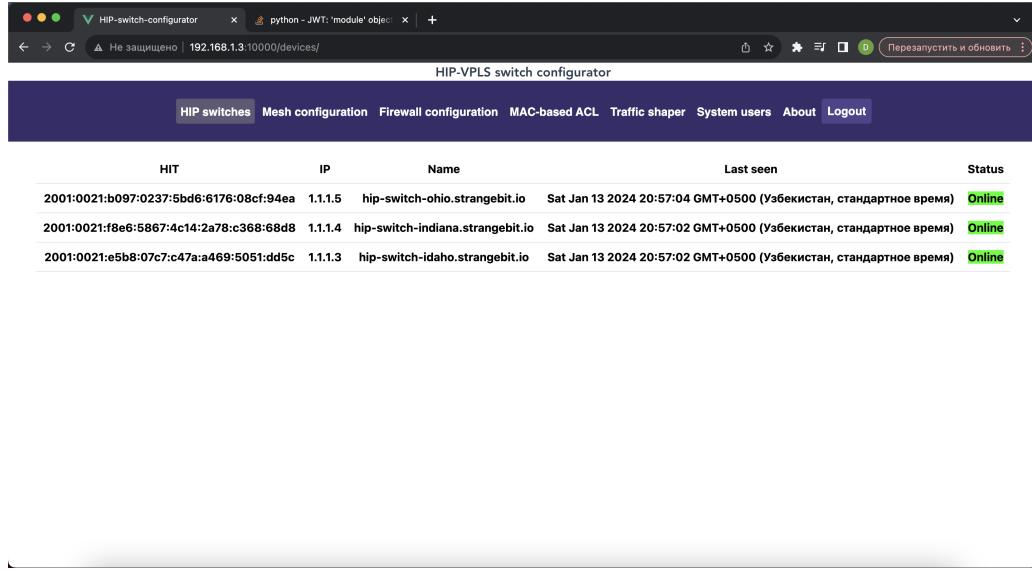


Figure 4.2: Prototype: login screen

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



The screenshot shows a web browser window titled 'HIP-switch-configurator'. The address bar indicates the URL is '192.168.1.3:10000/devices'. The page title is 'HIP-VPLS switch configurator'. A navigation bar at the top includes links for 'HIP switches' (which is the active tab), 'Mesh configuration', 'Firewall configuration', 'MAC-based ACL', 'Traffic shaper', 'System users', 'About', and 'Logout'. Below the navigation bar is a table displaying device status information:

HIT	IP	Name	Last seen	Status
2001:0021:b097:0237:5bd6:6176:08cf:94ea	1.1.1.5	hip-switch-ohio.strangebit.io	Sat Jan 13 2024 20:57:04 GMT+0500 (Узбекистан, стандартное время)	Online
2001:0021:f8e6:5867:4c14:2a78:c368:68d8	1.1.1.4	hip-switch-indiana.strangebit.io	Sat Jan 13 2024 20:57:02 GMT+0500 (Узбекистан, стандартное время)	Online
2001:0021:e5b8:07c7:c47a:a469:5051:dd5c	1.1.1.3	hip-switch-idaho.strangebit.io	Sat Jan 13 2024 20:57:02 GMT+0500 (Узбекистан, стандартное время)	Online

Figure 4.3: Prototype: devices status information

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

The screenshot shows a web-based application titled "HIP-VPLS switch configurator". The URL in the address bar is "192.168.1.3:10000/mesh". The interface has a dark blue header with navigation links: "HIP switches", "Mesh configuration" (which is selected), "Firewall configuration", "MAC-based ACL", "Traffic shaper", "System users", "About", and "Logout". Below the header is a table titled "Mesh configuration" with columns "HIT 1", "HIT 2", and "Action". The table lists six entries, each consisting of two switch names separated by a comma and a period, followed by a "Remove record" button. At the bottom right of the table is an "Add record" button. The table rows are:

HIT 1	HIT 2	Action
hip-switch-indiana.strangebit.io	hip-switch-ohio.strangebit.io	<button>Remove record</button>
hip-switch-indiana.strangebit.io	hip-switch-idaho.strangebit.io	<button>Remove record</button>
hip-switch-idaho.strangebit.io	hip-switch-indiana.strangebit.io	<button>Remove record</button>
hip-switch-idaho.strangebit.io	hip-switch-ohio.strangebit.io	<button>Remove record</button>
hip-switch-ohio.strangebit.io	hip-switch-indiana.strangebit.io	<button>Remove record</button>
hip-switch-ohio.strangebit.io	hip-switch-idaho.strangebit.io	<button>Remove record</button>

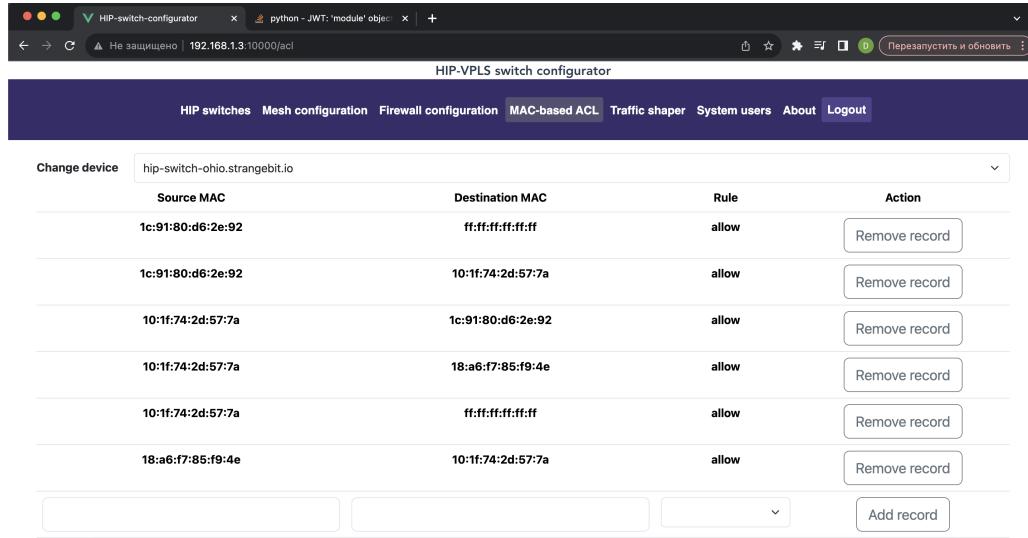
Figure 4.4: Prototype: mesh configuration

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

HIT 1	HIT 2	Rule	Action
hip-switch-ohio.strangebit.io	hip-switch-indiana.strangebit.io	allow	<button>Remove record</button>
hip-switch-ohio.strangebit.io	hip-switch-idaho.strangebit.io	allow	<button>Remove record</button>
hip-switch-indiana.strangebit.io	hip-switch-ohio.strangebit.io	allow	<button>Remove record</button>
hip-switch-indiana.strangebit.io	hip-switch-idaho.strangebit.io	allow	<button>Remove record</button>
hip-switch-idaho.strangebit.io	hip-switch-ohio.strangebit.io	allow	<button>Remove record</button>
hip-switch-idaho.strangebit.io	hip-switch-indiana.strangebit.io	allow	<button>Remove record</button>
		ALLOW	<button>Add record</button>

Figure 4.5: Prototype: HIP firewall

The final step is to configure MAC address-based access control (see Figure 4.6). Here, one needs to specify (in the outgoing direction) the necessary MAC address pairs of hosts in the network. Remember that this step needs to be completed for each HIP switch separately.



The screenshot shows a web-based interface titled "HIP-VPLS switch configurator". The top navigation bar includes links for "HIP switches", "Mesh configuration", "Firewall configuration", "MAC-based ACL" (which is highlighted in blue), "Traffic shaper", "System users", "About", and "Logout". Below the navigation is a table titled "Change device" with the value "hip-switch-ohio.strangebit.io". The table lists MAC-based Access Control (MAC-ACL) rules:

Source MAC	Destination MAC	Rule	Action
1c:91:80:d6:2e:92	ff:ff:ff:ff:ff:ff	allow	<button>Remove record</button>
1c:91:80:d6:2e:92	10:1f:74:2d:57:7a	allow	<button>Remove record</button>
10:1f:74:2d:57:7a	1c:91:80:d6:2e:92	allow	<button>Remove record</button>
10:1f:74:2d:57:7a	18:a6:f7:85:f9:4e	allow	<button>Remove record</button>
10:1f:74:2d:57:7a	ff:ff:ff:ff:ff:ff	allow	<button>Remove record</button>
18:a6:f7:85:f9:4e	10:1f:74:2d:57:7a	allow	<button>Remove record</button>

At the bottom of the table are three input fields and a "Add record" button.

Figure 4.6: Prototype: MAC address based access control lists

Overall, we were not satisfied with the HIP switches. We were observing roughly $120K\text{bits}/s$ throughput using the IPerf tool.

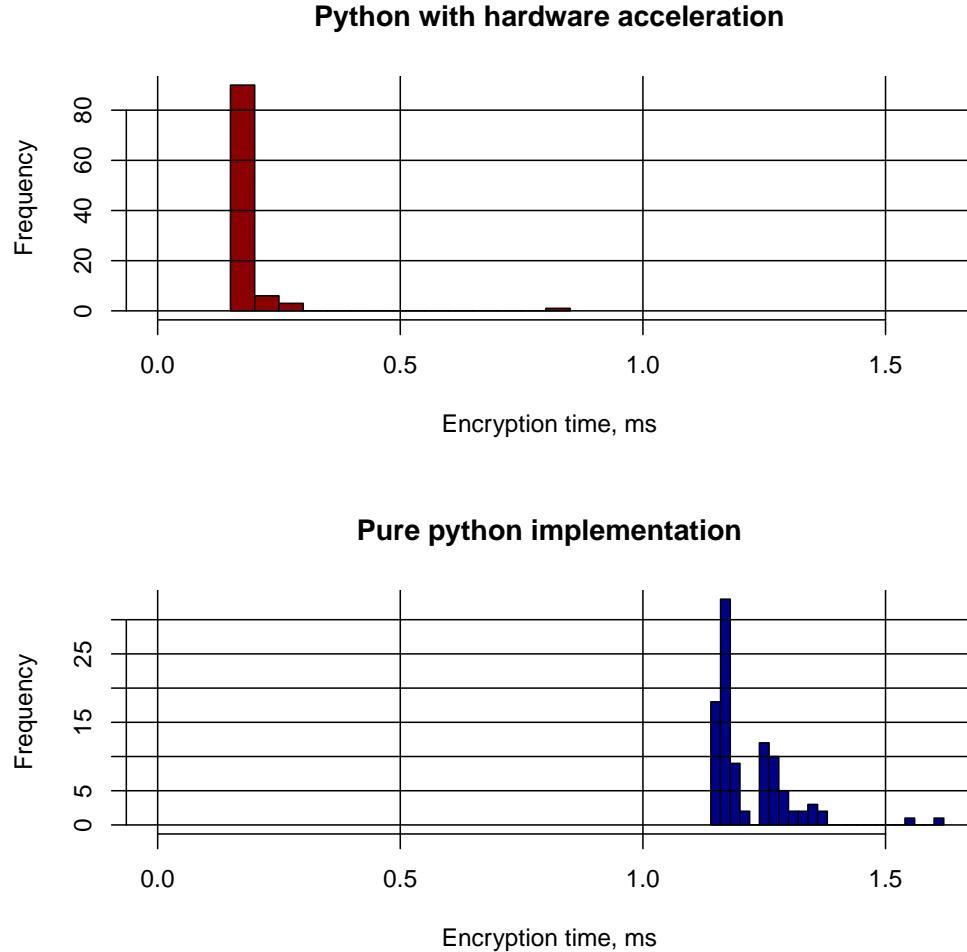


Figure 4.7: AES encryption performance

So we have tried to improve the performance by implementing in C++ and assembly language AES256 cryptographic algorithm. Luckily, the NanoPI R2S chip (ARM Cortex 53) supports the 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 ten times faster. Thus, our preliminary experiments

showed that we can achieve maximum 2.4Mbits/s throughput between pair of hosts (the entire distribution is shown in Figure 4.8).

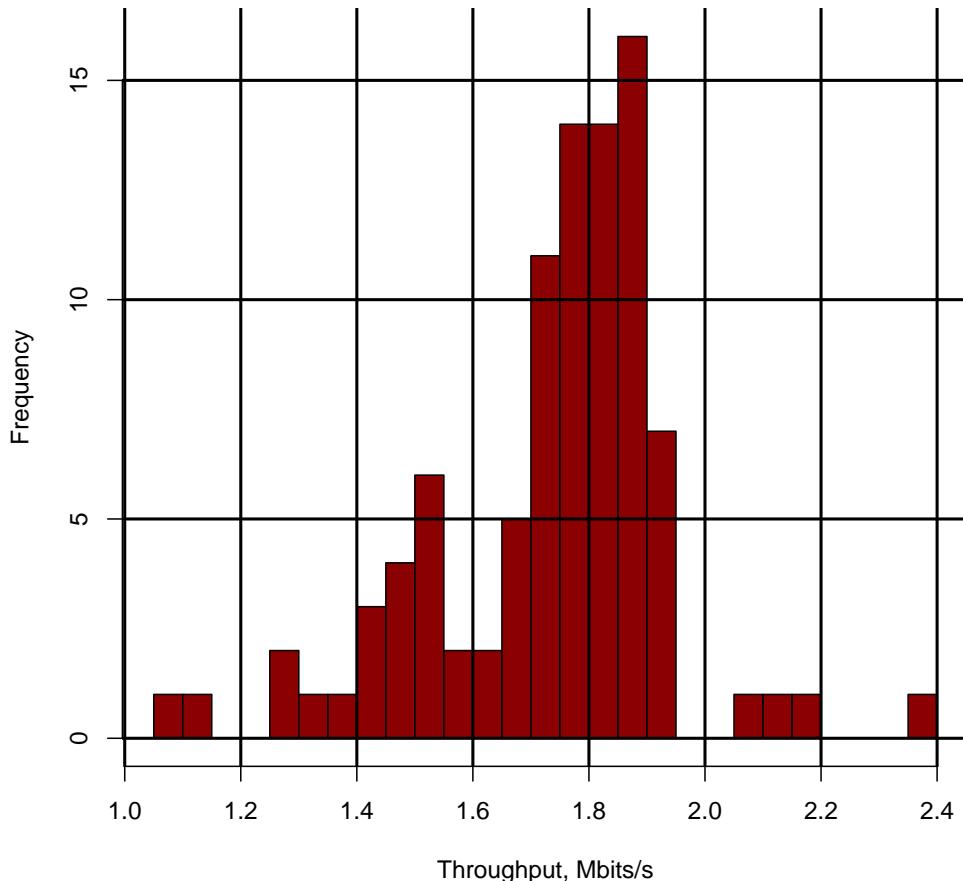


Figure 4.8: Throughput performance

Literature

- [1] HIP-based Virtual Private LAN Service (HIPLS).
<https://datatracker.ietf.org/doc/draft-henderson-hip-vpls/>.
- [2] Hip switch. <https://github.com/strangebit-io/hip-vpls-hw-with-controller>.
- [3] Hip-vpls controller. <https://github.com/strangebit-io/hip-vpls-controller>.
- [4] The Secure Sockets Layer (SSL) Protocol Version 3.0.
<https://datatracker.ietf.org/doc/html/rfc6101>.
- [5] The Secure Sockets Layer (SSL) Protocol Version 3.0.
<https://datatracker.ietf.org/doc/html/rfc6101>.
- [6] A. Gurtov. Host Identity Protocol (HIP): Towards the Secure Mobile Internet. Wiley Publishing, 2008.
- [7] D. Stinson. Cryptography: Theory and Practice, Second Edition. CRC/C&H, 2nd edition, 2002.